Capturing Database Dynamics by Deferred Updates

Carl-Alexander Wichert and **Burkhard Freitag**

Universität Passau
Fakultät für Mathematik und Informatik
D-94030 Passau, Germany

e-mail: {wichert,freitag}@fmi.uni-passau.de
Tel: +49 (0)851 509 3019 Fax: +49 (0)851 509 3002

Universität Passau
Technical Report MIP-9704

Abstract

In the field of logic databases the static semantics and the evaluation of queries is well understood. However, there is no consensus about the appropriate treatment of dynamic behaviour, i.e. the evolution of the database with time. Various methods to incorporate update constructs into logic databases have been proposed. In this paper we address the *explicit* construction of (complex) update programs from basic update operations like "insert tuple" and "delete tuple". The main contribution is the definition of a logical semantics that is based on sets of *deferred* update requests. We propose a logic language that allows to express not only the concurrent and sequential composition of update operations but also set-oriented updates or bulk updates.

1 Introduction

In the field of logic databases the static semantics and the evaluation of queries is well understood. However, there is no consensus about the appropriate treatment of dynamic behaviour, i.e. the evolution of the database with time. Various methods to incorporate update constructs into logic databases have been proposed, see [TU95] for a comprehensive classification. In this paper we address the *explicit* construction of (complex) update programs from basic update operations like "insert tuple" and "delete tuple". This is to be distinguished from the problem of view updates which in a sense *implicitly* define update programs.

Several rule-based languages for explicit update programs have been proposed, among them [AHV95, BGM95, BK94, BK96, LHL95, LML96, MW88, Zan93]. Basically, update requests may appear in the rule head or in the rule body. We adopt the updates-as-subgoals paradigm which is also used in Transaction Logic [BK94, BK96] and other approaches [BGM95, MW88]. In this setting, a *deductive rule* defines a (part of) a view [AHV95], whereas an *update rule* can be regarded as a (partial) definition of a transaction which is named by the head predicate and parameterized by the head variables. A named transaction can be referred to by update literals that appear as (sub-)goals in the bodies of update rules or update queries. Thus, complex update operations can be implemented in a modular and hierarchical way and the concept does integrate with nested transaction models [Mos85, WS92].

A major problem of this approach is the combination of set-orientedness, which one might prefer from the point of view of declarativity, with the inherent non-determinism of rule-based languages, that forces one to choose one of the possible state transitions that satisfy an update query. In particular, *set-oriented updates* or *bulk updates*, e.g. "FOR ALL P DO BEGIN raise salary of P by 10% END", which are often used in databases and which are expressible in SQL need special consideration. The existing update languages with updates as subgoals rely on explicit recursive rules to encode the universal quantifier of the example above. However, a true integration of set-oriented updates into the overall framework has not been addressed.

Transaction processing requires that the effects of a non-terminated transaction can be undone [AHV95] in case of a failure. In databases, this is achieved by deferring the updates caused by a transaction until 'Commit', or, in other words, by introducing hypothetical database states that can be accessed exclusively by the database operations issued by this transaction and materializing the changed database after the successful end of the transaction. Although in databases all this is normally done at the physical level, the idea of deferring updates and maintaining hypothetical states carries over to logic database programs containing updates.

In this paper we propose a logic which allows to express the construction of update operations by concurrent conjunction, sequential conjunction, and bulk quantification. Bulk quantification corresponds to simultaneous set-oriented updates. In our framework, bulk updates can be regarded as a special form of concurrent conjunction. We restrict ourselves to the basic update operations "insert tuple" and "delete tuple", although other sets of basic operations can be supported.

The semantics we present is based on the concept of *deferred updates*. Update queries are processed in two subsequent phases. In the *evaluation phase*, sets of basic update requests are computed which potentially cause the transition from the current database state to some new state. Note, that updates are not executed in this phase. After the valid update request sets have been determined, the *materialization phase* begins during which one of the computed update request sets is non-deterministically chosen and applied to the extensional database. By this execution of updates a new physical database state is materialized.

A basic update atom like "insert tuple" defines a basic update request set containing exactly this single basic update request. Each composition construct determines how the update request sets corresponding to its operands have to be merged. The process of collecting basic update requests can be interpreted as a form of abduction [EK89] with the instantiated basic update atoms, such as "insert tuple", as abducibles. For bulk updates the merging is essentially performed by an aggregation over successful instantiations. In general, the sequential conjunction of updates leads to different intermediate database states. The semantics handles these by a form of hypothetical reasoning which is done in the evaluation phase.

A prototype system based on the concept presented in this paper has been implemented on top of the deductive database system *LOLA* [FSS91, ZF96].

The rest of the paper is organized as follows: The syntax and semantics of the proposed language is defined in Sections 2 and 3. Section 4 describes the basic ideas underlying the current prototypical implementation. We discuss related work in Section 5. Section 6 concludes the paper with a short summary and an outlook.

2 The Update Language

In this paper we consider extended logic or deductive databases which consist of three components: an *extensional database (EDB)* comprising some base relations, e.g. a relational database, an *intensional database (IDB)* defined by a set of normal deductive rules, and an *update program* consisting of update rules to be defined below. Correspondingly, we distinguish the sets of *EDB predicates*, *IDB predicates*, and *update predicates*. The update predicates can be regarded as names of complex update operations (transactions).

2.1 Preliminaries

Definition 2.1 [Signature] A *signature* Σ is a tuple

$$(Const, Fun, Pred_{EDB}, Pred_{IDB}, Pred_{UP}),$$

where $Const$ is a (possibly infinite) countable set of constants, Fun is a finite set of function symbols, and $Pred_{EDB}$, $Pred_{IDB}$, $Pred_{UP}$ are finite sets of EDB, IDB, and update predicate symbols, respectively. □

Besides the signature Σ we assume that a countable infinite set of variables is available.

Definition 2.2 [**Terms, Atoms, DB Literals**] Let a signature Σ be given.

A *term* (over Σ) is a type- and arity-conform composition of function symbols (of Fun), constants (of $Const$) and variables. A term without variables is called *ground*. Groundness can be extended to the constructs defined below in a natural way.

Atoms (over Σ) are of the form $p(\vec{t})$, where $\vec{t}$ is a sequence $t_1, \ldots, t_n$ of terms (over Σ) and $p \in Pred_{EDB} \cup Pred_{IDB} \cup Pred_{UP}$ is an n-ary predicate. We can distinguish between *EDB atoms*, *IDB atoms*, and *update atoms* depending on which set of predicates p belongs to.

DB literals (over Σ) are EDB/IDB atoms $q(\vec{t})$ or negated EDB/IDB atoms $NOT\ q(\vec{t})$ (over Σ). □

Definition 2.3 [**Herbrand Universe**] The set of all ground terms over a signature Σ is called the *Herbrand universe* over Σ and denoted by $\mathcal{U}^{\Sigma}$. □

In the following, we implicitly assume that every ground term t is a member of $\mathcal{U}^{\Sigma}$.

Definition 2.4 [**Herbrand Base**] The set of all ground EDB and IDB atoms over a signature Σ is called the *Herbrand base* over Σ and denoted by $\mathcal{B}^{\Sigma}$. □

Remark 2.5 $\mathcal{U}^{\Sigma}$ and $\mathcal{B}^{\Sigma}$ are both countable, even in the case that $Const$ is not finite. □

Definition 2.6 [**Definable Update Base**] The set of all ground update atoms $p(\vec{t})$ over a signature Σ, i.e. atoms with $p \in Pred_{UP}$, is called the *definable update base* over Σ and is denoted by $\mathcal{B}^{\Sigma}_{DU}$. Its elements are called *definable update atoms*. □

In the following we will keep a global signature Σ fixed and consider only programs over this signature. In particular, Σ contains all constants needed in the database. The semantics we present below is well-defined, even if not all symbols of Σ actually occur in the programs. Consequently, we will omit the superscript Σ where possible.

2.2 Update Formulae

Besides the implication "$\Rightarrow$" and $\forall$-quantification, the update language contains the *basic update operators* INS ("insert tuple") DEL ("delete tuple"), and NOP ("no operation"), the *concurrent conjunction* ",", the *sequential conjunction* ":", and the *bulk quantifier* "#". The bulk quantifier is necessary to formulate set-oriented updates.

Definition 2.7 [**Update Literals**]

1. Every DB literal is an *update literal*.
2. NOP is an *update literal*.
3. The *basic update atoms* $INS\ r(\vec{t})$ and $DEL\ r(\vec{t})$, where $r(\vec{t})$ is an EDB atom, are *update literals*.
4. Every update atom $p(\vec{t})$ is an *update literal*.

□

Inductive Definition 2.8 [**Update Formulae**]

1. Every update literal is an *update formula*.
2. Let φ and ψ be update formulae. The implication $\varphi \Rightarrow \psi$ is an *update formula*.
3. Let φ be an update formula and $\vec{X}$ be a sequence $X_1, \ldots, X_n$ of variables. Then $\forall \vec{X}\ \varphi$ is an *update formula*.
4. Let φ and ψ be update formulae. The *concurrent conjunction* φ, ψ and the *sequential conjunction* $\varphi : \psi$ are *update formulae*.
5. Let φ be an update formula and $\vec{X}$ be a sequence $X_1, \ldots, X_n$ of variables. The *bulk quantification* $\#\vec{X}\ \varphi$ is an *update formula*.

In the following, we use square brackets "[...]" to indicate the operator bindings in a composite formula. □

Definition 2.9 [**Update Goals**] An *update goal* is an update formula not containing the implication "$\Rightarrow$" or the quantification "$\forall$", i.e. the set of update goals is defined by cases 1, 4, and 5 of Definition 2.8. □

Definition 2.10 [**Free Variables**] A variable occurring inside a formula is *free* unless being in the scope of a $\forall$-quantifier or a bulk quantifier "#". □

Definition 2.11 [**Replacement**] For an update formula φ we denote by $\varphi[\vec{X} / \vec{t}]$ the new formula φ' that results from replacing simultaneously the free occurrences of the variables $\vec{X}$ by the terms $\vec{t}$. □

2.3 Update Programs

In this paper we distinguish between IDB rules and update rules, although we consider the language of the update programs as an extension of the classical IDB rule formalism.

Let us first recall the definition of IDB rules.

Definition 2.12 As usual, *IDB rules* are universal closures of implications built from an IDB head atom $q(\vec{t})$ and a body $L_1, \ldots, L_n$ of DB literals. IDB rules are denoted by "$q(\vec{t})\ :-\ L_1, \ldots, L_n$.". Rules "$q(\vec{t})$." with an empty body are called *facts*. □

Definition 2.13 [Update Rules] An *update rule* r is a universal closure

$$\forall(U \Rightarrow p(\vec{t})),$$

also denoted by "$p(\vec{t})\ \Leftarrow\ U$.", where $p(\vec{t})$ is an update atom and U is an update goal. Rules without a head "$\Leftarrow U$." denote *(top-level) queries*. The empty body of an *update fact* "$p(\vec{t})$." corresponds to the update literal NOP. □

Definition 2.14 [Database Program] A *database program* P is a tuple (P_{IDB}, P_{UP}), where P_{IDB} denotes the set of IDB rules, and P_{UP} denotes the set of update rules. P_{IDB} is called the *IDB program* of P, and P_{UP} is called the *update program* of P. □

2.4 Syntactical Restrictions

In this paper we consider only update programs and update queries that satisfy the syntactical restrictions listed below.

Remark 2.15 [Syntactical Restriction 1] We do not allow update goals with free variables as top-level queries. □

The notion of the *dependency graph* is generalized to database programs in the obvious way.

Definition 2.16 [Stratification] The set of update rules is *stratified w.r.t. bulk quantification*, iff there is no dependency cycle through bulk quantification constructs. □

Remark 2.17 [Syntactical Restriction 2] We consider only update programs that are stratified w.r.t. bulk quantification. □

2.5 Example: Wine Cellar

The general set-up of the following example has been adopted from Gardarin and Valduriez [GV89]. Let the EDB relations $Wine(W\#, Name, Grape, Year)$ (wine types) and $Store(W\#, Amount)$ (wine storage) be represented by the EDB predicates *wine* and *store*, respectively. We assume that for every tuple in $Wine$, there exists a corresponding tuple in $Store$ and vice versa (referential integrity). The IDB predicate *out* characterizing wines that are not currently available in the cellar is defined by the IDB rule:

$$out(W)\ :-\ store(W, 0).$$

The following update rules define two (complex) updates on the $Store$ relation.

$$\begin{array}{lcl} order(W,D) & \Leftarrow & store(W,X),\ Y = X + D, \\ & & DEL\ store(W,X),\ INS\ store(W,Y). \\ consume(W,D) & \Leftarrow & store(W,X),\ X \geq D,\ Y = X - D, \\ & & DEL\ store(W,X),\ INS\ store(W,Y). \end{array}$$

Intuitively, $order(W,D)$ means that D bottles of wine W are bought from an external supplier. Here all subgoals are composed by concurrent conjunction, which means that they refer to one common database state and cause a single transition without considering any intermediate state. Note, that we tacitly assume a binding pattern such that X and D are bound before actually evaluating $Y{=}X{+}D$. Similarly, we assume that the operational semantics underlying the implementation ensures that the update goals $DEL\ store(W,X)$ and $INS\ store(W,Y)$ are evaluated *after* (ground) bindings for the variables W, X, and Y have been supplied. It should be mentioned, however, that the logical semantics defined in Section 3 does *not* depend on any binding strategy or subgoal (re-)ordering. $consume(W,D)$ is the counterpart of $order(W,D)$, i.e. represents an operation for removing a certain amount D of a wine W from the cellar.

To specify the simultaneous ordering of 24 bottles of all offered wines that are out, the bulk quantifier # is used in the following update rule.

$$order\_all\_out \ \Leftarrow\ \#\ W\ [\ out(W),\ order(W,24)\ ].$$

Essentially, this rule represents a bulk update "FOR ALL W DO BEGIN IF W is out THEN order 24 bottles of W END".

Sequential composition is exemplified by the following update rule

$$consume\_then\_order \ \Leftarrow\ consume(33,1)\ :\ order\_all\_out.$$

which specifies that one bottle of wine number 33 is consumed and subsequently all missing wines are ordered. Note, that the sequential composition requires that the second subgoal refers to the hypothetical state caused by $consume(33,1)$: if wine number 33 gets empty, it will have to be ordered, too.

It is worth mentioning that our running example is stratified w.r.t. bulk quantification. The interior goal $out(W), order(W,24)$ of the bulk quantifier in the defining rule of $order\_all\_out$ does not depend recursively on $order\_all\_out$.

3 Semantics

There are many different semantics for normal programs. Presently, the most commonly accepted semantics are subsumed by the *well-founded model semantics* [vGRS91].

For the rest of the paper we assume this semantics as given. However, as in Transaction Logic [BK94, BK96], our semantics of update formulae is not tied to the particular IDB semantics chosen.

3.1 Interpretation of Update Formulae

The semantics we present is based on the concept of *deferred* updates. Essentially this means that updates are not immediately executed when they are "encountered". Instead, each update goal determines (one or more) *potential* updates that can be *derived* without affecting the database state and later may or may not be materialized, i.e. applied to the database. Not earlier than at *materialization time* does a potential update cause a transition from the current database state to some new state.

Potential updates are represented as sets of basic update requests. From the point of view of the update semantics an update request set can be considered as an assertion about the next database state. A basic update like "insert tuple" defines a basic update request set containing exactly this single basic update request. Each composition construct determines how the update request sets corresponding to its operands have to be merged.

Definition 3.1 [EDB Instance] A set $DB \subseteq \mathcal{B}$ of ground EDB atoms is called an *EDB instance.* □

An EDB instance characterizes a database state and can be compared with the contents of the base tables in a database system.

Definition 3.2 [Update Request] A *ground basic update request* has the form $+r(\vec{t})$ (*insertion request*) or $-r(\vec{t})$ (*deletion request*), where $r(\vec{t}) \in \mathcal{B}$ is a ground EDB atom. □

Intuitively, a ground basic update request specifies the insertion or deletion of an EDB atom – which itself corresponds to a tuple in an EDB relation.

Definition 3.3 [Update Request Set] An *update request set* Δ is a set of ground basic update requests. □

Definition 3.4 [Consistency] An update request set Δ is *consistent*, iff there exists no atom $r(\vec{t}) \in \mathcal{B}$ such that $+r(\vec{t}) \in \Delta$ and $-r(\vec{t}) \in \Delta$.

In the following, $\mathcal{D}$ denotes the set of all consistent update request sets taken from the restriction of $\mathcal{B}$ to EDB atoms. □

Definition 3.5 [Compatibility] Two (consistent) update request sets are called *compatible* with each other, if their union is consistent. □

Definition 3.6 [Concurrent Composition] The *concurrent composition* of two consistent update request sets $\Delta_1, \Delta_2 \in \mathcal{D}$ which are compatible is defined as the set union

$\Delta_1 \cup \Delta_2$. The concurrent composition is not defined for incompatible update request sets. □

Definition 3.7 [Sequential Composition] The *sequential composition* $\Delta_1 \oplus \Delta_2$ of two consistent update request sets $\Delta_1, \Delta_2 \in \mathcal{D}$ is defined by:

$$\begin{aligned} \Delta_1 \oplus \Delta_2 \;:=\; & \{+r(\vec{t}\,) \mid (+r(\vec{t}\,) \in \Delta_1 \wedge -r(\vec{t}\,) \notin \Delta_2) \vee +r(\vec{t}\,) \in \Delta_2\} \;\; \cup \\ & \{-r(\vec{t}\,) \mid (-r(\vec{t}\,) \in \Delta_1 \wedge +r(\vec{t}\,) \notin \Delta_2) \vee -r(\vec{t}\,) \in \Delta_2\} \end{aligned}$$

□

Lemma 3.8 [Algebraic Properties]

1. The sequential composition $\oplus$ is associative.
2. The empty set $\emptyset$ is the neutral element of $\oplus$.
3. For arbitrary $\Delta_1, \Delta_2 \in \mathcal{D}$, $\Delta_1 \oplus \Delta_2 \in \mathcal{D}$ holds, i.e. $\Delta_1 \oplus \Delta_2$ is consistent.

Proof: First, we show the associativity of $\oplus$ (1). Let $\Delta_1, \Delta_2, \Delta_3 \in \mathcal{D}$ be arbitrarily chosen. The following equivalences hold for arbitrary insertion requests $+r(\vec{t}\,)$:

$$\begin{aligned}
& +r(\vec{t}\,) \in (\Delta_1 \oplus \Delta_2) \oplus \Delta_3 \\
\Longleftrightarrow\;\; & (+r(\vec{t}\,) \in \Delta_1 \oplus \Delta_2 \;\wedge\; -r(\vec{t}\,) \notin \Delta_3) \;\vee\; +r(\vec{t}\,) \in \Delta_3 \\
\Longleftrightarrow\;\; & [\;(\;(+r(\vec{t}\,) \in \Delta_1 \wedge -r(\vec{t}\,) \notin \Delta_2) \vee +r(\vec{t}\,) \in \Delta_2) \wedge \\
& \;-r(\vec{t}\,) \notin \Delta_3\;] \;\vee\; +r(\vec{t}\,) \in \Delta_3 \\
\Longleftrightarrow\;\; & (+r(\vec{t}\,) \in \Delta_1 \wedge -r(\vec{t}\,) \notin \Delta_2 \wedge -r(\vec{t}\,) \notin \Delta_3) \vee \\
& (+r(\vec{t}\,) \in \Delta_2 \wedge -r(\vec{t}\,) \notin \Delta_3) \;\vee\; +r(\vec{t}\,) \in \Delta_3 \\
\Longleftrightarrow\;\; & (+r(\vec{t}\,) \in \Delta_1 \wedge -r(\vec{t}\,) \notin \Delta_2 \wedge -r(\vec{t}\,) \notin \Delta_3) \vee \\
{\scriptstyle (subsmpt.)}\;\; & (+r(\vec{t}\,) \in \Delta_1 \wedge +r(\vec{t}\,) \in \Delta_3 \wedge -r(\vec{t}\,) \notin \Delta_3) \vee \\
& (+r(\vec{t}\,) \in \Delta_2 \wedge -r(\vec{t}\,) \notin \Delta_3) \;\vee\; +r(\vec{t}\,) \in \Delta_3 \\
\Longleftrightarrow\;\; & (+r(\vec{t}\,) \in \Delta_1 \wedge (-r(\vec{t}\,) \notin \Delta_2 \vee +r(\vec{t}\,) \in \Delta_3) \wedge -r(\vec{t}\,) \notin \Delta_3) \vee \\
& (+r(\vec{t}\,) \in \Delta_2 \wedge -r(\vec{t}\,) \notin \Delta_3) \;\vee\; +r(\vec{t}\,) \in \Delta_3 \\
\Longleftrightarrow\;\; & (+r(\vec{t}\,) \in \Delta_1 \wedge \neg\,[\;(-r(\vec{t}\,) \in \Delta_2 \wedge +r(\vec{t}\,) \notin \Delta_3) \vee -r(\vec{t}\,) \in \Delta_3\;]\;) \vee \\
& (+r(\vec{t}\,) \in \Delta_2 \wedge -r(\vec{t}\,) \notin \Delta_3) \;\vee\; +r(\vec{t}\,) \in \Delta_3 \\
\Longleftrightarrow\;\; & (+r(\vec{t}\,) \in \Delta_1 \;\wedge\; -r(\vec{t}\,) \notin \Delta_2 \oplus \Delta_3) \;\vee\; +r(\vec{t}\,) \in \Delta_2 \oplus \Delta_3 \\
\Longleftrightarrow\;\; & +r(\vec{t}\,) \in \Delta_1 \oplus (\Delta_2 \oplus \Delta_3)
\end{aligned}$$

Have a closer look at the equivalence marked "subsmpt.". '$\Rightarrow$' is trivial, as the three disjuncts on the left hand side also occur on the right hand side. However, the converse '$\Leftarrow$' also holds, as the disjunct

$$+r(\vec{t}\,) \in \Delta_1 \wedge +r(\vec{t}\,) \in \Delta_3 \wedge -r(\vec{t}\,) \notin \Delta_3$$

which does not occur on the left hand side implies the disjunct

$$+r(\vec{t}) \in \Delta_3.$$

As the definitions are symmetric, similar equivalences hold for deletion requests $-r(\vec{t})$.

To show that $\emptyset$ is the neutral element (2), let $\Delta \in \mathcal{D}$ be arbitrarily chosen.

$$\begin{aligned}
\Delta \oplus \emptyset &= \{+r(\vec{t}) \mid (+r(\vec{t}) \in \Delta \wedge -r(\vec{t}) \notin \emptyset) \vee +r(\vec{t}) \in \emptyset\} \cup \\
&\quad\ \{-r(\vec{t}) \mid (-r(\vec{t}) \in \Delta \wedge +r(\vec{t}) \notin \emptyset) \vee -r(\vec{t}) \in \emptyset\} \\
&= \{+r(\vec{t}) \mid +r(\vec{t}) \in \Delta\} \cup \{-r(\vec{t}) \mid -r(\vec{t}) \in \Delta\} \\
&= \Delta
\end{aligned}$$

$$\begin{aligned}
\emptyset \oplus \Delta &= \{+r(\vec{t}) \mid (+r(\vec{t}) \in \emptyset \wedge -r(\vec{t}) \notin \Delta) \vee +r(\vec{t}) \in \Delta\} \cup \\
&\quad\ \{-r(\vec{t}) \mid (-r(\vec{t}) \in \emptyset \wedge +r(\vec{t}) \notin \Delta) \vee -r(\vec{t}) \in \Delta\} \\
&= \{+r(\vec{t}) \mid +r(\vec{t}) \in \Delta\} \cup \{-r(\vec{t}) \mid -r(\vec{t}) \in \Delta\} \\
&= \Delta
\end{aligned}$$

Finally, we show the consistency property (3). Let $\Delta_1, \Delta_2 \in \mathcal{D}$ be consistent update request sets. Assume that $\Delta_1 \oplus \Delta_2$ is not consistent. Then an atom $r(\vec{t}) \in \mathcal{B}$ exists, such that $+r(\vec{t}), -r(\vec{t}) \in \Delta_1 \oplus \Delta_2$. We can reason as follows:

$$\begin{aligned}
& +r(\vec{t}) \in \Delta_1 \oplus \Delta_2 \ \wedge\ -r(\vec{t}) \in \Delta_1 \oplus \Delta_2 \\
\Longrightarrow\ & (\, (+r(\vec{t}) \in \Delta_1 \wedge -r(\vec{t}) \notin \Delta_2) \vee +r(\vec{t}) \in \Delta_2 \,) \wedge \\
& (\, (-r(\vec{t}) \in \Delta_1 \wedge +r(\vec{t}) \notin \Delta_2) \vee -r(\vec{t}) \in \Delta_2 \,) \\
\Longrightarrow\ & (+r(\vec{t}) \in \Delta_1 \wedge -r(\vec{t}) \notin \Delta_2 \wedge -r(\vec{t}) \in \Delta_1 \wedge +r(\vec{t}) \notin \Delta_2) \vee \\
& (+r(\vec{t}) \in \Delta_2 \wedge -r(\vec{t}) \in \Delta_1 \wedge +r(\vec{t}) \notin \Delta_2) \vee \\
& (+r(\vec{t}) \in \Delta_1 \wedge -r(\vec{t}) \notin \Delta_2 \wedge -r(\vec{t}) \in \Delta_2) \vee \\
& (+r(\vec{t}) \in \Delta_2 \wedge -r(\vec{t}) \in \Delta_2) \\
\Longrightarrow\ & (+r(\vec{t}) \in \Delta_1 \wedge -r(\vec{t}) \in \Delta_1) \ \vee\ (+r(\vec{t}) \in \Delta_2 \wedge -r(\vec{t}) \in \Delta_2)
\end{aligned}$$

The last statement contradicts the precondition saying that Δ_1 and Δ_2 are consistent. Thus, our assumption must be false, which means that $\Delta_1 \oplus \Delta_2$ is consistent. □

Example 3.9 Let the update request sets

$$\Delta_1 = \{-store(33, 1), +store(33, 0)\}$$

and

$$\Delta_2 = \{-store(33, 0), +store(33, 24)\}$$

be given. The concurrent composition of Δ_1 and Δ_2 is not defined, because the update request sets are not compatible due to

$$+store(33,0) \in \Delta_1 \text{ and } -store(33,0) \in \Delta_2.$$

However, the sequential composition is possible and leads to the update request set

$$\Delta_1 \oplus \Delta_2 = \{-store(33,1), -store(33,0), +store(33,24)\}.$$

The update request $+store(33,0)$ is not contained in the result. This reflects the fact that in this example the deletion should follow and thus invalidate the insertion. □

To capture database dynamics we must consider different database states. Let DB_0 be a (physical) instance of the EDB. We characterize every database state DB following DB_0 by an update request set Δ that would lead to DB if applied to DB_0. Because within the logical framework update request sets are not *physically* applied to DB_0, we call every state except the physical EDB instance DB_0 *hypothetical* and represent it in terms of the corresponding Δ. In particular, the current database state is represented by the update request set Δ_C. Note, that every database state comprises the contents of both EDB and IDB, but changes (hypothetical or physical) are only applied to the EDB.

To be able to refer to a hypothetical state without materializing it, we need to define how the facts that are true in this state are derived. Recall that a hypothetical state is represented by an update request set Δ w.r.t. the EDB instance DB_0.

Definition 3.10 [Execution] The *(hypothetical) execution* $DB \oplus_E \Delta$ of a consistent update request set $\Delta \in \mathcal{D}$ w.r.t. the EDB instance DB is defined by

$$DB \oplus_E \Delta \;:=\; \{r(\vec{t}\,) \mid (r(\vec{t}\,) \in DB \land -r(\vec{t}\,) \notin \Delta) \lor +r(\vec{t}\,) \in \Delta\}$$

□

Lemma 3.11 [Algebraic Properties] Let $DB \subseteq \mathcal{B}$ be an EDB instance.

1. Let $\Delta \in \mathcal{D}$ be a consistent update request set. Then for arbitrary EDB atoms $r(\vec{t}\,) \in \mathcal{B}$ the following holds:

$$\begin{aligned} +r(\vec{t}\,) \in \Delta &\implies r(\vec{t}\,) \in (DB \oplus_E \Delta) \\ -r(\vec{t}\,) \in \Delta &\implies r(\vec{t}\,) \notin (DB \oplus_E \Delta) \end{aligned}$$

2. For arbitrary update request sets $\Delta_1, \Delta_2 \in \mathcal{D}$ the following holds:

$$(DB \oplus_E \Delta_1) \oplus_E \Delta_2 = DB \oplus_E (\Delta_1 \oplus \Delta_2)$$

Proof: We first prove (1). Let $DB \subseteq \mathcal{B}$ and $\Delta \in \mathcal{D}$ be given. From Definition 3.10 follows:

$$\{r(\vec{t}\,) \mid +r(\vec{t}\,) \in \Delta\} \subseteq DB \oplus_E \Delta$$

In the case of deletion requests $-r(\vec{t})$, the consistency of Δ is essential. Assume that $-r(\vec{t}) \in \Delta$ and $r(\vec{t}) \in DB \oplus_E \Delta$. The following implications hold:

$$\begin{aligned}
& -r(\vec{t}) \in \Delta \ \wedge \ r(\vec{t}) \in DB \oplus_E \Delta \\
\Longrightarrow \ & -r(\vec{t}) \in \Delta \ \wedge \ [\ (r(\vec{t}) \in DB \wedge -r(\vec{t}) \notin \Delta) \ \vee \ +r(\vec{t}) \in \Delta\ ] \\
\Longrightarrow \ & (-r(\vec{t}) \in \Delta \wedge r(\vec{t}) \in DB \wedge -r(\vec{t}) \notin \Delta) \vee \\
& (-r(\vec{t}) \in \Delta \wedge +r(\vec{t}) \in \Delta) \\
\Longrightarrow \ & -r(\vec{t}) \in \Delta \wedge +r(\vec{t}) \in \Delta
\end{aligned}$$

The last statement contradicts the precondition saying that Δ is consistent. Thus, our assumption must be false and the assertion of (1) follows.

Assertion (2) can be proved similarly to the proof of the associativity of $\oplus$ in Lemma 3.8. □

Example 3.12 Let the EDB instance

$$DB = \{store(18,0),\ store(32,21)\}$$

and the update request set

$$\Delta = \{-store(18,0), +store(18,24)\}$$

be given. The execution of Δ yields the following EDB instance DB':

$$DB' = DB \oplus_E \Delta = \{store(18,24),\ store(32,21)\}$$

□

Definition 3.13 [Well-Founded Model] Let $\Phi = DB \cup P_{IDB}$ be the union of an EDB instance DB and a set of IDB rules P_{IDB}.

$WFM(\Phi)$ denotes the *well-founded model* (see [vGRS91]) of Φ. To be able to deal with three-valued models, we assume that $WFM(\Phi)$ is represented by a set of true and false ground atoms of $\mathcal{B}$. We use the prefix $\neg$ when referring to a false atom $p(\vec{t})$. □

Definition 3.14 [Semantics of DB Literals] Let DB_0 be the physical EDB instance and P_{IDB} the IDB program. We define a *DB interpretation* I_{DB}, which assigns to every $\Delta \in \mathcal{D}$ a well-founded model by

$$I_{DB}(\Delta) := WFM(P_{IDB} \cup (DB_0 \oplus_E \Delta)).$$

Recall, that terms are interpreted by the usual Herbrand pre-interpretation. □

We will now define the interpretation of update formulae. The main ideas have been adopted from Manchanda and Warren [MW88]. However, their interpretation is based on pairs of states, while our interpretation relies on differences between consecutive states, i.e. update request sets. Thus, imposing additional consistency constraints, we can also define a sensible semantics of concurrent conjunction and bulk updates. Note, that we define the semantics of quantifiers over a replacement of variables by ground terms. This is correct, because we use the Herbrand pre-interpretation, where every domain element can be represented by a ground term of $\mathcal{U}$.

An interpretation I is a mapping from the set of ground update formulae to the power-set of $\mathcal{D} \times \mathcal{D}$, i.e. $I(\varphi) \subseteq \mathcal{D} \times \mathcal{D}$ for every ground formula φ. As defined below, I will be an extension of an interpretation I_{UP} of update atoms. The first component of each pair $(\Delta_C, \Delta) \in I(\varphi)$ refers to a (hypothetical) current state DB_{Curr} w.r.t. DB_0 in which φ is to be evaluated, the second component refers to a set of update requests that would lead to the next database state DB_{Next} if applied to the hypothetical current state DB_{Curr}. Every pair (Δ_C, Δ) corresponds to an allowed state transition.

$$(\quad DB_0 \quad \xrightarrow{\Delta_C} \quad) \quad DB_{Curr} \quad \xrightarrow{\Delta} \quad DB_{Next}$$

If $I(\varphi)$ contains multiple pairs with the same first component Δ_C, φ has a non-deterministic update interpretation.

Note, that non-deterministic *choice* is *not* involved at the level of the logical semantics. However, an implementation has to perform a choice operation when materializing one of the new possible database states.

Inductive Definition 3.15 [Interpretation of a Ground Update Formula]
Let I_{DB} be the DB interpretation of Definition 3.14. Let the interpretation I_{UP} of the definable update atoms, i.e. a mapping

$$I_{UP} : \mathcal{B}_{DU} \rightarrow 2^{\mathcal{D} \times \mathcal{D}},$$

be given.

We define the interpretation I as an extension of I_{UP} to arbitrary ground update formulae inductively as follows.

(1) DB literal

Let $q(\vec{t})$ be an EDB or an IDB atom.

For all $\Delta_C, \Delta \in \mathcal{D}$ define:

$$(\Delta_C, \Delta) \in I(q(\vec{t})) \quad :\Longleftrightarrow \quad q(\vec{t}) \in I_{DB}(\Delta_C) \text{ and } \Delta = \emptyset$$
$$(\Delta_C, \Delta) \in I(NOT\ q(\vec{t})) \quad :\Longleftrightarrow \quad \neg q(\vec{t}) \in I_{DB}(\Delta_C) \text{ and } \Delta = \emptyset$$

(2) NOP

For all $\Delta_C, \Delta \in \mathcal{D}$ define:

$$(\Delta_C, \Delta) \in I(NOP) \quad :\Longleftrightarrow \quad \Delta = \emptyset$$

(3) Basic update atom

For all $\Delta_C, \Delta \in \mathcal{D}$ define:

$$(\Delta_C, \Delta) \in I(INS\ r(\vec{t})) \quad :\Longleftrightarrow \quad \Delta = \{+r(\vec{t})\}$$
$$(\Delta_C, \Delta) \in I(DEL\ r(\vec{t})) \quad :\Longleftrightarrow \quad \Delta = \{-r(\vec{t})\}$$

(4) Update atom

Let $p(\vec{t}) \in \mathcal{B}_{DU}$ be an update atom.

For all $\Delta_C, \Delta \in \mathcal{D}$ define:

$$(\Delta_C, \Delta) \in I(p(\vec{t})) \quad :\Longleftrightarrow \quad (\Delta_C, \Delta) \in I_{UP}(p(\vec{t}))$$

(5) Implication

For all $\Delta_C, \Delta \in \mathcal{D}$ define:

$$(\Delta_C, \Delta) \in I(\varphi \Rightarrow \psi) \quad :\Longleftrightarrow \quad (\Delta_C, \Delta) \in I(\varphi) \Longrightarrow (\Delta_C, \Delta) \in I(\psi)$$

(6) Universal quantification

For all $\Delta_C, \Delta \in \mathcal{D}$ define:

$(\Delta_C, \Delta) \in I(\forall \vec{X}\, \varphi) \quad :\Longleftrightarrow$ for arbitrary ground term tuples $\vec{t}$ the following holds: $(\Delta_C, \Delta) \in I(\varphi[\vec{X} / \vec{t}])$

(7) Concurrent conjunction

For all $\Delta_C, \Delta \in \mathcal{D}$ define:

$(\Delta_C, \Delta) \in I(\varphi, \psi) \quad :\Longleftrightarrow$ there exist $\Delta_1, \Delta_2 \in \mathcal{D}$ such that: $(\Delta_C, \Delta_1) \in I(\varphi)$ and $(\Delta_C, \Delta_2) \in I(\psi)$ and $\Delta = \Delta_1 \cup \Delta_2$

(8) Sequential conjunction

For all $\Delta_C, \Delta \in \mathcal{D}$ define:

$(\Delta_C, \Delta) \in I(\varphi : \psi) \quad :\Longleftrightarrow$ there exist $\Delta_1, \Delta_2 \in \mathcal{D}$ such that: $(\Delta_C, \Delta_1) \in I(\varphi)$ and $(\Delta_C \oplus \Delta_1, \Delta_2) \in I(\psi)$ and $\Delta = \Delta_1 \oplus \Delta_2$

(9) Bulk quantification

Let φ be an update formula containing exactly the free variables $X_1, \dots, X_n$, and let $I(\varphi[\vec{X} / \vec{t}])$ be already defined for every ground tuple $\vec{t}$.
For $\Delta_C \in \mathcal{D}$ let $T_{\varphi,\Delta_C} := \{ \vec{t} \mid \exists \Delta' \in \mathcal{D} : (\Delta_C, \Delta') \in I(\varphi[\vec{X} / \vec{t}])\}$.

For all $\Delta_C, \Delta \in \mathcal{D}$ define:

$$(\Delta_C, \Delta) \in I(\# \vec{X} \varphi) \quad :\Longleftrightarrow \quad \begin{array}{l} T_{\varphi,\Delta_C} \neq \emptyset \\ \text{and there exists a function} \\ f : T_{\varphi,\Delta_C} \to \mathcal{D} \text{ such that:} \\ \forall \vec{t} \in T_{\varphi,\Delta_C} : (\Delta_C, f(\vec{t})) \in I(\varphi[\vec{X} / \vec{t}]) \\ \text{and } \Delta = \bigcup_{\vec{t} \in T_{\varphi,\Delta_C}} f(\vec{t}) \end{array}$$

Note, that only consistent update request sets Δ are considered. □

Remark 3.16 The interpretation of ground update formulae is well-defined. Cases (1) to (4) define the base of the induction, the other cases define the interpretation of a composite formula in terms of the interpretations of its proper subformulae. In cases (6) and (9) the subformulae may be instantiated, but the replacement occurs at the term level and does not produce more complex update formulae.

Note, that the interpretation $I(\varphi)$ of an update formula φ may be empty or there may exist some $\Delta_C \in \mathcal{D}$ such that no $\Delta \in \mathcal{D}$ exists with $(\Delta_C, \Delta) \in I(\varphi)$. This can often be the effect of incompatible update request sets in cases (7) and (9). □

Remark 3.17 The right hand side of case (5) of Definition 3.15 must not be mistaken as the set inclusion

$$I(\varphi) \subseteq I(\psi)$$

This is only true if I is a model of the formula $\varphi \Rightarrow \psi$ (see Section 3.2). In Definition 3.15 the implication

$$(\Delta_C, \Delta) \in I(\varphi) \Longrightarrow (\Delta_C, \Delta) \in I(\psi)$$

is a constraint that must be satisfied by any Δ that is to be considered a valid update request set for $\varphi \Rightarrow \psi$ w.r.t. I. □

Lemma 3.18 [Algebraic Properties] The concurrent conjunction "," and the sequential conjunction ":" are both associative and have NOP as the neutral element. The concurrent conjunction "," is also commutative.

Proof: The assertions follow from Definition 3.15 and Lemma 3.8. □

In the next example we illustrate the interpretation of update literals and some composite update formulae.

Example 3.19 We consider a small database that represents the state of a switch s, i.e. whether s is "on" or "off". Let on be an EDB predicate and let $on(s)$ represent the fact that the switch s is on. Assume we have the signature

$$\Sigma = (\{s\}, \emptyset, \{on\}, \emptyset, \emptyset),$$

such that the set $\mathcal{D}$ of all consistent update requests sets is given by

$$\mathcal{D} = \{\ \emptyset,\ \{+on(s)\},\ \{-on(s)\}\ \}$$

and the following EDB instances are possible:

$$DB_{off} := \emptyset$$
$$DB_{on} := \{on(s)\}$$

Due to cases (2) and (3) of Definition 3.15 any interpretation I satisfies the following equalities regardless of the initial (physical) EDB instance.

$$\begin{array}{lcll} I(NOP) & = & \{ & (\emptyset,\emptyset),\ (\{+on(s)\},\emptyset),\ (\{-on(s)\},\emptyset)\ \ \} \\ I(INS\ on(s)) & = & \{ & (\emptyset,\{+on(s)\}), \\ & & & (\{+on(s)\},\{+on(s)\}),\ (\{-on(s)\},\{+on(s)\})\ \ \} \\ I(DEL\ on(s)) & = & \{ & (\emptyset,\{-on(s)\}), \\ & & & (\{+on(s)\},\{-on(s)\}),\ (\{-on(s)\},\{-on(s)\})\ \ \} \end{array}$$

The basic update atom $INS\ on(s)$ defines the update request set $\Delta_1 = \{+on(s)\}$, whereas $DEL\ on(s)$ defines the update request set $\Delta_2 = \{-on(s)\}$ as expected.

1. Let us first consider the concurrent conjunction

   $$INS\ on(s), DEL\ on(s)$$

   (cf. 3.15,(7)) of $INS\ on(s)$ and $DEL\ on(s)$. The update request sets Δ_1 and Δ_2 are not compatible, i.e. their union

   $$\Delta_1 \cup \Delta_2 = \{+on(s), -on(s)\} \notin \mathcal{D}$$

   is inconsistent. Therefore for any interpretation I,

   $$I(INS\ on(s), DEL\ on(s)) = \emptyset$$

   holds. This corresponds to the fact that it is not possible to set a switch simultaneously to the positions "on" and "off".

2. Consider now the sequential composition

   $$INS\ on(s) : DEL\ on(s)$$

(cf. 3.15,(8)) of $INS\ on(s)$ and $DEL\ on(s)$. Intuitively, the formula above specifies that s is to be switched on and then off. Consequently, $on(s)$ should not be contained in the database after an application of $INS\ on(s) : DEL\ on(s)$. Due to the sequential composition

$$\Delta_1 \oplus \Delta_2 = \{-on(s)\}$$

of Δ_1 and Δ_2 we get

$$\begin{aligned} & I(INS\ on(s) : DEL\ on(s)) \\ = & \ \{\ (\emptyset, \{-on(s)\}),\ (\{+on(s)\}, \{-on(s)\}),\ (\{-on(s)\}, \{-on(s)\})\ \} \\ = & \ I(DEL\ on(s)) \end{aligned}$$

for any interpretation I. One would expect this result because the only effect on the database that must be guaranteed by an interpretation is that the switch s is off after executing the transaction $INS\ on(s) : DEL\ on(s)$.

□

Remark 3.20 [No Compensation of Update Requests] Complementary basic update requests are *not* compensated by the sequential conjunction. Instead, the most recent update request remains in the resulting update request set. See Example 3.21 for an illustration of this fact. □

Example 3.21 Recall Example 3.19 where we saw that for any interpretation I,

$$\begin{aligned} & I(INS\ on(s) : DEL\ on(s)) \\ = & \ \{\ (\emptyset, \{-on(s)\}),\ (\{+on(s)\}, \{-on(s)\}),\ (\{-on(s)\}, \{-on(s)\})\ \} \\ = & \ I(DEL\ on(s)) \end{aligned}$$

holds. Assume that the complementary update requests $+on(s)$ and $-on(s)$ were *compensated* by the sequential composition. Informally speaking, this would mean that the composite update would have no effect on the database. As a consequence at the formal level, the update request set Δ corresponding to the update goal $INS\ on(s) : DEL\ on(s)$ would be empty, i.e.

$$\Delta = \emptyset.$$

Looking up Definition 3.15 we see that this is exactly the update request set resulting from the "no operation" update literal NOP. However, this does not agree with the intended effect of the specified action. Formally, this can also be seen by computing the hypothetical execution of the empty update request set:

$$DB_{off} \oplus_E \emptyset = \emptyset = DB_{off}$$

$$DB_{on} \oplus_E \emptyset = \{on(s)\} = DB_{on} \neq DB_{off}$$

Thus in general a compensation of complementary update requests is incorrect. □

Remark 3.22 [**No Serialization**] Unlike transaction semantics as used in database systems, the semantics of the concurrent conjunction can not be represented by the semantics of some serialization of the composed subtransactions. See Example 3.23 for an illustration of this fact. □

Example 3.23 Recall Example 3.19 and assume that the initial EDB instance is DB_{on}, i.e. $on(s)$ is contained in the database. Consider the update goal

$$\varphi :\equiv on(s), DEL\ on(s)$$

which has the intuitive meaning "if s is on, then switch it off, else cause a failure". The following equalities hold due to cases (1), (3), (7), and (8) of Definition 3.15:

$$\begin{array}{lcl} I(on(s)) & = & \{\ (\emptyset, \emptyset),\ (\{+on(s)\}, \emptyset)\ \}) \\ I(\varphi) & = & \{\ (\emptyset, \{-on(s)\}),\ (\{+on(s)\}, \{-on(s)\})\ \} \\ I(\varphi, \varphi) & = & \{\ (\emptyset, \{-on(s)\}),\ (\{+on(s)\}, \{-on(s)\})\ \} \\ I(\varphi : \varphi) & = & \emptyset \end{array}$$

The last two equations show that the concurrent conjunction of (the transaction represented by) the formula φ leads to the non-empty update request set $\Delta = \{-on(s)\}$ and thus defines some valid state transitions, however, this is not true for the sequential conjunction. The first occurrence of φ in $\varphi : \varphi$ defines the update request set $\Delta_1 = \{-on(s)\}$. Because $on(s)$ does not hold in any intermediate state corresponding to the execution of Δ_1, the conjunct $on(s)$ of the second occurrence of φ fails and therefore the entire subformula φ fails. According to case (8) of Definition 3.15 the formula $\varphi : \varphi$, i.e. the serial transaction, fails. □

Definition 3.24 [**Concurrent Conjunction over a Set**] Let Φ be a finite set of ground update formulae. The set $CConj(\Phi)$ of the *concurrent conjunctions* over Φ is inductively defined by

$$CConj(\Phi) := \begin{cases} \{NOP\}, & \text{if } \Phi = \emptyset \\ \bigcup\limits_{\varphi_0 \in \Phi} \{\ [\varphi_0, \varphi'] \mid \varphi' \in CConj(\Phi \setminus \{\varphi_0\})\ \}, & \text{if } \Phi \neq \emptyset \end{cases}$$

□

Remark 3.25 Let $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ be a non-empty set of ground update formulae. Then any formula $\varphi_{\pi(1)}, [\ldots, [\varphi_{\pi(n)}, NOP] \ldots]$, where π is a permutation of $\{1, \ldots, n\}$, is a concurrent conjunction over Φ. □

Lemma 3.26 Let Φ be a non-empty, finite set of ground update formulae. Then for an arbitrary concurrent conjunction ψ over Φ and $\Delta_C, \Delta \in \mathcal{D}$ the following holds:

$$\begin{array}{rcl} (\Delta_C, \Delta) \in I(\psi) & \iff & \text{there exists a function } f : \Phi \to \mathcal{D}, \text{ such that :} \\ & & \forall \varphi \in \Phi :\ (\Delta_C, f(\varphi)) \in I(\varphi) \\ & & \text{and } \Delta = \bigcup_{\varphi \in \Phi} f(\varphi) \end{array}$$

Proof: We show the assertion by induction on the cardinality of Φ.

Base case: $\Phi = \{\varphi_0\}$

Let ψ and $\Delta_C, \Delta \in \mathcal{D}$ be arbitrarily chosen. The following equivalence holds:

$$\begin{aligned} & (\Delta_C, \Delta) \in I(\psi) \\ \underset{(3.24)}{\iff} & (\Delta_C, \Delta) \in I(\varphi_0, NOP) \\ \underset{(3.18)}{\iff} & (\Delta_C, \Delta) \in I(\varphi_0) \end{aligned}$$

'$\Rightarrow$':

Let $(\Delta_C, \Delta) \in I(\psi)$. Define $f : \Phi \to \mathcal{D}$ by

$$f(\varphi) := \Delta$$

for all $\varphi \in \Phi$. The equivalence above shows that f has the required properties.

'$\Leftarrow$':

Let f be given. From $|\Phi| = 1$ follows

$$\Delta = f(\varphi_0)$$

and

$$(\Delta_C, \Delta) \in I(\varphi_0).$$

The assertion follows from the equivalence shown above.

Induction step:

Let $\Delta_C, \Delta \in \mathcal{D}$ be arbitrarily chosen. Let ψ be an arbitrary concurrent conjunction over Φ. According to Definition 3.24 there is a $\varphi_0 \in \Phi$ and a $\psi' \in CConj(\Phi \setminus \{\varphi_0\})$ such that $\psi = [\varphi_0, \psi']$.

Let $\Phi' := \Phi \setminus \{\varphi_0\}$.

The following equivalences follow from the definitions.

$$\begin{aligned} & (\Delta_C, \Delta) \in I(\psi) \\ \underset{(3.24)}{\iff} & (\Delta_C, \Delta) \in I(\varphi_0\,, \psi') \\ \underset{(3.15,(7))}{\iff} & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\ & (\Delta_C, \Delta_1) \in I(\varphi_0) \\ & \text{and } (\Delta_C, \Delta_2) \in I(\psi') \\ & \text{and } \Delta = \Delta_1 \cup \Delta_2 \end{aligned}$$

By the induction hypothesis, the following holds for arbitrary $\Delta'_C, \Delta' \in \mathcal{D}$:

$$
\begin{aligned}
(\Delta'_C, \Delta') \in I(\psi') \iff\ & \text{there exists a function } f' : \Phi' \to \mathcal{D}, \\
& \text{such that :} \\
& \forall \varphi \in \Phi' : \ (\Delta'_C, f'(\varphi)) \in I(\varphi) \\
& \text{and } \Delta' = \textstyle\bigcup_{\varphi \in \Phi'} f'(\varphi)
\end{aligned}
$$

'⇒':

Assume that $(\Delta_C, \Delta) \in I(\psi)$. Following from the equivalences above there exist $\Delta_1, \Delta_2 \in \mathcal{D}$ such that

$$
\begin{aligned}
& (\Delta_C, \Delta_1) \in I(\varphi_0) \\
\text{and} \quad & (\Delta_C, \Delta_2) \in I(\psi') \\
\text{and} \quad & \Delta = \Delta_1 \cup \Delta_2.
\end{aligned}
$$

We apply the induction hypothesis with $\Delta'_C := \Delta_C$ and $\Delta' := \Delta_2$. This yields a function $f' : \Phi' \to \mathcal{D}$. Define $f : \Phi \to \mathcal{D}$ by

$$
f(\varphi) := \begin{cases} \Delta_1, & \text{if } \varphi = \varphi_0 \\ f'(\varphi), & \text{if } \varphi \in \Phi' \end{cases}
$$

By distinction of cases and the properties of f' we can conclude

$$(\Delta_C, f(\varphi)) \in I(\varphi)$$

for all $\varphi \in \Phi$. We also have

$$\Delta = \Delta_1 \cup \Delta_2 = \Delta_1 \cup \bigcup_{\varphi \in \Phi'} f'(\varphi) = \bigcup_{\varphi \in \Phi} f(\varphi).$$

'⇐':

Let f be a function with the properties listed above. We set

$$\Delta_1 := f(\varphi_0)$$

and

$$\Delta_2 := \bigcup_{\varphi \in \Phi'} f(\varphi).$$

It is obvious that

$$(\Delta_C, \Delta_1) \in I(\varphi_0)$$

and

$$\Delta = \Delta_1 \cup \Delta_2.$$

f restricted to Φ' satisfies the preconditions of the induction hypothesis with $\Delta'_C := \Delta_C$ and $\Delta' := \Delta_2$ which can therefore be applied and lets us conclude that

$$(\Delta_C, \Delta_2) \in I(\psi').$$

The assertion follows from the equivalences shown above.

□

The following proposition shows a relation between concurrent conjunction and bulk quantification:

Proposition 3.27 [Conjunction and Bulk Updates] Let φ be an update formula containing exactly the free variables $X_1, \ldots, X_n$, and let $\Delta_C \in \mathcal{D}$ be an update request set. Define

$$T_{\varphi,\Delta_C} := \{\, \vec{t} \mid \exists \Delta' \in \mathcal{D} : (\Delta_C, \Delta') \in I(\varphi[\,\vec{X}\,/\,\vec{t}\,])\},$$
$$\Phi_{\varphi,\Delta_C} := \{\varphi[\,\vec{X}\,/\,\vec{t}\,] \mid \vec{t} \in T_{\varphi,\Delta_C}\}.$$

If T_{φ,Δ_C} is finite and non-empty, the bulk update $\#\,\vec{X}\,\varphi$ is semantically equivalent to any concurrent conjunction of all instances of φ taken over the set of ground terms T_{φ,Δ_C}. More formally, for any $\psi \in CConj(\Phi_{\varphi,\Delta_C})$,

$$(\Delta_C, \Delta) \in I(\#\,\vec{X}\,\varphi) \Longleftrightarrow (\Delta_C, \Delta) \in I(\psi)$$

holds for arbitrary $\Delta \in \mathcal{D}$.

Proof: Let T_{φ,Δ_C} be finite and non-empty. There exists a bijection between T_{φ,Δ_C} and Φ_{φ,Δ_C}, i.e. we can uniquely map $\vec{t}$ to $\varphi[\,\vec{X}\,/\,\vec{t}\,]$ and vice versa. Let $\psi \in CConj(\Phi_{\varphi,\Delta_C})$ be arbitrarily chosen.

For any update request set $\Delta \in \mathcal{D}$ holds:

$$\begin{array}{rl}
 & (\Delta_C, \Delta) \in I(\#\,\vec{X}\,\varphi) \\
\Longleftrightarrow & \text{there exists a function } f : T_{\varphi,\Delta_C} \to \mathcal{D}, \text{ such that :} \\
{\scriptstyle (3.15,(9))} & \forall \vec{t} \in T_{\varphi,\Delta_C} :\ (\Delta_C, f(\vec{t}\,)) \in I(\varphi[\,\vec{X}\,/\,\vec{t}\,]) \\
 & \text{and } \Delta = \bigcup_{\vec{t} \in T_{\varphi,\Delta_C}} f(\vec{t}\,) \\
\Longleftrightarrow & \text{there exists a function } f' : \Phi_{\varphi,\Delta_C} \to \mathcal{D}, \text{ such that :} \\
 & \forall \varphi' \in \Phi_{\varphi,\Delta_C} :\ (\Delta_C, f'(\varphi')) \in I(\varphi') \\
 & \text{and } \Delta = \bigcup_{\varphi' \in \Phi_{\varphi,\Delta_C}} f'(\varphi') \\
\underset{(3.26)}{\Longleftrightarrow} & (\Delta_C, \Delta) \in I(\psi)
\end{array}$$

□

Remark 3.28 Using transfinite induction to define the concurrent conjunctions of a countable set of ground update formulae and to extend the definition of an interpretation accordingly, the precondition of T_{φ,Δ_C} being finite can be dropped in Proposition 3.27. □

The next proposition shows that concurrent and sequential conjunction coincide for formulae which do not produce updates. Note, that in this case, both forms of conjunction also coincide with the conjunction used in conventional IDB rules.

Proposition 3.29 [Conjunction and Retrieval] Let φ and ψ be ground update formulae which do not produce any updates, i.e. for all $\Delta_C, \Delta \in \mathcal{D}$,

$$(\Delta_C, \Delta) \in I(\varphi) \text{ or } (\Delta_C, \Delta) \in I(\psi) \Longrightarrow \Delta = \emptyset$$

holds. Then

$$I(\varphi, \psi) = I(\varphi : \psi).$$

Proof: The assertion follows directly from Definition 3.15 and Lemma 3.8. □

3.2 Semantics of Update Programs

Let an extended deductive database with the database program $P = (P_{IDB}, P_{UP})$ be given, and let DB_0 be the physical EDB instance. We characterize the (minimal) models of the update program P_{UP}. As stated in Definition 3.14, the interpretation of the IDB literals is always given by the well-founded semantics w.r.t. P_{IDB} and DB_0. So we just have to find an interpretation I_{UP} of the update atoms that respects the intended semantics of P_{UP}. Recall, that in this setting, an interpretation I of the update formulae is completely determined by I_{UP}. We will therefore identify I_{UP} and its corresponding I and also write $I_{UP}(\varphi)$ for arbitrary ground update formulae φ.

Definition 3.30 [Models of an Update Program] An interpretation I_{UP} of update formulae is a *model* of a set of update rules P_{UP}, iff for each rule $r \in P_{UP}$

$$I_{UP}(r) = \mathcal{D} \times \mathcal{D}$$

holds. Recall, that rules denote universally closed implications. □

The definition of models may be surprising at first sight. Traditionally, an interpretation is called a model of a set of rules if it makes every single rule "true". Because rules are universal closures, it is sufficient to consider every ground instance of every rule. But, for a ground instance $\varphi \Rightarrow \psi$ of a rule being true w.r.t. an interpretation I means that if the body φ is true, then also the head ψ is true. Any sensible generalization of this notion of truth to (ground instances of) update rules would require that whenever the rule body φ defines a possible update Δ, this should also be considered a possible update for the rule head ψ. In other words, for an arbitrary current database state, represented by $\Delta_C \in \mathcal{D}$, and a possible update $\Delta \in \mathcal{D}$, the following should hold:

$$(\Delta_C, \Delta) \in I(\varphi) \Longrightarrow (\Delta_C, \Delta) \in I(\psi)$$

According to Definition 3.15 the above is equivalent to

$$(\Delta_C, \Delta) \in I(\varphi \Rightarrow \psi).$$

Summarizing, we have seen that it is adequate to require that for a model I of a set of rules

$$\mathcal{D} \times \mathcal{D} \subseteq I(\varphi \Rightarrow \psi)$$

holds for any ground instance $\varphi \Rightarrow \psi$ of a rule. According to Definition 3.15 this is exactly what is stated in Definition 3.30 above.

Remark 3.31 [Trivial (Maximal) Model] Let P_{UP} be an arbitrary update program, and let M be the constant mapping $M : \mathcal{B}_{DU} \to 2^{\mathcal{D}\times\mathcal{D}}$ with

$$M(p(\vec{t})) := \mathcal{D} \times \mathcal{D}$$

for all $p(\vec{t}) \in \mathcal{B}_{DU}$. Then M is a model of P_{UP}.

Proof: The assertion follows directly from Definitions 3.15 and 3.30. $\square$

The trivial interpretation defined in Remark 3.31 is a model of every program, as it satisfies the logical conditions. Informally speaking, it allows arbitrary state transitions for each defined update atom, in general more than specified by the update rules. Thus, in the following we aim at characterizing the intended update semantics by a minimal model.

Definition 3.32 [Interpretation Ordering] The ordering relation $\leq$ on interpretations is defined by:

$$I_1 \leq I_2 \quad :\Longleftrightarrow \quad I_1(p(\vec{t})) \subseteq I_2(p(\vec{t})) \text{ for all } p(\vec{t}) \in \mathcal{B}_{DU}$$

$\square$

Intuitively, $I_1 \leq I_2$ holds, if I_2 allows at least the same state transitions for a definable update atom as I_1. However, I_1 may be more restrictive.

Definition 3.33 [Minimal Model] Let $\mathcal{I}$ be a set of interpretations. A model $M \in \mathcal{I}$ is *minimal* w.r.t. $\mathcal{I}$, iff

$$M' \leq M \Longrightarrow M' = M$$

holds for any other model $M' \in \mathcal{I}$. A model M is *minimal*, iff it is minimal w.r.t. the set of all interpretations $I : \mathcal{B}_{DU} \to 2^{\mathcal{D}\times\mathcal{D}}$. $\square$

Lemma 3.34 Let I_1 and I_2 be interpretations, and let $\mathcal{P} \subseteq Pred_{UP}$ be a set of update predicates, such that $I_1(p(\vec{t})) = I_2(p(\vec{t}))$ holds for all $p \in \mathcal{P}$ and ground term tuples $\vec{t}$. Let U be a ground update goal, that does not contain any update predicate which is not in $\mathcal{P}$.

Then

$$I_1(U) = I_2(U).$$

Proof: We show by structural induction that $I_1(U) = I_2(U)$ for every ground update goal U satisfying the precondition. In each case, it suffices to show that for arbitrary $\Delta_C, \Delta \in \mathcal{D}$ the following holds:

$$(\Delta_C, \Delta) \in I_1(U) \Longrightarrow (\Delta_C, \Delta) \in I_2(U)$$

Base cases:

Essentially, we apply the Definition 3.15 of an interpretation.

1. DB literal

$$\begin{aligned} & (\Delta_C, \Delta) \in I_1(q(\vec{t})) \\ \Longrightarrow\ & q(\vec{t}) \in I_{DB}(\Delta_C) \text{ and } \Delta = \emptyset \\ \Longrightarrow\ & (\Delta_C, \Delta) \in I_2(q(\vec{t})) \end{aligned}$$

$$\begin{aligned} & (\Delta_C, \Delta) \in I_1(NOT\ q(\vec{t})) \\ \Longrightarrow\ & \neg q(\vec{t}) \in I_{DB}(\Delta_C) \text{ and } \Delta = \emptyset \\ \Longrightarrow\ & (\Delta_C, \Delta) \in I_2(NOT\ q(\vec{t})) \end{aligned}$$

2. NOP

$$\begin{aligned} & (\Delta_C, \Delta) \in I_1(NOP) \\ \Longrightarrow\ & \Delta = \emptyset \\ \Longrightarrow\ & (\Delta_C, \Delta) \in I_2(NOP) \end{aligned}$$

3. Basic update atom

$$\begin{aligned} & (\Delta_C, \Delta) \in I_1(INS\ r(\vec{t})) \\ \Longrightarrow\ & \Delta = \{+r(\vec{t})\} \\ \Longrightarrow\ & (\Delta_C, \Delta) \in I_2(INS\ r(\vec{t})) \end{aligned}$$

$$\begin{aligned} & (\Delta_C, \Delta) \in I_1(DEL\ r(\vec{t})) \\ \Longrightarrow\ & \Delta = \{-r(\vec{t})\} \\ \Longrightarrow\ & (\Delta_C, \Delta) \in I_2(DEL\ r(\vec{t})) \end{aligned}$$

4. Update atom

$$\begin{aligned} & (\Delta_C, \Delta) \in I_1(p(\vec{t})) \\ \underset{(p \in \mathcal{P})}{\Longrightarrow}\ & (\Delta_C, \Delta) \in I_2(p(\vec{t})) \end{aligned}$$

Induction step:

By the induction hypothesis, we can assume that $I_1(\varphi) = I_2(\varphi)$ for the direct proper subformula φ of the following composite formulae. Using Definition 3.15 we can reason as follows.

1. Concurrent conjunction

$$\begin{array}{rl}
& (\Delta_C, \Delta) \in I_1(\varphi, \psi) \\
\Longrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_1) \in I_1(\varphi) \\
& \text{and } (\Delta_C, \Delta_2) \in I_1(\psi) \\
& \text{and } \Delta = \Delta_1 \cup \Delta_2 \\
\Longrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
{\scriptstyle (IH)} & (\Delta_C, \Delta_1) \in I_2(\varphi) \\
& \text{and } (\Delta_C, \Delta_2) \in I_2(\psi) \\
& \text{and } \Delta = \Delta_1 \cup \Delta_2 \\
\Longrightarrow & (\Delta_C, \Delta) \in I_2(\varphi, \psi)
\end{array}$$

2. Sequential conjunction

$$\begin{array}{rl}
& (\Delta_C, \Delta) \in I_1(\varphi : \psi) \\
\Longrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_1) \in I_1(\varphi) \\
& \text{and } (\Delta_C \oplus \Delta_1, \Delta_2) \in I_1(\psi) \\
& \text{and } \Delta = \Delta_1 \oplus \Delta_2 \\
\Longrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
{\scriptstyle (IH)} & (\Delta_C, \Delta_1) \in I_2(\varphi) \\
& \text{and } (\Delta_C \oplus \Delta_1, \Delta_2) \in I_2(\psi) \\
& \text{and } \Delta = \Delta_1 \oplus \Delta_2 \\
\Longrightarrow & (\Delta_C, \Delta) \in I_2(\varphi : \psi)
\end{array}$$

3. Bulk quantification

   Consider a bulk quantification $\#\vec{X}\,\varphi$. All instances $\varphi[\vec{X} / \vec{t}]$ of the interior subgoal satisfy the preconditions of this lemma, i.e. by the induction hypothesis,

$$I_1(\varphi[\vec{X} / \vec{t}]) = I_2(\varphi[\vec{X} / \vec{t}])$$

holds. It follows that

$$T_{\varphi,\Delta_C} = \{\vec{t} \mid \exists \Delta' \in \mathcal{D} : (\Delta_C, \Delta') \in I_i(\varphi[\vec{X} / \vec{t}])\}$$

does not depend on the index i of the interpretation. Thus, by inserting the definitions, we get:

$$\begin{aligned} & (\Delta_C, \Delta) \in I_1(\# \vec{X}\, \varphi) \\ \Longrightarrow\ & (\Delta_C, \Delta) \in I_2(\# \vec{X}\, \varphi) \end{aligned}$$

□

Recall, that an interpretation of update formulae is completely determined by the interpretation of the definable update atoms. Thus a new interpretation can be defined by just supplying an explicit interpretation of the update atoms. For certain classes of logic programs the *model intersection property* [Llo87] holds, which allows to define a unique minimal model by intersecting all models of a program. In the following, we will apply the same technique to obtain minimal models of update programs that use the bulk quantifier only in a very limited way.

Lemma 3.35 Let $\mathcal{I}$ be a non-empty set of interpretations. Define a new interpretation I by

$$I(p(\vec{t})) := \bigcap_{I' \in \mathcal{I}} I'(p(\vec{t}))$$

for all $p(\vec{t}) \in \mathcal{B}_{DU}$. Let $\mathcal{P} \subseteq Pred_{UP}$ be a set of update predicates, such that for arbitrary $I', I'' \in \mathcal{I}$, all $p \in \mathcal{P}$ and ground term tuples $\vec{t}$, $I'(p(\vec{t})) = I''(p(\vec{t}))$ holds. Let U be a ground update goal such that all update predicates occurring in the scope of a bulk quantifier are in $\mathcal{P}$.

Then

$$I(U) \subseteq \bigcap_{I' \in \mathcal{I}} I'(U)$$

holds.

Proof: We show by structural induction that $I(U) \subseteq I'(U)$ holds for arbitrary $I' \in \mathcal{I}$, e.g. by comparing their elements $(\Delta_C, \Delta) \in \mathcal{D} \times \mathcal{D}$.

Base cases:

1. U does not contain any update predicate p with $p \notin \mathcal{P}$.

   We can apply Lemma 3.34, as for all update predicates $p \in \mathcal{P}$ and ground term tuples $\vec{t}$, $I(p(\vec{t})) = I'(p(\vec{t}))$ holds. Thus $I(U) = I'(U)$.
   Note, that U may be a composite formula.

2. U is an update atom $p(\vec{t}) \in \mathcal{B}_{DU}$ with $p \notin \mathcal{P}$.

   Then $I(p(\vec{t})) \subseteq I'(p(\vec{t}))$ by the definition of I.

Induction step:

By the induction hypothesis, $I(\varphi) \subseteq I'(\varphi)$ holds for any direct proper subformula φ of the following composite formulae.

1. Concurrent conjunction

$$\begin{array}{rl}
& (\Delta_C, \Delta) \in I(\varphi, \psi) \\
\Longrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_1) \in I(\varphi) \\
& \text{and } (\Delta_C, \Delta_2) \in I(\psi) \\
& \text{and } \Delta = \Delta_1 \cup \Delta_2 \\
\Longrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
{\scriptstyle (IH)} & (\Delta_C, \Delta_1) \in I'(\varphi) \\
& \text{and } (\Delta_C, \Delta_2) \in I'(\psi) \\
& \text{and } \Delta = \Delta_1 \cup \Delta_2 \\
\Longrightarrow & (\Delta_C, \Delta) \in I'(\varphi, \psi)
\end{array}$$

2. Sequential conjunction

$$\begin{array}{rl}
& (\Delta_C, \Delta) \in I(\varphi : \psi) \\
\Longrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_1) \in I(\varphi) \\
& \text{and } (\Delta_C \oplus \Delta_1, \Delta_2) \in I(\psi) \\
& \text{and } \Delta = \Delta_1 \oplus \Delta_2 \\
\Longrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
{\scriptstyle (IH)} & (\Delta_C, \Delta_1) \in I'(\varphi) \\
& \text{and } (\Delta_C \oplus \Delta_1, \Delta_2) \in I'(\psi) \\
& \text{and } \Delta = \Delta_1 \oplus \Delta_2 \\
\Longrightarrow & (\Delta_C, \Delta) \in I'(\varphi : \psi)
\end{array}$$

3. Bulk quantification

   Due to the precondition this case is covered by base case 1.

□

Lemma 3.36 [Model Intersection Property] Let P_{UP} be an update program, let $\mathcal{M}$ be a non-empty set of models of P_{UP}. Define a new interpretation M by

$$M(p(\vec{t})) := \bigcap_{M' \in \mathcal{M}} M'(p(\vec{t}))$$

for all $p(\vec{t}) \in \mathcal{B}_{DU}$. Let $\mathcal{P}$ be a set of update predicates, such that for arbitrary $M', M'' \in \mathcal{M}$, $p \in \mathcal{P}$, and ground term tuples $\vec{t}$, $M'(p(\vec{t})) = M''(p(\vec{t}))$ holds, i.e. the update predicates in $\mathcal{P}$ have a common interpretation. Let all update predicates, that occur in the scope of a bulk quantifier in the body of a rule of P_{UP}, also be elements of $\mathcal{P}$. Then M is a model of P_{UP}.

Proof: An interpretation I is a model of P_{UP}, iff every rule is interpreted by $\mathcal{D} \times \mathcal{D}$. By Definition 3.15, cases (5) and (6), this is exactly true for a rule r, if for every ground instance $U \Rightarrow p(\vec{t})$ of r the set inclusion $I(U) \subseteq I(p(\vec{t}))$ holds.

Now let $U \Rightarrow p(\vec{t})$ be a ground instance of a rule $r \in P_{UP}$. Because $\mathcal{M}$ is a set of models of P_{UP}, we know that $M'(U) \subseteq M'(p(\vec{t}))$ holds for every $M' \in \mathcal{M}$. We have to show that $M(U) \subseteq M(p(\vec{t}))$ holds, too.

As U is a goal that satisfies the conditions of Lemma 3.35 we can conclude

$$M(U) \subseteq \bigcap_{M' \in \mathcal{M}} M'(U) \subseteq \bigcap_{M' \in \mathcal{M}} M'(p(\vec{t})) = M(p(\vec{t}))$$

□

Theorem 3.37 [Existence of a Unique Minimal Model] Let P_{UP} be an update program such that no update predicate occurs in the scope of a bulk quantifier. Then P_{UP} has a unique minimal model M_{UP}.

Proof: P_{UP} has at least one model due to Remark 3.31. Let $\mathcal{M}$ be the non-empty set of all models of P_{UP}. Define M_{UP} by

$$M_{UP}(p(\vec{t})) := \bigcap_{M' \in \mathcal{M}} M'(p(\vec{t}))$$

for all $p(\vec{t}) \in \mathcal{B}_{DU}$. By Lemma 3.36 (with $\mathcal{P} := \emptyset$), M_{UP} is a model of P_{UP}. Due to the definition of M_{UP} by intersection, $M_{UP} \leq M'$ holds for all models of $M' \in \mathcal{M}$. The minimality and uniqueness of M_{UP} follow directly. □

Definition 3.38 Let $P = (P_{IDB}, P_{UP})$ be a database program such that in P_{UP} no update predicate occurs in the scope of a bulk quantifier. Let DB_0 be a (materialized) EDB instance.

Then

$$M_{UP}[P, DB_0]$$

denotes the unique minimal model of P_{UP} w.r.t. the fixed DB interpretation determined by DB_0 and P_{IDB}. □

The programs for which Theorem 3.37 holds can contain rules that arbitrarily – even recursively – combine update operations. However, Theorem 3.37 applies only, if bulk updates are only used at the level of basic updates and DB literals similar to Embedded SQL. For the following program, for example, a unique minimal model exists provided

q is not an update predicate.

$$p_1(X) \Leftarrow \# \, Y \, [\, q(X,Y), \; INS \; r(X,Y) \,].$$
$$p_2(X) \Leftarrow p_1(X) \; : \; [\, q(X,Y), \; p_2(Y) \,].$$

In the setting of Theorem 3.37, it is not allowed to compose a bulk update from a defined single update, like e.g. $\# \; X \; [\; s(X), \; p_1(X) \;]$. Nevertheless, we can define a particular minimal model for all programs which are stratified w.r.t. bulk quantification.

In a preliminary step, we formulate a necessary generalization of Theorem 3.37. Assume that an update program P_{UP} and an interpretation I is given. Our goal is to find a minimal model of P_{UP} such that all definable update atoms which are not defined by P_{UP} have the same interpretation as given by I. In effect, these non-defined update atoms are treated as basic updates.

Definition 3.39 [Restricted Interpretation] A *restricted interpretation* is an interpretation that is defined only for a subset of the definable update base $\mathcal{B}_{DU}$.

Let $\mathcal{P} \subseteq Pred_{UP}$ be a set of update predicates and I be an interpretation. By $I|_{\mathcal{P}}$ we denote the restriction of I to update atoms $p(\vec{t}) \in \mathcal{B}_{DU}$ with $p \in \mathcal{P}$. □

Lemma 3.40 Let P_{UP} be an update program, and let $\mathcal{P} \subseteq Pred_{UP}$ be a set of update predicates which do not occur in the head of any rule in P_{UP}. Let $I_{\mathcal{P}}$ be a restricted interpretation which assigns a fixed meaning to the predicates in $\mathcal{P}$. Let all update predicates occurring in the scope of a bulk quantifier in the body of a rule of P_{UP} also be elements of $\mathcal{P}$.

Then P_{UP} has a unique model M_{UP}, such that $M_{UP}|_{\mathcal{P}} = I_{\mathcal{P}}$ and M_{UP} is minimal w.r.t. the set of all interpretations I with $I|_{\mathcal{P}} = I_{\mathcal{P}}$.

Proof: In analogy to Remark 3.31, one can define a model M by

$$M(p(\vec{t})) := \begin{cases} I_{\mathcal{P}}(p(\vec{t})), & \text{if } p \in \mathcal{P} \\ \mathcal{D} \times \mathcal{D}, & \text{otherwise} \end{cases}$$

for all $p(\vec{t}) \in \mathcal{B}_{DU}$. M is a trivial model of P_{UP} with $M|_{\mathcal{P}} = I_{\mathcal{P}}$. Let $\mathcal{M}$ be the non-empty set of all models M of P_{UP} such that $M|_{\mathcal{P}} = I_{\mathcal{P}}$. Define M_{UP} by

$$M_{UP}(p(\vec{t})) := \bigcap_{M' \in \mathcal{M}} M'(p(\vec{t}))$$

for all $p(\vec{t}) \in \mathcal{B}_{DU}$. By Lemma 3.36, M_{UP} is a model of P_{UP}, and $M_{UP}|_{\mathcal{P}} = I_{\mathcal{P}}$. In addition, M_{UP} is minimal and unique w.r.t. $\mathcal{M}$. □

Lemma 3.41 [Stratification] For an update program P_{UP}, that is stratified w.r.t. bulk quantification, there exists a (disjoint) stratification $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of the update predicates $Pred_{UP}$ and a corresponding partition $R_1, \ldots, R_n$ of the program P_{UP} such that the following properties hold for $1 \le i \le n$:

- All predicates occurring in the head of a rule of R_i are contained in $\mathcal{P}_i$.

- For all update predicates p which occur in a rule body of R_i, $p \in \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_i$ holds.
- If an update predicate p occurs in the scope of a bulk quantifier of a rule of R_i, then $p \in \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}$ holds. In the case $i = 1$, no update predicate occurs in the scope of a bulk quantifier.

Proof: Such a stratification exists by the precondition, i.e. by the properties of the dependency graph of P_{UP}, cf. Definition 2.16. See [ABW88] for the analogous proof for stratified negation. □

Definition 3.42 [Construction of an Intended Model] For an update program P_{UP}, that is stratified w.r.t. bulk quantification, we construct an *intended model* M_{UP} as follows:

Choose a stratification $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of the update predicates $Pred_{UP}$ w.r.t. bulk quantification and a corresponding partition $R_1, \ldots, R_n$ of the program P_{UP} satisfying the properties of Lemma 3.41.

Define a sequence of interpretations $I_1, \ldots, I_n$ bottom-up by induction on $1 \leq i \leq n$.

Base case: $i = 1$

Define I_1 as the unique minimal model of R_1, which exists by Theorem 3.37, i.e. $I_1 := M_{UP}[(P_{IDB}, R_1), DB_0]$.

Induction step: $i > 1$

Let I_{i-1}, i.e. the interpretation of the lower program fragment $R_1 \cup \ldots \cup R_{i-1}$, be already defined.

Apply Lemma 3.40 to construct I_i as the unique minimal model of R_i w.r.t. the fixed restricted interpretation $I_{i-1}|_{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}}$ of the predicates in $\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}$.

Finally, define $M_{UP} := I_n$. □

Theorem 3.43 [Well-Definedness] An intended model M_{UP} of a program P_{UP} constructed as in Definition 3.42 is indeed a minimal model.

Proof: Let a stratification $\mathcal{P}_1, \ldots, \mathcal{P}_n$ (and $R_1, \ldots, R_n$) be given, and let the interpretations $I_1, \ldots, I_n$ be defined as in Definition 3.42.

We show by induction on $1 \leq i \leq n$ that I_i is a minimal model of the program fragment $R_1 \cup \ldots \cup R_i$. The assertion then follows from the case $i = n$.

Base case: $i = 1$

By definition, I_1 is a minimal model of R_1.

Induction step: $i > 1$

By the induction hypothesis, I_{i-1} is a minimal model of $R_1 \cup \ldots \cup R_{i-1}$.

First, we prove that I_i is a model of $R_1 \cup \ldots \cup R_i$. Let $U \Rightarrow p(\vec{t})$ be a ground instance of a rule $r \in R_1 \cup \ldots \cup R_i$. In analogy to the proof of Lemma 3.36, we have to show that $I_i(U) \subseteq I_i(p(\vec{t}))$. We distinguish the following two cases:

1. $r \in R_1 \cup \ldots \cup R_{i-1}$

   From the stratification condition follows that p and all update predicates occurring in U are contained in $\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}$. Hence, the following equations hold:

   $$I_i(U) = I_i|_{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}}(U) = I_{i-1}|_{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}}(U) = I_{i-1}(U)$$
   $$I_i(p(\vec{t})) = I_i|_{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}}(p(\vec{t})) = I_{i-1}|_{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}}(p(\vec{t})) = I_{i-1}(p(\vec{t}))$$

   From the induction hypothesis follows:

   $$I_i(U) = I_{i-1}(U) \subseteq I_{i-1}(p(\vec{t})) = I_i(p(\vec{t}))$$

2. $r \in R_i$

   By definition is I_i a model of the program R_i and thus

   $$I_i(U) \subseteq I_i(p(\vec{t})).$$

We will now prove that I_i is minimal. Let to this end I'_i be a model of $R_1 \cup \ldots \cup R_i$ with $I'_i \leq I_i$. We have to show that $I'_i = I_i$.

Define an interpretation I'_{i-1} by

$$I'_{i-1}(p(\vec{t})) := \begin{cases} I'_i(p(\vec{t})), & \text{if } p \in \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1} \\ \emptyset, & \text{otherwise} \end{cases}$$

for all $p(\vec{t}) \in \mathcal{B}_{DU}$.

Trivially $I'_{i-1} \leq I'_i \leq I_i$ holds. As $I_{i-1}|_{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}} = I_i|_{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}}$, also $I'_{i-1} \leq I_{i-1}$ holds. In analogy to case 1 above, one can show that I'_{i-1} is a model of $R_1 \cup \ldots \cup R_{i-1}$, because I'_i is supposed to be a model. By the induction hypothesis (minimality) $I'_{i-1} = I_{i-1}$ follows and thus

$$I'_i|_{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}} = I_{i-1}|_{\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}}.$$

As by assumption I'_i is a model of $R_1 \cup \ldots \cup R_i$, I'_i is also a model of R_i. The equality shown above says that I'_i coincides with I_{i-1} on the predicates in $\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_{i-1}$. Again by assumption we have $I'_i \leq I_i$. The minimality of I_i as a model of R_i with the coincidence property implies $I'_i = I_i$.

□

Remark 3.44 [Independence of the Stratification] We claim that the intended model does not depend on the chosen stratification. The proof essentially resembles the analogous proof for stratified negation [ABW88, Ull88]. □

Definition 3.45 Let $P = (P_{IDB}, P_{UP})$ be a database program such that P_{UP} is stratified w.r.t. bulk quantification. Let DB_0 be a (materialized) EDB instance.

Then

$$M_{UP}[P, DB_0]$$

denotes the (unique) intended model of P_{UP} w.r.t. the fixed DB interpretation determined by DB_0 and P_{IDB}. □

Note, that Definition 3.45 generalizes Definition 3.38.

We can now define the possible updates of a top-level query asked against the extended deductive database. Retrieval queries are not considered here.

Definition 3.46 [Updates of a Query] Let P be a database program and DB_0 be the physical database state. Let Q be an update query of the form "$\Leftarrow U$.", where U does not contain any free variables.

The set

$$Updates[P, DB_0](Q) := \{\Delta \in \mathcal{D} \mid (\emptyset, \Delta) \in M_{UP}[P, DB_0](U)\}$$

contains the *possible updates* for Q. □

To commit a transaction invoked by a top-level query, one possible update has to be chosen non-deterministically and all its basic updates have to be performed physically on the EDB. Note, that at this time, the *physical* EDB instance and thus the DB interpretation and the intended model of the update program change.

3.3 Example: Wine Cellar (Cont.)

Recall the example of Section 2.5. Let the physical EDB instance be given by:

Wine	W#	Name	Grape	Year
	16	richeaume	sauvignon	1990
	18	chapellet	chardonnay	1987
	19	chapellet	chardonnay	1984
	32	mondavi	gamay	1989
	33	mondavi	gamay	1972

Store	W#	Amount
	16	0
	18	0
	19	2
	32	21
	33	1

Let the IDB be defined by the following rules.

$$
\begin{aligned}
out(W) &\ :-\ store(W, 0).\\
red(W) &\ :-\ wine(W, N, gamay, Y).\\
red(W) &\ :-\ wine(W, N, merlot, Y).
\end{aligned}
$$

We will illustrate the update semantics as defined above using the following update rules.

$$
\begin{aligned}
order(W, D) &\Leftarrow store(W, X),\ Y = X + D,\\
&\ DEL\ store(W, X),\ INS\ store(W, Y).\\
consume(W, D) &\Leftarrow store(W, X),\ X \geq D,\ Y = X - D,\\
&\ DEL\ store(W, X),\ INS\ store(W, Y).\\
order\_some\_out &\Leftarrow out(W),\ order(W, 24).\\
order\_all\_out &\Leftarrow \#\ W\ [\ out(W),\ order(W, 24)\ ].\\
order\_all\_white &\Leftarrow \#\ W\ [\ order(W, 24),\ NOT\ red(W),\ NOT\ rose(W)\ ].\\
consume\_red &\Leftarrow red(W),\ consume(W, 1).\\
consume\_then\_order &\Leftarrow consume(33, 1)\ :\ order\_all\_out.\\
order\_and\_consume &\Leftarrow order(18, 24),\ consume\_red.
\end{aligned}
$$

The goal $order(W, D)$ is successful for a wine W, if the wine exists and its quantity is increased by D. For W=16 and D=24, for instance, the resulting update request set is

$$\Delta = \{-store(16, 0), +store(16, 24)\}.$$

The semantics of *consume* is similar. Note, that we do not get any possible update in the case D=0, as the update request sets of the subgoals are incompatible.

Due to the local variable W in its defining rule, *order_some_out* is a non-deterministic operation. During evaluation, W may be instantiated by 16 or 18. Both possible updates

$$\Delta_1 = \{-store(16, 0), +store(16, 24)\}$$

and

$$\Delta_2 = \{-store(18, 0), +store(18, 24)\}$$

reflect the fact that 24 bottles of *one* missing wine are bought. A similar non-determinism can be found in the semantics of *consume_red*.

Now consider the definition of *order_all_out*. The semantics says that we must find all successful instantiations of W w.r.t. the interior goal $out(W), order(W, 24)$, i.e. instantiations for which at least one possible update exists. This is true for W=16 and W=18, the corresponding update request sets are the same as for *order_some_out*. These possible updates have to be merged into one final update request set

$$\Delta = \{-store(16,0), -store(18,0), +store(16,24), +store(18,24)\}.$$

Operationally, both steps can be handled simultaneously using aggregation.

order_all_white works similarly. The computed possible update is

$$\begin{aligned}\Delta = \{ \quad & -store(16,0), -store(18,0), -store(19,2),\\ & +store(16,24), +store(18,24), +store(19,26) \quad \}.\end{aligned}$$

At the operational level, a correct sideways information passing strategy has to ensure that no floundering occurs, i.e. the variable W is bound prior to evaluating the filter literals $NOT\ red(W)$ and $NOT\ rose(W)$. The semantics defined above, however, is not affected by operational considerations.

consume_then_order represents the consumption of one bottle of wine number 33 followed by a purchase of all empty wines. In the semantics, the transition to the intermediate state which results from the application of $consume(33,1)$ is reflected by the (hypothetical) execution $\oplus_E$ of the update request set

$$\Delta_1 = \{ \quad -store(33,1), +store(33,0) \quad \}.$$

In this hypothetical state the goal *order_all_out* is evaluated and returns a second update request set

$$\begin{aligned}\Delta_2 = \{ \quad & -store(16,0), -store(18,0), -store(33,0),\\ & +store(16,24), +store(18,24), +store(33,24) \quad \}.\end{aligned}$$

Although Δ_1 and Δ_2 are incompatible, as $+store(33,0) \in \Delta_1$ and $-store(33,0) \in \Delta_2$ holds, the evaluation succeeds due to the semantics of the sequential composition $\oplus$, cf. Example 3.9:

$$\begin{aligned}\Delta = \{ \quad & -store(16,0), -store(18,0), -store(33,1), -store(33,0),\\ & +store(16,24), +store(18,24), +store(33,24) \quad \}\end{aligned}$$

order_and_consume is also defined using two complex update operations. However, these operations are composed by concurrent conjunction. The evaluation does not produce a hypothetical intermediate state. Consequently, $order(18,24)$ and *consume_red* can be evaluated simultaneously. As both operations cannot conflict – they operate on disjoint partitions of the database – the semantics is equal to that of any sequential composition, i.e. $order(18,24) : consume\_red$ or $consume\_red : order(18,24)$. The possible updates are

$$\Delta = \{-store(18,0), -store(32,21), +store(18,24), +store(32,20)\}$$

and

$$\Delta' = \{-store(18,0), -store(33,1), +store(18,24), +store(33,0)\},$$

i.e. we again encounter non-determinism.

4 Implementation by Program Transformation

An extended deductive database program $P = (P_{IDB}, P_{UP})$ can be transformed into a conventional deductive database program $P' = P'_{IDB} \cup P'_{UP}$ which explicitly represents update request sets and encodes the update semantics given in Section 3. In this paper, we will not go into the details of the transformation.

Query processing is performed for the transformed database program and consists of two steps: In the *evaluation phase*, the possible updates, i.e. a set of update request sets, is computed without changing the EDB instance DB_0. In a subsequent *materialization phase*, one set Δ is chosen and the collected update requests are applied by an external transaction which yields the new *physical* database state. In this setting, we can guarantee the atomicity and recoverability of a transaction. Some form of isolation is also ensured, as other transactions cannot see any effect or the intermediate states before 'Commit'. To guarantee full isolation, the evaluation phase must also run under the control of a transaction manager. The development of an adequate transaction concept is part of our future work and lies outside the scope of this paper.

The transformed programs can be evaluated in any deductive database system or logic programming environment that can evaluate (stratified) aggregation [MPR90] and provides the built-in predicates needed. Due to the occurrence of non-range-restricted rules, the system must be able to handle non-ground answers or apply some form of top-down control to instantiate the rules properly, e.g. a suitable form of the magic set transformation.

In our prototype, the transformation of the extended logic programs is done by a compiler written with Lex and Yacc [LMB90]. Provided that the IDB program P_{IDB} was modularly stratified, the object program P' can be evaluated by the deductive database system *LOLA* [FSS91] using the local magic set transformation [ZF96]. Once a consistent set of basic update requests has been computed and chosen for materialization, the updates are performed by a built-in operation via a side effect.

We are on the way to implement a new version of the prototype mainly reconsidering the data structures for the update request sets and improving the consistency checking.

5 Related Work

Transaction Logic [BK94, BK96] is one of the concepts which have significantly inspired our own work. It forms a logical framework for the representation of updates in which the sequential composition of updates is handled explicitly and arbitrary basic updates can be integrated. Transaction Logic has no explicit construct for bulk updates. Of course, the effect of bulk updates can be obtained using recursive rules (see e.g. [MW88] for an example). However, this seems to be somewhat unintuitive and requires temporary base relations to ensure termination. We consider a recursive or iterative approach as a low-level solution, possibly generated by a transaction scheduler. In [BKC93] a relational assignment operator for copying an IDB relation into the EDB is proposed. However, this operation is atomic and outside the scope of Transaction Logic. Bulk updates cannot be composed from existing single updates. Concurrent Transaction Logic [BK96] provides an explicit concurrency construct, by which one can specify that subtransactions are to be performed in an interleaving fashion. This form of concurrency requires a stronger form of compatibility assertion than the implicit concurrency in our concept. While in our approach, for a goal G_1, G_2 both G_1 and G_2 operate on the same state – which may be assumed as consistent – and produce its specific increments, in the interleaving semantics of Transaction Logic both goals concurrently manipulate the global database state. This leads to more non-determinism and verification problems like in other concurrent programming languages.

U-Datalog [BGM95] has made the first attempt to perform bulk updates by aggregation of success paths in the resolution tree of a query. However, this approach cannot handle sequential composition. Sequential updates can only be specified as complex transactions at the top-level. It is *not* possible to formulate such a complex operation by a rule. The reason is, that update aggregation and sequentiality are not integrated into the resolution process. During the logical evaluation of a query all subgoals refer to the original database state. The aggregation is rigorous and not controllable. One always gets a bulk update effect and cannot specify update alternatives as in Transaction Logic. In addition, the aggregation of the update requests does not distinguish between logical conjunction and disjunction and thus does not resemble the conventional semantics of Horn clauses and abduction.

The expressibility of the language DLP proposed by Manchanda and Warren [MW88] is comparable to that of the rule fragment of Transaction Logic restricted to basic insertions and deletions. Like in our approach, the implementation uses a non-immediate update semantics. The formal semantics, however, is based on a dynamic logic in which a formula is interpreted by pairs of database states. Consequently, the same problems in specifying bulk updates arise as in Transaction Logic [BK94, BK96]. Concurrent conjunction cannot be specified.

Chen [Che95] defines an update calculus and a corresponding update algebra. The latter is an extension of the relational algebra and – similarly to our approach – deals with increments, concurrent/sequential composition, and consistency constraints. In contrast, the update calculus is based on abduction and minimal changes of a relational database, where the basic update atoms are considered as assertions for the next state. It should be noted that neither IDB rules (or relational views), nor update rules are considered. But our update concept can probably be also implemented using the update algebra.

An alternative rule paradigm considers rules as condition-action specifications. Basic updates in a rule head are triggered, when the condition of the rule body is satisfied. There exist many different semantics depending on how the rules (set-oriented/tupleoriented, deterministically/non-deterministically, simultaneously/successively, etc.) are evaluated, see e.g. [AV88]. Some new approaches like Statelog [LHL95] and the concept of Zaniolo [Zan93] define a clear semantics based on temporal logic programming [AM89] by incorporating states into the rule formalism. However, the specification of update operations using active rules and explicit state identifiers is not very convenient. In a recent paper, Ludäscher, May, and Lausen [LML96] present an extension of Statelog by update procedures and sequential composition. The new language further abstracts from states. In essence, its semantics encodes a top-down control into a bottom-up, data/event-driven evaluation framework which may be also characterized by a Kripke structure. If we do not consider bulk updates, we get a similar semantics for our approach by applying a magic set transformation to the transformed programs (cf. Section 4).

Kakas, Mancarella [KM90] and Bry [Bry90] use abductive reasoning for translating intensional updates into changes on the EDB. They do not explicitly describe update methods but derive them from view definitions and integrity constraints. In contrast, we use special rules which *explicitly* describe the update operations, although their evaluation has an abductive flavour. As proposed in [MW88], suitable rules for performing view updates can be generated by a precompiler in our framework. In this setting, the possibility of expressing non-deterministic update operations is useful.

Some other approaches dealing with updates in deductive databases should be mentioned: Naish, Thom, and Ramamohanarao [NTR87] show the problems that arise from non-declarative update constructs in logic programming and propose a clean concept based on deferred updates. Reiter [Rei95] describes possible actions and its effects in the situation calculus and addresses the frame problem. In the language GOLOG [LRL$^+$96], procedural structures and complex operations are introduced, their semantics is described by macro expansion and second order constructs. Like in our approach, all states generated by a sequence of actions are represented by an increment w.r.t. a certain initial state. Kowalski [Kow92] uses the event calculus for characterizing dynamics, which has some similarities with the situation calculus. In [SWM93] a dynamic logic for verifying database updates is developed. However, the updates are programmed in a language like Embedded SQL.

6 Conclusion and Outlook

We have described a rule-based update specification language which generalizes the classical deductive database languages. Update operations can be hierarchically constructed and the conventional database update operations are supported. The main contribution is the definition of a model-theoretic semantics based on an extended logic. To be able to deal with bulk updates and concurrent conjunction, update formulae are not interpreted by pairs of states as in conventional modal logics. Instead, we use a semantics based on deferred updates, i.e. differences between consecutive states.

The consistency constraints are a point of discussion. Up to now, we require a minimal consistency, which means that no complementary basic update requests, i.e. insert and delete of the same tuple, occur in a state increment, such that it characterizes a unique and consistent new state. One future research topic will be to consider all update specifications as transaction specifications for which a specific compatibility relation can be provided. It is not clear, if the compatibility constraints should affect the logical semantics or only lead to freedom in the evaluation. Maybe, it is necessary to distinguish between logical constraints and optimization information.

The restriction of basic updates to insertions and deletions has the nice property that the concurrent/sequential composition and bulk updates can be represented as algebraic operations on sets of basic update requests. If we allow complex basic operations, a more sophisticated structure of the state increments will be needed. It is a topic of future work to find a tractable generalization of the current approach.

Another possible variation within the current update semantics is the improvement of the hypothetical reasoning. Up to now, the transformed rules require to recompute the intensional database for every intermediate state. To avoid recomputation of unchanged parts it is possible to refer to the results of the last committed state and to deal with increments also for the intensional relations. It is necessary to reconsider the rules for hypothetical reasoning by taking view maintenance into account. Teniente and Urpí [TU95] propose a suitable transformation.

Acknowledgement

The transformation of update programs into conventional deductive database programs has been implemented by Martin Köhrmann as part of his diploma project [Köh96].

References

[ABW88] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AM89] M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8:277–295, 1989.

[AV88] S. Abiteboul and V. Vianu. Procedural and declarative database update languages. In *Proc. 7th ACM Symp. on Principles of Database Systems*, pages 240–250, 1988.

[BGM95] E. Bertino, G. Guerrini, and D. Montesi. Towards deductive object databases. *Theory and Practice of Object Systems*, 1(1):19–39, 1995.

[BK94] A. J. Bonner and M. Kifer. An overview of Transaction Logic. *Theoretical Computer Science*, 133:205–265, 1994.

[BK96] A. J. Bonner and M. Kifer. Concurrency and communication in Transaction Logic. In *Proc. Joint Int. Conf. and Symp. on Logic Programming (JICSLP '96), Bonn, Germany*, pages 142–156, 1996.

[BKC93] A. J. Bonner, M. Kifer, and M. Consens. Database programming in Transaction Logic. In *Proc. 4th Int. Workshop on Database Prog. Languages, New York City, USA*, pages 309–337, 1993.

[Bry90] F. Bry. Intensional updates: Abduction via deduction. In *Proc. 7th Int. Conf. on Logic Programming (ICLP '90), Jerusalem, Israel*, pages 561–575, 1990.

[Che95] W. Chen. Declarative updates of relational databases. *ACM Transactions on Database Systems*, 20(1):42–70, 1995.

[EK89] K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In *Proc. 6th Int. Conf. on Logic Programming (ICLP '89), Lisbon, Portugal*, pages 234–254, 1989.

[FSS91] B. Freitag, H. Schütz, and G. Specht. *LOLA* – a logic language for deductive databases and its implementation. In *Proc. 2nd Int. Symp. on Database Systems For Advanced Applications, Tokyo, Japan*, pages 216–225, 1991.

[GV89] G. Gardarin and P. Valduriez. *Relational Databases and Knowledge Bases.* Addison-Wesley, 1989.

[KM90] A. C. Kakas and P. Mancarella. Database updates through abduction. In *Proc. 16th Int. Conf. on Very Large Data Bases*, pages 650–661, 1990.

[Köh96] M. Köhrmann. Realisierung komplexer Updates in deduktiven Datenbanken. Diplomarbeit, Universität Passau (FMI), 1996.

[Kow92] R. Kowalski. Database updates in the event calculus. *Journal of Logic Programming*, 12:121–146, 1992.

[LHL95] B. Ludäscher, U. Hamann, and G. Lausen. A logical framework for active rules. In *Proc. 7th Int. Conf. on Management of Data (COMAD '95), Pune, India*, 1995.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 2nd edition, 1987.

[LMB90] J. R. Levine, T. Mason, and D. Brown. *lex & yacc*. O'Reilly & Associates, Inc., 1990.

[LML96] B. Ludäscher, W. May, and G. Lausen. Nested transactions in a logical language for active rules. In *Proc. Int. Workshop on Logic in Databases (LID '96), Pisa, Italy*, 1996.

[LRL+96] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. submitted, 1996.

[Mos85] J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. MIT Press, 1985.

[MPR90] I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The magic of duplicates and aggregates. In *Proc. 16th Int. Conf. on Very Large Data Bases*, pages 264–277, 1990.

[MW88] S. Manchanda and D. S. Warren. A logic-based language for database updates. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 363–394. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[NTR87] L. Naish, J. A. Thom, and K. Ramamohanarao. Concurrent database updates in PROLOG. In *Proc. 4th Int. Conf. on Logic Programming (ICLP '87), Melbourne, Australia*, volume 1, pages 178–195, 1987.

[Rei95] R. Reiter. On specifying database updates. *Journal of Logic Programming*, 25(1):53–91, 1995.

[SWM93] P. Spruit, R. Wieringa, and J. J. Meyer. Dynamic Database Logic: the first-order case. In U. W. Lipeck and B. Thalheim, editors, *Modelling Database Dynamics*, pages 103–120. Springer-Verlag, 1993.

[TU95] E. Teniente and T. Urpí. A common framework for classifying and specifying deductive database updating problems. In *Proc. Int. Conf. on Data Engineering (ICDE '95), Taipei, Taiwan*, pages 173–182, 1995.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1988.

[vGRS91] A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(2):620–650, 1991.

[WS92] G. Weikum and H. J. Schek. Concepts and applications of multilevel transactions and open nested transactions. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[Zan93] C. Zaniolo. On the unification of active databases and deductive databases. In *Proc. 11th British Nat. Conf. on Databases, Keele, UK*, pages 23–39, 1993.

[ZF96] U. Zukowski and B. Freitag. Adding flexibility to query evaluation for modularly stratified databases. In *Proc. Joint Int. Conf. and Symp. on Logic Programming (JICSLP '96), Bonn, Germany*, 1996.