EXTENSIONAL REASONING

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Timothy L. Hinrichs
December 2007

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Michael Genesereth) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mark Stickel)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(John Mitchell)

Approved for the University Committee on Graduate Studies.

Abstract

Relational databases have had great industrial success in computer science; however, most automated theorem provers today do not take advantage of database query engines and therefore do not routinely leverage that source of power. Extensional Reasoning is an approach to automated deduction where the system automatically translates an entailment query expressed in classical logic into a query about a database system so that the answers to the two queries are the same. To prove the theorem, the system then evaluates the database query using an off-the-shelf database.

Extensional Reasoning was developed because many problems can be solved efficiently using a database but are naturally expressed using classical logic. In some cases, database query engines solve the database version of the query orders of magnitude faster than traditional theorem proving techniques solve the classical version. Extensional Reasoning helps us build systems that allow a non-expert to write problems down naturally, convert those problems to efficient representations automatically, and solve those problems using industrial-strength systems.

Conceptually, algorithms for performing Extensional Reasoning can be broken into two classes: those for complete theories and those for incomplete theories. A complete theory, one that can answer all the questions in its vocabulary, corresponds naturally to a relational database. Algorithms for this class of theories must recognize the theory is complete and then construct the appropriate database system. In the context of a logic with a finite domain, I present incomplete but low-order polynomial-time algorithms for performing these tasks. An incomplete theory, one for which there is some question in the vocabulary that cannot be answered, does not correspond naturally to a database system, and so the algorithms for performing Extensional Reasoning are more complex. In this case my approach is to construct a new, complete theory that captures the information pertinent to the original problem—a novel form of theory-completion. I present algorithms for performing this type

of theory-completion in the same finite logic.

Acknowledgements

While I've often thought an acknowledgements section pollutes the pristine, mathematical precision a Ph.D. thesis ought to embody, I have nevertheless found myself perusing the acknowledgements each and every time I read someone else's thesis. And now that it is time to write my own, I feel compelled to thank those people who both helped me survive the rough patches and celebrate the successes.

Starting with my mentor, Mike Genesereth expected nothing less than for me to become a world-class researcher. My shortcomings are my own, but I owe my strengths to him. I could not have been here at Stanford but for all his support and would have nothing to show even still, I wager, without his constant challenges and continuous flow of ideas.

Mark Stickel and I interacted several times during my tenure in the program, and toward the end I found myself continually using Mark as the canonical theorem proving expert, asking myself at every turn, "What would Mark think?" Mark made two very thorough reads of the final document, providing detailed and much appreciated comments. John Mitchell provided valuable feedback several times, ensuring that we who were so close to the project never missed any key questions. David Dill served the role of the in-the-trenches logician, requiring practical results that could be applied today. Just knowing that Mark, John, and David would be evaluating the work forced me to keep in mind the concerns of the three communities they represent, and the work has been all the better for it.

The members of MUGS as well as those attending MUGS meetings who may or may not consider themselves officially part of the group have tolerated many bad and several terrible talks I gave in group meetings and put up with my not infrequent wanderings around the office asking for advice on some esoteric subject. I can't say how I might begin to pay off that debt.

Technical help aside, I have constantly relied on the encouragement of my friends and family, probably more so than they realize. My parents have been incredibly supportive,

though that is no surprise, and my closest friends, Tim, Eran, and Michele, have been instrumental in getting me through the last year especially.

And finally, Nina, you've been my constant companion, your willingness to let me ramble on about models and proofs never faltered, and your ability to make sense of those ramblings and respond with something sensible continually amazed me. I seriously doubt I would have finished were it not for you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Researchers in the field of automated reasoning attempt to endow computer systems with the ability to solve problems automatically. The field includes communities that research boolean satisfiability, constraint satisfaction, relational databases, logic programming, and automated theorem proving, to name a few. Today, those communities are to some extent isolationist: each community promotes the approach it studies, sometimes even to the exclusion of the others. On the contrary, one of the underlying claims of this thesis is that each approach has its own strengths and weaknesses, and a computer system that could leverage the tradeoffs would help in bringing automated reasoning to the masses.

Experts in the field routinely employ different tools depending on the problem at hand. Each tool has a different syntax and semantics, which often leads to different computational properties. Given a particular problem to solve, an expert will decide which of the existing tools is most appropriate and will reformulate the problem into the language that tool accepts. A non-expert, when given the same task, is hindered by two requirements: understanding several different input languages and having the ability to reformulate a given problem into each of those languages. In extreme cases, the user is another computer system, which makes these issues especially formidable. This leads to a desire for a system that when given a problem stated in a formal language analyzes the structure of the problem, decides which of the existing tools are best suited to solving that problem, and automatically translates the problem into the language required by that approach.

This thesis begins the investigation into such a system by considering one automated reasoning problem in particular: automated deduction. Given a set of premises and a conclusion, do the premises logically entail the conclusion? Furthermore, this thesis limits

its examination to precisely one of the automated reasoning tools mentioned earlier: the relational database.

When compared to boolean satisfiability, constraint satisfaction, logic programming, and theorem proving, the relational database may be the most important technology for building an automated reasoning system with mass appeal. It is, after all, the most industrially successful application of formal logic today. Many people care a great deal about the class of problems databases can solve, as evidenced by the database industry's billions of dollars of yearly revenue; consequently, databases are well-known, and people expect an automated reasoning system to solve any problem they believe can be solved by a database. In addition, the problems expressible as database queries are a subset of those expressible in logic programming languages and a superset of those expressible as finite constraint satisfaction problems, paving the way for future work.

Extensional Reasoning (ER) is, as far as we know, a novel approach to automated deduction where when given an entailment query in classical logic, the system constructs a relational database and a query about it so that the answers to the two queries are the same. The system then answers the entailment query by answering the database query. More precisely, what we will see is that to build a relational database, the system constructs a set of extensional tables (explicitly represented tables) and a set of intensional tables or view definitions (implicitly represented tables). Extensional Reasoning is so-named because entailment queries are answered by answering database queries, which amount to reasoning about a set of extensions. ER marries the flexibility and modularity of classical logic with the theoretical results and industrial-strength implementations of the relational database.

Capitalizing on these industrial-strength database algorithms turns out to be challenging because a relational database corresponds semantically to a very special kind of premise set: one that is axiomatically complete. Such premise sets are rare, and even when the given premise set is complete, recognizing that fact and constructing the appropriate database is nontrivial. When the given premise set is incomplete, constructing the appropriate database is even more difficult. The approach to handling incomplete premise sets reported in this thesis first performs a novel form of theory completion and then leverages the results from the complete case. The central contribution of this thesis is a suite of algorithms for automatically translating an automated deduction query into a database system and a query about it.

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 a decidable logic, placing the issue of efficiency front and center. The particular logic studied thus far, Finite Herbrand Logic, 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 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.

Extensional Reasoning attempts to leverage one of the most successful applications of formal logic in computer science, the relational database, to more efficiently perform automated deduction on a classical logic. The novelty in this approach lies in the automation of the method. People frequently perform such transformations; this thesis introduces algorithms for automating that process.

1.1 Contributions of this Thesis

All of the results in this thesis are concerned with a decidable fragment of first-order logic, Finite Herbrand Logic, which is defined formally in the next section. In that logic, a certain class of logical sentence sets, or theories, correspond very naturally to a relational database system. Complete, satisfiable theories have the property that every closed sentence or its negation is entailed by the theory—such theories have essentially a single model. Since a database system is a compact representation of a single model (when compared to the naïve representation of a model as a set of relations), it is natural to represent a complete theory with a database system. The first results reported in this thesis are algorithms for detecting that a logical theory is complete, and converting that theory into a database system so that entailment queries about the theory can be answered by querying the database.

These results are then extended to handle incomplete theories by turning the detection algorithm into an anytime algorithm for extracting the portion of an incomplete theory that is complete. Using theory resolution, one can then construct a system that reasons about the complete portion of the theory with a database and the incomplete portion of the theory with resolution. The next result is an inference rule for performing theory resolution when the incomplete portion is in the universal fragment and the hidden subtheory is complete. Answering an entailment query about the original theory can then be performed by adding the negation of the query to the incomplete portion, and searching for the empty clause

using theory resolution.

The next results depart from the theory resolution approach. An incomplete theory differs from a complete theory because an incomplete theory is satisfied by more than one model. Checking just one of those models is insufficient for determining entailment; however, it turns out that it is possible to construct a single model and a query about it that is sufficient for determining entailment. The last batch of results are algorithms for performing entailment-preserving theory-completion: given premises and a conclusion construct a complete theory and a query about it so that the original premises entail the original conclusion whenever the new, complete theory entails the new query. Again this gives us the ability to answer an entailment query about the premise set by answering a database query.

Altogether, these algorithms constitute a method for automatically translating an entailment query written in Finite Herbrand Logic into a database system and a query about it so that the two queries have the same answer. When an entailment problem can be efficiently solved using a database system, these algorithms allow us to automatically translate that problem into a database problem, thereby allowing the system to leverage that source of efficiency.

1.2 Logic Needed for this Thesis

This thesis addresses the translation of logical entailment queries into database queries. The entailment queries are written in Finite Herbrand Logic, and the database queries are written in *nr-datalog*$^\neg$. Finite Herbrand Logic was chosen because it is a decidable, classical logic that shares some of the essential properties of more expressive logics, most importantly the ability to naturally express certain problems. *nr-datalog*$^\neg$ was chosen because its syntax is very similar to that of Finite Herbrand Logic, and it can easily be translated into SQL, the language used by industrial-strength database systems.

1.2.1 Finite Herbrand Logic

The logic used in this thesis, Finite Herbrand Logic (FHL), is first-order logic with equality and the following three restrictions.

- no function constants

- domain closure assumption
- unique names assumption

We use standard definitions for the syntax of FHL. A finite collection of object constants and predicates together with their arities is called a vocabulary. A term is a variable, denoted by a letter from the end of the alphabet, or an object constant, denoted by a letter from the beginning of the alphabet. An atom is a predicate of arity n applied to n terms, and a literal is an atom or $\neg$ applied to an atom. Logical sentences are defined inductively: if ϕ and ψ are sentences and x is a variable then all of the following are sentences: $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \Rightarrow \psi)$, $(\phi \Leftarrow \psi)$, $(\phi \Leftrightarrow \psi)$, $(\forall x.\phi)$, and $(\exists x.\phi)$. Sentences without variables are called ground. Variables not captured by a quantifier are called free variables. Sentences with free variables are called open sentences, and sentences without free variables are called closed sentences. The set of all sentences for a given vocabulary is called the language for that vocabulary.

We use fairly standard metalevel notation. A lower-case Greek letter denotes a single sentence, and an upper-case Greek letter denotes a finite set of sentences. Instead of writing out n variables as $x_1, \ldots, x_n$, we will often write $\overline{x}$; likewise objects $a_1, \ldots, a_n$ will be written $\overline{a}$. Sometimes a predicate will take some number of variables that we will want to group together; we will use similar shorthand. For example, $p(x_1, \ldots, x_n, y_1, \ldots, y_m)$ may be written $p(\overline{x}, \overline{y})$.

In FHL, every theory makes the unique names assumption (UNA) and the domain closure assumption (DCA). The UNA states that every pair of distinct object constants is unequal. The DCA states that every object in the universe must be one of the object constants in the vocabulary. These assumptions make the definitions of FHL semantics relatively simple.

We use the standard definitions for a model and for satisfaction but take advantage of the UNA and DCA to simplify those definitions. A model in FHL is a set of ground atoms from the language. 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* $p(t_1, \ldots, t_n) \in M$

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

$\models_M \phi \wedge \psi$ *if and only if* $\models_M \phi$ *and* $\models_M \psi$

$\models_M \phi \vee \psi$ *if and only if* $\models_M \phi$ *or* $\models_M \psi$ *or both*

$\models_M \phi \Leftarrow \psi$ *if and only if* $\models_M \phi \vee \neg\psi$

$\models_M \phi \Rightarrow \psi$ *if and only if* $\models_M \neg\phi \vee \psi$

$\models_M \phi \Leftrightarrow \psi$ *if and only if* $\models_M \phi \Leftarrow \psi$ *and* $\models_M \phi \Rightarrow \psi$

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

$\models_M \exists x.\phi(x)$ *if and only if* $\models_M \phi(a)$ *for some object constant* a.

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

A set of sentences is satisfiable (or consistent) when there is at least one model that satisfies it. A set of sentences constitutes an incomplete theory whenever there is more than one model that satisfies it. A set of sentences in FHL constitutes a complete theory whenever there is at most one model that satisfies it. Logical entailment is defined as usual: $\Delta \models \phi$ if and only if every model that satisfies Δ also satisfies ϕ. For clarity, we may write $\Delta \models_{FHL} \phi$. For complete, satisfiable theories, the notion of consistency and entailment coincide. To highlight this special case, we may write that a sentence is true in a theory to mean that the theory is complete, and the sentence is consistent with it (entailed by it).

It turns out that Finite Herbrand Logic has exactly the same expressiveness as propositional logic, i.e. every set of models captured by a set of sentences in FHL can be captured by a set of sentences in propositional logic and vice versa (modulo syntactic differences). Consequently, since entailment in propositional logic is decidable so is entailment in FHL decidable. That is, there is a proof system $\vdash$ that is sound and complete for FHL that terminates in all cases. Sound proof procedures find valid proofs: if $\Delta \vdash \phi$ then $\Delta \models \phi$. Complete proof procedures always find a proof if there is one: if $\Delta \models \phi$ then $\Delta \vdash \phi$. Satisfiability in FHL is NEXPTIME-complete in the length of the sentence; validity and entailment are therefore coNEXPTIME-complete. See Appendix A.1 for proof.

The naïve proof system for answering entailment queries in FHL relies on the usual reduction of entailment to unsatisfiability: $\Delta \models \phi$ if and only if $\Delta \cup \{\neg\phi\}$ is unsatisfiable.

To answer $\Delta \models \phi$, the proof system converts $\Delta \cup \{\neg\phi\}$ to propositional logic and then gives the result to a propositional SAT solver, which checks unsatisfiability. To convert FHL to propositional logic, the proof system replaces every universally quantified sentence by a conjunction of sentences and every existentially quantified sentence by a disjunction of sentences. If $\{a_1, \ldots, a_n\}$ is the set of object constants in the vocabulary then $\forall x.\phi(x)$ becomes $\phi(a_1) \wedge \cdots \wedge \phi(a_n)$, and $\exists x.\phi(x)$ becomes $\phi(a_1) \vee \cdots \vee \phi(a_n)$. This conversion requires time and space exponential in the number of quantifiers.

The purpose of this thesis is to report on an alternative proof system for FHL that converts a logical entailment query into a database query and then relies on industrial strength algorithms for answering that database query, an approach that is analogous to the SAT-solver approach described above.

1.2.2 *nr-datalog*$^\neg$

The language used in this thesis for representing database systems is nonrecursive datalog with negation, sometimes written *nr-datalog*$^\neg$, which is equivalent in expressiveness to relational algebra.

The syntax is defined as usual. A *nr-datalog*$^\neg$ system consists of (1) a set of tables, named using the Extensional Database predicates (EDB predicates) and (2) a set of *nr-datalog*$^\neg$ rules, where the predicates in the head of each rule are drawn from the Intensional Database predicates (IDB predicates). The EDB and IDB predicates are always disjoint. A *nr-datalog*$^\neg$ 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 nonrecursive, which is a property 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. That edge is labeled with $\neg$ if the literal in which v belongs is negative. To be nonrecursive, the dependency graph for a rule set must be acyclic. Analogous to the FHL case, we use a lower-case Greek letter to denote a database query or a single *nr-datalog*$^\neg$ rule, and we use

an upper-case Greek letter to denote a *nr-datalog*$^\neg$ system.

The semantics for a *nr-datalog*$^\neg$ system are the usual stratified semantics. A model for a *nr-datalog*$^\neg$ system is the same as that for FHL: a set of ground atoms. Satisfaction is defined the same for *nr-datalog*$^\neg$ rules as for FHL sentences. A model M satisfies a set of tables if and only if for every tuple $\overline{a}$ in table t, $t(\overline{a}) \in M$ and for every ground atom $t(\overline{a}) \in M$, the tuple $\overline{a}$ is in table t. Unlike FHL, where the semantics for a set of sentences is the set of models that satisfies those sentences, the semantics for a *nr-datalog*$^\neg$ system is always a single model. Without negation, this model is the smallest model that satisfies all the tables and *nr-datalog*$^\neg$ rules. Model M is smaller than model N exactly when $M \subset N$.

When negation is included, we need the notion of a stratum to define the semantics. We say that predicate p belongs to stratum i if and only if when examining the dependency graph, the maximum number of edges labeled with $\neg$ starting at p is i. Every predicate belongs to a finite stratum, so every rule can be assigned to the stratum of the predicate in its head. Stratum 0 consists of rules without negation and have a single minimal model M_0. To compute M_i for stratum i where $i > 0$, initialize M_i to M_{i-1}. Perform the following operation, adding the result to M_i until no changes occur. Let $h :- b_1 \wedge \cdots \wedge b_n$ be a rule in stratum i. For each substitution σ mapping all the variables to object constants so that $\models_{M_i} (b_1 \wedge \cdots \wedge b_n)\sigma$, output $h\sigma$.

The semantics for a *nr-datalog*$^\neg$ system where k is the largest stratum number is M_k. If the semantics for the *nr-datalog*$^\neg$ system Δ is model M then $\Delta \models \phi$ if and only if $\models_M \phi$. For clarity, we may write $\Delta \models_D \phi$. Just like FHL, entailment is decidable. A database query ϕ can be represented as additional datalog rules added into a *nr-datalog*$^\neg$ system Δ; thus, answering an entailment query amounts to constructing the semantics for a *nr-datalog*$^\neg$ system. The definition of semantics given above is a constructive procedure for building that model. The complexity of a *nr-datalog*$^\neg$ query is PSPACE-complete [VV98]. It is often broken up into the data complexity (where the size of the query is constant), which is LOGSPACE-complete, and the expression complexity (where the size of the database is constant), which is PSPACE-complete [Var82].

1.3 Overview

This thesis lies squarely in the realm of automated reasoning, touching on a number of its much-studied subfields: theorem proving, relational databases, constraint satisfaction,

boolean satisfiability, and logic programming. The majority of the thesis (Chapters 2-4) illustrates how to automatically translate automated deduction problems into database problems.

- Chapter 2 starts with an example that will be used throughout the thesis, illustrating problems encountered in Extensional Reasoning and hinting at their solutions.
- Chapter 3 covers the easy case, where the premise set is a complete theory, and presents algorithms for detecting that a premise set is complete and transforming it into a database system. It also reports on empirical and analytical comparisons of Extensional Reasoning and traditional automated deduction techniques.
- Chapter 4 covers the hard case, where the premise set is an incomplete theory, which is a strict generalization of the easy case. A slight variant of the algorithm for detecting that a theory is complete gives an algorithm for detecting the portion of the theory that is complete. One way to handle incomplete theories is with theory resolution, and algorithms for the case of theories without skolem symbols are presented. Alternatively, incomplete theories can be addressed using a new form of theory completion, which is defined and operationalized.

The remainder of the thesis consists of Chapter 5 on related work, Chapter 6 on future work, and Appendix A on proofs for the theorems found throughout the thesis.

Chapter 2

Example

The purpose of this chapter is to illustrate Extensional Reasoning and point out some of the obstacles that must be overcome, providing an outline for the rest of the thesis. Extensional Reasoning embodies a simple idea: solving automated deduction problems with relational databases. The techniques for performing Extensional Reasoning differ depending on whether the premise set about which queries are asked comprises a complete theory or an incomplete theory. We start with a complete theory.

Suppose we were interested in reasoning about the grid shown in Figure 2.1. If all we cared about were which cells were adjacent to which other cells, we could encode that information as shown below. The predicate *adj* is true of all the pairs of cells that are located next to one another in one of the four cardinal directions. Its definition is built on top of *edge*, which represents a directed version of *adj*. One could define *adj* without *edge*, but the resulting definitions would be about twice as large.

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

Figure 2.1: A 4×4 grid

$$adj(x,y) \Leftrightarrow (edge(x,y) \vee edge(y,x))$$

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

It turns out that these axioms comprise a complete theory in Finite Herbrand Logic, which means they are satisfied by (or represent) a single model. Because a relational database represents a single model, it is natural to answer questions about this theory by converting it into a database. The reason we know this theory is complete is because it is expressed as a set of nonrecursive biconditional definitions. Each sentence is a biconditional definition for the predicate on the left hand side, and the biconditionals altogether are nonrecursive because there is no predicate that is (transitively) defined in terms of itself.

A theory written as a nonrecursive set of biconditionals is convenient because (1) it is guaranteed to comprise a complete theory, and (2) it can easily be translated into a database system. Because of these properties, Chapter 3 introduces an algorithm that attempts to reformulate an arbitrary FHL theory into a nonrecursive set of biconditionals. The algorithm simultaneously addresses the problems of detecting that a theory is complete and rewriting it into a form that is amenable to *nr-datalog*$^\neg$ translation.

Once a theory has been written as a nonrecursive set of biconditionals, translating it into a *nr-datalog*$^\neg$ system is quite easy, though translating it into the optimal *nr-datalog*$^\neg$ system is more difficult. Recall that a *nr-datalog*$^\neg$ system consists of a set of extensions

(explicitly defined tables) and a set of intensions (logical sentences that say how to compute new tables from the ones given). Notice that each of the biconditionals above represent a table of values. The decision that must be made when translating to *nr-datalog*$^\neg$ is which of the biconditionals should be turned into extensional tables, leaving the rest to become intensional tables. A wrong choice can result in suboptimal performance.

In the grid example, suppose we choose to make *edge* an extensional table. Then the remaining predicate *adj* corresponds to an intensional table and must be defined using *nr-datalog*$^\neg$ rules. The transformation of a nonrecursive biconditional into *nr-datalog*$^\neg$ has been understood for decades. The *nr-datalog*$^\neg$ rules that follow in addition to Table 2.1 is the *nr-datalog*$^\neg$ system produced by running the transformation algorithm on the axioms above.

$$adj(x, y) :- edge(x, y)$$
$$adj(x, y) :- edge(y, x)$$

To answer a query such as $adj(a, e)$, we now have several options. One would be to use a resolution-style theorem prover giving to it the original axioms above; another would be to use the *nr-datalog*$^\neg$ system and a deductive database engine; a third would be to convert the original axioms to propositional logic and use a SAT solver, as discussed in the introduction. Chapter 3 discusses the tradeoffs of these options, both analytically and empirically.

This example was particularly easy to translate into *nr-datalog*$^\neg$ because the original axioms represented a single model, i.e. the theory was complete, just like a database system. Incomplete theories are more troublesome.

To illustrate, we consider a hospital environment built on top of the grid world considered thus far. In this hospital, patients and certain pieces of equipment are outfitted with badges that allow a sensor network distributed throughout the hospital to pinpoint, with some amount of uncertainty, the locations of the patients and the equipment. This sensor network is important because certain patients are dependent on certain pieces of equipment, so much so that if such a patient and such a piece of equipment are ever separated, the hospital staff need to be alerted so that they can take the appropriate action. The sensor readings are imperfect; any particular sensor reading indicates that the object of interest is located either in the cell from which the sensor reading originated or in one of the immediately surrounding cells.

edge	
a	b
a	e
b	c
b	f
c	d
c	g
d	h
e	f
e	i
f	g
f	j
g	h
g	k
h	l
i	j
i	m
j	k
j	n
k	l
k	o
l	p
m	n
n	o
o	p

Table 2.1: Complete definition for *edge*

The diagram in Figure 2.2 depicts a snapshot of the hospital where the system has received two sensor readings. The one on the left corresponds to a patient, and the one on the right corresponds to an oxygen tank: the piece of equipment that the patient relies upon. The patient must be located in one of the cells in the immediate vicinity of his sensor reading: a, b, or e; likewise, the oxygen tank must be located in l, o, or p. It is noteworthy that the patient, wherever he might be, is not adjacent to the oxygen tank, wherever it might be; thus, the patient has been separated from the oxygen tank.

Using FHL, we can formalize Figure 2.2 with the definitions for *adj* shown above and the following four sentences. The first two say that there is only one true location of the patient, and the patient must be in one of the cells close to his sensor reading. The second

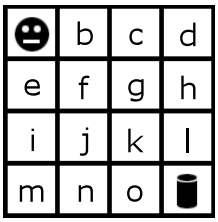

Figure 2.2: A snapshot of the hospital, where the patient is not adjacent to the oxygen

two say there is only one oxygen tank, and the oxygen must be located in one of the cells close to its sensor reading.

$$\forall xy.(patient(x) \wedge patient(y) \Rightarrow x = y)$$
$$patient(a) \vee patient(b) \vee patient(e)$$
$$\forall xy.(oxygen(x) \wedge oxygen(y) \Rightarrow x = y)$$
$$oxygen(l) \vee oxygen(o) \vee oxygen(p)$$

This theory is incomplete because there is more than one model that satisfies it. In one of the models, the patient is located in cell a, and the oxygen is in cell l; in another, the patient is in cell b while the oxygen is in cell p. Altogether there are nine models that satisfy these sentences: one for each possible combination of the three patient and three oxygen locations.

As pointed out earlier, we know that these sentences entail the query that asks, "Has the patient been separated from the oxygen tank?"

$$\forall xy.(patient(x) \wedge oxygen(y) \Rightarrow \neg adj(x, y))$$

In Extensional Reasoning, we will prove this theorem by constructing a database system and a query about that system so that the answers to the two queries are the same.

Our approach to reformulating the entailment query $\Delta \models \phi$ into a database system is a two-step process. First, staying within Finite Herbrand Logic, we construct a new, complete theory Δ' and a query about it, ϕ', so that entailment is preserved.

$$\Delta \models \phi \text{ if and only if } \Delta' \models \phi'$$

Then, we apply the results from the case of complete theories to $\Delta' \models \phi'$ to construct a

database system and a query about it so that the answer to the database query is the same as that for $\Delta' \models \phi'$, which in turn is the same answer as that for $\Delta \models \phi$.

The original hospital theory consists of the complete definitions for *adj* and the incomplete constraints on *patient* and *oxygen*. The new, complete theory consists of (1) the definitions for *adj* shown above and (2) definitions for two new predicates, $poss_{patient(x)}$ and $poss_{oxygen(y)}$, that represent all the instances of *patient* and *oxygen*, respectively, that are consistent with the original theory. In addition we construct a query about the new theory that is expressed in terms of $poss_{patient(x)}$, $poss_{oxygen(y)}$, and *adj*.

Consider just the sentences involving *patient*.

$$\forall xy.(patient(x) \wedge patient(y) \Rightarrow x = y)$$
$$patient(a) \vee patient(b) \vee patient(e)$$

We construct a definition for the predicate $poss_{patient(x)}(x)$ that represents all of the *patient* tuples that are consistent with the *patient* sentences above.[1] In this case, $patient(a)$, $patient(b)$, and $patient(e)$ are the only consistent *patient* atoms, and hence we know that $poss_{patient(x)}$ is true of exactly a, b, and e.

$$poss_{patient(x)}(x) \Leftrightarrow (x = a \vee x = b \vee x = e)$$

Similar reasoning concludes that $poss_{oxygen(y)}$ is true of l, o, and p.

$$poss_{oxygen(y)}(y) \Leftrightarrow (y = l \vee y = o \vee y = p)$$

The original entailment query, "Has the patient been separated from the oxygen?" is written in terms of $poss_{patient(x)}$ and $poss_{oxygen(y)}$.

$$\forall xy.(poss_{patient(x)}(x) \wedge poss_{oxygen(y)}(y) \Rightarrow \neg adj(x, y))$$

The original definitions for *adj* together with the definitions for $poss_{patient(x)}$ and $poss_{oxygen(y)}$ comprise a complete theory, and the *poss* query above is entailed by that

[1] It is tempting to use $poss_{patient}(x)$ instead of $poss_{patient(x)}(x)$. However, in general we will have complex sentences in the subscripts of *poss* predicates, e.g. $poss_{\exists x.(p(x,y) \wedge \neg q(y,z))}(y, z)$, and it is desirable to ensure that every *poss* predicate has a sentence in the subscript and takes as many arguments as there are free variables in that sentence. When there are free variables, the order in which the variables appear in the subscript determines which arguments to the predicate correspond to which variables.

theory exactly when the original entailment query holds. This form of theory-completion is the subject of Chapter 4, which addresses several questions. How do we automatically detect the portion of a theory that is complete? How do we automatically construct the set of *poss* predicates necessary to answer the query? How do we construct the new query out of those *poss* predicates? And finally, how do we automatically construct a definition for a given *poss* predicate?

In summary, this thesis introduces algorithms for performing the transformations illustrated above: reformulating a (part of a) theory into a set of nonrecursive biconditionals, translating a set of nonrecursive biconditionals into *nr-datalog*$^{\neg}$, and transforming an automated deduction problem about an incomplete theory into an automated deduction problem about a complete theory using *poss* predicates. Together, the algorithms for performing these transformations are sufficient for building a system that performs Extensional Reasoning.

Chapter 3

Complete Theories

Complete theories are incredibly important for Extensional Reasoning because a complete theory corresponds very naturally to a *nr-datalog*$^\neg$ system. Every satisfiable, complete theory in Finite Herbrand Logic can be transformed into *nr-datalog*$^\neg$ while preserving logical equivalence. In this chapter, we discuss algorithms for recognizing that a set of sentences is complete and transforming such a sentence set into a *nr-datalog*$^\neg$ system. While the definition for a complete theory is well-known, it is such a central concern in Extensional Reasoning that we give it a proper definition to highlight its importance.

Definition 2 (Axiomatic Completeness). *A set of sentences Δ is complete for a language (set of sentences) L if and only if for every closed sentence $\phi \in L$,*

$$\Delta \models \phi \text{ or } \Delta \models \neg\phi \text{ or both.}$$

A set of sentences Δ is complete for a vocabulary (set of object constants and predicates) V if and only if Δ is complete for the language built from V.

Performing Extensional Reasoning for the case of complete theories requires two steps: recognizing that a theory is complete and then transforming a complete theory into *nr-datalog*$^\neg$. The first two sections of this chapter address these two problems. The third section puts the two steps together with soundness and completeness results, and the fourth section performs a comparison of traditional techniques with Extensional Reasoning in the case of complete theories.

3.1 Recognizing Complete Theories

In Finite Herbrand Logic, a satisfiable, complete theory is satisfied by exactly one model—by exactly one set of ground literals. Because there are finitely many ground literals in any language and entailment in FHL is decidable, checking whether a satisfiable FHL theory is complete is also decidable. For each ground atom a in the language, 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.

In the context of Extensional Reasoning, this algorithm is unacceptable because it relies on an algorithm for solving the very problem Extensional Reasoning is meant to confront: logical entailment. Consequently, we developed an alternate algorithm for checking completeness that has a syntactic flavor. It performs an inexpensive test that is sufficient for ensuring completeness; however, if a theory fails the test, we cannot conclude that the theory is incomplete.

The test our algorithm runs was inspired by work done on complete theories in the nonmonotonic reasoning literature. Predicate completion was one of the early techniques used to define the semantics of Negation as Failure in Logic Programming [Llo84]. When applied to a nonrecursive set of rules, predicate completion produces a nonrecursive set of biconditional sentences, e.g.

$$\begin{aligned} &p(x) \Leftrightarrow (q(x) \wedge r(x)) \\ &q(x) \Leftrightarrow (x = a \vee x = b) \\ &r(x) \Leftrightarrow (x = e \vee x = f). \end{aligned} \tag{3.1}$$

With the caveats mentioned below, a nonrecursive set of biconditional definitions in Finite Herbrand Logic is guaranteed to comprise a complete theory.

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

When we state that there is a definition for predicate p, it means there is a biconditional definition where the predicate in the head is p.

Definition 4 (Nonrecursive Biconditional Definitions). *A set of biconditional definitions Δ is nonrecursive if and only if the Directed 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, nonrecursive set of biconditional definitions in FHL, where there is exactly one definition for each predicate in Δ and no definition for $=$. Δ is complete.*

While for the most part obvious, this theorem does not hold for all logics. For example, it does not hold in first-order logic because equality does not have a fixed definition. The proof of this theorem and further information about why it does not hold for first-order logic can be found in Appendix A.2.1.

In this section we examine an algorithm that checks for axiomatic completeness by attempting to reformulate a sentence set into a logically-equivalent set of nonrecursive biconditional definitions. Though incomplete, this algorithm runs in low-order polynomial time.

Conceptually this algorithm can be broken down into two pieces: the first finds sentences that are likely to contribute to a biconditional definition and the second determines whether those sentences do in fact entail such a definition. The first piece selects sentences so that if each selection entails a biconditional, the group of biconditionals together are guaranteed to be nonrecursive and therefore axiomatically complete.

To select a group of sentences that are likely to contribute to a biconditional definition, 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 3.1 into clausal form.

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

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. All the sentences that were produced from that biconditional must therefore mention exactly one predicate besides equality.

In the example above, we can apply this observation to select the following sentences, which comprise the set of sentences that originated in the definition for q.

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

Next, the selected sentences are checked to see whether they actually do entail a biconditional definition. If so, the algorithm recurses on the remaining sentences, this time looking for sentences with a single predicate besides $=$ and the predicate just found to have a biconditional definition. If the selected sentences do not entail a biconditional, the algorithm selects another set of sentences from those remaining and the process repeats.

At recursion depth i, definitions for predicates $p_1, \ldots, p_i$ have been found and the sentences that entail those definitions have been removed. The algorithm selects sentences with a single predicate besides $\{p_1, \ldots, p_i, =\}$ and checks whether they entail a biconditional. The recursion stops when either the algorithm has found a biconditional for every predicate, or there is no selection of the sentences remaining that entail a biconditional. This algorithm is a variation on the context-free grammar (CFG) marking algorithm for

determining whether the grammar is empty: when a biconditional for p is found, all occurrences of p in the remaining sentences are marked, and when a sentence has all but one of its predicates marked, it is a candidate for selection.

In the example, the selected sentences entail

$$q(x) \Leftrightarrow (x = a \vee x = b),$$

and the remaining sentences are

$$\begin{aligned}
&p(x) \vee \neg q(x) \vee \neg r(x)\\
&\neg p(x) \vee q(x)\\
&\neg p(x) \vee r(x)\\
&r(x) \vee x \neq e\\
&r(x) \vee x \neq f\\
&\neg r(x) \vee x = e \vee x = f.
\end{aligned}$$

After recursing, the last three sentences are selected and found to entail the biconditional

$$r(x) \Leftrightarrow (x = e \vee x = f),$$

leaving just the sentences

$$\begin{aligned}
&p(x) \vee \neg q(x) \vee \neg r(x)\\
&\neg p(x) \vee q(x)\\
&\neg p(x) \vee r(x).
\end{aligned}$$

Since the algorithm has already found definitions for q and r, all of the remaining sentences are selected for the definition of p.

$$p(x) \Leftrightarrow (q(x) \wedge r(x))$$

Algorithm 1 embodies the procedure outlined here. Each recursive call selects all those sentences with the same unmarked predicate p and calls REFORMULATE-TO-BICOND on those sentences. If the sentences entail a biconditional, REFORMULATE-TO-BICOND will return that biconditional; otherwise, REFORMULATE-TO-BICOND will return false. If the selected sentences 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
$Preds(d)$ contains just p and members of $basepreds\}$
2: $preds := \{p | \langle p, d\rangle \in sents\}$
3: $bicond :=$ false
4: **for all** $p \in preds$ **do**
5: $partition := \{d | \langle p, d\rangle \in sents\}$
6: $bicond :=$ REFORMULATE-TO-BICOND($partition$, p)
7: $pred := p$
8: **when** $bicond \neq$ false **then** exit **for all**
9: **end for**
10: **when** $bicond =$ false **then return** false
11: $remaining := \Delta - partition$
12: **when** $remaining = \{\}$ **then return** $\{bicond\}$
13: $rest :=$ TO-BICONDS($remaining$, $\{pred\} \cup basepreds$)
14: **when** $rest \neq$ false **then return** $\{bicond\} \cup rest$
15: **return** false

The properties of TO-BICONDS are dependent on the properties of REFORMULATE-TO-BICOND, an algorithm for answering metalevel entailment queries like $\Gamma \models p(x_1, x_2) \Leftrightarrow \phi(x_1, x_2)$. Our working implementation of REFORMULATE-TO-BICOND is sound but incomplete and runs in time polynomial in the size of the given sentences. The analysis of TO-BICONDS that follows explicitly references the properties of REFORMULATE-TO-BICOND that are needed for the analysis to hold, thereby separating the two components of this algorithm: the sentence selection and the metalevel entailment check.

TO-BICONDS runs in $O(|Preds(\Delta)|^2(|\Delta| + f_{\text{REFORMULATE-TO-BICOND}}(\Delta)))$, where $|Preds(\Delta)|$ is the number of predicates in Δ, $|\Delta|$ is the size of Δ, and $f_{\text{REFORMULATE-TO-BICOND}}(\Delta)$ is the maximum cost of REFORMULATE-TO-BICOND run on any subset of Δ. To see this, notice that TO-BICONDS will recurse at most once for each of the $|Preds(\Delta)|$ predicates because the algorithm recurses only after having found a biconditional for some predicate that did not already have a definition. Each of these $|Preds(\Delta)|$ recursive calls costs at most $O(|Preds(\Delta)|(|\Delta| + f_{\text{REFORMULATE-TO-BICOND}}(\Delta)))$. To see this, we suppose that each recursive call computes *sents*, the set of sentences that could entail a biconditional for some unmarked predicate, in time linear in the size of the theory Δ. Then, for each of the unmarked

predicates that occur in *sents*, the algorithm walks over *sents* and runs REFORMULATE-TO-BICOND. In the worst case there are $|Preds(\Delta)|$ such unmarked predicates, producing a cost of $O(|Preds(\Delta)|(|\Delta| + f_{\text{REFORMULATE-TO-BICOND}}(\Delta)))$ for each recursive call. Thus in the worst case, there are $|Preds(\Delta)|$ recursive calls, each of which costs $O(|Preds(\Delta)|(|\Delta| + f_{\text{REFORMULATE-TO-BICOND}}(\Delta)))$, giving us the $O(|Preds(\Delta)|^2(|\Delta| + f_{\text{REFORMULATE-TO-BICOND}}(\Delta)))$ upper bound complexity.

The polynomial complexity is achieved in this case by abandoning the completeness of the algorithm. Even assuming that REFORMULATE-TO-BICOND is complete, TO-BICONDS is incomplete; that incompleteness is precisely characterized by the completeness theorem that follows the soundness theorem below. The theorems rely on the soundness and completeness of REFORMULATE-TO-BICOND. Their proofs can be found in Appendix A.2.2.

Theorem 2 (Soundness of TO-BICONDS). *Under the conditions listed below, if* TO-BICONDS$(\Delta, \{=\})$ *returns a nonempty* Γ*, then* Γ *is a nonrecursive set of 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 nonrecursive 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 false.*

- Δ *is a satisfiable, nonempty sentence set.*
- Δ *is logically equivalent to a nonrecursive set of biconditionals with one definition per predicate besides equality:* $\{\beta_1, \ldots, \beta_n\}$*.*
- *There is a partitioning of* Δ *into* $S_1, \ldots, S_n$ *such that* β_i *is logically equivalent to* S_i *and* $Preds(\beta_i) = Preds(S_i)$*, 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 false.*

The incompleteness of TO-BICONDS can be illustrated with an example. The example is a complete theory that is written in a way such that TO-BICONDS does not select the appropriate sentences to give to REFORMULATE-TO-BICOND; hence, even though REFORMULATE-TO-BICOND is complete, TO-BICONDS fails to recognize that fact.

$$p(x) \Leftarrow q(x)$$
$$q(x) \Leftarrow p(x)$$
$$p(x) \Leftarrow x = a$$
$$q(x) \Leftarrow x = b$$

In this theory, the only object constants are a and b. Certainly $p(a)$ and $q(b)$ are entailed, and by applying modus ponens, we see that $q(a)$ and $p(b)$ are entailed using the first two implications. The theory is therefore complete since p and q are both true of all object constants in the universe; moreover, the theory is satisfiable since the model $\{p(a), p(b), q(a), q(b)\}$ satisfies each of the sentences.

When TO-BICONDS is run on the sentences, it finds all those sentences with one unmarked predicate besides $=$, which consists of just the second two sentences. Then for each of p and q, it runs REFORMULATE-TO-BICOND on $\{p(x) \Leftarrow x = a\}$ and $\{q(x) \Leftarrow x = b\}$, respectively. Since neither is a complete theory by itself, TO-BICONDS fails even though the theory is complete and satisfiable.

TO-BICONDS relies on REFORMULATE-TO-BICOND, an algorithm for checking whether a given set of sentences entails a biconditional definition. Implementations of REFORMULATE-TO-BICOND 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 a set of sentences 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 by first normalizing both and then using a simple syntactic equality check.

In the example above where TO-BICONDS succeeds, the last set of sentences to be selected were those that were derived from the biconditional for p.

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

These sentences can be written as necessary and sufficient conditions for p.

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

After performing a syntactic equality check we see that these implications entail a biconditional definition for p. Likewise the other sets of sentences can be shown to entail biconditional definitions for q and r, producing the following set of biconditionals. It is noteworthy that the way the sentences were selected guarantee that the set of biconditionals as a whole are nonrecursive, which is why there is no explicit check for recursion.

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

Clearly more work could be done in this area to enlarge the class of complete theories that can automatically be detected. We have chosen to investigate inexpensive algorithms in the hopes of making Extensional Reasoning attractive to people who are building more traditional theorem provers today. Our algorithms cost so little to run and (as we will see) can sometimes produce such substantial speedups that they are worth running even if they are successful in only a small percentage of cases. On the other hand, algorithms that could detect a larger class of complete theories may be necessary as there are fundamental features of complete theories that severely hamper traditional theorem proving approaches (see Section 3.4.1). For some problems, it may well be worth paying a larger cost to detect completeness so that Extensional Reasoning techniques can be employed.

3.2 Translating Complete Theories into *nr-datalog*$^\neg$

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

Translating a nonrecursive set of biconditionals in FHL into *nr-datalog*$^\neg$ takes place in two steps. First, the predicates in the language are partitioned into the portion that will represent extensions (explicit tables) and the portion that will represent intensions (implicit

tables). This process will be referred to as PARTITION. Second, explicit tables are built for the extensional predicates, and *nr-datalog*$^\neg$ view definitions are built for the intensional predicates.

The implementation of PARTITION reported in this thesis 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 those predicates whose definition mentions only equality to the extensional portion of the theory and all the remaining predicates to the intensional portion. More sophisticated partitioning algorithms are the subject of future work. See Section 5.16 for results from the database literature that will inform if not entirely address this issue.

Regardless of the implementation of PARTITION, the algorithms for constructing the corresponding *nr-datalog*$^\neg$ system are well-known.

Creating a *nr-datalog*$^\neg$ view definition for an intensional predicate when given its biconditional definition can be performed using the well-known Lloyd-Topor transformation [Llo84, LT84] followed by a linear-time post-processing step that ensures safety. Lloyd-Topor turns $\forall$ into $\neg\exists\neg$; it introduces new predicates when $\neg$ is applied to a complex sentence; boolean connectives are treated as usual. We demonstrate Lloyd-Topor by showing a step-by-step transformation of $r(x) \Leftrightarrow \forall y.(\neg p(y) \vee q(x,y))$.

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

The result is two rules: (1) is the definition for r, and (2) is the definition for *newreln*. Notice that neither rule is safe—there are variables in negative literals that do not occur in positive literals in the body. To be valid *nr-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 here.

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

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

VIEWS-TO-DATALOG can also be used to construct the extensional tables. Suppose we convert all the biconditionals into *nr-datalog*$^\neg$ views using VIEWS-TO-DATALOG, treating $=$ as an uninterpreted predicate. We can then add in a table for $=$ and *univ*, each of which has one row per element in the universe. Then we can materialize a table by constructing the appropriate *nr-datalog*$^\neg$ query and running it through a standard deductive database engine. Other approaches are feasible, but this one is perhaps the most straightforward.

Our partitioning scheme makes the implementation of PARTITION particularly simple because the only biconditionals necessary for constructing the tables are those whose bodies are defined entirely in terms of $=$. Taking advantage of this information, we can construct the extensional table for some predicate by adding the rule $x = x$ to the definition for that predicate, apply Lloyd-Topor to the biconditional, and invoke a Prolog engine to compute the table.

Regardless of the implementation that constructs the extensional tables, we will refer to the algorithm with the name MATERIALIZE. We assume that MATERIALIZE will always construct create an extensional definition for *univ* in case *univ* is used by VIEWS-TO-DATALOG. It is noteworthy that the implementations of MATERIALIZE and PARTITION can affect one another, as illustrated here.

Putting all the pieces together, PARTITION, MATERIALIZE, VIEWS-TO-DATALOG, and SENTENCE-TO-DATALOG are bundled together into BICONDS-TO-DATALOG, Algorithm 2, a procedure for converting an entailment query in FHL about a nonrecursive set of biconditionals into *nr-datalog*$^\neg$. PARTITION returns the set of predicates E that will be represented extensionally, leaving the remaining predicates to be represented intensionally. Then the tables for the predicates E are materialized, the view definitions for the remaining predicates are constructed, and the query is turned into a *nr-datalog*$^\neg$ query.

Because in general the definitions for extensional predicates may rely on the definitions

for intensional predicates, the algorithm MATERIALIZE must be given the definitions for the extensional and the intensional predicates. VIEWS-TO-DATALOG does not require the definitions for the extensional predicates, but given those definitions, it might be able to optimize the view definitions. Additionally, SENTENCE-TO-DATALOG when given the sentence ϕ returns a new query ψ possibly with a set of additional *nr-datalog*$^\neg$ rules, since some *nr-datalog*$^\neg$ evaluation engines only accept a restricted query language, e.g. literals or conjunctive queries.

Algorithm 2 BICONDS-TO-DATALOG(Δ, ϕ)

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

1: $E :=$ PARTITION($Preds(\Delta)$, Δ)
2: $I := Preds(\Delta) - (E \cup \{=\})$
3: $\Gamma :=$ MATERIALIZE(E, Δ)
4: $\Lambda :=$ VIEWS-TO-DATALOG(I, Δ)
5: $(\psi, \Theta) :=$ SENTENCE-TO-DATALOG(ϕ)
6: **return** $(\Gamma, \Lambda \cup \Theta, \psi)$

BICONDS-TO-DATALOG (Algorithm 2) ensures that Δ entails ϕ under FHL semantics if and only if $\Gamma \cup \Lambda \cup \Theta$ entails ψ under *nr-datalog*$^\neg$ semantics. BICONDS-TO-DATALOG ensures something much stronger as well: Δ and $\Gamma \cup \Lambda$ essentially represent the same exact model. This ensures that all the logical consequences are exactly the same.

The theorem that follows gives conditions on PARTITION, MATERIALIZE, VIEWS-TO-DATALOG, and SENTENCE-TO-DATALOG that are sufficient for ensuring equivalence is preserved. The theorem holds for any implementation that meets these criteria.

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

- $E =$ PARTITION(Δ). *E is a subset of the predicates mentioned in Δ and does not include $=$.*
- $\Gamma =$ MATERIALIZE(E, Δ). *For every ground atom ρ built using a predicate in E, $\Delta \models_{FHL} \rho$ if and only if $\Gamma \models_D \rho$.*

- $I = Preds(\Delta) - (E \cup \{=\})$ *and* $\Lambda =$ VIEWS-TO-DATALOG(I,Δ). *For any set* B *of biconditional definitions for the predicates in* E*, use* Δ_B *to denote* Δ *where the definitions for* E *are replaced by* B*. For every ground atom* ρ *built using a predicate in* I *and every finite, nonrecursive set* B *of biconditional definitions for the predicates defined in* E*,* $\Delta_B \models_{FHL} \rho$ *if and only if* MATERIALIZE$(E, \Delta_B) \cup \Lambda \models_D \rho$.

- $(\psi, \Theta) =$ SENTENCE-TO-DATALOG(ϕ). ψ *mentions a subset of the predicates in* Θ*, which is a set of nr-datalog*$^\neg$ *rules. Consider any model* M *with interpretations for exactly those predicates mentioned in* Δ*. Use* M_T *to denote the representation of* M *using extensional tables.* M *satisfies* ϕ *if and only if* $M_T \cup \Theta \models_D \psi$.

- *The predicates introduced by* VIEWS-TO-DATALOG *and* SENTENCE-TO-DATALOG*, if any, are disjoint.*

The proof of this theorem can be found in Appendix A.2.3.

3.3 Extensional Reasoning for Complete Theories

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 *nr-datalog*$^\neg$. Then an algorithm for processing *nr-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 *nr-datalog*$^\neg$ query and FHL-ENTAILEDP to answer an arbitrary entailment query in FHL in the case of an incomplete theory.

Algorithm 3 ER-ENTAILEDP(Δ, ϕ)

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

```
1: Σ := TO-BICONDS(Δ)
2: if Σ ≠ false 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, as guaranteed by Theorem 5, whose proof can be found in Appendix A.2.4.

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

- FHL-ENTAILEDP *is sound and complete for $\models_{FHL}$.*
- DB-ENTAILEDP *is sound and complete for $\models_{D}$.*

One of the implications of this theorem bridges the gap between how the semantics of logic and the semantics of *nr-datalog*$^\neg$ treat negation. Logic uses classical negation whereas *nr-datalog*$^\neg$ uses negation as failure (NAF). While NAF is often thought of as 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 Theorem Proving Techniques

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

3.4.1 Theoretical Comparison

NAF is used pervasively in computer science, and in fact classical negation is the interpretation that is uncommon. The majority of this section argues that a priori NAF can be a powerful rule of inference—one that should be used where possible. To do that, we compare two proof procedures: one that uses NAF, SLDNF resolution (a top-down *nr-datalog*$^\neg$ evaluation procedure), and one that treats negation classically, model elimination (a top-down resolution-style procedure). SLDNF resolution versus model elimination is a good comparison since the two procedures can be implemented so that they differ only in how they treat negation.

Consider the following biconditional definition for p.

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

We are going to compare how SLDNF resolution and model elimination process this biconditional, but to do that we must first convert the biconditional into the language each of the procedures accepts as input. SLDNF resolution requires *nr-datalog*$^\neg$ whereas model elimination requires clausal form.

Converting the biconditional above to *nr-datalog*$^\neg$ 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 one $p(x)$ implication for each of the *nr-datalog*$^\neg$ rules above along with implications for $\neg p(x)$. The naïve clausal form conversion algorithm constructs 27 implications for $\neg p(x)$, as indicated below, which is exponential in the size of the biconditional. [1]

[1] Structure-preserving clausal form conversion (Chapters 5 and 6 of [RV01]) will produce a polynomial number of clauses, but as will be discussed shortly, the exponential will still appear in the model elimination search space.

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

Now we can compare SLDNF resolution on the *nr-datalog*$^\neg$ with model elimination on the result of the naïve clausal form conversion. For the entailment query $\exists x.p(x)$, the two perform identically (assuming we turn off the reduction operation [AS92] for model elimination, which would only slow down model elimination as it is not necessary for completeness in this case); 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. 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 noted earlier, structure-preserving clausal-form conversion will avoid enumerating the exponential number of rules with $\neg p(x)$ in the head, but that does not necessarily prevent model elimination or 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.[2]

Lemma 1. *Consider the following biconditional β.*

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

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

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

Suppose a clausal form conversion algorithm converts β into Γ. Further suppose Γ 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 up to subsumption.*

$$\text{RES}(\Gamma \cup \{p(t)\}) \textit{ will produce at least } \prod_i n_i \textit{ clauses.}$$

Proof. Here we argue somewhat informally. Consider a clause with a negative p literal from the list above.

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

This clause, which is entailed by β, entails

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

[2]Many theorem provers today are not generatively complete but are only refutation complete: they will find a proof if there is one, but they will not necessarily enumerate all of the clausal consequences up to subsumption. Generative completeness is used in the lemma because it is sufficient to guarantee the result. We conjecture that even if a theorem prover is only refutation complete, asking a query about a set of nonrecursive biconditionals will sometimes result in the same exponential search space that is guaranteed by the lemma. Finding the precise statement of this conjecture and producing the proof are the subject of future work.

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 $\neg p(x)$ and the negated query $p(t)$, resolution can find a proof in one step, without enumerating the exponential number of consequences described above. Hence, it is appropriate to designate the result above as a lemma since it is useful mainly for proving other results about certain classes of theories.

The last few paragraphs of this section are devoted to demonstrating the opposite of what the rest of the section demonstrated: that traditional techniques are sometimes more efficient than Extensional Reasoning techniques even in the case of complete theories.

Consider the very simple entailment query where the premise set is a single sentence: $\forall xy.p(x,y) \models \forall xy.p(x,y)$. Resolution will prove that the entailment holds in one step, whereas Extensional Reasoning may require n^2 steps, where n is the size of the universe.

In the case of resolution, the premise set in clausal form is just $p(x,y)$, and the clausal form of the negation of the query is $\neg p(k_1,k_2)$. These being the only two sentences available, resolution will require at most one step to find the empty clause.

In the case of Extensional Reasoning, in the worst case, the decision is made to materialize a table for the predicate p, which consists of n^2 two-tuples, where n is the size of the universe. Then, using SLDNF resolution, the proof attempt $\forall xy.p(x,y)$ is performed by attempting to find a pair of elements in the universe that are false in p by enumerating all such pairs. Since every pair is true in p, the proof costs n^2, even if we assume perfect indexing.

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.

3.4.2 Empirical Comparison

Next we report on experiments designed to compare Extensional Reasoning techniques to traditional techniques in the case of complete theories. All tests were performed on complete theories written as nonrecursive biconditionals where the queries were ground literals. The traditional techniques consisted of (1) a first-order theorem prover, (2) a first-order model builder, and (3) a boolean SAT solver.

The *nr-datalog*$^\neg$ evaluation engine used throughout the tests was the Stanford Logic Group's SLDNF resolution implementation, which is now a part of Epilog, the group's automated reasoning library. The theorem prover used was Darwin[3] and was chosen because it performed the best of all the systems available on Systems On TPTP[4] for the hardest test problems. Likewise, Paradox[5] was chosen as the model builder because it performed best on the hardest tests. The SAT solver used was Minisat[6], chosen because it won the 2005 SAT solver competition. All tests were run on a 1.5 GHz PowerPC G4 processor with 1.25 GB of RAM.

Methodologically, it was unclear at the start how to compare these systems because each type of system requires its own translator from Finite Herbrand Logic into the language it accepts as input. The naïve methods for performing those translations are as follows. Below $\{a_1, \ldots, a_n\}$ is the set of object constants in the vocabulary.

Extensional Reasoning The purpose of Extensional Reasoning is to convert a problem in FHL into a *nr-datalog*$^\neg$ system. Because all the premises start as a set of nonrecursive biconditionals, those biconditionals can be translated into *nr-datalog*$^\neg$ as described in Section 3.2. All those biconditionals where the body of the definition mentions only the = predicate are turned into extensional tables; all the others are turned into intensional tables.

[3] `http://combination.cs.uiowa.edu/Darwin/`
[4] `http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP`
[5] `http://www.cs.chalmers.se/~koen/folkung/`
[6] `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/`

SAT Solver A SAT solver takes as input a set of propositional sentences, and Minisat like many other SAT solvers, requires those propositional sentences to be expressed in conjunctive normal form; thus, the translation for the SAT solver requires propositionalization followed by conversion to CNF. The simplest approach first propositionalizes each biconditional and then converts the result to clausal form. Propositionalization can be performed by replacing $\forall x.\phi(x)$ with $\phi(a_1) \wedge \cdots \wedge \phi(a_n)$ and likewise replacing $\exists x.\phi(x)$ with $\phi(a_1) \vee \cdots \vee \phi(a_n)$.

Theorem Prover A first-order theorem prover takes as input sentences in first-order logic. The only translation needed to ensure sentences in FHL meet this requirement is the addition of a domain closure axiom $\forall x.(x = a_1 \vee \cdots \vee x = a_n)$ and unique names axioms:

$$\begin{gathered} a_1 \neq a_2 \\ a_1 \neq a_3 \\ a_1 \neq a_n \\ \vdots \\ a_{n-1} \neq a_n \end{gathered}$$

Model Builder Same as for the theorem prover.

During preliminary testing, we discovered that the naïve translations worked well for Extensional Reasoning and the model builder; however, the naïve translations did not work as well for the SAT solver or the theorem prover. In fact, the run times (including both the reformulation and the evaluation) were high enough for these latter two techniques to make gathering a reasonable amount of data quite time consuming even for simple examples; consequently, the conversion algorithms we used for the SAT solver and theorem prover were more sophisticated than the naïve approach.

The more sophisticated translation for the theorem prover attempts to reduce the number of sentences given as input, causing what is sometimes a substantial reduction in search space. For a positive atomic query, the translation does not include the DCA or the UNA and every biconditional $r(\overline{x}) \Leftrightarrow \phi(\overline{x})$ is turned into $r(\overline{x}) \Leftarrow \phi(\overline{x})$. For the examples tested, this transformation preserves entailment while reducing the size of the theorem prover's search space. For a negative atomic query, the translation does not include the DCA (but does include the UNA), and it turns every biconditional $r(\overline{x}) \Leftrightarrow \phi(\overline{x})$ into $\neg r(\overline{x}) \Leftarrow \neg\phi(\overline{x})$. Again the translation preserves entailment for the queries tested. In both cases, the more

sophisticated transformation resulted in a substantial improvement over the naïve transformation, and because one could imagine a system that automatically detects that such transformations preserve entailment, the results acquired using this transformation seem reasonable. In general, however, these transformations do not preserve entailment, and the theorem prover performance may be significantly worse than what is reported here. Since the theorem prover is a competitor to Extensional Reasoning, reporting optimistic results is better than no results at all.

For the SAT solver, the more sophisticated translation method differs from the naïve translation more substantially than was the case with the theorem prover. The problem with the naïve translation is that the cost of converting the sentences to conjunctive normal form after propositionalizing can be expensive. Consider a simple example.

$$\exists x.(p(x) \wedge q(x))$$

The propositionalization produces a large disjunction of conjunctions.

$$(p(a_1) \wedge q(a_1)) \vee \cdots \vee (p(a_n) \wedge q(a_n))$$

The clausal form has 2^n clauses.

$$\begin{gathered}\{p(a_1), \ldots, p(a_{n-1}), p(a_n)\} \\ \{p(a_1), \ldots, p(a_{n-1}), q(a_n)\} \\ \{p(a_1), \ldots, q(a_{n-1}), p(a_n)\} \\ \vdots \\ \{q(a_1), \ldots, q(a_{n-1}), q(a_n)\}\end{gathered}$$

A simple technique for addressing this issue is the well-known formula renaming technique (Chapters 5 and 6 of [RV01]), which is at the heart of structure-preserving clausal form conversion. The idea is to introduce fresh predicates that are made equivalent to complex sentences and then replace those complex sentences with the new predicates. While semantically equivalent, the simplified form of the sentences makes transforming the result of propositionalization to clausal form less computationally demanding.

We experimented with several different combinations of formula renaming, propositionalization, and clausal form conversion. Among the options, the one we chose has the smallest

reformulation cost (for our test cases), which seemed to be the right choice because the majority of the cost in answering queries with the SAT solver was due to the reformulation. Our transformation algorithm first performs formula renaming on all the biconditionals, then propositionalizes, and finally converts the result to clausal form.

Applying formula renaming to the example above requires inventing a new predicate, say s, with a single argument. $s(x)$ is made equivalent to $p(x) \wedge q(x)$ so that the sentence above is turned into the following two sentences.

$$\forall x.(s(x) \Leftrightarrow (p(x) \wedge q(x))$$
$$\exists x.s(x)$$

The propositionalization of these sentences is then

$$(s(a_1) \Leftrightarrow (p(a_1) \wedge q(a_1)) \wedge \cdots \wedge (s(a_n) \Leftrightarrow (p(a_n) \wedge q(a_n))$$
$$s(a_1) \vee \cdots \vee s(a_n).$$

The clausal form of these sentences is linear in the size of the two sentences above.

$$\{s(a_1), \neg p(a_1), \neg q(a_1)\}$$
$$\vdots$$
$$\{s(a_n), \neg p(a_n), \neg q(a_n)\}$$
$$\{\neg s(a_1), p(a_1)\}$$
$$\vdots$$
$$\{\neg s(a_n), p(a_n)\}$$
$$\{\neg s(a_1), q(a_1)\}$$
$$\vdots$$
$$\{\neg s(a_n), q(a_n)\}$$
$$\{s(a_1), \ldots, s(a_n)\}$$

This transformation algorithm is more complex than what we had originally thought necessary to get reasonable data for the SAT solver. Work has been done to investigate how to efficiently propositionalize a theory [Sch02, RA05, GA07], but none of these techniques produced savings for the class of theories we considered.

Using these translators, four tests based on axiomatizations of the grid shown in Figure 3.1 were given to each of the systems. The first two tests consisted of one positive query

and one negative query about which cells were adjacent to which other cells[7]; all systems reported results. The second two tests also consisted of one positive query and one negative query, this time about which cells were located to the west of which other cells; only the theorem prover and Extensional Reasoning reported results for the positive query, and only Extensional Reasoning reported results for the negative query. All tests were entailment queries where entailment held. All reported times include the time required to reformulate the given FHL query into the language accepted by the system and the time required to answer the query once the transformation was complete.

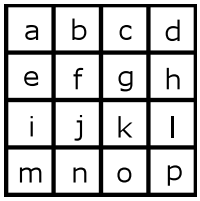

Figure 3.1: A 4×4 grid

To get curves for each system, the size of the grid was varied from 4×4 to 20×20. This means that the number of cells (the universe in each theory) ranged from size 16 to size 400. In the following figures, the x-axis represents the number of cells for a particular problem, and the y-axis represents the logarithm of the time required to answer the query. A logarithmic scale is used because it clearly illustrates the running time difference between systems that have an exponential range.

Figure 3.2 compares answering an entailment query $adj(a, b)$, where a and b are cells that are in fact adjacent to one another. Because the graph includes both the cost of reformulation and the cost of evaluation, as the size of the grid increases, the cost of every system must increase since at the very least each system must read the description of the grid, which increases linearly with the size of the grid.

Extensional Reasoning was at least an order of magnitude faster than the other three systems. Moreover, if the cost of reformulation were removed, the curve for Extensional Reasoning would be perfectly flat; the size of the grid would be irrelevant because of the database's indexing. For the other systems, it is noteworthy that the model builder did not report results higher than 15×15 because it reported out of memory errors. Also note that

[7]Recall from Chapter 2 that the definition for adjacency, *adj*, is the symmetric closure of the definition for *edge*, which is just a table that captures the topology of the grid.

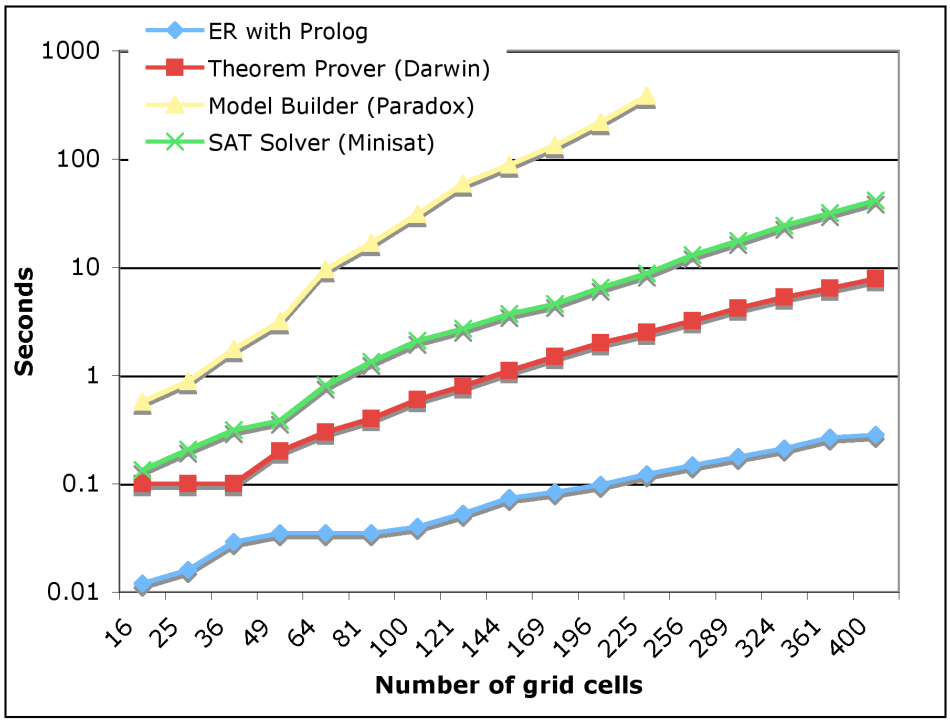

Figure 3.2: Comparison for an entailed query: $adj(a, b)$

the theorem prover was faster than both the SAT solver and the model builder.

Figure 3.3 compares results for the query $\neg adj(a, b)$ where a and b are cells that are not adjacent to one another. Again we see that Extensional Reasoning is faster by about an order of magnitude than the nearest competitor, but this time the nearest competitor is the SAT solver. The theorem prover is slower than all of the other systems and timed out for grids larger than 10×10. The model builder again ran out of memory, limiting its results.

The interesting thing about the first two tests is that for Extensional Reasoning, once the premise set has been converted to a *nr-datalog*$^\neg$ query, determining whether $adj(a, b)$ or $\neg adj(a, b)$ are entailed takes constant time. The next two tests do not have this feature; even after converting to *nr-datalog*$^\neg$, answering the query takes time proportional to the size of the grid. These queries are again about grids like the one shown in Figure 3.1 and are based on a complete definition for the binary predicate $westof$, which defines whether or not a cell is located to the west of another cell. The definition for $westof$ is broken down as follows.

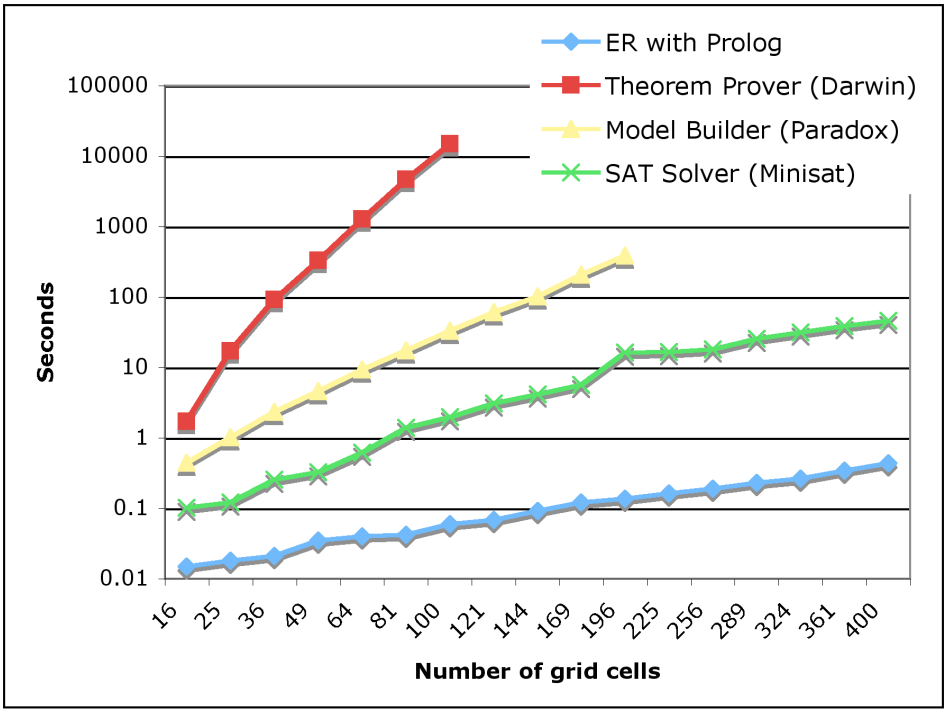

Figure 3.3: Comparison for an entailed query: $\neg adj(a, b)$

- $west(x, y)$: cell x is the cell immediately west of cell y.
- $north(x, y)$: cell x is the cell 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 in the same column as cell y for two different cells x and y.
- $westof(x, y)$: 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.

The formal definitions for a 4×4 grid are below.

$$westof(x,y) \Leftrightarrow (duewest(x,y) \vee \exists z.(vert(x,z) \wedge duewest(z,y)))$$
$$vert(x,y) \Leftrightarrow (duenorth(x,y) \vee duenorth(y,x))$$
$$duewest(x,y) \Leftrightarrow \left(\begin{array}{l} west(x,y) \vee \\ \exists z.(west(x,z) \wedge west(z,y)) \vee \\ \exists zw.(west(x,z) \wedge west(z,w) \wedge west(w,y)) \end{array} \right)$$
$$duenorth(x,y) \Leftrightarrow \left(\begin{array}{l} north(x,y) \vee \\ \exists z.(north(x,z) \wedge north(z,y)) \vee \\ \exists zw.(north(x,z) \wedge north(z,w) \wedge north(w,y)) \end{array} \right)$$
$$west(x,y) \Leftrightarrow \left(\begin{array}{l} (x = a \wedge y = b) \vee (x = b \wedge y = c) \vee (x = c \wedge y = d) \vee \\ (x = e \wedge y = f) \vee (x = f \wedge y = g) \vee \ldots \end{array} \right)$$
$$north(x,y) \Leftrightarrow \left(\begin{array}{l} (x = a \wedge y = e) \vee (x = e \wedge y = i) \vee (x = i \wedge y = m) \vee \\ (x = b \wedge y = f) \vee (x = f \wedge y = j) \vee \ldots \end{array} \right)$$

Figure 3.4 compares the results for the query $westof(a,b)$ where cell a is actually located to the west of cell b. The reason there is no data for the model builder or the SAT solver is that in both cases the third data point required more than 1000 seconds. In fact, the conversion to propositional logic for the 5×5 grid ran for over eight hours without finishing. The theorem prover, however, did report results. Comparing the Extensional Reasoning curve to the theorem prover curve, we can see that the two curves appear to be diverging, which assuming the trend continues, means that asymptotically ER outperforms the theorem prover on this test.

Figure 3.5 shows the results for the query, $\neg westof(a,b)$ where cell a is not located to the west of cell b. Just like the last query, there are no results for the model builder or SAT solver, but this time there are no results for the theorem prover either. Remember that for the two *adj* queries, the theorem prover performed better than the model builder and SAT solver on the positive query but worse on the negative query. The same thing happened with the *westof* queries, and the theorem prover failed to report results just like the model builder and SAT solver. Thus for the query $\neg westof(a,b)$, the graph contains only one curve: the one for Extensional Reasoning.

These experiments compare how quickly various systems can prove theorems when the

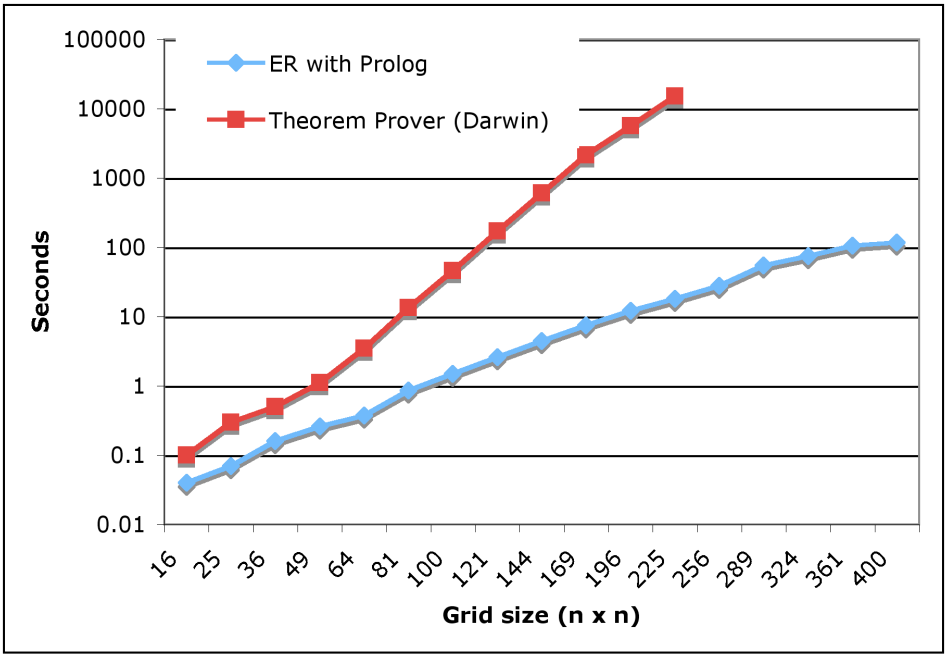

Figure 3.4: Comparison for an entailed query: $westof(a, b)$

majority of the cost comes from searching through data, a characteristic of the problems databases were built to solve. For such queries, we would expect Extensional Reasoning to outperform traditional techniques, and for the examples discussed in this section, that certainly was the case.

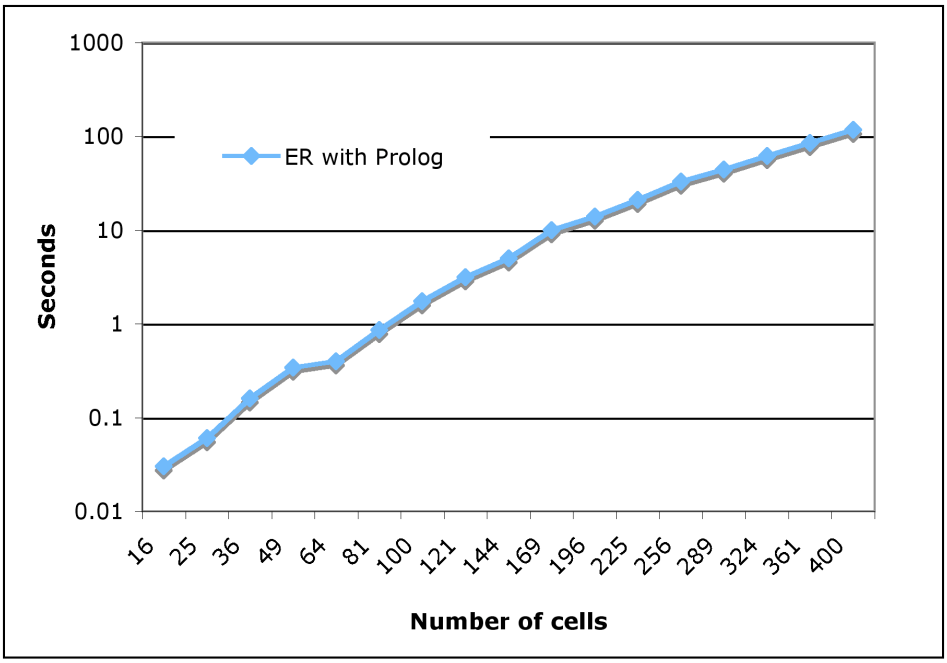

Figure 3.5: Comparison for an entailed query: $\neg westof(a, b)$

Chapter 4

Incomplete Theories

A complete theory is satisfied by a single model, which can be represented by a single database system, whereas an incomplete theory is satisfied by multiple models, each of which can be represented by a database system. The last chapter detailed techniques for automatically constructing database systems for reasoning about complete theories. This chapter generalizes those techniques by introducing algorithms for automatically constructing database systems for reasoning about incomplete theories.

The theories studied so far have been axiomatically complete, meaning that for every closed sentence in the language, the theory entails the sentence or its negation. Really, though, completeness is a relative concept: a theory is complete with respect to some set of sentences. For example, the theory with no premises is complete with respect to the set of tautologies (together with their negations). In general, every theory is complete with respect to the set of all its logical consequences (together with their negations).

One way to approximate the set of sentences for which a theory is complete is to use the set of predicates that have complete definitions. In Finite Herbrand Logic, for example, every theory is complete with respect to the set of all sentences that mention just the equality predicate since the only objects in the universe are the object constants in the vocabulary, and each pair of distinct object constants is unequal. Thus, every theory in FHL is complete with respect to at least $=$. A theory may be complete with respect to other predicates; a complete theory is complete with respect to all the predicates in the vocabulary.

In the context of Extensional Reasoning, this observation is useful because we can now conceptualize a theory as being decomposed into the complete portion C and the incomplete

portion I. The complete portion contains definitions for the complete predicates, and the incomplete portion contains constraints on the remaining predicates. The last chapter showed that when I is empty, we can use a database to reason about the theory sometimes orders of magnitude more efficiently than traditional techniques. As we will see, those speedups are sometimes large enough to absorb some extra overhead for dealing with theories where I is nonempty.

One of the benefits of the TO-BICONDS algorithm introduced as Algorithm 1 in the last chapter is that a three-line change turns it into an algorithm that can partition a given theory into $C \cup I$. Recall that TO-BICONDS takes as input a set of sentences and returns a nonrecursive set of biconditionals that is logically equivalent to the input if the algorithm finds one biconditional for each predicate; otherwise, it returns false. The new algorithm takes a sentence set and returns as many biconditionals as it finds. Those predicates with biconditional definitions are complete; the remaining predicates are incomplete.

TO-BICONDS-MAX, shown in Algorithm 4, was constructed from TO-BICONDS, Algorithm 1, by changing line (10) so that it returns the empty set instead of false when no biconditional can be found and replacing lines (14) and (15) so that the algorithm never throws away a biconditional. 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 perform the partitioning.

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

1: $sents := \{\langle p, d\rangle | d \in \Delta$ and $p \notin basepreds$ and
$Preds(d)$ contains just p and members of $basepreds\}$
2: $preds := \{p | \langle p, d\rangle \in sents\}$
3: $bicond :=$ false
4: **for all** $p \in preds$ **do**
5: $partition := \{d | \langle p, d\rangle \in sents\}$
6: $bicond :=$ REFORMULATE-TO-BICOND($partition$, p)
7: $pred := p$
8: **when** $bicond \neq$ false **then** exit **for all**
9: **end for**
10: **when** $bicond =$ false **then return** $\{\}$
11: $remaining := \Delta - partition$
12: **when** $remaining = \{\}$ **then return** $\{bicond\}$
13: $rest :=$ TO-BICONDS-MAX($remaining$, $\{pred\} \cup basepreds$)
14: **return** $\{bicond\} \cup rest$

This partitioning of a theory into the complete portion and the incomplete portion can be illustrated with the hospital sensor net example from Chapter 2. Recall that the hospital is represented as a grid of cells, and the adjacency of pairs of cells is captured by a predicate adj. In addition, there are two sensor readings: one for a patient and one for the oxygen tank that patient is dependent on. Because the sensor readings carry uncertainty, there are three possible locations for the patient and three possible locations for the oxygen. The query of interest asks whether it is certain that the patient has been separated from the oxygen.

The complete portion of the theory, C, consists of the definitions for adj, which can be found on page 10. The incomplete portion of the theory, I, is comprised of the constraints on the patient and the oxygen.

$$\forall xy.(patient(x) \wedge patient(y) \Rightarrow x = y)$$
$$patient(a) \vee patient(b) \vee patient(e)$$
$$\forall xy.(oxygen(x) \wedge oxygen(y) \Rightarrow x = y)$$
$$oxygen(l) \vee oxygen(o) \vee oxygen(p)$$

These sentences are incomplete because there are multiple models that satisfy them. Each model assigns the patient to one of three locations: a, b, or e, and it assigns the oxygen to one of three locations: l, o, or p. There are therefore nine different models of these sentences in FHL. Moreover, because neither the location of the patient nor the location of the oxygen is entailed (the same in all models), neither the predicate *patient* nor the predicate *oxygen* have complete definitions.

By the end of this chapter we will be able to answer the query of interest using Extensional Reasoning.

$$\forall xy.(patient(x) \wedge oxygen(y) \Rightarrow \neg adj(x, y)) \tag{4.1}$$

The utility of partitioning a theory into $C \cup I$ is that it supports two approaches to handling incomplete theories: theory resolution and theory completion.

The reason both these approaches are valuable is that Extensional Reasoning can be deployed in a variety of settings. The theorem proving setting assumes that neither the theory nor the query have ever been seen before or will ever be seen again. The database setting assumes that the complete portion of the theory, C, will change frequently but the incomplete portion of the theory and the query will remain the same. The theory resolution

approach is better suited in the theorem proving setting, whereas the theory completion approach is better suited for the database setting. This chapter examines theory resolution and theory completion, two approaches for addressing incomplete theories in the context of Extensional Reasoning.

4.1 Theory Resolution

Extensional Reasoning can be deployed in a theorem proving setting, where the assumption is that once given the theory and the query, the only measure that matters is how long it takes to determine whether the theory entails the query. This setting is in some sense the most difficult for reformulation techniques to be successful in because there is little opportunity to amortize the cost of reformulation. One of the well-known techniques within the theorem proving community for dealing with a partitioned theory is theory resolution.

Theory Resolution [Sti85] is an augmentation of the well-known resolution proof procedure [Rob65] for dealing with situations where part of the theory can be effectively represented and reasoned about with a specialized procedure. Theory resolution when applied to a theory $C \cup I$ allows us to use a database system to reason about C and resolution to reason about I.

Mathematically, theory resolution augments resolution by adding additional rules of inference that represent the piece of the theory hidden from resolution, in our case C. Because C is a complete theory, it can be conceptualized as a complete set of literals: a set that includes every ground atom built using a complete predicate or its negation. Because the part of the theory hidden from resolution is a set of unit clauses, the only resolutions that cannot be performed by resolution itself are certain unit resolutions. Of course, in actuality, C will be represented as a set of extensional tables and a set of view definitions, but for the purpose of developing inference rules for theory resolution, the true representation is unimportant.

Our results for theory resolution cover the case when the incomplete portion of the theory I is in the $\forall^*$ prefix class, i.e. when I is written in prenex normal form, the quantifiers are all $\forall$. This class is convenient because when it is converted to clausal form, no skolem symbols appear, and theory resolution takes on a particularly simple form. To illustrate, we first consider how resolution behaves when given $C \cup I$; we then introduce an inference rule that mimics that behavior when combined with resolution and applied to I alone.

Suppose p is a predicate with a complete definition, i.e. every ground p literal or its negation belongs to C. Now consider a clause generated by resolution from $C \cup I$.

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

This clause will resolve with every literal $\neg p(\bar{a})$ entailed by C where $p(\bar{a})$ unifies with $p(\bar{t})$. Each of the resolvents takes the following form.

$$\Phi\sigma, \text{ where } \sigma \text{ is the most general unifier of } p(\bar{a}) \text{ and } p(\bar{t}).$$

When moving to theory resolution, we must ensure that all these resolvents appear in the theory resolution closure of I. It turns out that ensuring the existence of all such resolvents is sufficient for ensuring the procedure is complete.

The new inference rule, which we call Complete Theory Resolution (CTR), when applied to a clause $\{p(\bar{t})\} \cup \Phi$ finds all those $\neg p(\bar{a})$ entailed by C and for each one produces the resolvent seen above. Likewise if the signs of the p literals are swapped.

Definition 5 (Complete Theory Resolution). *Complete Theory Resolution (CTR) is the following rule of inference, taking the complete theory C as a parameter. Applying CTR to closure on some set of clauses S is denoted $\text{CTR}(C, S)$. When in addition to closing under CTR, the closure includes all the usual resolution inference rules, we denote the result with $\text{CTR+RES}(C, S)$. In the inference rule below, p is a predicate with a complete definition in C, and $\mp p(\bar{t})$ has the opposite sign of $\pm p(\bar{t})$.*

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

Clearly CTR is a sound rule of inference because it amounts to n applications of resolution; when coupled with resolution it turns out to also be complete when the incomplete portion of the theory is in $\forall^*$.

Theorem 7 (CTR+RES Soundness and Completeness for $\forall^*$). *Suppose $C \cup I$ is a*

finite set of FHL sentences, where C is a satisfiable, complete theory, and I is in $\forall^*$. $C \cup I$ is unsatisfiable if and only if CTR+RES(C, I) *contains the empty clause.*

The proof of this theorem can be found in Appendix A.3.1.

The proof of this theorem is actually quite straightforward because as it turns out, when I is in $\forall^*$, theory resolution does more than simply use a database to reason about C. Theory resolution implicitly represents C as a set of ground literals, which happens to obviate the need to include in I an axiom that is a perennial problem for resolution-style theorem provers: the domain closure axiom. Without this axiom, there is no need for paramodulation, which makes the proof much simpler.

When using a first-order proof procedure for answering entailment queries in Finite Herbrand Logic, we usually need to include a domain closure axiom, e.g. $\forall x.(x = a \vee x = b \vee x = c)$, and unique names axioms. These sentences are sufficient to axiomatize the semantics of FHL in first-order logic, allowing us to use first-order proof techniques to answer FHL queries. These axioms, however, require the proof system to reason about equality, which in the case of resolution-style procedures is often done using an inference rule such as paramodulation [CL73]. When combined with a DCA, paramodulation is well-known to cause a significant blow-up in the size of the resolution search space; consequently, Reiter [Rei80] investigated sufficient conditions under which the DCA and paramodulation were unnecessary for preserving the completeness of resolution. His result applies to sentence sets in the $\forall^*$ prefix class: if Δ is in $\forall^*$ then resolution with paramodulation applied to $\Delta \cup \{DCA\} \cup UNA$ will produce the empty clause if and only if resolution without paramodulation applied to $\Delta \cup UNA$ produces the empty clause.

Using a decomposed theory such as we have, as long as $C \cup I$ is in $\forall^*$, we can throw away the DCA and paramodulation. The conditions on the completeness theorem above require I to be in $\forall^*$, and because mathematically C is represented as a set of literals by theory resolution, we see that $C \cup I$ is in $\forall^*$. Thus, Reiter's theorem applies: under the conditions of the theorem above, theory resolution is complete even without including a DCA.

This form of theory resolution is such a special case that it might be more appropriate to use the term procedural attachment [Wey80, Mye90a, Mye90b, Sik96]. It should be stressed, however, that regardless of whether we call the mechanism a procedural attachment or theory resolution, the novelty presented here is the application of that mechanism. Here we have an algorithm for automatically choosing the portion of the theory to hide from resolution and inference rules for handling the entire class of theories that algorithm chooses:

complete theories. This departs from traditional applications where the portion of the theory to hide is selected by a human being.

Clearly, to have a full solution to the theory resolution approach we need to deal with the case when the incomplete sentences do not belong to $\forall^*$. One option would be to turn any sentence with an existential quantifier into a sentence without such quantifiers by (partially) converting the sentence into propositional logic. Such propositionalization is always possible in this logic, but because it is often too expensive, in the remainder of this thesis we focus on techniques that do not rely upon it. In the case of theory resolution, we have only speculated about the inference rules necessary for handling sentences that do not belong to $\forall^*$. The difficulty is that existential quantifiers necessitate the use of skolem symbols, which must be dealt with by resolution and, more importantly, by the database; moreover, proofs of completeness are complicated by the fact that paramodulation is necessary. In this thesis 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.

$$p(x) \Leftrightarrow (x = a \vee x = b)$$
$$q(x) \Leftrightarrow (x = c)$$
$$\exists x.(\neg p(x) \wedge \neg q(x))$$

The first two sentences belong to C because they comprise complete definitions for p and q, and the last sentence belongs to I. These sentences are inconsistent because the first two say that every element in the universe (which consists of a, b, and c) 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 those shown here.

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

Concentrating on the $\{\neg p(k)\}$ clause, we know that $p(a)$ and $p(b)$ are both entailed by C. The only way $\neg p(k)$ is consistent with the theory is if $k \neq a$ and $k \neq b$. These two sentences are entailed by the premises. Suppose we include both consequences in the closure.

4. $\{k \neq a\}$

5. $\{k \neq b\}$

Now concentrating on the $\{\neg q(k)\}$ clause, given that $q(c)$ is entailed by C, the only way $\neg q(k)$ is consistent is if $k \neq c$. This produces the following consequence.

6. $\{k \neq c\}$

Together these three consequences are inconsistent with the DCA (clause 1). Because none of these four clauses is hidden from resolution, by the completeness of resolution, we know the empty clause will be produced.

The general idea for the case of unary predicates seems to be as follows. What is unclear is whether even for the case of unary predicates this inference rule if added to CTR+RES would form a complete inference procedure when I is in $\exists^*\forall^*$.

$$\frac{\begin{array}{l}\{\pm p(k)\} \cup \Phi, \text{ where } k \text{ is a skolem constant} \\ \text{Let } \{a_1, \ldots, a_n\} \text{ be the set of all object constants } a_i \text{ such that } C \models \mp p(a_i)\end{array}}{\begin{array}{c}\{k \neq a_1\} \cup \Phi \\ \vdots \\ \{k \neq a_n\} \cup \Phi\end{array}}$$

Further work needs to be done to expand this idea to handle the non-unary case and then determine whether such an inference rule would be complete. The next step would be to build an inference rule for handling skolem functions in addition to skolem constants. These ideas are reminiscent of the work on E-resolution [Mor69], RUE (resolution with unification and equality) [Dig79], and examples from the original theory resolution paper [Sti85].

One of the problems with the theory resolution approach as described here is that when the complete portion of the theory C is large, CTR can produce a large number of results when applied to nonground literals. One option for addressing this issue is to employ ordered theory resolution, delaying the evaluation of literals with definitions in C until the end, except for ground literals which are eagerly evaluated. Then, once there is a clause that only mentions predicates with definitions in C,

$$\{\pm p_1(\overline{t_1}), \ldots, \pm p_n(\overline{t_n})\},$$

the system could pose a conjunctive query to the database to ascertain whether this clause is by itself inconsistent with C:

$$\exists \overline{x}.(\mp p_1(\overline{t_1}) \wedge \cdots \wedge \mp p_n(\overline{t_n}))$$

If an inconsistency exists, the proof has been completed; otherwise, the clause in question can be removed because it is weaker than C. (To see this, note that the clause is equivalent to all of its groundings, each of which must be consistent with C. Consider any one of those groundings. Since the grounding is not inconsistent with C, it must not be the case that every literal's negation is entailed by C. This means that there is some literal in the clause entailed by C; that literal therefore subsumes this particular grounding of the clause. This argument applies to all groundings, which means they are all subsumed by C. Hence, the original clause can be removed because it is weaker than C.) Similar ideas are at the root of constraint resolution [Bur90] and constraint logic programming [JL87].

For the theorem proving setting, where all that matters is answering the given entailment query in as little time as possible, theory resolution has the nice property that the reformulation cost is no more than if the entire theory were complete, which is low-order polynomial. After reformulation the well-known tools and techniques from automated theorem proving with theory resolution can be used to manipulate the incomplete portion of the theory I while allowing the database to reason about the complete theory C. In this setting, resolution is used to generate a series of database queries, each of which is sufficient to conclude that the entailment query holds.

4.2 Theory Completion

In addition to the theorem proving setting investigated in the last section, Extensional Reasoning can also be deployed in what we are calling the database setting: the complete portion of the theory C changes frequently but the incomplete portion I and the query are both static. In this setting, the theory resolution approach described in the last section would require resolution to manipulate the same incomplete axioms I, in much the same way, for each C. In this section, we examine a different approach that compiles the query and the incomplete portion of the theory into a database query about a complete theory. This means that the manipulation of the incomplete theory is done once, and each time the complete theory changes, one can use an off-the-shelf database system to answer the query.

Another way to think about it is that the techniques introduced in this section handle incomplete theories by transforming an entailment query about an incomplete theory into an entailment about a complete theory and then using the techniques described in the previous chapter to answer questions about the completed theory. The fact that it is easy to change the complete portion C is a by-product of the approach to theory completion studied here.

Theory completion (or perhaps more accurately theory minimization) has been studied to a great extent in the Nonmonotonic Reasoning literature, e.g. the Closed-World Assumption [Rei78], Predicate Completion [Llo84], Circumscription [McC88]. Unlike Extensional Reasoning, these settings assume it is acceptable (and in fact desirable) for the reformulation to change the logical consequences of the theory. In contrast, the theory completion introduced here must be consequence-preserving in the sense described below, making the Nonmonotonic work insufficient for our purpose.

Because theory-completion in the context of Extensional Reasoning is performed solely for the sake of efficiency, there are constraints on how the theory can be completed. To preserve soundness and completeness, a theory must be completed so that an 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. Given a theory Δ and a conclusion ϕ, we need to construct a complete theory Δ' and a query ϕ' such that

$$\Delta \models \phi$$
$$\text{iff}$$
$$\Delta' \models \phi'.$$

Because the theory Δ has been decomposed into $C \cup I$, and C is already complete, we need to construct a complete theory I' and a query ϕ' such that

$$C \cup I \models \phi$$
$$\text{iff}$$
$$C \cup I' \models \phi'.$$

This approach therefore requires developing two transformation algorithms: one for the query and one for the incomplete portion of the theory.

Our approach to operationalizing this form of theory completion relies on a concept

that is central to logic itself: satisfiability. Satisfiability has long been used in refutational theorem proving to answer logical entailment queries: $\Delta \models \phi$ if and only if $\Delta \cup \{\neg\phi\}$ is unsatisfiable. Satisfiability is usually leveraged solely at the meta-level by people who are trying to justify the soundness and completeness of proof systems. In contrast, the techniques presented here use satisfiability within the logical theory, resulting in proof systems reasoning about satisfiability directly. The central activities required for this approach are the construction and manipulation of definitions for new predicates, each of which represents the satisfiability of a particular sentence in combination with a background theory. $poss_\phi$, a predicate representing the satisfiability of sentence ϕ, is true if and only if ϕ is consistent with the background theory Δ. The intuition is that $poss_\phi$ means that ϕ is possibly true in Δ, i.e. that there is some model of Δ that satisfies ϕ.[1]

These *poss* predicates are important for theory completion because the completed theory, I' will consist of complete definitions for various *poss* predicates, and the new query, ϕ' will be expressed in terms of various *poss* predicates. Below, we go through some examples to solidify our understanding of *poss*, and then we return to its utility for theory completion.

Consider a theory that consists of one sentence $\exists x.p(x)$. Certainly the sentence $p(a)$ is consistent with that sentence, and so $poss_{p(a)}$ is true. The sentence in the subscript of a *poss* predicate can be arbitrarily complex. Using the same background theory, we know that $\exists x.\neg p(x)$ is consistent with the theory and consequently that $poss_{\exists x.\neg p(x)}$ is true.

In addition to sentences in the subscript of a *poss* predicate where every variable has been captured by a quantifier, we are also interested in sentences in the subscript of *poss* predicates that have free variables. The intuition is that the *poss* predicate represents all the instances of the sentence in the subscript that are consistent with the background theory. If the sentence in the subscript has n free variables, then the *poss* predicate has arity n.

Again consider the theory $\exists x.p(x)$. Suppose the universe consists of three object constants, a, b, and c. The *poss* predicate where the sentence in the subscript is $p(x)$ takes a single argument, and it is true of all object constants d such that $p(d)$ is consistent with the theory. Since $p(a)$ is consistent with the theory, $p(b)$ is consistent with the theory, and $p(c)$ is consistent with the theory, $poss_{p(x)}$ must be true of a, b, and c.

$$poss_{p(x)}(x) \Leftrightarrow (x = a \vee x = b \vee x = c)$$

[1] Conceptually $poss_\phi$ can be interpreted as a classical encoding of the modal statement $\Diamond\phi$, which means that ϕ is possibly true; however, the *poss* predicates studied here correspond to a very special case of modal logic and deserve their own treatment. For a comparison of modal logic and *poss*, see Section 5.4.

Notice that $poss_{p(x)}$ is simply the name of a predicate, which could just as easily have been q or r. What is important is that this predicate represents all the instances of $p(x)$ that are consistent with the theory. It is only to avoid bookkeeping that we use predicates labeled with logical sentences.

To clarify one subtlety in this definition, consider again the theory $\exists x.p(x)$ with universe $\{a, b, c\}$ and this time the predicate $poss_{\neg p(x)}$. The *poss* predicate must be true of all the instances of $\neg p(x)$ that are consistent with the theory. Just like the last example, $\neg p(a)$ is consistent with $\exists x.p(x)$, as is $\neg p(b)$, and $\neg p(c)$. Thus, $poss_{\neg p(x)}$ is true of a, b, and c.

$$poss_{\neg p(x)}(x) \Leftrightarrow (x = a \vee x = b \vee x = c)$$

This example is important because it illustrates that the consistency test is applied to each instance individually and not to the conjunction of the instances. That is, even though the conjunction $\neg p(a) \wedge \neg p(b) \wedge \neg p(c)$ is inconsistent with the theory, the correct definition of $poss_{\neg p(x)}$ ensures that $poss_{\neg p(x)}(a) \wedge poss_{\neg p(x)}(b) \wedge poss_{\neg p(x)}(c)$ is true.

Formally, the algorithm that constructs *poss* definitions, which we call *poss*-DEFINITION, must abide by the following constraints.

Definition 6 (*poss*-Definition). *poss*-DEFINITION$(\phi(\overline{x}), \Delta)$ *is an operation on a sentence* $\phi(\overline{x})$ *with free variables* $\overline{x}$ *and a set of sentences* Δ *that produces a typed, complete definition for* $poss_{\phi(\overline{x})}(\overline{x})$ *such that* $poss_{\phi(\overline{x})}(\overline{a})$ *is true if and only* $\phi(\overline{a})$ *is consistent with* Δ. $poss_{\phi(\overline{x})}(\overline{x})$ *is typed so that it only applies to object constants in* Δ.[2]

Given an atom $poss_{\phi(x_1,\ldots,x_n)}(t_1, \ldots, t_n)$, there is a question about how the arguments t_1, $\ldots$, t_n correspond to the variables in the subscript: x_1, $\ldots$, x_n. While we could choose various options, in this thesis argument t_i corresponds to the i^{th} free variable in $\phi(x_1, \ldots, x_n)$, ordered from left to right.

In the context of the hospital sensor net example, we can illustrate the utility of these *poss* predicates for the purpose of theory completion. Suppose we wanted to know whether the following sentence were consistent with the hospital theory illustrated in Chapter 2. The sentence states that there is some location for the patient and some location for the

[2]It is important that every *poss* predicate be typed so that it is only true or false of the object constants contained in Δ. As we will see later, sometimes the completed theory will use object constants that do not appear in Δ. To capture the semantics of *poss* correctly, *poss*-DEFINITION must ensure that *poss* is never true nor false of those new object constants.

oxygen where the patient is adjacent to the oxygen.

$$\exists xy.(patient(x) \wedge oxygen(y) \wedge adj(x,y)) \tag{4.2}$$

Using the *poss* approach, we would first construct a complete definition for the 0-ary predicate

$$poss_{\exists xy.(patient(x) \wedge oxygen(y) \wedge adj(x,y))},$$

then convert that definition into a database system, and finally determine satisfiability by answering the database query $poss_{\exists xy.(patient(x) \wedge oxygen(y) \wedge adj(x,y))}$.

This particular satisfiability query is relevant to the hospital entailment query mentioned earlier, "Has the patient and the oxygen most certainly been separated?" because one answer is the negation of the other. The satisfiability query holds exactly when the entailment query fails to hold. If sentence 4.2 is consistent with the hospital theory then the patient has not necessarily been separated from the oxygen (since the satisfying model provides a counterexample), but if sentence 4.2 is inconsistent with the hospital theory, then the patient must have been separated from the oxygen (since there is no counterexample).

The reason this works is that sentence 4.2 is the negation of our entailment query.

$\Delta \models \forall xy.(patient(x) \wedge oxygen(y) \Rightarrow \neg adj(x,y))$
if and only if $\Delta \cup \{\neg \forall xy.(patient(x) \wedge oxygen(y) \Rightarrow \neg adj(x,y))\}$ is not satisfiable
if and only if $poss_{\neg \forall xy.(patient(x) \wedge oxygen(y) \Rightarrow \neg adj(x,y))}$ is not true
if and only if $poss_{\exists xy.(patient(x) \wedge oxygen(y) \wedge adj(x,y))}$ is not true
if and only if $\neg poss_{\exists xy.(patient(x) \wedge oxygen(y) \wedge adj(x,y))}$ is true

More generally, $\Delta \models \phi$ if and only if $\neg poss_{\neg \phi}$ is true. This *poss* approach is exactly the same as the refutational theorem proving approach. It shows that the entailment holds by showing that the negation of the query is inconsistent with the theory.

In our pursuit of theory completion, where when given an entailment query $C \cup I \models \phi$ we must construct I' and ϕ' so that $C \cup I \models \phi$ if and only if $C \cup I' \models \phi'$, *poss* predicates give us a straightforward method of constructing ϕ'. ϕ' is simply the sentence $\neg poss_{\neg \phi}$. The remainder of this chapter looks at methods for constructing I', i.e. methods for constructing a complete definition for $poss_{\neg \phi}$.

4.2.1 The Properties of *poss*

The last section introduced *poss* in order to show how it can be used in theory completion to construct a new query, ϕ', which is written in terms of *poss* predicates. This section begins the introduction of algorithms for automatically generating definitions that encode the semantics of *poss* predicates. Constructing those definitions is the last step in theory completion because it is these definitions that comprise the new theory I'. Done correctly, this construction guarantees that theory completion is performed as required: $C \cup I \models \phi$ if and only if $C \cup I' \models \phi'$.

As a rule of thumb, the definition for $poss_\phi$ is easier to construct the shorter the sentence ϕ. Fortunately, $poss_\phi$ can sometimes be broken up into predicates $poss_{\phi_1}, \ldots, poss_{\phi_n}$ so that ϕ_i is shorter than ϕ for every i. Instead of having to construct a single definition for a *poss* predicate with a long sentence in the subscript, we can construct *poss* definitions for several *poss* predicates with short sentences in their subscripts.

For example, in this section, we will see how to rewrite the hospital sensor net entailment query

$$\neg poss_{\exists xy.(patient(x) \wedge oxygen(y) \wedge adj(x,y))}$$

into the entailment query shown in Chapter 2:

$$\forall xy.(poss_{patient(x)}(x) \wedge poss_{oxygen(y)}(y) \Rightarrow \neg adj(x,y)).$$

Most of the properties of *poss* required to perform this rewriting state the conditions under which *poss* can be distributed over various logical connectives. For example, it turns out that *poss* can be distributed across disjunction: $poss_{\phi \vee \psi}$ is equivalent to $poss_\phi \vee poss_\psi$. That is, if we had a complete definition for $poss_{\phi \vee \psi}$ and definitions for $poss_\phi$ and $poss_\psi$, $poss_{\phi \vee \psi}$ would be true exactly when $poss_\phi \vee poss_\psi$ were true. This section introduces various properties of *poss*, illustrating their utility by showing how they can be used to perform the hospital query rewriting. Unless stated otherwise, all proofs for the theorems in this section can be found in Appendix A.3.2.

We start with a theorem that says *poss* distributes over $\vee$. The statement of the theorem applies to sentences with free variables because as will be illustrated shortly, sometimes the sentence in the subscript of a *poss* predicate contains free variables.

Theorem 8 (*poss* distributes over $\vee$). *In FHL, $poss_{\phi(\overline{x},\overline{y})\vee\psi(\overline{x},\overline{z})}(\overline{x},\overline{y},\overline{z})$ is logically equivalent to $poss_{\phi(\overline{x},\overline{y})}(\overline{x},\overline{y}) \vee poss_{\psi(\overline{x},\overline{z})}(\overline{x},\overline{z})$.*

Because *poss* distributes across disjunction, it is not surprising that it distributes over existential quantifiers because in FHL the two are interconvertible.

Theorem 9 (*poss* distributes over $\exists$). *In FHL, $poss_{\exists x.\phi(x,\overline{y})}(\overline{y})$ is logically equivalent to $\exists x.poss_{\phi(x,\overline{y})}(x,\overline{y})$.*

It is noteworthy that the arity of the *poss* predicate increases when it is distributed across the existential quantifier. Or perhaps more accurately, distributing *poss* amounts to replacing one *poss* predicate with a logical sentence that is equivalent to that predicate and expressed using other *poss* predicates.

In the hospital example, we can distribute *poss* across the existential quantifier.

$$\begin{aligned}&\neg poss_{\exists xy.(patient(x)\wedge oxygen(y)\wedge adj(x,y))}\\ &\Leftrightarrow \neg\exists xy.poss_{patient(x)\wedge oxygen(y)\wedge adj(x,y)}(x,y)\end{aligned}$$

It turns out that *poss* does not always distribute over universal quantifiers, conjunction, or negation. If *poss* did distribute over negation, it would also distribute over universal quantifiers and conjunction; hence, the counterexample of negation is the most important. Consider the propositional sentence p in a theory where both p and $\neg p$ are consistent. In this theory both $poss_{\neg p}$ and $poss_p$ are true. If *poss* could be distributed over $\neg$, then $poss_{\neg p}$ implies $\neg poss_p$, which contradicts the fact that $poss_p$ is true.

The counterexample relies on the fact that the theory is incomplete with respect to p; it turns out that completeness is both a necessary and sufficient condition for distributing *poss* across negation.

Theorem 10 (*poss* distributes over $\neg$). *In FHL, $poss_{\neg\phi(\overline{x})}(\overline{x})$ is equivalent to $\neg poss_{\phi(\overline{x})}(\overline{x})$ exactly when the background theory Δ is satisfiable and complete with respect to $\phi(\overline{x})$.*

This theorem does not help us simplify the hospital *poss* query above because the sentence in the subscript of that query is a conjunction of literals; however, it turns out that *poss* can be distributed across that conjunction because the conjuncts have a special relationship: they are independent.

$$\neg\exists xy.(poss_{patient(x)}(x) \wedge poss_{oxygen(y)}(y) \wedge adj(x,y))$$

Intuitively, the reason *poss* distributes in this case is that the possible locations for the patient are entirely independent of the possible locations for the oxygen, and those two are independent of adjacency. This intuition is materialized syntactically by the fact that the sentences constraining *patient* never mention *oxygen*, and the sentences constraining *oxygen* never mention *patient*; likewise for *adj* and each of *patient* and *oxygen*. Thus if $patient(t)$ were satisfied by some model, $oxygen(u)$ were satisfied by some model, and $adj(r, s)$ were satisfied by some model, there must be yet another model that satisfies $patient(t) \wedge oxygen(u) \wedge adj(r, s)$. We say that such sentences are *independent*.

Definition 7 (Sentence Independence). *Let Δ be a sentence set in FHL and $\phi(\overline{x}, \overline{z})$ and $\psi(\overline{y}, \overline{z})$ be sentences whose vocabularies are subsets of the vocabulary of Δ that share the variables $\overline{z}$. $\phi(\overline{x}, \overline{z})$ and $\psi(\overline{y}, \overline{z})$ are independent with respect to Δ if and only if for every tuple of object constants $\overline{a}$, $\overline{b}$, and $\overline{c}$, if $\phi(\overline{a}, \overline{c})$ is consistent with Δ and $\psi(\overline{b}, \overline{c})$ is consistent with Δ then $\phi(\overline{a}, \overline{c}) \wedge \psi(\overline{b}, \overline{c})$ is consistent with Δ.*

Whenever two sentences are independent, *poss* distributes over their conjunction.

Theorem 11 (*poss* distributes over $\wedge$). *In FHL, $\phi(\overline{x}, \overline{z})$ and $\psi(\overline{y}, \overline{z})$ are independent with respect to Δ exactly when $poss_{\phi(\overline{x},\overline{z})\wedge\psi(\overline{y},\overline{z})}(\overline{x}, \overline{y}, \overline{z})$ is equivalent to $poss_{\phi(\overline{x},\overline{z})}(\overline{x}, \overline{z}) \wedge poss_{\psi(\overline{y},\overline{z})}(\overline{y}, \overline{z})$.*

In light of the last theorem, independence is an important property. One sufficient condition for determining whether $p(\overline{x}, \overline{z})$ and $q(\overline{y}, \overline{z})$ are independent is syntactic: if the constraints on p and q never (transitively) mention the other predicate (where predicates with complete definitions can be ignored) then the two predicates are independent. Formally, this condition can be defined by examining the Semantic Dependency Graph of a sentence set.

Definition 8 (Semantic Dependency Graph). *A Semantic Dependency Graph for a set of sentences Δ is a graph $\langle V, E \rangle$.*

- *$u \in V$ if and only if u is a predicate in Δ without a complete definition*
- *The undirected edge (u, v) is in E if and only if there is some sentence in Δ that uses both the predicates u and v.*

If in this graph there is a path from predicate p to predicate q then p and q are deemed dependent. If there is no such path then p and q are independent. It is noteworthy that none of the predicates that have complete definitions occur in this graph. Thus, if all we know is that there is some sentence that mentions p and a complete predicate r and another sentence that mentions q and r, we cannot conclude that p and q are dependent. For p and q to be dependent, they must either both occur in the same sentence, or they must occur in sentences that transitively mention the same incomplete predicate. These graphs differ from the usual dependency graphs used in the deductive database literature [Ull89] because the edges are undirected and the predicates with complete definitions are not included. The "semantic" part of the name refers to the fact that it more closely approximates which predicates are actually dependent on one another by leaving out those predicates with complete definitions.

Theorem 12 (Syntactic Independence). *Let Δ be a satisfiable set of FHL sentences, and let G be its Semantic Dependency Graph. Suppose that in G each of the predicates used in $\phi(\overline{x},\overline{z})$ is in a different connected component than every predicate in $\psi(\overline{y},\overline{z})$. Then $\phi(\overline{x},\overline{z})$ and $\psi(\overline{y},\overline{z})$ are independent with respect to Δ.*

Connected-components can be computed in time linear in the size of the Semantic Dependency Graph, and this graph is linear in the size of Δ.[3] Given the components, checking syntactic independence of two sentences requires time linear in the size of the sentences.

Going back to the hospital example, syntactic independence guarantees that *patient*, *oxygen*, and *adj* are all independent. This pushes the simplification one step further than before.

$$\neg\exists xy.poss_{patient(x)\wedge oxygen(y)\wedge adj(x,y)}(x,y)$$
$$\Leftrightarrow \neg\exists xy.(poss_{patient(x)}(x) \wedge poss_{oxygen(y)}(y) \wedge poss_{adj(x,y)}(x,y))$$

The final step of the rewriting for this query requires one more theorem. *adj* is a predicate with a complete definition. Just as independence affects the properties of *poss*, completeness affects those properties as well, and to an even greater extent. (1) A sentence that is complete is independent of every other sentence. (2) $poss_\phi$ is logically equivalent to ϕ when ϕ is complete.

[3]Technically, a Semantic Dependency Graph might require a quadratic number of edges but there is a linear graph sufficient for the purpose of computing connected-components.

Theorem 13 (Completeness Guarantees Independence). *Let Δ be a satisfiable set of FHL sentences complete with respect to $\phi(\overline{x})$, i.e. for every tuple of object constants $\overline{a}$, $\Delta \models \phi(\overline{a})$ or $\Delta \models \neg\phi(\overline{a})$. Then $\phi(\overline{x})$ is independent of every other sentence ψ with respect to Δ.*

Theorem 14 (Completeness and *poss* Redundancy). *Let Δ be a satisfiable set of FHL sentences complete with respect to $\phi(\overline{x})$, i.e. for every tuple of object constants $\overline{a}$, $\Delta \models \phi(\overline{a})$ or $\Delta \models \neg\phi(\overline{a})$. Then*

$$\Delta \cup \{poss\text{-}\textsc{Definition}(\phi(\overline{x}), \Delta)\} \models poss_{\phi(\overline{x})}(\overline{x}) \Leftrightarrow \phi(\overline{x})^4$$

In the context of the hospital sensor net, the theorems above guarantee that (1) *adj* is independent of every other sentence and (2) $poss_{adj(x,y)}(x, y)$ is logically equivalent to $adj(x, y)$. Using all the theorems in this section, we see the following sequence of *poss* simplifications of the hospital query.

$\Delta \models \forall xy.(patient(x) \wedge oxygen(y) \Rightarrow \neg adj(x, y))$
$\Leftrightarrow \neg poss_{\neg\forall xy.(patient(x)\wedge oxygen(y)\Rightarrow\neg adj(x,y))}$ (*poss* Semantics)
$\Leftrightarrow \neg poss_{\exists xy.(patient(x)\wedge oxygen(y)\wedge adj(x,y))}$ (Log. Equiv.)
$\Leftrightarrow \neg\exists xy.poss_{patient(x)\wedge oxygen(y)\wedge adj(x,y)}(x, y)$ (Distr. over $\exists$)
$\Leftrightarrow \neg\exists xy.(poss_{patient(x)}(x) \wedge poss_{oxygen(y)}(y) \wedge poss_{adj(x,y)}(x, y))$ (Distr. over $\wedge$)
$\Leftrightarrow \neg\exists xy.(poss_{patient(x)}(x) \wedge poss_{oxygen(y)}(y) \wedge adj(x, y))$ (*poss* Redundancy)
$\Leftrightarrow \forall xy.(poss_{patient(x)}(x) \wedge poss_{oxygen(y)}(y) \Rightarrow \neg adj(x, y))$ (Log. Equiv.)

The original query and the final query look remarkably similar, almost as if *poss* were distributed across the universal quantifier. It turns out that just like the other connectives, there are precise model-theoretic conditions under which *poss* distributes over $\forall$.

Definition 9 (Nondisjunctive Existential Sentence). *The sentence $\exists x.\phi(x, \overline{y})$ is said to be a nondisjunctive existential sentence with respect to Δ if and only if for every tuple of object constants $\overline{b}$*

$$\Delta \models \exists x.\phi(x, \overline{b}) \text{ implies there is some } a \text{ such that } \Delta \models \phi(a, \overline{b})$$

[4] Assuming *poss*-DEFINITION is a conservative extension of Δ, i.e. every model that satisfies Δ is the reduct of some model that satisfies $\Delta \cup \{poss\text{-}\textsc{Definition}(\phi(\overline{x}), \Delta)\}$.

Logical Connective	Conditions on Sentence(s)
$\exists$	none
$\vee$	none
$\wedge$	Independent
$\neg$	Complete
$\forall$	Nondisjunctive Existential

Table 4.1: The conditions under which *poss* distributes over various logical connectives

Theorem 15 (*poss* distributes over $\forall$). *In FHL, $poss_{\forall x.\phi(x,\overline{y})}(\overline{y})$ is equivalent to $\forall x.poss_{\phi(x,\overline{y})}(x,\overline{y})$, where Δ is the background theory, exactly when the sentence $\exists x.\neg\phi(x,\overline{y})$ is a nondisjunctive existential sentence with respect to Δ.*

If we put all the theorems from this section together, which are summarized in Table 4.1, we can produce SIMPLIFY-POSS, shown as Algorithm 5. SIMPLIFY-POSS makes the assumption that it is always desirable to distribute *poss* if it is possible to do so. Since every sentence in the subscript of a *poss* predicate is finite, the theorems above can only be applied a finite number of times, and therefore the algorithm halts in all cases. Moreover, if we assume that, after some preprocessing, checking whether the theory is complete for a given sentence and checking whether two sentences are independent both cost time linear in the sentence(s), then SIMPLIFY-POSS runs in time linear in the size of the sentence. The theorems above guarantee that SIMPLIFY-POSS preserves logical equivalence, according to the semantics of *poss*.

What we have seen in this section is that any logical entailment query can be written as a *poss* query, and a *poss* query can sometimes be simplified by distributing *poss* over logical connectives, thereby reducing the complexity of constructing the necessary *poss* definitions. The conditions for distributing *poss* have been precisely characterized, and sufficient tests for all but one of those conditions, nondisjunctive existentials, have been developed. The simplification algorithm presented here requires time linear in the size of the query after two preprocessing steps: (1) a linear time algorithm in the size of the theory to compute independence and (2) a low-order polynomial time algorithm in the size of the theory to compute completeness.

Algorithm 5 SIMPLIFY-POSS($poss_\phi$)

Assumes: ϕ is written using $\forall$, $\exists$, $\vee$, $\wedge$, and $\neg$, where $\neg$ is only applied to atomic sentences.

1: **if** $\phi = \exists x.\psi(x)$ **then**
2: **return** $\exists x.$SIMPLIFY-POSS($poss_{\psi(x)}$)
3: **else if** $\phi = \psi_1 \vee \psi_2$ **then**
4: **return** SIMPLIFY-POSS($poss_{\psi_1}$) $\vee$ SIMPLIFY-POSS($poss_{\psi_2}$)
5: **else if** $\phi = \psi_1 \wedge \psi_2$ and INDEPENDENTP(ψ_1,ψ_2) **then**
6: **return** SIMPLIFY-POSS($poss_{\psi_1}$) $\wedge$ SIMPLIFY-POSS($poss_{\psi_2}$))
7: **else if** COMPLETEP(ϕ) **then**
8: **return** ϕ
9: **else**
10: **return** $poss_\phi$
11: **end if**
12: {Since we do not have a test for Nondisjunctive Existential sentences, there is no universal quantifier case that distributes *poss* over $\forall$.}

4.2.2 Quantifier-free Sentences

Applying SIMPLIFY-POSS to $\neg poss_{\neg\phi}$ results in a new query that is written in terms of *poss* predicates and predicates from the original language. In order to answer this query, we need to construct definitions for both types of predicates. Because the only non-*poss* predicates that occur in the new query have complete definitions in the original theory $C \cup I$, C contains the complete definitions for all those predicates. Thus, all that remains to perform the type of theory completion necessary for Extensional Reasoning is to construct complete definitions for the *poss* predicates.

It is important to keep in mind the purpose for which we are investigating *poss* definition construction and theory completion. Recall that theory completion was introduced to address the deployment of Extensional Reasoning in the database setting: where the incomplete portion of the theory I and the query are static, but the complete portion C changes frequently. The algorithms for constructing *poss* definitions therefore cannot rely on the contents of C; however, they can rely on the fact that the predicates with definitions in C will always have definitions. Thus, the algorithms considered in this thesis for *poss* definition construction will only manipulate I, ignoring the contents of C entirely.[5]

In the database setting, because the only substantial reformulation is performed once

[5]Some algorithms for computing *poss* definitions are independent of C but are dependent on the set of object constants. If such an algorithm were used to compute *poss* definitions, and then the definition of C changed in such a way that new object constants were introduced, the *poss* definitions might need to be recomputed.

and the result is used a large number of times, it is reasonable to amortize the cost of reformulation over all the instances of C. Especially in those settings where C is significantly larger than I, reformulation algorithms that might normally be considered expensive can be run both because of the amortization and because they are relatively inexpensive when compared to C.

The approach taken in this section and the next has a proof-theoretic flavor. The predicate $poss_\phi$ is true exactly when there is some model that satisfies the background theory Δ as well as ϕ. Equivalently, $poss_\phi$ is true exactly when it is not the case that all the models that satisfy Δ also satisfy $\neg\phi$, i.e. when $\Delta \models \neg\phi$. The proof theoretic approach to *poss* definition construction produces a definition for $poss_\phi$ that when evaluated can be interpreted as a search for a proof of $\neg\phi$. If the search fails then $poss_\phi$ is true; otherwise, $poss_\phi$ is false.

Given a set of sentences $C \cup I$, suppose we want to construct a definition for the predicate $poss_\phi$, where ϕ is quantifier-free. It is well-known that any quantifier-free sentence can be written in disjunctive normal form. Without loss of generality, we can assume that we want to construct the definition for the following *poss* predicate.

$$poss\left(\begin{array}{c} l_{11}(\overline{x_{11}}) \wedge \cdots \wedge l_{1n_1}(\overline{x_{1n_1}}) \\ \vee \cdots \vee \\ l_{m1}(\overline{x_{m1}}) \wedge \cdots \wedge l_{mn_m}(\overline{x_{mn_m}}) \end{array}\right)(\overline{x_{11}}, \ldots, \overline{x_{mn_m}})$$

Because *poss* distributes over disjunction, as illustrated in the last section, the above *poss* predicate is logically equivalent to a disjunction of *poss* predicates, each of which has a conjunction of literals in its subscript. Thus, if we can construct a definition for an arbitrary *poss* predicate where the sentence in the subscript is a conjunction of literals, then we can construct a definition for a *poss* predicate with an arbitrary quantifier-free sentence in the subscript. The *poss* predicates of interest are therefore of the following form.

$$poss_{l_1(\overline{x_1}) \wedge \cdots \wedge l_n(\overline{x_n})}(\overline{x_1}, \ldots, \overline{x_n})$$

The first algorithm is based on a rewriting procedure that transforms a sentence that mentions predicates both with and without complete definitions into a sentence where all the predicates mentioned have complete definitions. The idea is that without needing to know the particular definitions contained in C, we can compute an expression that represents

the same information contained in the original sentence but using only the predicates with complete definitions. Such an expression can then be evaluated using the complete theory C.

Formally, we want an algorithm that given I, a sentence ϕ, and a set of predicates P constructs a sentence ψ such that for every set of complete definitions C for the predicates P

$$C \cup I \models \phi \text{ if and only if } C \models \psi.$$

The algorithm we use that meets this definition, called proof-tree completion in [McI98], is a form of abduction. It is a generalization of the model-elimination proof procedure, which we will illustrate by example.

Consider the following propositional rules, which together comprise the incomplete portion I of the theory of interest.

$$r \Leftarrow q \wedge p$$
$$p \Leftarrow s$$
$$p \Leftarrow t$$

The complete portion of the theory contains definitions for predicates q, s, and t. Suppose we want to rewrite r in terms of the complete theory.

Proof-tree completion works virtually the same as model elimination except when a literal built from a complete predicate (a predicate with a complete definition in C) appears at the top of the stack. When that happens, because there are no rules to reason about, model elimination would normally fail and begin backtracking, but instead proof-tree completion saves the literal off to the side, and proceeds as though it has just proven that literal. If a proof attempt succeeds, all the literals saved off to the side are conjoined and provide a partial rewriting of the original query; accumulating all such rewritings and disjoining them gives the full rewriting of the query in terms of the predicates with complete definitions. Proof-tree completion is so-named because the output of the procedure is a set of sentences that would complete all possible proofs of the given conjecture.

In the example, we would see a proof trace that looks something like what follows.

In order to prove r, we must prove q and p. Since q has a complete definition, when it is time to prove q, we save it off to the side. In order to prove p, we need to prove either s or t. The trace above chooses to try and prove s first. But since s has a complete definition, it can also be saved off to the side. Saving q and s is enough to finish a proof attempt and

Call: r	r
Call: q	[r] q
Save: q	[r] [q]
Call: p	[r] p
Call: s	[r] [p] s
Save: s	[r] [p] [s]
Exit: p	[r] [p]
Exit: r	[r]

Figure 4.1: Proof-tree completion trace, showing both the trace and the stack

produce a partial rewriting of the query: $q \wedge s$.

Continuing the process produces the final partial rewriting of r: $q \wedge t$. The disjunction of these two partial rewritings gives us the full rewriting of r in terms of $\{q, s, t\}$. Regardless of the definitions for $\{q, s, t\}$ contained in C, it must be the case that

$$C \cup I \models r \text{ if and only if } C \models (q \wedge s) \vee (q \wedge t).$$

The generalization to FOL/FHL syntax is straightforward. One constraint is that the sentences in the incomplete portion of the theory must be quantifier-free, i.e. in the prefix class $\forall^*$. The biggest difference in the actual execution of the algorithm is that the saved literals may contain variables. Care must be taken to ensure that if a variable is shared between two saved literals, that variable means the same thing. Also, each partial rewriting may contain variables that do not occur in the original query; such variables are implicitly existentially quantified.

Proof-tree completion can be used to compute a definition for $poss_\phi$ by relying on the following fact: ϕ is consistent with $C \cup I$ if and only if $C \cup I$ does not entail $\neg\phi$. We can therefore employ proof-tree completion to construct a sentence ψ so that whatever the contents of C,

$$C \cup I \models \neg\phi \text{ if and only if } C \models \psi$$

That is, ψ embodies the conditions under which ϕ is inconsistent with $C \cup I$. But because we are interested in conditions under which ϕ is consistent with $C \cup I$, we will define $poss_\phi$ to be the negation of ψ.

$$poss_\phi \Leftrightarrow \neg\psi$$

To illustrate this, we introduce a new example that is commonly studied in the constraint satisfaction literature: map coloring. Given a graph and a set of colors, paint the regions of the graph so that no two adjacent regions are colored the same. One fairly natural way to encode this problem is to start with the following two constraints.

$$color(x, y) \Rightarrow region(x) \wedge hue(y) \tag{4.3}$$
$$color(x, y) \wedge adj(x, z) \Rightarrow \neg color(z, y)$$

The first says that $color(x, y)$ is a binary predicate where the first argument can only be true of regions and the second argument can only be true of hues (the set of colors). The second says that if we paint region x hue y and x is adjacent to another region z then we cannot paint z with hue y.

In addition to these sentences, which comprise the incomplete theory I, we would have complete definitions for adj, $region$, and hue for representing the connectivity of the graph, the set of regions, and the set of hues. Since the proof-tree completion approach ignores C, we need not bother with the definitions of these three predicates.

Suppose we are interested in the possible colorings for three particular regions in the graph: $color(r_1, x) \wedge color(r_2, y) \wedge color(r_3, z)$, where r_1, r_2, and r_3 are object constants and x, y, and z are variables. That is, we want a definition for the predicate

$$poss_{color(r_1,x) \wedge color(r_2,y) \wedge color(r_3,z)}(x, y, z).$$

To compute this definition, we run proof-tree completion on the negation of the conjunction: $\neg(color(r_1, x) \wedge color(r_2, y) \wedge color(r_3, z))$, using incomplete axioms 4.3, and the complete predicates $\{region, hue, adj\}$.

The result is a disjunction of nine partial rewritings, listed below.

1. $\neg region(r_1)$
2. $\neg region(r_2)$
3. $\neg region(r_3)$
4. $\neg hue(x)$
5. $\neg hue(y)$
6. $\neg hue(z)$
7. $adj(r_1, r_2) \wedge x = y$
8. $adj(r_2, r_3) \wedge y = z$
9. $adj(r_1, r_3) \wedge x = z$

The *poss* definition is the negation of the disjunction of the above nine conditions.

$$poss_{color(r_1,x)\wedge color(r_2,y)\wedge color(r_3,z)}(x,y,z)$$
$$\Leftrightarrow$$
$$\left(\begin{array}{l} region(r_1)\wedge hue(x)\wedge (adj(r_1,r_2)\Rightarrow x\neq y)\wedge \\ region(r_2)\wedge hue(y)\wedge (adj(r_2,r_3)\Rightarrow y\neq z)\wedge \\ region(r_3)\wedge hue(z)\wedge (adj(r_1,r_3)\Rightarrow x\neq z) \end{array}\right)$$

Notice that this formulation of the definition is a fairly natural one. It requires that each region be assigned a hue, and if two regions are adjacent the two regions must be assigned different hues. Of course, in some sense this is exactly the same information conveyed by the original axioms. The difference here is that while the permitted colorings were only consistent with the original axioms, each and every one of the permitted colorings is entailed by the new formulation. Said another way, if we were to add statements that assert the map is colored two different ways to the original formulation, the result could be an unsatisfiable theory, but adding those same statements to the formulation above does not change satisfiability because all such statements are already entailed, albeit using a different vocabulary.

The map coloring problem has been studied extensively in the constraint satisfaction literature. It may be surprising to logicians that the formulation 4.3 cannot be expressed in the input language used at the yearly CSP solver competition[6], but the *poss* formulation of the problem is expressible in that language. The two communities seem to have slightly different formalizations of constraint satisfaction; the definition in the CSP community essentially amounts to a special case of the database query: a conjunctive query over extensional tables. For further discussion, see Section 5.7.

The benefit of the proof-tree completion approach is that it relies on well-understood technology for constructing the *poss* definition. One of the drawbacks is that without relying on the finite universe, there may be no finite rewriting of a sentence in terms of the complete predicates.

The typical example is ancestry. Suppose we want to rewrite the sentence $anc(x,y)$ in

[6] `http://cpai.ucc.ie/06/Competition.html`

terms of the complete table *parent*.

$$anc(x, y) \Leftarrow parent(x, y)$$
$$anc(x, y) \Leftarrow parent(x, z) \wedge anc(z, y)$$

Without knowing the size of the universe, proof-tree completion will attempt to unroll the transitivity, resulting in a never-ending procedure.

Another drawback is that even when the rewriting is finite, proof-tree completion compiles away every remnant of the incomplete portion of the theory at reformulation time, which can be expensive even for the very small map coloring example shown above. When compared to the next algorithm, the proof-tree completion approach pays a larger cost at reformulation time but a smaller cost at evaluation time. The next algorithm was constructed to process less of the incomplete theory at reformulation time, which requires that the remaining incompleteness is processed at evaluation time.

Once again the map coloring example illustrates, where the clausal form of the incomplete axioms is shown below.

$$\{\neg color(x, y), region(x)\}$$
$$\{\neg color(x, y), hue(y)\}$$
$$\{\neg color(x, y), \neg adj(x, z), \neg color(z, y)\}$$

One way to enumerate the valid map colorings would be to walk over all possible instances of $color(r_1, x) \wedge color(r_2, y) \wedge color(r_3, z)$ and for each one check whether it contradicted one of the three clauses above.

$$poss_{color(r_1,x)\wedge color(r_2,y)\wedge color(r_3,z)}(x, y, z) \Leftrightarrow \neg inconsistent(r_1, x, r_2, y, r_3, z)$$

The clause $\{\neg color(x, y), region(x)\}$ is contradicted if one of the r_i is not a region. Since this constraint applies to every *color* conjunct, we could encode the conditions that create a contradiction with this clause with a disjunction.

$$inconsistent(x_1, y_1, x_2, y_2, x_3, y_3) \Leftarrow \neg region(x_1) \vee \neg region(x_2) \vee \neg region(x_3)$$

Instead, we will construct a selector predicate *oneof* that when given the six arguments to

inconsistent will select each $\langle x_i, y_i \rangle$ individually.

$$oneof(x_1, y_1, x_2, y_2, x_3, y_3, z, w) \Leftrightarrow \left(\begin{array}{l} (z = x_1 \wedge w = y_1) \vee \\ (z = x_2 \wedge w = y_2) \vee \\ (z = x_3 \wedge w = y_3) \end{array} \right)$$

Using *oneof*, the rules that capture the conditions that produce a contradiction with either of the first two clauses are shown below.

$$inconsistent(x_1, y_1, x_2, y_2, x_3, y_3) \Leftarrow oneof(x_1, y_1, x_2, y_2, x_3, y_3, x, y) \wedge \neg region(x)$$
$$inconsistent(x_1, y_1, x_2, y_2, x_3, y_3) \Leftarrow oneof(x_1, y_1, x_2, y_2, x_3, y_3, x, y) \wedge \neg hue(y)$$

Finally, the third clause is contradicted whenever there is some pair of regions that are adjacent and colored the same.

$$inconsistent(x_1, y_1, x_2, y_2, x_3, y_3) \Leftarrow \left(\begin{array}{l} oneof(x_1, y_1, x_2, y_2, x_3, y_3, x, y) \wedge \\ adj(x, z) \wedge \\ oneof(x_1, y_1, x_2, y_2, x_3, y_3, z, y) \end{array} \right)$$

In this example, those three conditions are both sufficient and necessary for a contradiction.

$$inconsistent(x_1, y_1, x_2, y_2, x_3, y_3)$$
$$\Leftrightarrow$$
$$\left(\begin{array}{l} (oneof(x_1, y_1, x_2, y_2, x_3, y_3, x, y) \wedge \neg region(x)) \vee \\ (oneof(x_1, y_1, x_2, y_2, x_3, y_3, x, y) \wedge \neg hue(y)) \vee \\ (oneof(x_1, y_1, x_2, y_2, x_3, y_3, x, y) \wedge adj(x, z) \wedge oneof(x_1, y_1, x_2, y_2, x_3, y_3, z, y)) \end{array} \right)$$

This example demonstrates our second approach to constructing *poss* definitions: compute the necessary and sufficient conditions under which an instance of the target sentence $\phi(\overline{x})$ contradicts the original theory. Just like the proof-tree completion approach, this approach ignores the complete portion of the theory during the construction of the *poss* definition.

In the map coloring example, it was sufficient to construct a *poss* definition that checked whether the instance of the target sentence contradicted any of the original clauses. In general, however, failing to find an input clause that is inconsistent with an instance does not guarantee consistency. The map coloring example worked because the clauses represented

all the clausal prime implicates of the incomplete portion of the theory. (Recall that a clausal prime implicate of a theory is a clause d entailed by the theory such that no other clause entailed by the theory subsumes it.) It is well-known that a conjunction is inconsistent with a quantifier-free theory if and only if it is inconsistent with one of the clausal prime implicates of that theory. If the incomplete portion of the theory does not contain all its clausal prime implicates, we must first compute all those implicates.

When the incomplete portion of the theory is quantifier-free, the standard way of computing the prime implicates is to run resolution to closure. (The resolution closure is the result of repeatedly applying the usual inference rules of resolution to a set of sentences until no new consequences are produced—a fixed point is reached.) Because we are hiding the complete portion of the theory, C, from the reformulation procedure, we will end up computing the closure of just the incomplete portion of the theory, I, which as shown later is sufficient for our purposes. Unfortunately, just like in the proof-tree completion approach, the closure can be infinite. But, because of the finite, fixed universe of FHL, the closure can forcibly be made finite, making this technique uniformly applicable.

The other deficiency in the map coloring example is that every literal in

$$color(r_1, x) \wedge color(r_2, y) \wedge color(r_3, z)$$

has the same sign, ensuring that every instantiation is self-consistent. In general this will not be the case, and we must also check whether the particular instantiation of literals is inconsistent all by itself.

These observations lead us to Algorithm 6 for constructing a *poss* definition for a quantifier-free conjunction of literals, where the incomplete portion of the theory is in the $\forall^*$ prefix class.

Algorithm 6 QF-POSS(C, $\forall^*.I$, $[\neg]p_1(\overline{t_1}) \wedge \cdots \wedge [\neg]p_n(\overline{t_n})$)

1: $I :=$ RES(CLAUSAL-FORM(I))
2: $B := \emptyset$ {body of *poss* definition}
3: {build check for whether conjunction is consistent with the clauses}
4: **for all** $c \in I$ such that $SignedPreds(\neg c) \subseteq \{[\neg]p_1, \ldots, [\neg]p_n\}$ **do**
5: $D := \emptyset$ {conditions under which sentence is inconsistent with a particular clause}
6: **for all** literals $\pm p(\overline{u}) \in$ NEGATION-NORMAL-FORM($\neg c$) **do**
7: **if** p has a complete definition **then**
8: $D := D \cup \{\pm p(\overline{u})\}$
9: **else**
10: $\{\overline{t_{i_1}}, \ldots, \overline{t_{i_m}}\} = \{\overline{t} | \pm p(\overline{t})$ is a literal in $[\neg]p_1(\overline{t_1}) \wedge \cdots \wedge [\neg]p_n(\overline{t_n})\}$
11: $D := D \cup \{\pm oneof(\overline{t_{i_1}}, \ldots, \overline{t_{i_m}}, \overline{u})\}$
12: **end if**
13: **end for**
14: $B := B \cup \{\bigwedge D\}$
15: **end for**
16: {build check for self-consistency}
17: **for all** $p \in \{p_1, \ldots, p_n\}$ **do**
18: $pos := \{\overline{u_{i_1}}, \ldots, \overline{u_{i_k}}\} = \{\overline{t} | p(\overline{t})$ is a literal in $[\neg]p_1(\overline{t_1}) \wedge \cdots \wedge [\neg]p_n(\overline{t_n})\}$
19: $neg := \{\overline{n_{j_1}}, \ldots, \overline{n_{j_l}}\} = \{\overline{t} | \neg p(\overline{t})$ is a literal in $[\neg]p_1(\overline{t_1}) \wedge \cdots \wedge [\neg]p_n(\overline{t_n})\}$
20: $B := B \cup \{(oneof(\overline{u_{i_1}}, \ldots, \overline{u_{i_k}}, \overline{t}) \wedge oneof(\overline{n_{j_1}}, \ldots, \overline{n_{j_l}}, \overline{t}))\}$
21: **end for**
22: $\overline{x} :=$ VARS($[\neg]p_1(\overline{t_1}) \wedge \cdots \wedge [\neg]p_n(\overline{t_n})$)
23: $newvars :=$ VARS(B) $- \overline{x}$
24: output

$$poss_{[\neg]p_1(\overline{t_1}) \wedge \cdots \wedge [\neg]p_n(\overline{t_n})}(\overline{x})$$
$$\Leftrightarrow$$
$$\neg\exists\ newvars. \bigvee B$$

25: Define *oneof* as appropriate. (Actually, we need different *oneof* predicates for each arity.)
26: {Above, $\pm$ indicates either a positive or negative literal; every subsequent $\pm$ in the context of a $\pm$ has the same sign as the first, and every $\mp$ has the opposite sign.}

In essence, this algorithm constructs a definition for a $poss_{\phi(\overline{x})}$ predicate by constructing a definition that says when a particular instantiation of $\phi(\overline{x})$ is inconsistent with the background theory. The first case is when the instance contradicts one of the clausal prime implicates, and the second case is when the instance is inconsistent all by itself.

This algorithm relies on the fact that if a ground conjunction of literals is inconsistent with $C \cup I$ then it is inconsistent either with just C or there is some clause d in the resolution closure of I such that the ground conjunction is inconsistent with C and d. The following lemma guarantees that this property holds when I is in the $\forall^*$ prefix class. It models the fact that C is hidden from the reformulation algorithm but afterwards is used to determine inconsistency using CTR (the inference rule for theory resolution) to apply C to the result of the reformulation.

Lemma 2 (Ground Conjunctive Inconsistency). *Suppose $C \cup I$ is a finite set of FHL sentences, where C is a satisfiable, complete theory, I is a set of clauses, and the resolution closure of I is finite. $C \cup I$ is inconsistent with the ground conjunction*

$$[\neg]p_1(\overline{a_1}) \wedge \cdots \wedge [\neg]p_n(\overline{a_n})$$

if and only if either $\text{CTR+RES}(C, \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\})$ *contains the empty clause or there is some clause* $d \in \text{RES}(I)$ *such that* $\text{CTR}(C, \text{RES}(\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}))$ *contains the empty clause.*

The proof of this lemma appears in Appendix A.3.3.

The definitions produced by the algorithm above implement this lemma directly. They ensure that $poss_{[\neg]p_1(\overline{x_1})\wedge\cdots\wedge[\neg]p_n(\overline{x_n})}(\overline{a_1}, \ldots, \overline{a_n})$ is true exactly when $[\neg]p_1(\overline{a_1})\wedge\cdots\wedge[\neg]p_n(\overline{a_n})$ is not inconsistent with $C \cup I$.

The most variable cost in QF-POSS is computing the resolution closure of I. In an effort to manage this cost, we have looked into how completeness affects the resolution closure. The rest of this section details our findings and then briefly compares the two algorithms presented in this section for computing *poss* definitions.

The important observation about how completeness affects resolution is that the incomplete predicates do not constrain the complete predicates, but the complete predicates do constrain the incomplete predicates. The influence only flows one way. The first implication of this observation is that when computing $\text{RES}(I)$ for the purpose of QF-POSS, we never need to resolve upon predicates that have complete definitions.

Lemma 3 (Restricted Ground Conjunctive Inconsistency). *The Ground Conjunctive Inconsistency lemma holds even when instead of computing* RES(I)*, we compute the resolution closure of I but never resolve on a predicate with a complete definition in C.*

Another way to reduce the cost of computing the resolution closure is to leverage the fact that sometimes a theory is built out of a number of independent subtheories. If we need to construct a *poss* definition, and the target sentence is only constrained by one of the independent subtheories, then it turns out that we need only compute the resolution closure of that subtheory. This idea of restricting resolution so that it only applies inference rules to the important portion of the theory is commonplace in resolution theorem-proving research. One of the earliest results in this area is the set of support restriction [CL73].

Given a satisfiable theory Δ, and a query ϕ, the set of support restriction says that the first application of binary resolution must use at least one clause from $\neg\phi$; every subsequent resolution must be performed using at least one clause from $\neg\phi$ or the result of another application of resolution. This ensures that every literal resolved upon must occur in a clause that is syntactically related to the query. When a theory is built out of several independent subtheories, where the predicates in the query only (transitively) occur in a small number of subtheories, the set of support restriction ensures that none of the clauses in the irrelevant subtheories are resolved upon. In those cases where the relevant subtheories are very small as compared to the irrelevant ones, the set of support restriction can produce large savings.

Returning to the hospital sensor net example, we can view the theory as being comprised of three independent subtheories: the sentences constraining *patient*, the sentences constraining *oxygen*, and the sentences defining *adj*. (Below the *adj* definitions have been omitted for brevity; the salient property of those definitions is that none of them mention *patient* or *oxygen*.)

$$patient(x) \wedge patient(y) \Rightarrow x = y$$
$$patient(a) \vee patient(b) \vee patient(e)$$
$$oxygen(x) \wedge oxygen(y) \Rightarrow x = y$$
$$oxygen(l) \vee oxygen(o) \vee oxygen(p)$$

We see here that the sentences that constrain *patient* do not mention *oxygen*, and vice versa. Thus, intuitively if we had an entailment query that just mentioned the predicate *patient*, we should only need to perform resolutions on the query and the two sentences mentioning *patient*; the set of support restriction tries to ensure that is the case.

This idea is important for constructing *poss* definitions because if we needed to compute the definition for $poss_{patient(x)}$ using QF-POSS, it is sufficient to compute the closure of the sentences constraining *patient*; likewise, for computing the definition for $poss_{oxygen(x)}$, it is sufficient to compute the closure of the sentences constraining *oxygen*. The reason this works is that *patient*, *oxygen*, and *adj* are all independent of one another.

Recall from Section 4.2.1 that the definition of Sentence Independence is that if two sentences are individually consistent with a theory then their conjunction is consistent with the theory. If we view a subtheory as a conjunction of sentences then this same definition applies equally well to subtheories. In the special case where all sentences that mention some set of predicates P_1 are independent of all sentences that mention some other set of predicates P_2, we say that predicates P_1 are independent of predicates P_2. Given all the predicates in a vocabulary, we can partition those predicates so that all the predicates in one partition are dependent on one another, and if two predicates are not in the same partition they are independent of one another.

As mentioned in Section 4.2.1, we can compute an approximation to this predicate partitioning by using a Semantic Dependency Graph. Each node in the graph corresponds to a predicate in the language, and there is an edge between two predicates if there is some sentence in which both predicates occur. The predicate partitioning consists of one partition for each of the connected-components in the graph: if one predicate can be reached from another then the two predicates belong to the same partition.

This algorithm for computing the predicate partitioning is not new. What is new is the result that says we can remove any predicate from the dependency graph that has a complete definition. Sometimes removing those nodes results in a graph that has more connected-components, which means that a larger number of the predicates are independent and the computation of the resolution closure may be less expensive.

This is illustrated in the hospital example. Consider just the four axioms listed above. The resulting dependency graph has three nodes: *patient*, *oxygen*, and =. There is an edge between *patient* and = as well as an edge between *oxygen* and =. Thus, the naïve partitioning algorithm would say that because there is a path between *patient* and *oxygen* that *patient* and *oxygen* are dependent; computing the resolution closure for say $poss_{patient(x)}$ would then require computing the closure of all four sentences. But because = has a complete definition, its node can be removed from the dependency graph, and our algorithm for predicate partitioning would determine (correctly) that *patient* and *oxygen* are actually

independent. Thus, when computing the resolution closure for $poss_{patient(x)}$, we would only need the closure of the two sentences mentioning *patient*.

We can think of this result as stating that dependence does not flow through predicates that have complete definitions. Formally the result appears in Section 4.2.1 under the title Syntactic Independence. That result is a corollary to the theorem below, which follows the definition of a Semantic Dependency Graph, repeated here for clarity.

Definition 8 (Semantic Dependency Graph). *A Semantic Dependency Graph for a set of sentences Δ is a graph $\langle V, E\rangle$.*

- *$u \in V$ if and only if u is a predicate in Δ without a complete definition*
- *The undirected edge (u, v) is in E if and only if there is some sentence in Δ that uses both the predicates u and v.*

Theorem 16 (Dependence Decomposition). *Let Δ be a set of satisfiable sentences in FHL and G its Semantic Dependency Graph whose connected-components are $O_1, \ldots, O_m$. Suppose Δ is partitioned into $C \cup I_1 \cup \cdots \cup I_m$, where C corresponds to the complete portion of the theory, and each I_i corresponds to all the sentences mentioning some predicate in O_i. Suppose further that ϕ is a sentence whose predicates are a subset of those in $C \cup I_{j_1} \cup \cdots \cup I_{j_k}$, where $j_l \in \{1, \ldots, m\}$ for every l.*

$$\Delta \models \phi \text{ if and only if } C \cup I_{j_1} \cup \cdots \cup I_{j_k} \models \phi$$

The proof can be found in Appendix A.3.3.

The restriction on resolution presented here, computing the closure of just the necessary subtheory by resolving only on incomplete predicates, reduces the cost of computing the resolution closure of the incomplete portion of the theory, I, and in so doing reduces the cost of QF-POSS for constructing *poss* definitions where both the sentence in the subscript and I are quantifier-free. In order to put reasonably good bounds on the run-time of QF-POSS, we would need reasonably good bounds on the cost of computing the resolution closure of an arbitrary set of quantifier-free sentences with a finite universe. To our knowledge no such bounds are known, and thus the complexity of QF-POSS is the subject of future work.

The two techniques discussed in this section are closely related. Both the proof-tree and the QF-POSS approach compute a closure; the proof-tree computes the closure top-down

while using the actual target sentence, and the resolution approach computes the closure bottom-up without that sentence. At least for the map coloring example, the proof-tree approach costs more at reformulation time but less at evaluation time than the QF-POSS approach. How well that statement generalizes is unclear; studying the tradeoffs of these two algorithms is the subject of future work.

With such algorithms at our disposal, we can construct all of the *poss* definitions necessary for using Extensional Reasoning to answer entailment queries of the following form.

$$C \cup \forall^*.I \models \forall \overline{x}.\phi(\overline{x})$$

This theorem is subsumed by the theorem that appears at the end of the next section; hence, its formal statement and proof are presented there.

4.2.3 Quantified Sentences

Once we have an algorithm that constructs *poss* definitions for quantifier-free sentences, it is straightforward to apply that same algorithm for sentences that include quantifiers as long as we can perform quantifier-elimination, i.e. rewrite a sentence with quantifiers into a sentence without quantifiers such that the two sentences are logically equivalent. In Finite Herbrand Logic, quantifier elimination is trivial. Suppose the set of object constants is $\{a_1, \ldots, a_m\}$. Then we can employ the following equivalences to perform quantifier-elimination.

$$\forall x.\phi(x) \text{ is logically equivalent to } \phi(a_1) \wedge \cdots \wedge \phi(a_m)$$
$$\exists x.\phi(x) \text{ is logically equivalent to } \phi(a_1) \vee \cdots \vee \phi(a_m)$$

Thus, given that we need to find a definition for $poss_{\phi(\overline{x})}$, we can perform quantifier-elimination on $\phi(\overline{x})$, and since the result has no quantifiers, use the algorithms from the last section to construct a definition for $poss_{\phi(\overline{x})}$.

Sometimes this approach is reasonable, but notice that the cost of quantifier-elimination is exponential in the number of quantifiers, n, where the base is the size of the universe: $|U|^n$. This cost must be paid at reformulation time, and then the resulting sentence, which is $|U|^n$ times larger than the original must be manipulated by the algorithms of the last section.

To avoid paying these exponentials at reformulation time, when because we are ignoring

the complete theory C, we may not have enough information to prudently prune the enumeration of the universe, we developed a way to delay that enumeration until evaluation time for the case of universal quantifiers.

Suppose we are interested in a sentence of the form $\forall x_1 \ldots x_n.\phi(x_1, \ldots, x_n, \overline{y})$, where $\phi(x_1, \ldots, x_n, \overline{y})$ is quantifier-free. Just as suggested above, our approach performs quantifier-elimination, but does so without enumerating a cross product of the universe. Then, the QF-POSS algorithm from the last section can be used, with a small change, to construct a definition for $poss_{\forall x_1 \ldots x_n.\phi(x_1,\ldots,x_n,\overline{y})}$.

To perform quantifier-elimination, we will leverage the fact that the universe is finite and known: $\{a_1, \ldots, a_m\}$. The first step is to construct a new object constant, call it u, which represents all the object constants in the original universe.

Then, for each universally quantified variable in the sentence, we will plug in this new object constant.

$$\forall x_1 \ldots x_n.\phi(x_1, \ldots, x_n, \overline{y}) \text{ becomes } \phi(u, \ldots, u, \overline{y})$$

The result is a sentence with free variables $\overline{y}$, possibly with a mix of the original object constants and our new über-constant. It is noteworthy that this procedure amounts to a careful application of Herbrandization.

The only change needed to use the QF-POSS algorithm from the last section involves the definition of $oneof$. Recall that $oneof(\overline{t_1}, \ldots, \overline{t_k}, \overline{t})$ is a selector: it is true whenever $\overline{t}$ is equal to $\overline{t_i}$ for some i. The assumption made in QF-POSS is that when a $oneof$ definition is evaluated, $\overline{t}$ consists of object constants from the original universe. Now, however, the new object constant u may occur in one of the $\overline{t_i}$ and so may be selected, which means that if we left the definition of $oneof$ as is, the assumption made by QF-POSS would not hold. The fix supposes u might appear anywhere within any of the $\overline{t_i}$, and whenever one of the $\overline{t_i}$ is selected as before, each instance of u is turned into all the elements in the original universe.

So that we can avoid changing the QF-POSS algorithm from the last section, we refer to the old definition of $oneof$ with $oneof'$. The new definition of $oneof$ is shown below and relies on $oneof'$.

$$oneof(\overline{x_1}, \ldots, \overline{x_k}, \overline{y}) \Leftrightarrow \exists \overline{x}.(oneof'(\overline{x_1}, \ldots, \overline{x_k}, \overline{x}) \wedge extract(\overline{x}, \overline{y}))$$

Below we show $extract$ where $\overline{x}$ is a single variable. Notice that the definition also

applies to the original object constants.

$$extract(x, y) \Leftrightarrow \left(\begin{array}{c} (x = u \wedge y = a_1) \vee \\ (x = u \wedge y = a_2) \vee \\ \vdots \\ (x = u \wedge y = a_m) \vee \\ (x = a_1 \wedge y = a_1) \vee \\ \vdots \\ (x = a_m \wedge y = a_m) \end{array} \right)$$

In summary, to build *poss* definitions for sentences with universal quantifiers, we can first replace every universally quantified variable with u, a new object constant, and then change the definition of *oneof* as described above. The benefit of delaying the expansion of the universal quantifiers until evaluation time is that sometimes during evaluation it becomes clear that not all of the elements in the universe actually need to be enumerated, and in those cases this approach can be more efficient than approaches that enumerate the entire universe at reformulation time. Finding the balance point—selecting which algorithm to use under what circumstances—is the subject of future work.

In the context of Extensional Reasoning, this algorithm is sufficient to construct all the *poss* definitions needed for answering entailment queries of the following form, where $\phi(\overline{x}, \overline{y})$ is quantifier free.

$$C \cup \forall^*.I \models \forall\overline{x}\exists\overline{y}.\phi(\overline{x}, \overline{y})$$

Theorem 17. *Suppose we have an algorithm for computing a definition for a poss predicate where the sentence in the subscript is a conjunction of literals that possibly mentions a new object constant u representing the set of all object constants. Further suppose we have an FHL entailment query in the class*

$$C \cup \forall^*.I \models \forall\overline{x}\exists\overline{y}.\phi(\overline{x}, \overline{y}),$$

where C is a complete theory, $C \cup I$ is satisfiable, and $\phi(\overline{x}, \overline{y})$ is quantifier-free. We can construct a completed theory Δ' and a new query ϕ' such that

$$C \cup \forall^*.I \models \forall\overline{x}\exists\overline{y}.\phi(\overline{x}, \overline{y}) \text{ if and only if } \Delta' \models \phi'.$$

Proof. This query only requires constructing *poss* definitions for quantifier-free sentences that mention u, as is demonstrated below.

$$
\begin{aligned}
& C \cup \forall^*.I \models \forall\overline{x}\exists\overline{y}.\phi(\overline{x},\overline{y}) \\
& \Leftrightarrow \neg poss_{\neg\forall\overline{x}\exists\overline{y}.\phi(\overline{x},\overline{y})} \text{ (by the definition of } poss) \\
& \Leftrightarrow \neg poss_{\exists\overline{x}\forall\overline{y}.\neg\phi(\overline{x},\overline{y})} \text{ (by logical equivalence)} \\
& \Leftrightarrow \neg\exists\overline{x}.poss_{\forall\overline{y}.\neg\phi(\overline{x},\overline{y})}(\overline{x}) \text{ (by distribution of } poss \text{ over } \exists) \\
& \Leftrightarrow \neg\exists\overline{x}.poss_{\neg\phi(\overline{x},\overline{u})}(\overline{x}) \text{ (by the semantics of } u)
\end{aligned}
$$

Moreover, the *poss* definition for a quantifier-free sentence can always be constructed given that we have an algorithm for constructing the *poss* definitions of conjunctions of literals that may mention u.

$$
\begin{aligned}
& \Leftrightarrow \neg\exists\overline{x}.poss\left(\begin{array}{c} l_{11}(\overline{x_{11}}) \wedge \cdots \wedge l_{1n_1}(\overline{x_{1n_1}}) \\ \vee \cdots \vee \\ l_{m1}(\overline{x_{m1}}) \wedge \cdots \wedge l_{mn_m}(\overline{x_{mn_m}}) \end{array}\right)(\overline{x}) \text{ (by QF equivalence to DNF)} \\
& \Leftrightarrow \neg\exists\overline{x}.\left(\begin{array}{c} poss_{l_{11}(\overline{x_{11}})\wedge\cdots\wedge l_{1n_1}(\overline{x_{1n_1}})}(\overline{x_1}) \\ \vee \cdots \vee \\ poss_{l_{m1}(\overline{x_{m1}})\wedge\cdots\wedge l_{mn_m}(\overline{x_{mn_m}})}(\overline{x_m}) \end{array}\right) \text{ (by distribution of } poss \text{ over } \vee)
\end{aligned}
$$

The last sentence above constitutes the new query, ϕ'. If we invoke our algorithm to construct definitions for each of the *poss* predicates mentioned in the last sentence above and combine those definitions with C, we get Δ'. Since the algorithm constructs the correct definitions for *poss* predicates, we know the conclusion of the theorem holds.

$$C \cup \forall^*.I \models \forall\overline{x}\exists\overline{y}.\phi(\overline{x},\overline{y}) \text{ if and only if } \Delta' \models \phi'.$$

□

4.2.4 The General Case

The previous two sections concentrated on the class of theories where the incomplete portion was quantifier-free in the hopes of leveraging well-known proof techniques for constructing *poss* definitions. This section considers the general case, demonstrating an algorithm that can construct a *poss* definition for an arbitrary sentence and an arbitrary theory, which

essentially amounts to a constructive proof that the problem is decidable. Unlike the previous two sections that took a proof-theoretic approach, the algorithm presented here takes a model-theoretic approach, constructing a definition that when evaluated walks over models looking for one that satisfies both the theory and the sentence in question.

Because Finite Herbrand Logic theories always have a fixed, known universe, we could construct a complete theory that enumerates all possible models in the language. The approach taken here does essentially that, but attempts to factor out commonalities so as to reduce the cost of reformulation. This approach produces definitions that can be conceptualized as a simple loop over models: enumerate all the models for the vocabulary, and for each one check whether it satisfies the theory and the sentence of interest. If there is such a model, then the *poss* predicate is true; otherwise, it is false.

Suppose we had a predicate m whose arity is equal to the number of predicates in the theory. We want it to be the case that if $m(a_1, \ldots, a_n)$ is true, then a_i provides the interpretation for predicate p_i. Suppose we also have predicates $sat_{C \cup I}$ and sat_ϕ that determine whether a particular model $a_1, \ldots, a_n$ satisfies $C \cup I$ and ϕ, respectively. With definitions for such predicates in place, we could construct a definition for $poss_\phi$ as follows.

$$poss_\phi \Leftrightarrow \exists \overline{x}.(m(\overline{x}) \wedge sat_{C \cup I}(\overline{x}) \wedge sat_\phi(\overline{x}))$$

The rest of the section is concerned with constructing definitions for these three predicates. We start with the definition for m, which takes as many arguments as there are predicates in the vocabulary. If $m(a_1, \ldots, a_n)$ is true, then we know each object constant a_i represents the interpretation for predicate p_i. Thus each of these object constants represents an entire relation of object constants from the original theory, and the first step to constructing a definition for m is to construct a set of object constants that represent all the necessary relations. Once that has been accomplished, m can be defined as the n-wide cross product of those object constants, where n is the number of predicates.

For example, consider the case of two monadic predicates $\{r, s\}$ and a universe of two values $\{a, b\}$. In this case, we use four über object constants $\{ab, a, b, \emptyset\}$, each of which represents a unary relation over the universe $\{a, b\}$. The constant ab represents the relation that includes two one-tuples: $\{\langle a \rangle, \langle b \rangle\}$. The constant a represents the relation that includes a single one-tuple $\{\langle a \rangle\}$; likewise b represents $\{\langle b \rangle\}$. The constant $\emptyset$ represents the empty relation.

In this example, the predicate m must be defined so that all combinations of interpretations for r and s are true. That is, m represents the following table, where each row represents a different model.

r	s
ab	ab
ab	a
ab	b
ab	$\emptyset$
a	ab
a	a
a	b
a	$\emptyset$
b	ab
b	a
b	b
b	$\emptyset$
$\emptyset$	ab
$\emptyset$	a
$\emptyset$	b
$\emptyset$	$\emptyset$

Of course, there is no reason to represent this table explicitly; if we had a predicate $univ^1$ true of the four object constants $\{ab, a, b, \emptyset\}$, then we could implicitly define m as follows.

$$m(x, y) \Leftrightarrow univ^1(x) \wedge univ^1(y)$$

The superscript on $univ^1$ indicates that it is true of all relations of arity one. If the theory contained predicates with arities different from one, we would need a $univ^k$ for every arity k. A predicate of arity k would have $2^{|U|^k}$ different relations, which means there would be that many object constants for representing those relations. Then, if the predicates $p_1, \ldots, p_n$ had arities $k_1, \ldots, k_n$, then the definition for m would be as follows.

$$m(x_1, \ldots, x_n) \Leftrightarrow univ^{k_1}(x_1) \wedge \cdots \wedge univ^{k_n}(x_n)$$

As it turns out, we can do better than this because the theory of interest has been partitioned into $C \cup I$. There is no need to consider any model that does not agree with C on the predicates with complete definitions. All we should be doing is walking over the interpretations of incomplete predicates. The machinery above is sufficient to ensure that $m(a_1, \ldots, a_n)$ means that incomplete predicate p_i has interpretation a_i, so it is this definition for m that we will assume in what follows. The definitions for the complete predicates are included in C itself.

Now that we have a way to construct the definition for m all that remains are methods for constructing definitions for $sat_{C \cup I}$ and sat_ϕ. Because we are using the definitions of C to tell us what the interpretations of the complete predicates are, every model we consider will necessarily satisfy C; thus, what we really need are definitions for sat_I and sat_ϕ. And since I is a finite set of sentences, if we have an algorithm for constructing sat_ψ for an arbitrary sentence ψ, then that algorithm will suffice for constructing both the definitions for sat_ϕ and sat_I.

sat_ψ takes as arguments the interpretations for each of the incomplete predicates in the vocabulary, as described above. The algorithm for constructing the definition for sat_ψ replaces the incomplete predicates in ψ with constructs that lookup the interpretations for those predicates in the arguments to sat_ψ. Since each interpretation is represented with an über object constant, we need some way of unpacking the über constants. The predicate *extract* is again used for this purpose; it is the construct used for looking up the interpretation of an incomplete predicate.

Define $extract(v, \overline{a})$ to be true whenever $\overline{a}$ is some tuple of object constants from the original theory that is included in the relation represented by the über object constant v. With the definition of *extract* in place, the algorithm for constructing a definition for $sat_\psi(x_1, \ldots, x_n)$ is straightforward. In ψ, find all occurrences of an atom, $p_i(\overline{t_i})$, where the predicate, p_i, does not have a complete definition. Replace each one with $extract(x_i, \overline{t_i})$, where x_i corresponds to the interpretation for p_i. Call the sentence that results from performing all those replacements $\psi'(x_1, \ldots, x_n)$, which unlike ψ may mention the free variables $x_1, \ldots, x_n$. The definition for $sat_\psi(x_1, \ldots, x_n)$ has exactly the body $\psi'(x_1, \ldots, x_n)$.

$$sat_\psi(x_1, \ldots, x_n) \Leftrightarrow \psi'(x_1, \ldots, x_n)$$

The intuition for why this works is that while the predicates with complete definitions are

left untouched in $\psi'(x_1, \ldots, x_n)$, which makes sense because they already have definitions in C, the incomplete predicates are defined only by the arguments handed to sat_ψ. Hence, only the atoms built from incomplete predicates must derive their interpretations from the arguments to sat_ψ.

For example, suppose we wanted a definition for the predicate $poss_{\exists x.(r(x) \wedge s(x))}$ with a background theory whose universe is $\{a, b\}$ and consists of the following two sentences.

$$r(x) \Leftrightarrow x = a$$
$$\exists x.s(x)$$

Notice that r has a complete definition, and s does not. The complete portion of the theory is a single sentence: $r(x) \Leftrightarrow x = a$. A definition for $poss_{\exists x.(r(x) \wedge s(x))}$ requires definitions for m, $sat_{\exists x.(r(x) \wedge s(x))}$, and $sat_{\exists x.s(x)}$, each of which takes a single argument representing the interpretation for the lone incomplete predicate s.

$$poss_{\exists x.(r(x) \wedge s(x))} \Leftrightarrow \exists y.(m(y) \wedge sat_{\exists x.(r(x) \wedge s(x))}(y) \wedge sat_{\exists x.s(x)}(y))$$

To build the definition for m, we start by constructing the über constants for the language. Because s is monadic, we need one constant for each subset of the universe $\{a, b\}$: $\{ab, a, b, \emptyset\}$. m is true of exactly those four über constants.

$$m(x) \Leftrightarrow (x = ab \vee x = a \vee x = b \vee x = \emptyset)$$

To compute the definition for $sat_{\exists x.(r(x) \wedge s(x))}(y)$, we look at the sentence $\exists x.(r(x) \wedge s(x))$ and replace $s(x)$, the only atom built out of an incomplete predicate, with $extract(y, x)$. Notice that y is the argument passed into the definition. This produces the following definition.

$$sat_{\exists x.(r(x) \wedge s(x))}(y) \Leftrightarrow \exists x.(r(x) \wedge extract(y, x))$$

The definition for $sat_{\exists x.s(x)}$ is shown below and was produced by replacing $s(x)$ with $extract(y, x)$ in $\exists x.s(x)$.

$$sat_{\exists x.s(x)}(y) \Leftrightarrow \exists x.extract(y, x)$$

The definition for *extract* is shown below. It simply enumerates the meaning of the four über constants $\{ab, a, b, \emptyset\}$, as described in text above. ab, a, and b are each translated into

their components, and because $\emptyset$ represents the empty relation, $extract(\emptyset, d)$ is false for all object constants d.

$$extract(x, y) \Leftrightarrow \left(\begin{array}{l} (x = ab \wedge y = a) \vee \\ (x = ab \wedge y = b) \vee \\ (x = a \wedge y = a) \vee \\ (x = b \wedge y = b) \end{array} \right)$$

Because r has a complete definition in C, that definition must be included with the definitions for m and the two *sat* predicates.

$$r(x) \Leftrightarrow x = a$$

This completes the definitions needed to support the definition for $poss_{\exists x.(r(x) \wedge s(x))}$, repeated below.

$$poss_{\exists x.(r(x) \wedge s(x))} \Leftrightarrow \exists y.(m(y) \wedge sat_{\exists x.(r(x) \wedge s(x))}(y) \wedge sat_{\exists x.s(x)}(y))$$

To see that all of these definitions are correct, first notice that the three sentences of interest, shown again below, are satisfied by the model $\{s(a), s(b), r(a)\}$; thus, $poss_{\exists x.(r(x) \wedge s(x))}$ should be true. To show that it is, we will show that the object constant ab, which represents (s's interpretation in) this model satisfies the three conjuncts in the body of the *poss* definition.

$$r(x) \Leftrightarrow x = a$$
$$\exists x.s(x)$$
$$\exists x.(r(x) \wedge s(x))$$

First, $m(ab)$ is true since m's definition includes the object constant ab. Second, for $sat_{\exists x.(r(x) \wedge s(x))}(ab)$ to be true, it must be the case that $\exists x.(r(x) \wedge extract(ab, x))$ is true. And that holds because $r(a)$ and $extract(ab, a)$ are both true. Lastly, $sat_{\exists x.s(x)}(ab)$ is true whenever $\exists x.extract(ab, x)$ is true, and this statement holds because $extract(ab, a)$ (and for that matter $extract(ab, b)$) is true.

The purpose of examining an algorithm of this sort is to provide a baseline that works for all cases. The reformulation cost of this algorithm is $2^{|U|^k}$ for each incomplete predicate of arity k, where U is the universe, plus the cost of constructing *extract*, which costs at least that much. This gives us an upper bound on the cost of constructing *poss* definitions, assuming the algorithm knows the universe. It is tempting to investigate algorithms that are not aware of the universe, or even its size, so that the reformulation cost cannot be

dependent on the size of the universe. However, it is our belief that without having access to the (size of the) universe, constructing a *poss* definition is undecidable (since satisfiability with finite models is undecidable), but that proof is the subject of future work.

4.2.5 Extensional Reasoning with Incomplete Theories

While the theory completion approach to handling incomplete theories was motivated by the database setting, where the incomplete portion of the theory I and the query are static but the complete portion C changes, it can also be deployed in the theorem proving setting, where the goal is to answer a single entailment query. Algorithm 7 demonstrates how theory completion can be applied to answer an entailment query: given Δ and ϕ, construct a complete theory Δ' and a new query ϕ' such that $\Delta \models \phi$ if and only if $\Delta' \models \phi'$ and then employ the algorithm from Chapter 3 to answer that query.

Algorithm 7 FULL-ER-ENTAILEDP(Δ, ϕ)

Assumes: Δ is a finite sentence set in FHL and ϕ is a closed FHL sentence

1: $C \cup I :=$ TO-BICONDS-MAX(Δ)
2: $cpreds :=$ PREDS-DEFINED(C)
3: $\phi' :=$ SIMPLIFY-POSS($\neg poss_{\neg\phi}$, $cpreds$, I)
4: $\Delta' := C \cup$ POSS-DEFINITIONS(POSS-PREDS(ϕ'), $cpreds$, I)
5: **return** ER-ENTAILEDP(Δ', ϕ')

The function PREDS-DEFINED returns all the predicates with complete definitions in the given sentence set. The function POSS-PREDS returns all the *poss* predicates in the given sentence. The function POSS-DEFINITIONS computes definitions for all the given *poss* predicates using the complete predicates and the incomplete portion of the theory I.

The beginning of this chapter introduced TO-BICONDS-MAX for partitioning a theory into the complete portion C and the incomplete portion I. Section 4.2.1 explained how to implement SIMPLIFY-POSS, and the subsequent sections explained different approaches for implementing POSS-DEFINITIONS.

Because FULL-ER-ENTAILEDP constructs a new entailment query over a complete theory and then calls ER-ENTAILEDP on that entailment query, it is sound and complete to the extent that ER-ENTAILEDP is sound and complete. The crucial argument shows that the constructed entailment query $\Delta' \models \phi'$ holds if and only if the original entailment query $\Delta \models \phi$ holds.

Lemma 4 (Entailment Preservation of FULL-ER-ENTAILEDP). *Under the following conditions, after executing lines (1) through (4),* $\Delta \models \phi$ *if and only if* $\Delta' \models \phi'$*, where* Δ' *is a complete, satisfiable theory.*

- ϕ *is a closed sentence*
- TO-BICONDS-MAX(Δ) *returns a* $C \cup I$ *that is logically equivalent to* Δ*, and where* C *is a complete, satisfiable theory.*
- SIMPLIFY-POSS($\neg poss_{\neg\phi}$, *cpreds*, I) *returns a sentence that is logically equivalent to* $\neg poss_{\neg\phi}$ *according to the semantics of poss and* C*.*
- POSS-DEFINITIONS(POSS-PREDS(ϕ'), *cpreds*, I) *returns a complete, satisfiable theory consistent with* C *such that each of the poss predicates that appear in* ϕ' *are defined according to the semantics of poss for the background theory* $C \cup I$*.*

This proof can be found in Appendix A.3.4.

These algorithms can also be deployed in the database setting, which is beneficial because the cost of theory completion can be amortized across multiple complete theories. After theory completion is performed, the algorithm hands each complete theory and the result of theory completion to the algorithm introduced in Chapter 3, ER-ENTAILEDP, for answering entailment queries about complete theories. AMORTIZED-FULL-ER-ENTAILEDP, shown as Algorithm 8, prints out the answer to the entailment query for each complete theory.

Algorithm 8 AMORTIZED-FULL-ER-ENTAILEDP(*cpreds*, Cs, I, ϕ)

Assumes: *cpreds* is a set of predicates, Cs is a set of sets of complete definitions for the predicates in *cpreds*, I is a finite sentence set in FHL, ϕ is a closed FHL sentence.
Assumes: for every $C \in Cs$, C is consistent with I.
1: $\phi' :=$ SIMPLIFY-POSS($\neg poss_{\neg\phi}$, *cpreds*, I)
2: $I' :=$ POSS-DEFINITIONS(POSS-PREDS(ϕ'), *cpreds*, I)
3: **for all** $C \in Cs$ **do**
4: **print** ER-ENTAILEDP($C \cup I'$, ϕ')
5: **end for**

4.2.6 Amortization

The last few sections focused on algorithms for reformulating an FHL entailment query into a database query. We know that since entailment in FHL is coNEXPTIME-complete

and query evaluation in *nr-datalog*$^\neg$ is PSPACE-complete, the reformulation is likely to be exponential in some cases[7]. Because the purpose of Extensional Reasoning is efficiency, an exponential reformulation cost can be problematic in some situations. We have already seen one setting in which this cost can be amortized over multiple problem instances: the database setting. This section discusses various ways of amortizing the cost of reformulation incurred while performing theory completion.

The opportunities for amortization can be grouped into two categories: those in which the theory does not change and those in which it does. In both cases, we are answering a stream of queries, the contents of which may or may not be known in advance.

When the theory is static and we have multiple queries to evaluate, it may be the case that the reformulation performed for the first query is sufficient for answering all queries. If the set of *poss* predicates necessary for answering the first query is the same as the set of *poss* predicates for answering the remaining queries then the reformulation cost can be amortized over all the queries.

For example, in the hospital sensor network setting, we may be interested in two related queries. (The second introduces a new predicate *nearby*, but suppose it has a complete definition.)

$$\forall xy.(patient(x) \wedge oxygen(y) \Rightarrow \neg adj(x,y)$$
$$\forall xy.(patient(x) \wedge oxygen(y) \Rightarrow nearby(x,y)$$

The first asks whether it is certain that the patient and the oxygen have been separated; the second asks whether it is certain that the patient and the oxygen have not been separated. When written in terms of *poss*, we have the following two queries.

$$\forall xy.(poss_{patient(x)}(x) \wedge poss_{oxygen(y)}(y) \Rightarrow \neg adj(x,y)$$
$$\forall xy.(poss_{patient(x)}(x) \wedge poss_{oxygen(y)}(y) \Rightarrow nearby(x,y)$$

The *poss* predicates needed to answer the two queries are the same; hence, the construction of those predicates can be amortized over both queries.

More generally, one would expect certain queries to rely on certain *poss* predicates while others to rely on other *poss* predicates. In this situation, when a new query arrives that cannot be answered solely by the existing *poss* predicate definitions, it may be the case that

[7]More precisely, the reformulation will require exponential time in some cases unless there is a collapse of this portion of the complexity hierarchy.

some of the existing *poss* predicate definitions can be reused. In this setting, the cost of constructing any particular *poss* definition can be amortized over all the queries that rely on that definition. If the query stream is finite and known ahead of time, we can construct all the necessary *poss* definitions before answering any queries. If the query stream is not known ahead of time, but we are willing to pay the cost of constructing *poss* definitions on the fly, the set of *poss* definitions can be constructed incrementally.

On the other hand, if the query stream is infinite or unknown and we are unwilling to pay the cost of constructing *poss* definitions at query-time, we may need to construct a database system that faithfully represents all the information contained in the original, incomplete theory. Notice that this is different from the *poss* definition approach because there only the fraction of the theory needed to answer the given query was represented by the database system. Here the goal is to construct a database system that represents the original theory in its entirety.

The model-theoretic approach to constructing *poss* definitions described in Section 4.2.4 provides enough machinery for performing such a reformulation. Recall that in order to define $poss_\phi$ with respect to a background theory Δ, we constructed definitions for $m(\overline{x})$, $sat_\Delta(\overline{x})$, and $sat_\phi(\overline{x})$ and defined $poss_\phi$ as follows.

$$poss_\phi \Leftrightarrow \exists \overline{x}.(m(\overline{x}) \wedge sat_\Delta(\overline{x}) \wedge sat_\phi(\overline{x}))$$

The predicate m represents all models (or more precisely the interpretations of the incomplete predicates in all models), and the predicates sat_ϕ and sat_Δ represented whether or not a particular model satisfied ϕ and Δ, respectively. This same machinery can be used to construct a database that faithfully represents all the information in the original theory.

The simplest way to see this is to define $m(\overline{x})$ to be true of exactly those models that satisfy Δ. Then, to answer the query $\Delta \models \phi$, construct the definition for sat_ϕ, which costs time linear in the size of ϕ. The query is then

$$\forall \overline{x}.(m(\overline{x}) \Rightarrow sat_\phi(\overline{x})).$$

Constructing an efficient definition for m is therefore the crucial part of this approach, but however it is accomplished, once we have a definition for m, answering a query requires only linear reformulation in the size of the query. Thus, the cost for constructing m, even if exorbitant, can be amortized away with a large enough query stream.

Static theories are normal in the context of automated deduction, but they are rare in the context of databases. As the system the database models changes, so too must the database. Likewise, if we use logic to model some complex system and that system changes, so too must the logical encoding of that system change. This database setting, as it has been called thus far, asks a series of entailment queries where the only thing that changes from one to the next is the complete theory C.

$$\begin{aligned} C_1 \cup I &\models \phi \\ C_2 \cup I &\models \phi \\ &\vdots \\ C_n \cup I &\models \phi \end{aligned}$$

In the hospital sensor net example, the whole point is to answer a fixed set of queries about an ever changing set of sensor readings. While the axiomatization shown in Chapter 2 is valuable pedagogically, a slightly different axiomatization allows us to amortize the cost of reformulation over multiple sensor readings. This axiomatization allows us to perform the reformulation exactly once and treat each sensor reading as simply a database update.

The current sensor readings are encoded as the binary predicate *signal* with a complete definition. If hospital had one patient *sam* and one piece of equipment, the oxygen tank *oxy*, the definition for the sensor readings at any point in time might be

$$signal(x, y) \quad \Leftrightarrow \quad \left(\begin{array}{l} (x = sam \wedge y = f) \vee \\ (x = oxy \wedge y = g) \end{array} \right)$$

Every signal says something about the location of a particular object: either the object is located in the same cell as the signal, or it is located in one of the surrounding cells.

$$signal(x, y) \quad \Rightarrow \quad \left(\begin{array}{l} location(x, y) \vee \\ \exists z.(adj(y, z) \wedge location(x, z)) \end{array} \right)$$

Every object is located in exactly one place.

$$location(x, y) \wedge location(x, z) \Rightarrow x = z$$

In addition, it is convenient to have a predicate $lifesupport$ that says which patient is dependent on which piece of equipment.

$$lifesupport(x, y) \Leftrightarrow (x = sam \wedge y = oxy)$$

Then with the usual definition of adjacency in place, the query asking whether there is some patient who is definitely separated from his equipment can be written as shown below.

$$\exists xy.(lifesupport(x, y) \wedge \forall zw.(location(x, z) \wedge location(y, w) \Rightarrow \neg adj(z, w)))$$

The important difference between this axiomatization and the original is that the sensor information in this version belongs entirely to the definition of *signal*, which is complete. Thus to update this theory to reflect a different set of sensor readings, all one must do is replace the definition for *signal*.

Recall that the algorithms presented in this chapter for constructing *poss* definitions ignored the complete portion of the theory. Using such an algorithm on this example would produce a database system with a definition for $poss_{location(x,y)}$ that works regardless of the definition for *signal*. For example, the algorithm might produce something like the definition is below.

$$poss_{location(x,y)}(x, y) \quad \Leftrightarrow \quad \left(\begin{array}{l} signal(x, y) \vee \\ \exists z.(signal(x, z) \wedge adj(z, y)) \end{array} \right)$$

This illustrates how, if written appropriately, one can amortize the cost of reformulation over many queries when it is natural to change the theory between queries. In this example, just the complete portion of the theory changed. In these settings, it is more likely that the complete portion of the theory is represented in a database system to begin with, and if the complete portion of the theory is large, e.g. the American Express transaction records over the past year, representing it with a database may be the only realistic option.

When part of the theory is already represented by a database, the setup for Extensional Reasoning is slightly different than the one considered throughout this thesis. Instead of starting with a set of logical sentences and a query about them, we start with a database system, a set of logical sentences, and a query. It turns out that the theory completion algorithms discussed in this thesis can be applied with only very minor modification to address this situation as well.

The settings outlined here illustrate opportunities to amortize the cost of reformulation. They differ based on whether the theory is static or dynamic, whether the query stream

utilizes the same *poss* predicates, whether it is acceptable to construct new *poss* predicate definitions at query-time, and whether part of the theory is represented as a database system at the start.

Chapter 5

Related Work

Work related to Extensional Reasoning covers a fairly broad range of topics. This chapter enumerates some of those topics and discusses how each relates to Extensional Reasoning.

The most directly related work automatically turns a set of logical axioms into a database or logic programming language. In [PAE98], a portion of Cyc [LG90] is automatically transformed into Prolog with the well-founded semantics [vGRS91]. In this work the translation process is lossy, and it is not known what logical consequences are preserved by the transformation. In this thesis, the preservation of logical consequence is a necessity, and the algorithms presented here reflect that.

Another paper [MG03] using databases for automated theorem proving, again in the context of Cyc, focuses on integrating heterogeneous database sources by mapping those sources into the Cyc ontology. Answering a query about the information contained jointly in the databases is accomplished using Cyc's inference engine. This work is close to the theory resolution approach in that the database is used to represent part of a theory. In Extensional Reasoning, on the other hand, the entire theory is reformulated into a database system.

5.1 The terms "extension" and "intension"

The terms "extension" and "intension" are used to differentiate what a term denotes and what it means. For example, "the morning star" means a star so bright it can be seen in the morning, and "the evening star" means the brightest star in the night sky. On planet Earth, "the morning star" and "the evening star" both happen to denote Venus, but on

another planet "the morning star" could denote something other than Venus; moreover, "the morning star" and "the evening star" could denote different things. That is, the intensions of the two terms are different; however, their extensions on planet Earth are the same. [Fit07]

In the database literature, the term "extensional" is used to refer to the base tables in a database system, and the term "intensional" is used to refer to the view definitions in a database system. The use of "intensional" reflects the fact that the predicates defined as views may have different extensions as the base tables change but always have the same meaning or definition. Just as the extension of "the morning star" may change as the planet on which it is evaluated changes, so too do the views' extensions change as the base tables change.

In set theory, the Axiom of Extension defines what is perhaps the most primitive concept in all of set theory: what it means for two sets to be equal. The axiom says that two sets are equal if and only if they have the same elements [Hal60]. This commitment to extensions occurs throughout mathematics: two mathematical objects are equal whenever their constituent pieces are equal.

Intensional Logic [Fit07] reifies both extensions and intensions. In short, it is a two-sorted first-order modal logic where one sort consists of extensions and the other sort consists of intensions. Each intension is a function from the states within a modal model to an object. This formalization embodies the idea that an intension represents a meaning whose denotation can change as the world changes. See Section 5.4 for a discussion of modal logic.

While intensional logic embodies within it the notions of intensions and extensions, Extensional Reasoning addresses those notions outside the logic. The algorithms that automatically construct a database system are aware of the tradeoffs between representing information with extensions, the base tables, and intensions, the view definitions. Extensional Reasoning is so-named simply because an entailment query is answered by answering a database query, which amounts to reasoning about a set of extensions.

5.2 Incomplete Databases

Virtually as soon as the relational database was invented, people began working on ways of representing incomplete information with a database. Chapter 19 of [AHV95] introduces several approaches, culminating in the notion of a conditional table. A row in a conditional

table is like a normal row in a database except that it may contain variables and is associated with a conjunction of equality constraints. Each constraint is of the form $x = y$, $x = a$, $x \neq y$, or $x \neq a$, where x and y are variables and a is a constant. Given any variable assignment, a tuple belongs to the model associated with that variable assignment whenever the constraints on the tuple are satisfied.

For example, the following two conditional tables, or c-tables for short, equivalently represent the three models that satisfy $p(a) \vee p(b)$: $\{p(a)\}$, $\{p(b)\}$, and $\{p(a), p(b)\}$.

a	$x = 1$
b	$x = 2$
a	$x = 3$
b	$x = 3$

x	$x \neq y$
y	

Figure 5.1: Two c-tables for representing the models of $p(a) \vee p(b)$

The c-table on the left allows variables to be assigned numbers, whereas the c-table on the right only allows the variables to be assigned a and b.

These c-tables are a compact way of describing what could be a potentially large number of models. In the context of Extensional Reasoning, they may turn out to be a useful target for compilation; that is, if we have a database that handles c-tables natively, we could investigate compiling a theory Δ into a series of c-tables and then use the native c-table support to determine whether all the models described by those c-tables satisfy ϕ. The important point here is that if the entailment query is written in logic, it is up to the system to transform the theory into c-tables; this is a common theme throughout the related work section. The question, "Couldn't this problem be solved using technique X?" is often answered in the affirmative, but if the problem is written in logic, then in order to use technique X, the problem must first be transformed into a language that technique X understands. Extensional Reasoning examines the automation of one particular transformation where the target is a run-of-the-mill relational database, one of the most industrially successful applications throughout all of computer science.

In addition to representational issues, incomplete databases bring into question what the correct answer to a query should be. Should the answer consist of all tuples that are possibly true (consistent with the database), or should the answer consist of all tuples that are necessarily true (entailed by the database)?

In Extensional Reasoning, this decision has been made because we have focused on

answering entailment queries; however, the theory completion approach requires algorithms for constructing definitions that enumerate all of the consistent instances of a particular sentence. Thus, the algorithms in this thesis allow the system to answer both types of questions. It is also noteworthy that the two types of queries are intimately connected in Extensional Reasoning: what is possibly true is used to compute what is necessarily true.

5.3 Nonmonotonic Reasoning

Theory-completion is much studied in the context of Nonmonotonic Reasoning. The premise is that a given set of logical sentences Δ is supposed to be reasoned about as if the only models that satisfied Δ were the minimal models that actually satisfy Δ. The better known approaches include the Closed-World Assumption [Rei78], Predicate Completion [Llo84], and Circumscription [McC88]. Each of these perform theory minimization, which changes the logical consequences of the original theory, whereas in Extensional Reasoning, theory completion is performed explicitly so that at least one logical consequence is preserved. Given an entailment query $\Delta \models \phi$, we construct a completed theory Δ' and a new query ϕ' in the vocabulary of Δ' so that

$$\Delta \models \phi$$
if and only if
$$\Delta' \models \phi'.$$

5.4 Modal Logic and Model Checking

The form of theory-completion studied in this thesis relies on predicates of the form $poss_\phi$. Such a predicate means that ϕ is possibly true, i.e. there is some model of the background theory Δ that satisfies ϕ. Possibility is one of the central topics of study in the area of Modal Logic, which actually refers to a family of logics, e.g. Deontic Logic and Temporal Logic. Here we briefly define the syntax and semantics of the pertinent propositional modal logic, following Chapter 1 of [BvBW06], and then explain how *poss* is related.

Syntactically, propositional modal logic is the same as propositional logic but with an extra logical symbol that represents the notion of possibility: $\Diamond$. A propositional constant is a sentence. If ϕ and ψ are sentences then so are $\neg\phi$, $\phi \wedge \psi$, and $\Diamond\phi$. The other standard propositional connectives are often admitted and defined in the usual way, e.g. $\Leftarrow$, $\vee$; likewise, $\Box$, which stands for necessity and is equivalent to $\neg\Diamond\neg$, is often included as a

second modal operator.

$\Diamond\phi$ means that ϕ is possibly true. To formalize this notion, satisfaction of a sentence in a given model in modal logic must give meaning to the notion of possibility. This is often done by conceptualizing a modal logic model as a set of connected propositional models: a Kripke model.

For a particular set of propositional constants P, a Kripke model is a three-tuple $\langle W, R, V\rangle$. W is the set of worlds. R is a subset of $W \times W$, which defines a directed graph over W; the utility of R will be explained shortly. V maps each propositional constant to the set of worlds in which it is true. Conceptually, a Kripke model is a directed graph where each node is a propositional model.

Given a Kripke model $M = \langle W, R, V\rangle$, each world $w \in W$ corresponds to a propositional model: it says which propositions are true in that world. In each such world, satisfaction for propositional logic is defined as usual. Satisfaction is also defined at each world for $\Diamond$, and this is where R is important. A sentence is possibly true at a particular world whenever the sentence is true in one of the worlds adjacent to that world in the graph defined by R.

- $\models_{M,w} p$ if and only if $w \in V(p)$
- $\models_{M,w} \neg\phi$ if and only if $\not\models_{M,w} \phi$
- $\models_{M,w} \phi \wedge \psi$ if and only if $\models_{M,w} \phi$ and $\models_{M,w} \psi$
- $\models_{M,w} \Diamond\phi$ if and only if for some $v \in W$ such that $R(w, v)$ holds, $\models_{M,v} \phi$.

Satisfiability can also be defined without reference to a particular world and is often called global satisfiability. A sentence ϕ is globally satisfied by model $M = \langle W, R, V\rangle$ exactly when for every world $w \in W$ it is the case that $\models_{M,w} \phi$. Entailment in modal logic is defined as usual: the premises Δ logically entail the conclusion ϕ whenever every Kripke model that satisfies Δ also satisfies ϕ.

First-order modal logic is more complicated because a first-order Kripke model represents a directed graph where each node is a first-order model. For this logic there are (at least) two competing semantics, which differ on how the universe of each of those first-order models is defined. The constant domain semantics asserts that the universe is the same for all first-order models, whereas the varying domain semantics allows the universe to differ from model to model. The conceptually simpler semantics is the constant domain semantics, which is the one we employ here.

Extensional Reasoning employs some of the same concepts as those employed by modal logic and can be viewed as a special case of modal logic model checking; however, this special case admits certain efficiencies if treated in its own right.

Recall that $poss_\phi$ is true exactly when there is some Herbrand model of Δ that satisfies ϕ. *poss* predicates relate to modal logic in at least two ways. (1) We can construct a Kripke model that satisfies $\Diamond\phi$ at a given world w exactly when $poss_\phi$ is true. (2) We can construct a Kripke model that globally satisfies $\Diamond\phi$ exactly when $poss_\phi$ is true.

The Kripke model in the first case satisfies $\Diamond\phi$ at world w exactly when $poss_\phi$ is true. The model corresponds to a directed graph where the nodes include all the Herbrand models of Δ and the given world w is connected to exactly the set of Herbrand models of Δ. $\Diamond\phi$ is satisfied at w exactly when there is some world v connected to w such that ϕ is true. Since the worlds connected to w are exactly the Herbrand models of Δ, we see that $\Diamond\phi$ is satisfied at w exactly when $poss_\phi$ is true.

The Kripke model in the second case globally satisfies $\Diamond\phi$ exactly when $poss_\phi$ is true. This means that $\Diamond\phi$ must be true at all worlds exactly when $poss_\phi$ is true. For this model, the set of worlds correspond exactly to the set of Herbrand models of Δ, and every world is connected to every other world. Thus, $\Diamond\phi$ is globally satisfied if for every world w there is some world connected to w that satisfies ϕ; because the graph is complete, this is just a redundant way of stating that there must be some world in which ϕ is true. Since the worlds correspond exactly to the Herbrand models of Δ, we again have an equivalence.

In the second Kripke model both the nodes and the edges of the graph are fixed. Studying such Kripke models is not without precedent. In the area of formal methods, model checking is used to verify properties of a dynamic system, which is often formalized as checking whether a given Kripke model satisfies a given sentence [JGP99]. In this setting, the Kripke model is represented with a description of the initial state of the system and a description of the transition function that says how the system changes over time. While this representation is quite natural, it is also vital because representing the associated Kripke model explicitly can require an enormous number of worlds. An explicit representation can be far too large to even communicate to a model checker; moreover, the implicit representation often contains structural information that can be leveraged to make model checking more efficient.

In Extensional Reasoning, the background theory Δ can be viewed as an implicit representation of the Kripke model described above. Both formal verification and entailment

in FHL can therefore be construed as checking whether an implicitly represented Kripke model satisfies a particular sentence. It is therefore tempting to claim the two problems are special cases and any algorithm for performing modal logic model checking would subsume the algorithms for solving those two problems. But because the Kripke models are represented implicitly, algorithms that utilize that representation natively have the potential to outperform algorithms that simply translate the implicit representation into an explicit one and use the usual modal model-checking algorithms.

For Extensional Reasoning in particular, the properties of *poss* described in Section 4.2.1 are reliant on properties of Δ. For example, $poss_{\phi \wedge \psi}$ is equivalent to $poss_{\phi} \wedge poss_{\psi}$ exactly when ϕ and ψ are independent with respect to Δ. By having Δ itself, properties like independence can sometimes be ascertained more quickly than would be possible by analyzing the corresponding Kripke model.[1]

In summary, the approach to Extensional Reasoning for incomplete theories investigated in this thesis shares certain concepts with modal logic, and Extensional Reasoning can be viewed as a special case of model checking in modal logic. However, because it is such a special case there are efficiencies to be gained by developing specialized algorithms for Extensional Reasoning.

5.5 Proof Theoretic Calculi

Proof theoretic calculi, one popular approach to automated deduction, especially for logics that allow a sentence set to be satisfied by infinitely many models, work by syntactically manipulating the premise set Δ, possibly together with the conclusion ϕ. This approach can be viewed as reformulating a set of sentences until the desired conclusion appears. On the contrary, Extensional Reasoning performs automated deduction by reasoning directly with models; however, it employs proof theoretic calculi to perform various kinds of reformulation. Conversely, some of the proof theoretic calculi rely on models in various ways and so are worth mentioning.

Injecting semantics into syntactic proof systems was probably first attempted by Gelernter in his Geometry Theorem-Proving Machine [Gel63]. Since then researchers have

[1] It is noteworthy that the two logical connectives over which *poss* distributes without condition are the same connectives that $\Diamond$ distributes over in modal logic (assuming constant domain semantics): $\Diamond \exists x.\phi(x)$ is equivalent to $\exists x.\Diamond \phi(x)$ and $\Diamond(\phi \vee \psi)$ is equivalent to $\Diamond\phi \vee \Diamond\psi$.

continued his efforts and produced various types of semantically-augmented proof techniques; model-guided proof systems [Rei73, Wan85, Sla93, PZ97, Cho00], and graphical reasoning methods [Bun73, Pas78, BPB95, AB96] are a few.

Both tableaux-style and resolution-style proof procedures incorporate models and semantics to varying degrees. Tableaux techniques, e.g. [MB88], search for a model of the premise set, and only when that search fails has the theorem been proven. Likewise, Model Evolution, a fairly new calculus [BT03] lifts DPLL [DP60, DLL62] to first-order logic, also proving unsatisfiability by failing to find a model of the axiom set.

Unlike the tableaux methods, resolution techniques search for a proof of unsatisfiability, and only when that search fails is the premise set deemed satisfiable. Sometimes these techniques use models to help guide the search for proofs. Semantic resolution[CL73] uses a model to construct and maintain the set of support. Hyperresolution [CL73] is a special case of semantic resolution where a particular model is chosen to partition the axiom set into the background theory and the set of support: the model that satisfies none of the ground atoms in the language. When the premise set is in the $\forall^*$ prefix class and is Horn, it so happens that hyperresolution builds a model that satisfies the premise set (Chapter 25 of [RV01]).

While some of these techniques can be seen as approaches to model-building, which is one way of looking at Extensional Reasoning, the techniques presented in this thesis differ from such model-building calculi most clearly in the case of an incomplete theory, where the first step is a structural reformulation of the problem. The output of that reformulation is a representation of a model that encodes the incompleteness in the original theory pertinent to the query at hand; for the case of incomplete theories, the model produced is not a model of the original theory. The calculi mentioned above can be used to help perform that reformulation step, but will not themselves construct a single model that encapsulates the incompleteness of the original theory.

In the case of a complete theory, one of the important steps from the computational perspective is constructing an implicitly defined model of the premise set that can be efficiently converted into a database language. Having an implicit definition of the model is important since materializing the model can sometimes be prohibitively expensive; thus, we need algorithms that are aware of the extensional/intensional tradeoff and can find a balance. This is the type of algorithm we investigate in Extensional Reasoning.

5.6 Characteristic Models

Model-based reasoning is an approach to automated reasoning that takes a set of logical sentences and constructs a set of models sufficient for answering questions about those sentences. It was developed both in the field of psychology under the name Mental Models [BG00] and in the field of automated reasoning under the name Characteristic Models [KKS93, KR94, Kha95, HI99]; we focus on work in the latter community, where the main motivation is computational efficiency. Coupling the construction of characteristic models with the checking of those models amounts to a direct implementation of the definition of entailment. To our knowledge, work in this area is limited to propositional logic where the number of characteristic models is always finite, and each model is also always finite in size; hence, the characteristic model approach is a decision algorithm, and the interesting part is computing the models efficiently.

In one sense, computing multiple models and reasoning about them individually is antithetical to Extensional Reasoning; on the other hand, computing multiple models and reasoning about them is what Extensional Reasoning does implicitly. In the case of incomplete theories, the goal of Extensional Reasoning is to construct a single model (database system) that represents all the information pertinent to the query at hand; at the same time that single model must encapsulate the information from multiple models. When the database query is evaluated, the system is implicitly constructing pieces of multiple models and checking to see whether they satisfy the properties necessary for the original query to be entailed. To what extent this tight coupling of model building and model checking produces computational benefit is the subject of future work.

5.7 Constraint Satisfaction

Constraint satisfaction plays a central role in Extensional Reasoning. A Constraint Satisfaction Problem (CSP) is often defined as a three-tuple $\langle V, D_V, C_V \rangle$: a set of variables, the domain of each variable, and the constraints on the variables. While there are options for how to represent the variables and the domains, the options for how to represent the constraints are most relevant for comparison to Extensional Reasoning. In the yearly CSP competition, each constraint is represented in one of three ways: (1) a permissible table, (2) a conflict table, or (3) a mathematical expression built out of functions with built-in semantics, e.g. $x > 2*(y+1)$. Each permissible table enumerates the combinations of values

that can be assigned to a given subset of the variables. Each conflict table enumerates the combinations of values that cannot be assigned to a given subset of the variables. Each mathematical expression allows the variables to be assigned all those values that make the expression true. The mathematical expression is an implicit definition of all the assignments that satisfy the constraint, whereas the permissible and conflict tables are explicit definitions of the allowed assignments.

Ignoring the mathematical expression representation of a constraint for the moment, every finite CSP where the constraints are expressed as permissible/conflict tables is a special case of the relational database query. To see this, suppose the CSP has variables $x_1, \ldots, x_k$, each of which has domain $d_1, \ldots, d_k$, respectively, where each domain is represented as an explicit table of values. Further suppose the permissible tables are $p_1(\overline{x_{i_1}}), \ldots, p_m(\overline{x_{i_m}})$ and the conflict tables are $n_1(\overline{x_{j_1}}), \ldots, n_s(\overline{x_{j_s}})$. That is, constraint p_l enumerates the permitted values for variables $\overline{x_{i_l}}$; likewise for the negative tables. Then, the CSP is a conjunctive database query over the domain, permissible, and conflict tables, all of which are represented as extensions.

$$d_1(x_1) \wedge \cdots \wedge d_k(x_k) \wedge p_1(\overline{x_{i_1}}) \wedge \cdots \wedge p_m(\overline{x_{i_m}}) \wedge \neg n_1(\overline{x_{j_1}}) \wedge \cdots \wedge \neg n_s(\overline{x_{j_s}})$$

For example, consider the map coloring problem: given a graph and a set of colors, paint all the nodes of the graph so that no two adjacent nodes are colored the same. One of the formulations of this problem studied in the CSP literature [McC82, PP80] uses permissible tables.

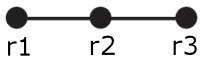

Figure 5.2: A 3-node graph

Suppose we have two colors $\{red, blue\}$ and a map with the three nodes $\{r_1, r_2, r_3\}$, connected as shown in Figure 5.2. Here we show that the CSP formulation is virtually identical to the database formulation. In the CSP formulation, there are three variables x, y, z, each of which represents the color assigned to region r_1, r_2, and r_3, respectively. Because regions r_1 and r_2 are connected, the permitted values for $\langle x, y\rangle$ are $\langle red, blue\rangle$ and $\langle blue, red\rangle$; the permitted values of $\langle y, z\rangle$ are the same as for $\langle x, y\rangle$ since they are also connected to one another. There are no constraints on $\langle x, z\rangle$. In the CSP world, we want

a satisfying assignment to variables $\langle x, y, z \rangle$. This CSP is depicted on the left-hand side of Figure 5.3. In the database formulation, shown on the right hand side of Figure 5.3, all the statements above are true but they have been implemented slightly differently. Instead of having two tables for the constraints on $\langle x, y \rangle$ and $\langle y, z \rangle$, there is one table named *next*. The variables and their domains are represented by the query $next(x, y) \wedge next(y, z)$. As one would expect, the two formulations give the same solutions: $\langle red, blue, red \rangle$ and $\langle blue, red, blue \rangle$.

CSP

Vars	Domains
x	$\{red, blue\}$
y	$\{red, blue\}$
z	$\{red, blue\}$

Constraints

x	y
red	blue
blue	red

y	z
red	blue
blue	red

DATABASE

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

Table: *next*

red	blue
blue	red

Figure 5.3: The CSP and database formulations of the map coloring problem

The claim here is just that the input format for a CSP solver (modulo mathematical expressions) is a special case of the input format for a relational database. However, it is not the case that the algorithms used by database query engines subsume the algorithms used by CSP solvers; the two classes of algorithms work on different classes of problems. The database algorithms were developed to handle the case of very large amounts of data, so large that the dominant cost is swapping data to and from secondary storage. CSP algorithms on the other hand were developed for solving problems where the constraints are quite hard to satisfy, and the problems fit into main memory. Just as Extensional Reasoning translates FHL into database systems, one could develop translation techniques from FHL to CSPs (which could be based on the techniques for Extensional Reasoning). Likely there would be some problems more amenable to solving with a CSP solver than with a database engine, but it is equally likely that the converse would also be true: some problems would be better solved using a database engine than with a CSP solver.

Leaving the comparison between CSPs and relational databases, constraint satisfaction is important in Extensional Reasoning for a different reason. Another way of representing constraints in a CSP, besides the permissible and conflict tables, is to use a set of logical sentences and the usual model-theoretic definition of satisfaction. A set of logical sentences

is satisfied by some set of models, and so one can say that an assignment of values to the variables of interest satisfies the constraints exactly when that assignment is satisfied by some model of the logical sentences.

Back to the map coloring problem, the logical axioms representing the constraints are as follows.

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

The variables and domains are represented implicitly with the statement $color(r_1, x) \wedge color(r_2, y) \wedge color(r_3, z)$. In this form, a solution to the constraint satisfaction problem is an assignment of the variables x, y, and z to values v_1, v_2, and v_3, respectively, such that $color(r_1, v_1) \wedge color(r_2, v_2) \wedge color(r_3, v_3)$ is consistent with the two axioms above.

Extensional Reasoning deals explicitly with these two representations of constraint satisfaction problems. In fact, the form of theory completion introduced in Section 4.2 essentially translates one of the representations into the other. Recall that one of the central issues in theory completion is constructing a complete definition for a predicate $poss_{\phi(\overline{x})}$ that is true of all those instances of $\phi(\overline{x})$ that are consistent with the background theory. If we think of the definition of $poss_{\phi(\overline{x})}$ as the table of values such that $\phi(\overline{x})$ is consistent with the original theory, then we see that an algorithm for constructing *poss* definitions transforms a constraint satisfaction problem where the constraints are represented as logical axioms into one where the constraints are represented as (implicitly defined) permissible tables.

In the map coloring example, the constraint satisfaction problem above expressed as logical axioms is translated into an implicitly defined permissible table when constructing a definition for $poss_{color(r_1,x) \wedge color(r_2,y) \wedge color(r_3,z)}$.

$$poss_{color(r_1,x) \wedge color(r_2,y) \wedge color(r_3,z)}(x, y, z)$$
$$\Leftrightarrow$$
$$\left(\begin{array}{l} region(r_1) \wedge hue(x) \wedge (adj(r_1, r_2) \Rightarrow x \neq y) \wedge \\ region(r_2) \wedge hue(y) \wedge (adj(r_2, r_3) \Rightarrow y \neq z) \wedge \\ region(r_3) \wedge hue(z) \wedge (adj(r_1, r_3) \Rightarrow x \neq z) \end{array} \right)$$

If each *poss* predicate is viewed as a CSP, then the theory-completion approach can be viewed as a method for constructing a series of CSPs and an expression relating their solutions that is true exactly when the original entailment query holds. Or equivalently, one could view theory completion as a technique for rewriting a logical entailment query as

a single CSP whose constraints are permissible tables that are represented both implicitly and explicitly. Leveraging these connections to CSPs is the subject of future work.

5.8 Finite Model Building

Finite Model Building is often used as a way to show that a premise set Δ does not entail a conclusion ϕ. If $\neg\phi$ together with Δ has a model, then we know that $\Delta \not\models \phi$; otherwise, $\Delta \models \phi$. Model builders have been used successfully in tandem with first-order theorem provers, enabling a theorem prover's search to halt once a model has been found; they can also be used for certain classes of theories to show entailment holds by demonstrating that there can be no satisfying model.

Here we briefly describe four approaches to finite model building for first-order logic, three of which are in active use today in the yearly theorem proving competition [SS98a, SS98b]. The so-called Mace-style approach [McC01, CS03, Tam01] works by converting the given clauses into propositional logic and handing them off to a SAT-solver. The SEM/Falcon-style approach [Zha96, ZZ95, McC03, JJD98, ABH00, Tam01] builds a model directly via traditional search techniques, often pruning the search by manipulating the given sentences to take the consequences of the partially built model into account. The Darwin-style approach [BFNT07] is similar to the Mace-style approach in that it relies on a decision procedure for a less expressive logic: quantifier-free, function-free first-order logic. The Darwin approach reduces a given set of sentences into this fragment, and then decides satisfiability using the decision procedure. Finally, it turns out that two proof-theoretic calculi for first-order logic, ordered resolution and hyperresolution, can be construed as a proof-theoretic approach to building a model in the case of function-free first-order logic [Pel00, RV01]. These refutation-style resolution-based calculi will either produce the empty clause or will produce a saturated clause set from which one can construct a model.

These approaches differ from Extensional Reasoning in qualitative ways. In the case of a complete theory, Extensional Reasoning techniques endeavor to construct a database that represents a model, where one of the primary decisions is which portion of the model to represent extensionally. The model building approaches mentioned above do not attempt to leverage this tradeoff of intensional representation and extensional representation.

In the case of incomplete theories, Extensional Reasoning answers an entailment query $\Delta \models \phi$ by constructing a definition for $\neg poss_{\neg\phi}$ and checking whether the definition entails

$\neg poss_{\neg\phi}$. Recall that $poss_\phi$ is true if and only if ϕ is consistent with Δ. Thus, Extensional Reasoning answers entailment queries the same way model builders do: it tries to show the existence of a model that satisfies Δ and $\neg\phi$. Failure indicates that the entailment holds. The real comparison therefore amounts to model building and the construction/evaluation of *poss* predicates.

The construction and evaluation of *poss* predicates differ from model building for at least two reasons. First, instead of constructing an actual model that satisfies ϕ and Δ, in Extensional Reasoning, we must only construct a database query that when evaluated returns true if there exists some model for ϕ and Δ. Constructing the actual model itself is unnecessary. Second, even if the database query ends up constructing a model in order to guarantee that one exists, the Extensional Reasoning approach takes place in two phases: first the database query is constructed and second that query is evaluated. This separation of the work into two phases is analogous to first looking at the sentences for which we want a model and subsequently writing a logic program that searches for one. The approaches to model building used for Extensional Reasoning are therefore quite different from the model-building approaches described above, though they take virtually the same approach to answering deductive queries: demonstrating that there is no model that satisfies the theory and the negation of the query.

5.9 Logic Programming

Logic Programming languages have a variety of semantics. *nr-datalog*$^\neg$, because it is nonrecursive and therefore stratified, lies in the intersection of the most popular semantics today: stratified [Ull89], disjunctive [LMR92], stable model [GL88], and well-founded [vGRS91]. The fundamental difference between stratified semantics and the other three is that the stratified semantics guarantee a complete theory, whereas the other three sometimes produce an incomplete theory: stable model semantics allow a theory to be satisfied by multiple models; well-founded semantics use three-valued models; disjunctive semantics use models that consist of ground disjunctions. In each of the three, it can be the case that neither a sentence nor its negation is entailed. While the stratified semantics guarantee that a theory is satisfied by a single model, it can only be applied if the logic program is written so that negation is stratified. Hence, while the stratified semantics guarantee a complete theory, the other semantics are uniformly applicable.

One might wonder whether translating an entailment query into a logic programming language that admits incomplete theories might be easier than translating into one that only admits complete theories. Perhaps it is, which would allow implementations of those languages to answer entailment queries. For example, answer set programming (ASP) [Lif02], a popular approach to logic programming today, could be used to answer entailment queries just like Extensional Reasoning uses database engines once the problem has been translated into the language an ASP system accepts. It is such a translation that is studied in this thesis. In regard to answer set programming, it is noteworthy that an answer set corresponds in some way to all the instances of a sentence that are consistent with the background theory, i.e. the tuples true of a particular *poss* predicate. Whether that connection can be leveraged in some way is the subject of future work.

5.10 Semantic Web: Language Integration

The Semantic Web community has paid a significant amount of attention to description logics (fragments of first-order logic) and their relationships to logic programming languages. The work we are aware of in this area that is related to Extensional Reasoning is concerned either with finding fragments of a description logic that can be reasoned about with a logic programming engine and vice versa [GHVD03, HV03, HHK$^+$05, KHSV05] or with developing a new logic that integrates logic programming languages and description logics [FT04, MR07].

Extensional Reasoning takes an extreme perspective on the first body of work mentioned in that the results show how to convert an arbitrary sentence set in FHL into a logic programming language that is guaranteed to have a single minimal model. The work mentioned above either translates only a fragment of the description logic into logic programming [GHVD03, HHK$^+$05, KHSV05], or it targets a logic programming language that has multiple models [HV03].

The latter work mentioned above considers the situation where the set of logical sentences one wants to reason about consists of sentences of two sorts: those that are to be interpreted using logic programming semantics and those that are to be interpreted using description logic semantics. The challenge is in defining a semantics that incorporates both the logic programming and description semantics. This hybrid approach can also be taken while employing Extensional Reasoning.

If in addition to a set of sentences Δ in FHL we were given a *nr-datalog*$^\neg$ system Γ, the algorithms presented in this thesis can be applied to translate $\Delta \cup \Gamma$ into *nr-datalog*$^\neg$, where we assume that the semantics of $\Delta \cup \Gamma$ is defined so that Δ is a conservative extension of the *nr-datalog*$^\neg$ semantics of Γ. The key insight here is that Γ contains complete definitions for some set of predicates P. We can therefore think of Δ as having been written in a logic FHL$'$, which includes complete definitions for $=$ as well as the predicates P. The algorithms TO-BICONDS and TO-BICONDS-MAX both take as arguments the set of predicates with complete definitions, and hence can be given $P \cup \{=\}$ instead of just $\{=\}$. Also, the algorithms for theory completion never touch the complete portion of the theory, which means they will not be bothered by the fact that some of the complete predicates were defined in *nr-datalog*$^\neg$ to start. Thus, in this larger framework Extensional Reasoning is a method of compiling a set of *nr-datalog*$^\neg$ and FHL sentences into *nr-datalog*$^\neg$.

5.11 Caching

Caching is often discussed in the context of the model-elimination proof procedure for first-order logic [AS92, Ged95] as well as logic programming [SSW94]. Since Extensional Reasoning deals with building models in classical logic, we will concentrate on model-elimination.

Caching amounts to storing the results of intermediate proof successes and failures, which allows a system to avoid repeated work. In the case of Horn clauses, caching successful intermediate proof attempts is an iterative approach to model building because each success amounts to a proof that some atom is entailed by the premise set. In the NonHorn case, caching for model elimination differs because intermediate proof successes are dependent on ancestor literals [AS92, Ged95], which means that a proof success does not guarantee a particular atom is entailed.

For example, consider the following propositional rule set.

$$p \Leftarrow q$$
$$q \Leftarrow \neg p$$

A model elimination proof for p is shown in Table 5.1, where each line but the first is the result of applying either an extension or a reduction operation to the line before. It turns out that this proof suffices to demonstrate that the original sentences entail p. Notice that there is an intermediate proof attempt for q in the context of $[p]$ that succeeds; however, q

p	Premise
[p] q	Extension with $p \Leftarrow q$
[p] [q] ¬p	Extension with $q \Leftarrow \neg p$
[p] [q] [¬p]	Reduction with [p]
[p] [q]	Drop ancestor literal
[p]	Drop ancestor literal
	Drop ancestor literal

Table 5.1: Model elimination proof trace

is not entailed by the premise set, which can be seen more easily when the rules are written as disjunctions.

$$p \vee \neg q$$
$$q \vee p$$

Caching positive results in the NonHorn case requires storing the context in which a proof attempt succeeds, and as this example demonstrates, the theory may say nothing at all about a literal involved in an intermediate proof success.

This distinction between Horn and NonHorn is important because while caching in the Horn case amounts to incremental model-building, the same cannot be said for the NonHorn case. Moreover, even in the Horn case, no model will ever be fully built unless the theory is complete.

Extensional Reasoning builds a single model of the premises regardless of whether the sentences are Horn or NonHorn and regardless of whether the theory is complete or incomplete. In addition, models built by Extensional Reasoning algorithms may be represented extensionally, as in caching, as well as intensionally. One of the advantages of caching over Extensional Reasoning is that caching will never compute a part of a model that it does not need to compute, whereas Extensional Reasoning has no such guarantee.

5.12 Boolean Satisfiability

Boolean satisfiability has become a common method for representing and reasoning about problems, especially in the area of formal methods [PBG05]. Given a set of sentences in propositional logic, find a model that satisfies those sentences if there is one. Even though this is the classic NP-complete problem [Sip96], fairly recent advances in SAT-solving technology have made the approach increasingly popular.

In the context of this thesis, SAT solvers constitute competition for Extensional Reasoning techniques because Finite Herbrand Logic can always be converted into propositional logic. The reason this works is that FHL has a known, finite domain. Any sentence of the form $\forall x.\phi(x)$ is logically equivalent to $\phi(a_1) \wedge \cdots \wedge \phi(a_n)$, where $\{a_1, \ldots, a_n\}$ is the universe. Likewise, $\exists x.\phi(x)$ is logically equivalent to $\phi(a_1) \vee \cdots \vee \phi(a_n)$. Thus every entailment query $\Delta \models \phi$ can be translated into a SAT problem by simply applying these transformations to $\Delta \cup \{\neg\phi\}$.

SAT-solving technology is undoubtedly very fast, partially because the most primitive operation in DPLL-style approaches [DP60, DLL62] amounts to flipping a single bit. In comparison, the most primitive operation when evaluating a database query is either unification for top-down approaches or matching for bottom-up approaches [Ull89].

One of the drawbacks to solving FHL entailment problems with a SAT solver is the loss of structural information that goes along with translating an FHL problem into propositional logic and then translating propositional logic into conjunctive normal form (CNF), as is required by many of today's SAT solvers. The SAT community has given attention to the loss of structural information, e.g. building SAT solvers that operate on non-CNF formulae [AG93, GS99, TBW04, MS06].

Another drawback to the SAT approach is that the grounding procedure itself can be very expensive–exponential in the size of the universe, though work is being done to combat this cost [Sch02, RA05, GA07]. A related drawback is that the grounding of the sentences depends on the universe, and when the universe changes, so does the grounding. This reduces the opportunities for amortizing the cost of the grounding.

While it is unclear at this point exactly how Extensional Reasoning and SAT techniques compare, the hope is that the properties of databases will enable Extensional Reasoning to outperform SAT solvers for some interesting classes of problems. First, the relational structure of the database can be used to preserve some of the relational structure inherent in the original FHL. Second, that relational structure also helps keep the cost of transforming an entailment query into a database relatively low as sometimes the transformation can be performed independent of the universe. Third, databases are well-equipped to handle problems that are too large to fit into memory, so much so in fact that sometimes the cost of answering a database query is measured in the number of swaps made from disk to memory. Finally, database semantics justify the use of Negation as Failure, which as shown in Section 3.4, can produce large computational savings.

5.13 Knowledge Compilation

Knowledge Compilation endeavors to reduce the complexity of query-answering by building a structure that is either logically equivalent but in a more vivid [Lev86, EBBK89, Dav94] form or not logically equivalent but a good approximation. The vast majority of work in this area is for propositional logic. The work of [KS91, SK91, KS92, HK93] uses two approximating Horn theories: the least upper bound (LUB) and the greatest lower bound (GLB) in terms of models. Thus the LUB has more models than the original theory, which means it is a weaker theory. The GLB has fewer models than the original, which makes it a stronger theory. Those queries entailed by the weaker theory (LUB) are entailed by the original theory, and those queries not entailed by stronger theory (GLB) are not entailed by the original. Computing the LUB and GLB allows one to approximate a theory and in some cases answer queries very quickly because algorithms exist for efficiently answering questions about Horn theories. [KKS95] goes on to generalize theory approximation with Horn theories to theory approximation with arbitrary languages, still in propositional logic. For a survey of other propositional techniques, see [CD97]. The only first-order approximation technique we are aware of is [dV96], which follows [KKS95]. It lifts an algorithm for computing the LUB in an arbitrary language to first-order under some mild constraints. Together all this work brought about a theoretical framework defining complexity classes and reduction techniques for problems that lend themselves to compilation [CDLS00]. This work assumes that one can amortize the cost of reformulation over many queries, and ignores the cost of reformulation altogether.

First, unlike the form of Knowledge Compilation described here, in Extensional Reasoning, we are concerned both with the cost of reformulation and the cost of evaluation. Second, part of the power of Extensional Reasoning is derived from dealing with a relational, instead of propositional, syntax. The fact that we can compile out extensional tables means that database algorithms can be used to effectively index that entire portion of the theory. Third, in the case of incomplete theories, Extensional Reasoning relies on a form of theory completion, where in a certain sense logical equivalence is preserved. This differs from the Knowledge Compilation above because (1) the compiled version is an approximation of the original theory and (2) the compiled version is not necessarily a complete theory. Extensional Reasoning reduces queries to queries over complete theories in all cases.

5.14 Inductionless Induction (Proof by Consistency)

Inductionless induction is a form of inductive theorem proving that employs first-order proof techniques. A conclusion is said to be an inductive consequence of a premise set exactly when the conclusion is true in all the Herbrand models that satisfy the premises. We write $\Delta \models_H \phi$ to mean that every one of the Herbrand models that satisfies Δ also satisfies ϕ. These Herbrand Semantics are discussed in detail in [HG06]. Inductionless induction uses traditional first-order proof techniques for computing entailment under Herbrand Semantics.

Recall that a resolution-style theorem prover can be viewed as a system that takes a set of clauses as input and determines whether those clauses are satisfied by a Herbrand model. If we want to know whether $\Delta \models_H \phi$, we can ask whether $\Delta \cup \{\neg\phi\}$ is satisfied by any Herbrand model and then negate the answer. If $\Delta \cup \{\neg\phi\}$ is in the $\forall^*$ prefix class, i.e. all the sentences when written in prenex form are quantified only by $\forall$, then any resolution-style theorem prover can be used to compute inductive consequences. It turns out that for this class of queries the inductive consequences are the same as the traditional first-order consequences.

Inductionless induction is targeted at a subclass of the entailment queries mentioned above: Δ is a set of Horn clauses and ϕ is an atomic query. In this case, it is well-known that Δ has a single, minimal Herbrand model, and $\Delta \models_H \phi$ if and only if that minimal Herbrand model satisfies ϕ. To determine whether $\Delta \models_H \phi$, we could use resolution to determine whether $\Delta \cup \{\neg\phi\}$ is unsatisfiable, just like always; however, because Δ is Horn and ϕ is atomic, we can also use resolution-style techniques to answer the query a different way. Suppose A is a clause set that is only satisfied by the minimal Herbrand model of Δ. Then, we know that $\Delta \models_H \phi$ if and only if $\Delta \cup A \models_H \phi$, which holds exactly when $\Delta \cup A \cup \{\phi\}$ has a Herbrand model. Since resolution checks whether a given clause set has any Herbrand model, if resolution applied to $\Delta \cup A \cup \{\phi\}$ fails to prove the empty clause, we are guaranteed that the minimal model of Δ satisfies ϕ and therefore that $\Delta \models_H \phi$. This technique is called Inductionless Induction, or more appropriately Proof by Consistency.

The idea behind Proof by Consistency is related to Extensional Reasoning (ER) in the case of a complete theory because ER constructs (a representation of) the model that satisfies the theory and concludes entailment based on consistency with that theory. For the case of an incomplete theory, there are also parallels. Once we have split a theory into $C \cup I$, in ER we compute a set of definitions for *poss* predicates that when joined with C

is satisfied by a single model; we then use that model to determine whether the conclusion holds. The analogy in Proof by Consistency is when the clauses A are added to the theory Δ to ensure their union is satisfied by a single Herbrand model. In addition, both Proof by Consistency and Extensional Reasoning target logics that use Herbrand Semantics.

Extensional Reasoning differs from Proof by Consistency (PBC) in several ways. First, ER considers a logic with only finite models, whereas PBC considers a logic with infinite models as well. But PBC is not a strict generalization of ER since the results there only cover premise sets that are Horn and quantifier-free. In ER, we can handle a more general class of premise sets, which is not surprising given that ER focuses on a decidable logic, whereas PBC focuses on an undecidable logic. Second, while the set of *poss* definitions in ER is analogous to the set of clauses A in PBC, the two are quite different. *poss* definitions capture the incompleteness in a theory, which in some sense maximizes the theory, whereas A shrinks the consequences of the theory in question. Further analysis is the subject of future work.

5.15 Dividing and Conquering Logic

Among other things, Eyal Amir's Ph.D. thesis [Ami01] considers the problem of automatically decomposing logical theories and then effectively reasoning about them. The algorithms he considers for automatic decomposition do not assume a calculus and therefore do not partition a theory based on performance, as different calculi perform very differently on the same set of axioms. Rather those algorithms compute (an approximation to) the optimal partitioning based on a dependency graph of the axioms. A good partitioning for a propositional theory minimizes the following: (1) the number of symbols shared between partitions, (2) the number of symbols in a partition, and (3) the number of partitions. That is, the quality of a decomposition in this work is defined by the syntactic character of the axioms. The thesis mentions semantic decompositions in related work, e.g. semantic resolution.

Extensional Reasoning also decomposes theories but does so using a semantic criterion: completeness; it so happens that this property can be approximated by syntactic characteristics of an axiom set, and such approximations have the benefit of more efficiently computing a decomposition. Secondly, the decomposition algorithms introduced were built to target one particular calculus, namely *nr-datalog*$^{\neg}$. Consequently, the criteria for success

is how efficiently the resulting *nr-datalog*$^{\neg}$ can be processed.

5.16 Answering Queries Using Views

In the database literature, there has been a significant amount of attention paid to the problem of answering queries with views [Hal01, BPT97, ACN00, Chi02]. There are two focuses for this work: (1) sometimes views are materialized and queries can be answered more quickly using those views because part of the query has already been computed and (2) in data integration, where databases in disparate schemas are queried through a single, mediated schema, answering queries requires rewriting a query in that mediated schema in terms of the actual schemas–those schemas are written as views of the mediated schema. The first topic is much more relevant than the second, which is why we focus on it.

The work on materializing views is of two types. Both are given as input a query stream, a set of extensional tables, and a set of view definitions. The first type of work consists of algorithms for automatically choosing the subset of the given view definitions to materialize in order to optimally answer the query stream [BPT97, ACN00]. The second type of work ignores the given view definitions and instead generates the optimal viewset for answering the query stream under a space constraint [Chi02].

All this work undoubtedly informs but does not entirely address one of the questions left unanswered in this thesis (Section 3.2): given a set of nonrecursive biconditionals, which predicates should be turned into extensional tables and which predicates should be turned into intensional tables?

It is unclear at this point how to directly apply the database results to solve this problem. The second body of work ignores the cost of materializing the views, which is not always appropriate, and requires time that is doubly exponential in the size of the query stream to compute the optimal viewset. The first body of work cannot be directly imported because it requires as input a set of extensional tables, a set of view definitions, and a query load, not a set of nonrecursive biconditional definitions. We can certainly transform the biconditionals into the required form, for instance by materializing all those predicates defined entirely in terms of equality. This approach, however, may rob an algorithm from employing the structure of the data within those materialized tables that would have been available had we not materialized.

For example, the following biconditional definition conveys a great deal of information

about p.

$$\begin{aligned} p(x,y,z) \quad \Leftrightarrow \quad & (x = a \vee x = b \vee x = c) \wedge \\ & (y = b \vee y = c \vee y = d) \wedge \\ & (z = c \vee z = d \vee z = e) \end{aligned}$$

But once p is materialized, exploiting that structure is more expensive because it must be recomputed. Moreover, this definition has nine literals whereas the materialized version has $27 * 3 = 81$ literals, exponential in the length of the definition.

Another option would be to materialize none of the definitions, and create a single extensional table representing $=$. This is also problematic, however, because unless the algorithms in (1) and (2) realize that the $=$ table actually represents equality, it may assume that the table could contain arbitrary data. That is, it will not take into account the fact that $=$ has a built-in semantics. Either way, it seems that information is lost, which may lead to suboptimal results.

5.17 Partial Deduction

Partial deduction, as described in [LB02], is partial evaluation applied to logic programming. Partial deduction specializes a set of rules based on a set of pre-defined queries. It differs from partial evaluation techniques in imperative and functional languages because arguments with variables are still amenable to evaluation. The operation central to partial deduction is that of unfolding, i.e. applying SLD resolution some number of times to the query in question to produce a partial SLD proof tree. The leaves of that tree imply the query, and the leaves of all such trees can sometimes be used to produce an equivalent rewriting of the query. Without negation, such a rewriting is equivalent under the typical minimal-model semantics of logic programming. With negation however, differences arise between the well-founded and the procedural Prolog semantics. According to [LB02], the work on partial deduction is broken into two components: local control and global control; the global control is only necessary if one is concerned with the Prolog procedural semantics. Since we are interested in declarative semantics, stratified to be precise, our only concern is with the work relating to local control. Local control refers to choosing how much a partial set of rules should be unfolded, i.e. when should the partial SLD tree be expanded?

The process of unfolding, i.e. constructing the tree, is difficult because applying too many resolutions can result in local code explosion, work duplication, and non-termination. If the unfold function is guaranteed to produce a tree with at most one non-failing branch, it is said to be determinate; determine unfolding will not produce code explosion or work duplication. It is undecidable to determine whether a tree with at most one non-failing branch can be constructed. Approximations are often used [LGBH04]. To deal with termination, binding-time analysis [CGLH04] is used (sometimes computed using abstract interpretation), which creates annotations that have decided whether to unfold or not. Instead of handling termination offline, another approach handles it online–during specialisation. The online version continues to unfold as long as the result is smaller, according to a well-founded or well-quasi ordering, than all its ancestors of the same relation [PAH04].

This unfolding process is, except for negation, almost identical to the proof-tree completion approach to computing *poss* definitions. The only difference is that in proof-tree completion, the predicates that are assumable are known ahead of time. While undecidability makes unfolding hard to control when dealing with logic programs, the finite domain in Finite Herbrand Logic guarantees that the unfolding process can always be made finite. Thus, while the approaches used in the logic programming community to perform unrolling will inform proof-tree completion approaches, the central issue in the proof-tree completion is efficiency, not termination, as in the LP work.

5.18 Gödel's Incompleteness Theorem

Gödel's Incompleteness Theorem implies a tradeoff of expressiveness and computability, phrased in the context of incomplete theories. The idea here is that if a particular logic has a complete proof system, i.e. there is a procedure $\vdash$ such that if $\Delta \models \phi$ then $\Delta \vdash \phi$, then entailment in that logic is at most semi-decidable. Consequently, in such a logic one can never axiomatize problems that are not semi-decidable, i.e. co-r.e. The theory of integer arithmetic, which is comprised of the integers, an ordering over the integers, multiplication, and addition, is expressive enough to encode problems that are co-r.e. Hence, no logic with a complete proof system can axiomatize any theory as expressive as the theory of integer arithmetic. Said another way, in a logic with a complete proof system, every set of axioms satisfied by the model that represents integer arithmetic must be incomplete, i.e. there is some sentence in the language such that neither it nor its negation is entailed.

The only real relevance of Gödel's Incompleteness theorem to Extensional Reasoning is that both are concerned with incomplete theories. Looking toward the future, however, Gödel's result serves as evidence of how pervasive incomplete theories are in logics that are more expressive than FHL.

Chapter 6

Conclusions and Future Work

Extensional Reasoning is a promising approach to automated deduction in the case of finite theories, and its foundations have been investigated in this thesis. It involves automatically translating a logical entailment query written in a classical logic into a *nr-datalog*$^\neg$ query so that the answers to the two queries are the same. That translation is easiest when the premise set in question is axiomatically complete, but when the premise set is incomplete, the translation is also possible if we leverage the notion of satisfiability to first complete the premise set. One of the key insights into reducing the cost of transformation is the separation of the premise set into two components, the complete portion C and the incomplete portion I. The reformulation algorithms work by manipulating only the incomplete portion, and when the complete portion is far larger than the incomplete portion, the cost of that reformulation can be negligible. One algorithm was presented for performing Extensional Reasoning for arbitrary entailment queries, but that algorithm is doubly-exponential in the best case. Another algorithm was presented that handles the restricted class of entailment queries shown below, but requires linear time and space in the best case.

$$C \cup \forall^* I \models \forall^*.\phi$$

We expect Extensional Reasoning to be powerful for theories with large amounts of data and a small number of axioms. Large knowledge bases, e.g. [LG90] and [NP01], often have a large amount of instance data and a relatively small number of simple axioms. Answering questions about such theories amounts to searching through a space that has a very high branching factor with a relatively small height, analogous to finding a needle in a

haystack. Assuming the semantic web continues to evolve, this class of entailment queries will require more attention than it is currently being given in the automated theorem proving community.

The implications of Extensional Reasoning differ based on one's viewpoint. From the theorem proving community's perspective, Extensional Reasoning is exciting because it enables automatic theorem provers to handle a very popular class of queries efficiently (those that can be phrased as database queries) without sacrificing generality. From the database community's perspective, Extensional Reasoning is exciting because it allows a standard relational database to be queried using the full flexibility and modularity of classical logic. From the logic programming community's perspective, Extensional Reasoning is exciting because it is a very restricted form of logic program synthesis where the specification is expressed in classical logic. From a holistic perspective, knowing how classical logic and database systems relate furthers our understanding of what features of logic are true sources of power that can and should be leveraged when possible.

There is much work that remains, both in the short term and in the long term. In the short term, there are five problems that, if solved, have the potential to substantially improve the performance of the current algorithms for Extensional Reasoning.

Besides enlarging the class of complete theories that we can detect automatically, 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 $p(\overline{x}) \Leftrightarrow \phi(\overline{x})$? Independent of Extensional Reasoning, we investigated the model theory for one particular metalevel logic [HG05] and made preliminary investigations into meta-resolution, a variant of resolution that targets metalevel logic. The hope is that those results will enable us to address this metalevel proof problem.

Second, the algorithm illustrated in Chapter 3 for converting a set of nonrecursive biconditional definitions into *nr-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, and append it to the body of every rule for every unsafe variable. Minimizing the cases where *univ* is used can dramatically reduce 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. As mentioned in Chapter 5, 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.

Fourth, the two main algorithms presented for constructing *poss* definitions rely on computing the resolution closure of the incomplete portion of the theory. Sometimes this closure would be infinite if it were not for the finite domain, and it is my belief that if the closure is naturally infinite, bounding it with a finite universe will not result in a more efficient method for proving theorems. Moreover, those algorithms do not produce the same definitions produced by hand, leading to the conclusion that they are not taking advantage of the same structure that people do when performing the transformation. Constructing new and better algorithms for constructing *poss* definitions will enable Extensional Reasoning to be effectively applied in a broader range of settings.

Fifth, some problems are most naturally expressed using function constants, even when the domain is finite. One might investigate whether function constants could be admitted into FHL by translating any functional term of arity n into a relation of arity $n+1$. Even if this approach does not work in general, it may work well enough to handle the common cases, enhancing the claim that Extensional Reasoning allows one to encode problems in a natural manner and automatically translate them into a representation that can be manipulated efficiently.

Each of the first four problems is worthy of study in its own right, and the fifth problem is a common concern in automated reasoning. The first is a strict generalization of the traditional automated theorem proving problem that would provide a query language expressive enough to handle many seemingly second-order questions. The second asks a fundamental question about the tradeoff of generation and validation; often it is less efficient to generate answers and then validate them than it is to tightly couple the generation and validation, but what algorithms will perform such a rewriting automatically? The third is a much-studied problem in the database literature, though in our setting there may be significantly more structure than is usually available. The fourth problem attempts to mechanize something human beings do quite regularly—converting a problem about consistency into one about deduction; if completely solved it might allow novices an enhanced ability to write problems down naturally while allowing our software systems to solve those problems

efficiently.

In the long term, three generalizations of the work reported in this thesis are worthy of investigation. Each deals with expanding the logic taken as input by the system. The most obvious generalization relaxes the finite domain assumption, to produce first-order semantics [End00], Finite Model Theory semantics [EF99], or possibly Herbrand semantics [HG06]. In all three, entailment is undecidable, which means that in order to allow for the possibility that the reformulation is decidable, the target must be a language expressive enough to encode undecidable problems. A full logic programming language such as Prolog [AK91] is the natural extension of *nr-datalog*$^\neg$ with the necessary expressiveness. Thus, one could consider translating first-order logic into Prolog, which just like *nr-datalog*$^\neg$ uses Negation as Failure, one of the features that produced sizable speedups for Extensional Reasoning.

The second generalization examines whether completeness with respect to some fragment of the universe can produce computational benefit. The work presented in this thesis examines theories that are complete with respect to some set of predicates: each predicate is complete for all elements in the universe. One can detect such completeness by reformulating part of a theory into a nonrecursive set of biconditionals. The extension suggested here examines theories that are partially complete for some set of predicates: each predicate is complete for some subset of the elements in the universe. One might try to detect such completeness by reformulating part of a theory into a nonrecursive set of guarded biconditionals, where a guarded biconditional takes the following form.

$$t_1(x_1) \wedge \cdots \wedge t_n(x_n) \Rightarrow (p(x_1, \ldots, x_n) \Leftrightarrow \phi(x_1, \ldots, x_n))$$

The third generalization of Extensional Reasoning has to do with dynamic systems descriptions. There has been significant attention paid in the formal methods community to checking whether a given dynamic system obeys some property. Often those descriptions are complete, but sometimes the description is incomplete—only partially specified. One might investigate whether translating an incomplete dynamic system description into a complete description would produce computational benefits similar to those for Extensional Reasoning. At the very least such a transformation would allow the same techniques used on complete descriptions to be applied to answer questions about incomplete descriptions.

The first and third long-term generalizations were actually part of the motivation for

looking into Extensional Reasoning in the first place. Finite Herbrand Logic was chosen in the end because it had many of the features of first-order logic, but was decidable in many respects, allowing us to investigate some of the central issues surrounding such translations without having to deal with undecidability. The hope is that the work described in this thesis will form the foundations upon which these generalizations can be studied.

Appendix A

Proofs

This appendix covers the proofs of the theorems found throughout the thesis. Some of the longer proofs include failed theorems to illustrate which plausible ideas failed. One of the central themes of this thesis is completeness, so it is fitting that some of the negative results are reported along with the positive ones.

A.1 Proofs for Chapter 1

The proofs covered in this section deal with the complexity of satisfiability in FHL. The result says that checking satisfiability of an FHL sentence ϕ is NEXPTIME-complete in the size of ϕ, where size is measured in the number of bits. We make the usual assumption that if ϕ mentions n predicates that each predicate is represented in $O(\log n)$ bits; likewise for object constants and variables. Thus, a sentence of length n can mention at most $O(n/\log n)$ predicates, object constants, and variables. [BGG97]

Theorem 18. *Let ϕ be an FHL sentence of length n in prenex form. Its satisfiability can be determined in NEXPTIME in n.*

Proof. This proof is inspired by [BGG97], Proposition 6.0.4. Say that ϕ mentions m object constants, has k quantifiers, and v variable occurrences. First, transform ϕ into propositional logic by applying the following two rules, where the set of object constants mentioned in ϕ is $\{a_1, \ldots, a_m\}$.

$$\forall x.\phi(x) \text{ becomes } \phi(a_1) \wedge \cdots \wedge \phi(a_m)$$
$$\exists x.\phi(x) \text{ becomes } \phi(a_1) \vee \cdots \vee \phi(a_m)$$

This transformation produces m^k copies of the original sentence where each variable is replaced by an object constant. Replacing each variable with an object constant means that the size of the sentence changes by $v(\lceil \log m \rceil - \lceil \log k \rceil)$. Since v, m, and k are all bounded from above by $\frac{n}{\log n}$, the total size increases by at most $\frac{n}{\log n} \log\left(\frac{n}{\log n}\right)$, if we set $\lceil \log k \rceil$ to zero. Since $\log\left(\frac{n}{\log n}\right)$ is bounded from above by $\log n$, we see that the size of the sentence increases by at most $\frac{n}{\log n} \log n = n$. Thus, the transformation requires time $O(nm^k)$ and outputs a sentence of size $O(nm^k)$.

One can easily transform this sentence into one in propositional logic in linear time and space; to avoid including equality axioms simply replace $c \neq d$, where c and d are distinct constants, with a proposition that is false in all models and replace $c = c$ with a proposition that is true in all models. It is well-known that the satisfiability of such a sentence can be checked in nondeterministic polynomial time in the size of the sentence. Thus its satisfiability can be checked in nondeterministic polynomial time in $O(nm^k)$.

Again we use the fact that both m and k are bounded from above by $\frac{n}{\log n}$, which ensures nondeterministic polynomial time in $n^{\frac{n}{\log n}^{\frac{n}{\log n}}}$. We show below that this function is bounded from above by $2^{O(n)}$; this gives us nondeterministic polynomial time in $2^{O(n)}$. Thus, there is some constant c such that a solution can be verified in time $\left(2^{O(n)}\right)^c = 2^{c*O(n)} = 2^{O(n)}$. That is, FHL satisfiability requires nondeterministic exponential time in the size of the sentence.

To finish the proof, we demonstrate that $n^{\frac{n}{\log n}^{\frac{n}{\log n}}}$ is bounded from above by $2^{O(n)}$. Set $t = n^{\frac{n}{\log n}^{\frac{n}{\log n}}}$ and take the log of both sides.

$$\begin{aligned}
\log t &= \frac{n}{\log n} \log\left(n^{\frac{n}{\log n}}\right) \\
&= \frac{n}{\log n} \log n + \frac{n}{\log n} \log\left(\frac{n}{\log n}\right) \\
&\leq \frac{n}{\log n} \log n + \frac{n}{\log n} \log n \text{ (Since } \tfrac{n}{\log n} \text{ is bounded by } n) \\
&= n + n \\
&\in O(n)
\end{aligned}$$

Exponentiating both sides guarantees us that $t \in 2^{O(n)}$, which completes the proof. □

Theorem 19. *Satisfiability of an FHL sentence is NEXPTIME hard in the size of the sentence.*

Proof. It is well known ([BGG97] Chapter 6, page 257) that FOL satisfiability for $\exists^*\forall^*.\phi$, where ϕ is quantifier-free, function-free, object-constant-free, and equality-free, is NEXPTIME-complete in the size of the sentence.

Consider a sentence $\exists^*\forall^*.\phi$ and the application of skolemization to produce $\forall^*.\phi'$. Skolemization requires time polynomial in the size of the sentence and preserves first-order satisfiability; also note that the size of ϕ' must be no larger than the size of ϕ since the number of object constants in ϕ' is less than the number of variables in ϕ, making the number of bits required to represent object constants no larger than the number of bits required to represent variables.

Because $\forall^*\phi'$ has no equality and is a universal sentence, Herbrand's theorem applies. $\forall^*\phi'$ is first-order satisfiable if and only if it is Herbrand satisfiable. Notice that $\forall^*\phi'$ is in Finite Herbrand Logic, ensuring that first-order satisfiability reduces to FHL satisfiability in this case. For the purpose of contradiction suppose FHL satisfiability were less than NEXPTIME. Then using the procedure above we could convert first-order satisfiability of $\exists^*\forall^*.\phi$ into FHL satisfiability in polynomial time and produce a solution in less than NEXPTIME. But that contradicts the NEXPTIME hardness result for $\exists^*\forall^*.\phi$. Thus, FHL satisfiability must be NEXPTIME hard in the size of the sentence. □

Clearly then, since satisfiability in FHL is contained within NEXPTIME in the size of the sentence and is NEXPTIME hard in the size of the sentence, we can conclude that satisfiability in FHL is NEXPTIME-complete in the size of the sentence. This ensures that validity—the complement of satisfiability—is coNEXPTIME-complete.

A.2 Proofs for Chapter 3

A.2.1 Completeness of Biconditional Definitions

This section is dedicated to proving the following theorem.

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

First, we start with some definitions used in the proof.

Definition 2 (Axiomatic Completeness). *A set of sentences Δ is complete for a language (set of sentences) L if and only if for every closed sentence $\phi \in L$,*

$$\Delta \models \phi \text{ or } \Delta \models \neg\phi \text{ or both.}$$

A set of sentences Δ is complete for a vocabulary (set of object constants and predicates) V if and only if for every closed sentence ϕ, whose vocabulary is a subset of V,

$$\Delta \models \phi \text{ or } \Delta \models \neg\phi \text{ or both.}$$

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

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

Notice this definition implicitly assumes the definitions are oriented, i.e. $r(\overline{x})$ is defined by $\phi(\overline{x})$ in the biconditional $r(\overline{x}) \Leftrightarrow \phi(\overline{x})$. But consider the sentence $p(\overline{x}) \Leftrightarrow q(\overline{x})$. Is q the definition for p or is p the definition for q?

The lemmas and theorems below assume we have a nonrecursive set of biconditional definitions. That is, if the set of definitions can be oriented so that the dependency graph is nonrecursive then those lemmas and theorems hold. Checking whether a given set of definitions can be so oriented is an algorithmic question, not a model-theoretic one.

This section proves that a nonrecursive set of biconditional definitions with one definition per predicate besides equality is complete. The structure of the proof is inductive: it starts by showing that every theory in FHL is complete for all sentences whose only predicate is $=$. Then for each biconditional definition, it must be shown that if the theory is complete for each of the predicates in the body then it must be complete with respect to the predicate in the head. That proof is broken into two steps: the first shows that if the theory is complete

with respect to each of the predicates in the body then it is complete with respect to the set of all predicates in the body. The second step shows that if the theory is complete with respect to the set of the predicates in the body then it is complete with respect to the predicate in the head.

First we prove a lemma that says in FHL if a theory is complete for all the ground atoms in some vocabulary then the theory is complete with respect to all the sentences in that vocabulary. This allows subsequent proofs to focus on showing that a theory is complete with respect to the ground atoms, and then quickly conclude that the theory is complete for all sentences.

Lemma 5 (Ground Atoms). *Suppose Δ is a set of sentences in FHL, and V is a vocabulary that includes all of the object constants mentioned in Δ. If Δ is complete with respect to all of the ground atoms formed from vocabulary V then Δ is complete with respect to V itself, i.e. to all the sentences formed from V.*

Proof. If Δ is unsatisfiable, then it is complete with respect to V since all sentences are entailed. So suppose Δ is satisfiable. For every ground atom $p(\overline{a})$ in V, Δ entails $p(\overline{a})$ or Δ entails $\neg p(\overline{a})$. By induction on the structure of a closed sentence built from V, we show that if ϕ is closed and built from V that Δ either entails ϕ, or it entails $\neg\phi$.

Base: By assumption, every ground atom or its negation is entailed by Δ. Inductive: Assume that if ϕ is a closed sentence from V that Δ either entails ϕ or it entails $\neg\phi$. Show that Δ entails each of the following sentences or their negations.

- $\neg\phi$: If $\neg\phi$ is closed, so must ϕ be closed; likewise, if the vocabulary of $\neg\phi$ is a subset of V so must ϕ's vocabulary be a subset. By the IH, Δ entails either ϕ or it entails $\neg\phi$. In the latter case, clearly, $\neg\phi$ is entailed, and in the former case its negation, $\neg\neg\phi$, is entailed.

- $\phi \vee \psi$: If $\phi \vee \psi$ is closed and from vocabulary V so must ϕ and ψ be closed and from vocabulary V. Thus by the IH, $\Delta \models \phi$ or $\Delta \models \neg\phi$; likewise for ψ. By the truth-table definition for $\vee$, we know that if either ϕ or ψ or both are entailed then $\Delta \models \phi \vee \psi$, thus ensuring our sentence is entailed. Otherwise, neither is entailed, which means both of $\neg\phi$ and $\neg\psi$ are entailed: $\Delta \models \neg\phi \wedge \neg\psi$. But by DeMorgan's, we have $\Delta \models \neg(\phi \vee \psi)$, which ensures the negation of our sentence is entailed.

- $\forall x.\phi(x)$: If $\forall x.\phi(x)$ is closed and from vocabulary V then x must be the only free variable in ϕ. In FHL, $\forall x.\phi(x)$ is equivalent to $\phi(a_1) \wedge \cdots \wedge \phi(a_n)$, where $\{a_1, \ldots, a_n\}$

is the set of all object constants in V. Each $\phi(a_i)$ is closed and because V includes all the object constants in Δ, each $\phi(a_i)$ is built from vocabulary V. By the IH, for each i, $\Delta \models \phi(a_i)$ or $\Delta \models \neg\phi(a_i)$. If Δ entails $\phi(a_i)$ for all i then $\phi(a_i)$ is true in all models of Δ for all i, and therefore, every model of Δ satisfies $\phi(a_1) \wedge \cdots \wedge \phi(a_n)$. This ensures $\Delta \models \phi(a_1) \wedge \cdots \wedge \phi(a_n)$ and consequently that $\Delta \models \forall x.\phi(x)$. In the other case, there is some k for which $\Delta \models \neg\phi(a_k)$. By disjunction introduction, we can add any disjuncts to $\neg\phi(a_k)$, and Δ will still entail the sentence. This ensures $\Delta \models \neg\phi(a_1) \vee \cdots \vee \neg\phi(a_n)$, which by DeMorgan's, ensures $\Delta \models \neg(\phi(a_1) \wedge \cdots \wedge \phi(a_n))$. Again, we can replace the conjunction over all object constants with a universal quantifier, giving us what we want: $\Delta \models \neg\forall x.\phi(x)$.

This completes the inductive step and the proof by mathematical induction. Since Δ is complete with respect to all closed sentences, and every open sentence $\phi(\overline{x})$ with free variables $\overline{x}$ is equivalent to the sentence $\forall^*.\phi(\overline{x})$, Δ is complete with respect to all sentences. □

Lemma 6 (Completeness of Equality). *Finite Herbrand Logic fixes the theory of equality, i.e. for every closed sentence s whose only predicate is equality, $\models s$ or $\models \neg s$ for every vocabulary V.*

Proof. Consider two Herbrand models M and N for vocabulary V and some ground atom s built from V whose only predicate is $=$. We show that $M \models s$ iff $N \models s$. This ensures that every theory in FHL is complete with respect to the ground $=$ atoms. By the Ground Atoms lemma, this guarantees that every theory in FHL is complete with respect to all sentences that mention only the equality predicate.

Consider any ground equality atom $t = u$. Since it is ground, by the definition of FHL satisfaction $t = u$ is satisfied (in both models) when t and u are the same term. $\neg(t = u)$ is satisfied (in both models) when t and u are distinct terms. One or the other must hold; thus in all cases, either $t = u$ is satisfied or $\neg t = u$ is satisfied by both models. Thus $M \models t = u$ if and only if $N \models t = u$. □

Next we show that if we have a theory that is complete for some set of predicates then the theory is complete for the set of all those predicates. That is, completeness is preserved across the union of vocabularies that differ in predicates. This property ensures that if we have a biconditional definition, and the theory is complete for every predicate in the body

then the theory is complete for all instances of the body itself. The unexpected part is that the lemma does not hold for full first-order logic, but it does hold in FHL.

Failed Lemma 1 (FOL Completeness Preserved Under Union). *Let Δ be a set of sentences in first-order logic with object constants O that is complete with respect to each of the vocabularies $R_1 \cup O, \ldots, R_n \cup O$. Δ is complete respect to the vocabulary $R_1 \cup \cdots \cup R_n \cup O$.*

Proof. The counterexample relies on monadic first-order logic. Consider the following sentence set.

$$\Delta = \{p(a), \neg p(b), \neg q(a), q(b)\}$$

First we show that Δ is complete for the vocabularies $\{p, a, b\}$ and $\{q, a, b\}$ individually. Then we show that Δ is incomplete for the vocabulary $\{p, q, a, b\}$ by finding a closed sentence in that vocabulary such that neither it nor its negation is entailed.

Consider any closed sentence built from the vocabulary $\{p, a, b\}$. Clearly all the ground sentences or their negations are entailed by Δ. That leaves the quantified sentences, of which there are only four that must be considered because p is monadic.

$$\exists x.p(x)$$
$$\neg\exists x.p(x)$$
$$\exists x.\neg p(x)$$
$$\neg\exists x.\neg p(x)$$

The first is entailed, and the second is the negation of the first. The third is also entailed, and the fourth is the negation of the third. Thus $p(a) \wedge \neg p(b)$ is complete for the vocabulary $\{p, a, b\}$. Likewise, $\neg q(a) \wedge q(b)$ is complete for the vocabulary $\{q, a, b\}$.

But, Δ itself is incomplete with respect to the vocabulary $\{p, q, a, b\}$ because there is a closed sentence built from this vocabulary such that neither it nor its negation is entailed:

$$\exists x.(p(x) \wedge q(x))$$
$$\forall x.(\neg p(x) \vee \neg q(x))$$

Δ does not entail the existential since neither a nor b are in both p and q. The universal is certainly not entailed by Δ, since in FOL ground sentences cannot entail universal sentences (without a DCA, of course). □

This lemma does work, however, when equality is fixed, e.g. in FHL, because quantifiers

can be rewritten in terms of ground sentences. Notice that having just the UNA or just the DCA is insufficient. The counterexample given above works for the UNA without the DCA. A theory with the DCA but without the UNA is difficult to make complete to begin with. Consider a theory that we might expect to be complete.

$$p(x) \Leftrightarrow (x = a \vee x = b)$$

This theory is incomplete without the UNA because neither the query $a = b$ nor its negation $a \neq b$ is entailed. However, with both the UNA and DCA, constructing complete theories is relatively simple, and union preserves completeness.

Lemma 7 (FHL Completeness Preserved Under Union). *Let Δ be a set of sentences in FHL with object constants O that is complete with respect to each of the vocabularies $R_1 \cup O, \ldots, R_n \cup O$. Δ is complete with respect to the vocabulary $R_1 \cup \cdots \cup R_n \cup O$.*

Proof. If Δ is unsatisfiable then it is complete for every vocabulary and the result holds. Suppose Δ is satisfiable. We show that Δ is complete for each ground atom in the vocabulary $R_1 \cup \cdots \cup R_n \cup O$; then by the Ground Atoms lemma, we know that Δ is complete with respect to the entire vocabulary. This result is immediate since each ground atom $r(a_1, \ldots, a_n)$ is a ground atom in $R_i \cup O$ for some i, and Δ is complete for $R_i \cup O$. Δ must therefore entail $r(a_1, \ldots, a_n)$ or $\neg r(a_1, \ldots, a_n)$. □

Next we show that if a theory is complete for all the predicates in the body of a biconditional then the theory is also complete for the predicate in the head of the biconditional.

Lemma 8 (Biconditional Completeness). *Suppose Δ is in FHL, has object constants O, is complete with respect to the vocabulary $\{r_1, \ldots, r_n\} \cup O$, and includes the definition $r(\overline{x}) \Leftrightarrow \phi(\overline{x})$, where the vocabulary of $\phi(\overline{x})$ is a subset of $\{r_1, \ldots, r_n\} \cup O$. Δ is complete with respect to the vocabulary $\{r\} \cup O$.*

Proof. If Δ is unsatisfiable, then it is complete with respect to $\{r\} \cup O$, as Δ entails every sentence and its negation. So suppose Δ is satisfiable. We show that Δ is complete for all ground atoms in $\{r\} \cup O$ and then by the Ground Atoms lemma, we conclude that Δ is complete with respect to the entire vocabulary.

Consider any ground atom $r(a_1, \ldots, a_n)$. Because the free variables in the head of the definition for r are nonrepeating, $r(a_1, \ldots, a_n)$ is always an instance of the head. Consequently, this ground atom is logically equivalent to $\phi(a_1, \ldots, a_n)$. Since Δ is complete with

respect to the vocabulary of $\phi(a_1, \ldots, a_n)$, Δ entails either $\phi(a_1, \ldots, a_n)$ or $\neg\phi(a_1, \ldots, a_n)$. In the former case, Δ must then also entail $r(a_1, \ldots, a_n)$, and in the latter case, Δ must entail $\neg r(a_1, \ldots, a_n)$. That is, Δ must be complete for all the ground atoms in the vocabulary $\{r\} \cup O$. □

The fact that the definitions are nonrecursive ensures that we can order the definitions and then perform an inductive proof that incrementally demonstrates that a set of biconditionals is complete for its entire vocabulary. Just as in the database literature, this ordering is formalized through the notion of stratum: each biconditional definition for predicate r belongs to the lowest stratum that is higher than all the strata for biconditionals defining the predicates in the body of that definition.

Definition 10 (Stratum). *Let Δ be a set of nonrecursive biconditional definitions with one definition for each predicate besides $=$. The stratum of each predicate is defined as shown in Algorithm 9. (Borrowed from [Ull89]).*

Algorithm 9 Algorithm for assigning a predicate to a stratum.

```
1: for all predicates r besides = do
2:     stratum[r] := 1
3: end for
4: repeat
5:     for all definitions r(x̄) ⇔ φ(x̄) do
6:         when φ(x̄) contains some predicate q such that stratum[q] > stratum[r] then
           stratum[r] := stratum[q] + 1
7:     end for
8: until there are no changes to any stratum or some stratum exceeds the number of
   predicates
9: return stratum
```

Lemma 9 (Stratum Bound). *If Δ is a nonrecursive set of biconditional definitions where there is one definition per predicate then stratum$[r]$ is no more than the number of predicates.*

Proof. (Adopted from [Ull89].) First we show that if the algorithm above changes the stratum of some predicate r i times then there must be a path in the dependency graph of length i that ends at r. Given that, for the purpose of contradiction suppose there were some predicate r assigned a stratum greater than the number of predicates. Then by the

claim there is some path of length greater than the number of predicates, and the existence of such a path ensures there must be a cycle. This contradicts the fact that a nonrecursive set of biconditionals has an acyclic dependency graph.

To show: if the algorithm above changes the stratum of predicate r i times then there must be a path in the dependency graph of length i that ends at r. By induction on the number of times the algorithm above changes the stratum for a predicate.

Base: 0. The stratum for predicate r has been changed zero times. Every predicate exists in the dependency graph, which means there is always a path of length 0 ending at each relation.

Inductive step: Assume that if $stratum[r]$ is changed n times then there must be a path in the dependency graph ending at r that is at least n long. Show for $n+1$. Consider the $(n+1)^{th}$ change made to $stratum[r]$. In order for that change to have occurred, there must have been some definition $r(\overline{x}) \Leftrightarrow \phi(\overline{x})$, where there is some p in $\phi(\overline{x})$ such that $stratum[p] \geq r$. By the IH, there must then be a path in the dependency graph ending at p of length n. And because this definition exists, we see that there must also be an edge from p to r, ensuring that there is also a path of length $n+1$ ending at r. This completes the inductive step and the proof by induction. □

The Stratum Bound lemma ensures that if Δ is finite every sentence belongs to some finite stratum. This allows us to incrementally prove that Δ is complete with respect to its entire vocabulary by showing that Δ is complete at each stratum with respect to the vocabulary of that stratum and all the lower strata.

Theorem 1 (Biconditional Completeness). *Let Δ be a finite, nonrecursive set of biconditional definitions in FHL, where there is exactly one definition for each predicate in Δ with no definition for $=$. Δ is complete with respect to its vocabulary.*

Proof. The Stratum Bound lemma ensures that every sentence in Δ belongs to some stratum less than or equal to k, the number of predicates in Δ. Thus, if we show that Δ is complete with respect to the vocabulary of strata $\{1, \ldots, k\}$, we have shown Δ is complete with respect to its entire vocabulary.

To show: Δ is complete with respect to the vocabulary of sentences in strata $\{1, \ldots, k\}$. By induction on k.

Base: 0. None of the definitions are assigned stratum 0. The vocabulary of stratum 0 is therefore $\{=\} \cup O$, and there is only one predicate used by all the sentences of interest:

=. By the Completeness of Equality lemma, every such sentence or its negation is entailed by Δ.

Inductive: Suppose that for stratum n, $\{r_1, \ldots, r_m\}$ is the set of predicates belonging to stratum $\{1, \ldots, n\}$ and that Δ is complete for vocabulary $\{r_1, \ldots, r_m\} \cup O$. Show that in stratum $n+1$, which is the stratum with definitions for predicates $\{s_1, \ldots, s_h\}$, that Δ is complete for vocabulary $\{r_1, \ldots, r_m, s_1, \ldots, s_h\} \cup O$.

Consider the definition for a predicate that belongs to stratum $n+1$: $s_i(\overline{x}) \Leftrightarrow \phi(\overline{x})$. Because this definition belongs to stratum $n+1$, we know that every one of the predicates mentioned in $\phi(\overline{x})$, $\{r_{i_1}, \ldots, r_{i_p}\}$, must have definitions that belong to stratum n or lower. Since $\{r_{i_1}, \ldots, r_{i_p}\}$ is a subset of $\{r_1, \ldots, r_m\}$, by the IH, we know that Δ is complete for vocabulary $\{r_{i_1}, \ldots, r_{i_p}\} \cup O$. Since the vocabulary of $\phi(\overline{x})$ is a subset of $\{r_{i_1}, \ldots, r_{i_p}\} \cup O$, by the Biconditional Completeness lemma, we therefore know that Δ is complete with respect to vocabulary $\{s_i\} \cup O$.

Since we chose the definition $s_i(\overline{x}) \Leftrightarrow \phi(\overline{x})$ arbitrarily, the above argument applies to all such definitions in stratum $n+1$, and therefore, Δ is complete with respect to $\{s_1\} \cup O$, $\ldots$, and $\{s_h\} \cup O$. Since Δ is also complete with respect to $\{r_1, \ldots, r_m\} \cup O$, by the Completeness Preserved Under Union lemma, we see that Δ is complete with respect to $\{r_1, \ldots, r_m, s_1, \ldots, s_h\} \cup O$.

This completes the inductive step and the proof by mathematical induction. □

A.2.2 Soundness and Completeness of TO-BICONDS

This section addresses the soundness and completeness of TO-BICONDS, the algorithm for reformulating a set of FHL sentences into a nonrecursive set of biconditionals. TO-BICONDS, Algorithm 1, can be found on page 22. Below, we use $Preds(\Delta)$ to indicate the set of predicates that appear in Δ.

Theorem 2 (Soundness of TO-BICONDS). *Under the conditions listed below, if* TO-BICONDS$(\Delta, \{=\})$ *returns a nonempty* Γ*, then* Γ *is a nonrecursive set of 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 nonrecursive biconditional definition for* p *entailed by* Σ*.*

Proof. If TO-BICONDS returns a nonempty Γ then Γ must be a set of biconditionals, each of which is individually nonrecursive, since REFORMULATE-TO-BICOND returns only such biconditionals. By straightforward induction on the number of predicates, we can see that TO-BICONDS returns a set of REFORMULATE-TO-BICOND results. Notice also that Γ contains one definition per predicate besides equality, again by straightforward induction on the number of predicates. It remains to be shown that (1) Γ is nonrecursive as a whole and (2) Γ is logically equivalent to Δ.

To show that Γ is nonrecursive, it is enough to realize that each call to TO-BICONDS constructs a nonrecursive definition for some predicate p whose body contains predicates that already have definitions. We show Γ is nonrecursive by induction on the number of predicates. $=$ has a built-in definition, which covers the base case. For the inductive step, assume the dependency graph built for n predicates is acyclic and REFORMULATE-TO-BICOND constructs a biconditional for p. Notice that the only predicates in the body of that biconditional must occur in *basepreds*, the argument passed to TO-BICONDS that contains the predicates with complete definitions. The new biconditional preserves the acyclic property in the dependency graph. p has no definition and therefore does not appear in the graph yet (else it would be in the *basepreds* argument and would not have been selected as the head of the biconditional). Since p occurs in the head of the definition but not in the body, edges are not added that both go into and out of p. Thus, the result is an acyclic dependency graph.

To show Γ is logically equivalent to Δ, consider any definition $p(\overline{x}) \Leftrightarrow \phi(\overline{x})$ returned by REFORMULATE-TO-BICOND applied to a subset of Δ, Δ', and predicate p. By the soundness of that function, we know that Δ' entails the biconditional, and since Δ' is a subset of Δ, Δ also entails the biconditional. Suppose for the purpose of contradiction there were some other definition for p entailed by Δ that were not logically equivalent to the one found by REFORMULATE-TO-BICOND: $p(\overline{x}) \Leftrightarrow \psi(\overline{x})$. If the biconditionals are not logically equivalent then $\phi(\overline{x})$ and $\psi(\overline{x})$ are not logically equivalent. But since Δ entails both biconditionals, it entails $\phi(\overline{x}) \Leftrightarrow \psi(\overline{x})$, which is inconsistent with our presumption since Δ is satisfiable. Thus no such alternate definition for p can exist.

Thus every predicate p is defined in Γ as it is defined in Δ, and therefore $\Delta \models \Gamma$. Since every predicate p in Δ has some definition in Γ, and Γ is complete, Γ is at least as strong as Δ, and we get $\Gamma \models \Delta$. Thus, Δ and Γ are logically equivalent. □

Before going on to completeness, we prove a lemma that will aid us in the proof. The

lemma tells us something about any set of sentences that is logically equivalent to a biconditional definition: that every sentence mentions the predicate defined by that biconditional.

Lemma 10. *Suppose β is a nonrecursive biconditional definition where the predicate in the head is p. Suppose further that Δ is a set of sentences that is logically equivalent to β such that $Preds(\Delta) \subseteq Preds(\beta)$. Every sentence in Δ is either a tautology or includes the predicate p.*

Proof. Suppose for the purpose of contradiction that there were some falsifiable sentence ϕ in Δ that did not mention p. Then, since Δ and β are logically equivalent, they entail the same set of sentences, and therefore they both entail ϕ. Since ϕ is falsifiable, there is some model that does not satisfy it; thus, there must be some interpretation of the predicates mentioned in ϕ (and therefore in the body of β) that do not satisfy ϕ. Since that interpretation does not satisfy ϕ, and the models that satisfy β are a subset of those that satisfy ϕ (since $\beta \models \phi$), that interpretation does not satisfy β either. We claim that because β is a biconditional definition, there can be no such interpretation, which gives us the needed contradiction.

Now, we must show that a biconditional $p(\overline{x}) \Leftrightarrow \psi(\overline{x})$ is satisfied by every interpretation for the predicates P mentioned in $\psi(\overline{x})$. Consider any model M that interprets P. Now, construct a model that is exactly the same as M for predicates P but whose definition for p is the set of all instances that satisfy $\psi(\overline{x})$. The result is a model that satisfies the biconditional. This argument holds regardless of the model M and of the interpretations for the predicates P; therefore, it holds for all interpretations. □

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 false.*

- *Δ is a satisfiable, nonempty sentence set.*
- *Δ is logically equivalent to a nonrecursive set of biconditionals with one definition per predicate besides equality: $\{\beta_1, \ldots, \beta_n\}$.*
- *There is a partitioning of Δ into $S_1, \ldots, S_n$ such that β_i is logically equivalent to S_i and $Preds(\beta_i) = Preds(S_i)$, 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 false.*

Proof. We need to show if that the conditions above are satisfied then the return value of TO-BICONDS(Δ, $\{=\}$) is not false.

Suppose for the purpose of contradiction that this greedy algorithm returns false but the above conditions are true, which means the greedy algorithm made a mistake. The TO-BICONDS algorithm partitions the sentences of Δ so that all the sentences necessary for defining each of the predicates belong to the same partition. Because REFORMULATE-TO-BICOND is complete, if the sentence sets it is given are always the right ones, TO-BICONDS will always find the right biconditionals. Thus, if the algorithm makes a mistake it is because the wrong sentences were given to REFORMULATE-TO-BICOND. This can happen because either (1) some sentence was handed to REFORMULATE-TO-BICOND too early, thereby causing the loss of a proof for some biconditional later on, or (2) one of the available sentences was not handed to REFORMULATE-TO-BICOND but was required for entailing the biconditional.

1) If some sentence σ is given to REFORMULATE-TO-BICOND too early, resulting in that sentence being unavailable for future REFORMULATE-TO-BICOND calls, it must have been selected for defining some predicate q when it was not actually needed for the definition. Moreover, if σ were removed too early, it must have been important for the definition of the some other predicate p. Since σ was handed to REFORMULATE-TO-BICOND, it must mention q, and because it is important for p, it must mention p according to the lemma above. But for σ to be selected for a definition of q, all its predicates besides q must already have definitions (otherwise q wouldn't be the only unmarked predicate in σ); this includes p. Thus, p must already have a definition, which contradicts the assumption that this sentence was important for the definition for p. This ensures that selecting σ was not a mistake.

2) The other possible mistake is that the algorithm considers too few sentences to give to REFORMULATE-TO-BICOND. Because the algorithm considers all sentences with exactly one unmarked predicate, there must be some sentence with either less than one or more than one unmarked predicate that is necessary to entail the definition for some predicate, say p. In the former case, all the predicates have been marked, and since p has not been marked, we know the sentence does not mention p. According to the lemma above, this means that the sentence could not be one of the sentences necessary for defining p; consequently, ignoring the sentences where every predicate is marked is not a mistake.

In the latter case, the missing sentence would need to mention two or more unmarked predicates. As shown in the last section, in order for a set of biconditionals to be nonrecursive, there must be an ordering on the biconditionals so that the body of biconditional 0

mentions only =, and the body of a biconditional assigned i uses a subset of the predicates that are defined by biconditionals 0 through $i-1$. Suppose some sentence with more than one unmarked predicate were necessary for defining the biconditional for p. Then because all but one of those unmarked predicates occur in the body of the biconditional for p (since the biconditional and the corresponding sentences in Δ have the same predicates), all but one of those unmarked predicates must have definitions that are assigned lower strata than the stratum assigned to the definition for p. Thus by straightforward induction, if some sentence is necessary for entailing the biconditional for p, eventually all the predicates in that sentence will have definitions and those predicates will be marked. Thus, considering a sentence with more than one unmarked predicate is unnecessary.

Since neither of the mistakes could be mistakes, the algorithm makes no mistakes, and consequently it never returns false when it should not. □

A.2.3 Equivalence Preservation of BICONDS-TO-DATALOG

This section details the conditions under which the algorithms presented for converting a nonrecursive set of biconditionals into *nr-datalog*$^\neg$ preserve logical equivalence.

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

0. $E =$ PARTITION*(Δ). E is a subset of the predicates mentioned in Δ and does not include =.*

1. $\Gamma =$ MATERIALIZE*(E, Δ). For every ground atom ρ built using a predicate in E, $\Delta \models_{FHL} \rho$ if and only if $\Gamma \models_D \rho$.*

2. $I = Preds(\Delta) - (E \cup \{=\})$ and $\Lambda =$ VIEWS-TO-DATALOG*(I, Δ). For any set B of biconditional definitions for the predicates in E, use Δ_B to denote Δ where the definitions for E are replaced by B. For every ground atom ρ built using a predicate in I and every finite, nonrecursive set B of biconditional definitions for the predicates defined in E, $\Delta_B \models_{FHL} \rho$ if and only if* MATERIALIZE*$(E, \Delta_B) \cup \Lambda \models_D \rho$.*

3. $(\psi, \Theta) =$ SENTENCE-TO-DATALOG(ϕ). *ψ mentions a subset of the predicates in Θ, which is a set of nr-datalog$^\neg$ rules. Consider any model M with interpretations for exactly those predicates mentioned in Δ. Use M_T to denote the representation of M using extensional tables. M satisfies ϕ if and only if $M_T \cup \Theta \models_D \psi$.*

4. *The predicates introduced by* VIEWS-TO-DATALOG *and* SENTENCE-TO-DATALOG*, if any, are disjoint.*

Proof. First, we know that $\Gamma =$ MATERIALIZE(E, Δ). By property (1), we see that for every ground atom ρ whose predicate is drawn from E, it is the case that

$$\Delta \models_{FHL} \rho \text{ if and only if } \Gamma \models_D \rho.$$

Second, we know that $\Lambda =$ VIEWS-TO-DATALOG(I, Δ). By property (2), we see that for every set B of biconditional definitions for E and every ground atom ρ whose predicate is drawn from I,

$$\Delta_B \models_{FHL} \rho \text{ if and only if MATERIALIZE}(E, \Delta_B) \cup \Lambda \models_D \rho.$$

In particular, this holds when B is the set of biconditional definitions for E that occur in Δ.

$$\Delta \models_{FHL} \rho \text{ if and only if } \Gamma \cup \Lambda \models_D \rho$$

Because E is the set of extensional predicates and I is the set of intensional predicates, the model that satisfies $\Gamma \cup \Lambda$ agrees with the model of Γ on the interpretations of the predicates E. Thus, for every ground atom ρ built from a predicate in E,

$$\Delta \models_{FHL} \rho \text{ if and only if } \Gamma \cup \Lambda \models_D \rho.$$

Equality has a fixed interpretation, which means that for every ground equality atom ρ,

$$\Delta \models_{FHL} \rho \text{ if and only if } \Gamma \cup \Lambda \models_D \rho.$$

Together, these three facts guarantee that for any ground atom ρ whose predicate is drawn from $E \cup I \cup \{=\}$,

$$\Delta \models_{FHL} \rho \text{ if and only if } \Gamma \cup \Lambda \models_D \rho.$$

Equivalently, we see that

$$\Delta \not\models_{FHL} \rho \text{ if and only if } \Gamma \cup \Lambda \not\models_{D} \rho.$$

Because Δ is a nonrecursive set of biconditionals with one definition per predicate besides equality, Δ is axiomatically complete. Likewise, by the *nr-datalog*$^{\neg}$ semantics, $\Gamma \cup \Lambda$ is complete. Completeness ensures that $\not\models \phi$ is equivalent to $\models \neg\phi$ when ϕ is a closed sentence. Thus we see that for a ground atom drawn from $E \cup I \cup \{=\}$

$$\Delta \models_{FHL} \neg\rho \text{ if and only if } \Gamma \cup \Lambda \models_{D} \neg\rho.$$

This guarantees that for any ground literal ρ built from the predicates $E \cup I \cup \{=\}$,

$$\Delta \models_{FHL} \rho \text{ if and only if } \Gamma \cup \Lambda \models_{D} \rho.$$

Notice that the vocabulary of $\Gamma \cup \Lambda$ includes $E \cup I \cup \{=\}$ but allows $\Gamma \cup \Lambda$ to include additional predicates; thus, the model that satisfies Δ may not be the same as the model that satisfies $\Gamma \cup \Lambda$ simply because of those extra predicates. But, the equivalence above ensures that the interpretations for $E \cup I \cup \{=\}$ are the same in the two models.

The rest of the proof is devoted to showing that the conditions on SENTENCE-TO-DATALOG are sufficient for ensuring that $\Delta \models_{FHL} \phi$ exactly when $\Gamma \cup \Lambda \cup \Theta \models \psi$.

We start with the fact that $(\psi, \Theta) =$ SENTENCE-TO-DATALOG(ϕ). Using property (3), we see that for any model M with interpretations for exactly the predicates $E \cup I \cup \{=\}$, $\models_M \phi$ exactly when $M_T \cup \Theta \models_D \psi$. This includes the model M^* that satisfies Δ. Thus, $\Delta \models \phi$ exactly when $M_T^* \cup \Theta \models_D \psi$. All that remains is to show that $M_T^* \cup \Theta \models_D \psi$ exactly when $\Gamma \cup \Lambda \cup \Theta \models \psi$. If we use $\hat{M}$ to name the model that satisfies $\Gamma \cup \Lambda$, this reduces the problem to showing that $M_T^* \cup \Theta \models_D \psi$ exactly when $\hat{M}_T \cup \Theta \models \psi$.

The only difference between M^* and $\hat{M}$ are the interpretations of extra predicates in the latter. But these interpretations cannot affect Θ or ψ because by property (4) neither Θ nor ψ mention any of those extra predicates. (The predicates mentioned in Θ and ψ not in $E \cup I \cup \{=\}$ must be disjoint from those mentioned by Λ. Γ mentions only predicates in Δ. Thus none of the new predicates in $\hat{M}$ are mentioned by Θ or ψ.) Thus, the model that satisfies $\hat{M}_T \cup \Theta$ is the same as the one that satisfies $M_T^* \cup \Theta$ except for the interpretations of the extra predicates. Because ψ mentions only the predicates in Θ and Δ, the model

that satisfies $M_T^* \cup \Theta$ must agree with the model that satisfies $\hat{M}_T \cup \Theta$ on whether or not ψ is satisfied. Thus, $M_T^* \cup \Theta \models_D \psi$ holds exactly when $\hat{M}_T \cup \Theta \models \psi$ holds, which concludes the proof.

□

A.2.4 Soundness and Completeness of ER-ENTAILEDP

This section proves the soundness and completeness of the Extensional Reasoning proof system for the case of complete theories.

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

- FHL-ENTAILEDP *is sound and complete for $\models_{FHL}$.*
- DB-ENTAILEDP *is sound and complete for $\models_D$.*

Proof. ER-ENTAILEDP(Δ, ϕ) returns true if and only if either

- TO-BICONDS(Δ) returns false and FHL-ENTAILEDP(Δ, ϕ) returns true or
- (1) TO-BICONDS(Δ) returns a nonempty set of biconditionals Σ,
  (2) BICONDS-TO-DATALOG(Σ, ϕ) returns $(\Gamma, \Lambda \cup \Theta, \psi)$, and
  (3) DB-ENTAILEDP$(\Gamma, \Lambda \cup \Theta, \psi)$ returns true

If TO-BICONDS(Δ) returns false, then ER-ENTAILEDP(Δ, ϕ) returns true iff FHL-ENTAILEDP(Δ, ϕ) returns T. Since we assume FHL-ENTAILEDP is sound and complete, in this case ER-ENTAILEDP(Δ, ϕ) returns true iff $\Delta \models_{FHL} \phi$.

Consider the case where TO-BICONDS(Δ) returns a nonempty Σ.

ER-ENTAILEDP(Δ, ϕ) returns true
iff DB-ENTAILEDP$(\Gamma, \Lambda \cup \Theta, \psi)$ returns true
iff $\Gamma \cup \Lambda \cup \Theta \models_D \psi$ (DB-ENTAILEDP is sound and complete)
iff $\Sigma \models_{FHL} \phi$ (by Equivalence Preservation of BICONDS-TO-DATALOG)
iff $\Delta \models_{FHL} \phi$ (by Soundness of TO-BICONDS)

□

A.3 Proofs for Chapter 4

A.3.1 Theory Resolution

In this section we prove the soundness and completeness of CTR+RES, the theory resolution inference procedure for determining the unsatisfiability of a theory $C \cup I$ when the incomplete portion of the theory I is in $\forall^*$.

Definition 5 (Complete Theory Resolution). *Complete Theory Resolution (CTR) is the following rule of inference, taking the complete theory C as a parameter. Applying CTR to closure on some set of clauses S is denoted $\text{CTR}(C, S)$. When in addition to closing under CTR, the closure includes all the usual resolution inference rules, we denote the result with $\text{CTR+RES}(C, S)$. In the inference rule below, p is a predicate with a complete definition in C, and $\mp p(\bar{t})$ has the opposite sign of $\pm p(\bar{t})$.*

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

Theorem 7 (Soundness and Completeness of CTR+RES for $\forall^*$). *Suppose $C \cup I$ is a finite set of FHL sentences, where C is a satisfiable, complete theory, and I is in $\forall^*$. $C \cup I$ is unsatisfiable if and only if $\text{CTR+RES}(C, I)$ 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 multiple applications of resolution, using literals entailed by C.

(Completeness) A database system compactly represents C, which is semantically a finite set of ground literals. We show that CTR+RES is complete by showing that every inference step that could occur using resolution on C, represented as a set of ground literals, together with I will also occur in $\text{CTR+RES}(C,I)$.

If we suppose $C \cup I$ is unsatisfiable in FHL then there is a resolution proof of the empty clause from $C \cup I \cup \{DCA\} \cup UNA$. Because C is effectively a set of ground literals, it is in $\forall^*$, and I is in $\forall^*$ by assumption; thus, $C \cup I$ is in $\forall^*$, and we can apply Reiter's result [Rei80] to conclude that $C \cup I \cup UNA$ includes the empty clause. In addition, Reiter shows that

in this case the transitivity and symmetry axioms for defining equality are not necessary for completeness, which means that paramodulation is unnecessary for completeness. Thus, the only necessary rules of inference are binary resolution and factoring; moreover, to prove completeness we need only consider inferences that use as a premise some literal from C.

Consider any step in which one of the literals from C is resolved with a non-unary clause.

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

We show that this resolvent is produced when CTR is applied to the first clause above, guaranteeing that this class of inferences will always occur in CTR+RES(C,I). When applied to the literal $\pm p(\bar{t})$ in the first clause, CTR finds all variable assignments $\nu_1, \ldots, \nu_m$ so that $\mp p(\bar{t})\nu_i$ belongs to C and then produces $\Phi\nu_i$ for every ν_i. In the original inference step above, because $\mp p(\bar{a})$ belongs to C, it must be ground, and therefore applying σ to $p(\bar{t})$ must produce $p(\bar{a})$. That is, σ is a substitution such that $\mp p(\bar{t})\sigma$ is in C; thus, σ must be one of the ν_i. Since CTR produces $\Phi\nu_i$ for every ν_i, clearly that includes the resolvent above, $\Phi\sigma$.

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 CTR+RES.

Hiding C does not result in the loss of any factoring step since factoring does not apply to unit clauses. Thus, every inference rule application necessary to produce the empty clause from $C \cup I$ using resolution can be mirrored using CTR+RES; hence, if there is a sequence of resolution steps that produces the empty clause, there is also a sequence of CTR+RES steps that produces the empty clause. □

A.3.2 *poss* Properties

In this section we prove the theorems that say how *poss* interacts with various logical connectives: $\vee$, $\exists$, $\neg$, $\wedge$, $\forall$.

Theorem 8 (*poss* distributes over $\vee$). *In FHL, $poss_{\phi(\overline{x},\overline{y})\vee\psi(\overline{x},\overline{z})}(\overline{x},\overline{y},\overline{z})$ is logically equivalent to $poss_{\phi(\overline{x},\overline{y})}(\overline{x},\overline{y}) \vee poss_{\psi(\overline{x},\overline{z})}(\overline{x},\overline{z})$.*

Proof. ($\Rightarrow$) Suppose $poss_{\phi(\overline{x},\overline{y})\vee\psi(\overline{x},\overline{z})}(\overline{a},\overline{b},\overline{c})$ is true. Then there is some model M that satisfies $\phi(\overline{a},\overline{b})\vee\psi(\overline{a},\overline{c})$. By the definition of disjunction, M must therefore satisfy either disjunct or both disjuncts. In any case, we see that either $poss_{\phi(\overline{a},\overline{b})}$ or $poss_{\psi(\overline{a},\overline{c})}$ is true. Consequently, $poss_{\phi(\overline{x},\overline{y})}(\overline{a},\overline{b})$ or $poss_{\psi(\overline{x},\overline{z})}(\overline{a},\overline{c})$ is true, and so their disjunction must be true: $poss_{\phi(\overline{x},\overline{y})}(\overline{a},\overline{b})\vee poss_{\psi(\overline{x},\overline{z})}(\overline{a},\overline{c})$.

($\Leftarrow$) Suppose $poss_{\phi(\overline{x},\overline{y})}(\overline{a},\overline{b})\vee poss_{\psi(\overline{x},\overline{z})}(\overline{a},\overline{c})$ is true. Then either there is a model that satisfies $\phi(\overline{a},\overline{b})$ or there is a model that satisfies $\psi(\overline{a},\overline{c})$ or there is a model that satisfies both. In the first case, since the model satisfies $\phi(\overline{a},\overline{b})$, it must satisfy every disjunction including it as a disjunct. That model must therefore satisfy $\phi(\overline{a},\overline{b})\vee\psi(\overline{a},\overline{c})$. Likewise for the second case. The third case guarantees the existence of a model that satisfies the conjunction and therefore the disjunction. Therefore, in all three cases, we know that there is a model that satisfies $\phi(\overline{a},\overline{b})\vee\psi(\overline{a},\overline{c})$, which means $poss_{\phi(\overline{x},\overline{y})\vee\psi(\overline{x},\overline{z})}(\overline{a},\overline{b},\overline{c})$ is true.

Because we have shown equivalence for all instances of the theorem, by Herbrand semantics, we have proven the theorem. □

Theorem 9 (*poss* distributes over $\exists$). *In FHL, $poss_{\exists x.\phi(x,\overline{y})}(\overline{y})$ is logically equivalent to $\exists x.poss_{\phi(x,\overline{y})}(x,\overline{y})$.*

Proof. ($\Rightarrow$) Suppose $poss_{\exists x.\phi(x,\overline{y})}(\overline{a})$ is true. Then there is some model M that satisfies $\exists x.\phi(x,\overline{a})$. In FHL, this ensures there is some object constant b such that $\models_M \phi(b,\overline{a})$. By the definition of *poss*, $poss_{\phi(x,\overline{y})}(b,\overline{a})$ must therefore be true, implying $\exists x.poss_{\phi(x,\overline{y})}(x,\overline{a})$ must be true.

($\Leftarrow$) Suppose $\exists x.poss_{\phi(x,\overline{y})}(x,\overline{a})$ is true. Then because of FHL semantics and the completeness of *poss* definitions, there is some b such that $poss_{\phi(x,\overline{y})}(b,\overline{a})$ is true. By the definition of *poss*, there must be some model that satisfies $\phi(b,\overline{a})$. That same model certainly satisfies $\exists x.\phi(x,\overline{a})$, and serves as a witness for the fact that $poss_{\exists x.\phi(x,\overline{y})}(\overline{a})$ is true.

Because we have shown equivalence for all instances of the theorem, by Herbrand semantics, we have proven the theorem. □

Theorem 10 (*poss* distributes over $\neg$). *In FHL, $poss_{\neg\phi(\overline{x})}(\overline{x})$ is equivalent to $\neg poss_{\phi(\overline{x})}(\overline{x})$ exactly when the background theory Δ is satisfiable and complete with respect to $\phi(\overline{x})$.*

Proof. Consider any instance of the theorem: $poss_{\neg\phi(\overline{x})}(\overline{a})$ is equivalent to $\neg poss_{\phi(\overline{x})}(\overline{a})$ exactly when Δ is complete with respect to $\phi(\overline{a})$.

First we assume completeness and show the equivalence. Suppose that $\Delta \models \phi(\overline{a})$ or $\Delta \models \neg\phi(\overline{a})$.

($\Rightarrow$) Suppose $poss_{\neg\phi(\overline{x})}(\overline{a})$ is true. Then there must exist a model M that satisfies $\neg\phi(\overline{a})$. Since Δ entails either $\phi(\overline{a})$ or its negation, and M does not satisfy $\phi(\overline{a})$, it must be the case that $\Delta \models \neg\phi(\overline{a})$. This ensures that every model of Δ satisfies $\neg\phi(\overline{a})$, which means that none of the models satisfy $\phi(\overline{a})$; hence, $\neg poss_{\phi(\overline{x})}(\overline{a})$ is true.

($\Leftarrow$) Suppose $\neg poss_{\phi(\overline{x})}(\overline{a})$ is true. Then there is no model of Δ that satisfies $\phi(\overline{a})$. Immediately we see that every model satisfies $\neg\phi(\overline{a})$, and because Δ is satisfiable there must be at least one such model; consequently, $poss_{\neg\phi(\overline{x})}(\overline{a})$ is true.

Second we assume the equivalence and show that Δ must be complete with respect to $\phi(\overline{a})$. Suppose $poss_{\neg\phi(\overline{x})}(\overline{a})$ is equivalent to $\neg poss_{\phi(\overline{x})}(\overline{a})$.

If $poss_{\neg\phi(\overline{x})}(\overline{a})$ is true then Δ is satisfiable and by the equivalence $\neg poss_{\phi(\overline{x})}(\overline{a})$ is true: there is no model that satisfies $\phi(\overline{a})$. Hence, every model of Δ satisfies $\neg\phi(\overline{a})$, which ensures that $\Delta \models \neg\phi(\overline{a})$.

If $poss_{\neg\phi(\overline{x})}(\overline{a})$ is false then there is no model that satisfies $\neg\phi(\overline{a})$, which means every model of Δ satisfies $\phi(\overline{a})$. This guarantees that $\Delta \models \phi(\overline{a})$. Also, by the equivalence, $\neg poss_{\phi(\overline{x})}(\overline{a})$ is false, which means $poss_{\phi(\overline{x})}(\overline{a})$ is true, ensuring that Δ is satisfiable.

In either case, we therefore see that either $\Delta \models \neg\phi(\overline{a})$ or $\Delta \models \phi(\overline{a})$ and Δ is satisfiable, which ensures Δ is satisfiable and complete with respect to $\phi(\overline{a})$.

Because we have shown the theorem holds for all instances, by Herbrand semantics, we have proven the theorem. □

Theorem 11 (*poss* distributes over $\wedge$). *In FHL, $\phi(\overline{x},\overline{z})$ and $\psi(\overline{y},\overline{z})$ are independent with respect to Δ exactly when $poss_{\phi(\overline{x},\overline{z})\wedge\psi(\overline{y},\overline{z})}(\overline{x},\overline{y},\overline{z})$ is equivalent to $poss_{\phi(\overline{x},\overline{z})}(\overline{x},\overline{z}) \wedge poss_{\psi(\overline{y},\overline{z})}(\overline{y},\overline{z})$.*

Proof. First we assume independence and show the equivalence. Suppose $\phi(\overline{a},\overline{c})$ and $\psi(\overline{b},\overline{c})$ are independent with respect to Δ.

($\Rightarrow$) Suppose $poss_{\phi(\overline{x},\overline{z})\wedge\psi(\overline{y},\overline{z})}(\overline{a},\overline{b},\overline{c})$ is true. Then there is some model that satisfies $\phi(\overline{a},\overline{c}) \wedge \psi(\overline{b},\overline{c})$. That same model must then also satisfy $\phi(\overline{a},\overline{c})$ and $\psi(\overline{b},\overline{c})$ individually, ensuring $poss_{\phi(\overline{x},\overline{z})}(\overline{a},\overline{c})$ and $poss_{\psi(\overline{y},\overline{z})}(\overline{b},\overline{c})$ are both true. Thus, their conjunction $poss_{\phi(\overline{x},\overline{z})}(\overline{a},\overline{c}) \wedge poss_{\psi(\overline{y},\overline{z})}(\overline{b},\overline{c})$ is true.

($\Leftarrow$) Suppose $poss_{\phi(\overline{x},\overline{z})}(\overline{a},\overline{c}) \wedge poss_{\psi(\overline{y},\overline{z})}(\overline{b},\overline{c})$ is true. Then there must be some model that satisfies $\phi(\overline{a},\overline{c})$ and some model that satisfies $\psi(\overline{b},\overline{c})$. By the independence of $\phi(\overline{x},\overline{z})$

and $\psi(\overline{y},\overline{z})$, since $\phi(\overline{a},\overline{c})$ is consistent with Δ and $\psi(\overline{b},\overline{c})$ is consistent with Δ, $\phi(\overline{a},\overline{c})\wedge\psi(\overline{b},\overline{c})$ is consistent with Δ. Hence, $poss_{\phi(\overline{x},\overline{z})\wedge\psi(\overline{y},\overline{z})}(\overline{a},\overline{b},\overline{c})$ is true.

Second we assume equivalence and show that $\phi(\overline{a},\overline{c})$ and $\psi(\overline{b},\overline{c})$ are independent. Suppose $poss_{\phi(\overline{x},\overline{z})\wedge\psi(\overline{y},\overline{z})}(\overline{a},\overline{b},\overline{c})$ is equivalent to $poss_{\phi(\overline{x},\overline{z})}(\overline{a},\overline{c})\wedge poss_{\psi(\overline{y},\overline{z})}(\overline{b},\overline{c})$. We must show that if $\phi(\overline{a},\overline{c})$ and $\psi(\overline{b},\overline{c})$ are individually consistent with Δ then their conjunction is also consistent with Δ.

So suppose they are individually consistent with Δ. That is, $poss_{\phi(\overline{x},\overline{z})}(\overline{a},\overline{c})$ and $poss_{\psi(\overline{y},\overline{z})}(\overline{b},\overline{c})$ are both true. Then by the equivalence above, we know that certainly $poss_{\phi(\overline{x},\overline{z})\wedge\psi(\overline{y},\overline{z})}(\overline{a},\overline{b},\overline{c})$ is true. This guarantees that $\phi(\overline{a},\overline{c})\wedge\psi(\overline{b},\overline{c})$ is consistent with Δ, which means $\phi(\overline{a},\overline{c})$ and $\psi(\overline{b},\overline{c})$ are independent.

Because we have shown the theorem holds for all instances, by Herbrand semantics, we have proven the theorem. □

In the text, the Syntactic Independence theorem appears next, but because its proof is a natural corollary to a theorem that appears later, it is proven in Appendix A.3.3.

Theorem 13 (Completeness Guarantees Independence). *Let Δ be a satisfiable set of FHL sentences complete with respect to $\phi(\overline{x})$, i.e. for every tuple of object constants $\overline{a}$, $\Delta\models\phi(\overline{a})$ or $\Delta\models\neg\phi(\overline{a})$. Then $\phi(\overline{x})$ is independent of every other sentence ψ with respect to Δ.*

Proof. Suppose that both $\phi(\overline{a})$ and ψ are individually consistent with Δ. Then there must be a model M that satisfies $\phi(\overline{a})$ and a model N that satisfies ψ. Because Δ entails either $\phi(\overline{a})$ or $\neg\phi(\overline{a})$ and M satisfies $\phi(\overline{a})$, Δ must entail $\phi(\overline{a})$ (for M is a countermodel to $\Delta\models\neg\phi(\overline{a})$). Therefore, every model that satisfies Δ must satisfy $\phi(\overline{a})$, including N. Hence, N must satisfy both $\phi(\overline{a})$ and ψ. Since the argument was made for an arbitrary $\overline{a}$, it holds for all $\overline{a}$, and by Herbrand semantics, $\phi(\overline{x})$ and ψ must therefore be independent with respect to Δ. □

Theorem 14 (Completeness and *poss* Redundancy). *Let Δ be a satisfiable set of FHL sentences complete with respect to $\phi(\overline{x})$, i.e. for every tuple of object constants $\overline{a}$, $\Delta\models\phi(\overline{a})$ or $\Delta\models\neg\phi(\overline{a})$. Then*

$$\Delta\cup\{poss\text{-}\textsc{Definition}(\phi(\overline{x}),\Delta)\}\models poss_{\phi(\overline{x})}(\overline{x})\Leftrightarrow\phi(\overline{x})^{1}$$

[1] Assuming *poss*-DEFINITION is a conservative extension of Δ, i.e. every model that satisfies Δ is the reduct of some model that satisfies $\Delta\cup\{poss\text{-}\textsc{Definition}(\phi(\overline{x}),\Delta)\}$.

Proof. Consider an arbitrary model M of $\Delta \cup \{poss\text{-}\textsc{Definition}(\phi(\overline{x}), \Delta)\}$.

($\Rightarrow$) Suppose $\models_M poss_{\phi(\overline{x})}(\overline{a})$. Then there must be a model of Δ that satisfies $\phi(\overline{a})$. Because Δ entails either $\phi(\overline{a})$ or $\neg\phi(\overline{a})$ and one of its models satisfies the former, it must be the case that $\Delta \models \phi(\overline{a})$. Certainly then, M satisfies $\phi(\overline{a})$ and therefore $poss_{\phi(\overline{x})}(\overline{a}) \Rightarrow \phi(\overline{a})$.

($\Leftarrow$) Suppose $\models_M \phi(\overline{a})$. This guarantees there is a model of Δ that satisfies $\phi(\overline{a})$. Consequently, $poss_{\phi(\overline{x})}(\overline{a})$ must be entailed by $\{poss\text{-}\textsc{Definition}(\phi(\overline{x}), \Delta)\}$, which ensures it must be true in every model, including M. Thus, M satisfies $poss_{\phi(\overline{x})}(\overline{a}) \Leftarrow \phi(\overline{a})$.

Because we have shown equivalence for an arbitrary model and an arbitrary instance, we have shown the equivalence for all models and all instances. And because the theorem has been proven for all instances of the theorem, by Herbrand semantics, we have proven the theorem. □

Definition 9 (Nondisjunctive Existential Sentence). *The sentence $\exists x.\phi(x, \overline{y})$ is said to be a nondisjunctive existential sentence with respect to Δ if and only if for every tuple of object constants $\overline{b}$*

$$\Delta \models \exists x.\phi(x, \overline{b}) \text{ implies there is some } a \text{ such that } \Delta \models \phi(a, \overline{b})$$

Theorem 15 (*poss* distributes over $\forall$). *In FHL, $poss_{\forall x.\phi(x,\overline{y})}(\overline{y})$ is equivalent to $\forall x.poss_{\phi(x,\overline{y})}(x, \overline{y})$, where Δ is the background theory, exactly when the sentence $\exists x.\neg\phi(x, \overline{y})$ is a nondisjunctive existential sentence with respect to Δ.*

Proof. Consider any instance of the theorem. $poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$ is equivalent to $\forall x.poss_{\phi(x,\overline{y})}(x, \overline{b})$ exactly when the sentence $\exists x.\neg\phi(x, \overline{b})$ is a nondisjunctive existential sentence. We proceed by showing that $poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$ always implies $\forall x.poss_{\phi(x,\overline{y})}(x, \overline{b})$. Then we show that the statement $\exists x.\neg\phi(x, \overline{b})$ is a nondisjunctive existential is equivalent to saying that $\forall x.poss_{\phi(x,\overline{y})}(x, \overline{b})$ implies $poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$. Finally, because the nondisjunctive existential condition is equivalent to the conjunction of these two implications, we conclude that the nondisjunctive existential condition is necessary and sufficient to distribute *poss* across $\forall$.

First we show that $poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$ always implies $\forall x.poss_{\phi(x,\overline{y})}(x, \overline{b})$. If we suppose the premise, then there is some model that satisfies $\forall x.\phi(x, \overline{b})$. Certainly then this model satisfies $\phi(a, \overline{b})$ for all object constants a; consequently, $poss_{\phi(x,\overline{y})}(a, \overline{b})$ is true for every a and therefore that $\forall x.poss_{\phi(x,\overline{y})}(x, \overline{b})$ is true.

Next we show that the statement $\exists x.\neg\phi(x,\overline{b})$ is a nondisjunctive existential sentence is equivalent to saying that $\forall x.poss_{\phi(x,\overline{y})}(x,\overline{b})$ implies $poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$. Suppose $\exists x.\neg\phi(x,\overline{b})$ is a nondisjunctive existential sentence.

$$\Delta \models \exists x.\neg\phi(x,\overline{b}) \text{ implies there is an } a \text{ such that } \Delta \models \neg\phi(a,\overline{b})$$

First we rewrite the antecedent.

$$\begin{aligned}
&\Delta \models \exists x.\neg\phi(x,\overline{b}) \\
&\Leftrightarrow \text{ for all models } M \ \models_M \exists x.\neg\phi(x,\overline{b}) \\
&\Leftrightarrow \text{ there is no model } M \text{ such that } \models_M \forall x.\phi(x,\overline{b}) \\
&\Leftrightarrow \neg poss_{\forall x.\phi(x,\overline{y})}(\overline{b})
\end{aligned}$$

Then we rewrite the consequent.

$$\begin{aligned}
&\text{there is an } a \text{ such that } \Delta \models \neg\phi(a,\overline{b}) \\
&\Leftrightarrow \text{ there is an } a \text{ such that for all models } M \ \models_M \neg\phi(a,\overline{b}) \\
&\Leftrightarrow \text{ there is an } a \text{ such that there is no model } M \text{ where } \models_M \phi(a,\overline{b}) \\
&\Leftrightarrow \text{ there is an } a \text{ such that } \neg poss_{\phi(x,\overline{y})}(a,\overline{b}) \\
&\Leftrightarrow \exists x.\neg poss_{\phi(x,\overline{y})}(x,\overline{b})^{\dagger} \\
&\Leftrightarrow \neg\forall x.poss_{\phi(x,\overline{y})}(x,\overline{b})
\end{aligned}$$

(†) Since there is an a such that $\neg poss_{\phi(x,\overline{y})}(a,\overline{b})$ holds, we can conclude that $\exists x.\neg poss_{\phi(x,\overline{y})}(x,\overline{b})$ holds; moreover, because every *poss* predicate has a complete definition, the two statements are logically equivalent.

Putting these two rewritings together ensures that $\neg poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$ implies $\neg\forall x.poss_{\phi(x,\overline{y})}(x,\overline{b})$. The contrapositive of that statement says that $\forall x.poss_{\phi(x,\overline{y})}(x,\overline{b})$ implies $poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$.

We have shown that the nondisjunctive existential condition c is equivalent to $\forall x.poss_{\phi(x,\overline{y})}(x,\overline{b})$ implies $poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$, which we denote $a \Rightarrow b$. We have also shown that the converse of the implication $poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$ implies $\forall x.poss_{\phi(x,\overline{y})}(x,\overline{b})$ unconditionally holds: $a \Leftarrow b$. Thus, c clearly implies $a \Leftrightarrow b$ and $a \Leftrightarrow b$ is sufficient for concluding c since $a \Rightarrow b$ is sufficient for concluding c.

Thus $\forall x.poss_{\phi(x,\overline{y})}(x,\overline{b})$ and $poss_{\forall x.\phi(x,\overline{y})}(\overline{b})$ are logically equivalent exactly when $\exists x.\neg\phi(x,\overline{b})$ is a nondisjunctive existential sentence.

Because we have shown the theorem to hold for all instances, by Herbrand semantics, we have proven the theorem. □

A.3.3 *poss* Definition Construction

In this section, we prove the theorems relating to the algorithms for constructing *poss* predicate definitions. In the text, we first give an algorithm QF-POSS for constructing a *poss* definition for quantifier-free sentences and state that the crucial property that QF-POSS exploits is given by the Ground Conjunctive Inconsistency lemma. Then we show that we can improve the algorithm in two ways: one by restricting the literals resolved upon and the other by partitioning the theory based on dependency information. It turns out that this partitioning of the theory is a model-theoretic property, and hence is applicable to any proof system. Because of its generality, we start with its proof, and then proceed to the proof of the Ground Conjunctive Inconsistency lemma and its corollary.

Dependence Decomposition

The first theorem allows us to partition or decompose a theory based on which predicates are dependent on which other predicates while using the fact that dependence cannot flow through complete definitions. It allows us to conclude that if we build a semantic dependency graph, which allows us to ignore the predicates with complete definitions, then the connected-components of that graph correspond to the dependent predicates of the theory.

The first lemma is borrowed from a paper by Reiter[Rei80], but we state it here as a lemma because of its importance. It tells us how to use resolution to reason about sets of sentences in FHL. Because it will be important to differentiate resolution with and without the paramodulation inference rule, we will denote resolution using just binary resolution and factoring as RES; if we want to include paramodulation as well, we will write RES+PARA.

Lemma 11 (RES for FHL). *Let Δ be a set of FHL clauses, UNA be the set of unique names axioms for the object constants in Δ, including both $a \neq b$ and $b \neq a$ for every pair of distinct object constants, and R be all ground instances of $x = x$. Δ is unsatisfiable in FHL if and only if $\text{RES}(\Delta \cup UNA \cup R)$ includes the empty clause.*

Proof. See Reiter's paper on the closed world assumption: [Rei80]. □

This lemma is important because it allows us to use resolution without paramodulation to prove properties about sets of clauses in FHL. An important piece of the proof of the

Dependency Decomposition theorem is the use of Algorithm 10, UNIT-SIMPLIFICATION, which allows us to take a complete ground theory and an incomplete ground theory and remove all occurrences of complete predicates from the incomplete theory. The algorithm is a simple combination of subsumption and unit resolution applied to ground clauses.

Algorithm 10 UNIT-SIMPLIFICATION(C, Δ)

Assumes: C is a set of ground literals and Δ is a set of ground clauses

1: {Subsumption}
2: **when** $\pm p(\overline{a}) \cup \Phi \in \Delta$ and $\pm p(\overline{a}) \in C$ **then** remove $\pm p(\overline{a}) \cup \Phi$ from Δ
3: {Unit Resolution with back subsumption}
4: **while** no changes **do**
5: **when** $\pm p(\overline{a}) \cup \Phi \in \Delta$ and $\mp p(\overline{a}) \in C$ **then** replace $\pm p(\overline{a}) \cup \Phi$ with Φ in Δ
6: **end while**
7: **return** Δ

This algorithm preserves logical equivalence (even under FOL semantics); moreover, it does what it was designed to do: remove all occurrences of predicates with complete definitions.

Lemma 12 (UNIT-SIMPLIFICATION preserves Equivalence). *Suppose C is a finite set of ground literals, and Δ is a finite set of clauses in FHL.*

$$C \cup \text{UNIT-SIMPLIFICATION}(C,\Delta) \textit{ is first-order logically equivalent to } C \cup \Delta$$

Proof. Each application of subsumption preserves logical equivalence because the only change in the sentence set is a removal of a sentence that is entailed by another. Thus, the first-order models that satisfy the sentences before subsumption are exactly those that satisfy the sentences after subsumption.

Each step of resolution likewise preserves first-order models because resolution is sound: the only sentences added are entailed by the sentences given.

Since UNIT-SIMPLIFICATION is a combination of resolution and subsumption, it preserves first-order logical equivalence. □

Lemma 13 (UNIT-SIMPLIFICATION removes all complete predicates). *Suppose C is a finite, satisfiable, complete set of ground literals for predicates P, and Δ is a finite set of ground clauses in FHL.* UNIT-SIMPLIFICATION(C,Δ) *results in a set of sentences with no mention of P.*

Proof. Consider any mention of a predicate $p \in P$ in Δ. Because Δ is ground, it must be a ground literal: $p(\overline{a})$ or $\neg p(\overline{a})$. Since C is a complete set of ground literals, either $p(\overline{a})$ or $\neg p(\overline{a})$ is included in C. If the occurrence of p in Δ is included in C then the entire clause is removed; otherwise, resolution is used to eliminate the p literal from the clause. Thus, UNIT-PROPAGATION eliminates this literal, and since the literal was chosen arbitrarily, the same holds for all such literals, and the conclusion holds. □

Another property that will be important is that UNIT-SIMPLIFICATION distributes across sets of sentences, i.e. applying it to a set of sentences is the same as applying it to each sentence individually and combining the results.

Lemma 14 (Distribution of UNIT-SIMPLIFICATION). UNIT-SIMPLIFICATION(C, $\Delta_1 \cup \cdots \cup \Delta_n$) *is equivalent to*

$$\text{UNIT-SIMPLIFICATION}(C,\Delta_1) \cup \cdots \cup \text{UNIT-SIMPLIFICATION}(C,\Delta_n).$$

Proof. The inference rules used in UNIT-SIMPLIFICATION only operate on a single clause besides those in C. Thus, the result of every operation is the same whether UNIT-SIMPLIFICATION is given all of $\Delta_1 \cup \cdots \cup \Delta_n$ at once or only some piece of it. The result of UNIT-SIMPLIFICATION is simply the union of all those operations. Thus, the lemma actually states something about how UNIT-SIMPLIFICATION can be implemented. □

After repeating the definition for a semantic dependency graph, which ignores predicates with complete definitions, we prove the ground clausal version of the dependence decomposition theorem. Afterwards, we generalize it to the ground but non-clausal version, and finally we lift that to the non-ground, non-clausal version.

Definition 8 (Semantic Dependency Graph). *A Semantic Dependency Graph for a set of sentences Δ is a graph $\langle V, E \rangle$.*

- *$u \in V$ if and only if u is a predicate in Δ without a complete definition*
- *The undirected edge (u, v) is in E if and only if there is some sentence in Δ that uses both the predicates u and v.*

The following results rely on the notion of a connected-component from graph theory. Recall that a connected-component consists of all nodes in a graph such that there is a

path from one node to the other. A graph can have multiple connected components if it is composed of subgraphs that are disjoint. The graph in our case is the Semantic Dependency Graph, where nodes correspond to predicates and connected components correspond to sets of predicates.

Lemma 15 (Ground Clausal Dependence Decomposition). *Let Δ be a set of ground, satisfiable clauses in FHL and G be its Semantic Dependency Graph whose connected-components are $O_1, \ldots, O_m$. Suppose Δ is partitioned into $C \cup I_1 \cup \cdots \cup I_m$, where C corresponds to the complete portion of the theory, and each I_i corresponds to all the sentences mentioning some predicate in O_i. Suppose further that ϕ is a ground sentence whose predicates are a subset of those in $C \cup I_{j_1} \cup \cdots \cup I_{j_k}$, where $j_l \in \{1, \ldots, m\}$ for every l.*

$$\Delta \models \phi \text{ if and only if } C \cup I_{j_1} \cup \cdots \cup I_{j_k} \models \phi$$

Proof. ($\Leftarrow$) Immediate since if a subset of Δ entails ϕ, so must Δ entail ϕ.

($\Rightarrow$) Suppose $\Delta \models \phi$. Then $C \cup I_1 \cup \cdots \cup I_m \models \phi$. Then $C \cup I_1 \cup \cdots \cup I_m \cup \{\neg\phi\}$ is FHL unsatisfiable. Use P to denote the clausal form of $\neg\phi$, which, because $\neg\phi$ is ground, is logically equivalent to $\neg\phi$. Also, use C' to denote the set of all ground literals FHL-entailed by C. C' and C are logically equivalent under FHL semantics. Thus, we see that

$$C' \cup I_1 \cup \cdots \cup I_m \cup P \text{ is FHL unsatisfiable.}$$

Since C' includes the unique names axioms and all instances of $x = x$, by the RES for FHL lemma,

$$\text{RES}(C' \cup I_1 \cup \cdots \cup I_m \cup P) \text{ contains the empty clause.}$$

Since resolution is sound,

$$C' \cup I_1 \cup \cdots \cup I_m \cup P \text{ is FOL unsatisfiable.}$$

Using UNIT-SIMPLIFICATION, we can remove the complete literals from the incomplete portion of the theory while preserving logical equivalence.

$$C' \cup \text{UNIT-SIMPLIFICATION}(C',\ I_1 \cup \cdots \cup I_m \cup P) \text{ is FOL unsatisfiable.}$$

Then distribute UNIT-SIMPLIFICATION across the I_l while preserving equivalence. Denote

UNIT-SIMPLIFICATION(C',I_l) as I'_l for all l; likewise, denote UNIT-SIMPLIFICATION(C',P) with P'. This ensures that

$$C' \cup I'_1 \cup \cdots \cup I'_m \cup P' \text{ is FOL unsatisfiable.}$$

Since UNIT-PROPAGATION preserves logical equivalence and $C' \cup I_1 \cup \cdots \cup I_m$ was satisfiable, so is $C' \cup I'_1 \cup \cdots \cup I'_m$. Thus, we can employ set-of-support resolution to produce a proof of the empty clause and therefore unsatisfiability where $C' \cup I'_1 \cup \cdots \cup I'_m$ is the background and P' is the set of support. We will denote this by writing

$$\text{SOS}(C' \cup I'_1 \cup \cdots \cup I'_m,\ P') \text{ contains the empty clause.}$$

Since UNIT-PROPAGATION eliminates all mention of the complete literals from the incomplete portion of the theory, by pure literal elimination, we can remove C' without losing any proofs. Thus

$$\text{SOS}(I'_1 \cup \cdots \cup I'_m,\ P') \text{ contains the empty clause.}$$

Consider any proof of the empty clause. We claim that the only predicates that ever appear in the set-of-support are those in the connected components $O_{j_1} \cup \cdots \cup O_{j_k}$. This claim is useful because by construction these predicates are only mentioned in the sentences $I_{j_1} \cup \cdots \cup I_{j_k}$ in the original theory, and since UNIT-SIMPLIFICATION adds no predicate to any sentence, the only mention of predicates $O_{j_1} \cup \cdots \cup O_{j_k}$ in $I'_1 \cup \cdots \cup I'_m$ occurs in $I'_{j_1} \cup \cdots \cup I'_{j_k}$. Thus, the claim implies that the proof of the empty clause is produced using only $I'_{j_1} \cup \cdots \cup I'_{j_k} \cup P'$. Assuming the claim, we know that

$$\text{SOS}(I'_{j_1} \cup \cdots \cup I'_{j_k},\ P') \text{ contains the empty clause.}$$

We also claim that it is straightforward to take any proof of the empty clause from $I'_{j_1} \cup \cdots \cup I'_{j_k} \cup P'$ and construct a proof of the empty clause from $C' \cup I_{j_1} \cup \cdots \cup I_{j_k} \cup P$.

Once we have demonstrated this claim, the proof concludes as follows.

$\text{SOS}(C' \cup I_{j_1} \cup \cdots \cup I_{j_k}, P)$ contains the empty clause.
implies $C' \cup I_{j_1} \cup \cdots \cup I_{j_k} \cup P$ is unsatisfiable. (By soundness of SOS)
implies $C' \cup I_{j_1} \cup \cdots \cup I_{j_k} \models_{FOL} \neg \wedge P$
implies $C' \cup I_{j_1} \cup \cdots \cup I_{j_k} \models_{FOL} \phi$ (Since P is the clausal form of $\neg\phi$)
implies $C \cup I_{j_1} \cup \cdots \cup I_{j_k} \models_{FOL} \phi$ (Since C entails C' under FOL semantics.)
implies $C \cup I_{j_1} \cup \cdots \cup I_{j_k} \models_{FHL} \phi$ (Since $\models_{FOL}$ is a subset of $\models_{FHL}$.)

This completes the proof, except for the two claims made above.

Claim 1. Consider any sequence of resolutions in $\text{SOS}(C' \cup I'_1 \cup \cdots \cup I'_m, P')$, where P' is the set of sentences in the SOS to start. We must show that the only predicates mentioned are those in $O_{j_1} \cup \cdots \cup O_{j_k}$: the predicates that represent the connected components in the semantic dependency graph of $I_1 \cup \cdots \cup I_m$ that include all the predicates mentioned in P. By induction on the number of resolutions, we show that the only predicates mentioned in clauses participating in a resolution and occur in the SOS are those from $O_{j_1} \cup \cdots \cup O_{j_k}$.

Base case: 0 resolutions. The SOS is initialized to include simply the clauses P, which by construction mention a subset of the predicates in $O_{j_1} \cup \cdots \cup O_{j_k}$.

Inductive step: Assume that for n resolutions, the SOS includes only predicates contained in $O_{j_1} \cup \cdots \cup O_{j_k}$. Show that the next resolution only applies to clauses that mention predicates in $O_{j_1} \cup \cdots \cup O_{j_k}$, which guarantees that the SOS will mention only those predicates. Recall that set-of-support resolution requires one of the premises to be drawn from the SOS; thus, by the inductive hypothesis, that clause must mention only predicates from $O_{j_1} \cup \cdots \cup O_{j_k}$, which ensures that the literal resolved upon must mention one of those predicates: p. If the other clause belongs to the SOS, clearly both premises mention only predicates from $O_{j_1} \cup \cdots \cup O_{j_k}$ and so too does the resolvent. So suppose the other clause does not belong to the SOS.

Because it resolves with a p literal, p must be mentioned in that clause. Consider q, any other predicate mentioned in the clause. In the semantic dependency graph of $I'_1 \cup \cdots \cup I'_m$, there must be an edge between p and q, as we have just found the sentence that mentions both, and neither can be complete predicates since none occur in $I'_1 \cup \cdots \cup I'_m$. Because I'_l is UNIT-SIMPLIFICATION(C',I_l), and UNIT-SIMPLIFICATION adds no predicates and removes only complete predicates, the semantic dependency graph of $I'_1 \cup \cdots \cup I'_m$ must be the same as that of $I_1 \cup \cdots \cup I_m$; thus, there must be an edge between p and q in the dependency graph

of $I_1 \cup \cdots \cup I_m$. Consequently, p and q belong to the same connected component, and since p belongs to $O_{j_1} \cup \cdots \cup O_{j_k}$ so must q. Since the argument holds for an arbitrary predicate q, it holds for all such predicates. Thus all predicates in the two premises are drawn from $O_{j_1} \cup \cdots \cup O_{j_k}$, and consequently, all the predicates in the resolvent and therefore all the predicates in the SOS are drawn from $O_{j_1} \cup \cdots \cup O_{j_k}$.

This completes the proof by induction, which demonstrates that every resolution mentions only predicates from $O_{j_1} \cup \cdots \cup O_{j_k}$; consequently, any proof of the empty clause mentions only those predicates.

Claim 2. Consider any proof of the empty clause from $I'_{j_1} \cup \cdots \cup I'_{j_k} \cup P'$. We must show how to construct a proof of the empty clause from $C' \cup I_{j_1} \cup \cdots \cup I_{j_k} \cup P$. Consider any premise Φ' in the proof, which must belong to either P' or I'_{j_l} for some l. By the construction of P' and I'_{j_l}, this premise is the result of applying some number of unit resolutions to some clause Φ in P or I_{j_l}, respectively, with literals $\{l_1, \ldots, l_n\}$ from C'.

$$\frac{\begin{array}{c}\Phi \\ l_1 \text{ (from } C')\end{array}}{\Phi_1} \quad \frac{\begin{array}{c}\Phi_1 \\ l_2 \text{ (from } C')\end{array}}{\Phi_2} \quad \cdots \quad \frac{\begin{array}{c}\Phi_n \\ l_n \text{ (from } C')\end{array}}{\Phi'}$$

Thus, every premise in the proof is a resolvent of $C' \cup I_{j_1} \cup \cdots \cup I_{j_k} \cup P$. So, to construct the proof of the empty clause from $C' \cup I_{j_1} \cup \cdots \cup I_{j_k} \cup P$, start with the proof of the empty clause from $I'_{j_1} \cup \cdots \cup I'_{j_k} \cup P'$ and prepend all the derivations for the premises of that proof. □

The above result also holds for ground sentences that are not written in clausal form because converting ground sentences to clausal form preserves equivalence. The only complication arises from the fact that the dependency graph of the nonclausal sentences is not necessarily the same as the dependency graph for the sentences after clausification.

Lemma 16 (Ground Dependence Decomposition). *Let Δ be a set of ground, satisfiable sentences in FHL and G be its Semantic Dependency Graph whose connected-components are $O_1, \ldots, O_m$. Suppose Δ is partitioned into $C \cup I_1 \cup \cdots \cup I_m$, where C corresponds to the complete portion of the theory, and each I_i corresponds to all the sentences mentioning some predicate in O_i. Suppose further that ϕ is a ground sentence whose*

predicates are a subset of those in $C \cup I_{j_1} \cup \cdots \cup I_{j_k}$*, where* $j_l \in \{1, \ldots, m\}$ *for every* l*.*

$$\Delta \models \phi \text{ if and only if } C \cup I_{j_1} \cup \cdots \cup I_{j_k} \models \phi$$

Proof. ($\Leftarrow$) Since a subset of Δ entails ϕ, so must Δ.

($\Rightarrow$) Suppose $\Delta \models \phi$. Because Δ is ground, it is logically equivalent to a set of ground clauses Δ', which also entails ϕ. Because Δ' is clausal, the Ground Clausal Dependency Decomposition lemma can be applied. Suppose the dependency graph for Δ' has connected-components $O'_1, \ldots, O'_l$, and the corresponding partitioning of Δ' is $C' \cup I'_1 \cup \cdots \cup I'_l$. Then the Ground Clausal Dependence Decomposition lemma ensures that $C \cup I'_{j_1} \cup \cdots \cup I'_{j_p} \models \phi$, where the predicates mentioned in ϕ belong to the partitions $I'_{j_1} \cup \cdots \cup I'_{j_p}$. This does not immediately allow us to conclude that $C \cup I_{j_1} \cup \cdots \cup I_{j_k} \models \phi$ because the dependency graphs of Δ and Δ' may be different, leading to different decompositions of the theory.

To prove the lemma, we show that $C \cup I_{j_1} \cup \cdots \cup I_{j_k}$ entails $C' \cup I'_{j_1} \cup \cdots \cup I'_{j_p}$, which guarantees that $C \cup I_{j_1} \cup \cdots \cup I_{j_k} \models \phi$.

First we show that the predicates in $O'_{j_1} \cup \cdots \cup O'_{j_p}$ (the connected-components from the dependency graph of the clausal Δ' that include all the predicates mentioned in ϕ) are a subset of those in $O_{j_1} \cup \cdots \cup O_{j_k}$ (the connected-components from Δ that mention the predicates of ϕ). To see this, observe that the dependency graph for Δ', G', is a subgraph of the dependency graph for Δ, G. (For any sentence ψ in Δ, the graph G connects any two predicates mentioned in ψ. Converting ψ to clausal form constructs several sentences from ψ without introducing new predicates; consequently, no edges belong to G' that did not belong to G, but some edges that belong to G might not belong to G'.) $O'_{j_1} \cup \cdots \cup O'_{j_p}$ is the minimal set of connected-components from G' that include all the predicates mentioned in ϕ. Because G' is a subgraph of G, any two predicates connected in G' are also connected in G. Thus, the predicates in G' connected to the predicates mentioned in ϕ must also be connected in G to the predicates mentioned in ϕ. Thus, every predicate in $O'_{j_1} \cup \cdots \cup O'_{j_p}$ is also in $O_{j_1} \cup \cdots \cup O_{j_k}$.

Now, consider any sentence ρ in $I'_{j_1} \cup \cdots \cup I'_{j_p}$. It must be the result of converting some sentence ψ from Δ to clausal form. Because clausal form introduces no new predicates, the predicates mentioned in ρ must be a subset of those in ψ. Since ψ mentions at least one of the predicates in ρ, it mentions one of the predicates in $O'_{j_1} \cup \cdots \cup O'_{j_p}$ and therefore one of the predicates in $O_{j_1} \cup \cdots \cup O_{j_k}$; thus, ψ must belong to $I_{j_1} \cup \cdots \cup I_{j_k}$. Since the argument

holds for an arbitrary ρ, every sentence in $I'_{j_1} \cup \cdots \cup I'_{j_p}$ is the result of clausifying some sentence in $I_{j_1} \cup \cdots \cup I_{j_k}$; thus, the latter entails the former. Moreover, C logically entails C', which means $C \cup I_{j_1} \cup \cdots \cup I_{j_k}$ entails $C' \cup I'_{j_1} \cup \cdots \cup I'_{j_p}$. Since $C' \cup I'_{j_1} \cup \cdots \cup I'_{j_p} \models \phi$ we therefore conclude that $C \cup I_{j_1} \cup \cdots \cup I_{j_k} \models \phi$. □

Because in FHL, every sentence set is logically equivalent to a ground sentence set, and grounding a sentence set does not change the Semantic Dependency Graph, the theorem holds for non-ground sentences.

Theorem 16 (Dependence Decomposition). *Let Δ be a set of satisfiable sentences in FHL and G its Semantic Dependency Graph whose connected-components are $O_1, \ldots, O_m$. Suppose Δ is partitioned into $C \cup I_1 \cup \cdots \cup I_m$, where C corresponds to the complete portion of the theory, and each I_i corresponds to all the sentences mentioning some predicate in O_i. Suppose further that ϕ is a sentence whose predicates are a subset of those in $C \cup I_{j_1} \cup \cdots \cup I_{j_k}$, where $j_l \in \{1, \ldots, m\}$ for every l.*

$$\Delta \models \phi \text{ if and only if } C \cup I_{j_1} \cup \cdots \cup I_{j_k} \models \phi$$

Proof. ($\Leftarrow$) If a subset of Δ entails ϕ, so does Δ.

($\Rightarrow$) The crux of this direction is the commutativity of two operations: (1) partitioning a theory based on the connected components of its semantic dependency graph and (2) grounding a theory using the following rewrite rules.

$$\forall x.\phi(x) \text{ becomes } \phi(a_1) \wedge \cdots \wedge \phi(a_n)$$
$$\exists x.\phi(x) \text{ becomes } \phi(a_1) \vee \cdots \vee \phi(a_n)$$

That is, whether we first partition Δ based on its dependency graph and then ground the result or we ground Δ and then partition based on the grounded Δ's dependency graph, the result is the same partitioning of ground sentences. Assuming this commutativity holds, the proof proceeds as follows.

Suppose $\Delta \models \phi$. Because Δ and ϕ are in FHL, they are logically equivalent to their groundings: Δ' and ϕ', respectively. Consequently,

$$\Delta' \models \phi'.$$

The partitions $I_{j_1}, \ldots, I_{j_k}$ of Δ based on the connected-components of its dependency graph

include all those sentences of Δ that mention a predicate in ϕ. Grounding each of those partitions produces $I'_{j_1}, \ldots, I'_{j_k}$. Because the predicates mentioned in ϕ and ϕ' are the same, we can leverage commutativity to conclude that these partitions are exactly the same partitions that result by first grounding Δ to produce Δ' and then partitioning Δ' using its dependency graph and the predicates mentioned in ϕ'. The Ground Dependency Decomposition lemma ensures that

$$C' \cup I'_{j_1}, \ldots, I'_{j_k} \models \phi'.$$

where C' denotes the grounding of C. Since ϕ' is logically equivalent to ϕ, C' is equivalent to C, and I'_{j_l} is equivalent to I_{j_l} for every l,

$$C \cup I_{j_1}, \ldots, I_{j_k} \models \phi,$$

which gives us the conclusion of the theorem.

All that remains is proving that partitioning a theory based on the connected components of its dependency graph and grounding a theory using the above rules are commutative. First, notice that the dependency graphs of Δ and the grounding of Δ' are the same. The dependency graph only reflects which predicates occur in sentences with which other predicates, and the above rewrite rules do not change that: each rewriting performs a combination of subsentence replication and variable instantiation, neither of which introduce new predicates nor split apart predicates that occur in the same sentence.

These properties of grounding also guarantee that grounding some sentence ψ produces a single sentence, ψ', which is identical to the original in terms of the predicates mentioned. Thus, if ψ is a member of I_h because it mentions a subset of the predicates in connected-component O_h, then ψ' must belong to I'_h because it too mentions a subset of the predicates in O_h. Since this holds for all ψ, the grounding of I_h gives I'_h for all h. Thus, whether grounding is performed and the result is partitioned using the semantic dependency graph, or the theory is partitioned using that same semantic dependency graph and the result is then grounded, the result is the same, and hence the operations are commutative. □

The next theorem appears for the first time in Chapter 3, but its proof is a corollary to the previous theorem. Recall that *poss* distributes over conjunction if the conjuncts are independent. Syntactic Independence gives us an inexpensive way to check whether two

sentences are independent.

Definition 7 (Sentence Independence). *Let Δ be a sentence set in FHL and $\phi(\overline{x},\overline{z})$ and $\psi(\overline{y},\overline{z})$ be sentences whose vocabularies are subsets of the vocabulary of Δ that share the variables $\overline{z}$. $\phi(\overline{x},\overline{z})$ and $\psi(\overline{y},\overline{z})$ are independent with respect to Δ if and only if for every tuple of object constants $\overline{t}$, $\overline{u}$, and $\overline{v}$, if $\phi(\overline{t},\overline{v})$ is consistent with Δ and $\psi(\overline{u},\overline{v})$ is consistent with Δ then $\phi(\overline{t},\overline{v}) \wedge \psi(\overline{u},\overline{v})$ is consistent with Δ.*

Theorem 12 (Syntactic Independence). *Let Δ be a satisfiable set of FHL sentences, and let G be its Semantic Dependency Graph. Suppose that in G each of the predicates used in $\phi(\overline{x},\overline{z})$ is in a different connected component than every predicate in $\psi(\overline{y},\overline{z})$. Then $\phi(\overline{x},\overline{z})$ and $\psi(\overline{y},\overline{z})$ are independent with respect to Δ.*

Proof. Assume the condition in the theorem. Suppose that the connected-components of G are $O_1,\ldots,O_m$, and Δ is partitioned into $C \cup I_1 \cup \cdots \cup I_m$, where C corresponds to the complete portion of the theory, and each I_i corresponds to all the sentences mentioning some predicate in O_i. If $\Delta \cup \{\phi(\overline{a},\overline{b})\}$ is satisfiable then by the soundness of resolution and grounding, we know that running resolution to closure on the grounding of Δ with the grounding of $\phi(\overline{a},\overline{b})$ does not include the empty clause. Likewise, if $\Delta \cup \{\psi(\overline{c},\overline{b})\}$ is satisfiable then resolution applied to its grounding does not include the empty clause. For the purpose of contradiction suppose that $\Delta \cup \{\phi(\overline{a},\overline{b}) \wedge \psi(\overline{c},\overline{b})\}$ were unsatisfiable, i.e. that $\Delta \models \neg\phi(\overline{a},\overline{b}) \vee \neg\psi(\overline{c},\overline{b})$. Then by Dependence Decomposition,

$$C \cup I_{\phi_1} \cup \cdots \cup I_{\phi_m} \cup I_{\psi_1} \cup \cdots \cup I_{\psi_k} \models \neg\phi(\overline{a},\overline{b}) \vee \neg\psi(\overline{c},\overline{b})$$

where $\{I_{\phi_1} \cup \cdots \cup I_{\phi_m}\}$ is the minimal set of I_j to which all the predicates in ϕ belong; likewise for $\{I_{\psi_1} \cup \cdots \cup I_{\psi_k}\}$ and ψ. Thus,

$$\text{RES}(\text{GROUNDING}(C \cup I_{\phi_1} \cup \cdots \cup I_{\phi_m} \cup I_{\psi_1} \cup \cdots \cup I_{\psi_k} \cup \{\phi(\overline{a},\overline{b}), \psi(\overline{c},\overline{b})\}))$$

must include the empty clause.

As shown before, we can eliminate all predicates with complete definitions in C using UNIT-SIMPLIFICATION while preserving satisfiability. The resulting clause set can be partitioned into two sets: those from I_{ϕ_j} for some j or $\phi(\overline{a},\overline{b})$, and those from I_{ψ_j} for some j or $\psi(\overline{c},\overline{b})$. By the conditions on the theorem and the way the I_{ϕ_j} and I_{ψ_j} were constructed,

those two sets of sentences share no predicates; consequently, no resolution can be performed across the sets. Thus, in order to generate the empty clause one of the two sets must do it by themselves, but that would contradict the fact that both $\phi(\overline{a}, \overline{b})$ and $\psi(\overline{c}, \overline{b})$ are individually consistent with Δ.

Since we have chosen the ground instances of $\phi(\overline{x}, \overline{z})$ and $\psi(\overline{y}, \overline{z})$ arbitrarily, the result holds for all instances, which ensures $\phi(\overline{x}, \overline{z})$ and $\psi(\overline{y}, \overline{z})$ are independent. □

Proof Theoretic Approach

In this section we prove certain properties about the proof theoretic algorithms for constructing complete definitions for *poss* predicates.

The hard part in these proofs comes about because of the two-phase approach to query answering. First, the system performs some reformulation (rewriting the entailment query in terms of *poss* and then constructing definitions for *poss* predicates), and then it invokes a database engine to answer the query just constructed. Proving the properties of interest therefore requires that both phases are modeled mathematically. Our proofs use resolution for both purposes—the complete theory resolution inference rule (CTR) introduced in Section 4.1 models the database evaluation, and the usual resolution inference rules model the compilation.

We now prove two properties about resolution. The first property just establishes that resolution is equivalence preserving when it is sound and complete.

Lemma 17 (RES Equivalence Preservation). *Suppose Δ is a set of clauses. If* RES *is sound and complete, then* RES(Δ) *is logically equivalent to Δ.*

Proof. Soundness guarantees that no model is lost since at each step, the consequences are always weaker than the premises (the premises entail the consequences). Completeness guarantees that no model can be gained since otherwise an unsatisfiable set of clauses would at some point turn into a satisfiable set of clauses. □

Lemma 18 (RES Distribution). *Consider the finite clause set $\Delta = \Delta_1 \cup \Delta_2$, and suppose that* RES($\Delta_2$) *is finite.* RES($\Delta_1 \cup \Delta_2$) *contains the empty clause if and only if* RES($\Delta_1 \cup$ RES(Δ_2)) *includes the empty clause.*

Proof. Since RES(Δ_2) is logically equivalent to Δ_2, using the lemma above, we know $\Delta_1 \cup \Delta_2$ is unsatisfiable if and only if $\Delta_1 \cup$ RES(Δ_2) is unsatisfiable. And since RES(Δ_2) is finite, so

must $\Delta_1 \cup \text{RES}(\Delta_2)$ be finite. Therefore $\text{RES}(\Delta_1 \cup \Delta_2)$ includes the empty clause
if and only if $\Delta_1 \cup \Delta_2$ is unsatisfiable
if and only if $\Delta_1 \cup \text{RES}(\Delta_2)$ is unsatisfiable
if and only if $\text{RES}(\Delta_1 \cup \text{RES}(\Delta_2))$ includes the empty clause, which is guaranteed by the soundness and completeness of RES and the finiteness of $\Delta_1 \cup \text{RES}(\Delta_2)$. □

This means that if we can break Δ into two pieces such that the closure of resolution on one is finite, we can first run resolution to closure on that piece, and then run resolution to closure on the result plus what remains. For example, we can split Δ into the complete portion and the incomplete portion, run resolution on the incomplete portion first and then run resolution on the result (assuming it is finite) together with the complete portion, while preserving soundness and completeness.

This observation is important because when constructing the *poss* definitions using the proof theoretic approach, we did so while completely ignoring the complete portion of the theory. We can model that mathematically by construing the algorithm as first applying resolution to the incomplete portion and then applying resolution to the result and the complete portion.

Moreover, as shown in Section 4.1, complete theory resolution (CTR) can be used to model the special case of resolution applied to a complete theory. Next we show that if we have already applied resolution to closure on the incomplete portion of the theory, and then we want to apply resolution to the complete portion plus the saturated incomplete portion, the only rule of inference needed is CTR, i.e. we do not need the inference rules for resolution, paramodulation, or factoring.

Corollary 1 (CTR Simplification). *Let $\Delta = C \cup I$ be a finite set of clauses, where C is a satisfiable, complete set of ground literals, and $\text{RES}(I)$ is finite. $\text{RES}(C \cup I)$ contains the empty clause if and only if $\text{CTR}(C, \text{RES}(I))$ contains the empty clause.*

Proof. $\text{RES}(C \cup I)$ contains the empty clause if and only if $\text{RES}(C \cup \text{RES}(I))$ contains the empty clause, by the RES distribution lemma. $\text{RES}(C \cup \text{RES}(I))$ contains the empty clause if and only if $\text{CTR+RES}(C, \text{RES}(I))$ contains the empty clause by the soundness and completeness of CTR+RES theory resolution. Now, because $\text{RES}(I)$ is saturated, there are no resolutions that can be performed among its clauses. Thus, when CTR+RES begins to apply the various inference rules to $\text{RES}(I)$, the only applicable one is CTR. Moreover, each

application of CTR removes a literal from a clause, producing a shorter, possibly more constrained clause. These resulting clauses therefore admit no new inference rule applications besides CTR. Consequently, we know that in this case, CTR+RES simplifies to CTR. Hence, RES($C \cup I$) contains the empty clause if and only if CTR(C, RES(I)) contains the empty clause. □

Next we prove the lemma that guarantees this two-phase approach to reasoning can be used to preserve the usual fact about resolution: a conjunction of literals is inconsistent with a $\forall^*$ theory exactly when it is inconsistent with one of the clauses in the resolution closure of the theory.

Lemma 2 (Ground Conjunctive Inconsistency). *Suppose $C \cup I$ is a finite set of FHL sentences, where C is a satisfiable, complete theory, I is a set of clauses, and the resolution closure of I is finite. $C \cup I$ is inconsistent with the ground conjunction*

$$[\neg]p_1(\overline{a_1}) \wedge \cdots \wedge [\neg]p_n(\overline{a_n})$$

if and only if either CTR+RES(C, $\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) *contains the empty clause or there is some clause $d \in$* RES(I) *such that* CTR(C, RES($\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}$)) *contains the empty clause.*

Proof. First, suppose that we say C' is the set of all ground literals entailed by C. C' is logically equivalent to C in FHL; therefore, we know that $C \cup I \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$ is unsatisfiable in FHL if and only if $C' \cup I \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$ is unsatisfiable. Notice that C must entail unique names axioms for every pair of object constants in the language, and it must entail all instances of $x = x$. Thus by the RES for FHL lemma we see that $C' \cup I \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$ is unsatisfiable if and only if RES($C' \cup I \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) contains the empty clause. Then, by distributing RES using the fact that RES(I) is finite, we see that this resolution closure contains the empty clause if and only if RES($C' \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\} \cup$ RES(I)) contains the empty clause. It therefore suffices to show that RES($C' \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\} \cup$ RES(I)) contains the empty clause if and only if either CTR+RES(C, $\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) contains the empty clause or there is some clause $d \in$ RES(I) such that CTR(C, RES($\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}$)) contains the empty clause.

($\Rightarrow$) Suppose RES($C' \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\} \cup$ RES(I)) contains the empty clause.

If the proof of the empty clause mentions no d in $\text{RES}(I)$ then
$\text{RES}(C' \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\})$ contains the empty clause and therefore (by the completeness of CTR+RES) $\text{CTR+RES}(C', \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\})$ contains the empty clause.

Suppose then that the proof of the empty clause uses some clause d in $\text{RES}(I)$, i.e. d is an ancestor of the empty clause. Just as argued in the CTR Simplification lemma, because $\text{RES}(I)$ is saturated, the only resolutions that can be performed on such a clause are with literals from C' or one of the $[\neg]p_i(\overline{a_i})$. Thus, there must be a proof of the empty clause using just one of the clauses from $\text{RES}(I)$. $\text{RES}(C' \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\})$ must therefore include the empty clause, which using the CTR+RES completeness and CTR simplification allows us to conclude that $\text{CTR}(C', \text{RES}(\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}))$ must include the empty clause.

Notice that in both these cases it is irrelevant whether we use C or C' since
$\text{CTR+RES}(C,\Delta)$ only cares about what ground literals are entailed; hence, in both the consequences above, we can replace C' by C.

($\Leftarrow$) By the soundness of CTR and resolution, if either
$\text{CTR+RES}(C, \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\})$ includes the empty clause or there is some clause $d \in \text{RES}(I)$ such that $\text{CTR}(C, \text{RES}(\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}))$ includes the empty clause, then $C \cup I$ is inconsistent with the ground conjunction $[\neg]p_1(\overline{a_1}) \wedge \cdots \wedge [\neg]p_n(\overline{a_n})$.

□

Finally we show that even if we restrict resolution so that it never resolves on a predicate with a complete definition when computing $\text{RES}(I)$, the same result holds.

Lemma 3 (Restricted Ground Conjunctive Inconsistency). *The Ground Conjunctive Inconsistency lemma holds even when instead of computing* $\text{RES}(I)$*, we compute the resolution closure of* I *but never resolve on a predicate with a complete definition in* C*.*

Proof. We denote the resolution closure where the only resolutions occur on predicates without complete definitions by RES_I.

As soundness is immediate, here we just show a patch of the above proof for completeness. To show: if $\text{RES}(C' \cup I \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\})$ contains the empty clause then either $\text{CTR+RES}(C', \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\})$ contains the empty clause or there is some clause $d \in \text{RES}_I(I)$ such that $\text{CTR}(C',\text{RES}(\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}))$ contains the empty clause.

Suppose RES($C' \cup I \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) contains the empty clause. Using Herbrand's theorem, we can ground I to produce I_g and result is still unsatisfiable: RES($C' \cup I_g \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) contains the empty clause. Using the UNIT-SIMPLIFICATION equivalence preservation lemma of the last section, we can remove all those clauses in I_g that contain one of the literals in C', i.e. use unit subsumption, while preserving logical equivalence. Call the result of unit subsumption I'_g. Consequently, we know that RES($C' \cup I'_g \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) contains the empty clause. Using the lemma above, we can conclude that either CTR+RES(C', $\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) contains the empty clause or there is some clause $d \in$ RES(I'_g) such that CTR(C', RES($\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}$)) contains the empty clause.

Notice that I'_g does not include both a ground atom and its negation from any of the complete predicates, since one of them belongs to C' and all clauses mentioning it have been removed by unit subsumption. Therefore, d must have been produced without resolving on one of the complete predicates. Consequently, since $d \in$ RES(I'_g), we know that $d \in$ RES$_I$(I'_g).

Consider the proof of d from I'_g using RES$_I$. Since I'_g is a subset of I_g, every resolution that occurred to produce d in I'_g can also occur in I_g. And therefore, we see that if there is some clause $d \in$ RES$_I$(I'_g) such that CTR(C', RES($\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}$)) contains the empty clause then d is also in RES$_I$(I_g) and CTR(C', RES($\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}$)) contains the empty clause.

We can then apply Robinson's lifting lemma [Rob65] to conclude that for each clause e in the proof of d from I_g, there is a clause e' produced from I such that e is an instance of e'. This includes a clause d' such that d is an instance. Since d' is no more constrained than d, every resolution that occurs in RES($\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}$) can also occur if d is replaced by d'.

Since CTR is just a sequence of resolution steps, all those applications of CTR can also be applied when d' takes the place of d. Thus, we can conclude that there must be a clause $d' \in$ RES$_I$(I) such that CTR(C',RES($\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d'\}$)) contains the empty clause.

Putting this together with the case where CTR+RES(C', $\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) contains the empty clause, we see that if RES($C' \cup I \cup \{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) contains the empty clause then either CTR+RES(C', $\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n})\}$) contains the empty clause or there is some clause $d \in$ RES$_I$(I) such that

$$\text{CTR}(C',\text{RES}(\{[\neg]p_1(\overline{a_1}), \ldots, [\neg]p_n(\overline{a_n}), d\}))$$

contains the empty clause. □

A.3.4 Extensional Reasoning with Incomplete Theories

Here we show that the algorithm on page 87, FULL-ER-ENTAILEDP, for answering entailment queries in incomplete theories uses a form of entailment-preserving theory-completion.

Lemma 4 (Entailment Preservation of FULL-ER-ENTAILEDP). *Under the following conditions, after executing lines (1) through (4), $\Delta \models \phi$ if and only if $\Delta' \models \phi'$, where Δ' is a complete, satisfiable theory.*

- *ϕ is a closed sentence*
- TO-BICONDS-MAX(Δ) *returns a $C \cup I$ that is logically equivalent to Δ, and where C is a complete, satisfiable theory.*
- SIMPLIFY-POSS($\neg poss_{\neg\phi}$, *cpreds*, I) *returns a sentence that is logically equivalent to $\neg poss_{\neg\phi}$ according to the semantics of poss and C.*
- POSS-DEFINITIONS(POSS-PREDS(ϕ'), *cpreds*, I) *returns a complete, satisfiable theory consistent with C such that each of the poss predicates that appear in ϕ' are defined according to the semantics of poss for the background theory $C \cup I$.*

Proof. Suppose we have an infinite, satisfiable, complete theory Γ such that for every sentence $\phi(\overline{x})$ in the language it entails a definition for $poss_{\phi(\overline{x})}(\overline{x})$: $poss_{\phi(\overline{x})}(\overline{a})$ is entailed if and only if $\phi(\overline{a})$ is consistent with Δ. Γ axiomatizes the semantics of all the *poss* predicates with background theory Δ. Here we assume that the predicates that appear in Γ are disjoint from the predicates that appear in Δ.

Because of this, we know that $\Gamma \models poss_{\neg\phi}$ if and only if $\neg\phi$ is consistent with Δ. This ensures that $\Gamma \models \neg poss_{\neg\phi}$ if and only if $\neg\phi$ is inconsistent with Δ. And this holds exactly when $\Delta \models \phi$. Thus we see that

$$\Delta \models \phi \text{ if and only if } \Gamma \models \neg poss_{\neg\phi}.$$

Since the predicates in Γ are disjoint from those in Δ, they are also disjoint from the predicates that appear in C; hence, $C \cup \Gamma$ is satisfiable and complete. Because SIMPLIFY-POSS returns a sentence ϕ' that is logically equivalent to $\neg poss_{\neg\phi}$ under the semantics of

poss and C, we can conclude

$$\Delta \models \phi \text{ if and only if } C \cup \Gamma \models \phi'.$$

We also know POSS-DEFINITIONS constructs a satisfiable, complete theory Λ that consists of definitions for all the *poss* predicates in ϕ'; moreover, Λ and C are consistent. Thus between Λ and C, there is a complete definition for every predicate in ϕ'; moreover, $\Lambda \cup C = \Delta'$ is guaranteed to be consistent and complete. We also know that Λ must define the *poss* predicates the same way Γ does. Consequently, since every predicate in ϕ' has the same definition in $C \cup \Gamma$ as it does in Δ', we see that

$$\Delta \models \phi \text{ if and only if } \Delta' \models \phi'$$

□

Bibliography

[AB96] Gerard Allwein and Jon Barwise. *Logical Reasoning with Diagrams*. Oxford University Press, 1996.

[ABH00] Gilles Audemard, Belaid Benhamou, and Laurent Henocque. Two techniques to improve finite model search. In *Proceedings of the 17th International Conference on Automated Deduction*, 2000.

[ACN00] Sanjay Agrawal, Surajit Chaudhuri, and Vivek Narasayya. Automated selection of materialized views and indexes in Microsoft SQL Server. In *Proceedings of Very Large Databases (VLDB)*, pages 496–505, 2000.

[AG93] A. Armando and E. Giunchiglia. Embedding complex decision procedures inside an interactive theorem prover. *Annals of Mathematics and Artificial Intelligence*, 8(3-4):475–502, 1993.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley Publishing Company, 1995.

[AK91] Hassan Ait-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press, 1991.

[Ami01] Eyal Amir. *Dividing and Conquering Logic*. PhD thesis, Stanford University, 2001.

[AS92] Owen Astrachan and Mark Stickel. Caching and lemmaizing in model elimination theorem provers. In *Proceedings of the Conference on Automated Deduction*, 1992.

[BFNT07] Peter Baumgartner, Alexander Fuchs, Hans Nivelle, and Cesare Tinelli. Computing finite models by reduction to function-free clause logic. *Journal of Applied Logic*, 2007.

[BG00] Ruth Byrne and Lisa Gilroy. Mental models website. `http://www.tcd.ie/Psychology/Ruth_Byrne/mental_models/index.html`, 2000.

[BGG97] Egon Borger, Erich Gradel, and Yuri Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.

[BPB95] Dave Barker-Plummer and Sidney Bailin. Proofs and pictures: Proving the diamond lemma with the grover theorem proving system. In *Proceedings of the AAAI Symposium on Reasoning with Diagrammatic Representations*, 1995.

[BPT97] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized view selection in a multidimensional database. In *Proceedings of the Conference on Very Large Databases*, pages 155–165, 1997.

[BT03] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In *Proceedings of the 19th International Conference on Automated Deduction*, pages 350–364, 2003.

[Bun73] Alan Bundy. Doing arithmetic with diagrams. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1973.

[Bur90] Hans-Jurgen Burckert. A resolution principle for clauses with constraints. In *Proceedings of the 10th International Conference on Automated Deduction*, pages 178–192, 1990.

[BvBW06] Patrick Blackburn, Johan van Benthem, and Frank Wolter. *Handbook of Modal Logic*. Elsevier Science, 2006.

[CD97] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4):137–150, 1997.

[CDLS00] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. Preprocessing of intractable problems. *Information and Computation*, 176(2):89–120, 2000.

[CGLH04] S. Craig, J. Gallagher, M. Leuschel, and K. Henriksen. Fully automatic binding-time analysis for Prolog. *Proceedings of the 14th International Symposium on Logic Based Program Synthesis and Transformation (LOPSTR)*, pages 53–68, 2004.

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

[Cho00] Seungyeob Choi. Semantically guided proof planning. `http://citeseer.ist.psu.edu/464153.html`, 2000.

[CL73] Chin-Liang Chang and Richard Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[CS03] K. Claessen and N. Sorensson. New techniues that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop on Model Computation*, 2003.

[Dav94] Ernest Davis. Lucid representations. Technical report, New York University, 1994.

[Dig79] V. J. Digricoli. Resolution by unification and equality. In *Proceedings of the 4th Workshop on Automated Deduction*, pages 43–52, 1979.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A computing procedure for quantification theory. *Communications of the ACM*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. A machine program for theorem proving. *Journal of the ACM*, 7:201–215, 1960.

[dV96] Alvaro del Val. Approximate knowledge compilation: The first order case. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 498–503, 1996.

[EBBK89] David Etherington, Alex Borgida, Ronald Brachman, and Henry Kautz. Vivid knowledge and tractable reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1146–1152, 1989.

[EF99] Heinz-Dieter Ebbinghaus and Jorg Flum. *Finite Model Theory*. Springer-Verlag, 1999.

[End00] Herbert Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2000.

[Fit07] Melvin Fitting. Intensional logic. *The Stanford Encyclopedia of Philosophy (Spring 2007 Edition), Edward N. Zalta (ed.)*, 2007.

[FT04] Enrico Franconi and Sergio Tessaris. Rules and queries with ontologies: A unified logical framework. In *Proceedings of Principles and Practice of Semantic Web Reasoning*, 2004.

[GA07] Igor Gammer and Eyal Amir. Solving satisfiability in ground logic with equality by efficient conversion to propositional logic. In *Proceedings of the 7th Symposium on Abstraction, Reformulation, and Approximation*, 2007.

[Ged95] Don Geddis. *Caching and Non-Horn Inference in Model Elimination Theorem Provers*. PhD thesis, Stanford University, 1995.

[Gel63] H. Gelernter. Realization of a geometry-theorem proving machine. *Computers and Thought*, pages 134–152, 1963.

[GHVD03] Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of the 12th International Conference on the World Wide Web*, 2003.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, 1988.

[GS99] Enrico Giunchiglia and Roberto Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas. In *Proceedings of the 6th Congress of the Italian Association for Artificial Intelligence*, pages 84–94, 1999.

[Hal60] Paul Halmos. *Naive Set Theory*. Van Nostrand Reinhold Company, 1960.

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

[HG05] Timothy Hinrichs and Michael Genesereth. Axiom schemata as metalevel axioms: Model theory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2005.

[HG06] Timothy Hinrichs and Michael Genesereth. Herbrand logic. Technical report, Stanford University, 2006.

[HHK+05] Pascal Hitzler, Peter Haase, Markus Krotzsch, York Sure, and Rudi Studer. DLP isn't so bad after all. In *Proceedings of the WS OWL – Experiences and Directions*, 2005.

[HI99] Takashi Horiyama and Toshihide Ibaraki. Ordered binary decision diagrams as knowledge-bases. In *Proceedings of the International Symposium on Algorithms and Computation*, 1999.

[HK93] Peter Hammer and Alexander Kogan. Optimal compression of propositional horn knowledge bases: Complexity and approximation. *Artificial Intelligence*, 64(1):131–145, 1993.

[HV03] S. Heymans and D. Vermeir. Integrating semantic web reasoning and answer set programming. In *Proceedings of the 2nd International ASP Workshop*, 2003.

[JGP99] Edmund M. Clarke Jr., Orna Grumberg, and Doron Peled. *Model Checking*. The MIT Press, 1999.

[JJD98] Daniel Jackson, Somesh Jha, and Craig Damon. Isomorph-free model enumeration: a new method for checking relational specifications. *ACM Transactions on Programming Languages and Systems*, 20(2):302–343, 1998.

[JL87] J. Jaffar and J.L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[Kha95] Roni Khardon. Translating between Horn representations and their characteristic models. *Journal of Artificial Intelligence Research*, 3:349–372, 1995.

[KHSV05] Markus Krotzsch, Pascal Hitzler, Michael Sintek, and Denny Vrandecic. Expressive OWL reasoning. Technical report, University of Kalrsruhe, 2005.

[KKS93] Henry Kautz, Michael Kearns, and Bart Selman. Reasoning with characteristic models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1993.

[KKS95] Henry Kautz, Michael Kearns, and Bart Selman. Horn approximations of empirical data. *Artificial Intelligence*, 74(1):129–145, 1995.

[KR94] Roni Khardon and Dan Roth. Reasoning with models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1994.

[KS91] Henry Kautz and Bart Selman. A general framework for knowledge compilation. In *Proceedings of the International Workshop on Processing Declarative Knowledge*, 1991.

[KS92] Henry Kautz and Bart Selman. Forming concepts for fast inference. In *Proceedings of the ECAI-Workshop on Knowledge Representation and Reasoning*, pages 200–215, 1992.

[LB02] Michael Leuschel and Maurice Bruynooghe. Logic program specialisation through partial deduction: Control issues. *Theory and Practice of Logic Programming*, 2(4-5):461–515, 2002.

[Lev86] Hector J. Levesque. Making believers out of computers. *Artificial Intelligence*, 30(1):81–108, 1986.

[LG90] D.B. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley, 1990.

[LGBH04] P. Lopez-Garcia, F. Bueno, and M. Hermenegildo. Determinacy analysis for logic programs using mode and type information. In *Proceedings of the 14th International Symposium on Logic Based Program Synthesis and Transformation (LOPSTR)*, pages 19–35, 2004.

[Lif02] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

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

[LMR92] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunctive Logic Programming*. The MIT Press, 1992.

[LT84] J. Lloyd and R. Topor. Making Prolog more expressive. *Journal of Logic Programming*, 1(3):225–240, 1984.

[MB88] Rainer Manthey and Francois Bry. SATCHMO: A theorem prover implemented in Prolog. In *Proceedings of the Conference on Automated Deduction*, pages 415–434, 1988.

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

[McC88] John McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1988.

[McC01] William McCune. MACE 2.0 reference manual and guide. `http://citeseer.ist.psu.edu/464179.html`, 2001.

[McC03] William McCune. Mace4 reference manual and guide. Technical report, Argonne National Laboratory, 2003.

[McI98] Sheila McIlraith. Logic-based abductive inference. Technical report, Stanford University, 1998.

[MG03] James Masters and Zelai Gungordu. Semantic knowledge source integration: A progress report. In *Proceedings of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, 2003.

[Mor69] James B. Morris. E-resolution: Extension of resolution to include the equality relation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 287–294, 1969.

[MR07] Boris Motik and Riccardo Rosati. A faithful integration of description logics with logic programming. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2007.

[MS06] R. Muhammed and P.J. Stuckey. A stochastic non-CNF SAT solver. In *Proceedings 9th Biennial Pacific Rim International Conference on Artificial Intelligence*, pages 120–129, 2006.

[Mye90a] Karen Myers. Automatically generating universal attachments through compilation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1990.

[Mye90b] Karen Myers. *Universal Attachments: A Logical Framework for Hybrid Reasoning.* PhD thesis, Stanford University, 1990.

[NP01] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the International Conference on Formal Ontology in Information Systems*, pages 2–9, 2001.

[PAE98] Brian Peterson, William Anderson, and Joshua Engel. Knowledge bus: Generating application-focused databases from large ontologies. In *Knowledge Representation Meets Databases*, 1998.

[PAH04] G. Puebla, E. Albert, and M. Hermenegildo. Efficient local unfolding with ancestor stacks for full Prolog. In *Proceedings of the 14th International Symposium on Logic Based Program Synthesis and Transformation (LOPSTR)*, pages 1–18, 2004.

[Pas78] D. Pastre. Automatic theorem proving in set theory. *Artificial Intelligence*, 10(1):1–27, 1978.

[PBG05] Mukul Prasad, Armin Biere, and Aarti Gupta. A survey of recent advances in SAT-based formal verification. *Software Tools for Technology Transfer*, 7(2):156–173, 2005.

[Pel00] Nicolas Peltier. Model building with ordered resolution. In *Proceedings of the International Workshop on First Order Theorem Proving (FTP)*, 2000.

[PP80] Pereira and Porto. Selective backtracking for logic programs. In *Proceedings of the Conference on Automated Deduction*, 1980.

[PZ97] David Plaisted and Yunshan Zhu. Ordered semantic hyper linking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1997.

[RA05] D. Ramachandran and E. Amir. Compact propositional encodings of first-order theories. In *Proceedings of the 20th AAAI Conference on Artificial Intelligence*, 2005.

[Rei73] Ray Reiter. A semantically guided deductive system for automatic theorem-proving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1973.

[Rei78] Ray Reiter. On closed world databases. In *Proceedings of 1978 ACM SIGMOD International Conference on Management of Data*, 1978.

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

[Rob65] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

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

[Sch02] S. Schulz. A comparison of different techniques for grounding near-propositional CNF formulae. In *Proceedings of the 15th International Florida Artificial Intelligence Research Society Conference*, pages 72–76, 2002.

[Sik96] Vishal Sikka. *Integrating Specialized Procedures into Proof Systems*. PhD thesis, Stanford University, 1996.

[Sip96] Michael Sipser. *Introduction to the Theory of Computation*. Brooks Cole, 1996.

[SK91] Bart Selman and Henry Kautz. Knowledge compilation using horn approximations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1991.

[Sla93] John Slaney. SCOTT: A model-guided theorem prover. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1993.

[SS98a] G. Sutcliffe and C.B. Suttner. The TPTP problem library: CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[SS98b] Christian Suttner and Geoff Sutcliffe. The CADE-14 ATP system competition. *Journal of Automated Reasoning*, 21(1):99–134, 1998.

[SSW94] K. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 442–453, 1994.

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

[Tam01] Tanel Tammet. Finite model building: improvements and comparisons. `http://citeseer.ist.psu.edu/675660.html`, 2001.

[TBW04] Christian Thiffault, Fahiem Bacchus, and Toby Walsh. Solving non-clausal formulas with DPLL search. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pages 663–678, 2004.

[Ull89] Jeffrey Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1989.

[Var82] Moshe Vardi. The complexity of relational query languages. In *Proceedings of the 14th annual ACM symposium on Theory of Computing*, pages 137–146, 1982.

[vGRS91] Allen van Gelder, Kenneth Ross, and John Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[VV98] Sergei Vorobyov and Andrei Voronkov. Complexity of nonrecursive logic programs with complex values. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 244–253, 1998.

[Wan85] Tie Cheng Wang. Designing examples for semantically guided hierarchical deduction. In *Proceedings of the International Joint Conference on Arttificial Intelligence*, 1985.

[Wey80] Richard Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.

[Zha96] J. Zhang. Constructing finite algebras with FALCON. *Journal of Automated Reasoning*, 17(1):1–22, 1996.

[ZZ95] J. Zhang and H. Zhang. SEM: A system for enumerating models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.