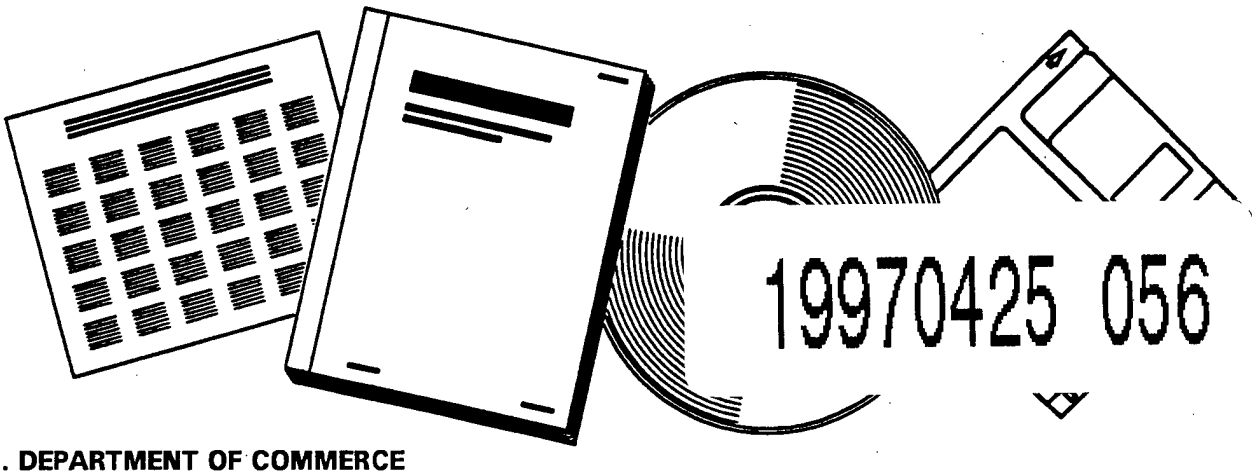
PB96-146006

UPDATING DATABASES WITH INCOMPLETE INFORMATION

STANFORD UNIV., CA

JAN 87



U.S. DEPARTMENT OF COMMERCE
National Technical Information Service

January 1987

Report No. STAN-CS-87-1143

Updating Databases with Incomplete Information

by

Marianne S. Winslett

Department of Computer Science

Stanford University
Stanford, CA 94305

UPDATING DATABASES WITH INCOMPLETE INFORMATION

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

By
Marianne Southall Winslett
December 1986

In memory of my mother, Virginia Custis Winslett

ACKNOWLEDGMENTS

All three members of my reading committee have played vital roles in the development of this dissertation. I would not have gotten past year one without the encouragement of Christos Papadimitriou. My claims would have been flimsy indeed without the emphasis on rigor of Moshe Vardi. And I would long ago have been derailed from my course without the academic acumen of Gio Wiederhold to illuminate the potholed road of scholarly life.

When the members of the KBMS project fell under Gio's edict to produce conference papers in the summer of 1983, Arthur Keller got me started on the topic of incomplete information when we wrote a paper together, a most productive collaboration. A year later incomplete information became my dissertation topic as well, and I monopolized Christos' spare moments unmercifully for quite a time thereafter with prattle of "marked nothings" and "empty facts". More recently, Christos suggested storing rather than executing expensive updates as a strategy for cost reduction, a line of attack that proved most fruitful.

In the fall of 1985 Moshe Vardi joined my reading committee. Moshe's advising technique has been to vehemently insist that he doesn't know what I'm talking about. By the time I explain something to his satisfaction, there is a solid theorem and proof in the explanation. Moshe's liberal use of punctuation ("?") on drafts has led to a much more comprehensible presentation. He has pointed out numerous errors and omissions, and provided invaluable discussion on many points.

Conversations with Devika Subramanian made me realize that this work would also be of interest to the artificial intelligence world; but I would never have been able to format my results in a form palatable to that community without the AI perspective and suggestions of David C. Wilkins. Many of the results presented in this thesis have appeared elsewhere [Winslett 86abc], and David redlined many a draft of the original articles. He also read the final draft of this thesis and compiled the index of definitions. Mark Manasse provided logic expertise during the creation of principle P5 of Chapter 8. My officemate, Peter K. Rathmann, helped with the $\mathcal{NP}$-completeness results in Chapter 6, and wisely left town as the thesis due date approached.

I am indebted to AT&T Bell Laboratories, especially Andy Salazar, Herb Burton, Charlie Roberts, Mark Rochkind, and Bob Lucky, for encouragement to leave their employ and come back to school; and also for the AT&T Bell Laboratories fellowship provided during my years of graduate study. Additional support was provided by DARPA under a series of grants to the Knowledge Based Management Systems projects, with Gio Wiederhold as principal investigator.

ABSTRACT

Suppose one wishes to construct, use, and maintain a database of facts about the real world, even though the state of that world is only partially known. In the artificial intelligence domain, this problem arises when an agent has a base set of beliefs that reflect partial knowledge about the world, and then tries to incorporate new, possibly contradictory knowledge into this set of beliefs. In the database domain, one facet of this situation is the well–known *null values* problem. We choose to represent such a database as a logical theory, and view the models of the theory as representing possible states of the world that are consistent with all known information.

How can new information be incorporated into the database? For example, given the new information that "b or c is true," how can one get rid of all outdated information about b and c, add the new information, and yet in the process not disturb any other information in the database? In current–day database management systems, the difficult and tedious burden of determining exactly what to add and remove from the database is placed on the user.

Our research has produced a formal method of specifying the desired change intensionally, by stating a well–formed formula that the state of the world is now known to satisfy. The database update algorithms we provide will automatically accomplish that change. Our approach embeds the incomplete database and the incoming information in the language of mathematical logic, and gives formal definitions of the semantics of our update operators, along with proofs of correctness for their associated algorithms. We assess the computational complexity of the algorithms, and propose a means of lazy evaluation to avoid undesirable expense during execution of updates. We also examine means of enforcing integrity constraints as the database is updated.

This thesis also examines the question of choices of semantics for update operators for databases with incomplete information, and proposes a framework for evaluation of competing choices of semantics. Several choices of semantics are evaluated with respect to that framework.

A experimental implementation of our method has been constructed, and we include the results of test runs on a range of patterns of queries and updates.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

Chapter 1: Introduction

How can new facts be added to a body of knowledge when the new facts may contradict preexisting information?

In the course of investigating this question, this dissertation gives partial answers to nagging questions in practical database work; in database theory and logical databases; and in artificial intelligence, particularly belief revision. In this introductory chapter we motivate the research from the perspectives of these three groups of readers, and conclude with a guide to the remainder of the thesis.

The majority of this thesis was written with a particular audience in mind: those who are comfortable with first–order logic and have a passing acquaintance with database updates. Tactical suggestions are offered below for readers with other backgrounds and interests.

From a traditional database perspective. A database management system faces two central tasks: evaluation of incoming queries and, quite separately, processing of incoming updates.

Much attention has been paid to the problem of answering queries in databases containing *null values*, or attribute values that are known to lie in a certain domain but whose value is currently unknown (see e.g. [Codd 79, Imielinski 84, Reiter 84, Vassiliou 79, Zaniolo 82]). There has been very little research on updating such databases, although, as one group of researchers aptly points out [Abiteboul 85], answering queries in databases containing nulls presupposes the ability to enter incomplete information into the database and, with any luck, to remove uncertainties when more information becomes available. Such a capability is needed not only in the case where the user directly requests the incorporation of uncertain values into the database, but also when updates indirectly spawn incomplete information, as in updating through views [Bancilhon 81, Dayal 82, Keller 82, 85] and in natural language updates [Davidson 81, 84].

As an example of the difficulties posed by even simple updates, suppose that we have the following two relations, containing one null value.

EMPLOYEE	DEPT	SALARY
Reid	?	30,000
Nilsson	CSD	40,000

MANAGER	DEPT
Nilsson	CSD

Suppose that the database user wishes to give all the the computer scientists a raise. Here is an expression of that update in a generic database manipulation language:

```
RANGE OF t IS EmpDeptSal

MODIFY t.SALARY TO BE t.SALARY*1.1
WHERE t.DEPT = ComputerScience
```

What happens to Reid's salary? How can we express the fact that Reid's salary depends on an unknown value in another field of the tuple, and how can that relationship be determined automatically?

Matters get more complicated if instead the user wishes to give Reid's boss a raise:

```
RANGE OF t IS EmpDeptSal
RANGE OF t2 IS EmpDeptSal
RANGE OF s IS ManDept

MODIFY t.SALARY TO BE t.SALARY*1.1
WHERE t2.EMP = Reid AND t2.DEPT = s.DEPT
AND s.MANAGER = t.EMP
```

What happens to Nilsson's salary? How can we express the fact that his salary depends upon an unknown value in a different relation, and how can that fact be derived automatically?

Unfortunately, although it is syntactically simple to allow null values in relational tables and update requests, any reasonable semantics for these updates will lead to result relations that cannot be stored as simple tables. Even with tight restrictions on the appearance of nulls, one quickly leaves the realm of the relational model, as in the example above. Our advice to a database management system designer operating under tight bounds of performance: don't try to treat a null value as anything more than an element in an ordinary domain with a few extra primitive operations, such as `ISNULL()`; otherwise naive users will drive processing costs up with their ill–advised updates, and will lay the blame on the wrong party.

For readers oriented toward practical database technology, the recommended route through this thesis is a quick trip through Chapter 3 followed by a tour of Chapter 9, the discussion of implementation.

From a logical databases perspective. Matters are less bleak if one is willing to cast aside the traditional relational restriction of databases to tables and instead view databases as simple, restricted theories in first–order logic with equality. This is the viewpoint adopted in this thesis, and readers who share this perspective should find themselves at home.

We use an extension of the logic framework set forth by Reiter [84, 84b] for the null value and disjunctive information problems. (Disjunctive information occurs when one knows that one or more of a set of tuples holds true, without knowing which one.) Given a relational database, Reiter shows how to construct a *relational theory* whose model corresponds to the world represented by the database, and extends this framework to allow disjunctive information and null values to appear in the relational theory. The use of an extension of Reiter's logic framework has four advantages: it allows a clean formalization of incomplete information; it allows a definition of the meanings of query and update operators without recourse to intuition or common knowledge; and it frees us from implicit or explicit consideration of implementation issues, by not forcing incomplete information into a tabular format. Through framing the update question in this paradigm, we will also gain insights into the more general problem of updating general logical theories, and lay groundwork for use in applications beyond ordinary databases, such as AI applications using a knowledge base built on top of base facts. We will show that in the logic paradigm it is natural to extend the concept of database updates to encompass databases with incomplete information.

From an artificial intelligence perspective. Suppose one wishes to construct, use, and maintain a knowledge base (KB) of beliefs about the real world, even though the facts about that world are only partially known. In the artificial intelligence (AI) domain, this problem arises when an agent has a base set of beliefs that reflect partial knowledge about the world, and then tries to incorporate new, possibly contradictory knowledge into this set of beliefs. We choose to represent such a KB as a logical theory, and view the models of the theory as representing possible states of the world that are consistent with the agent's beliefs.

How can new information be incorporated into the KB? For example, given the new information that "b or c is true," how can one get rid of all outdated information about b and c, add the new information, and yet in the process not disturb any other information in the KB? The burden may be placed on the user or other omniscient authority to determine exactly what to add and remove from the KB. But what's really needed is a way to specify the desired change intensionally, by stating some well-formed formula that the state of the world is now known to satisfy and letting the KB algorithms automatically accomplish that change. We investigate this problem for the simplest type of belief revision, that of bodies of ground beliefs with particularly simple axioms. In contrast, most work on belief revision by AI researchers focuses on the mechanisms for handling inference through axioms correctly (see e.g. [Doyle 79, McCarthy 80]).

The results of most interest to AI readers are collected in Winslett [86c]. In particular, the effect of removing the closed–world assumption [Lifschitz 85, Reiter 80] used in this dissertation is investigated there. Syntax, semantics, algorithms, and proofs of correctness are all presented for an open–world scenario. In addition, the long version of Winslett [86c] includes a more complete discussion of the enforcement of dependency axioms than is presented here. The AI–oriented reader will find Winslett [86c] to be the best introduction to this work, and then can browse through the offerings of these chapters at will.

Outline of the remaining chapters. Chapter 2 surveys work done by others that is related to the problem of updating databases with incomplete information.

The central ideas of this dissertation all appear in Chapter 3, which presents a logical language for the update problem, syntax and semantics for updates, and an algorithm for implementing the simplest types of updates. Chapter 4 extends these results to updates containing variables and quantifiers, and also gives an update algorithm for the case where the database, exclusive of higher–level rules and axioms, is any finite first–order theory containing quantifiers. Proofs of correctness for the update algorithms appear in Chapter 4, along with a discussion of computational complexity.

When null values appear in the database, updates can cause unacceptably large growth in the database when many data tuples "unify" with one another. Chapter 5 presents a lazy evaluation scheme coupled with simple user-supplied cost limits, used to avoid undesirable expense during execution of updates against databases that suffer from this unification problem. The goal of lazy evaluation is to delay execution of too–expensive updates as long as possible in the hopes that more information about the null values causing the expense will arrive in the interim. The techniques proposed have a strong flavor of database concurrency control.

There are many possible interpretations of the semantics of updates when additional general rules regarding the permissible states of and transformations on the database are considered along with the collection of base data. The "correct" interpretation of an update when such axioms are present depends on the intended semantics of the axiom. The study of artificial intelligence abounds with these phenomena; a simple example from the database arena is the choice of action when an integrity constraint would be violated by a requested update. Typically, a database management system either rejects the update or else makes additional changes in the database so that the constraint is still satisfied. Incomplete information complicates matters by providing additional reasonable roles for axioms that are not present in the complete–information case. These alternatives are studied in Chapter 6. After a discussion of these enforcement options, Chapter 6 shows how to provide what we call *strict* enforcement, for the class of universally quantified dependency axioms. Simple dependency axioms, such as type axioms, functional dependencies, and multi–valued dependencies, are easily strictly enforced.

Chapters 3 and 4 establish that it is computationally feasible to implement

updates when incomplete information is present, by presenting polynomial–time algorithms for a particular choice of update semantics, called the *standard* semantics. The standard semantics is by no means the only possible choice of semantics, however. Chapter 7 is devoted to a discussion of the properties of a number of candidate semantics for updates. A broad spectrum of possible semantics is identified there, and criteria of expressiveness, suitability for the application, comprehensibility, and computational feasibility are proposed for evaluating potential choices of semantics. Several points along that spectrum, including the standard semantics, are examined thoroughly with respect to those criteria. In addition, update algorithms are presented for two semantics other than the standard semantics, to show how our algorithmic approach can be extended to other choices of semantics.

Chapter 8 presents a series of results on update equivalence. Two updates U_1 and U_2 are equivalent if for any database, U_1 applied to that database produces the same result as does U_2 applied to that database. Update equivalence theorems are useful for clarifying the properties of candidates for update semantics. They provide one measure of comprehensibility for a particular choice of semantics: under that semantics, do two updates that look similar have the same effect on every database? Do updates that intuitively seem different produce different effects? Update equivalence theorems are given for the standard semantics and for the other semantics studied in Chapter 7. In addition, we provide several results on update equivalence that apply to a broad class of choices of semantics.

There is much to be learned in the reduction of a theory to practice. Clearly queries and updates will be more expensive in databases with incomplete information; how high might that extra cost be in a typical database scenario? Chapter 9 describes an implementation of the Update Algorithm of Chapter 4, and gives experimental results. The discussion focuses on the size of the stored database after a long series of updates that insert, reference, and remove incomplete information, and on the number of disk accesses required to answer a set of queries after that series of updates.

Conclusions are presented in Chapter 10, along with topics for future work.

Chapter 2: Related Work

When this research effort began, essentially no work had been done on the problem of updating (as opposed to querying) databases that contain incomplete information. The notable exception is the work of Fagin et al [83, 86]. Our work has a different motivation from that of Fagin et al, who were primarily concerned with applications such as updates through views and updates through integrity constraints. In such applications, one can attach importance to the particular formulas currently in the theory, such as view definitions or integrity constraints; and in fact Fagin et al take the formula as the primary unit of interest during update, producing a more syntactically oriented approach than our own. In contrast, our semantics is not concerned with the particular formulas in the theory being updated, but rather with the individual models of that theory. We define the semantics of an update by telling what effect the update should have on each model of the theory, independent of all other models. Unlike that of Fagin et al, the effect of an update in our paradigm is independent of the choice of formulas (other than schema and integrity constraints) used to represent that set of models. For example, let $\mathcal{T}_1$ be a database containing the formula Emp(Reid, CSD) $\wedge$ Emp(Nilsson, CSD), and let $\mathcal{T}_2$ be a database containing the two formulas Emp(Reid, CSD) and Emp(Nilsson, CSD). Then an update U might give different results when applied to $\mathcal{T}_1$ and $\mathcal{T}_2$ under the approach of Fagin et al. With our model–theoretic approach, however, U will give the same results when applied to the two databases.

One benefit of our approach is the feasibility of an efficient algorithm for update computation; this is not possible in the framework of Fagin et al. For example, Fagin et al define the deletion of a formula α from a first–order theory $\mathcal{T}$ as, roughly speaking, the set of all maximal subtheories of $\mathcal{T}$ that do not logically entail α. One cannot expect a polynomial–time procedure for testing whether α follows from a first–order theory. In contrast, our update algorithms require time linear in the size of the theory being updated.

As do Fagin et al, we identify multiple levels of formulas in a theory—axioms and non-axioms, in our case. However, we divide our axioms into different classes based on their intended semantics, and provide different sorts of algorithmic manipulations for the different classes under update. Fagin et al allow for an arbitrary number of levels of formulas, but do not note a need for different semantics at different levels or for certain formulas.

Our debt to Reiter [84, 84b] for his logical formulation of closed–world databases has already been mentioned. Reiter describes an encoding of databases containing disjunctive information and null values as first–order theories with

equality. His focus in this work is on algorithms for query evaluation over such databases.

Very recently, DeKleer [85] and Reiter [85] have investigated the problem of circuit diagnosis, formulated as the problem of updating a set of propositional formulas. They both take a logic theory describing the correct behavior of a circuit, and consider the problem of making minimal changes in that theory in order to make it consistent with a formula describing the circuit's observed behavior. This is closely allied to the problem we investigate (though circuit diagnosis does not require the use of selection clauses in update requests). However, the changes needed in the theory of the circuit are themselves the diagnosis of the circuit, and must be output to the user. As it is an $\mathcal{NP}$–hard problem just to determine whether any changes are needed at all in the circuit description (i.e., to do satisfiability testing and determine whether the circuit is functioning correctly), one cannot expect to find a polynomial–time algorithm for diagnosis. In contrast, we were particularly interested in producing polynomial–time algorithms to perform updates. The algorithms we present in Chapter 3 and subsequent chapters could be used for the circuit diagnosis problem only when the new "diagnosed" theory is of interest, rather than the exact changes made to the old theory.

In other recent work, Weber [86] takes a similar position on update semantics to that of DeKleer and Reiter, and provides an algorithm for implementing his update semantics for propositional theories. Extending his algorithm to first–order theories containing Skolem constants—that is, databases with null values—is not straightforward, however. Again, Weber does not consider updates with selection clauses or offer a polynomial–time algorithm for implementation.

Abiteboul and Grahne [Abiteboul 85] investigate the problem of updates on several varieties of *tables*, or relations containing null values and history constraints other than integrity constraints. They propose a semantics similar to our own for simple updates, and investigate the relationship between table type and ability to represent the result of an update correctly and completely. They do not consider updates with joins or disjunctions in selection clauses, comparisons between attribute values, or selection clauses referencing tuples other than the tuple being updated. Their conclusion was that only the most powerful and complex version of tables was able to fully support their update operators. Abiteboul and Grahne do not frame their investigation in the paradigm of mathematical logic, making their work less applicable to AI needs, one important application for this work.

In the AI realm, Levesque [84] considered the problem of updating knowledge bases with his TELL operation; however, TELL could only eliminate models from the set of models for the knowledge base, not change the internal contents of those models. In other words, one could only TELL the knowledge base new information that was consistent with what was already known. This is an important and vital function, but an agent also needs to be able to make changes in the belief set that contradict current beliefs [Harman 86]. For example, the agent should be able to change the belief that block A is on block B if, for example,

the agent observes an arm removing A from B.

Work on belief revision as pursued by researchers in artificial intelligence (see e.g. [Doyle 79, McCarthy 80]) typically focuses on the problem of how to obtain correct inferences from a set of axioms and base beliefs as the set of base beliefs itself undergoes revision. This approach assumes that a means is available of updating the base set of beliefs, and concentrates on the extremely difficult problem of revising derived beliefs correctly. However, we will show that when the base set of beliefs contains incomplete information, it may be quite difficult to see how to reflect new information in those beliefs. With the exception of Chapter 6, which gives a simple treatment of certain types of derived beliefs, this thesis is concerned solely with the problems of revising the base set of beliefs.

In the same vein, the interpretation of counterfactuals [Lewis 73, Ginsberg 85] faces very similar problems to those we address in the minimal-change semantics (Chapter 7): identification of potential states that satisfy a certain formula and differ as little as possible from a starting state.

This work provides a theoretical underpinning for the view update problem in database theory [Bancilhon 81, Dayal 82, Keller 82, 85]. As many researchers have noted, updates through views such as projections can produce incomplete information in the relations underlying the view. Given a view update policy, i.e., a method of translating updates expressed against views into updates on base relations, our approach can be used to implement those updates.

Chapter 3: Syntax, Semantics, and an Update Algorithm

Incomplete information occurs when, due to insufficient knowledge about the state of the world, there is more than one candidate database to represent the current state of the world. In the database world, one can imagine the user keeping a set of relational databases (even an infinite set, if one imagines vigorously), knowing that one of these databases corresponds to the actual state of the world, but needing more information to know which database is the correct one.[†] If the user wants to apply an ordinary relational update to this set of candidate databases, then the natural definition of the semantics of the update is to apply the update to each candidate database individually.

Though this imaginary scenario paints a clear picture of the semantics of ordinary updates when incomplete information is present, it is unsuitable for direct implementation due to the prohibitive expense of storing multiple databases. A more compact representation of the candidate databases is required for the sake of efficiency. Our solution is the *extended relational theory*, a formalization of the multiple–database scenario and an extension of Reiter's relational theories [Reiter 84, 84b]. Extended relational theories are sufficiently powerful to represent in one theory any realistic[††] set of relational databases all having the same schema and integrity constraints. Section 3.1 gives a formal description of the language and structure of extended relational theories.

Ordinary relational updates are not sufficiently powerful to express all desirable transformations on a set of candidate databases. For example, with ordinary updates there is no way to add new candidate databases to the set, or eliminate old candidates that are now known to be incorrect. Section 3.2 proposes a syntax and semantics suitable for updates to extended relational theories.

Though extended relational theories solve the compact representation problem, they raise another question: how can the effect of an update on a set of candidate databases be translated into an algorithm that operates directly on an extended relational theory? Section 3.3 presents the Update Algorithm for applying updates to extended relational theories, and Section 3.4 discusses the computational complexity of the Update Algorithm.

[†] Heuristic guidelines may be available that give likelihood estimates for the different possible states of the world [Michalski 86, Nilsson 86, Zadeh 79]. How to incorporate these into an update algorithm is an interesting open question.

[††] Not every set of relational databases can be represented as the models of a first–order theory. However, it is highly unlikely that any application of this work will ever run up against that particular limitation of logic.

3.1. Extended Relational Theories

We now give a formal presentation of *extended relational theories*, a method of representing multiple candidate databases in a single logic theory.

3.1.1. The Language

The language $\mathcal{L}$ for the theories contains the following symbols:

1. An infinite set of *variables*, for use in axioms.

2. An infinite set of *constants*. These represent the elements in the domains of database attributes, plus additional constants for technical reasons.

3. A finite set of *data predicates* of arity 1 or more, representing the attributes and relations of the database.

4. Punctuation symbols '(', ')', and ','.

5. Logical connectives, quantifiers, truth values, and the equality predicate: $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$, T, F, and $=$.

6. For each database predicate R (item 3 above), one *history predicate* H_R of arity one greater than R. Also, a unary history predicate H. The history predicates are present for technical reasons.

7. An infinite set of *Skolem constants* ϵ, ϵ_1, ϵ_2, ϵ_3, ... Skolem constants are the logical formulation of null values; they represent existentially quantified variables. For example, if a logical theory consists of the two wffs $R(\epsilon, c_1)$ and $R(c_2, \epsilon) \vee R(\epsilon_2, \epsilon_3)$, then this theory has the same models as the wff $\exists x_1 \exists x_2 \exists x_3 (R(x_1, c_1) \wedge (R(c_2, x_1) \vee R(x_2, x_3)))$ (see e.g. [Enderton 72]). $\Diamond$

Note that this language does not have any means of representing the null value commonly called "inapplicable." Inapplicable nulls do not fit into a logic framework, as they indicate a mismatch between the possible models of theories over a language and the real world that the models are intended to represent. Vassiliou [79] offers a lattice–theoretic treatment of imapplicable nulls; Zaniolo [82] offers another approach. Or, one can revamp the predicates of the language/database to prevent the occurrence of "inapplicable" nulls, for example along the lines of the structural model [Wiederhold 83]. Or, for the reader interested in working out the details of such a scheme, conventional wisdom has it that inapplicable nulls are computationally quite tractable to handle in traditional database queries and updates; it is said that one does not need to resort to logic or another sophisticated framework in order to describe the effect of a series of updates on a database containing inapplicable nulls.

We now present some terminology used in the remainder of this work.

Atomic formulas are well–formed formulas (wffs) without logical connectives. We consider Skolem constants to be functions; hence Skolem constants may occur in atomic formulas. For the purposes of this chapter, *atoms* are atomic formulas without variables as arguments. Atoms without Skolem constants are

called *null–free*. *Datoms* (pronounced "datums") are atoms *over data predicates*. For example, $R(\epsilon)$ and $R(c)$ are datoms; $R(x)$, $H_R(c)$, and $\epsilon{=}c$ are not datoms. The name "datom" is intended to invoke the image of a datum, i.e., a bit of data, and indeed, datoms will be the building blocks of our incomplete–information databases; and also to invoke thoughts of atoms, i.e., the atomic formulas of first–order logic, and indeed, datoms have a logical nature as well.

Definitions. σ is a *substitution* if σ defines a syntactic replacement of distinct Skolem constants and/or variables by constants and Skolem constants. In traditional form a substitution σ applied to a wff α is written, for example, as $(\alpha)^{\epsilon_1 \cdots \epsilon_n}_{c_1 \cdots c_n}$, or more concisely as $(\alpha)_\sigma$. The *wff form* of σ is the wff $\epsilon_1{=}c_1 \wedge \cdots \wedge \epsilon_n{=}c_n$. The wff form of the identity substitution (i.e., where no substitutions are specified) is the truth value T. If c_1 through c_n are all constants, then σ is a *constant substitution*. All substitutions are assumed to be nonredundant, i.e., if ϵ_i is replaced by c_i, then c_i is not later itself replaced in σ. $\Diamond$

In the discussions that follow, σ will be assumed to be in wff form whenever that follows logically from the context; for example, assume σ is in wff form when it is a subformula of $\alpha \wedge \sigma$.

On occasion we will speak of a more exotic type of syntactic replacement, that of one datom for another. For example, $(\alpha)^{R(c)}_{H_R(c,d)}$ calls for the replacement of all occurrences of $R(c)$ in α by the history atom $H_R(c,d)$. A datom substitution has no wff form. Datom substitutions will be so designated explicitly in the text.

3.1.2. Extended Relational Theories

Extended relational theories, an extension of Reiter's relational theories [Reiter 84], encode the semantics of databases with incomplete information. A theory $\mathcal{T}$ over $\mathcal{L}$ is an *extended relational theory* if $\mathcal{T}$ has exactly the following wffs:

1. *Body:* The body of $\mathcal{T}$ may be any finite set of wffs of $\mathcal{L}$ without variables. For example, the body might be the wff $\neg(R(c) \wedge R(\epsilon_1))$.

In ordinary relational databases, the convention is that all atoms not explicitly mentioned in the database are false; that is, the database contains only those atoms that are known to be true [Clark 78, Lifschitz 85, Reiter 80]. An analogue of this *closed–world assumption* is needed for extended relational theories, as otherwise $\mathcal{T}$ might have models in which, for example, an infinite number of datoms were true. An appropriate closed–world assumption is that $\mathcal{T}$ must include axioms stating that the only datoms that may be true in a model of $\mathcal{T}$ are those that unify† with subformulas of $\mathcal{T}$. This means that a datom not unifying with any datom of $\mathcal{T}$ should be false in all models of $\mathcal{T}$. The closed–world assumption is codified in the completion axiom section of $\mathcal{T}$.

† In this formulation, two atoms f and g unify if there exists a substitution σ for the Skolem constants and variables of f and g under which f and g are syntactically identical. If atoms f and g unify with one another under substitutions σ_1 and σ_2, then σ_1 is *more general* than σ_2 if there exists a substitution σ_3 such that $((f)_{\sigma_1})_{\sigma_3}$ is $(f)_{\sigma_2}$.

2. *Completion Axioms:* $\mathcal{T}$ contains one completion axiom for each n-ary data predicate R of $\mathcal{T}$. If no atom over R is a subformula of the body of $\mathcal{T}$, then $\mathcal{T}$ contains the completion axiom $\forall x_1 \cdots \forall x_n \neg R(x_1, \ldots, x_n)$. Otherwise, $\mathcal{T}$ contains the axiom

$$\forall x_1 \ldots \forall x_n (R(x_1, \ldots, x_n) \rightarrow \bigvee_{\sigma \in S} \sigma),$$

where S is the set of all most general substitutions σ such that some atom in the body of $\mathcal{T}$ unifies with $R(x_1, \ldots, x_n)$ under σ. $\Diamond$

Note that the completion axioms of $\mathcal{T}$ may be derived mechanically from the rest of $\mathcal{T}$. For example, $\forall x(R(x) \rightarrow (x{=}c \vee x{=}\epsilon_1))$ is a completion axiom for the example body given earlier. We say that $R(c_1, \ldots, c_n)$ is *represented* in the axiom if $(x_1 = c_1) \wedge \ldots \wedge (x_n = c_n)$ is a disjunct of the completion axiom.

A model $\mathcal{M}$ of $\mathcal{T}$ is a *standard* model if in addition to all the formulas in $\mathcal{T}$, $\mathcal{M}$ satisfies the unique name axioms $c_1 \neq c_2$ for each pair of distinct constants c_1, c_2 in $\mathcal{L}$. In this work, all models under discussion are assumed to be standard models; so, for example, a wff α is satisfiable iff it is satisfied by some standard model.

Each standard model includes a mapping from the constants and Skolem constants of $\mathcal{L}$ to elements in the universe of $\mathcal{M}$. The effect of this mapping on Skolem constants will often be of particular interest, and to allow easy reference to this information, we will now define a set of special wffs associated with $\mathcal{M}$, its *Skolem constant substitutions.* Let $\mathcal{L}'$ be an extension of $\mathcal{L}$, created as follows: for each Skolem constant ϵ of $\mathcal{L}$ that maps to an element c in $\mathcal{M}$ that is not named by any constant of $\mathcal{L}$, add an additional constant to $\mathcal{L}'$ and map it to c in $\mathcal{M}$. Then the Skolem constant substitution σ of $\mathcal{M}$ with respect to a finite set of wffs S is a substitution of constants of $\mathcal{L}'$ for all the Skolem constants of S, such that the wff σ is true in $\mathcal{M}$. Note that if $\mathcal{M}$ is a model of $\mathcal{T}$, then $\mathcal{M}$ is also a model of $(\mathcal{T})_\sigma$.

In an implementation of extended relational theories, one would not actually store the unique name or completion axioms. Rather, the axioms formalize our intuitions about the behavior of a query and update processor operating on the body of the extended relational theory. For example, PROLOG is a query processor that shares our unique name axioms, but has an entirely different closed-world assumption.

Another possible type of completion axiom, the *domain completion axiom* [Reiter 84], has not been included in the definition of extended relational theories. The domain completion axiom takes the form $\forall x((x = c_1) \vee \cdots \vee (x = c_n))$, implying that there are a finite number of elements in the universe, and they are all known and named by constants or Skolem constants in $\mathcal{L}$. This completion axiom can be maintained during updates by using the same techniques as for the other completion axioms. The universe completion axiom will be discussed in more detail in Chapter 6.

Recall that some predicates are present in $\mathcal{L}$ merely for technical reasons: the history predicates. Therefore the models of $\mathcal{T}$ give truth valuations for all history predicate atoms, even though history predicates are not of interest to users. For that reason we define the *alternative worlds* of $\mathcal{T}$, written Worlds($\mathcal{T}$), as the objects produced by reducing Models($\mathcal{T}$) (i.e., the models of $\mathcal{T}$) to data predicates. A model $\mathcal{M}$ *represents* an alternative world $\mathcal{A}$ if removing the truth valuation information in $\mathcal{M}$ for history predicates produces $\mathcal{A}$. World($\mathcal{M}$) is the alternative world represented by $\mathcal{M}$; Worlds(S), for S a set of models, is $\bigcup_{\mathcal{M} \in S}$ World($\mathcal{M}$).

Intuitively, an alternative world is a snapshot of the tuples of a complete–information relational database. The alternative worlds of an extended relational theory look like a set of ordinary relational databases all having the same schema and axioms.

With the inclusion of history predicates in $\mathcal{L}$, we depart from Reiter's paradigm. Because these predicates are "invisible" in alternative worlds, there may not be a one–to–one correspondence between the models of a relational theory and its alternative worlds, as two models may give the same truth valuations to all null–free datoms but differ on some null–free history atoms, and still represent the same alternative world. Alternative worlds contain just the information that would be of interest to a database user, while models may be cluttered with history atoms of no external interest. The history predicates do not actually extend the expressive power of $\mathcal{L}$; the proof of Theorem 7-1 will show that, with a few minor restrictions, given any extended relational theory $\mathcal{T}$ there exists an extended relational theory $\mathcal{T}'$ not containing history predicates, such that Worlds($\mathcal{T}$) = Worlds($\mathcal{T}'$).

3.2. A Language for Updates

As mentioned in the introduction to this chapter, traditional relational update languages are not sufficiently powerful for use when incomplete information is present. The traditional languages also lack sufficiently formal semantics for a rigorous examination of the properties of these languages. This section presents a data manipulation language that remedies these two deficiencies. Appropriate subsets of traditional update languages, such as those of SQL and INGRES without aggregation, may be embedded in this language. In this chapter, only updates without variables will be considered; Chapter 4 extends this approach to updates with variables.

3.2.1. Update Syntax

Let ϕ and ω be formulas over $\mathcal{L}$ without history predicates or variables. Then an *update* takes the form `INSERT` ω `WHERE` ϕ.

The reader may wonder what has happened to the traditional relational data manipulation operations of `MODIFY` and `DELETE`. Under the semantics presented below, any `DELETE` or `MODIFY` request can be phrased as an `INSERT` request, using negation. To simplify the presentation, `DELETE` and `MODIFY` are

omitted right from the start; details of the mapping will be presented at the end of Section 3.2.2.

Examples. Suppose the database schema has two relations, `Mgr(Manager, Department)` and `Emp(Employee, Department)`. Then the following are updates, with their approximate intended semantics offered in italics:

`INSERT Emp(Reid,`ϵ`)` $\wedge$ `(`ϵ`=CSD` $\vee$ ϵ`=EE) WHERE` $\neg$`Mgr(Nilsson,CSL).` *In alternative worlds where Nilsson doesn't manage CSL, insert the fact that Reid is in one of CSD and EE.*

`INSERT` $\neg$`Emp(Reid,`ϵ`) WHERE` $\neg$`Mgr(Nilsson,`ϵ`)` $\wedge$ `Emp(Reid,`ϵ`).` *For some department Nilsson does not manage, delete the fact that Reid is in that department.*

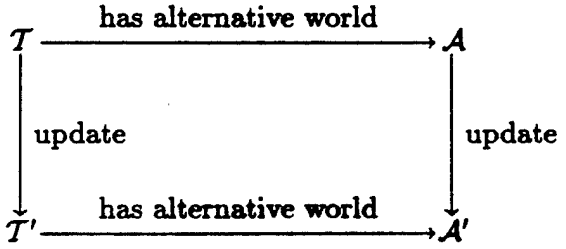
`INSERT F WHERE` $\neg$`Emp(Reid,CSL).` *Eliminate all alternative worlds where Reid isn't in CSL.*

`INSERT` $\neg$`Emp(Reid,CSL)` $\wedge$ `Emp(Reid,`ϵ`) WHERE Emp(Reid,CSL).` *In any alternative worlds where Reid was in CSL, reduce that belief to just believing that he is in some department.*

`INSERT` $\neg$`Emp(Reid,`ϵ`) WHERE T.` *Insert the fact that Reid is not a member of every department.*

3.2.2. Update Semantics

We define the semantics of an update operating on an extended relational theory $\mathcal{T}$ by its desired effect on the models of $\mathcal{T}$. In particular, the alternative worlds of the updated relational theory must be the same as those obtained by applying the update separately to each original alternative world. In database terms, this may be rephrased as follows: The database with incomplete information represents a (possibly infinite) set of alternative worlds, or complete–information relational databases, each different and each one possibly the real, unknown world. The correct result of an update is that obtained by storing a separate database for each alternative world and running the update in parallel on each separate database. A necessary and sufficient guarantee of correctness for any more efficient and practical method of update processing is that it produce the same results for updates as the parallel computation method. Equivalently, we require that the diagram below be commutative: both paths from upper–left–hand corner to lower–right–hand corner must produce the same result.



The general criteria guiding our choice of semantics are, first, that the semantics agree with traditional semantics in the case where the update request is to insert or delete a single atom, or to modify one atom to be another. Second, an update cannot directly change the truth valuations of any atoms except those that unify with atoms of ω. For example, the update `INSERT` Emp(Reid, CSD) `WHERE` T cannot change the department of any employee but Reid, and cannot change the truth valuation of formulas such as Mgr(Nilsson, CSD).

Two more important criteria are that the new information in ω is to represent the *most exact and most recent state of knowledge obtainable* about the atoms that the update inserts; and the update is to *override all previous information* about these atoms. These two criteria have a syntactic component: one should not necessarily expect two updates with logically equivalent ωs to produce the same results. For example, the update `INSERT` T `WHERE` T is different from `INSERT` Emp(Reid, CSD)$\vee\neg$Emp(Reid, CSD) `WHERE` T; one update reports no change in the information available about Reid's department, and the other reports that whether Reid is in CSD is now unknown.

For a formal definition of semantics that meets the criteria outlined in this section, let U be a null–free update and let $\mathcal{M}$ be a model of an extended relational theory $\mathcal{T}$. Then $U(\mathcal{M})$ contains just $\mathcal{M}$ if ϕ is false in $\mathcal{M}$. Otherwise, $U(\mathcal{M})$ contains every model $\mathcal{M}'$ with the same universe and mappings as $\mathcal{M}$, such that

(1) $\mathcal{M}'$ agrees with $\mathcal{M}$ on the truth valuations of all null–free atoms except possibly those in ω; and

(2) ω is true in $\mathcal{M}'$. $\Diamond$

Example. If the user requests `INSERT` Emp(Reid, CSD) $\vee$ Emp(Reid, EE) `WHERE` T, then three models are created from each model $\mathcal{M}$ of $\mathcal{T}$: one where Reid is in both CSD and EE, one where Reid is just in CSD, and one where Reid is just in EE—regardless of whether Reid was in CSD or EE in $\mathcal{M}$ originally. $\Diamond$

For simplicity, the semantics of U has been defined in terms of U's effect on the model $\mathcal{M}$ rather than in terms of U's effect on the alternative world represented by $\mathcal{M}$. However, because the semantics is independent of the truth valuations of history atoms in $\mathcal{M}$, U will have the same effect (i.e., produce the same alternative worlds) on *every* model representing the same alternative world as $\mathcal{M}$.

The remarks at the beginning of this section on correctness of update processing may be summed up in the following definition:

Definition. Given two extended relational theories $\mathcal{T}$ and $\mathcal{T}'$, $\mathcal{T}'$ accomplishes the null–free update U if

$$\text{Worlds}(\mathcal{T}') = \bigcup_{\mathcal{M}\in\text{Models}(\mathcal{T})} \text{Worlds}(U(\mathcal{M})). \qquad \Diamond$$

This semantics must be extended to cover the case where Skolem constants occur in U. Intuitively, the essential idea is that if the user only had more information, the user would not be requesting an update containing Skolem constants, but rather an ordinary update without Skolem constants. Under this assumption, the correct way to handle an update U with Skolem constants is to consider all the possible null–free updates represented by U and execute each of those in parallel, collecting the alternative worlds so produced in one large set. Then the result of the update the user would have requested had more information been available is guaranteed to be in that set.

For a more formal definition, a bit of new terminology is needed. If U is the update `INSERT` ω `WHERE` ϕ and σ is a substitution, then let $(U)_\sigma$ be the update `INSERT` $(\omega)_\sigma$ `WHERE` $(\phi)_\sigma$. If $\mathcal{T}$ is a theory or set of wffs, then let $(\mathcal{T})_\sigma$ be the theory resulting from applying σ to each formula in $\mathcal{T}$.

Definition. Given two extended relational theories $\mathcal{T}$ and $\mathcal{T}'$, $\mathcal{T}'$ accomplishes the update U if

$$\text{Worlds}(\mathcal{T}') = \bigcup_{\mathcal{M} \in \text{Models}(\mathcal{T})} \text{Worlds}((U)_\sigma(\mathcal{M})),$$

where σ is the Skolem constant substitution for $\mathcal{M}$ with respect to U. $\Diamond$

A moment's examination of the semantics given earlier shows that this definition simply amounts to replacing ω in rules 1 and 2 by $(\omega)_\sigma$.

The advantage of this approach to null values in updates is that σ associates any Skolem constant ϵ that occurs in both U and $\mathcal{T}$ with the same element in $\mathcal{M}$, so that the user can directly refer to entities such as "that department that we earlier noted that Reid is in, though we didn't know exactly which department it was."

Example. If $\forall x \neg R(x)$ is true in $\mathcal{M}$ and we then insert $R(\epsilon_1) \vee R(\epsilon_2)$ into $\mathcal{M}$, then $U(\mathcal{M})$ will contain every model $\mathcal{M}'$ such that just one or two null–free atoms of R are true in $\mathcal{M}'$, with truth valuations for other datoms unchanged. $\Diamond$

Under these definitions, the traditional relational operations of `DELETE` and `MODIFY` can be phrased as `INSERT` requests as follows: to delete a datom t in all alternative worlds where ϕ is true, use the update `INSERT` $\neg t$ `WHERE` ϕ. For example, `INSERT` ¬Emp(Reid, CSL) `WHERE` T will "delete" the atom Emp(Reid, CSL) from the Emp relation. To modify a datom t to be a different datom ω, use the update `INSERT` $\omega \wedge \neg t$ `WHERE` $\phi \wedge t$. For example, to change Reid's department from CSL to CSD, use the update `INSERT` Emp(Reid, CSD)∧¬Emp(Reid, CSL) `WHERE` Emp(Reid, CSL).

3.3. The Update Algorithm

The semantics presented in the previous section describes the effect of an update on the *models* of a theory; the semantics gives no hints whatsoever on how to translate that effect into changes in the extended relational theory. An algorithm for performing updates cannot proceed by generating models from the theory and updating them directly; this is because the number of non-isomorphic models may be exponential in the length of the theory, and it may be very difficult to find even one model, as that is equivalent to satisfiability testing of the theory. Any update algorithm must find a more efficient way of implementing this semantics.

The *Update Algorithm* proposed in this section for incorporating updates into an extended relational theory $\mathcal{T}$ may be summarized as follows: *For each atom f in $\mathcal{T}$ that unifies with an atom of ω, replace all occurrences of f in $\mathcal{T}$ by a history atom.* † *Then add a new formula to $\mathcal{T}$ that defines the correct truth valuation of f when ϕ is false, and another formula to give the correct valuation of f when ϕ is true.*

Before a more formal presentation of the Update Algorithm, let us motivate its workings in a series of examples that will illustrate the problems and principles underlying the algorithm.

3.3.1. A Simple Example

Let the body of $\mathcal{T}$ be ¬Emp(Reid, CSL), and the new update be `INSERT` Emp(Reid, CSL) `WHERE` T.

One's natural instinct is to add $\phi \rightarrow \omega$ to $\mathcal{T}$, because the update says that ω is to be true in all alternative worlds where ϕ is true now. Unfortunately, ω probably contradicts the rest of $\mathcal{T}$. For example, adding T→Emp(Reid, CSL) to $\mathcal{T}$ makes $\mathcal{T}$ inconsistent, because $\mathcal{T}$ already contains Emp(Reid, CSL). Evidently ω may contradict parts of $\mathcal{T}$, and those parts must be removed from $\mathcal{T}$; in this case it would suffice to simply remove the formula ¬Emp(Reid, CSL).

But suppose that the body of $\mathcal{T}$ contains more complicated formulas: Mgr(Nilsson, CSD)↔¬Emp(Reid,CSD) and Mgr(Nilsson, CSL)↔¬Emp(Reid, CSD). One cannot simply excise ¬Emp(Reid, CSL) or replace it by a truth value without changing the models for the remaining atoms of $\mathcal{T}$; but by the semantics for updates, no datom truth valuation except that of Emp(Reid, CSL) can be affected by the requested update.

The solution to this problem is to replace all occurrences of Emp(Reid, CSL) in $\mathcal{T}$ by another atom. However, the atom used must not be part of any alternative world, as otherwise the replacement might change that atom's truth valuation. This is where the special history predicates of $\mathcal{L}$ come into play; we can replace each atom of ω by a history atom throughout $\mathcal{T}$, and make only minimal changes in the truth valuations in the alternative worlds of $\mathcal{T}$. In the

† These history atoms are not visible externally, i.e., they may not occur in updates; they are for internal extended relational theory use only.

current case, Emp(Reid, CSL) is replaced by H_{Emp}(Reid, CSL, U), where U is a unique ID for the current update. † For convenience, we will write H_{Emp}(Reid, CSL, U) as H(Emp(Reid, CSL), U), to avoid the subscript. This atom may look forbidding, but it is really quite simple; read it as "Reid was in CSL at the time of update U". The datom substitution that replaces every datom f of ω by its history atom $H(f, U)$ is called the *history substitution* and is written σ_H. Again, $H(f,U)$ should be read as "f was true at the time of update U".

This is not the only possible means of removing the datoms of ω from $\mathcal{T}$. Contradictory wffs may be ferreted out and removed without using history predicates, by a process such as that used by Weber [86]: If f is a datom of ω that is a subformula of a wff α, then replace α by $(\alpha)^f_T \vee (\alpha)^f_F$. Unfortunately, in the worst case such a process will multiply the space required to store the theory by a factor that is exponential in the number of atoms in the update. In addition, if a datom f of ω also is a subformula of ϕ, then once f is removed entirely from $\mathcal{T}$, it may not be possible to identify the models of $\mathcal{T}$ where ϕ was true, i.e., where the update is to take place. Therefore this technique is useful only for atoms of ω that do not unify with any atom of ϕ. Further, this technique does not extend well to the case where Skolem constants occur in U or $\mathcal{T}$ (see the proof of Theorem 7-1). Finally, when this method is used it is expensive to specify the correct truth valuations for the datoms of ω in models where ϕ is false. As its worst–case characteristics are less pleasant than when a history substitution is used, this technique will not be considered further.

Now consider a slightly more complicated update U: `INSERT` Emp(Reid, CSL) `WHERE` Mgr(Nilsson, CSL), when $\mathcal{T}$ contains just ¬Emp(Reid, CSL). As just explained, the first step is to replace this body by $(\neg$Emp(Reid, CSL)$)_{\sigma_H}$, i.e., $\neg H$(Emp(Reid, CSL), U). Within a model $\mathcal{M}$ of $\mathcal{T}$, this step interchanges the truth valuations of every datom f in ω and its history atom $H(f,U)$; if ϕ was true in $\mathcal{M}$ initially, then $(\phi)_{\sigma_H}$ is now true in $\mathcal{M}$.

It is now possible to act on the original algorithmic intuition and add $(\phi)_{\sigma_H} \rightarrow \omega$ to the body of $\mathcal{T}$, establishing correct truth valuations for the atoms of ω in models where ϕ was true initially. In the employee example, the body of $\mathcal{T}$ now contains the two formulas

$\neg H$(Emp(Reid, CSL), U) and

Mgr(Nilsson, CSL)→Emp(Reid, CSL).

Unfortunately, the fact "if Nilsson is *not* the manager of CSL then Reid is *not* in CSL" has been lost! The solution is to also add formulas governing truth valuations for atoms in ω when ϕ is false: Add $(f \leftrightarrow H(f,U)) \vee (\phi)_{\sigma_H}$ to $\mathcal{T}$ for each atom f in ω. In other words, if ϕ was false in a model $\mathcal{M}$ when the update began, then f has the same truth valuation in $\mathcal{M}$ as it did originally. Then $\mathcal{T}$ contains

† If the argument U were not present, then a similar substitution in a later update involving Emp(Reid, CSL) would make big changes in the alternative worlds of $\mathcal{T}$ at that time. For that reason, the U in $H(f, U)$ should be a constant, so that $H(f, U)$ will not unify with any history atom used in any other update.

$\neg H$(Emp(Reid, CSL), U),

Mgr(Nilsson, CSL)$\rightarrow$Emp(Reid, CSL), and

(Emp(Reid, CSL)$\leftrightarrow H$(Emp(Reid,CSL), U)) $\vee$ Mgr(Nilsson, CSL).

To make the new version of $\mathcal{T}$ into an extended relational theory, the new atom Mgr(Nilsson, CSL) must be represented in the completion axiom for Mgr. Once this is done, yet another problem remains, for this newest theory has models in which Nilsson manages CSL, even though the completion axioms of the original theory disallowed that. The solution is to add $\neg$Mgr(Nilsson, CSL) to the body of $\mathcal{T}$. This is best accomplished at the very beginning of the update process, before the history substitution is applied. If we retroactively add this wff, $\mathcal{T}$ now has the desired alternative worlds.

Reviewing the example, we see that the update process falls into two phases. The first phase is best thought of as a preprocessing stage, where $\mathcal{T}$ is changed by representing new atoms in its completion axioms. This phase does not change the alternative worlds of $\mathcal{T}$. In the second phase, the alternative worlds of $\mathcal{T}$ are altered, first by the use of history predicates, and then by the addition of formulas governing the truth valuation of atoms in ω.

3.3.2. An Example with Skolem Constants

The informal algorithm proposed so far does not work when Skolem constants are present in either the theory or the update. The basic difficulty is that one must update every atom in the theory that unifies with something in ω, since truth valuations for that atom might possibly be changed by the new update. For example, suppose the body of $\mathcal{T}$ contains the formula Mgr(Nilsson, ϵ), and the new update is `INSERT` $\neg$Mgr(Nilsson, CSL) `WHERE` T. In other words, Nilsson was known to manage some department, and is now known not to manage CSL, quite possibly because he has just resigned that position.† A moment's thought shows that quite possibly Nilsson now manages no departments (e.g., if he has retired), and so the formula Mgr(Nilsson, ϵ), which unifies with Mgr(Nilsson, CSL), must be changed in some way; ($\epsilon \neq$CSL)$\rightarrow$Mgr(Nilsson, ϵ) is the obvious replacement. In the general case, it is necessary to replace all atoms in $\mathcal{T}$ that unify with datoms of ω by history atoms as part of the history substitution step.

Let's examine one final example. Suppose the theory initially contains the wff Mgr(Nilsson, CSL) and the new update takes the form `INSERT` Mgr(Nilsson, ϵ) `WHERE` T, implying that Nilsson may now manage another department. In the first phase of the update, Mgr(Nilsson, ϵ) is to be represented to the completion axiom for Mgr, *without* changing the models of $\mathcal{T}$. In earlier examples, it sufficed

† In other words, the update leaves open the possibility that the underlying state of the world has changed. To say that Nilsson does not manage CSL, while retaining the belief that Nilsson manages some department, the appropriate update is `INSERT` F `WHERE` Mgr(Nilsson, CSL); this new update says that the state of the world has not changed, but that we now have more information about its state. Although both updates talk about Nilsson's department, their semantics are quite different.

to add a disjunct to Mgr and add ¬Mgr(Nilsson, ϵ) to the body of $\mathcal{T}$. Unfortunately, this procedure would change the alternative worlds of $\mathcal{T}$ by permanently eliminating the possibility that ϵ is CSL:

Mgr(Nilsson, CSL),

¬Mgr(Nilsson, ϵ).

This problem arises because Mgr(Nilsson, ϵ) already is an implicit subformula[†] The solution is to add the wff Mgr(Nilsson, ϵ)→(ϵ=CSL) to the body of $\mathcal{T}$ rather than ¬Mgr(Nilsson, ϵ), that is, to add the fact that if Nilsson already manages a department ϵ, then ϵ must be a department already mentioned in $\mathcal{T}$ as a possible candidate for his management.

Continuing with phase two of the suggested algorithm, a theory is produced containing the four formulas

H(Mgr(Nilsson, ϵ), U) → (ϵ=CSL),

H(Mgr(Nilsson, CSL), U),

T→Mgr(Nilsson, ϵ), and

(Mgr(Nilsson, ϵ)↔H(Mgr(Nilsson, ϵ), U))∨ T.

Unfortunately, this theory has models where Mgr(Nilsson, CSL) is false! The problem is that the algorithm does not yet properly take care of the alternative worlds where ϵ is not bound to CSL; in those worlds, Mgr(Nilsson, CSL) must still be true, regardless of what the new information in the update may be. The solution is to add ($\epsilon\neq$CSL)→(Mgr(Nilsson, CSL)↔H(Mgr(Nilsson, CSL), U)) to $\mathcal{T}$, and in fact this new theory has the desired alternative worlds.

3.3.3. The Algorithm

The lessons of the preceding examples may be summarized as an algorithm for executing an update U given by `INSERT` ω `WHERE` ϕ against an extended relational theory $\mathcal{T}$.

The Update Algorithm (Version I)

Input. An extended relational theory $\mathcal{T}$ and an update U.

Output. $\mathcal{T}'$, an updated version of $\mathcal{T}$.

Procedure. A sequence of four steps:

Step 1. Maintain the closed–world assumption. To maintain the closed–world assumption, all datoms in ω and ϕ need to be represented in the completion axioms of $\mathcal{T}$. First change the body of $\mathcal{T}$ to reflect the new completion axioms: for each atom g that is a subformula of ω or ϕ but not of $\mathcal{T}$, let Σ_0 be the set of the most general substitutions σ such that for some datom f in $\mathcal{T}$, f unifies with

[†] If a datom f is not a subformula of a wff α, but there is a substitution σ such that f is a subformula of $(\alpha)_\sigma$, then f is an *implicit* subformula of α. For example, $R(c)$ and $R(d)$ are implicit subformulas of $R(\epsilon)\wedge(\epsilon=c)$. of T.

g under σ. If Σ_0 is the empty set, then add $\neg g$ to the body of $\mathcal{T}$; otherwise, add the wff

$$g \rightarrow \bigvee_{\sigma \in \Sigma_0} \sigma \tag{1}$$

to the body of $\mathcal{T}$. Then for every datom g of $\mathcal{T}$ not represented in the completion axioms, add a disjunct representing g to those axioms. Call the resulting theory $\mathcal{T}'$.

Example. If ω contains the datom $R(a, \epsilon_2, \epsilon_1)$, and the body of $\mathcal{T}$ contains the datoms $R(\epsilon_3, c, \epsilon_4)$ and $R(\epsilon_4, c, c)$, then add $R(a, \epsilon_2, \epsilon_1) \rightarrow ((a = \epsilon_3) \wedge (\epsilon_2 = c) \wedge (\epsilon_1 = \epsilon_4)) \vee ((a = \epsilon_4) \wedge (\epsilon_2 = c) \wedge (\epsilon_1 = c))$ to the body of $\mathcal{T}$, and add the disjunct $(x_1 = a) \wedge (x_2 = \epsilon_2) \wedge (x_3 = \epsilon_1)$ to the completion axiom for R. Intuitively, Formula (1) says that if g is true in some model of $\mathcal{T}$, this must be because g has unified with a preexisting atom of $\mathcal{T}$ in that model.

Step 2. Make history. For each atom f in $\mathcal{T}'$ that unifies with an atom of ω, replace all occurrences of f in the body of $\mathcal{T}'$ by the history atom $H(f, U)$. In other words, replace the body $\mathcal{B}$ of $\mathcal{T}'$ by $(\mathcal{B})_{\sigma_H}$.

Step 3. Define the scope of the update. Add the wff $(\phi)_{\sigma_H} \rightarrow \omega$ to $\mathcal{T}'$.

Step 4. Restrict the scope of the update. For each datom f in σ_H, let Σ be the set of all most general substitutions σ under which f unifies with an atom of ω. Add the wff

$$(f \leftrightarrow H(f, U)) \vee ((\phi)_{\sigma_H} \wedge \bigvee_{\sigma \in \Sigma} \sigma) \tag{2}$$

to $\mathcal{T}'$. Intuitively, for f an atom that might possibly have its truth valuation changed by update U, formula (2) says that the truth valuation of f can change only in a model where ϕ was true originally, and further that in any model so created, f must be unified with an atom of ω. $\Diamond$

Example. Let the body of $\mathcal{T}$ be the wff

$\neg$Emp(Reid, CSD)$\wedge$Emp(Reid, CSL)$\wedge$ Mgr(Nilsson, ϵ),

and the update be **INSERT** Emp(Reid, ϵ) $\wedge$ ($\epsilon \neq$EE) **WHERE** $\top$. Then the alternative worlds of $\mathcal{T}$ initially consist of all worlds where Reid is in CSL and Nilsson manages some one department. After the update, the alternative worlds should be those where Reid is in CSL and Reid is in a department managed by Nilsson, and that department is not EE.

Step 1. Add the wff Emp(Reid, ϵ)$\rightarrow$((ϵ=CSD) $\vee$(ϵ=CSL)) to the body of $\mathcal{T}$, and the corresponding disjunct to the completion axiom. Note that Step 1 does not change the alternative worlds of the theory.

Step 2. Replace Emp(Reid, CSD), Emp(Reid, CSL), and Emp(Reid, ϵ) by H(Emp(Reid, CSD), U), H(Emp(Reid, CSL), U), and H(Emp(Reid, ϵ), U) respectively. The body of $\mathcal{T}'$ now contains the two wffs

$\neg H$(Emp(Reid, CSD), U) $\wedge$ H(Emp(Reid, CSL), U) $\wedge$ Mgr(Nilsson, ϵ) and

H(Emp(Reid, ϵ), U) $\rightarrow$((ϵ=CSD)$\vee$(ϵ=CSL)).

Step 3. Add the wff $(\phi)_{\sigma_H} \rightarrow \omega$ (i.e., $\top\rightarrow$(Emp(Reid, ϵ)$\wedge$($\epsilon\neq$EE))) to the body of $\mathcal{T}'$.

Step 4. Add to $\mathcal{T}'$ the three wffs

(Emp(Reid, ϵ)$\leftrightarrow H$(Emp(Reid, ϵ), U)) $\vee$ $\top$,

(Emp(Reid, CSD) $\leftrightarrow H$(Emp(Reid, CSD), U)) $\vee$ (ϵ=CSD), and

(Emp(Reid, CSL) $\leftrightarrow H$(Emp(Reid, CSL), U)) $\vee$ (ϵ=CSL).

Examination of Worlds($\mathcal{T}'$) shows that $\mathcal{T}'$ accomplishes U. $\Diamond$

The models of $\mathcal{T}'$ produced by the Update Algorithm always represent exactly the alternative worlds that U is defined to produce from $\mathcal{T}$:

Theorem 3-1. Given an extended relational theory $\mathcal{T}$ and an update U, the extended relational theory $\mathcal{T}'$ produced by the Update Algorithm Version I accomplishes U. $\Diamond$

In other words, Worlds($\mathcal{T}'$) $=\bigcup_{\mathcal{M}\in\text{Models}(\mathcal{T})}$ Worlds($U(\mathcal{M})$). Theorem 3-1 is not proven here, as it follows immediately from Theorem 4-1.

3.4. Computational Complexity of the Update Algorithm

Let the *size* of a wff be defined as the number of occurrences of atoms in the wff, and let the size of an update U be the sum of the sizes of ϕ and ω. Let U be an update of size k; and let R be the maximum number of distinct datoms of $\mathcal{T}$ over the same predicate. When $\mathcal{T}$ and U contain no Skolem constants, the Update Algorithm will process U in time $\mathcal{O}(k \log R)$ (the same asymptotic cost as for ordinary database updates) and increase the size of $\mathcal{T}$ by $\mathcal{O}(k)$ worst case. This is not to say that an $\mathcal{O}(k \log R)$ implementation of updates is the best choice; rather, it is advisable to devote extra time to heuristics for minimizing the length of the formulas to be added to $\mathcal{T}$. Nonetheless, a worst-case time estimate for the algorithm is informative, as it tells us how much time must be devoted to the algorithm proper. The implementation assumptions necessary for this estimate to be achieved are described in the chapter on implementation, Chapter 9. Further, we assume that the schema is fixed, i.e., that the number of predicates is a constant.

When Skolem constants occur in $\mathcal{T}$ or in U, the controlling factor in costs is the number of atoms of $\mathcal{T}$ that unify with atoms of U. If n atoms of $\mathcal{T}$ each unify with one atom of U, then $\mathcal{T}$ will grow by $\mathcal{O}(n + k)$. In the worst case, *every* atom of $\mathcal{T}$ may unify with *every* atom of U, in which case after a series of m updates, the number of occurrences of atoms in $\mathcal{T}$ may *multiply* by $\mathcal{O}(mk)$. Theorem 3-2 summarizes these properties.

Theorem 3-2. Let $\mathcal{T}$ be an extended relational theory containing n different datoms (not occurrences of datoms) having Skolem constants as arguments.

Let k be a constant that is an upper bound on the size of updates. Then after a series of m updates not containing Skolem constants is performed by the Update Algorithm, in the worst case the size of $\mathcal{T}$ will increase by $\mathcal{O}(nmk)$. Under a series of m updates containing Skolem constants, in the worst case the size of $\mathcal{T}$ will increase by $\mathcal{O}(nmk + m^2k^2)$. $\Diamond$

Proof of Theorem 3-2. We show the space requirements for each step of the Update Algorithm.

Let g be a datom of U, the first of the m updates. If g already is a subformula of $\mathcal{T}$, then nothing is added to $\mathcal{T}$ for g in Step 1. Otherwise, g is not a subformula of $\mathcal{T}$, and the number of datoms in $\mathcal{T}$ that unify with g determines the size of Σ_0 in Step 1. By assumption, g unifies with at most n datoms of $\mathcal{T}$. By assumption, the predicates in $\mathcal{L}$ are fixed, and hence each substitution σ in Σ_0 is of size bounded by a constant. Therefore at most $\mathcal{O}(nk)$ occurrences of atoms are added to $\mathcal{T}$ for U. Under a series of m updates not containing Skolem constants, Step 1 can add as many as $\mathcal{O}(nmk)$ occurrences of atoms to $\mathcal{T}'$. If the updates contain Skolem constants, then each update can add k datoms containing Skolem constants to $\mathcal{T}$, so that the first update after U may have a Σ_0 of size $n+k$, the second may have size $n+2k$, and so on. As there may be k choices of Σ_0 for each update, after m updates the size of this compounding factor is $\mathcal{O}(m^2k^2)$.

Step 2 does not change the size of $\mathcal{T}'$. Step 3 adds $\mathcal{O}(k)$ occurrences of atoms to $\mathcal{T}'$.

For Step 4 of update U, a trick is helpful to keep down the size of formula (2). It can be quite expensive to repeatedly add $(\phi)_{\sigma_H}$ to $\mathcal{T}'$ for every choice of f in formula (2). Much more efficient is to add a single wff $H(U) \leftrightarrow (\phi)_{\sigma_H}$ to $\mathcal{T}'$ before Step 4, and then use $H(U)$ in place of $(\phi)_{\sigma_H}$ in all instantiations of formula (2). ($H(U)$ is simply a history atom not unifying with any atom in $\mathcal{T}'$.) We assume that this measure is taken, incurring a cost of $O(k)$ atoms per update.

If U does not contain Skolem constants, there are at most $n+1$ datoms in $\mathcal{T}'$ that unify with a datom of ω, giving a maximum of $n+1$ choices for f in formula (2). (If U contains Skolem constants, there may be as many as $n+k$ such datoms in $\mathcal{T}'$.) Let f be a datom in $\mathcal{T}'$ that unifies with a datom of ω. The size of formula (2) for g is $\mathcal{O}(k)$ worst case, so the cost of instantiating formula (2) for U will be $\mathcal{O}(nk)$ (or $\mathcal{O}((n+k)k$, if U contains Skolem constants). Therefore under a series of m updates not containing Skolem constants, Step 4 will add up to $\mathcal{O}(nmk)$ occurrences of atoms to $\mathcal{T}'$. If the updates contain Skolem constants, then each update can add k datoms containing Skolem constants to $\mathcal{T}$, so that again a compounding factor of $\mathcal{O}(m^2k^2)$ appears. $\Diamond$

As for the time complexity of the Update Algorithm, let us assume that an indexing scheme is available that enables any datom to be located in $\mathcal{T}$ in $\mathcal{O}(\log R)$ time. Then the running time of the Update Algorithm is $\mathcal{O}(kn \log R)$ worst case. This estimate assumes that the history step (Step 2) is optimized through special

data structures (see Chapter 9): the body of the extended relational theory must be represented as a set of logical relationships between *pointers*. All occurrences of a single datom in the body are linked together in a chain of pointers; only the head of the chain points to the stored record for the actual datom.

Happily, a large class of common types of updates—those with very simple ω and ϕ—can be performed in $\mathcal{O}(k \log R)$ time per update; Abiteboul and Grahne [Abiteboul 85] examine a subset of these simple updates. For the general case, however, potential growth of $\mathcal{O}(nmk)$ in the size of $\mathcal{T}$ is much too large, yet is unavoidable if the effect of the update is to be represented directly in the extended relational theory, for every datom of $\mathcal{T}$ that is an implicit subformula of the update *must* be changed in some way in $\mathcal{T}$. In some sense the information content of a single update is no more than its size, k, and so growth of more than $\mathcal{O}(mk)$ after m updates is too much. We can achieve growth of no more than $\mathcal{O}(mk)$ by simply storing the updates without incorporating them into $\mathcal{T}$. However, since query answering presupposes some means of integrating updates with the rest of the database to allow satisfiability testing, a means of at least temporary incorporation must be offered. We have devised a scheme of delayed evaluation and simplification of expensive updates, by bounding the permissible number of unifications for the atoms of an incoming update. This lazy evaluation technique is discussed in Chapter 5.

3.5. Summary and Conclusion

In this chapter we formalized databases containing incomplete information as logical theories, and viewed the models of these *extended relational theories* as representing possible states of the world that are consistent with all known information. For the purposes of this chapter, formulas in the body of an extended relational theory could be any sentences without universal quantification. Typically incomplete information appears in these theories as disjunctions or as Skolem constants (a.k.a. null values).

Within this context, we set forth a data manipulation language for updates, and gave model–theoretic definitions of the meaning of these updates. We presented the Update Algorithm as a means of incorporating updates into extended relational theories, and proved it correct in the sense that the alternative worlds produced under the Update Algorithm are the same as those produced by updating each alternative world individually.

For extended relational theories and updates without Skolem constants, the Update Algorithm has the same asymptotic cost as for an ordinary complete–information database update, but may increase the size of the extended relational theory. For updates involving Skolem constants, the increase in size will be severe if many atomic formulas in the theory unify with those in the update. Chapter 5 is devoted to a discussion of a means of preventing excessive growth in the theory.

We conclude that, first, one may extend the concept of a database update to databases with incomplete information in a natural way; second, that mathematical logic is a fruitful paradigm and tool for the investigation; third, that one

may construct an algorithm to perform these updates with a reasonable polynomial running time; and lastly, that some means is needed to prevent runaway growth in the database under a series of updates.

Chapter 4: Updates With Variables

We now consider how to extend the Update Algorithm to accept updates with variables—the type of update supported by traditional data manipulation languages. The two main results shown in this chapter are, first, that updates containing variables are no harder to perform than updates without variables, provided that variables and quantifiers are permitted in the bodies of extended relational theories; and second, that if quantifiers and variables are not permitted in theory bodies, updates are somewhat harder to perform but a reasonable algorithm is still possible, and its cost will depend on the number of substitutions for variables that lead to a satisfiable selection clause ϕ. In addition, the algorithm given in this chapter for updating extended relational theory bodies containing quantifiers is sufficiently general to use in updating any first–order theory.

Please note that variables will be permitted to occur in updates for the duration of this chapter only; subsequent chapters consider only ground updates, except when specifically noted otherwise.

4.1. Update Syntax

As usual, we confine our attention to `INSERT` requests: `INSERT` ω `WHERE` ϕ. The only change required in update syntax is that variables may now occur in ϕ and ω.

4.2. Update Semantics

We begin by presenting a desideratum for the extension of update semantics to updates with variables: the chosen semantics should agree with traditional semantics for relational data manipulation language updates with variables.

As an approach that meets this desideratum, let an extended relational theory update U containing variables correspond to a set of updates without variables, derived by binding constants and Skolem constants to all the variables of U. If we apply every possible binding[†] to the variables of U, then the result of applying U to an extended relational theory $\mathcal{T}$ should be that of simultaneously applying all the updates in the (probably infinite) set just generated.

To rephrase this definition more formally, let U: `INSERT` ω `WHERE` ϕ be an update containing variables. Let $\mathcal{M}$ be a model of an extended relational theory $\mathcal{T}$, and let σ be the Skolem constant substitution for $\mathcal{M}$ with respect to ϕ and

[†] Strictly speaking, this imagery is inadequate because not all elements of the universe are named in $\mathcal{L}$. Rather, one should consider an extension of $\mathcal{L}$ for each model in which all the elements in the model are named.

ω. Let Σ_v be the desired set of substitutions σ_v for all the variables of ϕ and ω. Let Ω be the set of all wffs $(\omega)_{\sigma_v}$ such that σ_v is in Σ_v, $(\phi)_{\sigma_v}$ is true in $\mathcal{M}$, and $((\omega)_{\sigma_v})_\sigma$ is satisfiable. Then $U(\mathcal{M})$ contains every model $\mathcal{M}'$ with the same universe and mappings as $\mathcal{M}$, such that

(1) $\mathcal{M}'$ agrees with $\mathcal{M}$ on the truth valuations of all null–free atoms except possibly those in Ω;

(2) All members of Ω are true in $\mathcal{M}'$. $\Diamond$

An extended relational theory $\mathcal{T}'$ accomplishes $U(\mathcal{T})$ if $\text{Worlds}(\mathcal{T}') = \bigcup_{\mathcal{M} \in \text{Models}(\mathcal{T})} \text{Worlds}(U(\mathcal{M}))$.

Examples. Consider the following three updates, to be applied to an extended relational theory $\mathcal{T}$ with body Emp(Reid, ϵ_1):

1. INSERT Emp(Reid, x) WHERE $\neg$Emp(Reid, x)
2. INSERT Emp(Reid, ϵ_1) WHERE $\neg$Emp(Reid, ϵ_1)
3. INSERT Emp(Reid, ϵ_2) WHERE $\neg$Emp(Reid, ϵ_2)

The first update applied to a model $\mathcal{M}$ of $\mathcal{T}$ makes Reid an employee of all departments in $\mathcal{M}'$ (and therefore, depending on the domains involved and the method used to calculate Σ_v, may be unsafe). The second update does not change the models of $\mathcal{T}$ at all; and the third update produces all models where Reid is in one or two different departments. $\Diamond$

If U does not contain variables and ω is satisfiable, U will produce one or more models from every model $\mathcal{M}$ to which U is applied. Once variables occur in U, this ceases to be true. For example, the update INSERT $R(x) \wedge \neg R(y)$ WHERE $x \neq y$ will probably be ill–advised when applied to a theory containing $R(a) \wedge R(b)$, because it asks for $R(a)$ and $R(b)$ to be both true and false: $R(a) \wedge \neg R(b)$, and $R(b) \wedge \neg R(a)$. We will not provide any syntactic means of avoiding conflicting updates; in our system, conflicting updates simply eliminate models where a conflict arises.

4.3. An Update Algorithm: No Variables in Body

This section presents an update algorithm for use with extended relational theories without quantifiers and variables in the theory body—the type of extended relational theory studied so far. Section 4.4 presents an algorithm for use when quantifiers and variables may occur in the theory body.

The semantics for updates with variables presented in Section 4.2 does not directly lend itself to algorithmic application when quantifiers and variables are not allowed in the body of $\mathcal{T}$. We must ensure that all but a finite number of the substitutions σ_v used lead to updates that do not change the models of $\mathcal{T}$ at all. If this is true, then all the substitutions that generate no-op updates can be ignored: only a finite set Σ_v of substitutions will be relevant. The method

traditionally used in database data manipulation languages to guarantee a finite Σ_v is the use of safe selection clauses [Ullman 82]. An adaptation of the concept presented there to the incomplete information situation might be to include a substitution σ_v in Σ_v iff σ_v substitutes constants and Skolem constants *already occurring in* $\mathcal{T}$ *or* U for all the variables of U. A domain completion axiom can be employed to this end. Another technique would be to require typed selection clauses, that is, to have type axioms (see Chapter 6) and require that the selection clause specify the type of all variables; INGRES [Stonebraker 85] and System R [Chamberlain 76] use a variant of this technique. We choose not to dictate the choice of a safe query mechanism, but rather operate on the assumption that one way or another, the query and update processor knows how to reduce an update with variables to a finite set of ground updates. In practice in today's database management systems, determination of Σ_v is typically initiated via index lookup on selection and join attributes. As is true in ordinary databases when variables occur in updates, an update with variables will often require more changes in the extended relational theory than a ground update does, because each instantiation of variables represents an additional change to be made in the theory.

The essential idea of the update algorithm is to create one ground update for each substitution σ_v in Σ_v. We do *not* require that $(\phi)_{\sigma_v}$ actually be true in some model of $\mathcal{T}$, as such a condition is equivalent to testing satisfiability, and hence might require exponential time to verify. The generation of Σ_v should be done by the algorithms used for query processing, and should require time polynomial in the size of the extended relational theory and exponential in the length of the update request.

We now present an extension of the Update Algorithm Version I to handle updates with variables. The new Update Algorithm must take into account that an atom of $\mathcal{T}$ may be affected simultaneously in several different ways by different instantiations of the variables in an update.

The Update Algorithm (Version II)

Input. An extended relational theory $\mathcal{T}$, an update U and a set Σ_v of substitutions σ_v for all the variables of U.

Output. $\mathcal{T}'$, an updated version of $\mathcal{T}$.

Procedure. A sequence of four steps:

Step 1. Maintain the closed–world assumption. To maintain the closed–world assumption, all datoms in $(\omega)_{\sigma_v}$ and $(\phi)_{\sigma_v}$, for all $\sigma_v \in \Sigma_v$, must be represented in the completion axioms of $\mathcal{T}$. First change the body of $\mathcal{T}$ to reflect the new completion axioms: for each datom g that is a subformula of some $(\omega)_{\sigma_v}$ or $(\phi)_{\sigma_v}$ but not of $\mathcal{T}$, let Σ_0 be the set of substitutions σ such that for some datom f of the body of $\mathcal{T}$, f unifies with g under the most general substitution σ. If Σ_0 is the empty set, then add $\neg g$ to the body of $\mathcal{T}$; otherwise, add the wff

$$g \rightarrow \bigvee_{\sigma \in \Sigma_0} \sigma \qquad (1)$$

to the body of $\mathcal{T}$. Then for every datom g of $\mathcal{T}$ not represented in the completion axioms, add a disjunct representing g to those axioms. Call the resulting theory $\mathcal{T}'$.

Step 2. Make history. For each atom f of the body of $\mathcal{T}'$ that unifies with an atom of $(\omega)_{\sigma_v}$ for some $\sigma_v \in \Sigma_v$, replace all occurrences of f in the body of $\mathcal{T}'$ by the history atom $H(f, U)$. In other words, replace the body $\mathcal{B}$ of $\mathcal{T}'$ by $(\mathcal{B})_{\sigma_H}$.

Step 3. Define the scope of the update. For every σ_v in Σ_v, add the wff $((\phi)_{\sigma_v})_{\sigma_H} \rightarrow (\omega)_{\sigma_v}$ to $\mathcal{T}'$.

Step 4. Restrict the scope of the update. For each σ_v and each datom f in σ_H, let Σ_{vf} be the set of substitutions σ such that f unifies with an atom of some $(\omega)_{\sigma_v}$ under the most general substitution σ. For each datom f in σ_H, add the wff

$$(f \leftrightarrow H(f,U)) \vee \bigvee_{\substack{\sigma_v \in \Sigma_v \\ \Sigma_{vf} \neq \emptyset}} (((\phi)_{\sigma_v})_{\sigma_H} \wedge (\bigvee_{\sigma \in \Sigma_{vf}} \sigma)) \qquad (2)$$

to $\mathcal{T}'$. Intuitively, for f an atom that might possibly have its truth valuation changed by update U, formula (2) says that the truth valuation of f can change only in a model where $(\phi)_{\sigma_v}$ (for some σ_v) was true originally, and further that in any model so created, f must be unified with an atom of $(\omega)_{\sigma_v}$ for that same σ_v. $\Diamond$

Example. Let U be `INSERT` $\neg$Emp(Reid, x) `WHERE` Emp(Reid, x), when $\mathcal{T}$ contains Emp(Reid, CSD)$\wedge$ Emp(Reid, ϵ_1). The alternative worlds of $\mathcal{T}$ initially consist of all worlds where Reid is in CSD and possibly one other department, and all else is false. After the update, $\mathcal{T}'$ should have one alternative world, in which everything is false. The set of substitutions Σ_v contains the two substitutions x=CSD and $x=\epsilon_1$.

Step 1. No actions are required, as both atoms that unify with $(\omega)_{\sigma_v}$ are already in $\mathcal{T}$.

Step 2. Upon application of σ_H, the body of $\mathcal{T}'$ becomes
H(Emp(Reid, CSD), U) $\wedge$ H(Emp(Reid, ϵ_1), U).

Step 3. Two wffs are added to the body of $\mathcal{T}'$:
H(Emp(Reid, CSD), U) $\rightarrow \neg$Emp(Reid, CSD) and
H(Emp(Reid, ϵ_1), U) $\rightarrow \neg$Emp(Reid, ϵ_1).
Because U is a simple update, at this point $\mathcal{T}'$ already has the correct alternative worlds, and Step 4 is superfluous.

Step 4. Add to $\mathcal{T}'$ the following two formulas:

$$\begin{aligned}(\text{Emp(Reid, CSD)} \leftrightarrow & H(\text{Emp(Reid, CSD)}, U)) \\ & \vee (H(\text{Emp(Reid, CSD)}, U) \wedge \top) \\ & \vee (H(\text{Emp(Reid}, \epsilon_1), U) \wedge (\epsilon_1 = c))\end{aligned}$$

$$\begin{aligned}(\text{Emp(Reid, } \epsilon_1) \leftrightarrow & H(\text{Emp(Reid, } \epsilon_1), U)) \\ & \vee (H(\text{Emp(Reid, CSD)}, U) \wedge (\epsilon_1 = c)) \\ & \vee (H(\text{Emp(Reid, } \epsilon_1), U) \wedge \top). \quad \Diamond\end{aligned}$$

Please note that Version I of the Update Algorithm is a special case of Version II.

The models of $\mathcal{T}'$ represent exactly the alternative worlds that U is defined to produce from $\mathcal{T}$:

Theorem 4-1. For any extended relational theory $\mathcal{T}$ and update U possibly containing variables, the Update Algorithm Version II accomplishes U. $\Diamond$

In other words, $\mathcal{T}'$ is an extended relational theory, and $\text{Worlds}(\mathcal{T}') = \bigcup_{\mathcal{M} \in \text{Models}(\mathcal{T})} \text{Worlds}(U(\mathcal{M}))$. Readers not interested in a formal proof of correctness for the Update Algorithm should skip to the next section. To prove Theorem 4-1, we will use a lemma showing that Step 1 of the Update Algorithm does not change the models of $\mathcal{T}$.

Lemma 4-1. Let $\mathcal{T}$ be a theory containing a completion axiom α for an n-ary predicate R, and let f be a ground datom $R(c_1, \ldots, c_n)$ not represented in α. Let Σ_0 be the set of all substitutions σ such that for some datom g in $\mathcal{T}$, f unifies with g under the most general substitution σ. Let $\mathcal{T}'$ be the theory created from $\mathcal{T}$ by adding the new disjunct $(x_1{=}c_1 \wedge\ x_2{=}c_2 \wedge \cdots \wedge\ x_n{=}c_n)$ to α, and then adding $\neg f$ to the body of $\mathcal{T}$ if Σ_0 is the empty set or adding

$$f \rightarrow \bigvee_{\sigma \in \Sigma_0} \sigma$$

otherwise. Then $\mathcal{T}$ and $\mathcal{T}'$ have the same models. $\Diamond$

Proof of Lemma 4-1. Let α' be α with the disjunct added to represent f, and let β be the wff $f \rightarrow (\bigvee_{\sigma \in \Sigma_0} \sigma)$. First consider the case where Σ_0 is nonempty.

Let $\mathcal{M}$ be a model of $\mathcal{T}$. Let σ be the Skolem constant substitution for $\mathcal{M}$ with respect to $\mathcal{T}$. $\mathcal{M}$ satisfies all wffs of $\mathcal{T}'$ other than α' and β, since all other wffs also are formulas of $\mathcal{T}$. But $\alpha \rightarrow \alpha'$, so $\mathcal{M}$ satisfies α'. As for β, if f is false in $\mathcal{M}$ then β is satisfied. If f is true in $\mathcal{M}$, then $(f)_\sigma$ must be represented by some disjunct of $(\alpha)_\sigma$. Let g be the datom represented by that same disjunct in α. Then g and f unify under substitution σ, and therefore β is satisfied in $\mathcal{M}$. We conclude that $\mathcal{M}$ is a model of $\mathcal{T}'$.

For the reverse implication, let $\mathcal{M}'$ be a model of $\mathcal{T}'$ and let σ be the Skolem constant substitution for $\mathcal{M}'$ with respect to $\mathcal{T}'$. $\mathcal{M}'$ satisfies all the wffs of $\mathcal{T}$ except possibly α. But if α is false in $\mathcal{M}'$, it must be because for some binding to the variables of α, the disjunct representing f is true in $\mathcal{M}'$, i.e., that f is true in $\mathcal{M}'$. But then by β there exists a datom g of $\mathcal{T}$ such that f unifies with g under σ. Since g is represented in α, $\mathcal{M}'$ satisfies α. Therefore $\mathcal{M}'$ is a model of $\mathcal{T}$.

Now consider the case where Σ_0 is the empty set, i.e., where f is false in all models of $\mathcal{T}$. Let $\mathcal{M}$ be a model of $\mathcal{T}$. Then $\alpha \rightarrow \alpha'$, and $\neg f$ is true in $\mathcal{M}$, so $\mathcal{M}$ is also a model of $\mathcal{T}'$.

Conversely, if $\mathcal{M}'$ is a model of $\mathcal{T}'$ and σ is the Skolem constant substitution for $\mathcal{M}'$ with respect to $\mathcal{T}'$, then $\mathcal{M}'$ satisfies all wffs of $\mathcal{T}$ except possibly α. But if α is false in $\mathcal{M}'$ for some instantiation of the variables of α, it must be because the disjunct representing f in α' is true in $\mathcal{M}'$. But we know that f is false in $\mathcal{M}'$. Therefore $\mathcal{M}'$ is a model of $\mathcal{T}$. $\Diamond$

Proof of Theorem 4-1. For simplicity of reference, let $\mathcal{T}$ be the original extended relational theory, $\mathcal{T}_1$ be the theory produced by step 1 of the Update Algorithm, $\mathcal{T}_2$ be the theory produced by step 2, and so on. $\mathcal{M}$ will always refer to a model of the original theory, $\mathcal{M}_1$ to a model of $\mathcal{T}_1$, and so on. We first show that the Update Algorithm produces a subset of the correct set of alternative worlds.

Suppose that $\mathcal{M}_4$ is a model of $\mathcal{T}_4$. Let σ_4 be the Skolem constant substitution for $\mathcal{M}_4$ with respect to $\mathcal{T}_4$. Our goal is to show that U should produce $\mathcal{M}_4$ from some model $\mathcal{M}$ of $\mathcal{T}$. It suffices to show that $\mathcal{T}_1$ has such a model $\mathcal{M}$, because by Lemma 4-1, the models of $\mathcal{T}$ and $\mathcal{T}_1$ are the same.

Let $\mathcal{F}$ be the set containing all datoms f in σ_H. Let $\mathcal{M}$ be a model that has same universe and constant and Skolem constant mappings as $\mathcal{M}_4$, and that agrees with $\mathcal{M}_4$ on the truth valuations for all null–free datoms except possibly those in $(\mathcal{F})_{\sigma_4}$, that is, except those obtained by applying σ_4 to datoms in $\mathcal{F}$. If f is in $\mathcal{F}$, then let the truth valuation of f in $\mathcal{M}$ be the same as that of $H(f,U)$ in $\mathcal{M}_4$. To show that $\mathcal{M}$ is actually a model of $\mathcal{T}_1$, let α be a wff of the body of $\mathcal{T}_1$. The descendant of α in $\mathcal{T}_4$ is $(\alpha)_{\sigma_H}$. Since $\mathcal{M}$ and $\mathcal{M}_4$ agree on the truth assignments to all atoms of $(\alpha)_{\sigma_H}$, therefore $(\alpha)_{\sigma_H}$ must be true in $\mathcal{M}$. This implies that α will be true in $\mathcal{M}$ if every atom f of $\mathcal{F}$ that is a subformula of α has the same truth assignment in $\mathcal{M}$ as does $H(f,U)$ in $\mathcal{M}$ and $\mathcal{M}_4$. But this is true by definition. As the completion axioms are the same in both theories, we conclude that $\mathcal{M}$ is a model of $\mathcal{T}_1$ and $\mathcal{T}$.

It remains to show that U applied to $\mathcal{M}$ produces the alternative world of $\mathcal{M}_4$. Let Σ_ϕ be the set of all σ_v in Σ_v such that $(\phi)_{\sigma_v}$ is true in $\mathcal{M}$. By the previous argument, $((\phi)_{\sigma_v})_{\sigma_H}$ is satisfied by $\mathcal{M}_4$ iff $\sigma_v \in \Sigma_\phi$. By the formula of Step 3, it follows that $(\omega)_{\sigma_v}$ is true in $\mathcal{M}_4$ for all $\sigma_v \in \Sigma_\phi$, so rule 2 of the definition of INSERT is satisfied by $\mathcal{M}_4$. For rule 1, if the truth valuation of a null–free datom f is different in $\mathcal{M}$ and $\mathcal{M}_4$, then $f \in (\mathcal{F})_{\sigma_4}$ and therefore f unifies with an atom of $(\omega)_{\sigma_v}$ for some set of $\sigma_v \in \Sigma_v$. If $(\phi)_{\sigma_v}$ is false in $\mathcal{M}$ for all such σ_v, then by formula (2), $f \leftrightarrow H(f,U)$ must be true in $\mathcal{M}_4$, and rule 1 is satisfied. We conclude that U produces the alternative world of $\mathcal{M}_4$ from $\mathcal{M}$.

We have shown that the Update Algorithm produces only correct alternative worlds; we now turn to the question of completeness: does the Update Algorithm produce every alternative world that should be derived under U?

Let $\mathcal{M}$ be a model of $\mathcal{T}$, and let σ_1 be the Skolem constant substitution for $\mathcal{M}$ with respect to $\mathcal{T}$ and U. By Lemma 4-1, $\mathcal{M}$ is also a model of $\mathcal{T}_1$.

Let Σ_ϕ be the set of all substitutions $\sigma_v \in \Sigma_v$ such that $(\phi)_{\sigma_v}$ is true in $\mathcal{M}$. Select one particular set v of truth valuations for the atoms of $((\omega)_{\sigma_v})_{\sigma_1}$, for all $\sigma_v \in \Sigma_\phi$, such that $(\omega)_{\sigma_v}$ is true under v for all $\sigma_v \in \Sigma_\phi$ and v is satisfiable. If no such v exists, then U produces no alternative worlds from $\mathcal{M}$, and the theorem follows.

Let $\mathcal{M}_4$ be the model that agrees with v on all datom valuations of v; where $H(f,U)$ is assigned the same valuation as f had in $\mathcal{M}$, for all datoms f in σ_H; that has the same universe and constant and Skolem constant mappings as $\mathcal{M}$; and that agrees with $\mathcal{M}$ on all other null–free atom truth valuations. Then $\mathcal{M}_4$ is a model of an arbitrary alternative world that should be produced by U from $\mathcal{M}$, and we claim that $\mathcal{M}_4$ is a model of $\mathcal{T}_4$.

Let σ_v be a substitution in Σ_v. First, $\mathcal{M}_4$ satisfies the completion axioms of $\mathcal{T}_4$, as every datom of $(\omega)_{\sigma_v}$ already is a subformula of $\mathcal{T}_1$, and $\mathcal{T}_4$ and $\mathcal{T}_1$ have identical completion axioms. For atoms f in $(\omega)_{\sigma_v}$, since $H(f,U)$ has the same truth valuation in $\mathcal{M}_4$ as does f in $\mathcal{M}$, it follows that $\mathcal{M}_4$ satisfies $(\mathcal{B})_{\sigma_H}$, that is, all the formulas of the body of $\mathcal{T}_1$, to which σ_H was applied in Step 2. Since $(\omega)_{\sigma_v}$ is true in $\mathcal{M}_4$ if $\sigma_v \in \Sigma_\phi$, the wff $((\phi)_{\sigma_v})_{\sigma_H} \rightarrow (\omega)_{\sigma_v}$ added to $\mathcal{T}_4$ in Step 3 is satisfied in $\mathcal{T}_4$. There is only one remaining class of wffs of $\mathcal{T}_4$ that $\mathcal{M}_4$ might not satisfy: formula (2) from Step 4.

Let f be an atom in $\mathcal{F}$. If f and $H(f,U)$ have the same truth valuations in $\mathcal{M}_4$, then formula (2) is satisfied. If f and $H(f,U)$ have different truth valuations in $\mathcal{M}_4$, then $(f)_{\sigma_1}$ must appear in v, and therefore also in $((\omega)_{\sigma_v})_{\sigma_1}$ for some $\sigma_v \in \Sigma_\phi$. Therefore $(\phi)_{\sigma_v}$ must be true in $\mathcal{M}$, and $((\phi)_{\sigma_v})_{\sigma_H}$ must be true in $\mathcal{M}_4$. This implies that formula (2) is satisfied, since $\sigma_1 \wedge ((\phi)_{\sigma_v})_{\sigma_H}$ is true in $\mathcal{M}_4$. We conclude that $\mathcal{M}_4$ is a model of $\mathcal{T}_4$, and the alternative world of $\mathcal{M}_4$ is produced by the Update Algorithm.

It remains to verify that $\mathcal{T}_4$ is an extended relational theory. $\mathcal{T}_4$ has disjuncts in its completion axioms for exactly the datoms in its body. The body of $\mathcal{T}_4$ is still finite and contains no variables. This concludes the proof of correctness for the Update Algorithm. $\Diamond$

The computational complexity of Version II of the Update Algorithm depends on the size of Σ_v. In particular, if V is the number of members of Σ_v, then the number of atoms that are added to $\mathcal{T}'$ will be as much as V times greater than that added by the same steps in Version I. Of course the same relationship holds between ordinary relational database insertions with and without variables. The time complexity of Version II will likewise by multiplied by a factor of V worst case: $\mathcal{O}(V \log R(nmk + m^2k^2))$.

4.4. An Update Algorithm: Variables in Body

This section presents an update algorithm for use with extended relational theories with arbitrary formulas in the theory body. This technique is of particular

interest, as it gives a method of updating theories for non-database applications. Having variables in the theory body makes more work for the query processor, but as we will see, makes life much easier for the update processor.

First, the definitions given earlier for extended relational theories, substitution, unification, etc., need to be modified slightly. *Please note that these definitions are in effect for the remainder of this chapter only; subsequent chapters will revert to the original definitions.* The changes needed are as follows:

- Substitutions: A variable may be substituted for another variable. For example, the atomic formulas Emp(Reid, x) and Emp(y, z) now unify under substitution Reid=$y \wedge x=z$.
- Variables are permitted in atoms and datoms. For example, Emp(Reid, x) and Emp(y, z) are now both datoms.
- An extended relational theory may now be any finite theory; that is, the extended relational theory contains only a body, which may be any finite set of wffs. All free variables in the theory are implicitly universally quantified over the scope of the formula in which they occur. As before, only standard models will be considered.

What happened to the completion axioms? Since quantifiers are now permitted in theory bodies, there is no reason to separate out the completion axioms from the rest of the theory. To implement a closed–world assumption for a predicate R, it suffices to include the wff $\forall x_1 \cdots \forall x_n \neg R(x_1, \ldots, x_n)$ in the body at the inception of the theory. Subsequent updates will maintain the closed–world assumption automatically, by modifying that formula. The examples given after the presentation of the new update algorithm will illustrate this technique.

Though the definitions of extended relational theories and other technical terms are changed slightly for this section, update syntax and semantics remain exactly as presented in Sections 4.1 and 4.2.

With these formalities out of the way, we turn to the main result of this section: a very simple version of the Update Algorithm accomplishes updates containing variables. This algorithm, Version III, adds only $\mathcal{O}(k)$ atoms to the size of the theory, where k is the size of the update. This is in contrast to Version II, which depends directly on the number of instantiations of variables in Σ_v (given to Version II as part of its input), and on the number of datoms in the theory that unify with datoms in the update. Further, this independence from Σ_v means that Version III works correctly even for updates with unsafe selection clauses—e.g., an infinite number of relevant instantiations of variables—and also for universe elements that are not named in $\mathcal{L}$. Version II, on the other hand, is restricted to safe selection clauses and bindings of variables to constants and Skolem constants in $\mathcal{L}$. We now present the Update Algorithm Version III.

The Update Algorithm (Version III)

Input. A theory $\mathcal{T}$ and an update U possibly containing variables.
Output. $\mathcal{T}'$, an updated version of $\mathcal{T}$.
Procedure. A sequence of three steps:
Step 1. Make history. Let σ_H be the substitution that replaces each atom f of $\mathcal{T}$ and U that unifies with an atom of ω by its history atom $H(f,U)$. Then replace all occurrences of f in $\mathcal{T}$ by $H(f,U)$. In other words, replace the body $\mathcal{B}$ of $\mathcal{T}$ by $(\mathcal{B})_{\sigma_H}$. Call the resulting theory $\mathcal{T}'$.
Step 2. Define the scope of the update. Add the wff $(\phi)_{\sigma_H} \rightarrow \omega$ to $\mathcal{T}'$.
Step 3. Restrict the scope of the update. Let y_1 through y_n be the variables appearing in U. For each n-ary predicate R that appears in ω, let x_1 through x_n be variables not appearing in $\mathcal{T}'$, and let Σ_v be the set containing all most general substitutions σ such that $R(x_1,\ldots,x_n)$ unifies with a datom of ω under σ. Add the wff

$$\Big(R(x_1,\ldots,x_n) \leftrightarrow H(R(x_1,\ldots,x_n),U)\Big) \vee \exists y_1 \cdots \exists y_n \Big((\phi)_{\sigma_H} \wedge \bigvee_{\sigma_v \in \Sigma_v} \sigma\Big)$$

to $\mathcal{T}'$. ◊

Example. Let $\mathcal{T}$ contain the single wff $\forall x \forall y \neg$ Emp(x, y), and let U be the update **INSERT** Emp(Reid, CSD) **WHERE** Emp(Reid, EE). As there are no employees initially in $\mathcal{T}$, this update should not change any datom truth valuations. Step 1 changes $\mathcal{T}$ to the wff $\forall x \forall y \neg$ H(Emp(x, y), U); Step 2 adds the wff H(Emp(Reid, EE), U) $\rightarrow$ Emp(Reid, CSD); and Step 3 adds the wff
(Emp(x_1, x_2) $\leftrightarrow H$(Emp(x_1, x_2), U))$\vee\exists y_1 \exists y_2$ (H(Emp(Reid, EE), U) $\wedge$ (y_1 = Reid) $\wedge$ (y_2 = CSD)).
Clearly there are still no employees after the update. ◊

Example. Let $\mathcal{T}$ contain the wff $\forall x \forall y$ Emp(x, y) $\rightarrow$(x=Reid) $\wedge$ y=ϵ). The models of this theory have either no employees or just one employee, Reid in some one department. Let U be the update **INSERT** Emp(Reid, CSD) **WHERE** Emp(Reid, EE). This update should change all models where Reid is in EE so that Reid is now also in CSD. Step 1 changes $\mathcal{T}$ to the wff
$\forall x \forall y \neg$ H(Emp(x, y), U) $\rightarrow$(x=Reid $\wedge$ y=ϵ).
Step 2 adds the wff H(Emp(Reid, EE), U) $\rightarrow$ Emp(Reid, CSD); and Step 3 adds the wff
(Emp(x_1, x_2) $\leftrightarrow H$(Emp(x_1, x_2), U))$\vee\exists y_1 \exists y_2$ (H(Emp(Reid, EE), U) $\wedge$ (y_1 = Reid) $\wedge$ (y_2 = CSD)).
Again the correct models obtain. ◊

Example. Let $\mathcal{T}$ contain the wffs Emp(Reid, CSD), Emp(Lantz, EE), and

$$\text{Emp}(x,y) \rightarrow \Big(((x = \text{Reid}) \wedge (y = \text{CSD})) \vee ((x = \text{Lantz}) \wedge (y = \text{EE}))\Big).$$

Let U be the update INSERT Emp(y, CSD) WHERE Emp(y, EE). After this update is completed, $\mathcal{T}$ should have one alternative world, in which Reid is in CSD and Lantz is in CSD and EE. After the history substitution step, $\mathcal{T}'$ contains the three formulas

$H(\text{Emp(Reid, CSD)}, U)$,

$H(\text{Emp(Lantz, EE)}, U)$, and

$H(\text{Emp}(x, y), U) \rightarrow$

$$(((x = \text{Reid}) \wedge (y = \text{CSD})) \vee$$
$$((x = \text{Lantz}) \wedge (y = \text{EE})).$$

Step 2 adds the formula

$H(\text{Emp}(y, \text{EE}), U) \rightarrow \text{Emp}(y, \text{CSD})$

to $\mathcal{T}$; and Step 3 contributes the formula

$(\text{Emp}(x_1, x_2) \leftrightarrow (H(\text{Emp}(x_1, x_2), U))$

$$\vee \exists\, y(H(\text{Emp}(y, \text{EE}), U) \wedge (x_1 = y) \wedge (x_2 = \text{CSD}))). \quad \Diamond$$

Theorem 4-2. Let $\mathcal{T}$ be a theory, let U be an update possibly containing variables, and let $\mathcal{T}'$ be the theory produced from $\mathcal{T}$ and U by the Update Algorithm Version III. Then $\mathcal{T}'$ accomplishes U. $\Diamond$

Proof of Theorem 4-2. We begin by showing that the Update Algorithm Version III produces a subset of the correct set of alternative worlds.

Suppose that $\mathcal{M}_3$ is a model of $\mathcal{T}_3$. Let σ_3 be the Skolem constant substitution for $\mathcal{M}_3$ with respect to $\mathcal{T}_3$. Our goal is to show that U should produce $\mathcal{M}_3$ from some model $\mathcal{M}$ of $\mathcal{T}$.

Let $\mathcal{F}$ be the set containing all datoms f in σ_H. Let $\mathcal{M}$ be a model with the same universe and constant and Skolem constant mappings as $\mathcal{M}_3$, and that agrees with $\mathcal{M}_3$ on the truth valuations for all null–free datoms except possibly those obtained by binding universe elements to the variables in $(\mathcal{F})_{\sigma_3}$.† If f is in $\mathcal{F}$, and b is a binding for all the variables of f, then let the truth valuation of $(f)_b$ in $\mathcal{M}$ be the same as that of $(H(f,U))_b$ in $\mathcal{M}_3$.

To show that $\mathcal{M}$ is actually a model of $\mathcal{T}$, let α be a wff of $\mathcal{T}$. The descendant of α in $\mathcal{T}_3$ is $(\alpha)_{\sigma_H}$. For any binding b of elements of the universe to all the variables of $(\alpha)_{\sigma_H}$, $\mathcal{M}$ and $\mathcal{M}_3$ agree on the truth assignments to all atoms of $((\alpha)_{\sigma_H})_b$, and therefore $(\alpha)_{\sigma_H}$ must be true in $\mathcal{M}$. This implies that $(\alpha)_b$ will be true in $\mathcal{M}$ if for every atom f of $\mathcal{F}$ that is a subformula of α, the datom $(f)_b$ has the same truth assignment in $\mathcal{M}$ as does $(H(f,U))_b$ in $\mathcal{M}$ and $\mathcal{M}_3$. But this is true by definition. We conclude that $\mathcal{M}$ is a model of $(\mathcal{T})_{\sigma_3}$ and $\mathcal{T}$.

† To be strictly correct, rather than talking about binding universe elements to variables, we should extend $\mathcal{L}$ to a language in which all universe elements are named by constants, and then bind those constants to variables.

It remains to show that U applied to $\mathcal{M}$ produces the alternative world of $\mathcal{M}_3$. Let Σ_ϕ be the set containing all bindings b for all the variables of U such that $(\phi)_b$ is true in $\mathcal{M}$. By the previous argument, $((\phi)_b)_{\sigma_H}$ is satisfied by $\mathcal{M}_3$ iff $b \in \Sigma_\phi$. By the formula of Step 3, it follows that $(\omega)_b$ is true in $\mathcal{M}_3$ for all $b \in \Sigma_\phi$, so rule 2 of the definition of INSERT is satisfied by $\mathcal{M}_3$. For rule 1, if the truth valuation of a ground datom f is different in $\mathcal{M}$ and $\mathcal{M}_3$, then f is not a subformula of any member of Ω. Therefore for every binding b to all the variables of U such that f appears in $((\omega)_b)_{\sigma_3}$, it follows that $b \notin \Sigma_\phi$, and $(\phi)_b$ is false in $\mathcal{M}$. ϕ is false in $\mathcal{M}$. But then by the formula of Step 3, $f \leftrightarrow H(f,U)$ must be true in $\mathcal{M}_3$, and rule 1 is satisfied. We conclude that U produces the alternative world of $\mathcal{M}_3$ from $\mathcal{M}$.

We have shown that the Update Algorithm produces only correct alternative worlds; we now turn to the question of completeness: does the Update Algorithm produce every alternative world that should be derived under U?

Let $\mathcal{M}$ be a model of $\mathcal{T}$, and let σ be the Skolem constant substitution for $\mathcal{M}$ with respect to $\mathcal{T}$ and U. Let Σ_ϕ be the set of all bindings b for all the variables of U such that $(\phi)_b$ is true in $\mathcal{M}$. Select one particular set v of truth valuations for the atoms of $((\omega)_b)_\sigma$, for all $b \in \Sigma_\phi$, such that $((\omega)_b)_\sigma$ is true under v for all $b \in \Sigma_\phi$ and v is satisfiable. If no such v exists, then U produces no alternative worlds from $\mathcal{M}$, and the theorem follows.

Let $\mathcal{M}_3$ be the model that has the same universe and constant and Skolem constant mappings as $\mathcal{M}$; that agrees with v on all datom valuations of v; where $(H(f,U))_b$ is assigned the same valuation as $(f)_b$ had in $\mathcal{M}$, for all datoms f in σ_H and bindings b to the variables of f; and that agrees with $\mathcal{M}$ on all other null-free atom valuations. Then $\mathcal{M}_3$ is a model of an arbitrary alternative world that should be produced by U from $\mathcal{M}$, and we claim that $\mathcal{M}_3$ is a model of $\mathcal{T}_3$.

For atoms f in ω, since $H(f,U)$ has the same truth valuation in $\mathcal{M}_3$ as does f in $\mathcal{M}$, it follows that $\mathcal{M}_3$ satisfies $(\mathcal{B})_{\sigma_H}$, that is, all the formulas of the body of $\mathcal{T}$, to which σ_H was applied in Step 1. Let b be a binding for all the variables of U. Since $(\omega)_b$ is true in $\mathcal{M}_3$ if $b \in \Sigma_\phi$, the wff $(\phi)_{\sigma_H} \rightarrow \omega$ added to $\mathcal{T}_3$ in Step 2 is satisfied in $\mathcal{T}_3$. There is only one remaining class of wffs of $\mathcal{T}_3$ that $\mathcal{M}_3$ might not satisfy: the formula from Step 3.

Let f be an atom, and let b be a binding for all the variables of f. If $((f)_\sigma)_b$ and $((H(f,U))_\sigma)_b$ have the same truth valuations in $\mathcal{M}_3$, then the formula of Step 3 is satisfied when x_1 through x_n are bound to the corresponding arguments of f. If they have different truth valuations in $\mathcal{M}_3$, then $(f)_\sigma$ must a member of $(\mathcal{F})_\sigma$, and $((f)_\sigma)_b$ must appear in v, and therefore also in $((\omega)_{b'})_\sigma$ for some $b' \in \Sigma_\phi$. This implies that the formula of Step 3 is satisfied for f, since $((\phi)_{b'})_\sigma$ is true in $\mathcal{M}_3$. We conclude that $\mathcal{M}_3$ is a model of $\mathcal{T}_3$, and the alternative world of $\mathcal{M}_3$ is produced by the Update Algorithm. This concludes the proof of correctness for the Update Algorithm. $\Diamond$

4.5. Summary and Conclusion

In this section we have extended the definitions and algorithms of Chapter 3

to include updates containing variables and extended relational theories with arbitrary formulas in their bodies. A very simple update algorithm, Version III, was proven sufficient to perform these updates when quantifiers and variables are permitted in the theory bodies. If quantifiers are not allowed, Version II of the Update Algorithm may be used. The time complexity and the length of the wffs added to the extended relational theory in Version II are V times greater than those of Version I, where V is the number of sets of substitutions for the variables of the update that are to be considered during update processing. In contrast, Version III adds to the extended relational theory only a number of atoms that is linear in the size of the update request, but makes query processing more difficult. Version III is also of interest as a method of updating arbitrary logical theories.

Chapter 5: Lazy Evaluation of Updates

Delayed, but nothing altered. —Shakespeare, Romeo and Juliet *i*.4

As Chapters 1–4 have shown, first–order logic provides an adequate framework for an examination of updates to databases containing incomplete information. However, from a practical point of view, updates can be quite expensive when Skolem constants occur in the extended relational theory. The cost of an update can be measured as a function of the increase in the size of the theory that would result from execution of the update, and by measures of the expected time to execute the update and to answer subsequent queries. Once a database administrator has established a policy on when an update is too expensive, the techniques of this chapter can be used to recognize and defer or reject too–expensive updates and queries. This involves use of a lazy evaluation technique to delay execution of expensive updates as long as possible.

Recall from Theorem 3-2 that when the extended relational theory $\mathcal{T}$ contains n atoms containing Skolem constants, a series of m updates of size k each may cause the size of $\mathcal{T}$ to grow by $\mathcal{O}(nmk + m^2k^2)$. This potential growth is much too large, yet large growth (at least $\mathcal{O}(nm)$) is unavoidable if the effect of an update is to be represented directly in the extended relational theory, for in the worst case every datom of $\mathcal{T}$ that unifies with a datom of the update *must* be changed in some way in $\mathcal{T}$. In some sense the information content of a single update is no more than its size, k, and so growth of more than $\mathcal{O}(mk)$ after m updates is too much. We can achieve growth of no more than $\mathcal{O}(mk)$ by simply storing the updates without incorporating them into $\mathcal{T}$. However, since the usual means of query answering presupposes some means of integrating updates with the rest of the database to allow satisfiability testing, a means of at least temporary incorporation must be offered. This chapter puts forth a scheme of delayed evaluation and simplification of expensive updates based on bounding the permissible number of unifications for the atoms of an incoming update. We begin with a general overview and a series of examples.

There is a lot to be said about lazy evaluation, and only part of this story is told here. As this chapter began to loom over the others in sheer bulk, the author chose to err on the side of informality rather than overload. Died-in-the-wool theorists will recognize that numerous additional theorems must be included in any definitive treatment of lazy evaluation; non–theorists will see a need for further elaboration and refinement of the cost estimation techniques used in lazy evaluation.

5.1. Overview and Motivation

The first element of a system for cost reduction of too-expensive updates is a cost evaluation function, so that we can decide which updates are too expensive to execute. If an incoming update U is determined to be too expensive, we will not execute U, but instead set U aside in the hopes that either no queries will be asked that require processing U completely, or intervening updates will reduce the cost of U sufficiently before it must be executed.

As the main data structure for this *lazy evaluation scheme*, we propose to use a *lazy graph*, a directed acyclic graph that keeps track of data dependencies between updates. The lazy graph helps minimize the amount of updating that must be performed before executing an incoming query Q, and keeps track of relevant update sequencing information. Some examples will clarify the potential benefits.

Example. The effect of the two updates `INSERT` Emp(Reid, CSD) `WHERE` T and `INSERT` ¬Emp(Reid, CSD) `WHERE` T is dependent upon the order in which they are executed; if these two are stored away for lazy execution, we must make sure that any eventual processing of them is done in the order in which they were received. On the other hand, neither of these two conflicts with the update `INSERT` Emp(Reid, CSL) `WHERE` T, which could be performed before, after, or between the other two. ◇

This example suggests a parallel between lazy evaluation sequencing control and concurrency control [Papadimitriou 86]. The main difference is that in database concurrency control, any execution equivalent to some serial execution is correct, while sequencing control requires that the execution be equivalent to the original update input order.

Example. Suppose the update U': `INSERT` ϵ =CSD `WHERE` T is received while the update U: `INSERT` Emp(Reid, ϵ) `WHERE` T is still unexecuted. Unlike information about the truth valuations of datoms, information about the bindings of Skolem constants is permanent and once asserted can never be refuted, only refined. (For example, if the user follows U' by the update `INSERT` ϵ=CSL `WHERE` T, then $\mathcal{T}$ will become inconsistent.) This pleasant property of permanence allows us to use the new information in U' about the value of ϵ to simplify not only $\mathcal{T}$, but also the pending update U: U can now be reduced to `INSERT` Emp(Reid, CSD) `WHERE` T, which may well be affordable enough to execute directly even if `INSERT` Emp(Reid, ϵ) `WHERE` T is not. ◇

Example. Another potentially useful feature is the ability to execute only part of an update, leaving the more expensive part for later incorporation. For example, suppose the update U: `INSERT` Emp(Reid, ϵ)∧Mgr(Nilsson, CSD) `WHERE` T is too expensive only because Emp(Reid, ϵ) unifies with too many datoms of $\mathcal{T}$. If a user later asks a query involving only Mgr(Nilsson, CSD), it is advantageous to split U into the two updates U_1: `INSERT` Mgr(Nilsson, CSD) `WHERE` T and U_2: `INSERT` Emp(Reid, ϵ) `WHERE` T and only execute U_1 before processing the query.

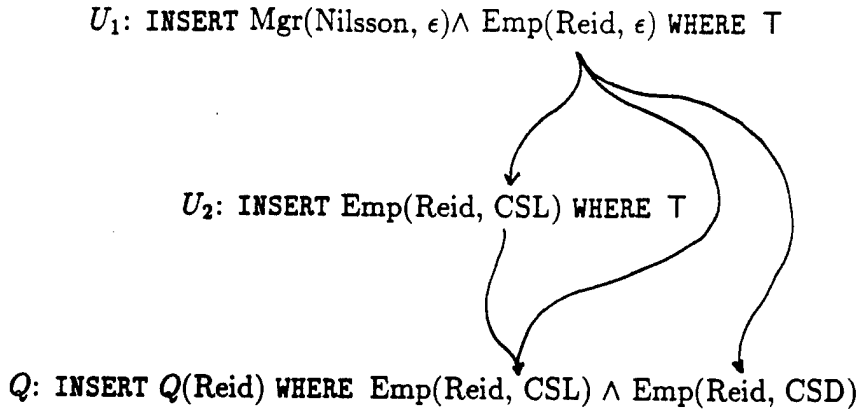


Figure 5-1. Example of lazy evaluation.

Example. Suppose an update U_1: INSERT Mgr(Nilsson, ϵ) ∧ Emp(Reid, ϵ) WHERE T arrives in the system, followed by the update U_2: INSERT Emp(Reid, CSL) WHERE T. Then U_2 and possibly U_1 as well contain new information about the truth valuation of Emp(Reid, CSL); both of these updates may write new information about Emp(Reid, CSL) into $\mathcal{T}$. In the language of concurrency control, there is a *write/write conflict* between Emp(Reid, ϵ) in U_1 and Emp(Reid, CSL) in U_2; the lazy graph of figure 5-1 depicts these relationships. Suppose that the query Q: INSERT Q(Reid) WHERE Emp(Reid, CSL) ∧ Emp(Reid, CSD) arrives next. (We have not formally defined queries yet; think of them as establishing a new relation that gives a view of the current database.) A *read/write conflict* occurs when one update "reads" a datom (i.e., the datom occurs in ϕ) that a later update "writes". There are read/write conflicts between Emp(Reid, CSL) of Q and Emp(Reid, ϵ) of U_1 and Emp(Reid, CSL) of U_2, and between Emp(Reid, CSD) of Q and Emp(Reid, ϵ) of U_1, as depicted in figure 5-1.

Assuming that both Emp(Reid, ϵ) and Mgr(Nilsson, ϵ) in U_1 are too expensive to execute because they unify with too many datoms of $\mathcal{T}$, the best procedure is to first split Mgr(Nilsson, ϵ) out of U_1, as depicted in figure 5-2, creating updates U_3 and U_4.

Then U_4 needs to be split on the two substitutions ϵ=CSL and ϵ =CSD, creating updates U_5, U_6, and U_7, depicted in figure 5-3. At this point Q and the updates Q depends upon are more likely to be affordable. ◊

With the algorithm and data structures presented in this chapter, if a query is rejected due to excessive expense, exact reasons for the high cost can be made available to the caller, so that assertions about the possible bindings for Skolem constants may be used to reduce the amount of uncertainty in the database and render the query affordable. Furthermore, any new binding information can be used to reduce the size of the extended relational theory, in effect retroactively reducing the cost of all earlier updates that contained those Skolem constants.

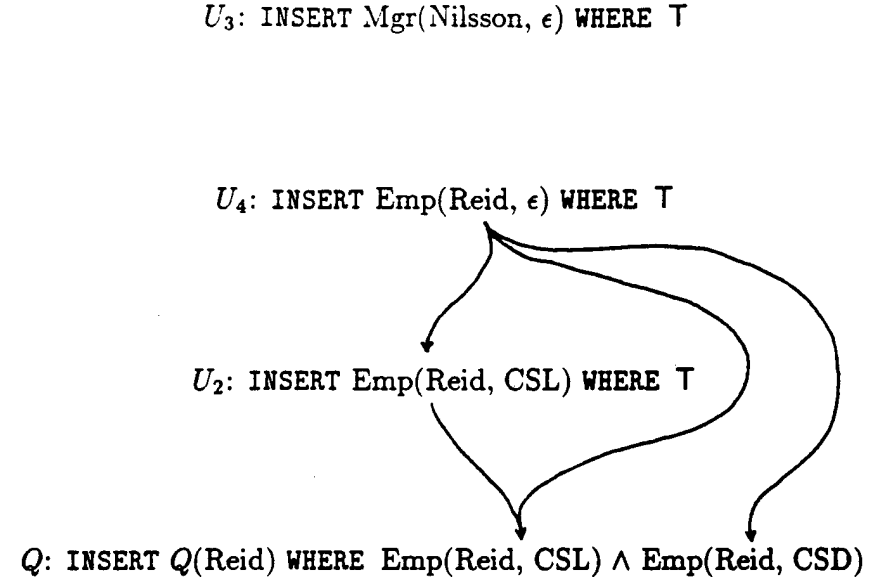


Figure 5-2. Lazy evaluation horizontal split.

These examples should suffice to give a flavor of the possible advantages of a lazy evaluation scheme. We now turn to the details of lazy evaluation, beginning with a definition of queries. The lazy graph data structure is then presented formally, followed by an algorithm for adding incoming updates and queries to the lazy graph. After a presentation of the Lazy Algorithm, the remainder of the chapter is devoted to a discussion of splitting techniques. The chapter concludes with a measure of the benefits afforded by lazy evaluation.

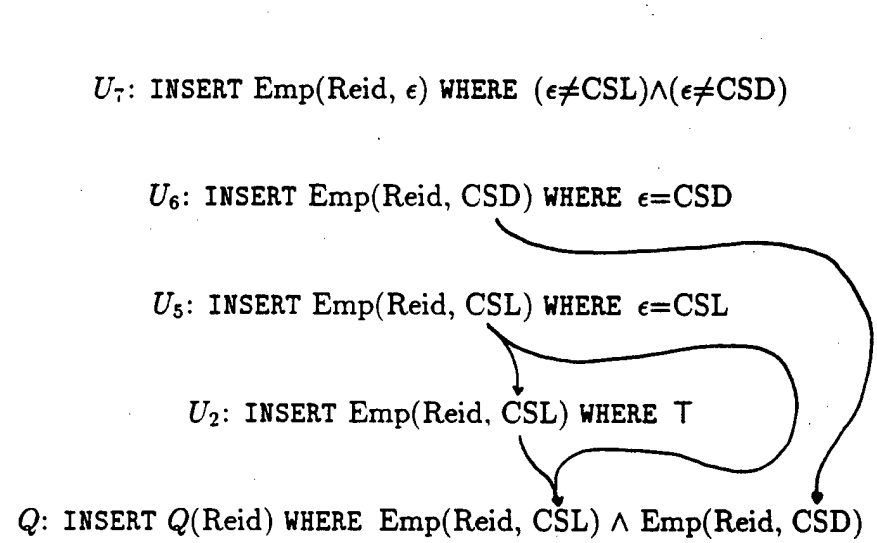


Figure 5-3. Lazy evaluation vertical split.

5.2. Queries

We define a query as a temporary materialized view, to wit, a short–lived relation. In keeping with our emphasis on mechanism rather than policy, we do not define what the user should actually "see" as output from a query. A user interface routine will be in charge of optimizing and reformulating the view relation produced by the query execution mechanism into a format judged acceptable for human or programmatic consumption. In this thesis only the process of creation of that view relation is of concern, not its display.

Syntactically, queries take the form `INSERT` Q $(c_1, \ldots, c_n)$ `WHERE` ϕ, where ϕ is a wff of the language $\mathcal{L}$ not containing history atoms or variables, Q is an n-ary predicate not in $\mathcal{L}$, and c_1 through c_n are constants or Skolem constants of $\mathcal{L}$. Note that Q cannot contain variables. Of course, in any database application, queries almost always contain variables, so this may seem a peculiar choice of definition for Q. The goal of this chapter, however, is to explore the issues arising in lazy evaluation and to present mechanisms for the basic tasks of lazy evaluation, much as the goal of Chapter 3 was to introduce a semantics for updates and to explain the basic technique for implementing such a semantics in polynomial time. As was the case in Chapter 3, the presence of variables in the operations under consideration would only obscure the principles at play. For that reason variables are not permitted in queries in this chapter. In like manner as the incorporation of variables into updates in Chapter 4 did not require major departures from the paradigms laid down in Chapter 3, the generalization of lazy evaluation to queries and updates containing variables will not involve radical changes in the techniques proposed here.

When the query Q arrives, the first step in handling Q is to add the new predicate Q to $\mathcal{L}$ and create a completion axiom $\forall x_1 \cdots \forall x_n \neg Q(x_1, \ldots, x_n)$ and add it to $\mathcal{T}$. (Q and its completion axiom can be flushed from the system once the user interface routine is done with it.) Q is then added to the lazy graph like any ordinary update request (Section 5.5). In fact, the only major difference between a query and an ordinary update request is that query Q must be either executed or rejected right away. The Lazy Algorithm (Section 5.6) will determine whether to accept or reject Q.

5.3. Cost Estimation

The first element of a system for lazy evaluation of too-expensive updates is a cost estimation function, so that we can decide which updates are too expensive to execute. Recall that one precious commodity in the system is the space required for extended relational theory storage. In fact, in the update algorithms discussed in previous chapters, the time to execute an update was just a logarithmic factor higher than the amount of additional space that the update added to $\mathcal{T}$. In lazy evaluation, the time required to answer a query will be traded off against the amount of space occupied by the extended relational theory; with lazy evaluation a large number of unexecuted updates may require attention before a query can

be answered. The techniques proposed in this chapter have the goal of minimizing storage space, necessarily to the detriment of query response time. In other words, in this discussion of lazy evaluation, an expensive update is one which adds too many atoms to the extended relational theory.†

The 80/20 rule says that in an ordinary database, 80% of the queries reference at most 20% of the data; 80% of that 80% (i.e., 64%) only reference at most 20% of that 20% (i.e., 4%); and so forth. Because of the 80/20 rule, we have assumed that executed updates are permanently incorporated into the extended relational theory $\mathcal{T}$. The alternative is to integrate the update with $\mathcal{T}$ during query execution, but then abort the update at the end of query execution to save space in $\mathcal{T}$. However, the 80/20 rule implies that if an update requires execution once, it will probably require execution again, and we might as well save the recomputation costs. Note, however, that this is based on a particular tradeoff between computation and storage costs, and one might take a different view in a system where processing was expensive and storage was affordable.

The amount of space consumed by an update U is proportional to the number of *relevant* (in a sense to be made precise later) unifications of datoms in $\mathcal{T}$ with atoms of U. To control the amount of space consumed by U, lazy evaluation estimates the number of datoms added to $\mathcal{T}$ by each step of the Update Algorithm while U is being executed, and refuses to execute U if this estimate is excessive.

The cost estimate and cost bound for an incoming update or query are to be computed by functions supplied by the database administrator. The cost functions must satisfy the following requirements:

1. The cost estimation function may overestimate but never underestimate the costs (as defined by the database administrator) associated with a set of updates.
2. The cost estimate function and bound function must be computable from the information stored in the lazy graph.

The cost information provided in the lazy graph includes a count of the number of datoms in $\mathcal{T}$ that unify with datoms of ω, as these unifications cause most of the expense incurred when executing an update. The cost estimation function will presumably rely heavily on this unification count. The obvious algorithm for unification counting is to use index lookup and Skolem constant instantiation until no more relevant unifications are found or else the cost of the unifications found so far exceeds the cost bound. For example, to count the number of datoms of $\mathcal{T}$ that unify with Emp(Reid, CSD), assuming that the database has indices on both Employees and Departments, begin by looking up Reid and all Skolem constants in the Employees index, and look up CSD and all Skolem constants in the Departments index. Then do a set intersection on the

† If query response time is a problem, then over-zealous lazy evaluation algorithms may be curbed by introducing constraints on the lazy graph (e.g., restrictions on height, flexible update cost limits, etc.).

two sets of tuple pointers thus generated, and count the number of pointers in the intersection.

Queries also have associated storage costs, for their temporary view relations. The bound function might well choose to allot much more space to queries than to updates, since that space will only be used temporarily.

Unification counting and cost estimation should be performed with a bit of optimization, and that is where the phrase "relevant occurrences" comes into play. The algorithms below use a test for satisfiability of bounded–length formulas to determine relevance. Other optimizations are also possible: an efficient implementation of the cost estimation procedure given below might do a much more thorough job of detecting spurious unifications. For example, any obviously "impossible" substitutions can be discounted: though Emp(Reid, CSD) unifies with Emp(ϵ, CSD), there is no need to count that unification if the wff where Emp(ϵ, CSD) occurs in $\mathcal{T}$ is Emp(ϵ, CSD)$\wedge\epsilon$=Nilsson; that unification is not relevant, because the two wffs are not simultaneously satisfiable. Such optimizations will be part of the heuristic component of an implementation of the Update Algorithm, and will be important also for any user interface routine for query answering. The choice of optimizations beyond that required by algorithms given here is left to the implementor.

5.4. The Lazy Graph

The lazy graph is the data structure needed for lazy evaluation. In the lazy graph, nodes represent the atoms of updates. *Update hyperedges* group atoms into updates. *Family hyperedges* associate updates that are descended via splitting from the same original update. In addition, there is a directed arc between two nodes if the atom labels of the two nodes unify and cause one update to become dependent upon the results of the other. More formally, the lazy graph contains the following information:

1. A set of *nodes*. Each node is labelled with a datom or history atom, and cost information.

2. A set of *update hyperedges*. Each node is on one update hyperedge. Each update hyperedge is labelled with an update or query, such as U: `INSERT` Emp(ϵ, CSD) `WHERE` T, and flagged as being either unexecuted (hereafter called *pending*) or executed.

3. A set of *family hyperedges*. Each node and update hyperedge is contained in one family hyperedge. Each family hyperedge is labelled with an update or query, such as U: `INSERT` Emp(ϵ, CSD) $\wedge$ Mgr(Nilsson, CSD) `WHERE` T, and flagged as being either an update or query. In addition, each family hyperedge has an associated cost bound.

4. A set of directed *labelled arcs* between nodes. Each arc is labelled with a substitution. These arcs represent dependencies between updates.

5. A set of directed *unlabeled arcs* between nodes. These arcs represent implied dependencies, such as that between ω of an update and ϕ of the same update.

We have chosen not to store cost estimate information for equality atoms, and hence they are not included in the lazy graph. This choice was made because equality atoms will be instrumental in reducing the size of the extended relational theory by eliminating Skolem constants, and we therefore felt that an actual and estimated cost of zero was most appropriate for any optimized implementation of the Update Algorithm.

The main expense in an update is typically due to datoms in ω. For example, if U_1 is the update INSERT Emp(Reid, CSD) $\vee$ Emp(Reid, CSL) WHERE T, and U_2 is the update INSERT ϵ=CSD $\vee$ ϵ=CSL WHERE T, then these two updates have the same size. Yet a count of the wffs added by the Update Algorithm shows that U_1 will cost *at least* 5 times as much as U_2 under the Update Algorithm—and that minimum is attained if no datom of ω of U_1 is an implicit subformula of $\mathcal{T}$. Therefore it seems reasonable for non-data atoms (i.e., atoms that are not datoms) to be assigned much lower cost estimates than other types of atoms in ω. For datoms g that occur only in ϕ, again a lower estimate would be appropriate. Except for Step 1 of the Update Algorithm, g will take up no more space than a non-data atom, so only if Step 1 is required for g will g be more expensive than a non-data atom.

We distinguish between update and query *execution*, or the incorporation of an update or query into the extended relational theory; update and query *processing*, or the act of reforming the lazy graph to make a particular update or query executable; and update and query *addition*, or the act of adding a new update or query to the lazy graph. These three phases are the topics of the next three sections.

5.5. The NAP Algorithm: Addition of Incoming Updates and Queries to the Lazy Graph

The NAP algorithm will be used in two scenarios: When an update or query U arrives in the system, the NAP algorithm adds U to the lazy graph as the first member of a new update family. If U needs to be incorporated into the extended relational theory, we then process and execute U. In addition, when an update is split into two subupdates, the NAP algorithm is called to add those subupdates to the lazy graph. In this case, the split–off updates are members of the same update family as the original update.

The NAP algorithm talks about updates containing history atoms. History atoms in updates! Is nothing sacred? Fear not, users still cannot mention history atoms in updates; history atoms are only present for technical reasons: they creep in when an update is split. For now, ignore any mysterious terminology, and all will be revealed in Section 5.7.

A helpful example of the operation of the NAP algorithm appears in Figure 5-4.

The NAP (Node Addition Procedure) Algorithm.

Input: A lazy graph G and a request U, flagged as an update or query; and the preexisting update family to which U belongs, if any.

Output: A new lazy graph G' containing U.

Procedure: A sequence of three steps:

Step 1. Add nodes and hyperedges. For each non-equality atom g of U, add a node labelled g to G. Add a new update hyperedge to G containing exactly the new nodes, and label that hyperedge with U. If U defines a new history atom, also add a node labelled with that atom to the hyperedge. Mark the update hyperedge as pending. If U is to be part of a preexisting update family, then add its nodes to the hyperedge for that family; otherwise (1) create a new family hyperedge, labeled with U, (2) mark the family hyperedge as a query or update, as appropriate, and (3) compute the family hyperedge cost bound. Call the new graph G'.

Step 2. Add relevant arcs. *Intra-update arcs.* Let n and n' be two different nodes on the update hyperedge for U. If n is in ω of U and n' is in ϕ of U, then add an unlabeled arc from n to n'. These arcs represent the fact that the truth valuations for the atoms in ω after U is executed will depend upon the truth valuations for the atoms of ϕ at the time U is executed.

History atom definition arcs. If a history atom h of U also is the label of a node of a pending update U', then add an unlabeled arc from h in U' to the node h of U. This ensures that history atoms are defined before they are used.

Inter-update arcs. If any update hyperedges other than U are pending, then the effect of executing U may depend upon the results of those other updates. Let U' be a pending update hyperedge of G' other than U. Let f be the label of a node on the update hyperedge U', and g the label of a node on the update hyperedge U. Place a directed arc labelled σ from node f to node g if

(1) f unifies with g under the most general substitution σ; and

(2) $\sigma \wedge \phi_U$ and $\sigma \wedge \phi_{U'}$ are both satisfiable; and

(3) if ϕ_U logically entails a wff α containing only equality atoms, then $\phi_{U'} \wedge \alpha$ is satisfiable; and either

(4a) (write/read conflict) f is a subformula of ω of U' and g is a subformula of ϕ of U; or

(4b) (read/write conflict) f is a subformula of ϕ of U' and g is a subformula of ω of U; or

(4c) (write/write conflict) f is a subformula of ω of U' and g is a subformula of ω of U.

Explanations and examples of these tests appear after the algorithm.

Step 3. Record cost information. As input to the cost estimation function, cost information must be recorded for each node of U that is labelled with a datom g. Record whether g is a subformula of $\mathcal{T}$ or is the label of any ancestor of g in the lazy graph. (In the latter case, g would appear in $\mathcal{T}$ by the time U is executed.) Also record the number of different datoms occurring in $\mathcal{T}$ or on

labels of ancestors of g that unify with g, up to a preset limit l. If the limit l is reached, then also record the fact that the unification count terminated early due to cost overrun. $\Diamond$

The unification count limit l is used to ensure that estimating the cost of an overly–expensive update does not take as much time as it would to execute it. The correct value for the limit l depends on the cost bound for that particular update family, and should be set so that the unification count for any affordable update will not exceed l.

In Step 2 of the NAP Algorithm, tests (1), (2), and (3) ensure that the conflict is relevant. If test (1) is failed, then there can be no conflict between U and U' on the basis of f and g, because those two atoms do not even unify. For example, Emp(Reid, CSD) and Mgr(Nilsson, CSD) cannot cause a conflict.

Test (2) of Step 2 ensures that the unification under which the conflict occurs can actually materialize in some model. For example, let U be `INSERT` Emp(Reid, ϵ) `WHERE` Mgr(Nilsson, ϵ) $\wedge((\epsilon = \text{EE}) \vee (\epsilon = \text{CSL}))$, and let U' be `INSERT` Emp(Reid, CSD) `WHERE` T. Then σ is ϵ = CSD, and U and U' can only conflict in models where ϵ is CSD. But in any model where ϵ is CSD, the selection clause ϕ of U must be false. Therefore U and U' cannot conflict.

Test (3) ensures that U' and U can take place in "overlapping" sets of alternative worlds. Test (3) is a useful heuristic for reducing the number of arcs in the lazy graph without incurring much additional expense. For example, suppose U_1 is `INSERT` Emp(Reid, CSD) `WHERE` ϵ = CSD and U_1 is `INSERT` Emp(Reid, CSD) `WHERE` ϵ = EE. Without test (3), a write/write conflict would be recorded between these two updates, even though in fact the updates must take place in disjoint sets of alternative worlds. Including this unnecessary write/write arc in the lazy graph would force extra serialization.

Example. Suppose the lazy graph contains the pending update U_1: `INSERT` Emp(Reid, ϵ_1)$\vee$Mgr(Nilsson, ϵ_2) `WHERE` T, and the update U_2: `INSERT` Q(CSD) `WHERE` Emp(Reid, CSD) arrives. Figure 5-4 shows the new lazy graph minus cost information. $\Diamond$

All Skolem constants and history atoms occurring in a pending update U are *pinned* in $\mathcal{T}$ until U has completed execution. This means that database optimization routines cannot remove those Skolem constants and formulas from the database, even if they are no longer logically necessary. For example, if the database system discovers that ϵ=Reid, at least the single wff "ϵ=Reid" must remain in $\mathcal{T}$ until all pending updates containing ϵ have been executed. (Alternatively, one might prefer to substitute the newly discovered values into the pending updates that reference them; for simplicity we do not consider this method.) If these atoms were not pinned, then errors might occur in execution. For example, suppose a user requests the update U_1: `INSERT` Emp(Reid, ϵ) `WHERE` T as soon as it becomes known that Reid is definitely a member of some department. Suppose that U_1 is too expensive to execute, and that U_1 is still pending two weeks later when the user discovers that Reid is in CSD, that is, that ϵ = CSD.

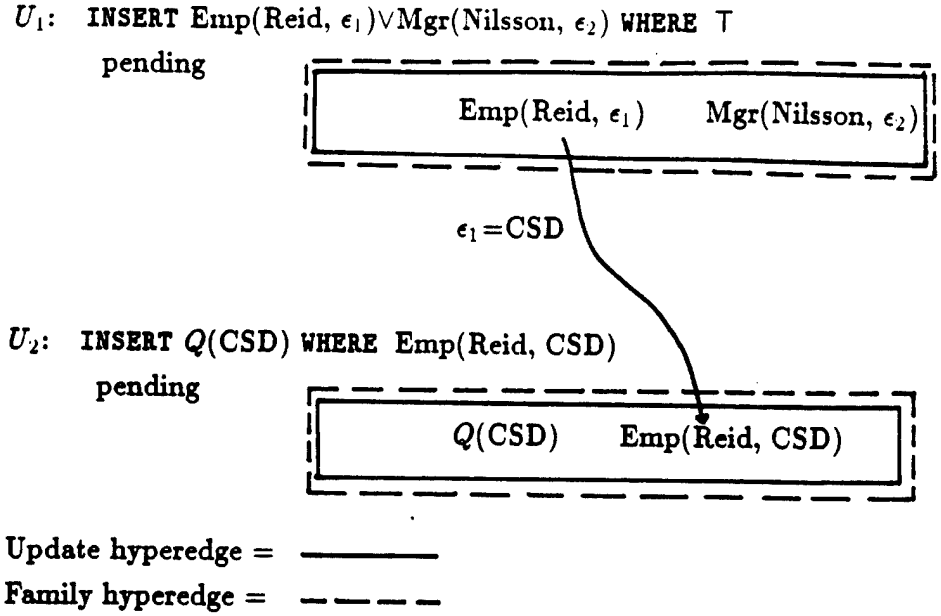


Figure 5-4. Lazy graph example.

This new update INSERT ϵ = CSD WHERE T is probably affordable, so assume that it is executed immediately. If all mention of ϵ is subsequently removed from the theory, and then U_1 is finally executed, U_1 will not add the fact that Reid is in CSD; rather, U_1 will erroneously declare that Reid is in some unknown and unrestricted department. For this reason, history atoms and atoms containing Skolem constants must be pinned.

It is important to show that the lazy graph does capture exactly the information needed to process all incoming updates correctly with the Update Algorithm. The arcs and hyperedges of the lazy graph induce a directed acyclic graph whose "nodes" are update hyperedges, and in which an arc goes from update U to U' if there is an arc in the lazy graph between a node of U and a node of U'. As is true of the entire lazy graph, this induced update graph contains no cycles, because when an update U is added to the lazy graph using the NAP algorithm, all new arcs go to nodes of U from nodes of preexisting updates. Therefore the lazy graph induces a partial order on updates, and one can use this ordering to sort the updates topologically. Recall that a topological sort of a directed acyclic graph is constructed by repeatedly selecting a root in the graph and deleting it and its incident arcs from the graph. If $U_1 \cdots U_n$ is a topological sort, then call the sequence $U_n \cdots U_1$ a *reverse* topological sort.

Theorem 5-1. Let $U_1 \cdots U_n$ be a sequence of updates and queries, and let T be an extended relational theory. Let G be the lazy graph created by sequentially inserting U_1 through U_n into an initially empty lazy graph. Let Toposort be

any reverse topological sort of all the updates in G. Then Worlds(Toposort($\mathcal{T}$)) = Worlds($U_n(\cdots(U_1(\mathcal{T}))\cdots)$). $\Diamond$

The proof of Theorem 5-1 uses a bit of new terminology:

Definition. Let n and n' be nodes in a lazy graph. Then n is an *ancestor* of n' if there is a path from n to n' in the lazy graph. If U and U' are update hyperedges, then U is an ancestor of U' if there is a path from a node of U to a node of U' in the lazy graph. $\Diamond$

Proof of Theorem 5-1. First, the sequence $S_1 = U_n \cdots U_1$ is a reverse topological sort of the lazy graph, because when an update or query U is inserted into the lazy graph with the NAP Algorithm, no new ancestors are created for any node except U. Let Toposort be a reverse topological sort other than S_1. There must be a rightmost position on which the two sorts differ; counting from the right, say that S_1 and Toposort agree on positions 1 through $i-1$, but differ in the ith position, where S_1 has U_i and Toposort has U_j. Let S_2 be the sort $U_n \cdots U_{j+1}U_{j-1}\cdots U_iU_jU_{i-1}\cdots U_1$. Since Toposort is a reverse topological sort, U_j must not have any ancestors in the sequence $U_{j-1}\cdots U_i$. In particular, U_{j-1} must not be an ancestor of U_j. Applying Lemma 5-1, Worlds($U_n \cdots U_{j+1}U_{j-1}U_jU_{j-2}\cdots U_1(\mathcal{T})$) = Worlds($U_n \cdots U_1(\mathcal{T})$). By induction, Worlds($S_2(\mathcal{T})$) = Worlds($S_1(\mathcal{T})$). By induction, it follows that Worlds(Toposort($\mathcal{T}$)) = Worlds($S_1(\mathcal{T})$). $\Diamond$

Lemma 5-1. Let $\mathcal{T}$ be an extended relational theory, and let U_1 and U_2 be updates or queries:

U_1: `INSERT` ω_1 `WHERE` ϕ_1,

U_2: `INSERT` ω_2 `WHERE` ϕ_2,

such that if first U_1 and then U_2 are inserted into a lazy graph G using the NAP Algorithm, U_1 is not an ancestor of U_2. Then Worlds($U_2(U_1(\mathcal{T}))$) = Worlds($U_1(U_2(\mathcal{T}))$). $\Diamond$

Proof of Lemma 5-1. Let $\mathcal{M}$ be a model of $\mathcal{T}$ having Skolem constant substitution σ with respect to $\mathcal{T}$, U_1, and U_2. Let $\mathcal{M}_1$ be a model of $U_1(\mathcal{T})$ such that World($\mathcal{M}_1$) $\in$ Worlds($U_1(\mathcal{T})$); and let $\mathcal{M}_2$ be a model of $U_2(\mathcal{T})$ such that World($\mathcal{M}_2$) $\in$ Worlds($U_2(\mathcal{T})$). Suppose that ϕ_1 is not satisfied in $\mathcal{M}$. Then World($\mathcal{M}$) = World($\mathcal{M}_1$), and Worlds($U_2(\mathcal{M}_1)$) = Worlds($U_2(\mathcal{M})$). Suppose U_2 is first applied to $\mathcal{M}$, producing a model $\mathcal{M}_1'$ of World($\mathcal{M}_2$), and then U_1 is applied to $\mathcal{M}_1'$. If U_1 does not change the alternative world of $\mathcal{M}_1'$, then World($\mathcal{M}_2'$) = World($\mathcal{M}_2$). If U_1 does change $\mathcal{M}_1'$, then though ϕ_1 was false in $\mathcal{M}$, ϕ_1 is true in $\mathcal{M}_1'$. Therefore there must be datoms f in ϕ_1 and g in ω_2 such that f unifies with g under σ. We claim that there is a read/write arc in the lazy graph between f and g, which violates the claim that U_1 is not an ancestor

of U_2. To see this, note that f and g pass test (1) of Step 2, as f and g unify under σ. For test (2), $\sigma \wedge \phi_1$ is true in $\mathcal{M}_1'$, and $\sigma \wedge \phi_2$ was true in $\mathcal{M}$, so both wffs are satisfiable. For test (3), suppose ϕ_2 logically entails α, where α consists of equality atoms. Then α is true in $\mathcal{M}$, and therefore also in $\mathcal{M}_1'$. As ϕ_1 is true in $\mathcal{M}_1'$, α must be consistent with ϕ_1. We conclude that if ϕ_1 is not true in $\mathcal{M}$, then Worlds($U_1(\mathcal{M})$) = Worlds($U_2(\mathcal{M})$). The proof is symmetric if ϕ_2 is false in $\mathcal{M}$, or if ϕ_2 is false in $\mathcal{M}_1$, or if ϕ_1 is false in $\mathcal{M}_1'$. We conclude that in all these cases, the lemma holds.

Now suppose that ϕ_1 is true in $\mathcal{M}$, and ϕ_2 is true in $\mathcal{M}_1$. Let $\mathcal{M}_1'$ be a model that agrees with $\mathcal{M}$ in all respects except for the truth valuations of the atoms of $(\omega_1)_\sigma$, which are the same in $\mathcal{M}_1'$ as in $\mathcal{M}_2$. Then World($\mathcal{M}_1'$) $\in$ Worlds($U_2(\mathcal{M})$). Now apply U_1 to $\mathcal{M}_1'$. If $(\omega_1)_\sigma$ and $(\omega_2)_\sigma$ are over disjoint sets of datoms, then World($\mathcal{M}_2$) $\in U_1(\mathcal{M}_1')$. Otherwise, we claim that there is a write/write conflict between U_1 and U_2, a contradiction. To see this, let f be a datom of ω_1 and g a datom of ω_2 such that f and g unify under σ. Step 2 of the NAP algorithm contains three tests for f and g: (1) f and g must unify, which they do by definition; (2) $\sigma \wedge \phi_1$ and $\sigma \wedge \phi_2$ are both satisfied, by assumption; and as ϕ_1 and ϕ_2 are both true in $\mathcal{M}$, test (3) is also satisfied. The symmetric proof holds if first U_2 and then U_1 is applied to $\mathcal{M}$. We conclude that Worlds($U_2(U_1(\mathcal{M}))$) = Worlds($U_1(U_2(\mathcal{M}))$). $\Diamond$

5.6. The Lazy Algorithm

When can a pending update U be executed? The cardinal rule is that U may be executed now if *U is affordable and all its ancestors in the lazy graph have been executed.* This determination is made by examining each update family in the lazy graph G. For U to be affordable, within each update family of G, the costs of the ancestors of U plus the costs of previously executed members of the family cannot exceed the cost limit for the family.

For example, let U be the incoming update `INSERT` Emp(Reid, CSD) `WHERE` T. Suppose the relevant portion of the lazy graph is as in figure 5-5. Summing estimated costs (actual costs may be used for U_1 if available), it appears that no splits will be needed in this lazy graph if the cost limit l is at least 10. If the cost limit is less than 10, a split of U_2 is the most appropriate course of action.

As another example, the update `INSERT` ϵ = CSD `WHERE` T must be a root in the lazy graph, since it contains no datoms or history atoms; if its estimated cost is zero, then it may be executed at any time.

The test for affordability may be described more formally as follows.

Definitions. Let S be a set of updates and/or queries in a lazy graph. For each family $\mathcal{F}$ with an update or query in S, let $S(\mathcal{F})$ be the set of all updates or queries in family $\mathcal{F}$ that are in S or have already been executed. Then S is *affordable* if for each family $\mathcal{F}$ with an update or query in S,

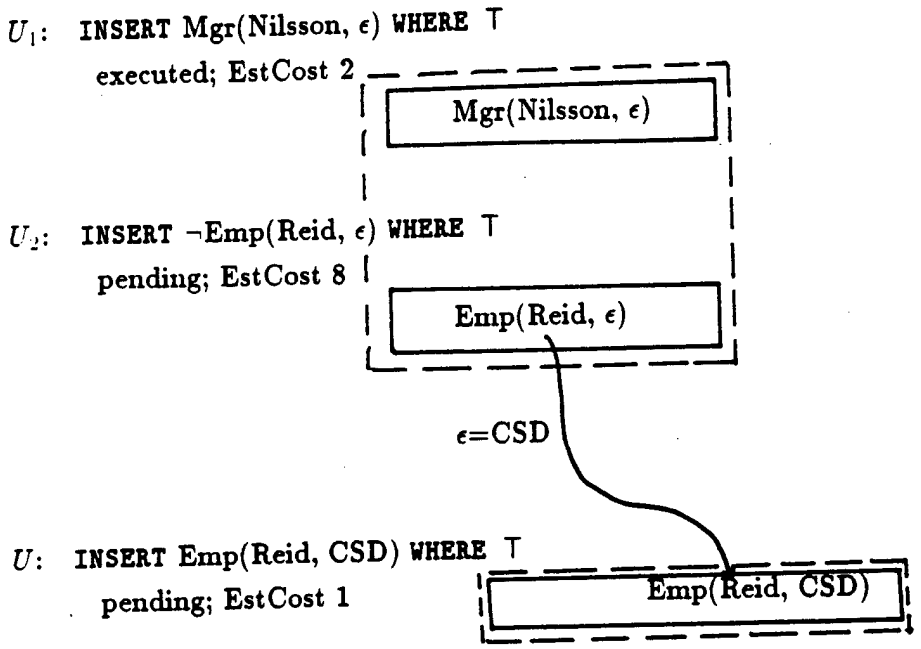


Figure 5-5. Determining whether an update is affordable.

$$\text{CostLimit}(\mathcal{F}) \quad \geq \quad \sum_{U \in S(\mathcal{F})} \text{EstCost}(U),$$

i.e., if the amount spent on executed updates and queries of $\mathcal{F}$ plus the amount estimated for updates and queries of $\mathcal{F}$ that are in S is no more than the cost limit for $\mathcal{F}$. If S is not affordable, then S is *expensive*. ◇

The Lazy Algorithm non-deterministically processes a query or update U of the lazy graph, working U into an executable position by splitting its ancestors to reduce their costs.

The Lazy Algorithm.

Input. A lazy graph G with one particular node, U, that is to be processed. Initially all nodes of G are marked as being *unexamined*.

Output. An equivalent version of G and either an ACCEPT or REJECT verdict. If the verdict is ACCEPT, then all ancestors of U in G are now affordable. If the verdict is REJECT, then the cause of the rejection is also returned.

Procedure. A sequence of three steps:

Step 1. Accept U. If the set of all ancestors of U in G is affordable, then terminate with an ACCEPT verdict.

Step 2. Reject U. If the examined ancestors of U are expensive, send the user a REJECT verdict along with information on the reason for the rejection. This information may include the family and update hyperedge labels and all family cost information for every update and query on any path from the expensive ancestor to U. Then restore the lazy graph to its original state and terminate execution.

Step 3. Split ancestors. Choose a nearest† pending unexamined ancestor U' of U. Guess a sequence of splits for U', and perform them using the Splitting Algorithm. Mark U' as examined, if it still exists; otherwise mark the updates split off from U' as examined. Go back to Step 1. $\Diamond$

If the Lazy Algorithm accepts node U, then to execute U, choose an affordable pending ancestor update U' whose nodes are all roots in the lazy graph G. Execute U', afterwards marking that hyperedge as executed. Repeat until U itself has been executed and so marked. If every update hyperedge in a family hyperedge has been executed, then all nodes, hyperedges, and incident arcs of that family can be removed from G.

In the case of a REJECT verdict, the Lazy Algorithm may return a great deal of information to the user. This is because there are many possible ways to make an update cheaper, including retroactively reducing the cost of previously executed members of an update family. To make the best choice for cost reduction, the user may need all that information.

For the Lazy Algorithm to work according to expectations, it must satisfy a number of requirements. First, if the Lazy Algorithm accepts an update or query U, then no family cost bounds may be exceeded during execution of the ancestors of U. Fortunately, this follows immediately from Step 1 and the fact that the cost estimate function is guaranteed not to underestimate costs as defined by the database administrator.

Second, we must show that the splits performed in Step 3 of the Lazy Algorithm map one correct lazy graph into another "equivalent" graph. The following section presents a large repertoire of splitting techniques and proves that they meet this requirement.

Finally, Theorem 5-1 guarantees that the extended relational theory will reach a correct final state as long as the updates in the lazy graph are executed in topographical sort order. However, we need a characterization of the *intermediate* state of the extended relational theory, in particular, of the state of the extended relational theory when an update or query U and all its ancestors have just been executed; for that is the state that the user glimpses. Intuitively, for U a leaf of

† A *nearest ancestor* of U with property P, if one exists, is an ancestor U' of U with property P such that no other ancestor of U with property P has a shorter path to U than U' does.

the lazy graph, at an intermediate stage the alternative worlds of the extended relational theory are correct *when projected onto just the atoms in the update or query* U.

To explore this last point more formally, new terminology is in order. We distinguish the case where ω contains the equality predicate or is unsatisfiable (an *assertion*). Assertions are different from other updates in that they may eliminate some alternative worlds of a theory to which they are applied. Updates that are not assertions, on the other hand, cannot eliminate any alternative worlds of a theory: for if ω is satisfiable and does not contain the equality predicate, an insertion always produces some model from any model to which it is applied. If ω is satisfiable but contains the equality predicate, then the update may eliminate some models by invalidating their Skolem constant mappings. For example, "ϵ = CSD" will eliminate all models where ϵ is not mapped to CSD. The distinction between these types of updates is important because most users will want execution of any query to force execution of all† pending assertions, because assertions may affect the answer to the query by eliminating the alternative worlds where some potential answers to the query are true.

Definition. Let S be a set of datoms. Let $\mathcal{T}$ be an extended relational theory, and let $\mathcal{M}$ be a model of $\mathcal{T}$ with Skolem constant substitution σ with respect to S. Then World($\mathcal{M}$) *restricted to* S (written World($\mathcal{M}$)$|S$) is the wff form†† of the truth valuations in $\mathcal{M}$ of atoms in $(S)_\sigma$. Further,

$$\text{Worlds}(\mathcal{T})|S = \bigcup_{\mathcal{M} \in \text{ Models}(\mathcal{T})} (\text{World}(\mathcal{M})|S). \quad \Diamond$$

Theorem 5-2. Suppose the updates and queries of a lazy graph G formed by the NAP algorithm have topological sort $U_1 \cdots U_n Q$. Let S be the set containing all datoms of Q, and let $\mathcal{T}$ be an extended relational theory. If Toposort is a reverse topological sort of the ancestors of Q, then Worlds(Toposort($\mathcal{T}$))$|S$ $\subseteq$ Worlds($Q(U_n(\cdots U_1(\mathcal{T})\cdots))$)$|S$. Further, if Toposort+Assertions is a reverse topological sort of Q and the assertions in G, and all their ancestors, then Worlds($Q(U_n(\cdots U_1(\mathcal{T})\cdots))$)$|S$ = Worlds(Toposort+Assertions($\mathcal{T}$))$|S$. $\Diamond$

Proof of Theorem 5-2. Choose a particular topological sort Fulltoposort of all the updates and queries of G. Let Toposort+Assertions be derived from Fulltoposort by deleting all updates and queries of Fulltoposort that are not assertions or ancestors of Q. Let Toposort be derived from Toposort+Assertions by deleting all updates and queries that are not ancestors of Q. Then by Theorem 5-1, Worlds(Fulltoposort($\mathcal{T}$)) = Worlds($Q(U_n(\cdots U_1(\mathcal{T})\cdots))$). It therefore suffices to show that Worlds(Fulltoposort($\mathcal{T}$))$|S$ = Worlds(Toposort+Assertions($\mathcal{T}$))$|S$, and Worlds(Toposort($\mathcal{T}$))$|S$ $\subseteq$ Worlds(Fulltoposort($\mathcal{T}$))$|S$.

† Well, up to the limits imposed by the user's patience.

†† A truth valuation v can be written in wff form as a conjunction of literals, such that the atom a is a conjunct of v in wff form iff a receives the truth valuation T under v, and $\neg a$ is a conjunct of v in wff form iff a receives the truth valuation F under v.

There must be a rightmost position in which Toposort+Assertions and Fulltoposort contain different updates or queries. Suppose that the occupant of that position is U in Fulltoposort. Then U does not appear in Toposort+Assertions. Therefore U is not an ancestor of Q or an assertion. Let $\mathcal{M}$ be a model of $\mathcal{T}$ with Skolem constant substitution σ with respect to Q, U_1 through U_n, and $\mathcal{T}$. When U is applied to $\mathcal{M}$, it may change the alternative world of $\mathcal{M}$ but it cannot eliminate that world, as ω of U must be satisfiable. If ϕ of U is false in $\mathcal{M}$, then U does not change the alternative world of $\mathcal{M}$, and so eliminating U from Fulltoposort would not change the alternative worlds eventually produced from $\mathcal{M}$. If ϕ is true in $\mathcal{M}$, then U may change the alternative world of $\mathcal{M}$. However, U is not an ancestor of Q. If $(U)_\sigma$ has no datoms in common with any member of $(\text{Left}(U))_\sigma$, that is, the sequence of queries and updates $Q \cdots U'$ appearing to the left of U in Fulltoposort, then $\text{Worlds}(\text{Left}(U)(U(\mathcal{M})))|S = \text{Worlds}(\text{Left}(U)(\mathcal{M}))|S$. If a datom or history atom f of ω_U unifies under σ with an atom g of U', for U' any member of $\text{Left}(U)$, then it must be the case that test (2) or (3) of Step 2 of the NAP algorithm is violated for f and g, i.e., that $\phi_{U'}$ must be false in $\mathcal{M}$ and in all descendants of $\mathcal{M}$. In this case, when U' is executed, it cannot change the alternative world of $\mathcal{M}$ or any descendant of $\mathcal{M}$. Therefore eliminating U from Fulltoposort cannot change the effect of U'. We conclude that U can be removed from Fulltoposort without changing the alternative worlds of Fulltoposort($\mathcal{T}$) restricted to S. By induction, $\text{Worlds}(\text{Fulltoposort}(\mathcal{T}))|S = \text{Worlds}(\text{Toposort+Assertions}(\mathcal{T}))|S$. By the same argument, $\text{Worlds}(\text{Toposort}(\mathcal{T}))|S \subseteq \text{Worlds}(\text{Toposort+Assertions}(\mathcal{T}))|S$, so it follows that $\text{Worlds}(\text{Toposort}(\mathcal{T}))|S \subseteq \text{Worlds}(\text{Toposort+Assertions}(\mathcal{T}))|S$. $\diamondsuit$

Theorem 5-2 implies that unless all assertions are executed, a query may give less precise answers than is otherwise possible. In particular, it may report that a ground wff α is true in some alternative worlds and false in others when, if all assertions were executed, it would be known that in fact α had the same truth valuation in all remaining alternative worlds.

5.7. Update Splitting

To reduce the cost of the ancestors of a query or update that needs to be executed, the Lazy Algorithm makes use of a formalization of the *splitting techniques* illustrated in Section 5.1. There are two basic varieties of *splits*, or divisions of an update U into a sequence of updates: *horizontal splits*, in which disjuncts, conjuncts, or atoms of ω or ϕ are removed from U, generating a sequence of two updates to replace U; and *vertical splits*, in which U is split into multiple updates by conjoining a substitution or other wff ϕ to one version of U and $\neg\phi$ to the other. When an update is split, the resulting updates belong to the same family as did the original, and hence apply to the same cost bound as did the original. In addition, there are certain logical manipulations of ϕ and ω that can be useful, and they will be discussed also.

There are many ways to skin a cat, and many ways to split an update. Given an update or query U in the lazy graph to process, in the worst case the best

way to split the ancestors of U will not be at all obvious. In fact, in a deterministic version of the Update Algorithm, one can easily spend time exponential in the size of the lazy graph (assuming $\mathcal{P} \neq \mathcal{NP}$) just trying to decide how to split U's ancestors; the update split that initially looks most advantageous may turn out to cause an unacceptable increase in the costs of that update's ancestors. This plethora of possibilities does not lead to nice theorems telling when the Lazy Algorithm will accept U, or even to a nice algorithm for trying out all the possibilities. For that reason, we present a large repertoire of splits but only present a characterization of the performance of the Lazy Algorithm for a small subset of these splits.

5.7.1. A Repertoire of Splits

In a horizontal split, selected datoms are removed from ϕ or ω of an update U. Horizontal splits can be helpful when U is an ancestor of the incoming query Q, and some expensive part of U is not actually relevant to Q at all. For example, in `INSERT` $\alpha \vee g$ `WHERE` ϕ, if g is expensive and not needed for the execution of Q, it will be advantageous to split g off, because the estimated cost of `INSERT` α `WHERE` ϕ will doubtless be lower than that for U. It is possible to split between conjuncts of ω or disjuncts of ω or ϕ, and also to remove individual datoms from ω. These four types of splits will be covered in Splitting Rules 1 through 4, which map an update U into an *equivalent* sequence of updates:

Definition. If S_1 and S_2 are two sequences of updates over a language $\mathcal{L}$, then S_1 and S_2 are *equivalent* if for every extended relational theory $\mathcal{T}$ over $\mathcal{L}$, $\text{Worlds}(S_1(\mathcal{T})) = \text{Worlds}(S_2(\mathcal{T}))$. $\Diamond$

One obstacle to splitting an update U into U_1 and U_2 is that when U_2 is executed, U_2 must have some means of locating those alternative worlds where U is not yet completed. For example, if U is `INSERT` ω `WHERE` ϕ and U_1 is `INSERT` ω_1 `WHERE` ϕ, then in general U_2 cannot also rely on selection clause ϕ, because ω_1 may have changed the truth valuations for atoms in ϕ. A more promising candidate for U_2's selection clause is $(\phi)_{\sigma_{HU_1}}$, where σ_{HU_1} is the history substitution for U_1. However, there are two drawbacks to the use of $(\phi)_{\sigma_{HU_1}}$ in U_2. First, a future update with ancestor U may need to write some of the datoms in $(\phi)_{\sigma_{HU_1}}$, and there will be a read/write conflict between U_2 and that update, forcing sequential execution. Second, U may be split many times before it is fully executed. Every split–off update will incur costs associated with ϕ. Even if ω is very simple, the added expense of dealing with $(\phi)_{\sigma_{HU_1}}$, $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$, etc. may push the total cost for U beyond the cost limit, and force rejection of queries.

The solution to this difficulty is to make ϕ_{U_2} as short as possible. The technique for doing so has been presented once before, in the discussion of computational complexity of the Update Algorithm Version I in Chapter 3. There the goal was to minimize the amount of space required for formula (2), by defining a new history atom $H(U)$ with the wff $H(U))\leftrightarrow\phi$, and adding this wff to $\mathcal{T}$ just

after Step 1 of the Update Algorithm. This is adapted to the current case as follows.

Definition. If an update U is split into U_1 and U_2, then U_1 *defines* $H(U)$ if $H(U)$ is a new† history atom and during the execution of U_1, after Step 1 of the Update Algorithm, the formula $H(U) \leftrightarrow \phi$ is added to $\mathcal{T}$. $\Diamond$

Once defined, $H(U)$ can be used by subsequent updates; $H(U)$ is just a history atom that will be true in a model $\mathcal{M}$ iff ϕ was true in the precursor to that model just before U_1 was executed. $H(U)$ is an inexpensive way of marking the models where ϕ is true so that one can come back later and finish U easily. In many of these splitting rules, U_1 will define a history atom that is subsequently used by U_2. In the earlier example, U_2 can use $H(U)$ as its selection clause rather than $(\phi)_{\sigma_{HU_1}}$. By this means, history atoms can now appear in split–off updates.

Please note that if U defines a history atom and U is itself to be split into U_1 and U_2, then U_1 inherits the job of defining that history atom.

Splitting Rule 1. *Splits between conjuncts of* ω. If no datom or history atom in ω_1 unifies with an atom of ω_2, then the update U: `INSERT` $\omega_1 \wedge \omega_2$ `WHERE` ϕ is equivalent under the Update Algorithm to the sequence of updates

U_1: `INSERT` ω_1 `WHERE` ϕ,

U_2: `INSERT` ω_2 `WHERE` $H(U)$,

where U_1 defines $H(U)$. $\Diamond$

Remark 5-1. When U is split into U_1 and U_2, if ϕ contains no datoms, then the expense of defining $H(U)$ is unnecessary. It is preferable in this case not to define $H(U)$, but rather just use ϕ directly. This will be done in the examples of this chapter.

Example. `INSERT` Emp(Reid, CSD) $\wedge$ Emp(Reid, EE) `WHERE` T is equivalent to the sequence of updates

`INSERT` Emp(Reid, CSD) `WHERE` T,

`INSERT` Emp(Reid, EE) `WHERE` T. $\Diamond$

Proofs of correctness for these splitting rules are collected in Section 5.7.2.

Selection clauses are not the only places where extra history atoms are useful for marking models where updating is to be completed later. The case where ω is a disjunction, e.g., $R(a) \vee R(b)$, is a good illustration. If we want to insert just $R(a)$ for now and complete the disjunction later, then there must be some way of identifying the models where $R(b)$ should be inserted later. A new history atom is the best solution.

† A *new* history atom is one which does not unify with any history atom in $\mathcal{T}$ or in a pending update.

Splitting Rule 2. *Splits between disjuncts of ω.* If no datom in ω_1 unifies with an atom of ω_2, then the update U: INSERT $\omega_1 \vee \omega_2$ WHERE ϕ is equivalent under the Update Algorithm to the sequence of updates

U_1: INSERT $\omega_1 \vee H(U_1)$ WHERE ϕ,

U_2: INSERT $H(U_1) \leftrightarrow \omega_2$ WHERE $H(U)$,

where $H(U_1)$ is a new history atom, and U_1 defines $H(U)$. $\Diamond$

Please note that in Splitting Rule 2, U_1 does not *define* $H(U_1)$, but merely uses it.

Example. Assuming $H(1)$ is a new history atom, the update U: INSERT Emp(Reid, CSD) $\vee$ Emp(Reid, ϵ) WHERE T is equivalent to the sequence of updates

INSERT Emp(Reid, CSD) $\vee$ $H(1)$ WHERE T,

INSERT $H(1) \leftrightarrow$ Emp(Reid, ϵ) WHERE T. $\Diamond$

It is worth noting that Splitting Rule 2 would not work if the Update Algorithm treated history atoms as it does datoms. For if it did, then U_2 would change the truth valuation of $H(U_1)$, rather than using it to identify the models where the update is incomplete. The proofs of these Splitting Rules will show that nothing in the Update Algorithm or in its proof of correctness prevents the use of history atoms in certain situations within updates; the system may as well make internal use of history atoms whenever this is convenient and correct.

To see the necessity of the restriction in Splitting Rules 1 and 2 that datoms of ω_1 and ω_2 must not unify, consider the update INSERT Emp(Reid, CSD) $\vee$ Emp(Reid, CSD) WHERE T. This update is not equivalent to the sequence of updates

INSERT Emp(Reid, CSD) $\vee$ $H(1)$ WHERE T,

INSERT Emp(Reid, CSD) $\leftrightarrow$ $H(1)$ WHERE T,

because those two updates may create alternative worlds where Emp(Reid, CSD) is false. For example, if $\mathcal{T}$ has an empty body, then U_1 will produce a model $\mathcal{M}$ where Emp(Reid, CSD) is true and $H(1)$ is false, and U_2 will make Emp(Reid, CSD) false in $\mathcal{M}$. A similar problem occurs with the update INSERT Emp(Reid, CSD) $\wedge \neg$ Emp(Reid, CSD) WHERE T.

Using DeMorgan's laws and Splitting Rules 1 and 2, one can completely pick apart many ωs, using no more splits than there are conjunctions and disjunctions[†] in ω. Splitting Rules 1 and 2 only apply when ω takes a special form, however, and even when ω is in that form, at times it may be annoying to have to dissect ω just to get at one important datom. Splitting Rule 3 allows a one–step isolation of any set of datoms in ω; however, it may require the use of more

[†] Express any other binary operations in ω in terms of $\wedge$, $\vee$, and $\neg$.

history atoms than would be needed if Splitting Rules 1 and 2 were repeatedly applied.

The formula for ω_{U_2} in Splitting Rule 3 is rather intimidating. However, the intent is quite simple. If f is a datom of U to be removed from ω, then replace f by a history atom $H(f, \mathcal{F})$ in ω. Call this history substitution $\sigma_{\mathcal{F}}$. Then let U_1 insert $(\omega)_{\sigma_{\mathcal{F}}}$, and let U_2 insert $f \leftrightarrow H(f, \mathcal{F})$. The alarming second term of ω_{U_2} in Splitting Rule 3 is vacuously true except in the case where f unifies with an atom of $(\omega)_{\sigma_{\mathcal{F}}}$—the same case that caused restrictions in Splitting Rules 1 and 2. The second conjunct of ω_{U_2} in Splitting Rule 3 simply says that in models where f unifies with a datom of $(\omega)_{\sigma_{\mathcal{F}}}$, U_2 cannot change the truth valuation of f. Just how U_2 accomplishes that is a bit mysterious: $H(f, U_2)$ is an atom from U_2's own history substitution. Doug Hofstadter watch out!

Splitting Rule 3. *Removal of selected datoms from ω.* Let U be the update INSERT ω WHERE ϕ. Let $\mathcal{F}$ be a subset of the datoms that occur in ω. Let $\sigma_{\mathcal{F}}$ be a history substitution for the datoms in $\mathcal{F}$, composed of the replacement of every datom f in $\mathcal{F}$ by a history atom $H(f, \mathcal{F})$.[††] Then U is equivalent under the Update Algorithm to the sequence of updates

U_1 : INSERT $(\omega)_{\sigma_{\mathcal{F}}}$ WHERE ϕ,

U_2 : INSERT $\bigwedge_{f \in \mathcal{F}} \Big((f \leftrightarrow H(f, \mathcal{F})) \wedge$

$\qquad ((\bigvee_{\sigma \in \Sigma} \sigma) \quad \rightarrow (f \leftrightarrow H(f, U_2))) \Big)$ WHERE $H(U)$,

where U_1 defines $H(U)$, and Σ is the set containing the wff F and all substitutions σ such that for some datom g in $(\omega)_{\sigma_{\mathcal{F}}}$, f unifies with g under most general substitution σ. $\Diamond$

Intuitively, this type of split is useful when the datoms in $\mathcal{F}$ are too expensive or else need to be isolated from the other datoms of ω to facilitate vertical splitting of ω. U_1 leaves placeholders for those datoms in ω, in the form of history atoms. When the datoms of $\mathcal{F}$ become affordable later on, their truth valuations can be tied to those of the history atoms in $\sigma_{\mathcal{F}}$ through update U_2.

Example. Let U be the update INSERT $(\neg R(a) \vee R(b)) \wedge (R(c) \vee \neg R(b))$ WHERE T, and let Q be the query INSERT $Q(b)$ WHERE $R(b)$. Suppose that U is expensive, and the only conflict preventing execution of Q is the write/read dependency on $R(b)$. Then $R(b)$ can be split out of U in one step by creating the two updates

U_1: INSERT $(\neg H(R(a), 1) \vee R(b)) \wedge (H(R(c), 1) \vee \neg R(b))$ WHERE T,

U_2: INSERT $(R(a) \leftrightarrow H(R(a), 1) \wedge (R(c) \leftrightarrow H(R(c), 1))$ WHERE T. $\Diamond$

[††] By analogy to U in $H(f, U)$, $\mathcal{F}$ in $H(f, \mathcal{F})$ is simply a unique constant not previously used in any history atom, so that $H(f, \mathcal{F})$ does not unify with any preexisting history atom.

Example. Let U be the update INSERT $(\epsilon = a) \wedge R(\epsilon) \wedge \neg R(a)$ WHERE T. As ω is unsatisfiable, this update should eliminate all alternative worlds of any theory to which it is applied. Splitting U with Splitting Rule 3 produces

U_1: INSERT $(\epsilon = a) \wedge R(\epsilon) \wedge \neg H(R(a), U_1)$ WHERE T,

U_2: INSERT $(R(a) \leftrightarrow H(R(a), U_1)) \wedge ((\epsilon = a) \rightarrow (H(R(a), U_1) \leftrightarrow H(R(a), U_2))))$ WHERE T.

Without this final conjunct of ω in U_2, U_1 and U_2 applied to an extended relational theory with empty body would produce an alternative world in which $R(a)$ is false. The additional conjunct correctly eliminates all alternative worlds. ◊

Splitting Rule 4. *Splits between disjuncts of ϕ.* The update U: INSERT ω WHERE $\phi_1 \vee \phi_2$ is equivalent under the Update Algorithm to the sequence of updates

U_1: INSERT ω WHERE ϕ_1,

U_2: INSERT ω WHERE $(\phi_2)_{\sigma_{HU_1}}$,

where σ_{HU_1} is the history substitution for U_1. ◊

Example. The update INSERT Emp(Reid, CSD) WHERE Emp(Reid, EE) ∨ Mgr(Nilsson, ϵ) is equivalent to the sequence of updates

U_1: INSERT Emp(Reid, CSD) WHERE Emp(Reid, EE),

U_2: INSERT Emp(Reid, CSD) WHERE Mgr(Nilsson, ϵ). ◊

Though Splitting Rule 4 shows that it is possible to split between disjuncts of ϕ, in general it is not possible to split between conjuncts of ϕ, as all conjuncts of ϕ are needed to determine whether an alternative world is to be affected by the update.

We now turn to an examination of vertical splitting. In Splitting Rule 5 below, typically ϕ' will be a substitution σ, and there will be an update or query Q that depends on the results of U for some pair of datoms of Q and U that unify under substitution σ. It may be much cheaper to execute U only in those models where σ is true, rather than in all models where ϕ is true. This typically occurs if Skolem constants in the updates U and U_2 cause the unacceptable expense in U. For example, if U is INSERT Mgr(Nilsson, ϵ) WHERE Emp(Reid, ϵ), and Q is INSERT Q(Nilsson) WHERE Mgr(Nilsson, CSD), then Q has a write/read dependency on U. However, this dependency only materializes in models where ϵ = CSD. If U is split into U_1: INSERT Mgr(Nilsson, ϵ) WHERE Emp(Reid, ϵ) ∧ ϵ = CSD and U_2: INSERT Mgr(Nilsson, ϵ) WHERE Emp(Reid, ϵ) ∧ $\epsilon \neq$ CSD, then U_1 may well be affordable though U is not. U_2 can be executed later, as it will not be an ancestor of Q.

Splitting Rule 5. *Vertical Splits.* Let U be the update INSERT ω WHERE ϕ. If ϕ' is a ground wff, then U is equivalent under the Update Algorithm to the sequence of updates

U_1: INSERT ω WHERE $\phi \wedge \phi'$ and
U_2: INSERT ω WHERE $H(U) \wedge \neg(\phi')_{\sigma_{HU_1}}$,
where U_1 defines $H(U)$. $\Diamond$

Example. Let U be the update INSERT Emp(Reid, ϵ) WHERE T, and let U' be the update INSERT Emp(Reid, CSD) WHERE Mgr(Nilsson, CSD). There is a write/write dependency between U and U'; but this dependency only occurs for models where ϵ is bound to CSD. If U is too expensive, try splitting U into
U_1: INSERT Emp(Reid, ϵ) WHERE $\epsilon =$ CSD,
U_2: INSERT Emp(Reid, ϵ) WHERE $\epsilon \neq$ CSD.
Then U' does not depend on U_2, and U_1 may well be affordable. $\Diamond$

Sometimes two updates are guaranteed not depend on one another by virtue of the fact that they take place in disjoint sets of alternative worlds. For example, the updates
INSERT Emp(Reid, ϵ) WHERE $\epsilon =$ CSD and
INSERT $\neg$Emp(Reid, ϵ) WHERE $\epsilon \neq$ EE
will produce the same effect no matter which update is executed first. The NAP Algorithm takes advantage of any such opportunities created by vertical splitting, by eliminating dependencies of this sort between updates. However, it is not sufficient that the selection clauses of the two updates be mutually exclusive; for example, the effect of the two updates
INSERT Emp(Reid, CSD) WHERE $\neg$ Emp(Reid, CSD) and
INSERT $\neg$Emp(Reid, CSD) WHERE Emp(Reid, CSD)
depends upon the order in which they are executed.

Logical massage of ϕ and ω can be used to reduce the cost of Step 1 of the Update Algorithm, by removing datoms from U that are not subformulas of $\mathcal{T}$ or of pending ancestors of U. By applying a substitution σ to ϕ or ω, sometimes the resulting datoms in ϕ and ω already are subformulas of $\mathcal{T}$ even though the original datoms did not. Of course, this sword cuts both ways: applying σ may turn a datom that did occur in $\mathcal{T}$ into one requiring expenditures during Step 1.

Splitting Rule 6. *Logical massage.* The four updates
U_1: INSERT ω WHERE $\phi \wedge \sigma$,
U_2: INSERT $(\omega)_\sigma$ WHERE $\phi \wedge \sigma$,
U_3: INSERT ω WHERE $(\phi)_\sigma \wedge \sigma$,
U_4: INSERT $(\omega)_\sigma$ WHERE $(\phi)_\sigma \wedge \sigma$,
where σ is a ground substitution, are all equivalent. $\Diamond$

Of course the splits and rearrangements presented in the preceding splitting rules are not the only possible manipulations of updates. For example, U can be replaced by any other equivalent update; see Chapter 8 for rules on when two updates will be equivalent.

5.7.2. Correctness Proofs for Splits

Readers not interested in formal proofs of correctness for the splits of the previous section should proceed to the next section.

Proof of Splitting Rule 1. Let $\mathcal{M}$ be a model of extended relational theory $\mathcal{T}$ with Skolem constant substitution σ with respect to $\mathcal{T}$, U_1, and U_2. Let $\mathcal{M}_{U_1}$ be a model of $U_1(\mathcal{T})$, such that the alternative world of $\mathcal{M}_{U_1}$ is produced from that of $\mathcal{M}$ under the semantics for updates. Let $\mathcal{M}_{U_2}$ be a model of $U_2(U_1(\mathcal{T}))$, such that the alternative world of $\mathcal{M}_{U_2}$ is produced from that of $\mathcal{M}_{U_1}$ under the semantics for updates. Then ϕ is true in $\mathcal{M}$ iff $H(U)$ is true in $\mathcal{M}_{U_1}$, by the arguments of Theorem 4-1. If ϕ is true in $\mathcal{M}$, then $\omega_1 \wedge \omega_2$ is true in $\mathcal{M}_{U_2}$, because $(\omega_1)_\sigma$ and $(\omega_2)_\sigma$ have no datoms or history atoms in common. Therefore $\mathcal{M}_{U_2}$ is a model of an alternative world produced by U applied to $\mathcal{M}$.

If ϕ is false in $\mathcal{M}$, then $H(U)$ is false in $\mathcal{M}_{U_1}$, and $\mathcal{M}_{U_2}$ is a model of an alternative world produced by applying U to $\mathcal{M}$.

The reverse implication is symmetric. $\Diamond$

Proof of Splitting Rule 2. This proof follows the outline of the proof of Splitting Rule 3, with significant differences only in the forward and reverse proofs of correctness for Step 3. The revised forward and reverse proofs for Step 3 follow:

In the definition of $\mathcal{M}_U$ and $\mathcal{M}_{U_2}$, also define the truth valuation of $H(U_1)$: let $H(U_1)$ be true in $\mathcal{M}_{U_2}$ iff ω_2 is true in $\mathcal{M}_U$.

Consider those wffs added to $U_2(U_1(\mathcal{T}))$ during Step 3 of U_1 or U_2. By definition $\mathcal{M}_{U_2}$ satisfies $(\phi)_{\sigma HU} \rightarrow (\omega_1 \vee \omega_2)$. We must show that $\mathcal{M}_{U_2}$ satisfies $((\phi)_{\sigma HU_1})_{\sigma HU_2} \rightarrow ((\omega_1)_{\sigma HU_2} \vee H(U_1))$ and $((\phi)_{\sigma HU_1})_{\sigma HU_2} \rightarrow (H(U_1) \leftrightarrow \omega_2)$. The latter formula is true by definition of the truth valuation of $H(U_1)$ in $\mathcal{M}_{U_2}$. For the other formula, since no datom of ω_1 unifies with an atom of ω_2, it follows that $(\omega_1)_{\sigma HU_2}$ is identical to ω_1. But then by definition of $\mathcal{M}_{U_2}$, $((\phi)_{\sigma HU_1})_{\sigma HU_2} \rightarrow ((\omega_1)_{\sigma HU_2} \vee H(U_1))$ is satisfied in $\mathcal{M}_{U_2}$.

For the reverse implication, consider the formula added to $U(\mathcal{T})$ during Step 3 of U: $(\phi)_{\sigma HU} \rightarrow \omega$. We know $\mathcal{M}_U$ satisfies

$$((\phi)_{\sigma HU_1})_{\sigma HU_2} \rightarrow ((\omega_1)_{\sigma HU_2} \vee H(U_1))$$

and

$$((\phi)_{\sigma HU_1})_{\sigma HU_2} \rightarrow (\omega_2 \leftrightarrow H(U_1)).$$

Again, because no datom of ω_1 unifies with an atom of ω_2, it follows that $(\omega_1)_{\sigma HU_2}$ is identical to ω_1. Therefore the latter two formulas together logically imply that $((\phi)_{\sigma HU_1})_{\sigma HU_2} \rightarrow (\omega_1 \vee \omega_2)$ is true in $\mathcal{M}_U$. Then by the definition of $\mathcal{M}_U$, $(\phi)_{\sigma HU} \rightarrow (\omega_1 \vee \omega_2)$ is true in $\mathcal{M}_U$. $\Diamond$

Proof of Splitting Rule 3. Let σ_{HU}, σ_{HU_1}, and σ_{HU_2} be the history substitutions for U, U_1, and U_2, respectively. First we show that any model produced by U is the model of an alternative world also produced by U_1 followed by U_2.

Let $\mathcal{M}_U$ be a model of $U(\mathcal{T})$. Let $\mathcal{M}_{U_2}$ be a model identical to $\mathcal{M}_U$ except that for every null–free datom f, the history atoms $H(f,U_1)$, $H(f,U_2)$, and $H(f,\mathcal{F})$ are given the following truth valuations in $\mathcal{M}_{U_2}$:†

$H(f,U_1)$ gets the same valuation as $H(f,U)$

$H(f,\mathcal{F})$ gets the same valuation as f

$H(f,U_2)$ gets the same valuation as f, if $\exists g|\{g \in (\omega)_{\sigma_{\mathcal{F}}}$ and $f \sim_{\sigma'} g\}$

$H(f,U_2)$ gets the same valuation as $H(f,U)$, otherwise.

In addition, let $H(U)$ be true in $\mathcal{M}_{U_2}$ iff ϕ is true in $\mathcal{M}$. Note that $H(U)$ is true in $\mathcal{M}_{U_2}$ iff $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$ is true in $\mathcal{M}_{U_2}$; this correspondence will be used throughout the formulas in this proof without special notice, by replacing occurrences of $H(U)$ by an equivalent expression over ϕ.

Clearly $\mathcal{M}_U$ and $\mathcal{M}_{U_2}$ represent the same alternative world. To show that $\mathcal{M}_{U_2}$ is a model of Worlds$(U_2(U_1(\mathcal{M})))$, we will consider all the possible reasons that a particular wff might be in $U_2(U_1(\mathcal{T}))$, and show that in each case, $\mathcal{M}$ satisfies that wff.

First suppose α is a wff of the body of $\mathcal{T}$. Then under the Update Algorithm, $U_2(U_1(\mathcal{T}))$ contains the wff $((\alpha)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$. $\mathcal{M}_{U_2}$ satisfies $(\alpha)_{\sigma_{HU}}$, and by definition therefore also satisfies $((\alpha)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$.

Now consider the wffs added to $\mathcal{T}$ during Step 1 of update U_1. If f is a datom of $(\omega)_{\sigma_{\mathcal{F}}}$, then $U_2(U_1(\mathcal{T}))$ contains the wff

$$((f \rightarrow \bigvee_{\substack{g \in \mathcal{T} \\ f \sim_\sigma g}} \sigma)_{\sigma_{HU_1}})_{\sigma_{HU_2}}. \qquad 5\text{-}1$$

Since $U(\mathcal{T})$ contains the wff

$$(f \rightarrow \bigvee_{\substack{g \in \mathcal{T} \\ f \sim_\sigma g}} \sigma)_{\sigma_{HU}}, \qquad 5\text{-}2$$

it follows by definition that formula 5-1 is satisfied by $\mathcal{M}_{U_2}$. If f is a datom in $\mathcal{F}$, then $U_2(U_1(\mathcal{T}))$ also contains the wff

$$(f \rightarrow (\bigvee_{\substack{g \in \mathcal{T} \\ f \sim_\sigma g}} \sigma \vee \bigvee_{\substack{g \notin \mathcal{T}, g \in (\omega)_{\sigma_{\mathcal{F}}} \\ f \sim_\sigma g}} \sigma))_{\sigma_{HU_2}}. \qquad 5\text{-}3$$

† For α a wff, theory, or substitution, and g an atom, the notation $g \in \alpha$ means "g is a subformula of α". If α is a hyperedge or set of nodes in a graph, then the notation $g \in \alpha$ means that the node g is on the hyperedge or in the set of nodes α. The notation $f \sim_\sigma g$ means that f unifies with g under most general substitution σ.

Since formula 5-2 implies formula 5-3 under the definition of $\mathcal{M}_{U_2}$, it follows that formula 5-3 is satisfied by $\mathcal{M}_{U_2}$.

Now consider those wffs added to $U_2(U_1(\mathcal{T}))$ during Step 3 of U_1 or U_2. We first show that $(\phi)_{\sigma_{HU}}$ is true in $\mathcal{M}_{U_2}$ iff $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$ is true in $\mathcal{M}_{U_2}$.

If $(\phi)_{\sigma_{HU}}$ is true in $\mathcal{M}_{U_2}$, then $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU}}$ must also be true there, by definition of the truth valuations of $H(f,U_1)$. Conversely, if a datom f of ϕ is a subformula of σ_{HU_2} but not of σ_{HU_1}, then it must be the case that f unifies with a datom of $\mathcal{F}$ and does not unify with any datom of $(\omega)_{\sigma_{\mathcal{F}}}$. But then by definition the truth valuation of $H(f,U)$ is the same as that of $H(f,U_2)$. It follows that $(\phi)_{\sigma_{HU}}$ is true in $\mathcal{M}_{U_2}$ iff $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$ is also true.

By definition $\mathcal{M}_{U_2}$ satisfies $(\phi)_{\sigma_{HU}} \rightarrow \omega$. We must show that $\mathcal{M}_{U_2}$ satisfies the wff $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}} \rightarrow ((\omega)_{\sigma_{\mathcal{F}}})_{\sigma_{HU_2}}$, introduced during Step 3 of U_1. If $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$ is true in $\mathcal{M}_{U_2}$, then $(\phi)_{\sigma_{HU}}$ is also true, and therefore ω is true in $\mathcal{M}_{U_2}$. By definition of $\mathcal{M}_{U_2}$, it follows that $((\omega)_{\sigma_{\mathcal{F}}})_{\sigma_{HU_2}}$ is true in $\mathcal{M}_{U_2}$. We must also show that

$$((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}} \rightarrow \bigwedge_{f \in \mathcal{F}} ((f \leftrightarrow H(f,\mathcal{F})) \wedge ((\bigvee_{\sigma \in \Sigma} \sigma) \rightarrow (f \leftrightarrow H(f,U_2))))$$

is satisfied in $\mathcal{M}_{U_2}$. But both conjuncts of this formula are true, by definition of Σ. Therefore $\mathcal{M}_{U_2}$ satisfies the wffs added during Step 3.

Now consider those wffs added to $U_2(U_1(\mathcal{T}))$ during Step 4 of U_1 or U_2. $\mathcal{M}_{U_2}$ satisfies the wff of $U(\mathcal{T})$

$$(f \leftrightarrow H(f,U)) \vee ((\phi)_{\sigma_{HU}} \wedge \bigvee_{\substack{\vartheta \in \omega \\ f \sim \sigma\vartheta}} \sigma), \qquad 5\text{-}4$$

for each datom f in $U(\mathcal{T})$ that unifies with a datom of $(\omega)_{\sigma_{\mathcal{F}}}$ or $\mathcal{F}$.

If f in $U_2(U_1(\mathcal{T}))$ unifies with a datom of $(\omega)_{\sigma_{\mathcal{F}}}$, then $U_2(U_1(\mathcal{T}))$ contains the wff

$$((f)_{\sigma_{HU_2}} \leftrightarrow H(f,U_1)) \vee (((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}} \wedge \bigvee_{\substack{\vartheta \in (\omega)_{\sigma_{\mathcal{F}}} \\ f \sim \sigma\vartheta}} \sigma),$$

which is true by definition of $H(f,U_2)$ and $H(f,U_1)$, if f unifies with a datom of $\mathcal{F}$; and true by formula 5-4, otherwise.

If f in $U_2(U_1(\mathcal{T}))$ unifies with a datom of $\mathcal{F}$, then $U_2(U_1(\mathcal{T}))$ contains the wff

$$(f \leftrightarrow H(f,U_2)) \vee (((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}} \wedge \bigvee_{\substack{f \sim \sigma\vartheta \\ \vartheta \in \mathcal{F}}} \sigma).$$

Since $(\phi)_{\sigma_{HU}}$ is true iff $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$ is true, by the definition of $\mathcal{M}_{U_2}$ and formula 5-4 this formula is also satisfied by $\mathcal{M}_{U_2}$. This establishes that $\mathcal{M}_{U_2}$ satisfies the wffs added during Step 4 of U_1 and U_2, and that $\mathcal{M}_{U_2}$ is a model of an alternative world of $U_2(U_1(\mathcal{T}))$.

To show that models produced by U_1 and U_2 represent alternative worlds produced by U, suppose $\mathcal{M}_{U_2}$ is a model of $U_2(U_1(\mathcal{T}))$. Let $\mathcal{M}_U$ be a model differing from $\mathcal{M}_{U_2}$ only in the following: $H(f,U)$ has the truth valuation in $\mathcal{M}_U$ of $H(f,U_1)$ in $\mathcal{M}_{U_2}$ if f unifies with a datom of σ_{HU_1}, and of $H(f,U_2)$ otherwise. Then $\mathcal{M}_{U_2}$ and $\mathcal{M}_U$ represent the same alternative world, and again we must show that $\mathcal{M}_U$ satisfies all the wffs of $U(\mathcal{T})$.

Let α be a wff of $\mathcal{T}$. Then $(\alpha)_{\sigma_{HU}}$ is a subformula of $U(\mathcal{T})$. $\mathcal{M}_U$ satisfies $((\alpha)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$, and therefore $\mathcal{M}_U$ satisfies $(\alpha)_{\sigma_{HU}}$.

Now consider wffs added to $\mathcal{T}$ in Step 1 of U: for each datom f that is a subformula of ω but not of $\mathcal{T}$, $U(\mathcal{T})$ contains the wff

$$(f)_{\sigma_{HU}} \rightarrow \bigvee_{\substack{g \in \mathcal{T} \\ f \sim_\sigma g}} \sigma. \qquad 5\text{-}5$$

For f in $(\omega)_{\sigma_{\mathcal{F}}}$, $\mathcal{M}_U$ satisfies the wff of $U_2(U_1(\mathcal{T}))$

$$(f)_{\sigma_{HU_1}} \rightarrow \bigvee_{\substack{g \in \mathcal{T} \\ g \sim_\sigma f}} \sigma,$$

which implies that $\mathcal{M}_U$ satisfies formula 5-5 as well.

For f in $\sigma_{\mathcal{F}}$, $\mathcal{M}_U$ satisfies the wff of $U_2(U_1(\mathcal{T}))$

$$(f)_{\sigma_{HU_2}} \rightarrow \Big(\bigvee_{\substack{g \in \mathcal{T} \\ g \sim_\sigma f}} \sigma \vee \bigvee_{\substack{g \notin \mathcal{T} \\ g \in (\omega)_{\sigma_{\mathcal{F}}} \\ g \sim_\sigma f}} \sigma \Big). \qquad 5\text{-}6$$

In formula 5-6, suppose $(f)_{\sigma_{HU_2}}$ is true in $\mathcal{M}_U$. If the left–hand disjunct of 5-6 is true, then formula 5-5 is satisfied. Otherwise, for some g from the right–hand disjunct, g is true in $\mathcal{M}_U$; therefore $\mathcal{M}_U$ must satisfy an instantiation of formula 5-5 for g. It follows that a left–hand disjunct of 5-6 must be true in $\mathcal{M}_U$, and therefore 5-5 is satisfied by $\mathcal{M}_U$.

The formulas added to $U(\mathcal{T})$ during Step 4 of U take the form, for f unifying with a datom of ω,

$$(f \leftrightarrow H(f,U)) \vee ((\phi)_{\sigma_{HU}} \wedge \bigvee_{\substack{g \in \omega \\ g \sim_\sigma f}} \sigma). \qquad 5\text{-}7$$

If f unifies with a datom of $(\omega)_{\sigma_{\mathcal{F}}}$, and f is a subformula of $\mathcal{T}$, ϕ, or $(\omega)_{\sigma_{\mathcal{F}}}$, then $\mathcal{M}_U$ satisfies the formula of $U_2(U_1(\mathcal{T}))$

$$((f)_{\sigma_{HU_2}} \leftrightarrow H(f,U_1)) \vee (((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}} \wedge \bigvee_{\substack{g \in (\omega)_{\sigma_{\mathcal{F}}} \\ g \sim_\sigma f}} \sigma).$$

For f any datom of $U(\mathcal{T})$ unifying with a datom of $\sigma_{\mathcal{F}}$, $\mathcal{M}_U$ also satisfies

$$(f \leftrightarrow H(f, U_2)) \vee ((((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}} \wedge \bigvee_{\substack{g \in \mathcal{F} \\ g \sim_\sigma f}} \sigma). \qquad 5\text{-}8$$

If f does not unify with datoms of both $(\omega)_{\sigma_{\mathcal{F}}}$ and $\sigma_{\mathcal{F}}$, then formula 5-7 is satisfied. Otherwise, if the left–hand disjunct of formula 5-8 is true in $\mathcal{M}_{U_2}$, then by definition of $\mathcal{M}_U$, formula 5-7 is satisfied in $\mathcal{M}_U$. If the left–hand disjunct is false, then the right–hand one must be true; since the occurrence of a datom g in $\mathcal{F}$ implies that g also is a subformula of ω, it follows that 5-7 is again satisfied for $\mathcal{M}_U$.

Now consider the formula added to $U(\mathcal{T})$ during Step 3 of U: $(\phi)_{\sigma_{HU}} \rightarrow \omega$. By the same argument used in the forward direction of this proof, $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$ is true in $\mathcal{M}_{U_2}$ iff $(\phi)_{\sigma_{HU}}$ is true in $\mathcal{M}_{U_2}$. It remains to show that ω is true when $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$ is true in $\mathcal{M}_{U_2}$.

When $((\phi)_{\sigma_{HU_1}})_{\sigma_{HU_2}}$ is true in $\mathcal{M}_{U_2}$, $\mathcal{M}_{U_2}$ must satisfy $((\omega)_{\sigma_{\mathcal{F}}})_{\sigma_{HU_2}}$,

$$\bigwedge_{f \in \mathcal{F}} (f \leftrightarrow H(f, \mathcal{F})),$$

$$\bigvee_{\substack{g \in (\omega)_{\sigma_{\mathcal{F}}} \\ f \sim_\sigma g \\ \text{or } \sigma = F}} \sigma \rightarrow (H(f, \mathcal{F}) \leftrightarrow H(f, U_2)),$$

and formula 5-8. But these together imply that $\mathcal{M}_{U_2}$ satisfies $(\omega)_{\sigma_{\mathcal{F}}}$. By definition of $\mathcal{M}_{U_2}$, if $(\omega)_{\sigma_{\mathcal{F}}}$ is true in $\mathcal{M}_{U_2}$, then ω must be true in $\mathcal{M}_{U_2}$. Therefore $\mathcal{M}_{U_2}$ satisfies the formulas added during Step 3 of U.

As $\mathcal{M}_U$ satisfies all the wffs added to $\mathcal{T}$ during the Update Algorithm for U, we conclude that $\text{Worlds}(U_2(U_1(\mathcal{T}))) = \text{Worlds}(U(\mathcal{T}))$. $\Diamond$

Proof of Splitting Rule 4. Let $\mathcal{M}$ be a model of $\mathcal{T}$, and let $\mathcal{M}_{U_1}$ be a model whose alternative world is derived from $\mathcal{M}$ by U_1 under the semantics for updates. First, by the arguments of Theorem 4-1, $(\phi_2)_{\sigma_{HU_1}}$ is true in $\mathcal{M}_{U_1}$ iff ϕ_2 was true in $\mathcal{M}$. It follows that Splitting Rule 4 is true for all models $\mathcal{M}$ of $\mathcal{T}$ where $\neg\phi_1 \wedge \neg\phi_2$, $\phi_1 \wedge \neg\phi_2$, or $\neg\phi_1 \wedge \phi_2$ is true. If $\phi_1 \wedge \phi_2$ is true in $\mathcal{M}$, then ω will be inserted into $\mathcal{M}$ twice. But insertion of a wff is idempotent; for any update U, $\text{Worlds}(U(U(\mathcal{M}))) = \text{Worlds}(U(\mathcal{T}))$. It follows that U is equivalent to the sequence of updates U_1 and U_2. $\Diamond$

Proof of Splitting Rule 5. Let $\mathcal{M}$ be a model of $\mathcal{T}$, and let $\mathcal{M}_{U_1}$ be a model whose alternative world is derived from $\mathcal{M}$ by U_1 under the semantics for updates. By the proof of Theorem 4-1, $H(U)$ will be true in $\mathcal{M}_{U_1}$ iff ϕ was true in $\mathcal{M}$ before U_1 began. Therefore ϕ and ϕ' are true in $\mathcal{M}$ iff $(\phi)_{\sigma_{HU_1}}$ and $(\phi')_{\sigma_{HU_1}}$, respectively, are true in $\mathcal{M}_{U_1}$. Reusing the proof of Splitting Rule 4, it follows that $\text{Worlds}(U_2(U_1(\mathcal{T}))) = \text{Worlds}(U(\mathcal{T}))$. $\Diamond$

Proof of Splitting Rule 6. We will show that U_1 and U_4 are equivalent, and the proofs for the rest follow. Let $\mathcal{M}$ be a model of $\mathcal{T}$, and let σ_1 be the Skolem constant substitution for $\mathcal{M}$ with respect to $\mathcal{T}$, ϕ, ω, and σ. Let $\mathcal{M}_{U_1}$ be a model produced from $\mathcal{M}$ by the Update Algorithm for update U_1. Then $\phi \wedge \sigma$ is true in $\mathcal{M}$ iff $(\phi)_\sigma \wedge \sigma$ is true in $\mathcal{M}$. If $\phi \wedge \sigma$ is false in $\mathcal{M}$, the theorem follows. If $\phi \wedge \sigma$ is true in $\mathcal{M}$, then σ_1 logically entails σ, so if ω is true in $\mathcal{M}_{U_1}$ then $(\omega)_\sigma$ is also. This implies that $\mathcal{M}_{U_1}$ is also a model of an alternative world produced by U_4.

For the other direction, let $\mathcal{M}$ be as before, and let $\mathcal{M}_{U_4}$ be a model produced from $\mathcal{M}$ by U_4. Suppose that $(\phi)_\sigma \wedge \sigma$ is true in $\mathcal{M}$, as otherwise the theorem follows. Then $(\omega)_\sigma$ is true in $\mathcal{M}_{U_4}$. Since σ_1 logically entails σ, $((\omega)_\sigma)_{\sigma_1}$ is identical to $(\omega)_{\sigma_1}$, so $(\omega)_{\sigma_1}$ is true in $\mathcal{M}_{U_4}$. It follows that $\mathcal{M}_{U_4}$ is a model of an alternative world produced by U_4. $\Diamond$

5.7.3. The Splitting Algorithm

The Splitting Algorithm shows how to split update hyperedges in the lazy graph. Suppose an update U is to be split into the sequence of updates $U_1 \cdots U_n$. Intuitively, the job of the Splitting Algorithm is to move back in time to the moment when U was added to the lazy graph, and instead of adding U, successively add U_1 through U_n. Then all the updates that arrived after U can be added back into the lazy graph. As the proof of correctness for the Splitting Algorithm will illustrate, this can be done quite efficiently as long as in all vertical splits (Splitting Rule 5), ϕ' contains no history atoms or datoms.

The Splitting Algorithm.

Input: A lazy graph G containing node U, and the sequence of updates U_1 and U_2, produced by splitting U in accordance with Splitting Rules 1–4; or produced by Splitting Rule 5, if ϕ' contains only equality atoms; or a single update U_1, produced in accordance with Splitting Rule 6.

Output. An equivalent lazy graph G' in which U has been replaced by the new updates U_1 and/or U_2, as appropriate.

Procedure. A sequence of three steps:

Step 1. Add new nodes. Set G' to be G. Remove the nodes of U, the update hyperedge of U, and all arcs incident to U from G'. Let G_U be the subgraph of G' containing only those nodes that are ancestors of nodes in U. Apply the NAP algorithm to add update hyperedge U_1 to the family hyperedge of U in the subgraph G_U. Repeat for U_2, if U_2 exists.

Step 2. Check arcs to children of U. If there is an arc in G from a node of U to a node g not in U, apply Step 2 of the NAP Algorithm and create, if Step 2 so requires, an arc from a node of U_1 or U_2 to g in G'.

Step 3. Reestimate costs for children of U. If there was an arc in G from a node of U to a node g not in U, then the cost information for g may change

in G'. In particular, the number of unifications for g may may decrease if some datom f of U that unified with g no longer is a subformula of $\mathcal{T}$ or in any pending ancestor of g. Adjust the unification counts to reflect these changes. ◊

Example. Given the lazy graph of figure 5-4, if update U_1 is split horizontally according to Splitting Rule 2, the Splitting Algorithm produces

U_3: **INSERT** Emp(Reid, ϵ_1)∨H(1) **WHERE** T and

U_4: **INSERT** Mgr(Nilsson, ϵ_2) ↔ $H(1)$ **WHERE** T,

shown in the lazy graph produced by the Splitting Algorithm in figure 5-6. ◊

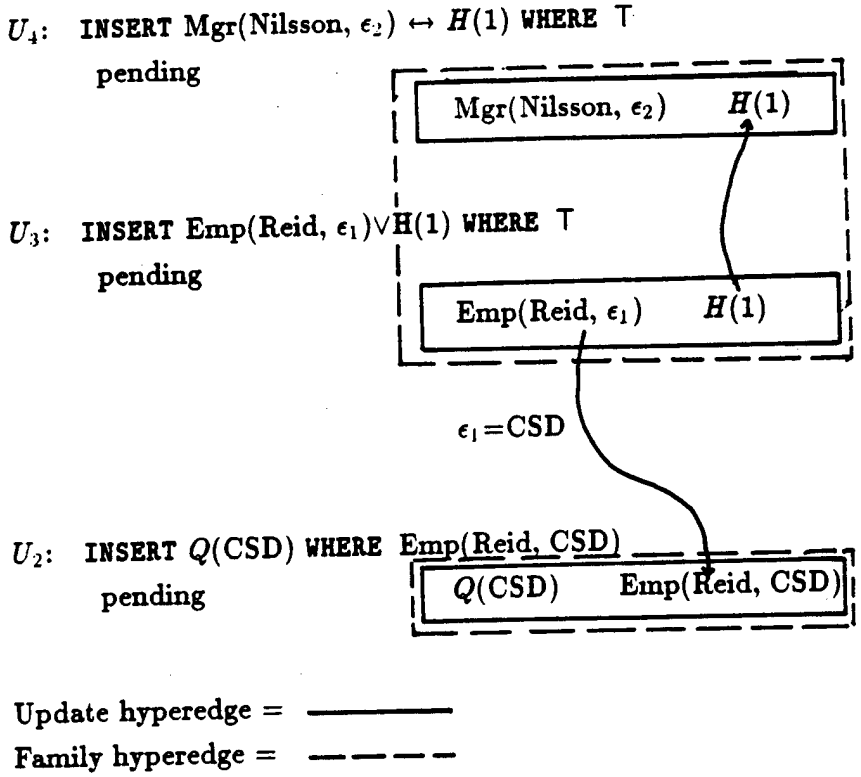


Figure 5-6. Splitting Algorithm example.

The part of the Splitting Algorithm in serious need of formal justification is its assumption that U_1 and U_2 have no descendants or ancestors other than those nodes that were ancestors or descendants of U. Theorem 5-3 shows that this assumption is in fact warranted. According to Theorem 5-3, close examination of any seemingly missing arcs of G' will show that the unifications under which those conflicts would occur will never materialize.

Theorem 5-3. Let S be a sequence of updates and queries containing update or query U. Let G be a lazy graph, created with the NAP and Splitting Algorithms, with topological sort S. Let G' be the lazy graph produced from G by the Splitting Algorithm when U is split into U_1 and U_2 (just U_1 if splitting in accordance with Splitting Rule 6). Let S' be the sequence created by replacing U in S by U_1 U_2. Use the NAP Algorithm to insert sequentially the updates and queries of S' into an initially empty lazy graph G''.

If S_1 and S_2 are reverse topological sorts of G' and G'', respectively, then S_1 and S_2 are equivalent. $\Diamond$

The proof of this theorem will show that there may be arcs that the NAP Algorithm would include but the Splitting Algorithm does not; but for any such arc, external factors will prevent the conflict predicted by the arc from materializing. First a bit of terminology: If U is an update or query, let ϕ_U be the set of nodes on the lazy graph hyperedge for U whose labels are subformulas of ϕ of U; and let ω_U be defined analogously.

Proof of Theorem 5-3. Assume inductively that all splits previously performed in G have this property. Then if an arc labelled σ appears in G'' and not in G' (a *new* arc), its presence cannot lead to a violation of Theorem 5-3 unless one endpoint of the arc is in U_1 or U_2 and the other is outside U_1 and U_2. Suppose first that the endpoint is in U_1.

Looking at the formula that defines U_1 for the splits of Splitting Rule 1, there are no atoms in U_1 that were not also subformulas of U; and the selection clause ϕ is the same as it was in U. Therefore, by Step 2 of the NAP Algorithm, no new arc could possibly have an endpoint in U_1.

Looking at the formula that defines U_1 for the splits of Splitting Rule 2, there is only one datom or history atom, $H(U_1)$, that was not also a subformula of U; and the selection clause ϕ is the same as it was in U. Therefore, by Step 2 of the NAP Algorithm, any new arc must have $H(U_1)$ as an endpoint. But by definition $H(U_1)$ is not a subformula of any other update except U_2, or unify with any atom in any update except U_2. Therefore there can be no new arc with an endpoint in U_1.

For Splitting Rule 3, the same argument holds as for Splitting Rule 2.

For Splitting Rule 4, again there are no new atoms in U_1; however, ϕ has changed, so perhaps some unification that failed test (2) or (3) of Step 2 of the NAP Algorithm will now succeed. However, for test (2), if $\sigma \wedge (\phi_1 \vee \phi_2)$ is unsatisfiable, then $\sigma \wedge \phi_1$ must be unsatisfiable as well. For test (3), if $\phi_1 \vee \phi_2$ logically entailed α, then so does ϕ_1. Therefore there can be no new arcs with an endpoint in U_1.

For Splitting Rule 5, by assumption ϕ' contains no non-equality atoms. Therefore there are no new datoms or history atoms in U_1, and by the argument for Splitting Rule 4, there can be no new arcs with an endpoint in U_1.

For Splitting Rule 6, there may indeed be new datoms in ω or ϕ, created by applying σ to previously existing atoms. However, if σ_1 is the label of the new arc, by Step 2 of the NAP Algorithm, it must be the case that $\sigma_1 \wedge \phi \wedge \sigma$ is satisfiable, and so that arc should have been in G all along. We conclude that there can be no new arcs with an endpoint in U_1, for any type of split.

Now consider new arcs with an endpoint in U_2 and, say, U' as the other endpoint. For Splitting Rule 1, the only new atom in U_2 is $H(U)$. But by the argument used above for U_1 of Splitting Rule 1, no new arc can have $H(U)$ as an endpoint. Therefore if there is a new arc between U_2 and U', it must be because that arc formerly failed test (2) or (3) of the NAP Algorithm Step 2, and now passes the test.

If test (3) was failed, suppose first that U_2 lies at the head of the new arc, and update or query U' lies at the tail of the arc. Let $\mathcal{T}$ be an extended relational theory with model $\mathcal{M}$. Then by the definition of test (3), whenever ϕ_{U_2} is true in a model $\mathcal{M}$ of $\mathcal{T}$, it must be the case that α, defined in test (3), is also true in $\mathcal{M}$. As α contains only equality atoms, this property still holds for the descendants of $\mathcal{M}$ after any sequence of updates is applied to $\mathcal{M}$. But this means that when U' is executed, its selection clause must be false in all the descendants of $\mathcal{M}$. This means that the conflict predicted by the new arc can never materialize, as the result of applying U and U' to $\mathcal{M}$ is independent of the order in which they are applied. The proof is symmetric if U' lies at the head of the new arc and U_2 at the tail.

If test (2) was failed, $\sigma \wedge \phi$ was unsatisfiable. By the arguments of Theorem 4-1, $(\sigma \wedge \phi)_{\sigma_{HU_1}}$ will also be unsatisfiable; by the same arguments, after any sequence of history substitutions, this property still holds. Therefore if $H(U)$ is true in a model, it must be the case that σ is false in that model, and therefore the predicted conflict does not actually occur because the unification needed for the dependency does not take place.

Consider the split of Splitting Rule 2. The same argument applies to $H(U)$ as for Splitting Rule 1. As $H(U_1)$ is not an implicit or explicit subformula outside of U_1 and U_2, the theorem follows for that type of split.

The case of Splitting Rule 3 is identical to that of Splitting Rule 2.

For Splitting Rule 4, the arguments used for Splitting Rule 1 eliminate the possibility that any new arc from outside could have a history atom of $(\phi_2)_{\sigma_{HU_1}}$ as an endpoint. Therefore if there is a new arc with U_2 as an endpoint, it must be because that arc formerly failed test (2) or test (3) of the NAP Algorithm Step 2, and now passes these tests. If test (2) was failed, $\sigma \wedge (\phi_1 \vee \phi_2)$ was unsatisfiable, which implies that $\sigma \wedge \phi_2$ was unsatisfiable. By the arguments of Theorem 4-1, $(\sigma \wedge \phi_2)_{\sigma_{HU_1}}$ will also be unsatisfiable. And if test (3) was failed with selection clause $\phi_1 \vee \phi_2$ for U, then test (3) must still be failed when the selection clause is changed to ϕ_2. It follows that there can be no new arcs with an endpoint in U_2.

For Splitting Rule 5, ϕ' contains no non-equality atoms, so there are no new non-equality atoms in U_2 other than $H(U)$. By the argument used for

Splitting Rule 1, no new arc can have $H(U)$ as an endpoint. The only remaining possibility is that some substitution now passes tests (2) and (3) of Step 2 of the NAP Algorithm, but the argument used for Splitting Rule 1 also rules that out. We conclude that Theorem 5-3 is true. $\Diamond$

5.8. Assertions

To drive the Lazy Algorithm, we need a policy on when updates should be processed and executed. At the very least, queries should force the execution of as many updates as are necessary to give a correct answer to the query. But update processing and execution cannot be entirely query-driven: early execution or at least special handling is required for assertions that are entered in response to a query rejection. For example, if the user is told that a query about an employee cannot be executed because of the datom Emp(ϵ, CSD), the user might assert the value of ϵ and then reenter the query. The cost estimation function must take note of this new assertion about ϵ and reduce the cost estimates of pending updates in which ϵ occurs. Furthermore, the new information about ϵ can be used to reduce the size of the extended relational theory, in effect retroactively reducing the cost of all earlier updates that contained ϵ! Since the earlier updates have become more affordable than they originally were, their estimated costs should be decreased in accordance with the savings realized in $\mathcal{T}$. We omit the algorithm for this aspect of lazy evaluation.

By letting update execution be entirely query–driven, we would miss some other opportunities to reduce the size of the extended relational theory and to reduce the cost estimates of other updates. For example, it's a good idea to execute helpful assertions (ones that narrow down the range of possible values for a Skolem constant) right away. If helpful updates are being blocked from execution by expensive ancestors or by the presence of expensive datoms in the same update, it may be worthwhile to use the Lazy Algorithm to force execution of the helpful part of the update, rather than to keep it waiting in the wings until query processing begins.

Another argument for early execution of assertions is that the user interface routines will probably force processing and execution of as many pending assertions as possible before presenting the user with the answer to a query, even though not all assertions need be executed before the query is executed. This is necessary if the most exact answer to a query is to be given, because any assertion can eliminate an alternative world that was important to the query, and in the process eliminate some candidate answer to the query.

5.9. The Costs and Benefits of the Lazy Algorithm

As mentioned earlier, we have no nice worst–case theorems telling when a query or update U will be processable. This is due to the difficulty of splitting the selection clause ϕ of an update; if ϕ were as easy to split as ω is, then an excellent characterization would be possible of the benefits of the Lazy Algorithm. For this

reason, we characterize the behavior of the Lazy Algorithm for a certain class of selection clauses ϕ. This characterization depends on three assumptions:

Assumption 1. The *non-interacting ϕ requirement:* Let Q be the incoming query to be processed by the Lazy Algorithm. Then no datom that is a subformula of ϕ of any proper ancestor of Q in the lazy graph can also be a subformula of in ω of another ancestor of Q. $\Diamond$

Assumption 1 says that any datom read by a parent or higher ancestor of Q cannot also be written by an ancestor of Q.

Assumption 2. The cost estimation function must satisfy the following equation for any hyperedge or set of nodes U in the lazy graph:

$$\text{EstCost}(U) = \sum_{f \in U} \begin{cases} 0, & f \text{ an equality atom;} \\ 1, & f \text{ a history atom;} \\ \text{EstCost}(f), & f \text{ a datom.} \end{cases}$$

For a particular datom node f, EstCost(f) must depend only on the cost information stored at that node in the lazy graph. $\Diamond$

The purpose of Assumption 2 is to ensure that the cost estimation function does not depend on hard–to–handle factors such as the number of update hyperedges in the lazy graph or the distance of an update from a root in the lazy graph. This is needed in order to establish a simple relationship between the estimated cost of the nodes of an update before and after the update is split.

Assumption 3. *Judicious use of history atoms:* Let Q be an incoming query. At the time Q is added to the lazy graph, every history atom arc in the lazy graph must have one endpoint in an executed update. $\Diamond$

History arcs should rarely prevent execution of an otherwise affordable update. The purpose of a history arc is to make sure that a history atom is defined before it is used; a split of U into U_1 and U_2 should always be performed so that if U_1 defines the history atom and U_1 is an ancestor of the query or update that is to be executed, then U_1 is not an ancestor solely because of history atom arcs. For example, consider the update

U: `INSERT` Emp(Reid, ϵ)$\wedge$Mgr(Nilsson, CSD) `WHERE` Mgr(Kennedy, EE)

and the incoming query

Q: `INSERT` Q(Nilsson) `WHERE` Mgr(Nilsson, CSD).

Suppose that Emp(Reid, ϵ) makes U too expensive, and so U is split according to Splitting Rule 1 into

U_1: INSERT Emp(Reid, ϵ) WHERE Mgr(Kennedy, EE)

and

U_2: Mgr(Nilsson, CSD) WHERE $H(U)$.

This was a most foolish choice of splits, because Q still depends on U_1 through a history atom definition arc for $H(U)$. Either Mgr(Kennedy, EE) should have been used as the selection clause of U_2, or else Emp(Reid, ϵ) should have been split out of U into U_2 rather than into U_1. Any reasonable choice of splitting heuristics should satisfy assumption 3.

Theorem 5-4 below gives a simple sufficient condition for queries to be accepted by the Lazy Algorithm. First a bit of terminology: For any wff α, let $\|\alpha\|$ be the number of different datoms and history atoms in α.

Theorem 5-4. Let Q be an incoming query. If assumptions 1, 2, and 3 are satisfied, then Q will be accepted by the Lazy Algorithm if Q is affordable and for each update family $\mathcal{F}$ in the lazy graph that contains a *parent* of Q, the difference between the cost bound for $\mathcal{F}$ and the amount spent so far on executed updates of $\mathcal{F}$ is at least

$$\sum_{\substack{U \in \mathcal{F} \\ U \text{ parent of } Q}} \Big(\text{EstCost}(\phi_U) + \|\omega_U\| - 2 + \sum_{\substack{f \in \omega_U \\ g \in Q \\ f \sim_\sigma g}} (\text{EstCost}(f) + 3) \Big). \quad \diamond \quad \text{5-10}$$

Theorem 5-4 implies that when assumptions 1, 2, and 3 are satisfied, there is a quick test for acceptability that only requires looking at the parents of the query, rather than all ancestors of the query. The proof of the theorem will show what splits to use to achieve this bound.

Proof of Theorem 5-4. Let us examine how to split a particular parent U of Q to achieve the bound in formula 5-10. Suppose f_1 through f_n are the datoms of ω_U that have arcs going out to datoms in Q. Then first split U horizontally according to Splitting Rule 3, removing all datoms and history atoms of ω_U from ω_{U_1} except f_1 through f_n. Then U_1 is still an ancestor of Q, with the same estimated costs for its nodes as those nodes had in U before the split, by assumption 2; and with the same arcs going to Q as went from U to Q. U_2, however, is not an ancestor of Q, by assumption 1.

The next step is to separate out the datoms of ω_{U_1} into individual updates. To accomplish this, split U_1 $n-1$ times according to Splitting Rule 3, removing datom f_i from U_1 on the ith split. By assumption 2, these splits will not alter the cost information for any node of a split update that originally appeared in U. Note that there is no need to define a new history atom $H(U)$ at each split of Splitting Rule 3; every update that is split off can use the same selection

clause $H(U)$, defined by U_1 with the formula $H(U) \leftrightarrow \phi_U$. This optimization was mentioned earlier, in Remark 5-1. The small savings realized through reuse of $H(U)$ has been included in formula 5-10.

In the lazy graph resulting from this second round of splits, there are n updates split off from U that are now ancestors of Q; for simplicity, rename the updates in the graph as necessary so that these n updates are called U'_1 through U'_n. We now review the form and estimate the costs of U'_1 through U'_n, beginning with U'_1.

Only one datom is a subformula of $\omega_{U'_1}$: f_n. All other datoms of ω_U have been replaced by history atoms in $(\omega)_{U'_1}$: there will be $\|\omega_U\| - 1$ different history atoms in $\omega_{U'_1}$. By assumption 2, each history atom has an estimated cost of 1. The selection clause of U'_1 is the same as that for U. By assumption 2, it follows that $\text{EstCost}(\phi_{U'_1}) = \text{EstCost}(\phi_U)$. U'_1 defines a new history atom $H(U)$, using the formula $H(U) \leftrightarrow \phi$. Therefore the estimated cost of U'_1 is no more than

$$\text{EstCost}(f_n) + (\|\omega_U\| - 1) + \text{EstCost}(\phi_U) + 1.$$

U'_2 through U'_n all take the form

INSERT $(f \leftrightarrow H(f, c_i)) \wedge ((\bigvee_{\substack{\sigma \in S \\ f_i \sim \sigma\sigma}} \sigma) \rightarrow (H(f, \mathcal{F}) \leftrightarrow H(f, d_i))))_{\sigma'}$
WHERE $H(U)$,

for some constants c_i and d_i, where S is the set of datoms remaining in ω at the time f is being split out of ω. This form has an upper bound on estimated costs of $\text{EstCost}(f) + 3$.

Summing the estimated costs of U_1 through U_n produces an upper bound estimate of

$$\text{EstCost}(f_n) + (\|\omega_U\| - 1) + \text{EstCost}(\phi_U) + 1 + \sum_{1 \le i < n} (\text{EstCost}(f_i) + 3),$$

which simplifies to

$$\|\omega_U\| - 2 + \text{EstCost}(\phi_U) + \sum_{1 \le i \le n} (\text{EstCost}(f_i) + 3). \qquad \text{5-11}$$

If these two stages of splitting are applied to all the parents of Q, then summing formula 5-11 over all parent families gives formula 5-10. However, the theorem is not quite proven: Q may still have ancestors in the lazy graph other than its parents. It will require two more rounds of splitting to remove these undesirable ancestors. These rounds of splits, however, will not change the estimated costs of the parents of Q.

Let V be a parent of Q in the lazy graph after the first two rounds of splitting. To eliminate unwanted ancestors of V, let ϕ' be the disjunction of all the substitutions labelling arcs that go from V to Q. Split V vertically on ϕ'

according to Splitting Rule 5, producing updates V_1 and V_2. Splitting V with Splitting Rule 5 does not change the cost estimates for the nodes of V, because vertical splitting does not change the datoms or history atoms of the split update.

After this third stage of splitting, suppose there is still a parent A of V_1 such that A is not a parent of Q. Then by assumptions 1 and 3, there must be a datom f of ω_A that unifies with a datom f' of ω_{U_1} under a most general substitution σ'. Since f' is the only datom in ω_{U_1}, f' must unify with some datom g of Q under a most general substitution σ. Further, by definition of U_1, σ must be one of the substitutions in the selection clause ϕ' of ϕ_{U_1}:

$$\phi' \text{ is } \phi_U \wedge \bigvee_{\substack{g \in Q \\ f' \sim_{\sigma} g}} \sigma.$$

Since A is a parent of V_1, by Step 2 of the NAP Algorithm it must be the case that $\sigma' \wedge \phi_U \wedge (\bigvee_{\substack{g \in Q \\ f \sim_{\sigma} g}} \sigma)$ is satisfiable; therefore for some choice of σ, $\sigma' \wedge \sigma$ must be satisfiable. This implies that f unifies with g under substitution $\sigma' \wedge \sigma$. Yet A is not a parent of Q; therefore it must be the case that there is no arc from f to g because that arc fails test (2) or (3) of the NAP Algorithm Step 2.

First consider the case where the arc fails test (3). In this case there is an equality wff α such that, say, ϕ_A logically entails α and ϕ_Q logically entails $\neg\alpha$. Let ϕ' be the wff $\neg\alpha$, and split V_1 vertically with Splitting Rule 5 on ϕ', creating updates V_3 and V_4. Then by test (3) of the NAP Algorithm Step 2, V_4 is not a parent of Q, and A is not a parent of V_3. Therefore A is no longer an ancestor of Q by any path that goes through V_3 or V_4.

Now consider the case where the arc from f in A to g in Q passed test (3) but failed test (2) of Step 2 of the NAP Algorithm. In this case either $\sigma'' \wedge \phi_Q$ or $\sigma'' \wedge \phi_A$ must be unsatisfiable, where σ'' is the most general substitution under which f and g unify. A simplified diagram of the relevant portion of the lazy graph, including the illegal arc σ'' from A to Q, appears in figure 5-7.

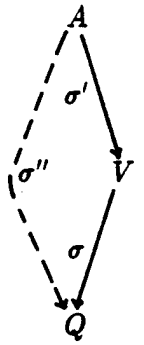


Figure 5-7. Portion of simplified lazy graph.

Let Σ be the set of all choices for σ'', that is, the set of all substitutions σ'' such that $f \in \omega_A$, $g \in Q$, and $f \sim_{\sigma''} g$. Let ϕ' be $\bigvee_{\sigma'' \in \Sigma} \tau(\sigma'')$, where $\tau(\sigma'')$ is σ''

if $\sigma'' \wedge \phi_A$ is unsatisfiable, and is $\neg\sigma''$ otherwise. Split V_1 vertically with Splitting Rule 5 on ϕ', creating updates V_3 and V_4. We will now show that there is no path from A to Q through V_3 or V_4.

Suppose that $\sigma'' \wedge \phi_Q$ is unsatisfiable. Then ϕ_Q logically entails $\neg\sigma''$. Since σ'' is at least as general as σ, $\neg\sigma''$ logically entails $\neg\sigma$. It follows that $\sigma \wedge \phi_Q$ is also unsatisfiable. Therefore no arc labelled σ can go from V_3 or V_4 to Q, by test (2) of the NAP Algorithm Step 2. Further, ϕ_{U_3} logically entails $\bigvee_{\sigma'' \in \Sigma} \tau(\sigma'')$. Therefore ϕ_{U_3} logically entails $\neg\sigma''$, and ϕ_{U_3} logically entails $\neg\sigma$ and $\neg\sigma'$ as well. We conclude that there can be no arc labelled σ from V_3 to Q and no arc labelled σ' from A to V_3, when $\sigma'' \wedge \phi_Q$ is unsatisfiable.

Following the same line of reasoning when $\sigma'' \wedge \phi_A$ is unsatisfiable leads to the conclusion that no arc labelled σ' can go from A to V_3 or V_4, no arc labelled σ can go from V_4 to Q, and no arc labelled σ' can go from A to V_4, when $\sigma'' \wedge \phi_Q$ is unsatisfiable.

It follows that the only arcs between A, V_3, V_4, and Q fall into the following three classes:

1. Arcs from V_3 to V_4.
2. Arcs labelled σ from V_3 to Q, when $\sigma'' \wedge \phi_A$ is unsatisfiable.
3. Arcs labelled σ' from A to V_4, when $\sigma'' \wedge \phi_Q$ is unsatisfiable.

Figure 5-8 depicts a simplified lazy graph containing A, V_3, V_4, Q, the phantom arc σ'' from A to Q, and for clarity V and its arcs as well, though of course the Splitting Algorithm would have removed V from the lazy graph before V_3 and V_4 were inserted.

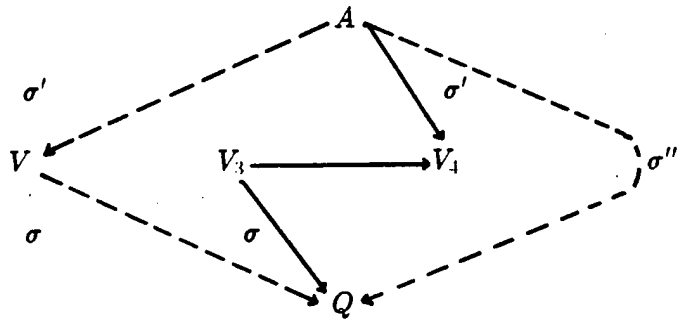


Figure 5-8. Portion of simplified lazy graph.

Note that there is no path from A to Q via V_3 or V_4. Therefore after applying this final round of splitting to all ancestors of Q that are not parents of Q, all such ancestors will no longer be ancestors of Q, and the theorem follows. ◇

An improvement to this theorem immediately suggests itself. Read/write dependency arcs coming into an update need not prevent execution of that update, because history predicates can be used as a versioning mechanism to eliminate the read/write conflicts. This means that it is quite possible to execute updates in the lazy graph without following a pure topological sort, as read/write arcs need not determine execution order. For example, if update U_1 is INSERT Emp(Reid, CSD) WHERE Emp(Reid, CSL), and update U_2 is INSERT $\neg$Emp(Reid, CSL) WHERE T, then there is a read/write conflict between U_1 and U_2. However, U_2 can be executed ahead of U_1 as long as U_1 reads H(Emp(Reid, CSL), U_2) rather than Emp(Reid, CSL).

The situation is a bit more complex if the read/write dependency arc has a substitution label other than T. For example, if U_1 were INSERT Emp(Reid, CSD) WHERE Emp(Reid, ϵ), then U_1 could not get by with reading H(Emp(Reid, ϵ), U_2), for H(Emp(Reid, ϵ), U_2) may be true in models where Emp(Reid, ϵ) never was true. This anomaly can occur whenever Emp(Reid, ϵ) is not already a subformula of $\mathcal{T}$ at the time U_2 is executed. The solution is for U_1 to be replaced by INSERT Emp(Reid, CSD) WHERE (Emp(Reid, ϵ)$\wedge$ ($\epsilon \neq$CSL)) $\vee$(H(Emp(Reid, ϵ), U_2)$\wedge$ (ϵ =CSL)), if U_2 is executed before U_1.

5.10. Summary and Conclusion

As noted in Chapter 3, the Update Algorithm may lead to excessive increases in the size of an extended relational theory $\mathcal{T}$ as expensive updates are incorporated into $\mathcal{T}$. To control the growth of $\mathcal{T}$, we propose a scheme of lazy evaluation for updates. Lazy evaluation strictly bounds the growth of the extended relational theory caused by each update, via user-specified limits on permissible size increases. Under lazy evaluation, an overly-expensive update U will be stored away rather than executed, in the hopes that new information on costly null values will reduce the expense of executing U before the information contained in U is needed for an incoming query. If an incoming query unavoidably depends on the results of an overly expensive portion of an update, the query must be rejected, as there is no way to reason about the information in the update other than by incorporating it directly in the extended relational theory. When a query is rejected, the originator of the query is notified of the exact reasons for the rejection. The query may be resubmitted once the range of possible values of the troublesome nulls has been narrowed down. The bottom line for an efficient implementation of updates, however, is that null values should not be permitted to occur as attribute values for attributes heavily used in update selection clauses—particularly those used as join attributes.

Chapter 6: Enforcement of Dependency Axioms

Until now, we have considered extended relational theories without *type axioms* (an encoding of the schema of the database) and *dependency axioms* (e.g., functional and multivalued dependencies [Ullman 82]), because the complications introduced by those axioms are orthogonal to the other issues in updating extended relational theories.

Dependency axioms can play a number of roles in ordinary relational databases during updates. A policy decision must be made for each axiom, based on the intended semantics of the axiom. If a requested update would lead to a database state that violated the axiom, then the database management system's possible enforcement policies include rejecting the update; performing the update and also making additional changes in the database to make it obey the axiom; and performing the update and ignoring the temporary inconsistency. In databases with incomplete information, dependency axioms play another important role, that of identifying and eliminating "impossible" alternative worlds. For example, if a manager can only manage one department at a time and we know that Mgr(Nilsson, CSD)$\vee$ Mgr(Nilsson, EE) is true, then the alternative world where Nilsson manages both CSD and EE is inconsistent with the axiom and can be eliminated out of hand. As yet another possible enforcement policy, if a requested update would create alternative worlds that violate the axiom and the update is known to be correct, then the axiom can be changed. This policy is implemented in Step 1 of the Update Algorithm, for the completion axioms. These axiom enforcement policies are summarized in Table 6-1.

All five of these axiom enforcement policies are reasonable; the correct choice of a policy for a particular axiom depends on the semantics of the axiom and the database, and we delegate this decision to a higher authority, such as the database administrator. The remainder of this chapter presents a mechanism to enforce axioms by permanently weeding out impossible alternative worlds (called *strict enforcement*), to be employed as the database administrator sees fit. En route we will point out how to perform *passive enforcement*, that is, to ignore temporary inconsistencies between the theory body and its axioms—perhaps not a logically sound procedure, but one used daily by humans with spectacular success.

We begin by defining the class of type and dependency axioms under consideration here, then explain what kind of axiom enforcement is provided by Versions I and II (presented in Chapters 3 and 4, respectively) of the Update Algorithm, and extend the Update Algorithm to provide strict axiom enforcement. To simplify the presentation of this material, all updates are assumed to

If an update U applied to alternative world $\mathcal{A}$ would create an alternative world $\mathcal{A}'$ that violates axiom α, then ...

1. Reject U.
2. Make additional changes in $\mathcal{A}'$ so that it does not violate α.
3. Ignore the temporary inconsistency and permit a later update U' to remove the violation of α in $\mathcal{A}'$.
4. Eliminate $\mathcal{A}'$ permanently.
5. Change α.

Table 6-1. Axiom enforcement policies.

be variable–free, and Version I of the Update Algorithm will be used as the point of departure.

6.1. Extended Relational Theories with Type and Dependency Axioms

Type axioms encode the relationship between predicates and attributes, so that, for example, given that Emp(Reid, CSD) is true, it follows that Reid must be an element in the employee domain and CSD must be a department. Type axioms are useful for controlling the effects of negation and ensuring that queries and updates are safe.† For example, Reiter [84b] suggests that all constants, Skolem constants, and variables occurring in queries should be typed so that the query will be safe. System R [Chamberlain 76] and INGRES [Stonebraker 85] employ a similar mechanism on a higher level by requiring `RANGE OF` and `SELECT FROM` statements for all the tuple variables of requests.

For a formal definition of type axioms, distinguish a particular set A of unary predicates of $\mathcal{L}$ as the *attributes* of $\mathcal{L}$. For each n-ary predicate R not in A, an extended relational theory $\mathcal{T}$ with type axioms must contain exactly one axiom of the form

$$\forall x_1 \cdots \forall x_n (R(x_1, \ldots, x_n) \rightarrow (A_1(x_1) \wedge \cdots \wedge A_n(x_n))),$$

where $A_1, \ldots, A_n$ are predicates in A. Further, each predicate in A must appear in one or more type axioms.

Strict enforcement of type axioms may be painful if experienced directly by users. For example, rather than just requesting `INSERT` Emp(Reid, CSD) `WHERE` T, under strict enforcement the user must remember to ensure that Reid and CSD are elements in the correct domains: `INSERT` Emp(Reid, CSD)∧ Employee(Reid)∧ Department(CSD) `WHERE` T. A better alternative is to enforce the type axioms through axiom modification, as is done for the completion axioms: to modify the type axioms during the update so that the update cannot violate them.

† A domain completion axiom can be employed to the same end.

The definition of extended relational theories must be extended to include a set of universally quantified axioms that are to be strictly enforced (*strict axioms*, for short). As far as the Update Algorithm is concerned, all strict axioms can be lumped together in a Strict Axiom section of the extended relational theory. So in addition to a body and a set of completion axioms, items 1 and 2 in the definition of an extended relational theory in Section 3.1, every extended relational theory $\mathcal{T}$ now includes a third section:

3. *Strict Axioms:* A set of *strictly enforceable* (to be defined in Section 6.3) universal sentences not containing history predicates. $\Diamond$

For example, a typical functional dependency would be $\forall x_1 \forall x_2 \forall x_3$ $((\text{Emp}(x_1, x_2) \wedge \text{Emp}(x_1, x_3)) \rightarrow (x_2 = x_3))$.

Some universal sentences not containing history predicates will be too expensive to enforce easily, and these are excluded from the strict axiom section. Section 6.3 will single out the affordable axioms in its definition of strict enforceability. We do not present that definition here because it relies on intuitions that will be developed in later sections.

6.2. Semantics of Updates Revisited

The semantics of updates must be augmented with one additional proviso for strict enforcement of axioms: *Every model $\mathcal{M}'$ in $U(\mathcal{M})$ must satisfy the strict axioms of $\mathcal{T}$*. In the proofs below, this new provision is referred to as rule 3 in the definition of INSERT, to be appended to rules 1 and 2 of the original definition:

(3) for all strict axioms α of $\mathcal{T}$, α is true in $\mathcal{M}'$.

A particular update algorithm *strictly enforces* α for an update U if for every extended relational theory $\mathcal{T}$, rule 3 is satisfied by all members of Worlds($U(\mathcal{T})$). For the remainder of this chapter, "enforcement" will mean strict enforcement unless otherwise noted.

In this discussion, the strict axioms will be permanently fixed for each database schema. It is trivial to extend the types of updates permitted to allow addition of new dependencies, constants, or relations.

6.3. The Update Algorithm Revisited

What sort of axiom enforcement is provided by the Update Algorithm Version I? Suppose the extended relational theory body consists of the wffs Emp(Reid, EE) and Emp(Reid, CSD)$\vee$ Mgr(Nilsson, CSD). If there is a functional dependency stating that an employee can be in at most one department at a time, then some models of the body of this theory are inconsistent with that functional dependency, and will not be models of the theory containing that functional dependency as a strict axiom. It follows that in every alternative world of the full theory containing that axiom, Nilsson is the manager of CSD. Unfortunately, when using the Update Algorithm Version I, a later update may "rescue" the alternative worlds where Reid is in two departments and pop them back into

existence. In the current example, a new update INSERT ¬Emp(Reid, EE) WHERE T, when processed according to the Update Algorithm, would blithely produce alternative worlds where Nilsson is not the manager of CSD, though these worlds are in fact impossible if the axiom and update are interpreted strictly.

These "rescued" worlds arise because the versions of the Update Algorithm seen so far enforce the department axiom passively, merely using it to shape answers to queries at the current moment. This *passive enforcement* allows the update INSERT ¬Emp(Reid, EE) WHERE T to rescue the alternative world where Reid is in both CSD and EE and make its descendant a legitimate world for future computations. In general, any time an update removes all axiom violations from an "impossible" alternative world, Versions I, II, and III of the Update Algorithm will rescue the descendants of that alternative world.

As mentioned in the introduction to this chapter, passive enforcement will be the best choice for some applications, and when passive enforcement is desired, then the update algorithms of previous chapters may be used. If an axiom is to be strictly enforced, however, then the alternative worlds produced by these update algorithms will be incorrect. In the remainder of this chapter, we consider how to alter the Update Algorithm to move from passive to strict enforcement: how to eliminate forever the alternative world where Reid is in both CSD and EE.

Proposition 6-1 suggests a means of axiom enforcement through instantiation of axioms.

Proposition 6-1. Let $\mathcal{T}$ be an extended relational theory and let $\mathcal{T}+\alpha$ be that theory plus a universal sentence α without history predicates, to be strictly enforced. If Worlds($\mathcal{T}$) = Worlds($\mathcal{T}+\alpha$), then the Update Algorithm Version I will strictly enforce α on the next update U applied to $\mathcal{T}+\alpha$. $\Diamond$

Proof of Proposition 6-1. By definition all models that are produced by the Update Algorithm do satisfy α. The worry is that some model produced might not be descended from a model that satisfies α. Let $\mathcal{T}'$ be the theory produced from $\mathcal{T}$ by the Update Algorithm Version I. For the alternative world of a model $\mathcal{M}'$ of $\mathcal{T}'$ to be produced by U, $\mathcal{M}'$ must be derived from a model $\mathcal{M}$ that satisfies $\mathcal{T}$, as proved in Theorem 4-1. Then by assumption $\mathcal{M}$ also satisfies α, so $\mathcal{M}'$ is descended from a model of $\mathcal{T}+\alpha$. $\Diamond$

This proposition immediately suggests one means of axiom enforcement. If variables are allowed to occur in the body of the extended relational theory, as in Section 4.4, one can keep a copy of the dependency axiom in the body of the theory at all times, and the axiom will always be strictly enforced. In the remainder of this discussion, we assume that variables are *not* permitted to occur in the body of the extended relational theory. The variable-free case is important because management of very large volumes of data relies on regularity and simplicity of the data to allow efficient access to and inference from the data.

When variables appear in the body of the theory rather than solely in its axioms, the assumptions of regularity and simplicity must be abandoned, and processing costs will rise steeply.

To ensure that the alternative worlds of $\mathcal{T}+\alpha$ are the same as those of $\mathcal{T}$, at first glance it might seem sufficient to add a ground instantiation $(\alpha)_\sigma$[†] of α to $\mathcal{T}$ for every substitution σ of constants and Skolem constants such that $(\alpha)_\sigma$ is not valid. Unfortunately, this assumption is flawed on two accounts. First, there may be elements in the universe that are not named by any constants or Skolem constants in $\mathcal{T}$. It is not possible to instantiate the axiom with these elements, yet they must be considered: the user might pose a yes/no query using only symbols in $\mathcal{L}$ that would detect rescues of worlds involving these unnamed elements.[††] As shown below, the completion axioms will help to prevent this anomaly.

Second, even if all elements in the universe are named by constants or referenced by Skolem constants, there may be an infinite number of these constants, and an infinite number of instantiations may be needed to enforce α. Infinite theory bodies are unpleasant to contemplate. Fortunately, the fact that only a finite number of datoms can be true at any given moment will make it unnecessary to resort to infinite theory bodies.

To see why only a finite number of instantiations of a strict axiom α are needed to prevent the rescue of a model, let $\mathcal{M}$ be a model of $\mathcal{T}$ that fails to satisfy α. Then $\mathcal{M}$ fails to satisfy $(\alpha)_b$, for some binding b of all the variables of α to universe elements of $\mathcal{M}$.[†††] Let σ be the Skolem constant substitution for $\mathcal{M}$ with respect to $\mathcal{T}$. Create β by replacing every datom of $(\alpha)_b$ that does not appear in $(\mathcal{T})_\sigma$ by the truth value F, and β is false in $\mathcal{M}$. Now the set of all possible bindings b is infinite, giving an infinite number of possible $(\alpha)_b$s, but the set of all possible βs is finite because $\mathcal{T}$ is finite. Further, the universe elements appearing in β are all named by constants or Skolem constants in $\mathcal{L}$, because all those elements appear in $(\mathcal{T})_\sigma$. Let $\text{Inst}(\alpha)$ be the finite set of βs for $\mathcal{M}$. Then adding $\text{Inst}(\alpha)$ to $\mathcal{T}$ will prevent $\mathcal{M}$ from being rescued by *any* incoming update.

At this point it may seem once again that our task is complete: we have shown how to prevent $\mathcal{M}$ from being rescued by adding a finite set of formulas $\text{Inst}(\alpha)$ to $\mathcal{T}$. Further, although $\mathcal{T}$ may have an infinite number of models $\mathcal{M}$, there are only a finite number of possible formulas for $\text{Inst}(\alpha)$, so this technique

---

† For α a universal sentence in prenex form with prefix $\forall x_1 \cdots \forall x_n$, and σ a substitution of the form $\frac{x_1}{c_1} \cdots \frac{x_n}{c_n}$, $(\alpha)_\sigma$ is the wff obtained from α by removing α's prefix and then applying σ. A substitution for a subset of the variables of α is defined analogously.

†† Example: Axiom is $\forall x(R(a) \rightarrow R(x))$, body of T is initially empty, as always $\mathcal{L}$ contains an infinite number of Skolem constants, and the universe contains two elements, of which only a is named in $\mathcal{L}$. If the user requests `INSERT` $R(a)$ `WHERE` T, followed by `INSERT` $\neg R(a)$ `WHERE` T, then the resulting theory should have no alternative worlds, as the first update eliminates them all, due to the completion axioms. But no matter what set of formulas an update algorithm adds to the body of T, the wff $\neg R(\epsilon)$ will be consistent with the new body.

††† As usual, $(\alpha)_b$ is not a proper object: strictly speaking, we should extend $\mathcal{L}$ or at least map all of α to the corresponding elements and relations in $\mathcal{M}$. As in previous chapters, this hybrid notation will be used freely.

can be extended to prevent rescues of all models. Unfortunately, there are two obstacles to implementation of this scheme:

1. Exactly which formulas are in Inst(α)? The set of formulas in Inst(α) may depend on the universe of $\mathcal{M}$. How can this be predicted efficiently?
2. Do the same formulas occur in Inst(α) for every choice of $\mathcal{M}$? If not, how can the different sets be merged into one large set for inclusion in the body of $\mathcal{T}$?

If different formulas occur for different choices of $\mathcal{M}$, then one cannot just add Inst(α) to the body of $\mathcal{T}$ without changing the models of $\mathcal{T}$. For example, let $\mathcal{L}$ contain a single constant a, and let α be $\forall x(R(a) \rightarrow R(x))$. If the body of $\mathcal{T}$ is the single wff $R(a)$, then $\mathcal{T}$ has one model, where a is the only element of the universe. Let $\mathcal{M}$ be a model of $\mathcal{T}-\alpha$ in which $R(a)$ is true and the universe contains two elements, say a and b. Then an appropriate instantiation of α that will prevent $\mathcal{M}$ from being rescued by the update `INSERT` $\neg R(a)$ `WHERE` T is the wff $R(a) \rightarrow$ F. But adding this wff to the body of $\mathcal{T}$ would eliminate the one legitimate model of $\mathcal{T}$.

There are certain cases where questions (1) and (2) above have easy answers. For example, if the universes of all models of $\mathcal{T}$ are isomorphic, then the Inst(α) constructed for one model $\mathcal{M}$ is identical to that constructed for every other model. There are two ways of guaranteeing essentially identical universes: either include a domain completion axiom in $\mathcal{T}$ ($\forall x((x = c_1) \vee \cdots \vee (x = c_m))$[†], where each c_i is a *constant*) and thereby standardize the universe; or else guarantee that the universe of every model is infinite, by including an infinite set of constants in $\mathcal{L}$. In this latter case, universes may vary greatly from model to model, but every universe is guaranteed to be sufficiently large that Inst(α) will contain the same wffs for every model, giving an answer to question (1). Further, with respect to question (1), in the case of an infinite universe it is easy to generate Inst(α), because Inst(α) contains every possible instantiation of α, roughly speaking.

In the case where $\mathcal{T}$ includes a domain completion axiom, question (2) has an easy answer: there is a reasonable way of constructing Inst(α) such that Inst(α) is the same for all models of $\mathcal{T}$. (Note that no Skolem constants can appear in this domain completion axiom; otherwise, the universe would not necessarily be standard across all models of $\mathcal{T}$.) However, question (1) remains unanswered: exactly which formulas are in Inst(α)? As will be shown after the presentation of the new Update Algorithm, it takes time polynomial in the size of $\mathcal{T}$ and the domain to determine whether a particular formula is in Inst(α). The problem is $\mathcal{NP}$–complete if the number of variables in α is also part of the input; but since the number of variables in α is in practice small and bounded, this is not cause for alarm.

[†] Domain completion axioms conflict with the requirement of Section 3.1.1 that $\mathcal{L}$ contain an infinite number of constants. This requirement is present to insure that one will always be able to find a history atom $H(f, U)$ that does not unify with any history atom in the body of $\mathcal{T}$. If $\mathcal{L}$ contains only a finite set of constants, as implied by a domain completion axiom, then the whole system must come to a pause if the Update Algorithm runs out of "new" history atoms.

Why might it be expensive to test a candidate formula β for membership in Inst(α)? One must test whether there exists a substitution σ over $\mathcal{L}$ such that β may be obtained from $(\alpha)_\sigma$ through replacing by F all datoms of $(\alpha)_\sigma$ that do not unify with any atom of $\mathcal{T}$. In general, this may require exhaustive generation and testing of potential σs. The problem may be formalized as that of finding a set of assignments to variables from a finite domain given certain forbidden combinations of assignments. Since the number of possible substitutions is m^n for a domain of size m with n variables in α, a generate–and–test strategy gives an algorithm that is polynomial in the size of $\mathcal{T}$ and the domain, but still quite expensive. On the other hand, in practice one expects the domain to be very large, and the datoms of $\mathcal{T}$ to be sparse within the cross product of the domain, making it very likely that a suitable substitution σ can be found quickly, if one exists. More will be said on this issue after the presentation of the new Update Algorithm.

Setting constraints on the universe is not the only way to achieve enforceability. For example, a strict axiom can only cause rescues if it contains data predicates; if there are no data predicates in α, then α is trivially enforceable, as even the Update Algorithm Version I will enforce α. For example, suppose $\mathcal{T}$ is an extended relational theory and α a domain completion axiom. Let $\mathcal{M}$ be a model of $\mathcal{T}$ that violates α. No update to $\mathcal{M}$ can remove that axiom violation, because α contains no atomic formulas whose truth valuations could be changed in $\mathcal{M}$ to effect a rescue.

There is one other type of axiom where questions (1) and (2) have easy answers. Interestingly, this class encompasses the axioms of traditional interest, such as functional dependencies and multivalued dependencies. The key feature of this axiom type is that an instantiation $(\alpha)_b$ of such an axiom α is guaranteed to be valid if *any* variable of α is bound to a universe element not named in $\mathcal{L}$. If $(\alpha)_b$ is valid, then no model can violate $(\alpha)_b$, and hence there is no need to include $(\alpha)_b$ in Inst(α).

Strict axioms that are easy to enforce are called *enforceable*:

Definition. A universal sentence α not containing history predicates is *strictly enforceable* if any of the following four conditions is satisfied.

1. α contains no data predicates.
2. $\mathcal{L}$ contains an infinite set of constants.
3. Among its completion axioms, $\mathcal{T}$ contains a domain completion axiom $\forall x((x = c_1) \vee \cdots \vee (x = c_m))$, for each c_i a constant.
4. For x a variable of α, (a) substitute x for a subset S of the variables of α, creating β; (b) replace the atomic formula $x = x$ by T wherever it occurs in β; and (c) replace any other atomic formulas of β containing x by the truth value F. If β is valid for all choices of x and S, then α is strictly enforceable. $\Diamond$

Please note that if β in condition 4 is logically equivalent to α, then though α fails to be strictly enforceable, α can be replaced by the equivalent and simpler axiom β, and β may well be strictly enforceable. For example, $\forall x \forall y \forall z \forall w(((\text{Emp}(x,y) \wedge \text{Emp}(x,z)) \rightarrow (y=z)) \wedge (\text{Emp}(w,w) \vee \neg \text{Emp}(w,w)))$ fails to meet condition 4 because of w, but in this case β is logically equivalent to $\forall x \forall y \forall z ((\text{Emp}(x,y) \wedge \text{Emp}(x,z)) \rightarrow (y = z))$, which is strictly enforceable because it satisfies condition 4.

As mentioned earlier, most common types of dependencies with semantics that are suitable for strict enforcement satisfy condition 4. For example, all functional dependencies and multi-valued dependencies satisfy condition 4. To see this, consider the dependency $\forall x \forall y \forall z((\text{Emp}(x,y) \wedge \text{Emp}(x,z)) \rightarrow (y = z))$. If the atomic formulas over any one variable are false, then the preconditions of the implication are false, and therefore the axiom is satisfied. On the other hand, an axiom stating that an employee must be in every department save possibly one—$\forall x \forall y \forall z((\neg \text{Emp}(x,y) \wedge \neg \text{Emp}(x,z)) \rightarrow (y = z))$—does not meet condition 4, as, for example, $\forall x \forall y \forall z ((\mathsf{T} \wedge \neg \text{Emp}(x,z)) \rightarrow \mathsf{F})$ is not valid; nor is it logically equivalent to the original axiom.

If a potential strict axiom α does not meet conditions 1, 2, 3, or 4, then questions (1) and (2) above do not have easy answers. Section 6.5 presents a method of enforcing such axioms, using a rather painful method of instantiation.

We are now ready to extend the Update Algorithm to handle strictly enforceable axioms α. The technique for preventing rescues of alternative worlds that violate α has already been presented: a finite set $\text{Inst}(\alpha)$ of "instantiations" of α is constructed and added to $\mathcal{T}$. At this point it only remains to present an efficient method of generating the set $\text{Inst}(\alpha)$. This will be accomplished by Step 1 of the new Update Algorithm, which instantiates the strict axioms with a subset of the constants and Skolem constants in the body of $\mathcal{T}$. In Step 3 those instantiations will be added to the body of $\mathcal{T}$. The key point of Step 1 is to instantiate the axioms as few times as possible, to minimize the size of $\mathcal{T}'$. One would like to instantiate the axioms so as to produce only datoms that unify with datoms already in $\mathcal{T}$; since all other datoms are known to be false, they can be replaced by the truth value F in the instantiation. In fact this is exactly what Step 1 does, albeit a bit more conservatively.

The goal of minimizing the size of $\text{Inst}(\alpha)$ leads to a complicated instantiation process for α. Step 1 is an iterative process, where successively more variables of α are carefully bound. Step 1 is difficult to understand, and for that reason several parenthetical remarks are included in the presentation of the algorithm.

Steps 2, 4, 5, and 6 in the new version of the Update Algorithm are identical or nearly identical to Steps 1, 2, 3, and 4, respectively, of the Update Algorithm Version I.

The Update Algorithm (Version IV)

Input. An extended relational theory $\mathcal{T}$, including a strict axiom section, and a ground update U.

Output. $\mathcal{T}'$, an updated version of $\mathcal{T}$.

Procedure. A sequence of six steps:

Step 1. Instantiate strict axioms. This step constructs the set of wffs Inst(α) for each strict axiom α. (Examples of the operation of Step 1 appear at the end of the presentation of the algorithm.) Let Inst(α) contain the set of all wffs $(\alpha)_\sigma$ constructed as follows:

a. *Choose initial binding.* Choose a datom f in ω and an atomic formula g in α such that f and g unify under a most general substitution σ'. Let σ be a substitution containing just the substitutions for variables in σ'. (Intuitively, Step 1a guarantees that some datom of ω appears in $(\alpha)_\sigma$. If this condition were not met, then U could not possibly rescue a world violating $(\alpha)_\sigma$.)

b. *Bind additional variables.* Repeat the following zero or more times:

   Choose a datom f in $\mathcal{T}$ and an atomic formula g in $(\alpha)_\sigma$ such that f and g unify under a most general substitution σ'. Append the substitutions for variables in σ' to σ.

(Intuitively, Step 1b instantiates variables of α so that the datoms appearing in $(\alpha)_\sigma$ might possibly be true in some model of $\mathcal{T}$. Intuitively, all variables unbound after Step 1b will be instantiated to universe elements not referenced by $\mathcal{T}$, so that non-ground atomic formulas in $(\alpha)_\sigma$ can be replaced by F. The actual process is a bit more complicated; for example, the equality predicate will require special treatment.)

c. *Decide which equality atomic formulas will be true.* Repeat the following zero or more times:

   If a variable x occurs in an equality atomic formula $x{=}y$ or $y{=}x$ in $(\alpha)_\sigma$, for y a variable, constant, or Skolem constant, then append to σ the substitution $\frac{x}{y}$.

This completes the construction of σ. The final four phases of Step 1 are devoted to simplifying the formulas in Inst(α).

d. *Remove true equality formulas.* For each wff β in Inst(α), replace the reflexive equality formula (e.g., $x = x$) by the truth value T wherever it is a subformula of β.

e. *Discard unwanted βs.* For each wff β in Inst(α), remove β from Inst(α) if

   - α satisfies condition 4 in the definition of strictly enforceable axioms and β contains variables. (In this case, Step 1f applied to β will yield a valid wff, so β need not be included in Inst(α).)
   - α satisfies condition 3 and Falsifiable(β, $\mathcal{T}$) is false. (In this case, some model of $\mathcal{T}$ might not satisfy β if all non-ground atoms of β were replaced by F. The Falsifiable() routine will be described separately.)

f. *Eliminate remaining variables.* For each wff β in Inst(α), replace all non-ground atomic formulas in β by the truth value F.

g. *Simplify β.* For each wff β in Inst(α), replace datoms in β that do not unify with any datom in $\mathcal{T}$ or ω by F, if desired; and replace β by a simpler logically equivalent wff, if desired. (These simplifications will help to minimize the number of different instantiations and their total size.)

Step 2. Maintain the closed–world assumption. To maintain the closed–world assumption, all datoms in ω, ϕ, and Inst(α) need to be represented in the completion axioms of $\mathcal{T}$. First change the body of $\mathcal{T}$ to reflect the new completion axioms: for each atom g that is a subformula of ω, ϕ, or Inst(α) but not $\mathcal{T}$, let Σ_0 be the set of the most general substitutions σ such that for some datom f that is a subformula of $\mathcal{T}$, f unifies with g under σ. If Σ_0 is the empty set, then add $\neg g$ to the body of $\mathcal{T}$; otherwise, add the wff

$$g \rightarrow \bigvee_{\sigma \in \Sigma_0} \sigma \tag{1}$$

to the body of $\mathcal{T}$. Then for every datom g of $\mathcal{T}$ not represented in the completion axioms, add a disjunct representing g to those axioms.

Step 3. Add Inst(α) to $\mathcal{T}$. Add all the wffs in Inst(α) to the body of $\mathcal{T}$.

Step 4. Make history. (Same as Step 2 of the Update Algorithm Version I) For each atom f that is a subformula of $\mathcal{T}'$ and unifies with an atom of ω, replace all occurrences of f in the body of $\mathcal{T}$ by the history atom $H(f,U)$. In other words, replace the body $\mathcal{B}$ of $\mathcal{T}$ by $(\mathcal{B})_{\sigma_H}$.

Step 5. Define the scope of the update. (Same as Step 3 of the Update Algorithm Version I) Add the wff $(\phi)_{\sigma_H} \rightarrow \omega$ to $\mathcal{T}'$.

Step 6. Restrict the scope of the update. (Same as Step 4 of the Update Algorithm Version I) For each datom f in σ_H, let Σ be the set of all most general substitutions σ under which f unifies with an atom of ω. Add the wff

$$(f \leftrightarrow H(f,U)) \vee ((\phi)_{\sigma_H} \wedge \bigvee_{\sigma \in \Sigma} \sigma) \tag{2}$$

to $\mathcal{T}'$. (Intuitively, formula (2) says that for f an atom that might possibly have its truth valuation changed by update U, the truth valuation of f can change only in a model where ϕ was true originally, and further that in any model so created, f must be unified with an atom of ω.) $\Diamond$

Example. For an example of the operation of Step 1, let U be the update `INSERT` $\neg$Emp(Reid, ϵ) `WHERE` T, when the body of $\mathcal{T}$ contains just the wffs Emp(Reid, EE) and Emp(Reid, CSD)$\vee$ Mgr(Nilsson, CSD), and the strict axiom section of $\mathcal{T}$ contains just the single–department axiom, $\forall x \forall y \forall z((\text{Emp}(x,y) \wedge \text{Emp}(x,z)) \rightarrow (y = z))$. This axiom satisfies condition 4 for strict enforceability. The Update Algorithm Version I would produce an alternative world in which

Reid is in CSD, which is incorrect if α is to be enforced strictly. As will be shown, Step 1 of Version IV prevents this anomaly. The two alternative worlds that should be produced under strict enforcement have the sets of true datoms {Mgr(Nilsson, CSD), Emp(Reid, EE)} and {Mgr(Nilsson, CSD)}, respectively.

Step 1a ensures that some datom in the instantiation of α being created will unify with a datom of ω. If this were not the case, no datom in that instantiation of α would be affected at all by the update, and the update could not possibly cause a rescue of any alternative world that violated that particular instantiation of α. For Step 1a of the current example, first let f be Emp(x, y) of α, which unifies with Emp(Reid, ϵ) in ω. Then σ is $\frac{x}{Reid}\ \frac{y}{\epsilon}$, and $(\alpha)_\sigma$ is $\forall z((\text{Emp(Reid, }\epsilon) \wedge \text{Emp(Reid, } z)) \rightarrow(\epsilon{=}z))$.

One option for $(\alpha)_\sigma$ at this point is not to include any additional variable substitutions in σ. Intuitively, this corresponds to instantiating those variables with universe elements not named in $\mathcal{L}$. For the current example, however, if no further variable substitutions are done, $(\alpha)_\sigma$ will be eliminated from Inst(α) in Step 1e, because α satisfies condition 4 and hence that instantiation of α would be valid. Whenever $(\alpha)_\sigma$ is valid, every possible alternative world satisfies $(\alpha)_\sigma$, and there is no way that U can cause a rescue of an alternative world violating $(\alpha)_\sigma$, as there is no such world.

Another option for $(\alpha)_\sigma$ is to add no variable substitutions to σ in Step 1b and then to replace z by ϵ in Step 1c. But the resulting wff $(\alpha)_\sigma$, ((Emp(Reid, ϵ) $\wedge$ Emp(Reid, ϵ)) $\rightarrow(\epsilon{=}\epsilon)$, is also valid, and hence may be replaced by T in Step 1g. The wff T is a particularly uninteresting axiom instantiation.

The final option for $(\alpha)_\sigma$ is for a variable replacement to take place in Step 1b. The only variable in $(\alpha)_\sigma$ at that point is z; the only atomic formula containing z in $(\alpha)_\sigma$ is Emp(Reid, z); therefore g of Step 1b must be Emp(Reid, z). There are two possible choices for f in $\mathcal{T}$: Emp(Reid, EE) or Emp(Reid, CSD). The former choice gives a σ of $\frac{x}{Reid}\ \frac{y}{\epsilon}\ \frac{z}{EE}$, and $(\alpha)_\sigma$ of ((Emp(Reid, ϵ) $\wedge$ Emp(Reid, EE)) $\rightarrow(\epsilon{=}$EE); the latter choice leads to a σ of $\frac{x}{Reid}\ \frac{y}{\epsilon}\ \frac{z}{CSD}$, and $(\alpha)_\sigma$ of ((Emp(Reid, ϵ) $\wedge$ Emp(Reid, CSD)) $\rightarrow(\epsilon{=}$CSD).

As no variables remain in either version of $(\alpha)_\sigma$, Steps 1c through 1f do not change these two wffs. In Step 1g, no simplifications look promising, so the two wffs can remain unchanged. Therefore Inst(α) contains the two wffs ((Emp(Reid, ϵ) $\wedge$ Emp(Reid, EE)) $\rightarrow(\epsilon{=}$EE) and ((Emp(Reid, ϵ) $\wedge$ Emp(Reid, CSD)) $\rightarrow(\epsilon{=}$CSD).

Returning to Step 1a for the next iteration, the only other choice for g in α is Emp(x, z), and the reader can quickly verify that this choice of g generates the same two wffs as did Emp(x, y). Therefore Inst(α) remains unchanged.

How will the two wffs of Inst(α) prevent the rescue of those undesirable alternative worlds? At the end of Step 6 of the Update Algorithm, the body of $\mathcal{T}'$ will contain the following wffs:

1. H(Emp(Reid, EE), U)

—Original body plus history substitution

2. H(Emp(Reid, CSD),U) $\vee$ Mgr(Nilsson, CSD)
   —Original body plus history substitution

3. H(Emp(Reid, ϵ), U) $\rightarrow$((ϵ =CSD) $\vee$ (ϵ =EE))
   —From Step 2, plus history substitution

4. ((H(Emp(Reid, ϵ), U) $\wedge$ H(Emp(Reid, EE), U)) $\rightarrow$(ϵ=EE)
   —From Inst(α), plus history substitution

5. ((H(Emp(Reid, ϵ), U) $\wedge$ H(Emp(Reid, CSD), U)) $\rightarrow$(ϵ=CSD)
   —From Inst(α), plus history substitution

6. T$\rightarrow\neg$Emp(Reid, ϵ)
   —From Step 5, $(\phi)_{\sigma_H} \rightarrow \omega$

7. (Emp(Reid, ϵ) $\leftrightarrow H$(Emp(Reid, ϵ))) $\vee$ (T$\wedge$T)
   —From Step 6, formula (2)

8. (Emp(Reid, EE) $\leftrightarrow H$(Emp(Reid, EE))) $\vee$ (T$\wedge$ (ϵ=EE))
   —From Step 6, formula (2)

9. (Emp(Reid, CSD) $\leftrightarrow H$(Emp(Reid, CSD))) $\vee$ (T$\wedge$ (ϵ=CSD))
   —From Step 6, formula (2)

It is not obvious that this theory has the correct alternative worlds, but a little dedicated cranking will grind them out. We do so here to show that α has been enforced, that is, that Reid is not in CSD in any model of this theory.

By the completion axioms, the only possible true datoms are Emp(Reid, EE), Emp(Reid, CSD), Mgr(Nilsson, CSD), and Emp(Reid, ϵ). The last of these four is ruled out immediately by wff #6.

For the remaining three datoms, first assume ϵ=EE. Then #9 implies $\neg$Emp(Reid, CSD). #6 implies $\neg$Emp(Reid, EE). #1 implies H(Emp(Reid, ϵ)), and with #5 this implies $\neg H$(Emp(Reid, CSD)), so by #2 Mgr(Nilsson, CSD) must be true.

Now assume $\epsilon \neq$EE. Then #1 and #8 imply Emp(Reid, EE), which with α implies $\neg$Emp(Reid, CSD). It remains to show Mgr(Nilsson, CSD). If $\epsilon \neq$CSD then #9 implies $\neg H$(Emp(Reid, CSD)), which with #2 implies Mgr(Nilsson, CSD). If ϵ=CSD, then #1 and #4 imply $\neg H$(Emp(Reid, ϵ)). This last and #2 imply Mgr(Nilsson, CSD). We conclude that T' has the correct alternative worlds.

Example. Let α be $\forall x$ (Emp(Reid, English) $\rightarrow$Emp(Reid, x)), and let $\mathcal{L}$ be infinite. This axiom satisfies condition 2 for strict enforceability but not condition 4. If the body of $\mathcal{T}$ contains just Emp(Reid, English), then $\mathcal{T}$ has no models, by the completion axioms. If the incoming update is `INSERT` $\neg$Emp(Reid, English) `WHERE` T, the Update Algorithm Version I would rescue the alternative world where all datoms are false. To enforce α strictly, the Update Algorithm Version IV must produce a theory with no models.

In Step 1a, if any variable substitutions are put into σ then $(\alpha)_\sigma$ is Emp(Reid, English) $\rightarrow$Emp(Reid, English), which is valid and therefore will not be helpful in preventing rescues. Since Emp(Reid, English) of α already is a subformula of ω, however, σ can remain the empty substitution.

In Step 1b, again any substitution for variables will make $(\alpha)_\sigma$ valid. Steps 1c and 1d are inapplicable, since α does not contain the equality predicate. Step 1e is also inapplicable.

In Step 1f, Emp(Reid, x) is replaced by the truth value F, so that β is Emp(Reid, English) $\rightarrow$F. Note that this wff was generated *without considering the contents of the body of* $\mathcal{T}$ *at all;* rather, it was generated using the fact that since the universe was infinite while the body of $\mathcal{T}$ was finite and ground, it followed that Emp(Reid, x) *had to be false* for some x in the universe. This is the key technique that keeps the set of needed instantiations finite for α when $\mathcal{L}$ contains an infinite set of constants.

In Step 1g, β can be simplified to $\neg$Emp(Reid, English). As $\mathcal{T}$ contains Emp(Reid, English), at the end of Step 6 $\mathcal{T}'$ will contain both H(Emp(Reid, English)) and $\neg H$(Emp(Reid, English)), and hence will be unsatisfiable, as desired.

We now describe the function Falsifiable(β, $\mathcal{T}$). This function takes as input a universal formula β and an extended relational theory $\mathcal{T}$ containing a domain completion axiom. The function forms the set S containing all non-ground atomic formulas of β, and then determines whether there exists a binding b for all the variables in S such that (1) no datom in $(S)_b$ unifies with a datom of $\mathcal{T}$ and (2) all equality atoms in $(S)_b$ are unsatisfiable.

More formally, let x_1 through x_n be the variables of β, and let c_1 through c_m be the constants in the domain completion axiom of $\mathcal{T}$. Then Falsifiable(β, $\mathcal{T}$) returns the value 'true' iff the following formula is satisfiable:

$$\exists x_1 \cdots \exists x_n \Big(\bigwedge_{1 \le i,j \le n} (x_i \neq x_j) \wedge \bigwedge_{1 \le i \le n} ((x_i = c_1) \vee \cdots \vee (x_i = c_m)) \wedge \bigwedge_{\substack{f \in S \\ g \in \mathcal{T} \\ f \sim \sigma g}} \neg \mathrm{red}(\sigma) \Big),$$

where red(σ) is formed from σ by replacing all equality atoms containing Skolem constants by the truth value T.

Falsifiable() checks for satisfiability by generating and testing bindings for the x_i variables. The name "Falsifiable" derives from the fact that the function is trying to ensure that the *non-ground* atomic formulas of β are falsifiable, that is, are false in some model of $\mathcal{T}$.

We do not specify the exact technique used to compute Falsifiable(β, $\mathcal{T}$); some heuristics to guide the generation process (e.g., a list of unlikely attribute values, such as "Nilsson" as an unlikely department name and "CSD" as an unlikely employee name) and backtracking should be entirely satisfactory in practice, since datoms may reasonably be expected to be quite sparse in the cross product of the domain. The worst–case time complexity of Falsifiable() is $\mathcal{O}(cm^n)$, where c is the number of occurrences of constants in $\mathcal{T}$.

Theorem 6-1. For an extended relational theory $\mathcal{T}$ with strict axioms and a ground update U, the Update Algorithm Version IV accomplishes U. $\Diamond$

Proof of Theorem 6-1. As usual, let $\mathcal{T}_1$ be the theory produced by Step 1, and so on. First, by Lemma 4-1, Steps 1 and 2 do not change the models of $\mathcal{T}$; so $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}$ all have the same models. Lemma 6-1 below shows that Inst(α) is finite, so the body of $\mathcal{T}_3$ is finite and $\mathcal{T}_3$ is an extended relational theory. Lemma 6-1 also shows that $\mathcal{T}_2$ and $\mathcal{T}_3$ have the same models. From the proof of Theorem 4-1 it follows that Version IV is correct and complete with respect to the definition of INSERT, except for two possible aberrations: Given that a model $\mathcal{M}_6$ is produced by Version IV, we must verify that $\mathcal{M}_6$ satisfies the strict axioms of $\mathcal{T}$; and also verify that World($\mathcal{M}_6$) $\in$ Worlds($U(\mathcal{M})$), for $\mathcal{M}$ a model of $\mathcal{T}$. The former of these follows immediately, as $\mathcal{T}_6$ contains the same type and dependency axioms as did $\mathcal{T}$. It remains to verify that $\mathcal{M}_6$ is descended from a model $\mathcal{M}$ of $\mathcal{T}$.

From the proof of Theorem 4-1 and the fact that $\mathcal{M}_6$ must satisfy the strict axioms of $\mathcal{T}_6$, it is immediate that World($\mathcal{M}_6$) $\in$ Worlds($U(\mathcal{M})$) for some model $\mathcal{M}$ of $\mathcal{T}$ *minus the strict axioms*. To review the construction of $\mathcal{M}$, let σ_6 be the Skolem constant substitution for $\mathcal{M}_6$ with respect to $\mathcal{T}_6$. Let $\mathcal{M}$ be a model that differs from $\mathcal{M}_6$ in only the following respects: if f is a datom of σ_H, then the truth valuation of f in $\mathcal{M}$ is the same as the truth valuation of $H(f, U)$ in $\mathcal{M}_6$.

If $\mathcal{M}$ violates α, then for some binding b of universe elements to all the variables of α, $(\alpha)_b$ is false in $\mathcal{M}$. Suppose that we are able to show that whenever $(\alpha)_b$ is false in a model, then some wff β of Inst(α) is also false there. But β is a formula of $\mathcal{T}_3$, and $\mathcal{M}$ is a model of $\mathcal{T}_3$, by the proof of Theorem 4-1; therefore $\mathcal{M}$ must be a model of $(\alpha)_b$ and of α. The remainder of this proof is a construction of a wff β such that $\beta \in$ Inst(α) and β is false in $\mathcal{M}$ if $(\alpha)_b$ is false in $\mathcal{M}$.

The wff $((\alpha)_{\sigma_6})_b$ must contain at least one datom that is a subformula of $(\omega)_{\sigma_6}$, because otherwise $(\alpha)_b$ and α would be false in $\mathcal{M}_6$. (This implies that α does not satisfy condition 1 for strict enforceability.) We will use this datom

as the starting point for the construction of β. Choose an atomic formula g of α such that $((g)_{\sigma_\epsilon})_b$ is a datom of $(\omega)_{\sigma_\epsilon}$. Let f be a datom of ω such that $(f)_{\sigma_\epsilon}$ is the same datom as $((g)_{\sigma_\epsilon})_b$. Then there exists a most general substitution σ' under which f and g unify. Let σ contain just the substitutions for variables in σ'; then $(\alpha)_\sigma$ is false in $\mathcal{M}$, $(((\alpha)_\sigma)_{\sigma_\epsilon})_b$ is $(\alpha)_b$, and $(\alpha)_\sigma$ was produced by the Update Algorithm Step 1a.

Repeat the following as many times as possible:

Choose a non-ground atomic formula g of $(\alpha)_\sigma$ such that $(g)_b$ unifies with a datom f of $\mathcal{T}$. Let σ' be a most general substitution under which f and g unify, and append the substitutions for variables in σ' to σ. Then $(\alpha)_\sigma$ is false in $\mathcal{M}$, $(((\alpha)_\sigma)_{\sigma_\epsilon})_b$ is identical to $((\alpha)_{\sigma_\epsilon})_b$, and $(\alpha)_\sigma$ was produced by the Update Algorithm Step 1b.

Now let $x = y$ be a non-ground equality atomic subformula of $(\alpha)_\sigma$, for x and y distinct and, say, x a variable. If $(x = y)_b$ is true in $\mathcal{M}$, then append to σ the substitution $\frac{x}{y}$. Repeat for all such subformulas of $(\alpha)_\sigma$, and then $(\alpha)_\sigma$ is still false in $\mathcal{M}$, $(((\alpha)_\sigma)_{\sigma_\epsilon})_b$ is identical to $((\alpha)_{\sigma_\epsilon})_b$, and $(\alpha)_\sigma$ was produced by the Update Algorithm Step 1c.

Let β be the wff $(\alpha)_\sigma$. Replace all occurrences of the reflexive equality formula in β (e.g., $x = x$) by the truth value T. Then $(\beta)_b$ is false in $\mathcal{M}$, and β was produced by the Update Algorithm Step 1d.

If β is not ground, then possibly β will be removed from Inst(α) in Step 1e of the Update Algorithm. This will occur if α satisfies condition 4 for strict enforceability. But if α satisfies this condition, then by the construction of β, $(\alpha)_b$ must be a valid wff, and hence $\mathcal{M}$ must satisfy $(\alpha)_b$, a contradiction.

β will also be removed from Inst(α) in Step 1e if α satisfies condition 3 for enforceability and Falsifiable(β, $\mathcal{T}$) is false. But in this case, there must be a non-ground atomic formula g of β such that $(g)_b$ unifies with a datom of $\mathcal{T}$. But by the construction of β, there cannot be any such atomic formula remaining in β at this point.

In Step 1f, any remaining non-ground equality atomic formulas of β are replaced by F in β. But by the construction of β, any such atomic formula is false in $\mathcal{M}$ under binding b, and hence the new version of $(\beta)_b$ is still false in $\mathcal{M}$.

In Step 1f, all remaining non-ground atomic formulas of β over data predicates are also replaced by F in β. But by the construction of β, again any such atomic formula is false in $\mathcal{M}$ under binding b, and hence the new version of β is false in $\mathcal{M}$.

After Step 1f, β is a ground wff that is false in $\mathcal{M}$. By the completion axioms, one may replace by F any datom of β that does not unify with a datom of $\mathcal{T}$, and β will still be false in $\mathcal{M}$. Then by Step 1g, β or a wff logically equivalent to β is in Inst(α). This concludes the proof of correctness for the Update Algorithm Version IV. $\Diamond$

Lemma 6-1 shows that Step 3 does not change the models of $\mathcal{T}$.

Lemma 6-1. For an extended relational theory $\mathcal{T}$ and an update U, let β be a wff in Inst(α), produced by Step 1 of the Update Algorithm Version IV. Then adding β to the body of $\mathcal{T}$ does not change the models of $\mathcal{T}$. Further, Inst(α) contains a finite number of wffs. $\Diamond$

Proof of Lemma 6-1. If the strict axiom α satisfies condition 1 for strict enforceability, then Inst(α) is the empty set, and the theorem is satisfied. Otherwise, to show that Inst(α) is finite, recall that the body of $\mathcal{T}$ is finite and so is ω. Therefore there can be only a finite number of different unifications of atomic formulas of α with datoms of $\mathcal{T}$ and ω, and Inst(α) must be finite. The remainder of the proof shows that all models of $\mathcal{T}$ satisfy Inst(α).

Since all models of $\mathcal{T}$ satisfy α, all models also satisfy $(\alpha)_\sigma$, for any substitution σ constructed in Step 1. Let β be the wff $(\alpha)_\sigma$, and assume that β is not discarded in Step 1e. Then if α satisfies condition 4 for strict enforceability, β is a ground formula at this point.

If the strict axiom α satisfies condition 2 but not condition 4, then β may still contain variables after Step 1e, and every non-ground atomic formula of β will be replaced by F in Step 1f, creating β'. To show that every model $\mathcal{M}$ of $\mathcal{T}$ satisfies β', let c_1 through c_n be constants of $\mathcal{L}$ such that no datom with c_i as an argument is true in $\mathcal{M}$. There must be such constants, because $\mathcal{L}$ contains an infinite number of constants. Then substitute c_i for x_i, for every variable x_i remaining in β, creating the wff $(\beta)_c$. Then $\mathcal{M}$ satisfies $(\beta)_c$, and for every non-ground atomic formula g of β, $(g)_c$ is false in $\mathcal{M}$. Therefore β' is also true in $\mathcal{M}$.

If the strict axiom α satisfies condition 3 but not condition 4, then β may still contain variables after Step 1e, and every non-ground atomic formula of β will be replaced by F in Step 1f, creating β'. In this case Falsifiable(β, $\mathcal{T}$) must be true, meaning that there exists a constant substitution b for the remaining variables of β such that for every non-ground atomic formula g of β, $(g)_b$ is false in $\mathcal{M}$. Therefore β' is true in every model of $\mathcal{T}$.

As for Step 1g, by the completion axioms any datom of β that does not unify with any datom in $\mathcal{T}$ must be false in all models of $\mathcal{T}$. Therefore if we replace such a datom or datoms in β by F, the resulting wff will also be true in all models of $\mathcal{T}$, and adding it to $\mathcal{T}$ will not change the models of $\mathcal{T}$.

Finally, any wff logically equivalent to β will also be true in all models of $\mathcal{T}$, and hence adding such a wff to $\mathcal{T}$ will not change the models of $\mathcal{T}$. $\Diamond$

6.4. Computational Complexity Revisited

Step 3 of the Update Algorithm Version IV does increase the size of $\mathcal{T}$. The amount of the increase depends on the type of axiom being strictly enforced and on the wffs currently in the body of $\mathcal{T}$. Since the dependency axioms for the database will be known ahead of time and will change only rarely if at all, the axioms can be preanalyzed and specialized enforcement routines prepared. We

begin by showing the computational complexity for functional and multi-valued dependencies. Let k be the number of different datoms in ω, and let R be the maximum number of datoms in $\mathcal{T}$ over the same data predicate.

If the dependency axiom is a functional dependency, then the size of Inst(α) is $\mathcal{O}(kR)$ worst case (when a datom that seems to conflict with every other atom over the same predicate is "deleted") and size zero best case (when no potential conflicts occur). The time to construct Inst(α) is $\mathcal{O}(kR\log(R))$ worst case, assuming that every atom over every predicate occurring in ω is located through an index in time $\mathcal{O}(\log(R))$; and $\mathcal{O}(k\log(R))$ best case.

These worst–case estimates are high indeed; when extended to updates containing variables, the potential cost becomes most alarming. But just how many conflicts are likely to occur in practice? More revealing than worst–case scenarios is the *average* size of Inst(α). In an actual database application, there are a number of mitigating factors that lead one to expect that Inst(α) will be quite small on average. We now examine these factors individually.

First, practical updates and queries do contain variables. Typically, a query references many tuples in order to provide summary information to a user. In updates, however, we argue that variables play a different role. The typical update in a real database does *not* modify multiple tuples. It selects one tuple, and changes just that one tuple. Variables in such an update play the role of placeholders for "don't–care" values while a selection is being done on a key, and do not lead to large numbers of database modifications per update request. This profile should to carry over to extended relational theories.

Second, a new datom in ω may conflict with every preexisting datom over its predicate, but just how likely is that situation? How often are there going to be many datoms that unify on the "ruling part" of their attributes? In any useful collection of data, those unifications should be few, as datoms will be sparse within the cross product of their domains. For example, when inserting the Emp(Reid, CSD) datom, one would not expect there to be large numbers of unidentified employees, nor for Reid to be rumored to be a member of many different departments.

Third, functional dependencies are typically used to identify keys. One very reasonable restriction for a practical database management system is to forbid null values to occur in keys. This action would drastically reduce the number of potential conflicts.

In sum, axiom instantiation should not be dismissed as a technique for strict axiom enforcement on the basis of its worst–case behavior. There are good reasons for expecting satisfactory performance of this technique in real–life situations.

Returning to the investigation of the computational complexity of particular classes of strict axioms, if the dependency axiom is a predicate–inclusion dependency (e.g., $\forall x \forall y$ (Mgr(x, y) $\rightarrow$ Emp(x, y))), then the time complexity is again is $\mathcal{O}(kR\log(R))$ worst case (when "deleting" a datom that seemed to invalidate every atom over some other predicate) and $\mathcal{O}(k\log(R))$ best case (when

no potential conflicts occur). Size complexity is the same as for functional dependencies. The same cost functions hold for a multivalued dependency as well.

The space and time bounds given above for enforcing α are precise because α has been pinned down to a particular class of axioms, such as functional dependencies. It is not possible to give a meaningful space bound for Inst(α) without such information on the format of α, the occurrences of equality predicates in α, and the patterns of occurrences of variables within the atomic formulas of α. For a size bound computed without this information, one must assume that every atomic formula in α unifies with everything in $\mathcal{T}$ at each stage of the reduction, a ridiculous assumption that leads to very high estimated bounds. For this reason we choose not to include a general worst–case size bound for Inst(α).

As usual in these algorithms, time complexity to construct Inst(α) differs from the size of Inst(α) by a $\mathcal{O}(\log(R))$ factor, representing the time needed to look up the atoms in indices. There is an additional multiplicative factor for strict axioms that satisfy only condition 3 for enforceability: the time complexity of Falsifiable(). As discussed earlier, the worst–case behavior of Falsifiable() is linear in the size of $\mathcal{T}$ and polynomial in the size of the underlying domain. However, it seems reasonable to assume that in practice a call to Falsifiable() can be completed in constant time.

6.5. Strict Enforceability Revisited

The previous section explained why certain potential strict axioms α were not considered strictly enforceable. In particular, it may be difficult to determine which wffs are in Inst(α) for a particular model $\mathcal{M}$, and in addition, different models may have different sets of formulas in Inst(α). Actually, any sentence α *can* be strictly enforced; it just may be an unpleasant proposition to do so.

If α does not meet conditions 1–4 for strict enforceability, then $\mathcal{L}$ has a finite set of constants. $\mathcal{T}$, however, may have models with universes of different sizes, both finite and infinite. To enforce α strictly will require a pastiche of the techniques for finite universes (condition 3) and infinite universes (condition 2). Suppose that $\mathcal{L}$ contains n constants, c_1 through c_n. Then in the final set Inst(α) that is added to the body of $\mathcal{T}$:

- Include all the wffs β that would be included in Inst(α) if $\mathcal{T}$ contained a domain closure axiom for c_1 through c_n. Such βs are true in all models of $\mathcal{T}$.
- Find every *additional* wff β that would be included in Inst(α) if $\mathcal{L}$ contained one additional constant c_{n+1} and $\mathcal{T}$ contained a domain closure axiom for c_1 through c_{n+1}. Then include in Inst(α) the formula $((\epsilon_1 \neq c_1) \wedge \cdots \wedge (\epsilon_1 \neq c_n)) \rightarrow \beta$.
- ...
- Find every *additional* wff β that would be included in Inst(α) if $\mathcal{L}$ contained k additional constants, c_{n+1} through c_{n+k}, and $\mathcal{T}$ contained a domain closure axiom for c_1 through c_{n+k}. Then include in Inst(α) the

formula $((\epsilon_1 \neq c_1) \wedge \cdots \wedge (\epsilon_1 \neq c_n) \wedge \cdots \wedge (\epsilon_k \neq c_1) \wedge \cdots \wedge (\epsilon_k \neq c_n) \wedge (\epsilon_k \neq \epsilon_1) \wedge \cdots \wedge (\epsilon_k \neq \epsilon_{k-1})) \rightarrow \beta$.

- Stop adding formulas to Inst(α) when:
  - If $\mathcal{T}$ does not contain a domain completion axiom, then when every β has been generated that would be included in Inst(α) if $\mathcal{L}$ contained an infinite number of constants;
  - If $\mathcal{T}$ contains a domain completion axiom containing j Skolem constants, then stop after the iteration where $k = j$.

Intuitively, those additional elements in the universe are needed in order to be able to guarantee the ability to falsify a subset of the datoms of β. Note the combinatorial explosion in the length of members of Inst(α) as k grows large.

Consider an example where α is $\forall x\ (R(a) \rightarrow R(x))$. If U is `INSERT` $\neg R(a)$ `WHERE T`, $\mathcal{L}$ contains the single constant a, and the body of $\mathcal{T}$ contains just $R(a)$, then in every model of $\mathcal{T}$, a must be the only element in the universe. After U is executed, a should still be the only element in the universe; yet the Update Algorithm Version I would rescue all the alternative worlds with more than one element. If $\mathcal{T}$ contained the domain completion axiom $\forall x(x = a)$, then Step 1 of Version IV would produce the set of formulas Inst(α) = $\{R(a) \rightarrow R(a)\}$. If $\mathcal{T}$ contained one additional constant, then Version IV would produce Inst(α) = $\{R(a) \rightarrow \mathsf{F}\}$. Adding a filter to the latter set and merging the two sets produces Inst(α) = $\{(\epsilon \neq a) \rightarrow \neg R(a)\}$. Note that adding $\neg R(a)$ to $\mathcal{T}$ would make $\mathcal{T}$ inconsistent; but adding $(\epsilon \neq a) \rightarrow \neg R(a)$does not change the models of $\mathcal{T}$ at all, and in fact does strictly enforce α and prevent models with universes containing more than just a from being rescued by U. Lest this example look too attractive, note that it was made tractable by restricting the language $\mathcal{L}$ to just one constant; a large set of constants would cause explosive growth in the number of formulas needed to enforce α.

To merge this technique into Version IV, process U as though α had a domain completion axiom containing all the constants of $\mathcal{L}$. At the time of the call to Falsifiable(β, $\mathcal{T}$) in Step 1e, do not discard β if the call fails. Instead, repeatedly add single constants to the domain completion axiom until Falsifiable(β, $\mathcal{T}$) succeeds, say on the kth call. Then replace β in Inst(α) by $((\epsilon_1 \neq c_1) \wedge \cdots \wedge (\epsilon_{k-1} \neq \epsilon_k)) \rightarrow \beta$, where ϵ_1 through ϵ_k do not appear in $\mathcal{T}$.

6.6. Summary and Conclusion

Type and dependency axioms can play many different roles in an extended relational theory. This chapter shows how to conduct two of those roles, passive and strict enforcement, and concentrates on the machinery necessary to enforce universally quantified axioms strictly, i.e., to eliminate permanently all alternative worlds that fail to satisfy that axiom. If universally quantified variables are permitted to occur in the body of $\mathcal{T}$, then this is trivial to accomplish. If the body of $\mathcal{T}$ is restricted to ground wffs, then the technique used is to try to instantiate

the axiom with constants and Skolem constants from $\mathcal{T}$, and to add those instantiations to the body of $\mathcal{T}$. Through logical manipulations, the required set of instantiations can always be made finite. However, it may be difficult to make the set *small*; some combinations of axioms α and languages $\mathcal{L}$ are just too expensive for strict enforcement when variables are not allowed in the body of $\mathcal{T}$. However, α will always be strictly enforceable if $\mathcal{L}$ contains an infinite set of constants, if the universe is known to be finite, or if, roughly speaking, for any variable x in α, replacing all atomic formulas of α containing x by the truth value F yields a valid formula. For example, all functional and multi–valued dependencies are strictly enforceable, and the cost of enforcing them during an update is quite reasonable.

Chapter 7: Other Semantics

Chapter 3 presented a semantics for updates (hereafter called the "standard semantics") along with a set of desiderata providing a rationale for the selection of the standard semantics. However, the standard semantics is not the only possible choice: there is a spectrum of reasonable candidates for update semantics, and the standard semantics is just one possible choice. The purpose of this chapter is to define this spectrum of semantics, give a closer examination to a few of the points along the spectrum, and explain why some candidate semantics are more reasonable than others. In the process, the motivations behind the choice of the standard semantics will become clear. In addition, this chapter shows how to adapt the Update Algorithm to work with other choices of semantics; the basic technique is to change the formula of Step 4. To simplify the presentation, we will restrict attention to extended relational theories *without* strict axioms. We begin by providing a framework for evaluation of competing semantics.

7.1. Criteria for Choice of Semantics

In this discussion, we assume that two of the three main desiderata for semantics that were presented in Section 3.3 are still applicable. In particular:

- The alternative worlds of the updated extended relational theory must be the same as those obtained by applying the update separately to each original alternative world.
- An update cannot directly change the truth valuations of any datoms except those that unify with datoms of ω.

With regard to the latter point, it is acceptable for an update indirectly to cause changes in the truth valuations of other datoms. For example, if type axioms (Chapter 6) are being enforced through axiom modification, then the update `INSERT` Emp(Reid, CSD) `WHERE` T might cause the datoms Employee(Reid) and Department(CSD) to become true. Any such side effects, however, must be due to axiom enforcement policies and not to the intrinsic semantics of the update.

Within the set of semantics that meet these two requirements, we suggest four criteria for evaluation of candidate semantics:

- The semantics of the intended application.
- Computational tractability.

- Comprehensibility.
- Expressive power.

The following sections examine each of these criteria in turn, with special reference to the properties of the standard semantics.

The semantics of the intended application. Some applications inherently require a particular choice of semantics. Updates contain new information destined for a database or knowledge base; the source and intended use of that information may impart a particular interpretation to the new facts. For example, in a diagnosis application, one wishes to make as few changes as possible in a description of the correct functioning of a device in order to make the description conform to the actual observed behavior of the device. When the observed behavior is provided as incoming information in an update, the updated theory should not include *every* possible combination of conditions that would lead to the observed behavior, but rather only the minimal sets of combinations of such conditions. For example, if a patient is hot and flushed, the diagnostician should suggest an infection as the cause of the problem, and not include the combination of an infection and a broken leg, or an infection and a broken leg and a headache. It turns out that the standard semantics is not appropriate for diagnosis.

Computational tractability. A *computationally tractable* semantics is one for which an algorithm with a reasonable running time implements the semantics. For example, the standard semantics is computationally tractable, as was shown in the discussions of the computational complexity of the Update Algorithm.

Comprehensibility. The user must be able to look at an update and understand what it will do: the update must be *comprehensible.* Our suggested tool for measuring comprehensibility versus trickiness is update equivalence: Do two updates that look similar produce the same effect on an extended relational theory? If two updates look different, do they produce different effects? Chapter 8 is devoted to a discussion of update equivalence. As will be shown there, syntax plays a moderate role in the criteria for update equivalence under the standard semantics.

Expressive power. The semantics must have adequate *expressive power.* One must be able to express every type of update, every transition between sets of alternative worlds, that is needed for the application. For example, Theorem 7-1 below shows that the standard semantics can be used to move from the set of alternative worlds of any extended relational theory to the set of worlds of any other extended relational theory, with one restriction; and hence the standard semantics has satisfactory expressive power.

The restriction on movement between sets of alternative worlds under the standard semantics lies in Skolem constant mapping requirements; for example, one cannot move from the extended relational theory with body ϵ=Reid to the theory with empty body. In general, any desired change can be made in the models of an extended relational theory, except that restrictions on the ranges

of particular Skolem constants can never be revoked once those restrictions have entered the theory. This is not an important limitation, since we can always just rename the Skolem constants in the alternative worlds we would like to move to, and abandon the old Skolem constants. The following definition shows how to ignore selected Skolem constants in models.

Definition. Let $\mathcal{D}$ be a subset of the language $\mathcal{L}$, formed by removing a finite set of Skolem constants from $\mathcal{L}$. If $\mathcal{A} = \text{World}(\mathcal{M})$ for some model $\mathcal{M}$ over $\mathcal{L}$, then $\mathcal{A}$ *restricted to* $\mathcal{D}$ (written $\mathcal{A}|\mathcal{D}$) is formed from $\mathcal{A}$ by removing from $\mathcal{A}$ the Skolem constant maps for every Skolem constant not in $\mathcal{D}$. $\Diamond$

Theorem 7-1. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be extended relational theories over the same language, containing disjoint sets of Skolem constants,† such that $\mathcal{T}_1$ is consistent. Then under the standard semantics, there exists an update U such that $\text{Worlds}(U(\mathcal{T}_1))|\mathcal{D} = \text{Worlds}(\mathcal{T}_2)|\mathcal{D}$, where $\mathcal{D}$ is $\mathcal{L}$ minus the Skolem constants appearing in $\mathcal{T}_1$. $\Diamond$

(The proof of Theorem 7-1 is given just before the proof of Theorem 7-5, a similar theorem.)

Further, the application's commonly used transitions between sets of alternative worlds should not be overly difficult to express; for example, the standard semantics is not well suited for diagnostic applications for this reason as well.

7.2. Minimal–Change Semantics

An obvious alternative to the standard semantics is a semantics for updates where the third main desiderata that determined the standard semantics, namely:

> The information in ω is to represent the *most exact and most recent state of knowledge obtainable* about those datoms; and that information is to *override all previous information* about those datoms,

is replaced by the following:

> An alternative world of $\mathcal{T}$ where ϕ is true should be changed *as little as possible* to make ω true.

The meaning of "as little as possible" is subject to interpretation, as has been debated in other contexts [Todd 77, Bancilhon 81, Davidson 81, 84, Dayal 82, DeKleer 85, Fagin 83, 86, Ginsberg 85, Keller 82, 85, Lewis 73, Reiter 85, Weber 85] and will be discussed in Section 7.3. For the purposes of this section, "as little as possible" means that if one set of changes to an alternative world $\mathcal{A}$ is a proper subset of another set of changes, and both sets of changes will make ω true in the updated version of $\mathcal{A}$, then the larger set of changes should not be performed. More formally, let $\mathcal{M}$ be a model of an extended relational theory

† This restriction is needed to avoid naming conflicts; if it is violated, rename the Skolem constants of one of the theories before testing.

$\mathcal{T}$. Then under the minimal–change semantics, inserting a wff ω into $\mathcal{M}$ should produce the alternative worlds of every model $\mathcal{M}'$ with the same universe and constant and Skolem constant mappings as $\mathcal{M}$, such that

(1) if $\mathcal{M}'$ differs from $\mathcal{M}$ in the truth valuations of a set D of null–free datoms, then no other model $\mathcal{M}^*$ with the same universe and mappings as $\mathcal{M}'$ and where ω is true differs from $\mathcal{M}$ in only a proper subset of D; and

(2) ω is true in $\mathcal{M}'$.

A simplified version of this semantics is used by DeKleer [85] and Reiter [85]; they do not consider selection clauses (ϕ) or Skolem constants.

It follows from the minimal–change definition that two null–free updates `INSERT` ω_1 `WHERE` ϕ and `INSERT` ω_2 `WHERE` ϕ are equivalent under the minimal–change semantics if ω_1 and ω_2 are logically equivalent. Exact conditions for equivalence, however, are surprisingly complex (see Theorem 8-12), though the minimal–change semantics score is still satisfactory on this measure of comprehensibility. The effect of a minimal–change update is very strongly tied to the current models of the extended relational theory. A example will perhaps be revealing: what is the effect of inserting $(a \wedge \neg b) \vee (c \wedge d)$, where a, b, c, and d are datoms? The quick response: the alternative worlds will be changed as little as possible to get $(a \wedge \neg b) \vee (c \wedge d)$ to be true. But *exactly what* will be true afterwards? That depends on what the models are now; for any particular model it will take a bit of careful thought to determine what the update produces.

The minimal–change semantics suffers from lack of expressive power, as does any semantics where logically equivalent ωs lead to equivalent updates. For example, if the user wants to say that there is no longer any information about the truth or falsity of a particular datom g, this can be done by inserting $g \vee \neg g$ under the standard semantics. Under a semantics based on logical equivalence, however, such an insertion would be equivalent to inserting the truth value T, and would therefore have no effect on the alternative worlds of the extended relational theory. In general, if an update is needed to express a loss of knowledge—i.e., one formerly believed that some proposition was true but now one is unsure—the minimal–change semantics does not offer a mechanism to accomplish the change, since every model of the old extended relational theory already satisfies ω.

Theorem 7-2. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be consistent extended relational theories such that Models($\mathcal{T}_1$) is a proper subset of Models($\mathcal{T}_2$), and Worlds($\mathcal{T}_1$) $\neq$ Worlds($\mathcal{T}_2$). Then for no update U does Worlds($U(\mathcal{T}_1)$) = Worlds($\mathcal{T}_2$). $\Diamond$

Proof of Theorem 7-2. Suppose that U is such an update, and let $\mathcal{M}$ be a model of $\mathcal{T}_1$ and therefore also of $\mathcal{T}_2$. Suppose that U changes $\mathcal{M}$, that is, that $U(\mathcal{M}) \neq \{\mathcal{M}\}$. Then ω is false in $\mathcal{M}$. By the definition of the minimal–change semantics, it follows that World($\mathcal{M}$) $\notin$ Worlds($U(\mathcal{M})$). Since World($\mathcal{M}$) $\in$ Worlds($\mathcal{T}_2$), it follows that for some other model $\mathcal{M}'$ of $\mathcal{T}_1$, World($\mathcal{M}$) $\in$ Worlds($U(\mathcal{M}')$). But then ω is true in $\mathcal{M}$, a contradiction. We conclude that U

does not change $\mathcal{M}$. But $\mathcal{M}$ is an arbitrary model of $\mathcal{T}_1$; therefore Worlds($U(\mathcal{T}_1)$) = Worlds($\mathcal{T}_1$), so Worlds($U(\mathcal{T}_1)$) $\neq$ Worlds($\mathcal{T}_2$). $\Diamond$

If instead of a single update U, we allow a series of updates U_1 through U_n, Theorem 7-2 still holds if $\mathcal{L}$ contains a finite set of constants.† An example follows.

Example. Let $\mathcal{L}$ contain the single constant a and the single predicate R, and let the body of $\mathcal{T}_1$ be the formula $R(a)$. Then there is no way to move from $\mathcal{T}_1$ to the alternative worlds of the theory with body $R(a) \vee \neg R(a)$ under a series of updates. $\Diamond$

If $\mathcal{L}$ contains an infinite set of constants, it is possible to remove all knowledge about the truth valuation of a datom under the minimal–change semantics by eliminating the requirement that the transition be accomplished in a *single* update. This is because additional updates and constants can be used to do *encoding*, much as history atoms were used with lazy evaluation and update splitting in Chapter 5 to encode information for future updates. One cannot encode using just the equality predicate and Skolem constants, as such encoding makes permanent changes in alternative worlds; hence the need for an infinite set of constants. An example follows.

Example. Under the minimal–change semantics, one can remove all knowledge about the truth valuation of a datom a by the following series of four updates:

U_1: `INSERT` $f \vee g$ `WHERE` $\top$,

U_2: `INSERT` $\neg a$ `WHERE` g,

U_3: `INSERT` a `WHERE` f,

U_4: `INSERT` $\neg f \wedge \neg g$,

where f and g are datoms not unifying with a, with each other, or with any datom of $\mathcal{T}$. $\Diamond$

The use of ordinary datoms such as f and g for encoding, however, has a number of disadvantages. For correctness all the steps of the procedure, including the awkward and potentially expensive location of "unused" datoms for coding, must be bundled into a single transaction, which is not always convenient. More importantly, encoding runs counter to the spirit of our endeavor, as it is beyond the capabilities of average users, be they humans or programs. Finally, encoding

† This conflicts with the requirement of Section 3.1.1 that $\mathcal{L}$ contain an infinite number of constants. This requirement is present to insure that one will always be able to find a history atom $H(f, U)$ that does not unify with any history atom in the body of $\mathcal{T}$. If $\mathcal{L}$ contains only a finite set of constants, as is reasonable for many applications, then the whole system must come to a pause if the Update Algorithm runs out of "new" history atoms.

runs afoul of another requirement of the expressiveness criteria, namely, that common update requests be standard to express.

We conclude that under the minimal–change semantics, a separate operator is needed to accomplish updates reporting a loss of knowledge; such an operator could have the standard semantics or another expressive model–based semantics. Alternatively, one could define a minimal–change DELETE operator, such as that of Fagin et al [86]; however, it turns out that a minimal–change DELETE cannot really have a model–based semantics, and so must mark a considerable departure from all other semantics discussed in this thesis. Of course, yet some other hybrid approach is also possible. In any case, we conclude that a little syntactic element—name-dropping—can be useful in update semantics.

Interestingly, the Update Algorithm is sufficiently general to serve under the minimal–change semantics (and other choices of semantics as well) simply by altering the formula of Step 4. Recall that in Step 4 of the Update Algorithm using the standard semantics,

$$(f \leftrightarrow H(f,U)) \vee ((\phi)_{\sigma_H} \wedge \bigvee_{\sigma \in \Sigma} \sigma) \tag{2}$$

is added to $\mathcal{T}'$ for every datom f of $\mathcal{T}'$ that unifies with an atom of ω. If the current update is INSERT Emp(Reid, EE) ∨ Emp(Reid, CSD) WHERE T and there are no Skolem constants in $\mathcal{T}$, then this means that $\mathcal{T}'$ gets the two new wffs

$$(\text{Emp(Reid, EE)} \leftrightarrow H(\text{Emp(Reid, EE)}, U)) \vee (\mathsf{T} \wedge \mathsf{T})$$
$$(\text{Emp(Reid, CSD)} \leftrightarrow H(\text{Emp(Reid, CSD)}, U)) \vee (\mathsf{T} \wedge \mathsf{T}),$$

both of which are logically equivalent to T. To move to the minimal–change semantics, one needs to add the two wffs

$$(\text{Emp(Reid, EE)} \leftrightarrow H(\text{Emp(Reid, EE)}, U)) \vee$$
$$(\neg H(\text{Emp(Reid, EE)}, U) \wedge \neg H(\text{Emp(Reid, CSD)}, U) \wedge \neg \text{Emp(Reid, CSD)})$$
$$(\text{Emp(Reid, CSD)} \leftrightarrow H(\text{Emp(Reid, CSD)}, U)) \vee$$
$$(\neg H(\text{Emp(Reid, EE)}, U) \wedge \neg H(\text{Emp(Reid, CSD)}, U) \wedge \neg \text{Emp(Reid, EE)}).$$

These formulas say that the truth valuation of Emp(Reid, EE) can only change in an alternative world if Reid is in neither department before the update and Reid is not in CSD in that alternative world after the update. If Emp(Reid, EE) and/or Emp(Reid, CSD) are already true in an alternative world, then they remain true there; and otherwise, Reid is put into exactly one of those two departments.

In the general case, formula (2) needs to contain extra terms that are true exactly when f is part of a set of minimal changes that makes ω true in an alternative world. To capture those states, let v_ω be a truth valuation for all the atoms of ω.† Unfortunately, when Skolem constants occur in ω, knowing v_ω may

† A truth valuation v can be written in wff form as the conjunction of T and a set of literals, such that the atom a is a conjunct of v in wff form iff a receives the truth valuation T under v, and $\neg a$ is a conjunct of v in wff form iff a receives the truth valuation F under v.

not be sufficient to determine the minimal changes needed to make ω true in an alternative world $\mathcal{A}$ satisfying v_ω. In particular, if two datoms of ω unify with one another, additional information may be needed, as the following example shows.

Example. Consider the update INSERT $R(a) \vee (R(\epsilon) \wedge R(b))$ WHERE T, applied to a theory with empty body. If ϵ is not a, then one minimal change that will make ω true is to make both $R(\epsilon)$ and $R(b)$ true. If ϵ is a, however, then this is not a minimal change, as making just $R(\epsilon)$ true will satisfy ω. $\Diamond$

For this reason, one must know exactly which datoms in ω unify with one another in $\mathcal{A}$ before attempting to determine the minimal changes needed in $\mathcal{A}$. To that end, let PossUnif(ω) be the set containing T and all most general substitutions σ under which a datom of ω unifies with another datom of ω; and let Unif(ω) be any satisfiable truth valuation of all the atoms in PossUnif(ω).

Example. Again consider the update U: INSERT $R(a) \vee (R(\epsilon) \wedge R(b))$ WHERE T. PossUnif(ω) contains the formulas T, $\epsilon{=}b$, and $\epsilon{=}a$. The possible values for Unif(ω) are $\epsilon{=}b \wedge \epsilon{\neq}a$, $\epsilon{\neq}b \wedge \epsilon{=}a$, and $\epsilon{\neq}b \wedge \epsilon{\neq}a$. $\Diamond$

As explained above, $v_\omega \wedge$Unif(ω) includes all the information about an alternative world that is needed to compute the minimal sets of changes in that world that will make ω true there. We now need to identify those minimal sets. Given a satisfiable $v_\omega \wedge$Unif(ω), let Stable(v_ω, Unif(ω)) be a subset of the truth valuations of v_ω such that

(1) If v'_ω is created from v_ω by negating the truth valuations of all datoms of v_ω except members of Stable(v_ω, Unif(ω)), then ω is true under $v'_\omega \wedge$Unif(ω);

(2) No proper superset of Stable(v_ω, Unif(ω)) within v_ω has property (1).

Then Stable(v_ω, Unif(ω)) exactly characterizes a legal minimal-change transition for $\mathcal{A}$, by pinpointing a maximal subset of the datoms of ω that can retain their current truth valuations when ω is made true. Let StableSets(v_ω, Unif(ω)) be the set containing all choices of Stable(v_ω, Unif(ω)). A simple algorithm for finding StableSets(v_ω, Unif(ω)), called *full reduction*, is given in Section 8.4 and proven correct in Theorem 8-13.

Example. Again consider the update INSERT $R(a) \vee (R(\epsilon) \wedge R(b))$ WHERE T. There are eight possible values for v_ω; not all of them are satisfiable given Unif(ω). We show StableSets($\neg R(a) \wedge \neg R(b) \wedge \neg R(\epsilon)$, Unif($\omega$)) for each value of Unif(ω).

Unif(ω)	StableSets($\neg R(a) \wedge \neg R(b) \wedge \neg R(\epsilon)$, Unif($\omega$))
$\epsilon{=}b \wedge \epsilon{\neq}a$	$R(a)$, $R(b) \wedge R(\epsilon)$
$\epsilon{\neq}b \wedge \epsilon{=}a$	$R(b)$
$\epsilon{\neq}b \wedge \epsilon{\neq}a$	$R(a)$, $R(b) \wedge R(\epsilon)$

$\Diamond$

The set of all legal minimal–change transitions is given by the set S_Δ, which contains the wff $(v_\omega)_{\sigma_H} \wedge \text{Unif}(\omega) \wedge \text{Stable}(v_\omega, \text{Unif}(\omega))$, for all wffs v_ω and $\text{Unif}(\omega)$ such that $v_\omega \wedge \text{Unif}(\omega)$ is satisfiable, and for all $\text{Stable}(v_\omega, \text{Unif}(\omega)) \in \text{StableSets}(v_\omega, \text{Unif}(\omega))$. For technical reasons, S_Δ must contain the wff F as well. (Since the number of atoms in U is not related to the size of $\mathcal{T}$ and will be small, the computation of S_Δ will be feasible even though S_Δ may be of size exponential in the size of U.) Intuitively, each member of S_Δ represents a minimal–change transition from one alternative world (denoted by $(v_\omega)_{\sigma_H} \wedge \text{Unif}(\omega)$) to another under the minimal–change semantics.

We now incorporate S_Δ into a new version of Step 4 of the Update Algorithm Version I that implements the minimal–change semantics:

Step 4′. Restrict the scope of the update. For each datom f in σ_H, let Σ be the set of all most general substitutions σ under which f unifies with an atom of ω. Add the wff

$$(f \leftrightarrow H(f,U)) \vee \left((\phi)_{\sigma_H} \wedge \bigvee_{\sigma \in \Sigma} (\sigma \wedge \bigvee_{\substack{\tau \in S_\Delta \\ f \notin (\tau)_\sigma}} \tau)\right) \qquad (2')$$

to $\mathcal{T}'$. (By $f \notin (\tau)_\sigma$ we mean that f is not a subformula of $(\tau)_\sigma$.) Intuitively, for f an atom that might possibly have its truth valuation changed by update U, formula (2′) says that the truth valuation of f can change only in a model where ϕ was true originally, and further that in any model so created, f must be unified with an atom of ω, and must be part of a minimal change in that model to make ω true. ◇

Example. For the update `INSERT` Emp(Reid, EE)∨ Emp(Reid, CSD) `WHERE` T applied to a theory not containing Skolem constants, the two wffs added in Step 4′ are

$$\begin{gathered}(\text{Emp(Reid, EE)} \leftrightarrow H(\text{Emp(Reid, EE)}, \text{U}))\vee \\ (\text{T} \wedge (\text{T} \wedge\ (\text{F} \vee \text{F} \vee \text{F}\vee \\ (\neg H(\text{Emp(Reid, EE)}, \text{U}) \wedge \neg H(\text{Emp(Reid, CSD)}, \text{U}) \wedge \text{T} \wedge \neg \text{Emp(Reid, CSD)})))) \\ (\text{Emp(Reid, CSD)} \leftrightarrow H(\text{Emp(Reid, CSD)}, \text{U}))\vee \\ (\text{T} \wedge (\text{T} \wedge\ (\text{F} \vee \text{F} \vee \text{F}\vee \\ (\neg H(\text{Emp(Reid, EE)}, \text{U}) \wedge \neg H(\text{Emp(Reid, CSD)}, \text{U}) \wedge \text{T} \wedge \neg \text{Emp(Reid, EE)})))).\end{gathered}$$

Theorem 7-3. Given an extended relational theory $\mathcal{T}$ and an update U, the Update Algorithm Version I, with Step 4 replaced by 4′, correctly and completely performs U under the minimal–change semantics. ◇

Proof of Theorem 7-3. This new algorithm produces a set of alternative worlds that is a subset of those produced by the Update Algorithm Version I,

as formula (2') logically entails formula (2). Therefore the proof of Theorem 4-1 can be used to show that the new algorithm is correct and complete, with one exception: Let $\mathcal{M}_4$ be a model produced by the new algorithm. For some model $\mathcal{M}$ of $\mathcal{T}$ such that World($\mathcal{M}_4$) $\in$ Worlds($U(\mathcal{M})$), we must show that there is no model $\mathcal{M}_4'$ such that World($\mathcal{M}_4'$) $\in$ Worlds($U(\mathcal{M})$) and the differences in datom truth valuations between $\mathcal{M}_4'$ and $\mathcal{M}$ are a proper subset of the differences in datom truth valuations between $\mathcal{M}$ and $\mathcal{M}_4$.

Let $\mathcal{M}_4$ be a model of $\mathcal{T}_4$ where $(\phi)_{\sigma_H}$ is true; let σ_4 be the Skolem constant substitution for $\mathcal{M}_4$ with respect to $\mathcal{T}$ and U; and let $\mathcal{M}$ be defined exactly as in the proof of Theorem 4-1. Let g be a null-free datom on which $\mathcal{M}$ and $\mathcal{M}_4$ disagree in truth valuation. Then g is a subformula of $(\omega)_{\sigma_4}$. Let S be the set of datoms f in ω and $\mathcal{T}$ such that $(f)_{\sigma_4}$ is g. For any $f \in S$, by formula 2' there must be $\sigma \in \Sigma$ such that $\sigma \wedge ((v_\omega)_{\sigma_H} \wedge \text{Unif}(\omega) \wedge \text{Stable}(v_\omega, \text{Unif}(\omega)))$ is true in $\mathcal{M}_4$. It follows that $v_\omega \wedge \text{Unif}(\omega)$ is true in $\mathcal{M}$. But in any model where $v_\omega \wedge \text{Unif}(\omega)$ is true, Stable(v_ω, Unif(ω)) determines a minimal change in that model that will make ω true. Since Stable(v_ω, Unif(ω)) is true in $\mathcal{M}_4$, it follows that $\mathcal{M}_4$ does constitute a minimal change from $\mathcal{M}$ to make ω true. $\Diamond$

As defined, the set of formulas added to $\mathcal{T}'$ in Step 4' will always have size exponential in the number of atoms in U. More precisely, Step 4' adds $\mathcal{O}(nk^3 2^{\frac{1}{2}(k^2+k)} \binom{k}{k/2})$ occurrences of atoms to $\mathcal{T}'$ in the worst case, where k is the size of the update and n is the maximum number of datoms in $\mathcal{T}_3$ that unify with a datom of ω. In other words, the size increase is linear in the number of datoms of $\mathcal{T}$ that unify with datoms of ω, and exponential in the size of the update. Though a size increase in $\mathcal{T}$ that is exponential in the size of U is unavoidable in the worst case, this estimate is greatly exaggerated for the typical theory and update. For example, for simple ω formulas such as conjunctions and disjunctions of literals, a much smaller size increase—linear and quadratic, respectively, in the size of U—is possible for ground theories and updates, with commensurate savings when Skolem constants are present. Therefore, rather than applying the worst-case formulas blindly and adding their instantiations directly to $\mathcal{T}$, a heuristic minimization procedure—e.g., recognition of conjunctions and disjunctions, Karnaugh mapping—should be applied first to reduce the length of the formulas.

Because we cannot offer as efficient an algorithm for the minimal-change semantics as for the standard semantics, and because it will be more difficult to minimize the length of the formulas added to $\mathcal{T}$, we conclude that the extra effort required to implement minimal-change semantics is not worthwhile unless the semantics of the application call for a minimal-change semantics.

7.3. A Spectrum of Candidate Semantics

Before embarking on a further investigation of semantics, it will be helpful to point out that all these semantics fall into a broad spectrum ranging from the standard semantics on one end to variants of the minimal-change semantics on

the other. For example, the standard semantics has the most lenient rules about which alternative worlds fall into the result of an update; any way of making ω true will do. The truth–maintenance semantics (to be presented below) rules out a number of these models as containing unacceptable changes in truth valuations. The minimal–change semantics rules out even more of these models. One can easily imagine populating the hierarchy with ever more exotic choices of semantics.

For example, consider the minimal–change semantics. The definition we gave for this semantics said that in making ω true in an alternative world, if one set of changes to an alternative world $\mathcal{A}$ is a proper subset of another set of changes, then the latter set can be eliminated. An even more minimal change can be obtained by requiring the sets of changes to have the minimum possible cardinality, instead of just using the set/subset relationship. This variant is another point in the semantics spectrum.

As another variant, perhaps the definition of minimal–change semantics should take any strictly enforced axioms into consideration. The quickest route to satisfying ω, i.e., the minimal sets of changes in an alternative world that will make ω true, might produce only alternative worlds that violate the strict axioms.† A more generous set of changes might lead to a viable alternative world.

Another point in the semantics spectrum was introduced recently by Weber [86]. He proposes a system where the goal is to preserve as many truth valuations as possible in an alternative world when updating. However, if there is more than one minimal set of changes that will make ω true in an alternative world $\mathcal{A}$, then the truth valuations of *all* datoms in *any* minimal set are allowed to change. This is a most interesting choice, and one unforeseen by this author when postulating the existence of spectrum of reasonable semantics. Given a particular update U under Weber's semantics, the effect of U may vary greatly depending on the models of the extended relational theory. In fact, the effect can range from coinciding with the minimal–change semantics, if there are no conflicts in the minimal sets of changes that will make ω true in various alternative worlds; to coinciding with the standard semantics, when the current worlds of the theory are squarely at odds with the new information in ω.

7.4. The Truth–Maintenance Semantics

It will perhaps lend additional credence to the claim of existence of a spectrum of reasonable semantics, and also of the flexibility of the Update Algorithm, that the majority of the research in this thesis was carried out with a semantics other than the standard semantics in mind. The *truth-maintenance semantics* corresponds to the author's original intuitions about what updates should mean; she was only detached from this semantics after proving some very unpleasant theorems about update equivalence under the truth–maintenance semantics (see Chapter 8), and indeed it is on the score of comprehensibility that the truth–maintenance

† This problem does not arise with the standard semantics.

semantics must be found lacking. We begin with an intuitive justification for this semantics.

Suppose the user requests an insertion of $a \vee b$, where a and b are datoms. What effect should this update have on a model $\mathcal{M}$ where a is already true? In particular, should this update produce any model where a is false? The user did not mention $\neg a$; there are no negation signs anywhere in this update. Is there really any justification for making a false in any models produced by this update? Since a appears only positively in the update is it not correct to maintain the current truth valuation of a?

Definition. A null–free datom g *appears positively* (resp. *negatively*) in a null–free wff α having no connectives other than $\neg$, $\wedge$, and $\vee$, if g is a subformula of α and does not occur governed by an odd (resp. even) number of negation signs. $\Diamond$

For example, g does not appear positively or negatively in $g \vee \neg g$ or $f \vee \neg f$, and appears positively in $\neg(f \wedge \neg g)$. It is easy to prove that if ω is satisfiable, then it is satisfiable with a truth valuation in which any given subset of the datoms that appear positively in ω receive the truth valuation T, and any given subset of the datoms that appear negatively in ω receive the truth valuation F:

Proposition 7-1. Let α be a ground null–free wff with no connectives other than $\wedge$, $\vee$, and $\neg$. Let v be a truth valuation for the atoms of α such that α is true under v. If g is a datom in α that appears positively (resp. negatively) in α and is false (resp. true) under v, then there exists a truth valuation v' created from v by negating the truth valuation of g, such that α is satisfied under v'. $\Diamond$

Proof of Proposition 7-1. Put α into conjunctive normal form; this operation will not affect which datoms appear positively and negatively in α. If g appears positively in α, then changing v by making g true can only make additional conjuncts of α true, and α will still be satisfied. If g appears negatively in α, then changing v by making g false can only make additional conjuncts of α true, and α will still be satisfied. $\Diamond$

According to the intuitive justification given earlier, if g appears positively (resp. negatively) in ω and g is true (resp. false) in $\mathcal{M}$, then the truth valuation of g should not be changed by inserting ω into $\mathcal{M}$.

We now present formal definitions of the truth–maintenance semantics for updates. Let U be an update and let $\mathcal{M}$ be a model of an extended relational theory $\mathcal{T}$ with Skolem constant substitution σ with respect to ω. Then $U(\mathcal{M})$ contains just $\mathcal{M}$ if ϕ is false in $\mathcal{M}$. Otherwise, $U(\mathcal{M})$ contains exactly every model $\mathcal{M}'$ with the same universe and mappings as $\mathcal{M}$, such that

(1) $\mathcal{M}'$ agrees with $\mathcal{M}$ on the truth valuations of all null–free atoms except possibly those in $(\omega)_\sigma$; and

(1a) If a datom g appears positively in $(\omega)_\sigma$, then if g is true in $\mathcal{M}$, g is also true in $\mathcal{M}'$; and

(1b) If a datom g appears negatively in $(\omega)_\sigma$, then if g is false in $\mathcal{M}$, g is also false in $\mathcal{M}'$; and

(2) ω is true in $\mathcal{M}'$. $\Diamond$

As usual, $\text{Worlds}(\mathcal{T}') = \bigcup_{\mathcal{M}\in\text{Models}(\mathcal{T})} \text{Worlds}(U(\mathcal{M}))$.

Skolem constants complicate matters a bit. One cannot tell whether $R(a)$ appears positively in $R(a)\vee\neg R(\epsilon)$ until one knows whether ϵ is equal to a. In general, to decide whether a datom g appears positively or negatively in a wff α, one must specify exactly which of the possible unifications within α involving g are true. If v is a wff telling exactly which of these unifications are true, such as $\epsilon{=}a$ or $\epsilon{\neq}a$ in the current example, then one can determine whether g appears positively or negatively in α *given* v. For example, $R(a)$ appears positively in $R(a)\wedge\neg R(\epsilon)$ given $\epsilon{\neq}a$, and $R(a)$ appears neither positively nor negatively in $R(a)\wedge\neg R(\epsilon)$ given $\epsilon{=}a$.

As with the minimal–change semantics, the Update Algorithm Version I need be altered only slightly to implement the truth–maintenance semantics. The only change is in formula (2) in Step 4, which must be split into three separate cases:

Step 4″. Restrict the scope of the update. For each datom f in σ_H, let Σ be the set of all most general substitutions σ under which f unifies with an atom of ω. Let V_Σ be the set containing all satisfiable wffs v, where v is the conjunction containing a conjunct σ or $\neg\sigma$ for every substitution $\sigma\in\Sigma$. Add the wff

$$(f \leftrightarrow H(f,U)) \vee ((\phi)_{\sigma_H} \wedge \bigvee_{v\in V_\Sigma} (v \wedge \tau)) \qquad (2'')$$

to $\mathcal{T}'$, where τ is defined as

$$\tau = \begin{cases} H(f,U), & \text{if } f \text{ appears negatively in } \omega \text{ given } v; \\ \neg H(f,U), & \text{if } f \text{ appears positively in } \omega \text{ given } v; \\ \mathsf{T}, & \text{otherwise.} \quad \Diamond \end{cases}$$

Example. Let $\mathcal{T}$ be an extended relational theory not containing Skolem constants, and let U be `INSERT` $\neg$Emp(Reid, CSD)$\vee$Emp(Reid, ϵ) `WHERE` T. Then Step 4′ produces the two wffs

$$\begin{gathered} (\text{Emp(Reid, CSD)} \leftrightarrow H(\text{Emp(Reid, CSD)}, U))\vee \\ (\mathsf{T} \wedge ((\epsilon = \text{CSD} \wedge \mathsf{T}) \vee (\epsilon \neq \text{CSD} \wedge H(\text{Emp(Reid, CSD)}, U)))) \\ (\text{Emp(Reid, } \epsilon) \leftrightarrow H(\text{Emp(Reid, } \epsilon), U))\vee \\ (\mathsf{T} \wedge ((\epsilon = \text{CSD} \wedge \mathsf{T}) \vee (\epsilon \neq \text{CSD} \wedge \neg H(\text{Emp(Reid, } \epsilon), U)))).\Diamond \end{gathered}$$

Theorem 7-4. Given an extended relational theory $\mathcal{T}$ and an update U, the Update Algorithm Version I, with Step 4 replaced by 4″, accomplishes U under the truth–maintenance semantics. ◊

Proof of Theorem 7-4. This new algorithm produces a set of alternative worlds that is a subset of those produced by the Update Algorithm Version I, since formula (2″) logically entails formula (2). Therefore the proof of Theorem 4-1 can be used to show that the new algorithm is correct, with one exception: if $\mathcal{M}_4$ is a model produced by the new algorithm, we must show that $\mathcal{M}_4$ obeys rules 1a and 1b in the truth–maintenance semantics. Let $\mathcal{M}_4$ be a model of $\mathcal{T}_4$ where $(\phi)_{\sigma_H}$ is true; let σ_4 be the Skolem constant substitution for $\mathcal{M}$ with respect to $\mathcal{T}$ and U; and let $\mathcal{M}$ be defined exactly as in the proof of Theorem 4-1. If g is a datom that appears positively in $(\omega)_{\sigma_4}$, then g also appears positively in $((\omega)_\sigma)_{\sigma_4}$, for some $\sigma \in \Sigma$. Therefore by formula 2″, if g has the same truth valuation in $\mathcal{M}$ as in $\mathcal{M}_4$ then rules 1a and 1b are satisfied by $\mathcal{M}_4$. If g has a different truth valuation in $\mathcal{M}_4$ than in $\mathcal{M}$, then by formula 2″, $H(g, U)$ must be false in $\mathcal{M}_4$, and therefore g must be false in $\mathcal{M}$. In this case, g also satisfies rules 1a and 1b. Similarly, if g is a datom that appears negatively in $(\omega)_{\sigma_4}$, then g also appears negatively in $((\omega)_\sigma)_{\sigma_4}$, for some $\sigma \in \Sigma$. Therefore by formula 2″, if g has the same truth valuation in $\mathcal{M}$ as in $\mathcal{M}_4$ then rules 1a and 1b are satisfied by $\mathcal{M}_4$. If g has a different truth valuation in $\mathcal{M}_4$ than in $\mathcal{M}$, then by formula 2″, $H(g, U)$ must be true in $\mathcal{M}_4$, and therefore g must be true in $\mathcal{M}$. Since g satisfies rules 1a and 1b, we conclude that World($\mathcal{M}_4$) $\in$ Worlds($U(\mathcal{M})$). ◊

We conclude this section by analyzing the truth–maintenance semantics in the same framework as for other proposed semantics. First, the computational complexity of the Update Algorithm for the truth–maintenance semantics is no higher than for the standard semantics in the case of ground updates and theories; when Skolem constants are present, the size of the formulas added in Step 4″ depends on the number of datoms in ω that unify with one another. If there are D such most general unifications, then $\mathcal{O}(k2^D)$ is the maximum size of an instantiation of formula 2″, implying that in the worst case Step 4″ can add $\mathcal{O}(nk2^D)$ occurrences of atoms to $\mathcal{T}'$. D can be as large as $\binom{k}{2}$ (when every datom of ω unifies with every other datom of ω).

For comprehensibility and intuition, the semantics initially scores well; but when more complicated updates are contemplated and the question of update equivalence is raised (Chapter 8), the criteria for equivalence are more complex than we find ideal, due to dependence on the syntax of the update.

Theorem 7-5 shows that the truth–maintenance semantics has sufficient expressive power.

Theorem 7-5. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be extended relational theories over the same language, with the same strict axioms, containing disjoint sets of Skolem

constants,† such that $\mathcal{T}_1$ is consistent. Then under the truth-maintenance semantics, there exists an update U such that $\text{Worlds}(U(\mathcal{T}_1))|\mathcal{D} = \text{Worlds}(\mathcal{T}_2)\ |\mathcal{D}$, where $\mathcal{D}$ is $\mathcal{L}$ minus the Skolem constants appearing in $\mathcal{T}_1$. $\Diamond$

As for other aspects of expressive power, it may be argued that the definition of positive and negative presence in a wff should be *more* strongly related to syntax and less strongly to logical implication. For example, if one were to request the insertion of the exclusive–or of two datoms a and b, is it intuitive or desirable that a and b do not appear positively in ω?

Before the proof of Theorem 7-5, we present the proof of Theorem 7-1, which is the equivalent of Theorem 7-5 for the standard semantics. Both proofs make use of a Reformat() function that systematically removes all history atoms from the body of $\mathcal{T}$, when that body is expressed as a conjunction of formulas $\mathcal{B}$. The idea is to remove each history atom h from $\mathcal{B}$ by replacing $\mathcal{B}$ by $(\mathcal{B})^h_T \vee (\mathcal{B})^h_F$. If $\mathcal{B}$ contains no Skolem constants, this operation preserves the alternative worlds of $\mathcal{B}$. If $\mathcal{B}$ contains Skolem constants, then the truth valuation of h may determine the truth valuation of other datoms, such as in the formula $H(\epsilon) \wedge \epsilon{=}c \wedge \neg H(c)$. For formulas such as this, all unifications of h with atoms of $\mathcal{B}$ must be taken into account before h is replaced by a truth value.

Definition. Let α be a ground wff, and let $\mathcal{T}$ be an extended relational theory. Then *Reformat*(α) is the wff β formed from α by the following procedure:

1. *Initialize.* Set β to be α.
2. *Repeat.* If β contains no history atoms, then the procedure terminates.
3. *Remove a history atom.* Let h be a history atom in β. Let Σ be the set of most general unifications under which h unifies with an atom of β, and let S be the power set of Σ. Replace β by the formula

$$\bigvee_{s\in S}\Big(\bigwedge_{\sigma\in s}\sigma \wedge \bigwedge_{\substack{\sigma\in\Sigma\\ \sigma\notin s}}\neg\sigma \wedge ((\beta)^{hh_1\cdots h_n}_{TT\cdots T} \vee (\beta)^{hh_1\cdots h_n}_{FF\cdots F})\Big),$$

where h_1 through h_n are the atoms in β that unify with h under a most general substitution σ in s. Go back to step 2. $\Diamond$

Proof of Theorem 7-1. Let U be the update

INSERT $\bigwedge_{g\in\mathcal{T}_1}(g\vee\neg g) \wedge$ Reformat($\mathcal{B}$) WHERE T,

where g ranges over the datoms of $\mathcal{T}_1$ and $\mathcal{B}$ is the conjunction of all the formulas in the body of $\mathcal{T}_2$. The first conjunct of ω establishes that every datom in $\mathcal{T}_1$ can change its truth valuation. From this and the second conjunct, the theorem will follow if we can show that $\mathcal{T}_2'$, the extended relational theory with body

† This restriction is needed to avoid naming conflicts; if it is violated, rename the Skolem constants of one of the theories before testing.

Reformat($\mathcal{B}$), has the same alternative worlds as $\mathcal{T}_2$. Assume without loss of generality that the body of $\mathcal{T}_2$ is the single formula $\mathcal{B}$.

We first show that models of $\mathcal{T}_2$ are also models of $\mathcal{T}_2'$. Suppose that $\mathcal{M}$ is a model of $\mathcal{T}_2$ but not of $\mathcal{T}_2'$. Then $\mathcal{B}$ is true in $\mathcal{M}$ but Reformat($\mathcal{B}$) is false in $\mathcal{M}$.

Let s be the maximal member of S such that all the substitutions σ in s are true in $\mathcal{M}$. Let h be any history atom in $\mathcal{B}$, and let h_1 through h_n be the atoms of $\mathcal{B}$ with which h unifies under substitutions in s. Then $\mathcal{M}$ is a model of $\bigwedge_{\sigma\in s}\sigma \wedge \bigwedge_{\sigma\notin s}\neg\sigma$. Further, if h is true in $\mathcal{M}$ then $\mathcal{M}$ is a model of $(\mathcal{B})^{hh_1\cdots h_n}_{TT\cdots T}$, and if h is false in $\mathcal{M}$ then $\mathcal{M}$ is a model of $(\mathcal{B})^{hh_1\cdots h_n}_{FF\cdots F}$. Inducting, we conclude that Reformat(β) must be true in $\mathcal{M}$, a contradiction.

We now show the reverse direction. Suppose that there is an alternative world $\mathcal{A}$ such that $\mathcal{A} \in$ Worlds($\mathcal{T}_2'$) but $\mathcal{A} \notin$ Worlds($\mathcal{T}_2$). For $\mathcal{M}$ any model of $\mathcal{T}_2'$ with alternative world $\mathcal{A}$, $\mathcal{B}$ is false in $\mathcal{M}$. We will construct a particular choice of $\mathcal{M}$ from $\mathcal{A}$ by iteratively adding history atom truth assignments to $\mathcal{A}$. We will choose these truth assignments by running the Reformat() procedure backward. Initially, let $\mathcal{M}$ contain just the truth assignments and mappings given in $\mathcal{A}$.

Let s be the maximal member of S such that all the substitutions σ in s are true in $\mathcal{M}$. Then $\mathcal{M}$ is a model of $\bigwedge_{\sigma\in s}\sigma \wedge \bigwedge_{\sigma\notin s}\neg\sigma$. Let h be the last history atom in $\mathcal{B}$ to be replaced by the Reformat() procedure, and let β be the partially reformatted version of $\mathcal{B}$ right before h is removed. Let h_1 through h_n be the atoms of β with which h unifies under σ. Reformat($\mathcal{B}$) is true in $\mathcal{M}$; therefore either $(\beta)^{hh_1\cdots h_n}_{TT\cdots T}$ or $(\beta)^{hh_1\cdots h_n}_{FF\cdots F}$ is true in $\mathcal{M}$. If the former disjunct is true, assign T to h in $\mathcal{M}$; otherwise, make h false in $\mathcal{M}$. Then $\mathcal{M}$ satisfies β.

Inducting, we conclude that $\mathcal{M}$ must be a model of $\mathcal{B}$ and $\mathcal{T}_2$, a contradiction. We conclude that all models of $\mathcal{T}_2'$ are models of $\mathcal{T}_2$, and that $\mathcal{T}_2$ and $\mathcal{T}_2'$ have the same alternative worlds. ◇

Proof of Theorem 7-5. Let U be the update

INSERT $\bigwedge_{g\in\mathcal{T}_1}(g\vee\neg g) \wedge \bigwedge_{g\in\mathcal{T}_2}(g\vee\neg g) \wedge$ Reformat($\mathcal{B}$) WHERE T,

where g ranges over the datoms of $\mathcal{T}_1$ and $\mathcal{T}_2$. Then no datom appears positively or negatively in U, and the remainder of the proof follows from the proof of Theorem 7-1. ◇

7.5. Summary and Conclusion

In this chapter we identified a spectrum of possible update semantics, ranging from the standard semantics at one extreme to variants of the minimal–change semantics on the other. We showed how the Update Algorithm was easily adapted to other choices of semantics by changing the formula of Step 4.

Unless the semantics of the application dictate otherwise, we find the standard semantics to be preferable to the minimal–change semantics for two reasons: First, a separate type of update is needed under the minimal–change semantics in

order to move from a situation with more knowledge to a situation with less. For example, under the minimal–change semantics one cannot directly observe that the truth valuation of a fact is now unknown, since formulas such as Emp(Reid, CSD)∨¬Emp(Reid, CSD) are already true in all models of any extended relational theory. The new type of update could have the standard semantics, a non-model–based minimal–change semantics, or some hybrid semantics.

Second, the Update Algorithm is more expensive under the minimal–change semantics: in executing a minimal–change update, the size of $\mathcal{T}$ may increase by a number of atoms that is exponential in the size of the update. This is true even though the alternative worlds produced by the minimal–change semantics are always a subset of those that would be produced by the standard semantics: the subset is more difficult to characterize than the set as a whole.

Chapter 8: Equivalence of Updates

Chapter 7 delved into the properties of a variety of semantics: the standard semantics, defined in Chapter 3; the minimal–change semantics, a subset of which was used by DeKleer [85] and Reiter [85]; Weber's semantics [Weber 85]; and the truth–maintenance semantics. One of the criteria for choosing a semantics for updates, as discussed in Chapter 7, is update *comprehensibility*: a user should be able to look at an update and understand what the update will do. Though a qualitative discussion of the merits of different choices for semantics is indispensable, we have found that theorems on equivalence of updates also go a long way toward exposing the peculiarities of a particular choice of semantics. Such theorems tell exactly whether two updates look similar but really aren't, and whether two different-looking updates really are the same; they provide an impassionate demonstration of the properties of different semantics. Equivalence theorems can be used to evaluate how well a given semantics meets intuition: if a pair of updates should be the same according to intuition, but an equivalence theorem says that they are different (or vice versa), then the discrepancy can be registered as a mark against that semantics.

Section 8.1 of this chapter includes a set of semantics–independent theorems on update equivalence. These theorems will simplify the proofs of subsequent sections, and demonstrate patterns that recur across a wide class of semantics. Section 8.1 shows that strict axioms can be eliminated from consideration when considering update equivalence for a broad class of semantics; Section 8.2 does the same for Skolem constants. Section 8.3 includes theorems exactly characterizing when two updates are equivalent under the standard semantics, and Sections 8.4 and 8.5 do the same for the minimal–change and truth–maintenance semantics, respectively. We begin by defining update equivalence.

Definitions. If U_1 and U_2 are two updates over a language $\mathcal{L}$, then U_1 and U_2 are *equivalent* if for every extended relational theory $\mathcal{T}$ over $\mathcal{L}$, Worlds($U_1(\mathcal{T})$) = Worlds($U_2(\mathcal{T})$). U_1 and U_2 are *equivalent with respect to a model* $\mathcal{M}$ of $\mathcal{T}$ if Worlds($U_1(\mathcal{M})$) = Worlds($U_2(\mathcal{M})$). U_1 and U_2 are *equivalent with respect to an extended relational theory* $\mathcal{T}$ if Worlds($U_1(\mathcal{T})$) = Worlds($U_2(\mathcal{T})$).

8.1. Semantics–Independent Theorems on Update Equivalence

This section includes theorems on update equivalence that are in large part independent of the choice of semantics for updates. The goal of this section is to build up general principles for use in attacking the update equivalence problem.

In particular, we will develop a technique for reducing the question of equivalence of updates with different selection clauses ϕ to the case of a single selection clause; eliminate the need to consider extended relational theories with strict axioms when investigating equivalence; and suggest a methodology for deriving theorems about update equivalence.

The standard, minimal–change, and truth–maintenance semantics are all members of the larger class of semantics under consideration in this section. This class is composed of those semantics that satisfy a set of five basic properties. In particular:

(1) These semantics must be defined by the alternative worlds an update produces when applied to the individual alternative worlds of an extended relational theory;

(2) For any update U applied to an extended relational theory $\mathcal{T}$ over a language $\mathcal{L}$, there must exist another extended relational theory $\mathcal{T}'$ over $\mathcal{L}$ such that Worlds($\mathcal{T}'$) = Worlds($U(\mathcal{T})$);

Property (2) ensures that updates map extended relational theories to extended relational theories under any semantics in this class.

(3) If two updates U_1 and U_2 over $\mathcal{L}$ are equivalent, then they must be equivalent over all extensions of $\mathcal{L}$.

Property (3) ensures that the semantics does not include explicit tests for whether specific constants are in $\mathcal{L}$.

Properties (4) and (5) are a bit more complicated; their intent is to ensure that the effect of any update U on a model $\mathcal{M}$ of an extended relational theory $\mathcal{T}$ is independent of the other models of $\mathcal{T}$. More formally:

(4) If $\mathcal{M}$ is a model of two extended relational theories $\mathcal{T}_1$ and $\mathcal{T}_2$ without strict axioms, then $U(\mathcal{M})$ must be the same regardless of whether $\mathcal{T}_1$ or $\mathcal{T}_2$ is being updated.

Weber's semantics fails to satisfy basic property (4): Weber's semantics examines every model of $\mathcal{T}$ before deciding what the effect of an update should be.

(5) For two models $\mathcal{M}_1$ and $\mathcal{M}_2$ of an extended relational theory $\mathcal{T}$ and any update U, if $\mathcal{M}_1$ and $\mathcal{M}_2$ agree on the truth valuations of all datoms and equality atoms, then $U(\mathcal{M}_1) = U(\mathcal{M}_2)$.

Property (5) says that when two models "represent" the same complete–information database, an update will affect them identically.

Properties (1) and (4) together ensure that although update syntax may be important for a semantics in this class, syntax does not play a role in the bodies of extended relational theories: if two extended relational theories have the same axioms, then they will have identical sets of alternative worlds after a series of updates under any semantics in this class if the bodies of the two theories are logically equivalent. Properties (4) and (5) also have ramifications for update equivalence testing:

Theorem 8-1. Two null–free updates U_1 and U_2 are equivalent iff for all models $\mathcal{M}$ of extended relational theories, U_1 and U_2 are equivalent with respect to $\mathcal{M}$. $\Diamond$

Proof of Theorem 8-1. To show that this condition is sufficient for equivalence, note that if U_1 and U_2 are equivalent with respect to every model of an extended relational theory $\mathcal{T}$, then they must be equivalent with respect to $\mathcal{T}$.

To show that this condition is necessary for equivalence, suppose that U_1 and U_2 are equivalent, but for some model $\mathcal{M}$ of an extended relational theory $\mathcal{T}$, Worlds($U_1(\mathcal{M})$) $\neq$ Worlds($U_2(\mathcal{M})$). By basic property (3), U_1 and U_2 must be equivalent with respect to $(\mathcal{T})_\sigma$, where σ is the Skolem constant substitution for $\mathcal{M}$ with respect to $\mathcal{T}$. Then $\mathcal{M}$ is a model of $(\mathcal{T})_\sigma$, and Worlds($U_1(\mathcal{M})$) $\neq$ Worlds($U_2(\mathcal{M})$). But any other model $\mathcal{M}'$ of $(\mathcal{T})_\sigma$ agrees with $\mathcal{M}$ on the truth valuation of all atoms, and therefore by basic property (5), U_1 and U_2 are not equivalent with respect to $(\mathcal{T})_\sigma$, a contradiction. $\Diamond$

Some of the theorems presented in this section only apply to semantics that meet additional, less basic criteria:

P1. *The Irrelevance Principle.* Let $\mathcal{M}$ be a model of an extended relational theory, and let U be the update `INSERT` ω `WHERE` F. Then Worlds($U(\mathcal{M})$) = {World($\mathcal{M}$)}.

P2. *The ϕ-Independence Principle.* Given the two updates `INSERT` ω `WHERE` ϕ_1 and `INSERT` ω `WHERE` ϕ_2, if ϕ_1 and ϕ_2 are both true in a model $\mathcal{M}$, then these two updates are equivalent with respect to $\mathcal{M}$.

Principle P1 says that an update with selection clause F does not change the alternative world of any model to which it is applied. Principle P2 ensures that ϕ does not have any effect on the outcome of the update other than determining the alternative worlds in which changes can take place.

Proposition 8-1. The standard, minimal–change, and truth–maintenance semantics satisfy principles P1 and P2. $\Diamond$

Proof of Proposition 8-1. Left to the reader. $\Diamond$

The following theorem will motivate a strategy of attack for proving update equivalence under a variety of semantics.

Theorem 8-2. Let U_1 through U_4 be null–free updates under a semantics that satisfies principles P1 and P2:

U_1: INSERT ω_1 WHERE ϕ,

U_2: INSERT ω_2 WHERE ϕ,

U_3: INSERT ω_1 WHERE ψ,

U_4: INSERT ω_2 WHERE ψ.

If ψ logically entails ϕ and U_1 and U_2 are equivalent, then U_3 and U_4 are equivalent. $\Diamond$

Proof of Theorem 8-2. Let $\mathcal{T}$ be an extended relational theory and $\mathcal{M}$ a model of $\mathcal{T}$. If ψ is false in $\mathcal{M}$, then by principles P1 and P2, U_3 and U_4 are equivalent with respect to $\mathcal{M}$.

If ψ is true in $\mathcal{M}$, then ϕ is also true in $\mathcal{M}$. It follows by principle P2 that U_1 and U_3 are equivalent when applied to $\mathcal{M}$. Similarly, U_2 and U_4 must be equivalent with respect to $\mathcal{M}$. By Theorem 8-1, it follows that U_3 and U_4 are equivalent with respect to $\mathcal{M}$. We conclude that U_3 is equivalent to U_4. $\Diamond$

The value of Theorem 8-2 is that it provides a necessary condition for two updates U_1 and U_2 with selection clause ϕ to be equivalent: For every truth valuation v of all the atoms of ϕ under which ϕ is satisfied, INSERT ω_1 WHERE v† and INSERT ω_2 WHERE v must be equivalent. Theorem 8-3 shows that these conditions are both necessary and sufficient.

Theorem 8-3. Let U_1 and U_2 be two null–free updates with semantics that satisfy principles P1 and P2:

U_1: INSERT ω_1 WHERE ϕ,

U_2: INSERT ω_2 WHERE ϕ.

Then U_1 and U_2 are equivalent iff for all truth valuations v for all the atoms of ϕ such that ϕ is satisfied under v, INSERT ω_1 WHERE v is equivalent to INSERT ω_2 WHERE v. $\Diamond$

When threshing out theorems on update equivalence, Theorem 8-3 suggests that the most fruitful strategy may be to first concentrate on updates where ϕ is a conjunction of literals. Theorems 8-12 and 8-16 show this technique in use for the minimal–change and truth–maintenance semantics, respectively.

Proof of Theorem 8-3. The necessity of this condition is shown by Theorem 8-2. For sufficiency, suppose that U_1 and U_2 satisfy this condition but are not equivalent. Then U_1 and U_2 produce different sets of alternative worlds when applied to some extended relational theory $\mathcal{T}$. In particular, they must

† A truth valuation v can be written in wff form as a conjunction of literals, such that the atom a is a conjunct of v in wff form iff a receives the truth valuation T under v, and $\neg a$ is a conjunct of v in wff form iff a receives the truth valuation F under v.

produce different sets of alternative worlds when applied to some model $\mathcal{M}$ of $\mathcal{T}$. By principle P1, this implies that ϕ is true in $\mathcal{M}$. But given that ϕ is true, by principle P2 the effect of U_1 and U_2 on $\mathcal{M}$ is independent of ϕ, and must be the same as the effect on $\mathcal{M}$ of `INSERT` ω_1 `WHERE` v and `INSERT` ω_2 `WHERE` v, where v is the truth valuation in $\mathcal{M}$ of the atoms of ϕ. Since these two updates are equivalent by definition, by Theorem 8-1 they are equivalent with respect to $\mathcal{M}$. It follows that U_1 and U_2 must be equivalent with respect to $\mathcal{M}$, and therefore U_1 and U_2 must be equivalent. $\Diamond$

The next theorem shows that it suffices to consider only extended relational theories without strict axioms when checking update equivalence. Strict axioms were introduced for the standard semantics in Chapter 6; Principle P3 extends that definition to other choices of semantics.

Principle P3. *The Strict Axiom Principle.* Let $\mathcal{T}$ be an extended relational theory without strict axioms, and let α be a set of strict axioms. Then for any update U,

$$\text{Worlds}(U(\mathcal{T}+\alpha)) = \text{Worlds}\Big(\bigcup_{\mathcal{M}\in(Models(\mathcal{T})\cap Models(\alpha))} U(\mathcal{M}) \cap \text{Models}(\alpha)\Big). \quad \Diamond$$

In other words, the sole effect of strict axioms on update semantics is the requirement that the strict axioms be satisfied by every model to which U is applied and by every model produced by U.

Theorem 8-4. Two updates U_1 and U_2 under a semantics satisfying principle P3 are equivalent iff U_1 and U_2 are equivalent with respect to all extended relational theories without strict axioms. $\Diamond$

Proof of Theorem 8-4. If U_1 and U_2 are equivalent when applied to extended relational theories with strict axioms, then they must be equivalent when applied to extended relational theories without strict axioms, as these constitute a proper subset.

Suppose now that U_1 and U_2 are equivalent with respect to any extended relational theory without strict axioms, but are not equivalent with respect to some theory $\mathcal{T}$ that contains strict axioms. Let $\mathcal{M}$ be a model of $\mathcal{T}$. Let $\mathcal{T}'$ be an extended relational theory formed from $\mathcal{T}$ by removing the strict axioms of $\mathcal{T}$. Let $\mathcal{M}'$ be a model identical to $\mathcal{M}$; then $\mathcal{M}'$ is a model of $\mathcal{T}'$. By Theorem 8-1, since U_1 and U_2 are equivalent with respect to $\mathcal{T}'$, they must be equivalent with respect to $\mathcal{M}'$. But then by principle P3, U_1 and U_2 must be equivalent with respect to $\mathcal{M}$, because $\text{Worlds}(U_1(\mathcal{M})) = \text{Worlds}(U_1(\mathcal{M}') \cap \text{Worlds}(\text{Models}(\alpha)))$,

and similarly for U_2. We conclude that it suffices to consider extended relational theories without strict axioms when proving results about update equivalence. ◇

From now on, the statements of the theorems of this chapter will cover extended relational theories both with and without strict axioms, but the proofs of theorems will only consider extended relational theories without strict axioms.

For a pair of updates U_1 and U_2 with selection clauses ϕ_1 and ϕ_2, respectively, one could easily imagine a scenario where even though ϕ_1 and ϕ_2 were mutually exclusive, U_1 just happened to produce exactly the same sets of alternative worlds as U_2 did. Fortunately, Theorem 8-5 relegates this scenario to the realm of fantasy for semantics that satisfy principles P1 and P2.

Theorem 8-5. Let U_1 through U_4 be null–free updates under a semantics that satisfies principles P1 and P2:

U_1: INSERT ω_1 WHERE ϕ_1,

U_2: INSERT ω_2 WHERE ϕ_2,

U_3: INSERT ω_1 WHERE $\phi_1 \wedge \phi_2$,

U_4: INSERT ω_2 WHERE $\phi_1 \wedge \phi_2$.

Then U_1 and U_2 are equivalent iff

(1) U_3 and U_4 are equivalent;

(2) If $\phi_1 \wedge \neg\phi_2$ is true in a model $\mathcal{M}$ of an extended relational theory, then $\text{Worlds}(U_1(\mathcal{M})) = \{\text{World}(\mathcal{M})\}$; and

(3) If $\phi_2 \wedge \neg\phi_1$ is true in a model $\mathcal{M}$ of an extended relational theory, then $\text{Worlds}(U_2(\mathcal{M})) = \{\text{World}(\mathcal{M})\}$. ◇

Proof of Theorem 8-5. For sufficiency, since U_3 and U_4 are equivalent, it follows by principle P2 that U_1 and U_2 are equivalent with respect to any model where $\phi_1 \wedge \phi_2$ is true. With respect to models where $\neg\phi_1 \wedge \neg\phi_2$ is true, by principle P1, U_1 and U_2 again are equivalent. For models where ϕ_1 and $\neg\phi_2$ or $\phi_2 \wedge \neg\phi_1$ is true, conditions (2) and (3) and principle P1 guarantee equivalence.

For necessity, suppose condition (2) is violated in a model $\mathcal{M}$ of $\mathcal{T}$. Then by principle P1, $\text{Worlds}(U_2(\mathcal{M})) = \{\text{World}(\mathcal{M})\}$. By Theorem 8-1, for U_1 and U_2 to be equivalent, $\text{Worlds}(U_1(\mathcal{M}))$ must be $\{\text{World}(\mathcal{M})\}$. We conclude that condition (2) is necessary and, by symmetry, condition (3) as well.

Now suppose condition (1) is violated. Then for some model $\mathcal{M}$ of an extended relational theory $\mathcal{T}$, $\phi_1 \wedge \phi_2$ is true in $\mathcal{M}$ and U_3 and U_4 are not equivalent with respect to $\mathcal{M}$. Then by principle P2, U_1 and U_2 also are not equivalent with respect to $\mathcal{M}$. By Theorem 8-1, it follows that condition (1) is necessary. ◇

8.2. Skolem Constants and Update Equivalence

In this section we show that the question of equivalence for updates and extended relational theories containing Skolem constants can be reduced to the question of equivalence for null–free updates and theories.

First we show that it suffices to consider null–free extended relational theories when proving update equivalence, for all semantics that satisfy principle P4:

P4. *The Substitution Principle.* Let $\mathcal{T}$ be an extended relational theory, and let U be the update `INSERT` ω `WHERE` ϕ. Let Σ be the set of Skolem constant substitutions σ for the models of $\mathcal{T}$. Then

$$\text{Worlds}(U(\mathcal{T})) = \bigcup_{\sigma \in \Sigma} \text{Worlds}((U)_\sigma((\mathcal{T})_\sigma)). \quad \Diamond$$

Proposition 8-2. The standard, minimal–change, and truth–maintenance semantics satisfy principle P4. $\Diamond$

Proof of Proposition 8-2. Left to the reader. $\Diamond$

The semantics used by Abiteboul and Grahne [85] fails to satisfy principle P4: the occurrence of a Skolem constant in an update is not tied to its occurrence in $\mathcal{T}$ under their semantics.

Theorem 8-6. Under a semantics satisfying principle P4 on substitution, two updates U_1 and U_2 over a language $\mathcal{L}$ are equivalent iff they are equivalent when applied to every null–free extended relational theory over $\mathcal{L}$ or an extension of $\mathcal{L}$. $\Diamond$

Theorem 8-6 shows that even when Skolem constants appear in U_1 and U_2, it suffices to consider equivalence with respect to theories not containing Skolem constants.

Proof of Theorem 8-6. The necessity of this condition follows from basic property (2) and the fact that extended relational theories not containing Skolem constants are a proper subset of all extended relational theories over $\mathcal{L}$ and extensions of $\mathcal{L}$.

For sufficiency, let $\mathcal{T}$ be an extended relational theory, and let Σ be the set of Skolem constant substitutions σ for the models of $\mathcal{T}$. Then by principle P4, $\text{Worlds}(U_1(\mathcal{T})) = \bigcup_{\sigma \in \Sigma} \text{Worlds}((U_1)_\sigma((\mathcal{T})_\sigma))$, and similarly for U_2. But $(\mathcal{T})_\sigma$ is a null–free extended relational theory over $\mathcal{L}$ or an extension of $\mathcal{L}$, so by definition

$(U_1)_\sigma$ and $(U_2)_\sigma$ are equivalent with respect to $(\mathcal{T})_\sigma$. It follows that U_1 and U_2 are equivalent with respect to every extended relational theory. $\Diamond$

We need some way of reducing the question of update equivalence for updates containing Skolem constants to a question of equivalence of a null–free updates. The key observation is that it suffices to consider a finite set of substitutions σ for the Skolem constants of the updates, as long as the semantics is *constant–independent*, as defined in principle P5. This principle and the following theorem make use of a new variety of syntactic replacement called a *constant swap*. A constant swap is a simultaneous syntactic replacement σ of one set of constants by another, such that for any wff α, $((\alpha)_\sigma)_\sigma$ is α. For example, $\begin{smallmatrix} c & c' \\ c' & c \end{smallmatrix}$ is a typical constant swap, one that replaces all occurrences of c by c' and vice versa. Constant swaps differ from substitutions in that all replacements are accomplished simultaneously.

P5. *The Constant–Independence Principle.* Let U be a null–free update, and let $\mathcal{T}$ and $\mathcal{T}'$ be null–free extended relational theories such that Worlds($\mathcal{T}'$) = Worlds($U(\mathcal{T})$). Let σ be a constant swap over any extension of $\mathcal{L}$. Then Worlds($(U)_\sigma((\mathcal{T})_\sigma)$) = Worlds($(\mathcal{T}')_\sigma$). $\Diamond$

The idea of principle P5 is that renaming constants in $\mathcal{T}$ and U and then performing U should be equivalent to first performing U and then renaming the constants in the resulting theory. This is very similar to principle P4, but with a different goal: assuring that no elements in the universe get special treatment in the semantics.

Proposition 8-3. The standard, minimal–change, and truth–maintenance semantics satisfy principle P5. $\Diamond$

Proof of Proposition 8-3. Left to the reader. $\Diamond$

Theorem 8-7. Let U_1 and U_2 be updates under a semantics satisfying principles P3, P4, and P5:

U_1: `INSERT` ω_1 `WHERE` ϕ,

U_2: `INSERT` ω_2 `WHERE` ϕ.

Suppose that U_1 and U_2 contain n Skolem constants. Let C be the set containing all the constants of U_1 and U_2 plus n additional constants (extend $\mathcal{L}$ and the unique name axioms if necessary). Let Σ_c be the set of all substitutions of constants in C for all the Skolem constants of U_1 and U_2. Then U_1 and U_2 are equivalent iff

(1) ϕ is unsatisfiable; or

(2) For all substitutions $\sigma_c \in \Sigma_c$, $(U_1)_{\sigma_c}$ is equivalent to $(U_2)_{\sigma_c}$. $\Diamond$

The suggested set of substitutions Σ_c in Theorem 8-7 is of size exponential in the number of constants and Skolem constants in the update; the size of this set can in general be reduced. For example, if there are no equality atoms in U_1 and U_2 that contain Skolem constants, only one substitution need be included in Σ_c. If there is just one equality atom, containing a single Skolem constant, only two substitutions need be considered: one where the equality atom is true, and another where it is false.

Theorem 8-7 and principle P5 can be generalized by permitting special treatment in the semantics for any finite set of constants. For example, an analogue of Theorem 8-7 will be true if all datoms over constants occurring in the body or strict axioms of $\mathcal{T}$ are treated specially by the semantics. All such constants must be included in C.

The proof of Theorem 8-7 uses a special property of constant swaps:

Proposition 8-4. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two theories, and let σ be a constant swap. Then $\text{Worlds}(\mathcal{T}_1) = \text{Worlds}(\mathcal{T}_2)$ iff $\text{Worlds}((\mathcal{T}_1)_\sigma) = \text{Worlds}((\mathcal{T}_2)_\sigma)$. $\Diamond$

Proof of Proposition 8-4. Let $\mathcal{M}_1$ be a model of $\mathcal{T}_1$, and $\mathcal{M}_2$ a model of $\mathcal{T}_2$. Let $\mathcal{M}_1'$ be a model created from $\mathcal{M}_1$ as follows: Let the constant and Skolem constant mappings of $\mathcal{M}_1'$ be the same as those of $\mathcal{M}_1$, except that if $\frac{c'}{c}$ appears in σ, then in $\mathcal{M}_1'$ c' is mapped to the universe element that c was mapped to in $\mathcal{M}_1$. Let every atom g in $\mathcal{M}_1'$ have the truth valuation of $(g)_\sigma$ in $\mathcal{M}_1$. The unique name axioms are satisfied by $(\mathcal{M}_1)_\sigma$, because σ is invertible. Then $(\mathcal{M}_1)_\sigma$ is a model of $(\mathcal{T})_\sigma$, because the truth valuation of every atom g in $\mathcal{M}_1$ is the same as that of $(g)_\sigma$ in $\mathcal{M}_1'$.

Construct $\mathcal{M}_2'$ similarly. Then $\mathcal{M}_1$ and $\mathcal{M}_2$ have the same universe and constant and Skolem constant mappings iff $\mathcal{M}_1'$ and $\mathcal{M}_2'$ do. Further, an atom g is true in $\mathcal{M}_1$ and $\mathcal{M}_2$ iff $(g)_\sigma$ is true in both $\mathcal{M}_1'$ and $\mathcal{M}_2'$. We conclude that $\text{Worlds}(\mathcal{M}_1') = \text{Worlds}(\mathcal{M}_2')$ iff $\text{World}(\mathcal{M}_1) = \text{World}(\mathcal{M}_2)$. $\Diamond$

Proof of Theorem 8-7. Necessity is immediate for any semantics satisfying principle P4.

For sufficiency, suppose that $\mathcal{M}$ is a model of an extended relational theory $\mathcal{T}$ without Skolem constants, such that U_1 and U_2 are not equivalent with respect to $\mathcal{M}$. Let σ be the Skolem constant substitution for $\mathcal{M}$ with respect to U_1 and U_2. Then σ is not in Σ_c, and must include some constant c that is not in C and therefore does not occur in U_1 or U_2. There are at most n of these constants. Let σ' be the constant swap that replaces all such constants c by constants c' that occur in C but not in $(U_1)_\sigma$ or $(U_2)_\sigma$. By definition of Σ_c, this must be possible. Then for some substitution $\sigma_c \in \Sigma_c$, $((U_1)_\sigma)_{\sigma'} = (U_1)_{\sigma_c}$, and $((U_2)_\sigma)_{\sigma'} = (U_2)_{\sigma_c}$, so $((U_1)_\sigma)_{\sigma'}$ and $((U_2)_\sigma)_{\sigma'}$ are equivalent by assumption.

Because $\text{Worlds}((U_1)_\sigma(\mathcal{M})) \neq \text{Worlds}((U_2)_\sigma(\mathcal{M}))$, it follows from Theorem 8-1 that $\text{Worlds}((U_1)_\sigma((\mathcal{T})_\sigma)) \neq \text{Worlds}((U_2)_\sigma((\mathcal{T})_\sigma)$. By principle P5

and Proposition 8-4, $\text{Worlds}(((U_1)_\sigma)_{\sigma'}(((\mathcal{T})_\sigma)_{\sigma'})) \neq \text{Worlds}(((U_2)_\sigma)_{\sigma'}(((\mathcal{T}_2)_\sigma)_{\sigma'})$. But $((U_1)_\sigma)_{\sigma'}$ and $((U_2)_\sigma)_{\sigma'}$ are equivalent by assumption. We conclude that this condition is sufficient for equivalence. $\Diamond$

8.3. The Standard Semantics and Update Equivalence

Under the standard semantics, what conditions govern equivalence when two updates have different selection clauses ϕ? Theorem 8-5 says that if U_1 and U_2 are two equivalent null–free updates with selection clauses ϕ_1 and ϕ_2, respectively, then U_1 must not make any changes in any model where ϕ_2 is false, and vice versa. This characterization is almost sufficient; it only lacks exact conditions under which a standard–semantics update will not change an alternative world.

Theorem 8-8. Let U_1 and U_2 be two null–free updates under the standard semantics:

U_1: `INSERT` ω_1 `WHERE` ϕ_1,

U_2: `INSERT` ω_2 `WHERE` ϕ_2.

Then U_1 and U_2 are equivalent iff

(1) `INSERT` ω_1 `WHERE` $\phi_1 \wedge \phi_2$ is equivalent to `INSERT` ω_2 `WHERE` $\phi_1 \wedge \phi_2$;

(2) $\phi_1 \wedge \neg\phi_2$ logically entails ω_1 and $\phi_2 \wedge \neg\phi_1$ logically entails ω_2; and

(3) If $\phi_1 \wedge \neg\phi_2$ is satisfiable, then ω_1 is uniquely satisfiable[†]; and if $\phi_2 \wedge \neg\phi_1$ is satisfiable, then ω_2 is uniquely satisfiable. $\Diamond$

Proof of Theorem 8-8. By Theorem 8-5, condition (1) is necessary. To see that condition (3) is necessary, suppose that, say, $\phi_1 \wedge \neg\phi_2$ is satisfiable with truth valuation u for the datoms of ϕ_1 and ϕ_2. Let $\mathcal{T}$ be an extended relational theory with body u. Let $\mathcal{M}$ be a model of $\mathcal{T}$; then $U_2(\mathcal{M}) = \{\mathcal{M}\}$. For U_1 to be equivalent to U_2, then, U_1 cannot change the alternative world of $\mathcal{M}$, by Theorem 8-1. Since the number of alternative worlds U_1 produces from $\mathcal{M}$ will be equal to the number of valuations for ω_1 that satisfy ω_1, if U_1 is equivalent to U_2 there must be only one valuation, v, that satisfies ω_1; therefore condition (3) is necessary. To show that condition (2) is necessary, since $\text{Worlds}(U_1(\mathcal{M}))$ must be $\{\mathcal{M}\}$, v must agree with u on all datoms in v. Since u may be any valuation satisfying $\phi_1 \wedge \neg\phi_2$, v must be a subset of every valuation satisfying $\phi_1 \wedge \neg\phi_2$; in other words, $\phi_1 \wedge \neg\phi_2$ logically entails v; since v is logically equivalent to ω_1, $\phi_1 \wedge \neg\phi_2$ logically entails ω_1, implying that condition (2) is also necessary. The proof is symmetric if $\phi_2 \wedge \neg\phi_1$ is satisfiable.

We now turn to the reverse implication, namely, that if conditions (1) through (3) are met, then U_1 and U_2 are equivalent. By Theorem 8-5, it suffices to show that conditions (2) and (3) imply that U_1 will not change the alternative world of a model where $\phi_1 \wedge \neg\phi_2$ is true. But this follows immediately from

[†] A wff α is *uniquely satisfiable* if there exists exactly one truth valuation v for all the atoms of α such that α is true under v.

conditions (2) and (3). A similar line of reasoning holds if $\phi_2 \wedge \neg\phi_1$ is true in a model. We conclude that U_1 and U_2 are equivalent when applied to $\mathcal{T}$. $\Diamond$

We now turn to the question of equivalence for pairs of updates having the same selection clause ϕ. We begin with a simple sufficient criterion for equivalence under the standard semantics:

Theorem 8-9. Let U_1 and U_2 be two null–free updates under the standard semantics:

U_1: `INSERT` ω_1 `WHERE` ϕ,

U_2: `INSERT` ω_2 `WHERE` ϕ.

If ω_1 and ω_2 are logically equivalent and the same datoms occur in ω_1 and ω_2, then U_1 is equivalent to U_2. $\Diamond$

Proof of Theorem 8-9. Assume that ω_1, and therefore ω_2, is satisfiable, as otherwise the theorem follows immediately. For any extended relational theory $\mathcal{T}$ without strict axioms, consider the effects of U_1 and U_2 on a model $\mathcal{M}$ of $\mathcal{T}$. U_1 must produce a model $\mathcal{M}'$ from $\mathcal{M}$, since ω_1 is satisfiable. We wish to show that $\text{World}(\mathcal{M}') \in \text{Worlds}(U_2(\mathcal{M}))$. If ϕ is false in $\mathcal{M}$, this follows immediately. Otherwise ω_2 must be true in $\mathcal{M}'$, because ω_1 and ω_2 are logically equivalent; and therefore rule 2 in the standard–semantics definition of `INSERT` is satisfied for U_2 by $\mathcal{M}'$. Rule 1 in the definition of `INSERT` is also satisfied for U_2 by $\mathcal{M}'$, since U_1 and U_2 contain the same datoms. $\Diamond$

To see that the criteria of Theorem 8-9 are sufficient but not necessary, consider the two equivalent updates `INSERT` f `WHERE` $f \wedge g$ and `INSERT` g `WHERE` $f \wedge g$. These two updates fail the test of Theorem 8-9 because ω_1 and ω_2 contain datoms whose truth valuation is logically entailed by both ϕ and ω. To produce necessary and sufficient criteria, it will be advantageous to remove all such datoms from ω by *reducing* ω:

Definition. Let U be the update `INSERT` ω `WHERE` ϕ under the standard semantics. The *reduction of* ω *with respect to* ϕ, written $\text{red}(\omega, \phi)$, is formed from ω by making the following substitutions for every datom g in ω:

1. If ϕ and ω both logically entail g, replace g by T in ω.
2. If ϕ and ω both logically entail $\neg g$, replace g by F in ω. $\Diamond$

This definition may seem a bit odd for the case where ϕ is unsatisfiable, but such updates aren't very interesting anyway:

Proposition 8-5. Under the standard semantics, any update U: `INSERT` ω `WHERE` ϕ is equivalent to `INSERT` $\text{red}(\omega, \phi)$ `WHERE` ϕ. $\Diamond$

Proof of Proposition 8-5. Proof by induction: consider a single step in the reduction process. If g is a datom of ω that is eliminated in this step, then the insertion of ω into a model where ϕ is true cannot change the truth valuation of g. When g is replaced by T or F, creating ω', every set of truth valuations that satisfied ω maps into one set that satisfies ω', such that the two sets agree on every datom in common. Therefore inserting ω' into a model where ϕ is true will have the same effect as inserting ω into that model. $\Diamond$

Once the updates being tested for equivalence have been reduced, little work remains:

Theorem 8-10. Let U_1 and U_2 be null–free updates under the standard semantics:

U_1: `INSERT` ω_1 `WHERE` ϕ,

U_2: `INSERT` ω_2 `WHERE` ϕ.

U_1 and U_2 are equivalent iff (1) ϕ is unsatisfiable or (2) red(ω_1, ϕ) and red(ω_2, ϕ) contain the same datoms and are logically equivalent. $\Diamond$

Examples. If U_1 is `INSERT` g `WHERE` T and U_2 is `INSERT` $g \vee$ T `WHERE` T, then these two updates are reduced. Since g is not logically equivalent to $g \vee$ T, the two updates must not be equivalent; they differ on producing models where g is false. For updates `INSERT` g `WHERE` $g \wedge f$ and `INSERT` f `WHERE` $g \wedge f$, in both cases ω is replaced by T during reduction, and the two updates become identical. Theorem 8-10 will proclaim these two updates equivalent.

Proof of Theorem 8-10. By Proposition 8-5, it suffices to prove this theorem for the case where ω_1 and ω_2 are already reduced with respect to ϕ. Assume that ϕ is satisfiable, as otherwise the theorem follows immediately.

We first show that ω_1 and ω_2 must contain the same datoms. Suppose that g is a datom that is a subformula of, say, ω_1 but not of ω_2. Let $\mathcal{M}$ be a model of an extended relational theory $\mathcal{T}$ such that ω_1 is true in $\mathcal{M}$. Let $\mathcal{M}'$ be created from $\mathcal{M}$ by negating the truth valuation in $\mathcal{M}$ of g, and negating as few additional truth valuations as possible in order to make ϕ true in $\mathcal{M}'$. By the definition of reduction, the truth valuation of g in $\mathcal{M}'$ need not be negated again in order to satisfy ϕ. Then World($\mathcal{M}$) $\in$ Worlds($U_1(\mathcal{M}')$), but World($\mathcal{M}$) $\notin$ Worlds($U_2(\mathcal{M}')$). We conclude that U_1 and U_2 cannot be equivalent unless they contain the same datoms.

We now show that if ω_1 and ω_2 are not logically equivalent, then U_1 and U_2 are not equivalent. Select a truth valuation v for all the atoms of ω_1 and ω_2 such that, say, v satisfies ω_1 but not ω_2. Let u be a truth valuation for all the datoms of ϕ such that ϕ is satisfied under u. Create an extended relational theory $\mathcal{T}$ without strict axioms, with body u, and let $\mathcal{M}$ be a model of $\mathcal{T}$. Let $\mathcal{M}'$ be a model that agrees with v on all valuations of v, and with $\mathcal{M}$ on all other

information. Since ω_1 is satisfied in $\mathcal{M}'$ by construction, and $\mathcal{M}'$ agrees with $\mathcal{M}$ on all datoms not in ω_1, it follows that World($\mathcal{M}'$) $\in$ Worlds($U_1(\mathcal{M})$). $\mathcal{M}'$ cannot be a model of an alternative world of $U_2(\mathcal{M})$, because ω_2 is false in $\mathcal{M}'$. From Theorem 8-1, it follows that ω_1 and ω_2 must be logically equivalent if U_1 and U_2 are equivalent.

We now turn to the reverse implication, namely, that if ω_1 and ω_2 are logically equivalent and contain the same datoms, then U_1 and U_2 are equivalent. Assume that ω_1, and therefore ω_2, is satisfiable, as otherwise the theorem follows immediately.

For any extended relational theory $\mathcal{T}$ without strict axioms, consider the effects of U_1 and U_2 on a model $\mathcal{M}$ of $\mathcal{T}$ where ϕ is true. Since ω_1 is satisfiable, $U_1(\mathcal{M})$ is nonempty. Suppose $\mathcal{M}'$ is in $U_1(\mathcal{M})$. Since ω_1 and ω_2 are logically equivalent, ω_2 will be true in $\mathcal{M}'$, so rule 2 in the definition of **INSERT** is satisfied by U_2. Since ω_1 and ω_2 contain the same datoms, rule 1 is also satisfied by $\mathcal{M}'$. We conclude that U_1 and U_2 are equivalent. ◊

8.4. The Minimal–Change Semantics and Update Equivalence

We begin with simple sufficient conditions for update equivalence under the minimal–change semantics:

Theorem 8-11. Let U_1 and U_2 be two null–free updates under the minimal–change semantics:

U_1: **INSERT** ω_1 **WHERE** ϕ_1,

U_2: **INSERT** ω_2 **WHERE** ϕ_2,

Then U_1 and U_2 are equivalent if

(1) If $\phi_1 \wedge \phi_2$ is satisfiable, then ω_1 and ω_2 are logically equivalent; and

(2) $\phi_1 \wedge \neg\phi_2$ logically entails ω_1 and $\phi_2 \wedge \neg\phi_1$ logically entails ω_2. ◊

Proof of Theorem 8-11. First note that condition (1) implies that **INSERT** ω_1 **WHERE** $\phi_1 \wedge \phi_2$ is equivalent to **INSERT** ω_2 **WHERE** $\phi_1 \wedge \phi_2$. Therefore by Theorem 8-5, it suffices to show that condition (2) implies that if $\phi_1 \wedge \neg\phi_2$ is true in a model $\mathcal{M}$, then $U(\mathcal{M}) = \{\mathcal{M}\}$. But this follows immediately from the fact that ω_1 is already true in $\mathcal{M}$. A similar line of reasoning holds if $\phi_2 \wedge \neg\phi_1$ is true in $\mathcal{M}$. We conclude that conditions (1) and (2) are sufficient for equivalence. ◊

The conditions in Theorem 8-11 are in fact both necessary and sufficient if no datom appears in both ω_1 and ϕ_1 or in both ω_2 and ϕ_2. In the general case, however, ω_1 and ω_2 must be reduced before testing is done for logical equivalence. For example, although $R(a)$ and $R(b)$ are not logically equivalent, U_1: **INSERT** $R(a)$ **WHERE** $R(a) \wedge R(b)$ is equivalent to U_2: **INSERT** $R(b)$ **WHERE** $R(a) \wedge R(b)$, due to interactions between the atoms of ϕ and ω. The following definitions

show how to reduce ω for the minimal–change semantics; the procedure is more complex than for the standard semantics.

Definition. Let U be the update INSERT ω WHERE ϕ, under the minimal–change semantics. Then *the reduction of* ω *with respect to* ϕ, written red(ω, ϕ), is the wff formed from ω as follows:

1. Put ω into disjunctive normal form†.
2. If ϕ logically entails a literal l, and l appears as a conjunct of ω, then replace that conjunct of ω by T. ◊

Examples. The reduction of $R(a) \wedge (R(b) \vee \neg R(c))$ with respect to $\neg R(a) \wedge R(b)$ is the wff $(R(a) \wedge \mathsf{T}) \vee (R(a) \wedge \neg R(c))$. For ω any wff, red(ω, T) is ω in disjunctive normal form.

Unfortunately, even this stronger version of reduction does not lead to as elegant a theorem of equivalence as was possible under the standard semantics. A counterexample will illustrate why Theorem 8-10 fails to hold for the minimal–change semantics.

Example. Let f and g be datoms. Then the reduced update U_1: INSERT $f \vee g$ WHERE $f \vee g$ is equivalent to U_2: INSERT T WHERE $f \vee g$, even though Theorem 8-10 predicts inequivalence, if co-opted for the minimal–change semantics. The problem persists even if the requirement is removed in Theorem 8-10 that the same datoms appear in ω_1 and ω_2. ◊

Before the presentation of the equivalence theorem for the general case, one more bit of terminology:

Definitions. A wff ω is *basic* if ω is ground and does not contain Skolem constants or the equality predicate. An update U: INSERT ω WHERE ϕ is basic iff ω and ϕ are basic.

Theorem 8-12. Let U_1 and U_2 be basic updates under the minimal–change semantics:

U_1: INSERT ω_1 WHERE ϕ_1,

U_2: INSERT ω_2 WHERE ϕ_2.

Let ϕ be the wff red($\phi_1 \wedge \phi_2$, T). Then U_1 and U_2 are equivalent iff

(1) For every satisfiable disjunct D of ϕ, red(ω_1, D) is logically equivalent to red(ω_2, D).

(2) $\phi_1 \wedge \neg\phi_2$ logically entails ω_1 and $\phi_2 \wedge \neg\phi_1$ logically entails ω_2. ◊

† For our purposes here, a *literal* is an atom, a negated atom, or a truth value; and ω is in *disjunctive normal form* if ω is a disjunction of conjunctions of literals, and no disjunct contains both a literal and the negation of that literal as conjuncts.

The proof of Theorem 8-12 uses three auxiliary results: two lemmas and a theorem. Lemmas 8-1 and 8-2 give useful logical properties of reduced wffs. Theorem 8-13 is interesting in its own right; it shows that the reduction process may be used to find the minimal sets of atom truth valuation changes in a model that will make a particular wff true in that model.

Lemma 8-1. Two basic wffs ω_1 and ω_2 are logically equivalent iff for all basic satisfiable conjunctions of literals ϕ, red(ω_1, ϕ) and red(ω_2, ϕ) are logically equivalent. $\Diamond$

Proof of Lemma 8-1. To show that this condition is sufficient, recall that red(ω, T) is logically equivalent to ω, for all basic wffs ω. Therefore if red(ω_1, T) is logically equivalent to red(ω_2, T), then ω_1 and ω_2 must be logically equivalent.

To show that this condition is necessary, let l be a basic satisfiable literal. Assume that ω_1 and ω_2 are logically equivalent, but red(ω_1, l) and red(ω_2, l) are not. Let v be a truth valuation for all the atoms of ω_1 and ω_2 except the atom of l, such that $v \wedge \neg l$satisfies red(ω_1, l) $\wedge\neg$red(ω_2, l), say. (We choose $v \wedge \neg l$ rather than $v \wedge l$ because $v \wedge l$ must either satisfy both red(ω_1, l) and red(ω_2, l) or else fail to satisfy either, as otherwise $\omega_1 \wedge \neg\omega_2$ would be satisfied by $v \wedge l$, an impossibility since ω_1 and ω_2 are logically equivalent.) Suppose first that red(ω_2, l) is true under $v \wedge l$. Then there exists a disjunct d of ω_2 containing l such that d is true under $v \wedge l$. But red(d, l) must also be true under $v \wedge \neg l$, which implies that red(ω_2, l) is true under $v \wedge \neg l$, a contradiction.

Now suppose that red(ω_1, l) and red(ω_2, l) are both false under $v \wedge l$. Then ω_1 and ω_2 are also false under $v \wedge l$. By definition, red(ω_2, l) is false under $v \wedge \neg l$. It follows that ω_2 is false under $v \wedge \neg l$, because if all disjuncts d of ω_2 are such that red(d, l) is false under $v \wedge \neg l$, then d must also be false under $v \wedge \neg l$. As ω_1 and ω_2 are logically equivalent, ω_1 must also be false under $v \wedge \neg l$. As red(ω_1, l) is true under $v \wedge \neg l$ by definition, there must be some disjunct d of red(ω_1, T), such that d is false under $v \wedge l$ and red(d, l) is true under $v \wedge \neg l$. Then d must contain both l and $\neg l$, which is forbidden by the definition of disjunctive normal form. Therefore red(ω_1, l) and red(ω_2, l) must be logically equivalent.

As red(ω_1, $l_1 \wedge \cdots \wedge l_n$) = red($\cdots$ red(ω_1, l_1), $\cdots$, l_n), for l_i a basic literal, $1 \leq i \leq n$, we conclude that if ω_1 and ω_2 are logically equivalent, then red(ω_1, ϕ) and red(ω_2, ϕ) are logically equivalent. $\Diamond$

Given that two wffs ω_1 and ω_2 are not logically equivalent, Lemma 8-2 shows how to reduce ω_1 and ω_2 and still preserve that logical inequivalence.

Lemma 8-2. Let ω_1 and ω_2 be basic wffs, and let D be a satisfiable basic conjunction of literals T $\wedge$ $l_1 \wedge \cdots \wedge l_n$, for $n \geq 0$. If red(ω_1, D) and red(ω_2,

D) are not logically equivalent, then there exists a truth valuation v for all the atoms of D, ω_1, and ω_2 such that D is true under v and red(ω_1, v) and red(ω_2, v) are not logically equivalent. ◊

Proof of Lemma 8-2. Suppose red(ω_1, D) ∧ ¬red(ω_2, D) is satisfiable under truth valuation v'. Let v be formed by concatenating D and the truth valuations of v' that are consistent with D. Then red(ω_1, v) is satisfied under truth valuation v', but red(ω_2, v) cannot be satisfied by v'. ◊

In order to characterize the minimal sets of atom truth valuation changes in a model $\mathcal{M}$ that will make a particular wff true in $\mathcal{M}$, we need a slightly stronger notion of reducibility:

Definition. For ω a basic wff and ϕ a basic conjunction of literals, the *full reduction of* ω *with respect to* ϕ, written fred(ω, ϕ), is obtained as follows:

1. Set fred(ω, ϕ) to be red(ω, ϕ).
2. If one satisfiable disjunct d of fred(ω, ϕ) logically entails another, then remove d from fred(ω, ϕ). Repeat until no such disjunct remains. ◊

Example. An earlier example showed that red($R(a) \wedge (R(b) \vee \neg R(c))$, $\neg R(a) \wedge R(b)) = (R(a) \wedge \mathsf{T}) \vee (R(a) \wedge \neg R(c))$; for a full reduction, fred($R(a) \wedge (R(b) \vee \neg R(c))$, $\neg R(a) \wedge R(b)) = (R(a) \wedge \mathsf{T})$. ◊

Theorem 8-13. Let ω be a basic wff, and let v be any truth valuation for the atoms of ω. Let $\mathcal{M}$ be a model that satisfies v, and let $\mathcal{M}'$ be a model that agrees with $\mathcal{M}$ except on the truth valuations of a set S of datoms. Then for any satisfiable disjunct d of fred(ω, v):

1. ω is true in $\mathcal{M}'$ if S contains exactly the datoms of d.
2. ω is false in $\mathcal{M}'$ if S is a proper subset of the datoms of d. ◊

In other words, the disjuncts of a fully reduced ω represent the minimal changes needed to make ω true in a particular class of models. This suggests the following property, whose proof we omit:

Proposition 8-6. The basic update `INSERT` ω `WHERE` ϕ is equivalent to `INSERT` red(ω, ϕ) `WHERE` ϕ and `INSERT` fred(ω, ϕ) `WHERE` ϕ under the minimal-change semantics. ◊

Proof of Theorem 8-13. Assume without loss of generality that ω is in disjunctive normal form. We first show that if S contains exactly the datoms of d, then ω is true in $\mathcal{M}'$. Let d' be a disjunct of ω that is transformed into d during the full reduction of ω with respect to v, by replacing literals of d' by

T. Then d' contains all the literals of d as conjuncts, and other literals l as well (though not both any literal and its negation). But if l is removed from d' during the reduction process, then l is a conjunct of v, and therefore l is already true in $\mathcal{M}$. Therefore d' is also true in $\mathcal{M}'$, and ω is true in $\mathcal{M}'$.

To prove condition 2, let ω' be fred(ω, v). We first show that ω logically entails ω'. If ω is satisfied under truth valuation v', then some disjunct D of ω must be true under v'. Every conjunct of D must be true under v', and therefore red(D, ϕ) must be true under v' for any basic truth valuation ϕ. It follows that red(ω, ϕ) is true under v'. As red(ω, v) and ω' are logically equivalent, ω' must also be true under v'.

Suppose that S contains a proper subset of the datoms of a satisfiable disjunct d of ω'. By the definition of $\mathcal{M}$, every conjunct of ω' is false in $\mathcal{M}$, so S must contain all the datoms of *some* satisfiable disjunct d' of ω' if ω' is true in $\mathcal{M}'$. But then d logically entails d', so d should have been removed from ω' during the full reduction process. We conclude that ω', and hence ω, is false in $\mathcal{M}'$. $\Diamond$

Proof of Theorem 8-12. To show that condition 2 is necessary for equivalence, if $\mathcal{M}$ is a model where $\phi_1 \wedge \neg\phi_2$ is true and ω_1 is false, then U_1 applied to $\mathcal{M}$ will change or eliminate the alternative world of $\mathcal{M}$, whereas U_2 cannot affect that world.

To show that condition 1 is also necessary for equivalence, suppose there is some satisfiable disjunct D of ϕ such that red(ω_1, D) and red(ω_2, D) are not logically equivalent. Then by Lemma 8-2, there is a satisfiable extension D' of D that includes every atom of ω_1 and ω_2, such that red(ω_1, D') and red(ω_2, D') are not logically equivalent. Let ω_1' be fred(ω_1, D'), let ω_2' be fred(ω_2, D'), and let $\mathcal{M}$ be a model where D' is satisfied. Choose any one disjunct d of ω_1' or ω_2', say of ω_1', that does not subsume any disjunct of ω_2'; such a disjunct must occur in ω_1' or ω_2'. Let $\mathcal{M}'$ be a model that only disagrees with $\mathcal{M}$ on the truth valuations of atoms in d. By Theorem 8-13, $\mathcal{M}' \in U_1(\mathcal{M})$, but $\mathcal{M}' \notin U_2(\mathcal{M})$. But then Worlds($U_1(\mathcal{M})$) $\neq$ Worlds($U_2(\mathcal{M})$), and by Theorem 8-1 U_1 and U_2 are not equivalent.

We now show that conditions 1 and 2 are sufficient to guarantee that U_1 and U_2 are equivalent. By condition 2, U_1 and U_2 are equivalent when applied to all models where $\neg\phi$ is true, as U_1 and U_2 have no effect on such models. To show that condition 1 is sufficient for models where ϕ is true, suppose that for all satisfiable disjuncts D of ϕ, red(ω_1, D) is logically equivalent to red(ω_2, D). Then by Lemma 8-1, for every extension of D to a satisfiable conjunction D' of literals that includes all the atoms of ω_1 and ω_2, red(ω_1, D') is logically equivalent to red(ω_2, D'). For an arbitrary choice of D', let ω_1' be fred(ω_1, D'), and let ω_2' be fred(ω_2, D'). Then as ω_1' and red(ω_1, D') are logically equivalent, and similarly for ω_2', it follows that ω_2' and ω_1' are logically equivalent. In addition, by the definition of full reduction, ω_1' and ω_2' must contain the same disjuncts, up to reordering of literals within disjuncts.

By Theorem 8-13, the minimal sets of atom truth valuation changes that would make ω_1 true in a model $\mathcal{M}$ satisfying D' are given by the disjuncts of ω_1'. Whenever one of these sets of changes is made in $\mathcal{M}$, creating $\mathcal{M}'$, exactly one disjunct of ω_2' is also satisfied in $\mathcal{M}'$. Therefore Worlds($U_1(\mathcal{M})$) $\subseteq$ Worlds($U_2(\mathcal{M})$). By a symmetric argument, Worlds($U_1(\mathcal{M})$) = Worlds($U_2(\mathcal{M})$). As this construction holds for any model that satisfies ϕ, U_1 and U_2 are equivalent. $\Diamond$

8.5. The Truth–Maintenance Semantics and Update Equivalence

This section discusses update equivalence under the truth–maintenance semantics. As usual, we begin by showing how to reduce the case of updates with different selection clauses ϕ to the case of updates with a common selection clause.

Theorem 8-14. Let U_1 through U_4 be null–free updates under the truth–maintenance semantics:

U_1: INSERT ω_1 WHERE ϕ_1,

U_2: INSERT ω_2 WHERE ϕ_2,

U_3: INSERT ω_1 WHERE $\phi_1 \wedge \phi_2$,

U_4: INSERT ω_2 WHERE $\phi_1 \wedge \phi_2$.

Then U_1 and U_2 are equivalent iff

(1) U_3 and U_4 are equivalent; and

(2) If $\phi_1 \wedge \neg\phi_2$ is satisfiable, then

- There must be exactly one truth valuation v for all the datoms of ω_1 such that:
  - ω_1 is true under v; and
  - Each datom that appears positively or negatively in ω_1 is true or false, respectively, under v.
- Further, $\phi_1 \wedge \neg\phi_2$ must logically entail v.

(3) The analogous condition holds if $\phi_2 \wedge \neg\phi_1$ is satisfiable. $\Diamond$

Proof of Theorem 8-14. The necessity of condition (1) follows from Theorem 8-5. For condition (2), suppose that, say, $\phi_1 \wedge \neg\phi_2$ is satisfiable with valuation u. Let $\mathcal{T}$ be an extended relational theory without strict axioms, with body u. Let $\mathcal{M}$ be a model of $\mathcal{T}$; then $U_2(\mathcal{M}) = \{\mathcal{M}\}$. By Theorem 8-1, then, for U_1 to be equivalent to U_2, it must be the case that $U_1(\mathcal{M}) = \{\mathcal{M}\}$.

If ω_1 is unsatisfiable, then $U_1(\mathcal{M})$ is the empty set. We conclude that there is a truth valuation v for all the datoms of ω_1 such that ω_1 is satisfied under v. By Proposition 7-1, there exists such a valuation v where in addition all datoms that appear positively in ω_1 have the truth valuation T, and all datoms that appear negatively in ω_1 have the truth valuation F. Therefore when U_1 is

applied to any model where ϕ_1 is true, an alternative world is produced where v is true. Therefore if $\phi_1 \wedge \neg\phi_2$ is true in $\mathcal{M}$, for U_1 and U_2 to be equivalent, v must be true in $\mathcal{M}$, and there must be only one such valuation v. The proof of necessity of condition (3) is symmetric.

We now turn to the reverse implication, namely, that if conditions (1) through (3) are met, then U_1 and U_2 are equivalent. Let $\mathcal{T}$ be an extended relational theory without strict axioms, and let $\mathcal{M}$ be a model of $\mathcal{T}$. If $\neg\phi_1 \wedge \neg\phi_2$ is true in $\mathcal{M}$, then $\text{Worlds}(U_1(\mathcal{M})) = \text{Worlds}(U_2(\mathcal{M}))$. If $\phi_1 \wedge \phi_2$ is true in $\mathcal{M}$, then since U_3 and U_4 are equivalent, again U_1 and U_2 must be equivalent with respect to $\mathcal{M}$. If $\phi_1 \wedge \neg\phi_2$ is true in $\mathcal{M}$, then $\text{Worlds}(U_2(\mathcal{M})) = \{\mathcal{M}\}$. any model produced by U_2 represents the same alternative world as $\mathcal{M}$ does. By condition (2), the same is true of $U_1(\mathcal{M})$. We conclude that U_1 and U_2 are equivalent when applied to $\mathcal{T}$. $\Diamond$

We now present simple sufficient criteria for update equivalence:

Theorem 8-15. Let U_1 and U_2 be two null-free updates under the truth-maintenance semantics:

U_1: `INSERT` ω_1 `WHERE` ϕ,

U_2: `INSERT` ω_2 `WHERE` ϕ.

If

(1) ω_1 and ω_2 are logically equivalent;

(2) The same datoms appear in ω_1 and ω_2;

(3) The same datoms appear positively in ω_1 and ω_2; and

(4) The same datoms appear negatively in ω_1 and ω_2,

then U_1 and U_2 are equivalent.

Proof of Theorem 8-15. Assume that ω_1, and therefore ω_2, is satisfiable, as otherwise the theorem follows immediately. For any extended relational theory $\mathcal{T}$ without strict axioms, consider the effects of U_1 and U_2 on a model $\mathcal{M}$ of $\mathcal{T}$. U_1 must produce a model $\mathcal{M}'$ from $\mathcal{M}$, since ω_1 is satisfiable. We wish to show that $\mathcal{M}'$ is also a model produced by U_2 acting on $\mathcal{M}$.

First, ω_2 must be true in $\mathcal{M}'$, because ω_1 and ω_2 are logically equivalent; therefore rule 2 in the definition of `INSERT` is satisfied for U_2 by $\mathcal{M}'$. By condition (2), rule 1 is satisfied for U_2 by $\mathcal{M}'$. Conditions (3) and (4) guarantee that rules 1a and 1b are satisfied. $\Diamond$

To see that the criteria of Theorem 8-12 are sufficient but not necessary, consider the two equivalent insertions of `INSERT` g `WHERE` T and `INSERT` $g \vee (\mathsf{F} \wedge \neg g)$ `WHERE` T, which do not satisfy condition (3) for equivalence. For necessary and sufficient criteria for two updates to be equivalent, as with the standard semantics

we need the concept of a *reduced* update; but the truth–maintenance reduction process will include additional steps beyond that for the standard semantics.

Definition. Let U be the update INSERT ω WHERE ϕ under the truth–maintenance semantics. To *reduce* ω *with respect to* ϕ, written red(ω, ϕ), first put ω into disjunctive normal form† and then make the following atom substitutions for every datom g in ω:

1. If ϕ and ω both logically entail g, replace g by T in ω.
2. If ϕ and ω both logically entail $\neg g$, replace g by F in ω.
3. If ϕ logically entails g and g appears positively in ω, replace g by T in ω.
4. If ϕ logically entails $\neg g$ and g appears negatively in ω, replace g by F in ω. ◊

Proposition 8-7. Under the truth–maintenance semantics, any update U: INSERT ω WHERE ϕ is equivalent to INSERT red(ω, ϕ) WHERE ϕ. ◊

We omit the proof of Proposition 8-5 here; intuitively, U cannot change the truth valuation of any datom g that is removed from ω, because of rules 1a and 1b in the definition of INSERT under the truth–maintenance semantics (Chapter 7).

Unfortunately, even this stronger version of reduction does not lead to as elegant a theorem of equivalence as was possible under the standard semantics. A counterexample will illustrate why Theorem 8-10 fails to hold for the truth–maintenance semantics.

Example. Let f and g be datoms. Then U_1: INSERT $g \vee \neg f \vee T$ WHERE $(f \wedge g) \vee (\neg f \wedge \neg g)$ is not equivalent to U_2: INSERT $f \vee g \vee \neg f \vee \neg g$ WHERE $(f \wedge g) \vee (\neg f \wedge \neg g)$, even though Theorem 8-10 would suggest so, if co-opted for the truth–maintenance semantics. The problem arises because even though ω_1 and ω_2 are logically equivalent, U_1 will never produce an alternative world where $f \wedge \neg g$ is true, but U_2 will. If Theorem 8-10 is strengthened to require that the same datoms appear positively and negatively in ω_1 and ω_2, then inequivalence would correctly be predicted for this example; but in general, this extra condition is too strong. For example, consider INSERT $g \vee T$ WHERE $\neg g$ and INSERT $g \vee \neg g$ WHERE $\neg g$. These two updates are already reduced, and they are equivalent although g appears positively in ω_1 but not in ω_2. ◊

Theorem 8-16. Let U_1 and U_2 be two null–free updates under the truth–maintenance semantics, where ϕ is in disjunctive normal form:

† For the truth–maintenance semantics, we need slightly more rigid rules on what constitutes disjunctive normal form: Add the requirement that if F appears in a disjunct of a wff in disjunctive normal form, then F is the only conjunct in that disjunct.

U_1: INSERT ω_1 WHERE ϕ,

U_2: INSERT ω_2 WHERE ϕ.

Then U_1 and U_2 are equivalent iff for all satisfiable disjuncts D of ϕ,

(1) red(ω_1, D) and red(ω_2, D) contain the same datoms and are logically equivalent; and

(2) If a datom g appears positively (resp. negatively) in only one of red(ω_1, D) and red(ω_2, D), then g appears negatively (resp. positively) in D. ◊

Example. Consider the counterexample given earlier:

U_1: INSERT $g \vee \neg f \vee \mathsf{T}$ WHERE $(f \wedge g) \vee (\neg f \wedge \neg g)$,

U_2: INSERT $f \vee g \vee \neg f \vee \neg g$ WHERE $(f \wedge g) \vee (\neg f \wedge \neg g)$.

Reduction with respect to $f \wedge g$ yields red(ω_1, $f \wedge g$) = $(\mathsf{T} \vee \neg f \vee \mathsf{T})$ and red(ω_2, $f \wedge g$) = $(f \vee g \vee \neg f \vee \neg g)$. As these two wffs do not contain the same datoms, Theorem 8-16 correctly predicts inequivalence. ◊

Example. Consider the other counterexample given earlier:

U_1: INSERT $g \vee \mathsf{T}$ WHERE $\neg g$,

U_2: INSERT $g \vee \neg g$ WHERE $\neg g$.

These two updates are already reduced. As ω_1 and ω_2 are logically equivalent and contain the same datoms, condition (1) for equivalence is satisfied. For condition (2), g appears positively in ω_1 and not in ω_2, but g does appear negatively in ϕ, so condition (2) is satisfied. Therefore Theorem 8-16 correctly predicts equivalence of U_1 and U_2. ◊

In the spectrum of choices of semantics, the truth–maintenance semantics falls somewhere between the standard semantics and the minimal–change semantics. It is interesting to note that same intermediate nature in Theorem 8-16, which falls between Theorems 8-10 and 8-12 in its requirements. For example, as for the minimal–change semantics, reduction in Theorem 8-16 must be done with respect to the individual disjuncts of ϕ rather than all of ϕ at once. The syntactic element in the standard semantics crops up in the requirement of Theorem 8-16 that red(ω_1, D) and red(ω_2, D) contain the same datoms.

Proof of Theorem 8-16. If ϕ is false in a model $\mathcal{M}$, then $U_1(\mathcal{M}) = U_2(\mathcal{M})$. If ϕ is true in $\mathcal{M}$, then some disjunct D of ϕ is true in $\mathcal{M}$. Let U_1' be the update INSERT red(ω_1, D) WHERE D, and let U_2' be the update INSERT red(ω_2, D) WHERE D. By principle P2 and Proposition 8-7, Worlds($U_1'(\mathcal{M})$) = Worlds($U_1(\mathcal{M})$), and similarly for U_2. If U_1 and U_2 are equivalent, then, U_1' and U_2' must be equivalent. Conversely, if U_1 and U_2 are not equivalent, there must be some disjunct D of ϕ such that U_1' and U_2' are not equivalent.

We now turn to the question of equivalence for the updates U_1' and U_2'. We first show that ω_1' and ω_2' (i.e., red(ω_1, D) and red(ω_2, D)) must be logically

equivalent if U_1' and U_2' are equivalent. Let v be a particular truth valuation that satisfies $\omega_1' \wedge \neg\omega_2'$, say. Let u be a truth valuation that satisfies D and agrees with v on the datoms in v that are not also in D. Let $\mathcal{M}$ be a model of an extended relational theory without strict axioms, having body u. Then v is true in some model $\mathcal{M}' \in U_1'(\mathcal{M})$; the definition of reduction ensures that rules 1a and 1b in the truth-maintenance definition of INSERT are satisfied by v. But World($\mathcal{M}'$) cannot be a member of Worlds($U_2'(\mathcal{M})$), because ω_2 is false in $\mathcal{M}'$. Therefore ω_1' and ω_2' must be logically equivalent for U_1' and U_2' to be equivalent.

We now show that ω_1' and ω_2' must contain the same datoms if U_1' and U_2' are equivalent. If g is a datom that is a subformula of ω_1' but not of ω_2', and if g does not occur in D, then U_1' and U_2' cannot be equivalent, as by the definition of reduction, U_1' can change the truth valuation of g but U_2' cannot. If g does occur in D, then by the definition of reduction, there exists a truth valuation u for the datoms of ϕ and ω_1, such that D is true under u, and also a truth valuation v for the datoms of ω_1', such that that ω_1' is true under v; and where in addition g has different truth valuations in u and v. Let $\mathcal{M}$ be a model of an extended relational theory without strict axioms, having body u; then Worlds($U_1'(\mathcal{M})$) includes an alternative world where v is true, because by the definition of reduction, v satisfies rules 1a and 1b in the truth-maintenance definition of INSERT. We conclude that ω_1' and ω_2' must contain the same datoms if U_1' and U_2' are equivalent.

We now show that condition (2) is necessary. Suppose g is a datom that occurs positively in ω_1' but not in ω_2', and g does not occur negatively in D. Let $\mathcal{M}$ be a model of an extended relational theory without strict axioms, having body $D \wedge g$. Then g is true in every model of $U_1'(\mathcal{M})$. There must be some model $\mathcal{M}'$ of $U_2'(\mathcal{M})$ in which g is false, because $\neg g$ is a conjunct of some disjunct of ω_2' by assumption, and by the definition of disjunctive normal form, that disjunct must be satisfiable. But then U_1' and U_2' cannot be equivalent. The proof is symmetric if g occurs negatively in fred(ω_1, D).

To show that these conditions are sufficient for U_1' and U_2' to be equivalent, suppose ω_1' and ω_2' contain the same datoms and are logically equivalent. Let $\mathcal{M}$ be a model of an extended relational theory $\mathcal{T}$ without strict axioms. If D is false in $\mathcal{M}$, then U_1' and U_2' are equivalent. Otherwise, for any model $\mathcal{M}'$ in $U_1'(\mathcal{M})$, ω_2' is true in $\mathcal{M}'$, because ω_1' and ω_2' are logically equivalent. Therefore rule 2 in the truth-maintenance definition of INSERT is satisfied for U_2'. Rule 1 is satisfied by assumption. For rule 1a, suppose that g is true in $\mathcal{M}$, appears positively in ω_2', but is false in $\mathcal{M}'$. If g appears in D, then by the definition of reduction, g is not a datom of ω_2', a contradiction. If g does not appear in D, then by condition (2) g also appears positively in red(ω_1', D), a contradiction. The proof is symmetric for rule 1b. We conclude that U_1' and U_2' are equivalent. $\Diamond$

After proving Theorems 8-14 and 8-16, the author was led to the conclusion that the truth-maintenance semantics would not be the most fruitful paradigm for investigation, due to its complexity.

8.6. Summary and Conclusion

This chapter has shown that it is possible to develop necessary and sufficient conditions to determine when two updates will be equivalent, under a variety of choices of semantics. Further, a number of basic principles regarding equivalence are shared by a broad class of semantics, as illustrated by the theorems in Section 8-1. Among the semantics examined in detail, the conditions for equivalence are most simple for the standard semantics, due to its name–dropping element, and are a bit more complicated for the minimal–change semantics. These theorems on equivalence are useful when debating the merits of different candidate semantics for an application.

Chapter 9: Implementation

This chapter describes an implementation constructed for the Update Algorithm, and gives experimental results from this implementation. The goal of the implementation effort was to gauge the expected performance of the update and query processing algorithms in a traditional database management system application. The implementation was tailored to this environment, and for that reason the techniques used and results obtained will apply only partially, if at all, to other application environments, such as knowledge–based artificial intelligence applications. In particular, the following assumptions and restrictions were made.

- Update syntax was modified and restricted, to encourage use of simple constructs.
- A fixed data access mechanism (query language) was assumed.
- A large, disk–resident database supplying storage for the body of the extended relational theory was assumed.
- Performance was equated with the number of disk accesses required to perform queries and updates after a long series of updates, and the storage space required after a long series of updates.

These assumptions and restrictions are all appropriate to traditional database management scenarios; they will be discussed in more detail in later sections. We begin with a brief high–level description of the implemented system, and then examine its components in more detail. The chapter concludes with a description of the experimental results.

9.1. Overview

The Update Algorithm Version II was chosen for implementation. This version of the Update Algorithm permits both null values[†] and variables to occur in update requests. Since we can assume that the parameters of the average case are known in advance in a traditional database management system, it is possible to gear the implementation of the Update Algorithm toward this expected case, rather than orienting the implementation toward the worst case as was done in the presentation of the Update Algorithm in Chapters 3 and 4. Orienting the implementation toward the average case allowed us to greatly optimize the algorithm to improve performance during update processing. A query processor was also constructed;

[†] Skolem constants, in logical parlance; in this chapter we will use the non-logical term.

because the expected case is also known during traditional query processing, the query processing algorithms were also thoroughly optimized. Both the query and the update processing routines can make use, when necessary, of a heuristic satisfiability tester to help optimize performance. The implementation is coded entirely in C, runs on a VAX, and is approximately 121 Kbytes long in executable form. We do not include this code here; those parties interested in more details of the implementation than are presented here are invited to contact the author directly.

Lazy evaluation was not implemented; for that reason, to keep the size of the extended relational theory within reasonable limits, null values were not permitted to occur in attributes on which joins were performed in the selection clauses of updates.

The exact pattern of the data, and the individual queries and updates was determined using random numbers and probability distributions, as described in Section 9.4. Updates and queries are modeled chiefly in terms of their *selectivity* rather than their syntax. In other words, because the goal of the implementation is a count of the disk accesses required for processing, the exact syntax of a selection clause is unimportant; what matters is how many disk accesses are required to process that selection clause. Profiles for updates and queries were chosen on the basis of selectivity classes rather than on the basis of syntactic features such as numbers of disjuncts and conjuncts. For example, all selection clauses that require accessing 10 datoms of $\mathcal{T}$ are identical for the purposes of performance measurement, whether those selection clauses contain just single datoms or conjuncts and disjuncts galore.

The implemented version of update syntax differs from that presented in Chapter 3. The goal of the modifications was to tailor syntax to the operations expected to be most common in ordinary database management systems. This decision is expected to have the side effect of mildly discouraging the use of less common (and presumably harder to perform) forms of update requests. The restricted syntax does, however, have the same expressive power as the original syntax; some changes to the extended relational theory that could be accomplished in one update may, however, now require multiple updates. The exact restrictions on syntax are described in Section 9.5.

9.2. Data Structures and Access Techniques for Storing Extended Relational Theories

The extended relational theory is mapped into a set of data structures for storage on disk. The data structures required fall into five general categories:

- Datom space.
- History atom space.
- Equality atom space.
- Logical relationship space.

Body of $\mathcal{T}$: $R(a) \vee R(b)$, $R(a) \vee R(\epsilon)$.

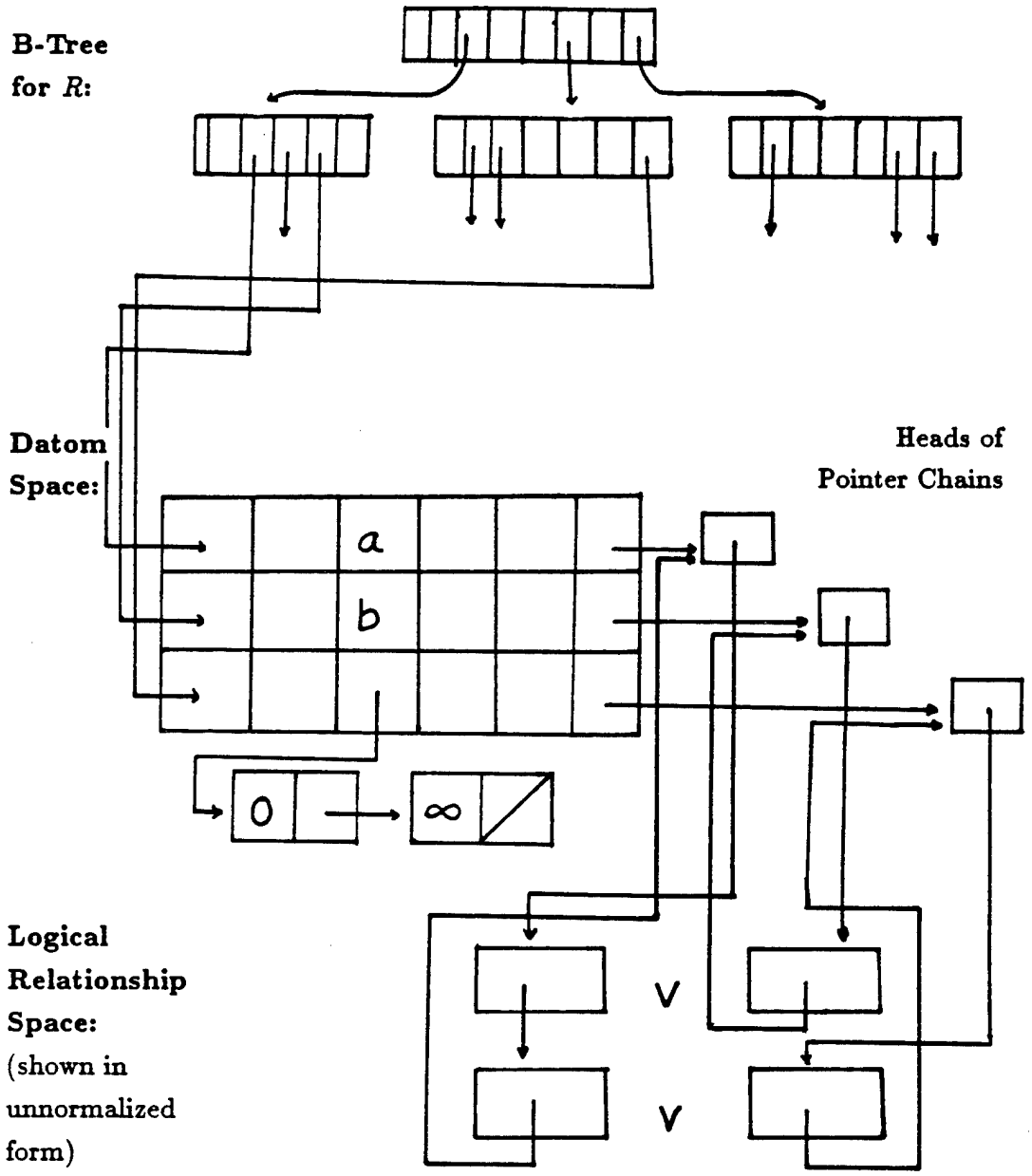


Figure 9-1. Datom and logical relationship space.

- Data structures for satisfiability testing.

The unique name axioms and completion axioms are implicitly present and are not stored. Figure 9-1 shows a simplified version of these data structures for the extended relational theory with body $R(a) \vee R(\epsilon)$.

In the following four subsections we give a high–level overview of each type of data structure. The final subsection gives the full details of the data structures.

9.2.1. Datom Space

Datom space is organized much as ordinary database tuple storage, with datoms packed into disk blocks and accessed using B–tree indices on their attributes. In fact, to obtain the running time estimates given in Chapters 3 and 4, all datoms in the body of an extended relational theory $\mathcal{T}$ must have a lookup and insertion time of $\mathcal{O}(\log R)$, where R is the maximum number of datoms in $\mathcal{T}$ over the same predicate. The predicates are not assumed to have keys.

9.2.2. History Atom Space

As presented in Chapters 3 and 4, the Update Algorithm makes heavy use of history atoms. But history atoms are not strictly necessary; there is no reason to use them if there is an equivalent method of performing a particular update that uses less space than would be required under the history atom method. For this reason, the implementation of the Update Algorithm only makes use of history atoms when it is difficult or impossible to get along without them. For example, the "typical" update in practice is expected to have a very simple selection clause, typically one that is true in all alternative worlds. If ω is also simple, one can almost always "update in place", and no datoms need be replaced by history atoms—Steps 2 and 4 become superfluous.

History atoms are stored separately from datoms. Much less information must be stored for a history atom than for a datom, because little is important about a history atom except its unique ID. In particular, its attribute values are only important insomuch as they determine which other history atoms that atom unifies with; and if there are no unifications with other atoms, then nothing need be stored for that atom other than its unique ID. Further, the set of atoms with which a history atom unifies is fixed at the time an update is performed. Since we expect few unifications in practice, the implementation reduces all history atoms $H(f,U)$ to unique IDs $h_1, h_2, \ldots$ (predicate constants, in the language of mathematical logic). If some history atom h_1 unifies with another atom h_2 under most general substitution σ, then the additional formula $\sigma \rightarrow (h_1 \leftrightarrow h_2)$ must also be stored in $\mathcal{T}$. This simplification of history atoms is expected to reduce the size of $\mathcal{T}$ greatly, as history atom unifications will be rare.

9.2.3. Equality Atom Space

Equality atoms that are true in all alternative worlds have special data structures dedicated to them. Restrictions on the possible values of a Skolem constant are stored in the same disk block as one of the datoms in which that null value occurs (its *home* datom). In addition, if the null value is known to be equal to any other null values (e.g., $\epsilon_1{=}\epsilon_2$), the data structure for each datom in which it occurs includes a pointer to a list of those other null values.

9.2.4. Logical Relationship Space

An *outside logical relationship* of a datom f is a wff in the body of $\mathcal{T}$ that contains f *and* other atoms. For example, if f only occurs as a separate formula in the body of $\mathcal{T}$, then f does not take part in any outside logical relationships. If the formula $f \vee \alpha$ occurs in the body of $\mathcal{T}$, however, then f participates in an outside logical relationship.

The history substitution step (Step 2) is the bottleneck for the Update Algorithm. To make renaming fast and achieve the time bounds presented in Chapters 3 and 4, all occurrences of a datom f in the body of $\mathcal{T}$ must be represented on disk by pointers to a block of storage where f is actually kept, so that the substitution of $H(f,U)$ for f, if required, can be done in constant time. In other words, rather than storing the wffs of the body of $\mathcal{T}$ directly, the wffs are mapped into a data structure that contains pointers into a separate name space where names of datoms are kept—datom space.

All occurrences of the same atom or history atom in logical relationship space are linked together in a chain whose head is either an index entry for a datom or a history atom unique ID.

To facilitate satisfiability testing, logical relationships are not stored as they would appear in the body of $\mathcal{T}$, but rather are converted into a normal form that is convenient for satisfiability testing. This normal form uses only one logical operation, a variant on exclusive–or called *exact–or*. When normalized, the body of the extended relational theory is just a list of *alternative sets*, or sets of pointers to atoms, related by exact–or. If a set of atoms is related by exact–or, then exactly one of those atoms must be true in any model of the extended relational theory. Exact–or differs from exclusive–or in that exact–or is not associative—not a proper operator at all. It is extremely easy to write a heuristic satisfiability tester (described in Section 9.2.4) that works on alternative sets. Such a tester can be captured in a page of C code, unlike a satisfiability tester for formulas containing $\wedge$, $\vee$, and $\neg$. The potential pitfall of using alternative sets is that like any other normal form, conversion to alternative sets may exponentially increase the length of a formula in the worst case.

9.2.5. Data Structures for Satisfiability Testing

A heuristic satisfiability tester is an important part of an efficient implementation of the Update Algorithm. "Heuristic" means that when testing satisfiability of a wff α, in addition to the obvious responses of "satisfiable" and "unsatisfiable", the satisfiability tester may decide that it's too hard to tell whether α is satisfiable, and respond accordingly. This satisfiability tester is guaranteed to stop in a polynomial number of steps (that is, polynomial in the number of stored atoms).

To test satisfiability efficiently, once a decision has been made on the truth valuation of an atom f, all other occurrences of f in the body of $\mathcal{T}$ must be located quickly. To achieve this, in the implementation all occurrences of the same atom in the body of $\mathcal{T}$ are linked together in a list whose head is an index entry.

The other data structure needed for heuristic satisfiability testing is an array of bits to keep track of the decisions made so far on atom truth valuations.

9.2.6. Details of Data Structures

In agreement with traditional relational database terminology, in this discussion the arguments to a predicate are generically termed *attributes* and the values for those attributes in a particular datom are called *attribute values*.

For an efficient implementation, much more must be stored on disk for each datom than just its attribute values. The following data structure description shows what data structure fields our implementation stores for a three–attribute datom; there are 13 fields, the last three of which contain the attribute value information for the datom.

1. Datom ID(s).
2. Does this datom contain null values or participate in outside logical relationships? (Yes/No)
3. Does this datom participate in outside logical relationships? (Yes/No)

4–6. Is there a Skolem constant for the attribute value of attribute 1–3? (Yes/No for each attribute)

7–9. Pointers to MarkLists for attributes 1–3.

10. Pointer to AltSetList.

11–13. Attribute value or pointer to null value for attributes 1–3.

The data structure for datoms contains pointers to MarkLists, AltSetLists, and null values. Let us first describe the data structures used for null values.

If an `INSERT` request is received for a datom that includes a null value as an attribute value, then that field is so flagged in a header for the datom, and in place of the attribute value a pointer is stored to a null value data structure in the same block of disk storage. The null value data structure consists of an ordered list of begin/end range values, with a provision for open ranges. This gives a reasonable simulation of Skolem constants in a variety of domains (e.g., strings, integers, reals). The implementation uses a linked list, but there might be a better choice; the actual data structure is not important for performance measurement, as long as one can get the null value information during the same disk access as the rest of the datom.

MarkLists are lists of the equality atoms in which a Skolem constant occurs. "Marked nulls" is the traditional name in the database community literature for the case when two null values are known to be equal to one another. The actual data structure used is a header followed by a linked list of datom IDs and attribute numbers. The performance measurements assume that each MarkList can be fully contained on a block of disk storage.

One of the drawbacks of using alternative sets of atoms are that alternative sets are very fussy about only one of their member atoms being true in any single

alternative world. This causes a lot of pain in the case where one wants to express concepts such as disjunction. Since the implementation of alternative sets lists member datoms and history atoms by atom ID, to implement disjunctions the representation cheats by using multiple datom IDs for the same datom. This was easy to implement; due to the presence of null values and the lack of keys, the implementation already had to provide for more than one datom per index value.

The implemented AltSetList looks like a MarkList. The main difference between them is that some atom IDs in an AltSetList may be over a special predicate, the history predicate. These history atom unique IDs occur plentifully when alternative set normal form is used. A history atom unique ID is just an index into the HistoryAtomArray; in that array a list is stored for each history atom, containing pointers to the alternative sets where that history atom appears.

Associated with the AltSetLists are a few extra bits to help the satisfiability testing routine remember which truth valuations have been decided so far. The bits are arranged so that the satisfiability tester would not have to hop all over disk to turn those bits off after the testing is done; they are kept together in one array, hashed on datom and history atom ID. This array is to be loaded into main memory when the database is first opened.

The HistoryAtomArray fits on as many contiguous blocks of disk as its length requires. Its storage is managed by a manager that keeps track of free slots. Each slot represents a different history atom, and contains a pointer off to a list of the alternative sets that that history atom occurs in.

9.3. Implementation of the Update Algorithm

The Update Algorithm as implemented does not look like very much like its presentation in Chapter 4. This is because the presentation in Chapter 4 was geared toward streamlined handling of the general case, that is, the worst case. In contrast, the implemented version is geared toward streamlined handling of the expected case. The "typical" datom in the extended relational theory is expected to be true in all alternative worlds, and hence the query and update algorithms are oriented heavily toward dealing with datoms that are true (or false) in all alternative worlds.

This orientation leads to the use of a hierarchy of update processing routines. At the top of the hierarchy are procedures that work correctly when datoms do not contain null values and are not involved in any outside logical relationships. At the lowest level are routines that know how to handle arbitrary outside logical relationships. The implemented version of the Update Algorithm operates at all times in the highest possible level of this uncertainty hierarchy. A simplified version of the hierarchy for ground requests follows.

1. All atoms involved in this query/update are true in all alternative worlds, so process this request as though in an ordinary database.

2. There are null values in one of the atoms involved in this request, but the null values are not relevant to this particular request, so they can be ignored. Further, the atoms are not involved in any outside logical relationships.

3. There are null values in one of the atoms involved in this request, but the atom and its null values are not involved in any outside logical relationships, so the uncertainty can be dealt with locally.

4. Some atoms or null values of the request are involved in outside logical relationships, and a heuristic satisfiability tester needs to be called before any updates are performed or the query answer is returned.

The determination of the correct level of the hierarchy is done as rapidly as possible; dedicated fields in the stored datom are maintained to give that information. The determination of the correct level of the hierarchy is done separately for each set of bindings to variables in the update or query.

The hierarchy is organized in accordance with the expected frequency of different types of uncertainty in the extended relational theory. For example, null values are expected to be the most common type of uncertainty, and null values are expected to be less frequent in the "important" attributes (i.e., those appearing in joins or equality atoms in ϕ and ω). For that reason, the implementation is optimized to work most efficiently at higher levels of the hierarchy. In particular, if no uncertainty is present at all, then queries and updates will be processed as fast as though the database management system had never heard of uncertainty (except for effects due to the slightly higher space required for tuple storage). Under this school of thought, if one doesn't use the expressive power of the extended relational theory, then one needn't pay for it.

Conceptually, a practical implementation of the Update Algorithm will begin by instantiating the variables of the update or query request U, attempting to satisfy the selection clause ϕ of U. The process of instantiation will be guided by the use of safe selection clauses, construed in this implementation to mean roughly that each instantiation of a variable should be a most general choice that will lead a datom of ϕ to unify with a datom already in the extended relational theory. The instantiation process stops as soon as ϕ is satisfied and all variables in ω are bound. As an example optimization used in the implementation, at this point the bound version ϕ' of ϕ is minimized in length. For example, all atoms in ϕ' that are known to be true (resp. false) in all alternative worlds are replaced by the truth value T (resp. F). In the average case, ϕ' will be reduced to T if the request is an insertion. In the worst case, ϕ' should be reduced to a conjunction of a very small number of literals, say no more than three. This important minimization reduces the number of atoms that must be added to $\mathcal{T}$ to perform U.

9.4. Data

This section describes the data used as input to the performance measurement runs.

For performance measurement, an extended relational theory containing "real" data (e.g., employees, managers, and departments) could not give sufficiently empirical results. For example, what would constitute a "representative"

set of queries and updates? Therefore for performance measurement, the important parameters of individual queries and updates are chosen randomly according to pre-specified probability distributions (described below). The individual queries and updates are reduced to a set of statistical profiles, so that datoms are selected to satisfy selection clauses according to the dictates of probability distributions.

A simple approach to queries is to divide query answers into three categories: sets of datoms known to satisfy the query, sets of datoms known to satisfy the query in some alternative worlds but not in others, and sets of datoms that may possibly satisfy the query. The latter class consists of those sets of datoms for which the heuristic satisfiability tester was unable to reach a conclusion on theoremhood.

The extended relational theory $\mathcal{T}$ contains three database predicates, each with three attributes. Indexes are stored for all three attributes. At the beginning of a run, all three database predicates have the same number of datoms occurring in $\mathcal{T}$; the exact number is a parameter set at the beginning of the run, typically 1000 or 200. Non-Skolem–constant datom attribute values are distributed uniformly over an infinite range. The chance of null values occurring as attribute values in datoms, both initially and when datoms are added using `INSERT` and `MODIFY`, is controlled by probabilities selected at the beginning of a performance run. These probabilities control the number of introduced datoms having zero, one, and two null values as attribute values. In addition, the type of range restrictions, if any, on null values at the time of their home datom insertion is controlled by probabilities set at the beginning of a performance run. Null values can either be unrestricted, meaning they can take on any value in the underlying attribute domain; or they are restricted to a range, the size of which is chosen uniformly on an interval also selected at run time; or else they are restricted to three values. Of course these restrictions can be altered by subsequent updates.

Disk block size is also a parameter set at run time. Datom size is set to 100 bytes. A block packing factor of 69% (derived from various studies; see [Wiederhold 83]) is assumed.

9.5. Updates and Queries

We first cover the syntax for updates and queries, then look at the method used to generate particular profiles of updates and queries for performance measurement.

Rather than using one single update operation, four operators are made available: `INSERT`, `DELETE`, and `MODIFY` (all discussed in Chapter 3), and also an operation called `ASSERT`, with syntax and semantics as follows:

`ASSERT` ϕ: Eliminate every alternative world of $\mathcal{T}$ that is not a model of $\{\mathcal{T}, \phi\}$.

The mix of the different types of updates and queries is controlled by parameters selected at the start of each performance measurement run.

In the implemented version of update requests, no more than one datom can occur in ω. The wff ω can contain just f, or $f \vee \mathsf{T}$ (written MAYBE(f) for ease of programming), or either of these in conjunction with range restrictions on null values. Further, any null values in f cannot already occur in $\mathcal{T}$. (Assertions can be used later to equate pairs of null values.) More complicated ωs containing additional atoms can be simulated using multiple updates within one transaction.

9.5.1. Selection Clause Profiles

As implemented, processing of every selection clause other than the truth value T begins with a selection phase. The relation and attribute for the initial selection is chosen randomly from a uniform distribution. These selections fall into five different groups, based on the number of datoms they select.

1. One datom is selected via index lookup on attribute value. Any additional datoms with null values that could be equal to that attribute value are included in the result.

2. A small set of datoms is selected via index lookup. A uniform distribution is used to determine the size of the set, which can range between zero and a parameterized upper limit (typically 50 datoms). Any additional datoms with null values that could be equal to that attribute value are included in the result.

3. A percentage of the datoms over a predicate are selected via index lookup. The percentage is selected using an Erlang distribution ($m = 2$, $l = 2$, total = 2.5) that typically selects 10% of the datoms over a predicate. The Erlang distribution (see e.g. [Wiederhold 83]) is often used to model natural phenomena; the graph of its probability distribution starts off at $m = 2$ at a high probability, quickly rises to its maximum, then falls into a long tail. Any additional datoms with null values that could be equal to that attribute value are included in the result.

4. Range selection: All datoms within two delimiting points in an attribute index are selected. Size of range is chosen uniformly from an interval selected at the beginning of the performance run. (A Zipfian distribution [Knuth 73] would have been more appropriate, as discussed below, but was bypassed due to the difficulty of implementing it.) Any additional datoms with null values that could fall within that range are included in the result.

5. A sequential scan is conducted of the datoms in the extended relational theory over some predicate, resulting in the eventual selection of a small set of datoms over that predicate. Again, the size of the set is chosen uniformly from an interval selected at the beginning of the performance run, and any additional datoms with null values that could be equal to that attribute value are included in the result.

The type of the selection clause of the current request is selected at run time using random numbers and expected distributions of selection clause types for queries and for updates. Updates are strongly biased towards selection of individual datoms (type 1) or the truth value T, in accordance with traditional

database updates. (Of course, if every datom in the extended relational theory has a null value for some attribute, then even a type 1 selection clause could return all the datoms in the theory.) Further, no selection clauses of type 3 are allowed in updates, as it is our belief that the number of tuples changed by an update is not a function of the size of the database.

Once the size of the result of a selection has been decided, the actual datoms satisfying the selection must be chosen. The implementation uses a uniform probability distribution to select datoms from the predicates. Choice of predicate for the selection is also made uniformly.

9.5.2. Join Profiles

After the initial selection phase, between zero and two joins are executed. Expected percentages of updates and queries with zero, one, and two joins are chosen at the beginning of the performance measurement run; generally updates are expected to have a high likelihood of having no more than one join. After the initial selection phase, the order of joining of relations is chosen, using a uniform distribution.

9.5.3. Projection

Projection is not modeled, because it is not expected to have a large effect on the comparative disk access costs for extended relational theories and complete–information relational databases.

9.6. Update Implementation Technique

It was our belief that the most economical route in the long run was to minimize the amount of information in logical relationship space, at the expense of datom space. In other words, if there are two ways to represent the result of an update in the extended relational theory, and one way adds more to datom space but less to logical relationship space than does the other, then the former method is preferred. The idea is to have as much information as possible stored in a simple, flat format that will not require use of expensive procedures for analysis. With this goal in mind, the implementation avoids having to store equality atoms by making heavy use of a procedure called *tuple splitting* [Keller 85], described briefly below.

Consider an extended relational theory with body Emp(ϵ, CSD). Suppose an update arrives with selection clause Emp(Reid, CSD). Then, loosely speaking, the datom Emp(ϵ, CSD) satisfies that selection clause in some alternative worlds and not in others. If the update is `INSERT` Mgr(Nilsson, CSD) `WHERE` Emp(Reid, CSD), then the truth valuation of the new datom Mgr(Nilsson, CSD) is going to depend on the value of ϵ. We chose to avoid proliferation of atoms such as ϵ=Reid by *splitting* Emp(ϵ, CSD) into two stored datoms Emp(Reid, CSD) and Emp(ϵ, CSD), where (1) the new datom Emp(ϵ, CSD) has a range restriction that ϵ is not Reid, and (2) the two datoms appear together in an alternative set. Then the

selection clause is satisfied by the datom Emp(Reid, CSD) in all of the alternative worlds where Emp(Reid, CSD) is true, and is not satisfied by the other stored datom in any world where that datom is true. In the implementation, whenever a datom only satisfies the selection clause of an update in some of the worlds where that datom is true, then that datom is split until this is no longer the case. When a datom is split, its alternative sets must be changed, and also all tuples on its mark list may require splitting to preserve the alternative worlds of the extended relational theory. We prefer not to present the details of this process, as it is quite intricate.

9.7. Experimental Results and Discussion

In this section we describe the behavior of the implementation with respect to two parameters: disk accesses required to execute a set of queries after a certain number of updates has been completed, and size of relations (i.e., number of stored datoms over the same predicate) after a series of updates. We first examine relation size, then disk access count, and then give some general comments.

9.7.1. Relation Size

As described earlier, the input to a simulation run consists of three relations/predicates, each with three attributes, and each with the same number of stored datoms. Each stored datom is known to be true in all alternative worlds at the beginning of the run. Then a long series of hundreds or thousands of updates is applied while the size of the three relations is monitored. Most of our runs used an initial relation size of 200 datoms; experiments were also performed with initial sizes of 1000 and 20. These runs were all very interesting to watch; a number of phenomena deserve mention. First we describe the parameter settings used for this set of runs, summarized in Table 9-1.

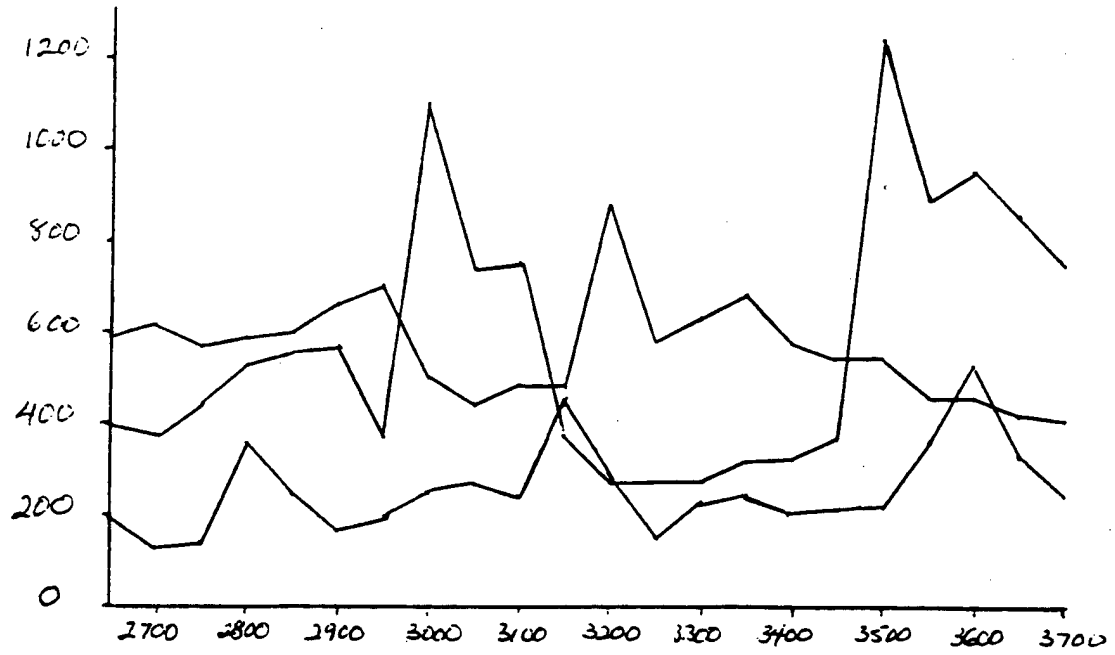
These parameters are intended to model a scenario where 80% of the incoming `INSERT` requests are for datoms that are to be true in all alternative worlds. Of the remaining 20%, one null value appears in 18% of the datom insertion requests, and two null values appear in the remaining 2%. Every inserted datom has some non–null value as an argument, because one attribute is required to be null–free, to permit joins at a reasonable cost for these large relations. In keeping with this 80/20 approach, a `MODIFY` request has an 80% chance of modifying an attribute value to be a constant, and a 20% chance of modifying it to be a null value. Half of the unknown values in inserted datoms are restricted to small sets, containing three possible values initially. These represent inserted datoms like Emp(ϵ, CSD)$\wedge$(ϵ=Reid $\vee$ ϵ=Nilsson). The other half of the inserted null values have unrestricted ranges, meaning that they can assume any value from an infinite domain. Emp(ϵ, CSD) is an example of this type of unrestricted insertion.

The breakdown of update types was 40% `MODIFY` requests and 20% each `INSERT`, `DELETE`, and `ASSERT` requests. The sensitivity of our results to the value of the `ASSERT` parameter will be discussed below.

200	Number of datoms in each relation at start of run
20	Upper bound on "small set" size for type 2 selection clauses
.20	Percentage of updates that are insertions
.40	Percentage of updates that are modifications
.20	Percentage of updates that are deletions
.20	Percentage of updates that are assertions
.20	Percentage of modifications that introduce set nulls
.80	Percentage of type 1 selection clauses for insertions
.20	Percentage of type 2 selection clauses for insertions
.58	Percentage of type 1 selection clauses for modifications
.34	Percentage of type 2 selection clauses for modifications
.00	Percentage of type 3 selection clauses for modifications
.06	Percentage of type 4 selection clauses for modifications
.02	Percentage of type 5 selection clauses for modifications
.75	Percentage of type 1 selection clauses for deletions
.15	Percentage of type 2 selection clauses for deletions
.00	Percentage of type 3 selection clauses for deletions
.05	Percentage of type 4 selection clauses for deletions
.05	Percentage of type 5 selection clauses for deletions
.20	Percentage of type 1 selection clauses for queries
.20	Percentage of type 2 selection clauses for queries
.20	Percentage of type 3 selection clauses for queries
.20	Percentage of type 4 selection clauses for queries
.20	Percentage of type 5 selection clauses for queries
.40	Percentage of queries with no joins in the selection clause
.35	Percentage of queries with one join in the selection clause
.25	Percentage of queries with two joins in the selection clause
.50	Chance of an inserted set null having an unrestricted domain
.80	Chance of an inserted tuple having no null values
.18	Chance of an inserted tuple having one null value
.02	Chance of an inserted tuple having two null values

Table 9-1. Major input parameters for a series of runs.

Figures 9-2 and 9-3 show the number of stored datoms for each of the three relations over a long series of updates, taken from a run with an initial relation size of 200 datoms. Figure 9-2 gives a close–up view of the behavior of the relations between updates 2700 and 3700. The starting and stopping points were taken at random from a run of over 6000 updates. Figure 9-3 takes a longer–term view, covering updates 2500 through 5250. These figures bring out two important points about this typical run:



X axis: number of updates; Y axis: relation size.

Figure 9-2. Relation size after a series of updates.

X axis: number of updates; Y axis: relation size.

Figure 9-3. Relation size after a series of updates.

- Relation size does not increase with time.
- Relation size has a high variance.

The sudden, dramatic rises in relation size, followed immediately by major collapses, made this run and its brethren very exciting to watch. What causes those dramatic peaks? Over such a long series of updates, events with low probability of occurring at any one moment have a high probability of occurring at some point. Those peaks are caused by repeated splits of datoms like Emp(ϵ, CSD): a long series of updates all have selection clauses that unify with Emp(ϵ, CSD), and that datom is split again and again, causing a sudden explosion in the size of the alternative sets of the datom. Then the law of averages takes effect: an ASSERT request establishes that a datom in one of those large alternative sets is true in all alternative worlds, and the entire huge alternative set vanishes with that ASSERT request. Graphs on a larger scale than those of Figures 9-2 and 9-3 would show that such explosions typically take place within a short series of perhaps 30 updates, and vanish even more quickly.

Of course any practical implementation of this theory would need to prevent sudden bursts in relation size: the growth and collapse use a lot of resources. The obvious means would be a limit on the number of times any one tuple can be split before more complete information on its null values is required.

Sudden collapses in relation size, other than those following a sudden burst in size, are very rare. This is because the base size of the relation—its size when not in a dramatic peak—is determined not by datoms that have been split many times, but by datoms that are either true in all alternative worlds or at most have been split just a few times. There is no way to delete many of these datoms within a few updates, because the number of datoms selected by the DELETE operator is not a function of relation size.

Figures 9-2 and 9-3 do not show the initial growth of the relations from 200 to their eventual base size of approximately 400 datoms. Recall that at the beginning of a simulation run, all datoms are known to be true in all alternative worlds. The initial phase of growth lasts for several hundred updates, as the initial datoms in the relations are replaced by datoms that are not so likely to be true in all alternative worlds. As such datoms are likely to be split several times before they are DELETEd or ASSERTed, the long-term expected size of the relation is greater than its initial size.

We had planned to do the simulation runs with much larger initial relation sizes, say 10 000 tuples. However, the VAX did not take kindly to keeping all the data structures for statistics for such large relations. In addition, the random numbers used at every phase of execution gave terrible locality to the datom access patterns. We tested the behavior of relations with initial sizes of between 20 and 5000 tuples to see what relationship held between starting size and eventual size, with the hope that a smaller starting size would suffice. We found that in all cases, the relation size after a long series of updates is a function of the initial relation size, and that for initial sizes over 100 tuples, the relations grew to between 1.5 and 2 times their initial size before stabilizing. For example, an initial

relation size of 1000 tuples grew to an average base size of 2000 tuples after a series of 1500 or more updates. Smaller starting sizes had to grow proportionately more before reaching stability; relations of fewer than 80 tuples stood in danger of being wiped out by an unlucky sequence of ASSERT and DELETE requests. For example, the smallest relation in Figure 9-2 drops down below 100 tuples after update 4500, and to size zero soon thereafter. Even 1500 updates later, that relation still had fewer than 20 datoms. This opened up the possibility that 100 datoms was a size threshold for stability; however, this hypothesis was discredited by a separate simulation run using initial sizes of 20 datoms. In this latter run, the relation size stabilized at a base of 100 tuples. From the group of test runs we conducted, we concluded that simulation runs with an initial relation size of 200 datoms were adequate for our purposes.

The exact pattern of relation size peaks and valleys is highly variable. For example, changing the random number seed, or changing the initial relation size from 200 to 199 or 201 datoms, was found to lead to a very different pattern of growth and shrinkage.

What effect did assertions have in keeping down the size of the relations? A sample run with the assertion routines disabled showed slow, steady growth in the size of the relations, so that an initial relation size of 200 datoms had grown to over 20,000 datoms in the three relations after approximately 850 updates.

When the percentage of ASSERT requests was lowered to 10% by disabling the ASSERT routine half the time, then after 1000 updates, the three relations of initial size 200 had grown to a combined base size of over 3500 tuples. After 1500 updates, the combined base size was over 6000 tuples. Relations of initial size 100 headed towards size infinity at the same steady pace.

The second phase of experimentation involved measuring disk accesses required for a series of queries after a long series of updates. From examination of the pattern of growth and shrinkage, we determined that 1000 updates were sufficient to "randomize" the initial relations fully and to stabilize the relation sizes. The base relation size remained the same from the 1000th update on through the 10,000th, which was the largest number of iterations we tried.

Because relation size was subject to dramatic temporary fluctuations, we did not want to measure the disk access cost of queries at a moment when the relations were at an unrealistic size peak that would not have been permitted in a more practical setting. However, this turned out not to be a problem, as for the test runs we used in measuring disk accesses, the relation sizes were all quite reasonable after exactly 1000 updates.

9.7.2. Disk Access Measurements

This section compares the performance during query processing of an extended relational theory and a complete–information relational database. The first task of such a comparison is to decide what constitutes a fair comparison: what should be the characteristics of the complete–information database? To determine this,

we examined the internal state of an extended relational theory after a long series of updates, in order to determine the approximate size of each of its relations in its alternative worlds by examining the cardinality of its alternative sets. In the process we garnered information about the number and makeup of the alternative sets in an extended relational theory after a long series of updates.

The extended relational theory used in this discussion was generated by applying a series of 1000 updates to relations of initial size 200. This theory was described in the previous section, and the major input parameters for the theory are listed in Table 9-1. At the end of the generation process, the three relations had sizes 485, 600, and 358, respectively. An examination of the alternative sets of the theory showed that an alternative world of this theory would have approximately 225, 199, and 139 tuples, respectively, in its three relations. The largest alternative set found contained 130 datoms, most of them from relation three. This correlated with the findings of test runs on complete–information relations of initial size 200, which showed that the average relation size was still approximately 200 tuples after 1000 updates. We concluded that a fair comparison could be obtained by running the queries on a complete–information relational database with 200 tuples in each relation.

Experimental runs of 100 to 500 queries on this complete–information database showed wide variation—as much as a factor of three—in the seeks, latencies, and block transfers needed, depending on the choice of a random number seed. We traced this problem to the presence of joins whose result was the cartesian product of the two relations. To achieve greater stability, such joins were prohibited. Once this step was taken, disk access requirements were fairly uniform over different choices of random number seeds. We averaged the results from eight typical runs of 100 queries to get complete–information seek, latency, and block transfer totals. These figures are shown in Table 9-2.

Another factor threatened to prevent a realistic comparison. If joins are done on attributes containing unrestricted nulls, then datoms containing nulls on those attributes will match with every datom in the joining relation. The volatility of this type of n^2 join had already been demonstrated in the complete–information case. To avoid spurious comparisons, we chose to restrict joins to the null–free attribute of each of the relations.

As mentioned earlier, we assume that each alternative set and mark list fits on one block of disk storage. When the satisfiability tester is called, it recursively visits all alternative sets that each selected datom appears in, all alternative sets that the atoms in those sets appear in, and so on. Once an alternative set has been visited during a query, it is completely read into memory at that time and remains in memory until the end of that query. Similarly, once a MarkList is referenced it is assumed to remain in memory until the end of the query. At the end of each query, the alternative sets and mark lists are flushed from memory.

The disk access requirements shown in Table 9-2 for the incomplete–information theory are the averages of a set of six runs taken with different random number seeds at query time and otherwise identical input data; the same

	Seeks	Latencies	Disk Block Transfers
Complete Information	7 750	9 440	9 440
Incomplete Information	25 821	28 502	28 502

Table 9-2. Disk accesses during processing of 100 queries.

input data for queries were used as for the complete–information database.

Table 9-2 shows that the presence of incomplete information causes a threefold increase in disk access costs for a series of queries. Examination of the raw data showed that most of the extra cost does not come from accesses to alternative sets and mark lists; rather, the more mundane accessing of datom space records alone more than doubles the average disk access requirements. Since there are twice as many stored datoms in the incomplete–information theory as in the complete–information database, this is not surprising.

9.7.3. General Discussion

When this project began, it was unclear what level of **ASSERT** requests would be required to keep the extended relational theory from growing without bound. We found that 20% assertions provided size stability, and 10% produced slow growth, when 40% of the updates were modifications and the rest were evenly split between insertions and deletions. As for disk access costs, the presence of incomplete information in the database caused an approximate doubling in the number of stored datoms and an approximate threefold increase in disk accesses required during query processing, for the case where joins were not permitted on attributes containing unrestricted null values. This increase seems reasonable, since intuitively the presence of null values will cause many more datoms to appear to satisfy the selection clause of an incoming query.

One unusual feature of the implementation is its use of tuple splitting in an attempt to avoid complicated logical inferences over datoms. It is not clear whether tuple splitting has any advantages as an implementation strategy. On the one hand, tuple splitting made the relation size a clear indicator of the proliferation of uncertainty within the extended relational theory. On the other hand, tuple splitting was responsible for the sudden spurts and drops in relation size. In a practical implementation of this approach, those irregularities in relation size would have to be ironed out by establishing limits on the permissible number of splits. This points out another potential advantage of tuple splitting, in that it is easy to detect the most common situations where uncertain data will have a big impact on processing costs. On the other hand, such record–keeping could probably be incorporated into a more direct implementation of logical relationships as well. Finally, it is not clear to what extent the high disk access requirements for the database were due to the use of tuple splitting.

The implementation uses a uniform probability distribution to select

datoms from a relation during the selection phase of query and update processing. A more realistic model (and one which would lead to more optimistic results) would be to use Zipf's law [Knuth 73] and use a distribution that directed 90% of the updates to 10% of the extended relational theory datoms, 90% of that 90% to 10% of that 10%, and so on; and that directed 80% of the queries to 20% of the extended relational theory datoms, 80% of that 80% to 20% of that 20%, and so on. Zipf's law would improve the in-memory performance of the implementation, because it would tend to localize uncertainties into little clusters. The most expensive processing occurs when a chain of interrelated uncertainties sprawls across the extended relational theory; a Zipfian distribution would tend to keep uncertainty local. Because processing cost may be exponential in the length of the chain of interrelated uncertainty, short localized chains—in particular, chains of guaranteed bounded length—would put a tight cap on the CPU cost of query answering. In particular, if chain lengths are bounded by a constant, then query answering will no longer be NP–complete[†]; the worst–case running time of a query will be polynomial in the size of the extended relational theory. This $\mathcal{O}(1)$ *hypothesis*—the belief that chains will be of bounded length—is a very important point, so let us elaborate on it a bit.

It is not uncertainty per se that makes query processing expensive; in-memory processing only gets expensive when attribute values are uncertain *and* they depend on other uncertain values, which in turn depend on other uncertain values, and so on. For example, it's not really a problem if I don't know your salary, and I don't know the number of orders outstanding in the warehouse, and I don't know who your boss is; it only really becomes a problem if in addition your salary *depends on* who your boss is, and that in turn depends on the number of orders outstanding in the warehouse, and so on. The length of that chain of uncertainty is what determines the cost of query processing. I hypothesize that in "real life," that chain of uncertainty is short: $\mathcal{O}(1)$, i.e., of length bounded by a constant. In other words, your salary is not going to depend on datoms far off in another corner of the extended relational theory. Zipf's law, which has been observed to hold for many natural phenomena, supports the $\mathcal{O}(1)$ hypothesis. Of course there is no formal method to prove or disprove this hypothesis; consider it an argument against entropy in the extended relational theory, where entropy is defined as the Murphy's–Law state of affairs wherein the length of chains of interrelated uncertainties grows as the size of the extended relational theory.

Entropy does have an ally, however. Since people are naturally messy, their extended relational theories will tend to get messy and cluttered with old, irrelevant uncertainties. A certain amount of energy will have to be expended into keeping the extended relational theory clean with ASSERT. Feedback on performance bottlenecks should suffice to motivate periodic clean-ups.

[†] Or co-$\mathcal{NP}$–complete, depending on the exact query language allowed [Vardi 85].

Chapter 10: Open Questions, Summary, and Conclusions

This chapter reviews summarizes the findings of Chapters 3 through 9 and proposes a number of questions for future work.

10.1. Open Questions

Some obvious directions for future work are in the areas of the relaxation of the closed–world assumption, minimization of the size of extended relational theories, the cost of query answering, axiom enforcement, lazy evaluation, and incorporation of other types of incomplete information.

The closed–world assumption. How would the algorithms presented in this thesis differ if the open–world assumption were used rather than a closed–world approach? Essentially, Step 1 of the Update Algorithm can be eliminated, and dependency enforcement becomes more difficult. This question is investigated in some detail in [Winslett 86c], and the reader is referred to that publication for details. Conditions for update equivalence under the open–world assumption have yet to be specified, and there is much more to be said about axiom enforcement in the AI realm, as different types of enforcement are needed there than those natural in the database environment. The problem of axiom enforcement gradually shades into that of revision of beliefs in the face of conflicting evidence.

Minimization of the size of extended relational theories. A more thorough investigation is needed into efficient heuristic techniques for minimizing the size of the formulas added to the theory during updates. Simplification heuristics are vital for efficient execution, and were at the core of the implementation of the Update Algorithm coded by the author.

The cost of query answering. There is more to be said on exactly when answering a query placed to an extended relational theory takes a long time. Of course query answering will be at least as hard as satisfiability testing, and the exact difficulty will depend on the query language allowed [Vardi 85]. An interesting question is how the history predicates affect the complexity. In the worst case, query answering will be $\mathcal{NP}$-hard in the number of history atoms present in the extended relational theory, but we conjecture that the time taken up by computations on history atoms can be bounded by a function of the complexity of the database before those atoms were introduced.

Axiom enforcement. The work on strict enforcement of type and dependency axioms should be extended, so far as is possible, to axioms containing existential quantification, and to different types of enforcement.

Lazy evaluation. There is a good deal more to be said about lazy evaluation of updates, and quite a bit more to be proven about lazy evaluation. For example, refinement of the update cost estimation function of Chapter 5 would be helpful, as would measures of the effectiveness of lazy evaluation for updates with arbitrary selection clauses.

Other types of incomplete information. How might an update paradigm such as those discussed in this thesis be combined with other types of incomplete information? For example, how could this approach be integrated with fuzzy sets [Zadeh 79], probabilistic logic [Nilsson 86], or inapplicable null values [Vassiliou 80, Zaniolo 82]?

10.2. Summary and Conclusions

In this thesis we represent databases containing incomplete information as logical theories, and view the models of the theory as representing possible states of the world that are consistent with all known information. The bodies of these *extended relational theories* allow incomplete information to appear in the form of disjunctions or Skolem constants (a.k.a. null values). Any ground formula may appear in the theory body and, depending on the needs of the application, quantified formulas may be permitted as well. In this latter case, the extended relational theory may be any first–order theory.

We set forth a language and semantics for updates, and a series of algorithms for incorporating updates into the extended relational theory. These Update Algorithms are proven correct in the sense that the alternative worlds produced under the algorithms are the same as those produced by processing the update in each alternative world individually. For updates and theories without Skolem constants, the Update Algorithm has the same asymptotic cost as for an ordinary complete–information database update, but may increase the size of the extended relational theory. For updates involving Skolem constants, the increase in size will be severe if many atomic formulas in the theory unify with those in the update; if desired, a lazy evaluation technique may be used to control expansion.

As a corollary to this work, in the case where quantifiers are allowed to appear in the body of the extended relational theory, the Update Algorithm serves as an efficient means of implementing updates to arbitrary first–order theories under a model–based semantics.

When dependency axioms are added to this framework, additional mechanisms are required to enforce those axioms when updates are processed. For a particular definition of enforcement, we developed mechanisms to handle universally quantified dependency axioms during updates, and incorporated those mechanisms into the Update Algorithm. The cost of enforcement is reasonable for common varieties of axioms, such as functional and multi–valued dependencies.

A simulation program has been constructed for the Update Algorithm, with heavy emphasis on optimization of the algorithms for the expected types of queries and updates in a typical database management system scenario. We

found that for a reasonable pattern of input queries and updates, the size of the extended relational theory fluctuated from moment to moment, but did not grow over time. For this pattern of queries and updates, the disk access requirements of the extended relational theory were found to be three times greater during query processing than for a comparable complete–information relational database.

The techniques used in the Update Algorithm can be used as the basis for efficient processing of updates under a wide range of semantics. We discussed the general class of semantics for which the Update Algorithm approach is helpful, and gave special attention to several interesting choices of semantics. We introduced the concept of update equivalence as one means of investigating the properties of a potential choice of semantics, and provided theorems on update equivalence for a variety of semantics. It is our hope that the use of mathematical logic in this work, and the attempt to free the approach from considerations native to any one application domain, will render these update techniques useful in the future for applications in a wide variety of domains.

In sum we have shown that, first, the concept of a database update can be extended to databases with incomplete information in a natural way; second, that first–order logic is a fruitful paradigm and tool for the investigation of incomplete information; and third, that one may construct an algorithm to perform these updates with a reasonable running time.

Index of Definitions

References

[Abiteboul 85] S. Abiteboul, G. Grahne, "Update Semantics for Incomplete Databases," *Proc. of the Conference on Very Large Data Bases*, Stockholm, August 1985.

[Bancilhon 81] F. Bancilhon and N. Spyratos, "Update Semantics and Relational Views", *ACM Trans. on Database Systems* **6**:4, December 1981.

[Chamberlain 76] D. D. Chamberlain, M. M. Astrahan, K. P. Eswarsn, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade, "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control", *IBM Journal of Research and Development* **20**:6, November 1976.

[Clark 78] K. Clark, "Negation as Failure", *Logic and Databases*, H. Gallaire and J. Minker, eds., Plenum Press, New York, 1978.

[Codd 79] E. F. Codd, "Extending the Relational Data Base Model to Capture More Meaning", *ACM Trans. on Database Systems* **4**:4, December 1979.

[Davidson 81] J. E. Davidson and S. J. Kaplan, "Natural Language Access to Databases: Interpretation of Update Requests", *Proc. of the 7th Int. Joint Conference on Artificial Intelligence,* Los Angeles, April 1984.

[Davidson 84] J. E. Davidson, "A Natural Language Interface for Performing Database Updates", *Proc. of the 1st Data Engineering Conference,* Los Angeles, April 1984.

[Dayal 82] U. Dayal and P. Bernstein, "On the Correct Translation of Update Operations on Relational Views", *ACM Trans. on Database Systems* **7**:3, September 1982.

[DeKleer 85] J. DeKleer, B. C. Williams, "Diagnosing Multiple Faults", *Artificial Intelligence*, in press.

[Doyle 79] J. Doyle, "A Truth Maintenance System," *Artificial Intelligence* **12**, July 1979.

[Enderton 72] H. B. Enderton, *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[Fagin 83] R. Fagin, J. D. Ullman, and M. Y. Vardi, "On the Semantics of Updates in Databases," *Proc. of the 2nd ACM Symposium on Principles of Database Systems*, April 1983.

[Fagin 86] R. Fagin, G. M. Kuper, J. D. Ullman, and M. Y. Vardi, "Updating Logical Databases," in *Advances in Computing Research* **3**, 1986.

[Ginsberg 85] M. L. Ginsberg, "Counterfactuals," *Artificial Intelligence* **30**:1, 1986.

[Harman 86] G. Harman, *Change in View*, MIT Press, Cambridge MA, 1986.

[Imielinski 84] T. Imielinski and W. Lipski, "Incomplete Information in Relational Databases," *Journal of the ACM* **31**:4, October 1984.

[Keller 82] A. M. Keller, "Updates to Relational Databases Through Views Involving Joins", *Improving Database Usability and Responsiveness*, P. Scheuermann, ed., Academic Press, New York, 1982.

[Keller 85] A. M. Keller, Algorithms for Translating View Updates to Database Updates for Views Involving Selections, Projections, and Joins", *Proc. of the 4th ACM Symposium on Principles of Database Systems*, Portland OR, March 1985.

[Keller 85] A. M. Keller and M. Winslett–Wilkins, "On the Use of an Extended Relational Model to Handle Changing Incomplete Information", *IEEE Trans. on Software Engineering* **SE-11**:7, July 1985.

[Knuth 73] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching*, Volume 3, Addison–Wesley, Reading MA, 1973.

[Levesque 84] H. Levesque, "Foundations of a Functional Approach to Knowledge Representation," *Artificial Intelligence* **23**, July 1984.

[Lewis 73] D. Lewis, *Counterfactuals*, Harvard University Press, Cambridge MA, 1973.

[Lifschitz 85] V. Lifschitz, "Closed–World Databases and Circumscription", *Artificial Intelligence* **27**:2, November 1985.

[McCarthy 80] J. McCarthy, "Circumscription: A Form of Non-Monotonic Reasoning", *Artificial Intelligence* **13**, 1980.

[Michalski 86] R. S. Michalski and P. H. Winston, "Variable Precision Logic", *Artificial Intelligence* **29**:2, August 1986.

[Nilsson 86] N. J. Nilsson, "Probabilistic Logic", *Artificial Intelligence* **28**:1, February 1986.

[Papadimitriou 86] C. H. Papadimitriou, *A Theory of Database Concurrency Control*, Computer Science Press, 1986.

[Reiter 80] R. Reiter, "Equality and Domain Closure in First Order Databases", *Journal of the ACM* **27**, 1980.

[Reiter 84] R. Reiter, "Towards a Logical Reconstruction of Relational Database Theory," in M. Brodie, J. Myopoulos, and J. Schmidt (eds.) *On Conceptual Modelling*, Springer-Verlag, 1984.

[Reiter 84b] R. Reiter, "Logical Foundations for Database Theory," *Conference On Theory in Data Bases*, Benodet, France, July 1984.

[Reiter 85] R. Reiter, "A Theory of Diagnosis from First Principles," University of Toronto Computer Science Dept. Tech. Report, 1985.

[Stonebraker 85] M. Stonebraker, ed., *The INGRES Papers*, Addison Wesley, Reading MA, 1985.

[Todd 77] S. Todd, "Automatic Constraint Maintenance and Updating Defined Relations", *Proc. IFIP 1977* (B. Gilchrist, ed.), North–Holland, 1977.

[Ullman 82] J. D. Ullman, *Principles of Database Systems*, 2nd ed., Computer Science Press, 1982.

[Vardi 85] M. Vardi, "Querying Logical Databases," *Proc. of the 4th ACM Symposium on Principles of Database Systems*, March 1985.

[Vassiliou 79] Y. Vassiliou, "Null Values in Data Base Management: A Denotational Semantics Approach", Proc. ACM SIGMOD, Boston, May 1979.

[Weber 86] A. Weber, "Updating Propositional Formulas," *Proc. of the Expert Database Systems Conference*, Charleston SC, April 1986.

[Wiederhold 83] Gio Wiederhold, *Database Design*, 2nd ed., McGraw–Hill, 1983.

[Winslett 86a] M. Winslett–Wilkins, "A Model-Theoretic Approach to Updating Logical Databases," Stanford Computer Science Dept. Tech. Report, Jan. 1986; a preliminary version appeared in "A Model-Theoretic Approach to Updating Logical Databases (Extended Abstract)," *Proc. of the 5th ACM Symposium on Principles of Database Systems*, Cambridge, March 1986.

[Winslett 86b] M. Winslett, "Updating Logical Databases With Null Values," *Proc. 1st International Conference on Database Theory,* Rome, September 1986.

[Winslett 86c] M. Winslett, "Is Belief Revision Harder Than You Thought?", Stanford Computer Science Dept. Tech. Report, to appear; a shortened version appeared under the same name in *Proc. of the 5th National Conference on Artificial Intelligence*, Philadelphia, August 1986.

[Zadeh 79] L. A. Zadeh, "Approximate Reasoning Based on Fuzzy Logic", *Proc. of the 6th International Joint Conference on Artificial Intelligence*, August 1979.

[Zaniolo 82] C. Zaniolo, "Database Relations with Null Values", *Proc. of the 1st ACM Symposium on Principles of Database Systems*, March 1982.

Reproduced by NTIS

National Technical Information Service
Springfield, VA 22161

This report was printed specifically for your order from nearly 3 million titles available in our collection.

For economy and efficiency, NTIS does not maintain stock of its vast collection of technical reports. Rather, most documents are printed for each order. Documents that are not in electronic format are reproduced from master archival copies and are the best possible reproductions available. If you have any questions concerning this document or any order you have placed with NTIS, please call our Customer Service Department at (703) 487-4660.

About NTIS

NTIS collects scientific, technical, engineering, and business related information — then organizes, maintains, and disseminates that information in a variety of formats — from microfiche to online services. The NTIS collection of nearly 3 million titles includes reports describing research conducted or sponsored by federal agencies and their contractors; statistical and business information; U.S. military publications; audiovisual products; computer software and electronic databases developed by federal agencies; training tools; and technical reports prepared by research organizations worldwide. Approximately 100,000 *new* titles are added and indexed into the NTIS collection annually.

For more information about NTIS products and services, call NTIS at (703) 487-4650 and request the free *NTIS Catalog of Products and Services*, PR-827LPG, or visit the NTIS Web site **http://www.ntis.gov**.

NTIS
Your indispensable resource for government-sponsored information—U.S. and worldwide