Logical Transactions and Serializability

Carl-Alexander Wichert
Burkhard Freitag
Alfred Fent

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

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

Universität Passau
Technical Report MIP-9807

Abstract

The concept of logic databases can serve as a clear and expressive foundation of various kinds of information systems. However, classical logic languages refer to a single database state, whereas in modern information systems it is necessary to take dynamic behaviour into account. For this purpose, several update languages have been proposed, among them [AHV95, BK94, BK96, Che97, LHL95, MW88, MBM97, WF97a], which allow to specify complex transitions from one database state to another. From the evaluation point of view, a complex state transition can and should be considered as a transaction. Up to now, the isolation property of transactions has been poorly addressed in the context of logic update languages, although it is an important problem even for classical, sequential transactions (see [BHG87] for instance). In this paper we present the ULTRA approach and describe how isolation of independent transactions can be supported. We illustrate the results using an extended example.

1 Introduction and Motivation

In [WF97a, WF97b] we propose the rule-based update language ULTRA, whose semantics is based on the concept of deferred updates. Updates are implemented by goals, and rules can be used to define complex operations to be reused in other goals. The concept is similar to procedures/functions in classical programming languages. Thus, it is possible to build arbitrarily complex database operations in a modular fashion. Constructs for concurrent composition, sequential composition, and bulk updates are provided, that subsume the classical database operations. The semantics of an update goal w.r.t. a fixed database state is defined in terms of update request sets which collect insertion and deletion requests for ground EDB tuples and thus describe (hypothetical) transitions to other database states. This form of *deferred updates* was already used in Heraclitus [GHJ96], U-Datalog [MBM97], and the update calculus of Chen [Che95]. As two or more update request sets can be better combined than two or more different database extensions, the concept turns out to be well-suited for the representation of bulk updates. We are also able to formulate a concurrent conjunction which is compositional, i.e. the semantics of the concurrent conjunction φ, ψ is defined in terms of the semantics of both subformulas φ and ψ. Compositionality is important for modularization and avoids the well-known problems of interleaving parallelism found in various programming languages, e.g. Java. Note, that Chen independently worked on the integration of deferred updates with the logic programming paradigm and developed a concept [Che97] that is similar to the ULTRA approach [WF97a]. For the discussion of other related work, e.g. [AHV95, BK94, BK96, LHL95, MW88, MBM97], we refer the reader to [WF97a, WF97b] and the survey article [BK98]. The semantics of ULTRA has been slightly modified as compared to [WF97a, WF97b].

From the evaluation point of view, a complex state transition specified in an update language can and should be considered as a transaction. In particular, the well-known ACID properties [BHG87] should hold. While the atomicity and durability properties are easy to guarantee in most logical update concepts, up to now, the isolation property of transactions has been poorly addressed, although it is an important problem even for classical, sequential transactions [BHG87]. In this paper we describe how transaction isolation – in terms of serializability – can be achieved for the ULTRA concept. To be able to deal with the problem while relying neither on a particular evaluation strategy, nor on technical details of the extensional database system, we enrich the logical semantics of [WF97a]. This is done in such a way that all elementary read operations affecting base relations, which may occur during an evaluation, are recorded as *read tags* in the update request sets, too. Consequently, the isolation property can be investigated at the semantical level. Before the computed update requests of a top-level goal are performed on the database, the additional logging information provided through the read tags is used to determine whether the current transaction is compatible with other independent transactions that have been evaluated in parallel. This essentially implies the desired isolation property. The isolation check can be easily integrated into the current ULTRA prototype. It should be mentioned, however, that we assume that

the extensional database system does not allow *local* transactions [BHG87], i.e. other transactions than invoked by the logical layer (ULTRA).

Note, that we do not attempt to embed the operational transaction processing into the logical framework. Our update programs adopt the single transaction view known from database systems. Thus, we define a logical semantics of single transactions which supports the (local) logging of read and update operations and is therefore suitable as the basis of a sensible operational semantics. Although the read information is included in the update request sets and thus appears as part of the semantics of update formulas, it can be ignored by a programmer of complex operations, who is only interested in the resulting state changes.

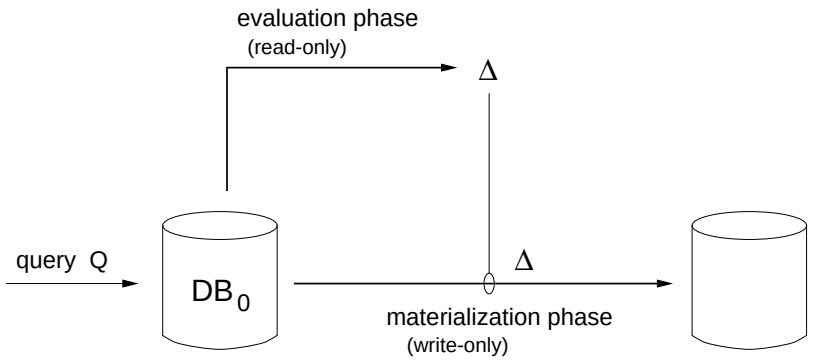

Figure 1: Evaluation and materialization phase

Our prototype system, which has been implemented on top of the deductive database system *LOLA* [ZF97], uses a two-phase transaction processing method, see Figure 1. A transaction is invoked by the submission of a top-level update query Q. In the *evaluation phase*, sets Δ of basic update requests are computed which potentially cause the transition from the initial database state DB_0 to some new final state. This evaluation is done according to the logical semantics defined below. Note, that updates are not executed in this phase, i.e. the database state remains unchanged. After one or more 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. The serializability results can be well integrated into this architecture. Before the materialization is started, the collected read tags are analyzed to detect read/write conflicts with other transactions that have been evaluated in the same state and have already been materialized. If there are no conflicts, the materialization of the current transaction can take place, otherwise the transaction must be aborted and restarted in the new physical database state. The examination of the read tags corresponds to the read (lock) certification in non-blocking, optimistic transaction protocols [GR93], e.g. ODL [HD91]. The materialization phase is completely independent from the evaluation phase, as all relevant information is stored in the update request sets. It must only be guaranteed that the evaluation engine always accesses a single, consistent database state. The need to determine the dependency between

intensional queries and extensional updates (see [Elk90] for instance) forces us to keep the granularity of the read information at the level of EDB relations, i.e. when an IDB predicate is evaluated, all EDB relations which contribute to the state interpretation of the queried predicate are marked as read. The consideration of finer-grained, more concise logging information is a topic of future research.

The rest of the paper is organized as follows: After formalizing syntax and semantics of the update language ULTRA in Sections 2 and 3, we present a sufficient condition (in terms of update request sets) to guarantee the isolated execution of two concurrent transactions in Section 4. The technical descriptions are illustrated by a non-trivial running example, namely a calendar tool. Finally, we give an outline of our prototypical architecture in Section 5 and conclude the paper by Section 6. In the appendices, one can find the detailed proofs of assertions made in Sections 3 and 4 as well as the complete running example.

The work described in this paper has been funded by the German Research Agency (DFG) under contract number Fr 1021/3-1. A short version of this paper has been published in the LNCS series of the Springer-Verlag [WFF98].

2 The Update Language ULTRA

We consider extended logic databases (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, i.e. DATALOG with function symbols and negation, and an *update program* consisting of update rules to be defined below. Correspondingly, we distinguish the sets of *EDB predicates*, *IDB predicates*, and *definable update predicates*. The definable update predicates can be regarded as names of complex update operations.

Definition 2.1 [Terms, Atoms, DB Literals] A *term* is a type- and arity-conform composition of function symbols, constants, and variables. A term without variables is called *ground*.

Atoms are of the form $p(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms and p is an n-ary predicate. We can distinguish between *EDB atoms*, *IDB atoms*, and *definable update atoms* depending on which set of predicates p belongs to. *DB literals* are EDB/IDB atoms or negated EDB/IDB atoms, the latter written as $NOT\ q(t_1, \ldots, t_n)$. The groundness property of terms can be generalized to atoms, literals, and formulas defined below in the natural way. □

Definition 2.2 The set of all ground terms is called the *Herbrand universe* and is denoted by $\mathcal{U}$. The set of all ground EDB and IDB atoms is called the *Herbrand base* and is denoted by $\mathcal{B}$. The set of all ground definable update atoms is called the *definable update base* and is denoted by $\mathcal{B}_{DU}$. □

In the following, we use the abbreviation $\vec{t}$ for a finite sequence $t_1, \ldots, t_n$ of terms. Similarly, we write $\vec{X}$ for a finite sequence $X_1, \ldots, X_n$ of variables.

Definition 2.3 [EDB Instance] A set DB of ground EDB atoms $r(\vec{t}) \in \mathcal{B}$ is called an *EDB instance*. □

After having recalled some preliminaries, we now define the specific elements of the ULTRA language. Its basic elements are called update literals.

Definition 2.4 [Update Literals] The set of *update literals* is defined by the following cases:

1. Every DB literal is an *update literal*.
2. NOP is an *update literal* ("no operation").
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* ("insert", "delete").
4. Every definable update atom $p(\vec{t})$ is an *update literal*.

□

Update literals specify retrieval (DB literals), simple modifications on the EDB (INS, DEL), and calls to defined complex operations (definable update atoms). Below, we will see how update literals can be composed.

To illustrate the properties and capabilities of the ULTRA language and the formal results presented in this paper, we use a simplified version of a personal calendar. In the examples only a part of the full program (see Appendix B) is shown.

Example 2.5 [Personal Calendar] In the calendar model used in our examples, appointments have a unique identifier and may occupy one or more consecutive time slots. Time slots may be of any length, but we assume that a slot represents one hour. The calendar is based on two EDB relations: a relation $date(Day, Slot, ID)$ which associates a time slot $Slot$ on day Day with an appointment identifier ID and a relation $description(ID, Text)$ that contains the descriptive text $Text$ for an appointment with identifier ID. The reserved identifier 0 represents free time slots. In a real world implementation, free slots would of course not be recorded in the database but rather be computed, either by joining the *date* relation with a relation representing all possible slots per day, or by using a computed predicate that enumerates the possible slots. To simplify the example we consider only one fixed week.

In the rest of the paper we always refer to the EDB instance DB_0 shown in Table 1. The constant *mon* refers to the Monday of the fixed week. This EDB instance can be interpreted as follows: The owner of the calendar has a meeting with Mr. Dean on Monday from 12pm to 2pm, she wants to visit the hairdresser on Monday at 3pm, etc. The time slots on Monday from 10am to 12pm and from 2pm to 3pm are not reserved, yet. To keep the tables short we omit the entries for the other days.

date	*Day*	*Slot*	*ID*
	mon	9	21
	mon	10	0
	mon	11	0
	mon	12	7
	mon	13	7
	mon	14	0
	mon	15	8
	mon	16	10

description	*ID*	*Text*
	7	Meeting Mr. Dean
	8	Hairdresser
	10	Review
	21	Call Mr. Miller

Table 1: Sample EDB instance

In this setting, the basic update atoms $DEL\ date(mon, 10, 0)$, $INS\ date(mon, 10, 23)$, and $INS\ description(23, \text{“Presentation”})$ are possible. They specify the deletion of the tuple $(mon, 10, 0)$ from the *date* relation, the insertion of the tuple $(mon, 10, 23)$ into the *date* relation, and the insertion of the tuple (23, “Presentation”) into the *description* relation, respectively. Together, this amounts to inserting a new appointment “Presentation” on Monday from 10am to 11am into the calendar database. □

Update literals can be composed to form more complex update formulas. First, we define the general notion of update formulas, and then we define the subset of update goals.

We define a concurrent and a sequential conjunction for specifying simultaneous and successive update operations, respectively, and a bulk quantification for specifying set-oriented updates in terms of single updates. The intuitive reading of the bulk quantification is “for all $\vec{X}$ such that A holds perform update φ”. The implication and the universal quantification are used to construct update rules for the definition of complex update operations.

Definition 2.6 [Update Formulas] The set of *update formulas* is defined inductively by the following cases:

1. Every update literal is an *update formula*.
2. Let φ and ψ be update formulas. The implication $\varphi \rightarrow \psi$ is an *update formula*.
3. Let φ be an update formula and $\vec{X}$ be a finite sequence $X_1, \ldots, X_n$ of variables. Then $\forall \vec{X}\, \varphi$ is an *update formula*.
4. Let φ and ψ be update formulas. The *concurrent conjunction* φ, ψ and the *sequential conjunction* $\varphi : \psi$ are *update formulas*.
5. Let A be an EDB or an IDB atom, let φ be an update formula, and let $\vec{X}$ be a finite sequence of variables. The *bulk quantification* $\#\vec{X}\,[A \mapsto \varphi]$ is an *update formula*.

If necessary we use square brackets "[...]" to indicate the operator bindings in a composite formula. □

Definition 2.7 [Update Goals] An *update goal* is an update formula not containing the implication "$\rightarrow$" or the $\forall$-quantification, i.e. the set of update goals is defined by cases 1, 4, and 5 of Definition 2.6. □

Update goals form rule bodies or top-level update queries. See below for some examples.

Definition 2.8 A variable occurring inside a formula is *free* unless being in the scope of a $\forall$-quantifier or a bulk quantifier "#". An update formula is called *ground*, if it does not contain any free variables. □

Definition 2.9 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 $\vec{t}$. □

Definition 2.10 [IDB Rules] *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 (denoted by $q(\vec{t}) \leftarrow L_1, \ldots, L_n$). Rules with an empty body are called *facts*. □

Definition 2.11 [Update Rules] An *update rule* is a universal closure $\forall(U \rightarrow p(\vec{t}))$, also denoted by $p(\vec{t}) \leftarrow U$, where $p(\vec{t})$ is a definable update atom and U is an update goal. Rules without a head $\leftarrow U$ denote *(top-level) queries*. An update rule with an empty body is called an *update fact*. The empty body corresponds to the update literal NOP. □

Definition 2.12 [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. □

Remark 2.13 [Syntactical Restriction] We do not allow update goals with free variables as top-level queries. □

The semantics of update queries that have solutions with different instances of the free variables is not obvious. By Remark 2.13, we avoid this problem. Note, that we independently allow EDB/IDB queries with free variables as pure retrieval transactions (see Section 4).

Example 2.14 [Personal Calendar (Cont.)] In our calendar example there is – among others – the update rule

$$do_insert(D,S,L,T) \quad \leftarrow \quad \mathsf{newid}(ID),\ do_allocate(D,S,L,ID),\ INS\ description(ID,T)$$

which specifies the insertion of a new entry with descriptive text T, starting at time slot S on day D and having a duration of L slots. The definition of the *do_insert* predicate consists of three components. newid is a (built-in) predicate that returns a

new identifier ID. From the semantical point of view it can be regarded as a predicate which is true for exactly one constant, however the constant varies between independent evaluations. Such a feature is provided in many database systems to avoid concurrency problems when searching for unused key values. *do_allocate* is a recursive auxiliary update predicate which allocates L consecutive time slots needed for identifier ID in relation *date* (see Appendix B for details). As before, $INS\ description(ID, T)$ inserts the descriptive text of the new appointment into the relation *description*.

Since the subgoals $do_allocate(D, S, L, ID)$ and $INS\ description(ID, T)$ refer to different EDB relations, they can be evaluated simultaneously and thus are combined by concurrent conjunction ",". Note, that also the subgoal $\mathsf{newid}(ID)$ is connected by concurrent conjunction. From the logical point of view it can be evaluated concurrently with the rest. However, in any sensible implementation it would be implicitly scheduled to be evaluated first. This kind of partial sequentialization can be achieved by a suitable sideways information passing strategy, but we will not consider such topic in this paper.

The rule above shows one of the main advantages of the rule-based approach with updates in the rule body: complex update operations can be easily composed from predefined (sub-)operations. Every constructed operation has a clear logical semantics (cf. Section 3.2). As will be shown in Section 4, in ULTRA even transactional behaviour can be accomplished.

Similarly, a predicate *do_delete* for the deletion of an entry can be defined that uses an auxiliary predicate *do_deallocate*. The latter predicate may be defined as follows using the bulk quantifier, which says that for all entries in *date* with appointment identifier ID the corresponding slot has to be marked free.

$$\begin{aligned} do_deallocate(ID) \quad \leftarrow \quad & \#\ D, S \\ & [\ date(D, S, ID)\ \mapsto \\ & \quad [\ DEL\ date(D, S, ID),\ INS\ date(D, S, 0)\ ]\ ] \end{aligned}$$

Appointments are frequently moved from one time interval to another. This can be achieved using the update predicate *do_move* shown below which moves an appointment with identifier ID to day D and time slot S. Again, *do_move* is built upon the predefined auxiliaries *do_allocate* and *do_deallocate*. The predicate *duration_of* counts the number L of slots reserved for the appointment ID.

$$\begin{aligned} do_move(ID, D, S) \quad \leftarrow \quad & [\ duration_of(ID, L),\ do_deallocate(ID)\ ]\ : \\ & do_allocate(D, S, L, ID) \end{aligned}$$

Here we need the sequential conjunction ":" to specify that the second subgoal is to be evaluated in the (hypothetical) database state that results from completing the operation specified by the subgoals on the left. In our example, this is necessary to ensure that an entry does not block its own movement. So, first all time slots assigned to that particular entry are freed, and then the allocation of the slots at the new starting time is attempted. For *do_move* to succeed, both subgoals of the rule body

must succeed. □

For the development of isolation concepts we have to regard the dependencies within the IDB program. The definitions are as usual, cf. [ABW88].

Definition 2.15 [Dependency Graph] Let P_{IDB} be an IDB program, and let p and q be EDB or IDB predicates. p *depends directly* on q, denoted by $p \leftarrow_{P_{IDB}} q$, iff there exists a rule in P_{IDB} such that p is the head predicate and q occurs in the body. The relation $\leftarrow_{P_{IDB}}$ on the set of EDB and IDB predicates defines the edges of the *dependency graph* of P_{IDB}. □

Note, that we do not consider dependencies w.r.t. update predicates, which result from the update program. The relation $\leftarrow_{P_{IDB}}$ only refers to the retrieval part, i.e. EDB and IDB predicates.

The set of EDB predicates a predicate q depends on will be used to detect read/write conflicts between two transactions.

Definition 2.16 Let P_{IDB} be an IDB program and q be an EDB or IDB predicate. We define

$$Def_E[P_{IDB}](q) := \{r \mid r \text{ EDB predicate } \wedge\ q \leftarrow^*_{P_{IDB}} r\},$$

where $\leftarrow^*_{P_{IDB}}$ denotes the reflexive and transitive closure of the relation $\leftarrow_{P_{IDB}}$. □

Whenever a predicate q is accessed by a transaction, the predicates contained in $Def_E[P_{IDB}](q)$ will be kept in a log as if they were accessed instead of q. Then it is easy to detect conflicts between read access and basic update operations on the EDB.

3 Semantics of Formulas and Programs

In this section we present the model-theoretic semantics of ULTRA (cf. [WF97a]), which is based on the concept of *deferred updates*. Essentially this means that updates are not immediately executed when they are "encountered" during evaluation. Instead, each update goal determines (one or more) *possible* transitions that can be *derived* without affecting the database state and may or may not be materialized later. Not earlier than at *materialization time* does a possible transition cause an actual transition from the current database state to some new state. Thus, all states except the initial state are *hypothetical* for the logical semantics. Obviously, read access to an EDB relation cannot be deferred but must be done immediately. However, any read access is logged for certification purposes. Deferred updates with logged read operations can be well combined with optimistic synchronization protocols [BHG87, GR93].

Possible transitions are represented as sets of basic update requests. From the point of view of the update semantics an update request can be considered as an assertion about the next database state. Update request sets store also a read tag for each

EDB relation accessed during derivation and can thus be considered as local logs (cf. [BHG87, GR93]).

3.1 Interpretation of Update Formulas

Definition 3.1 [Update Request] A *ground basic update request* has the form $+r(\vec{t})$ or $-r(\vec{t})$, 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, i.e. a tuple in an EDB relation.

The read tags defined below are used to record possible conflicts between read and update operations. Intuitively, a read tag $?r$ expresses that the EDB relation associated with r has (possibly) contributed to the result. Thus, to ensure transaction isolation, the read tags have to be considered during certification, i.e. before the computed update request sets are actually materialized (see also Section 1). Obviously, the granularity of the read tags is rather coarse. Thus, conflicts may be noticed at the syntactical level which are – semantically – no proper conflicts. The determination of a minimal set of relevant EDB data for any derived information, however, is a hard problem [Elk90] and lies outside the scope of this paper. But note, that even many standard database systems use read locks on whole base relations, if serializability is to be ensured.

Definition 3.2 [Read Tag] A *read tag* has the form $?r$, where r is an EDB predicate. □

In the following, read tags are treated like update requests. Note, however, that read access is an immediate operation as opposed to update operations which are deferred. Only the certification of a read access is also deferred.

Definition 3.3 [Update Request Set, Consistency] An *update request set* Δ is a set of ground basic update requests and read tags.

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 $-r(\vec{t}) \in \Delta$.

Let $\mathcal{D}$ denote the set of all consistent update request sets taken from the restriction of the Herbrand base $\mathcal{B}$ to EDB atoms. □

Definition 3.4 [Write-Compatibility] Two or more consistent update request sets are called *write-compatible* with each other, if their union is consistent. Otherwise they are called *incompatible*. □

Note, that Definition 3.4 defines an intra-transaction compatibility which is necessary to define a clear semantics for the concurrent conjunction. Within a transaction, a read access does supposedly not conflict with an update. Update goals composed by the concurrent conjunction are evaluated w.r.t. the same current state and specify individual update request sets which are merged in order to express a simultaneous transition.

Definition 3.5 [Concurrent Composition] The *concurrent composition* $\sqcup$ of two update request sets Δ_1 and Δ_2 is defined by the set union, i.e.

$$\Delta_1 \sqcup \Delta_2 := \Delta_1 \cup \Delta_2.$$

Similarly, we define the concurrent composition $\bigsqcup$ for a set of update request sets. □

Example 3.6 Consistent updates request sets are for example:

$$\begin{aligned}
\Delta_1 &= \{?date, -date(mon, 10, 0), +date(mon, 10, 23)\} \\
\Delta_2 &= \{+description(23, \text{“Presentation”})\} \\
\Delta_3 &= \{?date, -date(mon, 10, 0), +date(mon, 10, 23), \\
&\quad + description(23, \text{“Presentation”})\}
\end{aligned}$$

Δ_1 and Δ_2 are write-compatible, as their union Δ_3 is consistent. Further, $\Delta_3 = \Delta_1 \sqcup \Delta_2$, i.e. Δ_3 is the concurrent composition of Δ_1 and Δ_2.

The following update request sets are consistent, but incompatible with each other:

$$\begin{aligned}
\Delta_4 &= \{?date, -date(mon, 10, 0)\} \\
\Delta_5 &= \{?date, +date(mon, 10, 0)\}
\end{aligned}$$

The concurrent composition of Δ_4 and Δ_5 contains both the insertion and deletion of the fact $date(mon, 10, 0)$ and thus is not a consistent update request set. □

Whereas the concurrent composition is used for combining update request sets that may be materialized simultaneously, we now define a sequential composition for combining two update request sets that must be materialized successively.

Definition 3.7 [Sequential Composition] The *sequential composition* $\oplus$ of two update request sets Δ_1 and Δ_2 is defined by:

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

□

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. Recall, that within the logical framework update request sets are not *physically* applied to DB_0 and every state except the physical EDB instance DB_0 is *hypothetical*.

$$DB_0 \xrightarrow{\Delta} DB$$

In particular, the current database state is represented by the update request set Δ_C. At the implementation level, Δ_C can be considered as a redo log. Note, that every database state comprises the contents of both EDB and IDB, but changes (hypothetical or physical) are only applied to the EDB.

Definition 3.8 [Execution] The *(hypothetical) execution* $\oplus_E$ of a consistent update request set $\Delta \in \mathcal{D}$ w.r.t. the EDB instance DB is defined by

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

□

Example 3.9 The execution of the update request set Δ_3 of Example 3.6 w.r.t. the EDB instance DB_0 of Example 2.5 results in the new state

$$\begin{array}{lcl} DB_0 \oplus_E \Delta_3 \;=\; DB_0 \;\cup\; \{ & date(mon, 10, 23), \\ & description(23, \text{“Presentation”}) \;\} \\ \setminus \;\; \{ & date(mon, 10, 0) \;\}. \end{array}$$

□

In [WF97b] we prove several useful properties about the algebraic operators defined above. The following proposition collects some important assertions.

Proposition 3.10 [Algebraic Properties]

1. The concurrent composition $\sqcup$ is commutative and associative.
2. The sequential composition $\oplus$ is associative.
3. Let Δ_1 and Δ_2 be not necessarily consistent update request sets. Then the following holds:

   (a) $\Delta_1 \in \mathcal{D} \;\wedge\; \Delta_2 \in \mathcal{D} \;\Longleftarrow\; \Delta_1 \sqcup \Delta_2 \in \mathcal{D}$
   (b) $\Delta_1 \in \mathcal{D} \;\wedge\; \Delta_2 \in \mathcal{D} \;\Longrightarrow\; \Delta_1 \oplus \Delta_2 \in \mathcal{D}$

4. Let DB be an EDB instance and $\Delta_1, \Delta_2 \in \mathcal{D}$ be consistent update request sets. Then the following holds:

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

   If Δ_1 and Δ_2 are write-compatible, then even the following equalities hold:

   (b) $(DB \oplus_E \Delta_1) \oplus_E \Delta_2 = DB \oplus_E (\Delta_1 \sqcup \Delta_2) = (DB \oplus_E \Delta_2) \oplus_E \Delta_1$

Proof: See Appendix A. □

In our approach, the state semantics, i.e. the semantics of the current IDB, is given by the *well-founded model* [vGRS91]. It is denoted $WFM(\Phi)$ for a union $\Phi = DB \cup P_{IDB}$ of an EDB instance DB and a set of IDB rules P_{IDB}. 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. We use the prefix $\neg$ when referring to a false atom. Note, that the ULTRA approach can accommodate other state semantics without any essential modifications.

Definition 3.11 [DB Interpretation] Let DB_0 be the physical EDB instance and P_{IDB} the IDB program. We define the *DB interpretation* I_{DB} w.r.t. DB_0 and P_{IDB}, which assigns to every $\Delta \in \mathcal{D}$ a well-founded model by

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

Note, that terms are interpreted by the usual Herbrand pre-interpretation [Llo87]. □

In our prototype implementation and even in the running example, we make use of stratified aggregation and extra-logical constructs (built-ins). This is not considered in the pure semantics, but can be integrated under certain conditions, see e.g. [vG92] for well-founded aggregation.

We will now define the interpretation of update formulas. 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 [Llo87], 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 formulas 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 definable 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.

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

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. In Example 4.2 in Section 4, we consider a non-deterministic update operation.

Definition 3.12 [Interpretation of Update Formulas] Let the physical EDB instance DB_0 and the IDB program P_{IDB} be fixed. Let I_{DB} be the state interpretation w.r.t. to DB_0 and P_{IDB} (cf. Definition 3.11). Let I_{UP} be an interpretation of the definable update atoms, i.e. a mapping

$$I_{UP} : \mathcal{B}_{DU} \to 2^{\mathcal{D} \times \mathcal{D}}.$$

We define the *interpretation I of update formulas* as an extension of I_{UP} to arbitrary ground update formulas inductively as follows. Recall, that $\mathcal{D}$ only contains consistent update request sets.

(1) DB literal

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

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

$(\Delta_C, \Delta) \in I(q(\vec{t})) \;:\Longleftrightarrow$

$q(\vec{t}) \in I_{DB}(\Delta_C)$ and $\Delta = \{?r \mid r \in Def_E[P_{IDB}](q)\}$

$(\Delta_C, \Delta) \in I(NOT\ q(\vec{t})) \;:\Longleftrightarrow$

$\neg q(\vec{t}) \in I_{DB}(\Delta_C)$ and $\Delta = \{?r \mid r \in Def_E[P_{IDB}](q)\}$

(2) NOP

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

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

(3) Basic update atom

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

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

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

(4) Definable update atom

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

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

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

(5) Implication

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

$(\Delta_C, \Delta) \in I(\varphi \rightarrow \psi) \;:\Longleftrightarrow$

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

(6) Universal quantification

Let φ be an update formula containing at most the free variables $\vec{X}$.

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

$(\Delta_C, \Delta) \in I(\forall \vec{X}\, \varphi) \;:\Longleftrightarrow$

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:

$(\Delta_C, \Delta) \in I(\varphi, \psi) \;:\Longleftrightarrow$

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 $\Delta = \Delta_1 \sqcup \Delta_2$

(8) Sequential conjunction

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

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

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 A be an EDB/IDB atom with predicate q that contains exactly the variables $\vec{X}$, let φ be an update formula containing at most the free variables $\vec{X}$, and let $I(\varphi[\vec{X}/\vec{t}])$ be already defined for every ground tuple $(\vec{t})$.

For $\Delta_C \in \mathcal{D}$ let $T_{A,\Delta_C} := \{(\vec{t}) \mid A[\vec{X}/\vec{t}] \in I_{DB}(\Delta_C)\}$.

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

$(\Delta_C, \Delta) \in I(\#\vec{X}\,[A \mapsto \varphi]) \ :\Longleftrightarrow$

$T_{A,\Delta_C} = \emptyset$ and $\Delta = \{?r \mid r \in Def_E[P_{IDB}](q)\}$

or

$T_{A,\Delta_C} \neq \emptyset$ and there exists a function $f : T_{A,\Delta_C} \to \mathcal{D}$ such that:

$\forall(\vec{t}) \in T_{A,\Delta_C} : (\Delta_C, f(\vec{t})) \in I(\varphi[\vec{X}/\vec{t}])$

and $\Delta = \{?r \mid r \in Def_E[P_{IDB}](q)\} \ \sqcup \ \bigsqcup_{(\vec{t}) \in T_{A,\Delta_C}} f(\vec{t})$

□

Definition 3.12 essentially formalizes what kind of update request sets Δ are necessary to satisfy the corresponding update formulas. In operational terms this amounts to the definition of the update request sets Δ that are generated during the evaluation. Note, how basic update atoms are related to basic update requests and how the concurrent/sequential conjunction is related to the concurrent/sequential composition of update request sets. Case (9) concerning the bulk quantification should be explained in more detail. Consider a fixed current state represented by Δ_C. First, all ground term tuples $(\vec{t})$, such that the instance $A[\vec{X}/\vec{t}]$ of the atom A is true in the current state, have to be collected in T_{A,Δ_C}. Then either T_{A,Δ_C} is empty and the bulk quantification is successful without any single updates, or for each term tuple $(\vec{t})$ in T_{A,Δ_C}, a corresponding update request set (w.r.t. the ground update formula $\varphi[\vec{X}/\vec{t}]$) must be chosen and added to Δ, which represents the resulting bulk update. The choice is reflected by the function f. In both cases, also read tags for the atom A are added to Δ.

In [WF97b] some algebraic properties of the conjunction constructs have been stated. The main results, that also hold for the extended semantics defined above, are restated in the following proposition.

Proposition 3.13 [Algebraic Properties] The following properties hold for the connectives "," and ":".

1. The concurrent conjunction "," is commutative.
2. The concurrent conjunction "," is associative.
3. The sequential conjunction ":" is associative.

Proof: See Appendix A. □

Due to the associativity of "," and ":", we do not need to use precedence brackets in formulas of the form $\varphi_1, \ldots, \varphi_n$ and $\varphi_1 : \ldots : \varphi_n$.

Example 3.14 Consider the update formula

$$\varphi :\equiv date(mon, 10, 0),\ DEL\ date(mon, 10, 0),\ INS\ date(mon, 10, 23)$$

which is an instance of one of the rule bodies defining the update predicate *do_allocate* (see Appendix B). Let the EDB instance DB_0 of Example 2.5 be given. By Definition 3.12 the following holds regardless of the interpretation I_{UP} of definable update atoms:

$$\begin{array}{lcl} (\emptyset, \{?date\}) & \in & I(date(mon, 10, 0)) \\ (\emptyset, \{-date(mon, 10, 0)\}) & \in & I(DEL\ date(mon, 10, 0)) \\ (\emptyset, \{+date(mon, 10, 23)\}) & \in & I(INS\ date(mon, 10, 23)) \end{array}$$

Applying case (7) twice we can deduce

$$(\emptyset, \{?date, -date(mon, 10, 0), +date(mon, 10, 23)\}) \in I(\varphi).$$

Up to now, we only considered the initial state DB_0 as the current state. Next, we consider a hypothetical current state, where the fact $date(mon, 10, 0)$ has been removed from the EDB:

$$(\{-date(mon, 10, 0)\}, \{?date, -date(mon, 10, 0), +date(mon, 10, 23)\}) \notin I(\varphi)$$

holds, because $date(mon, 10, 0)$ is false in the hypothetical state and case (1) of Definition 3.12 is thus not applicable. □

3.2 Semantics of Update Programs

Let an extended deductive database with database program (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 3.11, 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 definable update atoms that respects the intended semantics of P_{UP}. Recall, that in this setting, an interpretation I of the ground update formulas is completely determined by I_{UP}. We will therefore identify I_{UP} and the corresponding interpretation I of ground update formulas and also write $I_{UP}(\varphi)$ for arbitrary ground update formulas φ.

Definition 3.15 [Models of an Update Program] An interpretation I_{UP} of update formulas 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. □

An interpretation I_{UP} is a model of an update program P_{UP}, iff $I_{UP}(\varphi) \subseteq I_{UP}(\psi)$ holds for every ground instance $\varphi \rightarrow \psi$ of a rule in P_{UP}. Informally speaking, this means that each rule head allows at least all state transitions specified by the corresponding rule body. The formal definition of a model states that every rule must be interpreted as valid w.r.t. arbitrary pairs of update request sets, where each pair corresponds to a state transition.

Definition 3.16 Let $\mathcal{I}$ denote the set of all interpretations $I : \mathcal{B}_{DU} \rightarrow 2^{\mathcal{D}\times\mathcal{D}}$.
Further define $I_\bot, I_\top : \mathcal{B}_{DU} \rightarrow 2^{\mathcal{D}\times\mathcal{D}}$ by

$$I_\bot(p(\vec{t})) := \emptyset \quad \text{and} \quad I_\top(p(\vec{t})) := \mathcal{D} \times \mathcal{D}$$

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

Definition 3.17 [Interpretation Ordering] The ordering relation $\leq$ on interpretations is defined by:

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

□

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.

Lemma 3.18 [Lattice of Interpretations] The set $\mathcal{I}$ of all interpretations together with its ordering relation $\leq$ (cf. Definition 3.17) forms a complete lattice.

The bottom element and the top element of $\mathcal{I}$ are the constant mappings $I_\bot$ and $I_\top$, respectively.

Proof: The assertions follow from similar properties of the power-set lattice. □

Remark 3.19 [Trivial (Maximal) Model] Every update program P_{UP} has at least one trivial model, i.e. the maximal interpretation $I_\top$.

Proof: The assertion follows directly from Definitions 3.12 and 3.15. □

Definition 3.20 [Minimal Model] A model $M \in \mathcal{I}$ is *minimal*, iff

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

holds for any other model $M' \in \mathcal{I}$. □

Lemma 3.21 Let $\mathcal{J} \subseteq \mathcal{I}$ be a non-empty set of interpretations. Define a new interpretation I by

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

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

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

holds for every ground update goal U.

Proof: The assertion can be proved by structural induction. See Appendix A for details. □

Lemma 3.22 [Model Intersection Property] Let P_{UP} be an update program, let $\mathcal{M} \subseteq \mathcal{I}$ 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 $p(\vec{t}) \in \mathcal{B}_{DU}$. Then M is a model of P_{UP}.

Proof: The assertion follows easily from Lemma 3.21 (see Appendix A). □

Theorem 3.23 [Existence of a Unique Minimal Model] Let P_{UP} be an update program. Then P_{UP} has a unique minimal model M_{UP}.

Proof: The assertion follows from Remark 3.19 and Lemma 3.22 (see Appendix A for details). □

Note, that the semantics of the bulk quantifier presented in this paper fits with the model intersection property. Thus it is not necessary to select a special minimal model as done by the stratification concept in [WF97a, WF97b].

Example 3.24 [Personal Calendar (Cont.)] Consider the update rules

$$\begin{array}{lcl} do_allocate(D,S,1,ID) & \leftarrow & date(D,S,0), \\ & & DEL\ date(D,S,0),\ INS\ date(D,S,ID) \\ do_allocate(D,S,L,ID) & \leftarrow & L > 1,\ do_allocate(D,S,1,ID), \\ & & S1 = S+1,\ L1 = L-1, \\ & & do_allocate(D,S1,L1,ID) \end{array}$$

of our running example (see Appendix B) and recall Example 3.14, in particular the definition

$$\varphi :\equiv date(mon,10,0),\ DEL\ date(mon,10,0),\ INS\ date(mon,10,23)\ .$$

Because

$$(\emptyset, \{?date, -date(mon,10,0), +date(mon,10,23)\}) \in I(\varphi)$$

holds for arbitrary interpretations $I \in \mathcal{I}$, it follows that

$$\begin{array}{l} (\emptyset, \{?date, -date(mon,10,0), +date(mon,10,23)\}) \\ \quad \in M(do_allocate(mon,10,1,23)) \end{array}$$

holds for every model M of P_{UP}. Of course, there are many models M for which

$$(\{-date(mon, 10, 0)\}, \{?date, -date(mon, 10, 0), +date(mon, 10, 23)\}) \\ \in M(do_allocate(mon, 10, 1, 23))$$

holds, but as this is not justified by any rule body, it does not hold for the minimal model of P_{UP}. □

We now provide an inductive fixpoint characterization of the unique minimal model of a given update program which will be helpful in proving properties of the minimal model. The well-known results for definite programs [Llo87] essentially hold for the update programs in ULTRA, too.

Definition 3.25 [Immediate Consequence Operator] Let P_{UP} be an update program. We define an *immediate consequence operator* $T_{P_{UP}} : \mathcal{I} \to \mathcal{I}$ on the set of all interpretations as follows:

$$T_{P_{UP}}(I)(p(\vec{t})) := \{ \; (\Delta_C, \Delta) \in \mathcal{D} \times \mathcal{D} \mid \\ \text{there exists a ground instance } U \to p(\vec{t}) \\ \text{of a rule } r \in P_{UP}, \text{ such that } (\Delta_C, \Delta) \in I(U) \; \}$$

□

To be able to apply the fixpoint theory we must show the monotonicity of the immediate consequence operator.

Lemma 3.26 [Monotonicity] The immediate consequence operator $T_{P_{UP}}$ of Definition 3.25 is monotonic.

Proof: The proof is straightforward. Lemma 3.21 applied to sets $\{I_1, I_2\} \subseteq \mathcal{I}$ of two interpretations with $I_1 \leq I_2$ can serve as an auxiliary. □

Lemma 3.27 Let P_{UP} be an update program and $I \in \mathcal{I}$ be an arbitrary update interpretation. Then the following holds:

I is a model of P_{UP}, iff $T_{P_{UP}}(I) \leq I$ holds, i.e. iff I is a pre-fixpoint of $T_{P_{UP}}$.

Proof: The assertion can be easily shown using Definitions 3.15 and 3.25. □

Definition 3.28 The *ordinal powers* $T_{P_{UP}} \uparrow \gamma$ of the immediate consequence operator of an update program P_{UP} are defined as usual (see Appendix A and [Llo87]). □

Theorem 3.29 [Fixpoint Characterization] Let P_{UP} be an update program and M_{UP} be its unique minimal model. Then M_{UP} is the least fixpoint of $T_{P_{UP}}$, and there exists a closure ordinal $\alpha_{P_{UP}}$, such that $M_{UP} = T_{P_{UP}} \uparrow \alpha_{P_{UP}}$.

Proof: The well-known proof for the case of definite programs [Llo87] can be easily adapted. The preconditions are given by Lemmata 3.26 (monotonicity) and 3.27 (pre-fixpoint). □

Remark 3.30 If the Herbrand universe $\mathcal{U}$ is finite or P_{UP} uses bulk quantification exclusively at the level of basic updates (cf. bulk updates in SQL), it is possible to show

that the immediate consequence operator $T_{P_{UP}}$ is continuous. In this case $\alpha_{P_{UP}} \leq \omega$ holds [Llo87], i.e. the minimal model of P_{UP} can be generated by the iterated application of $T_{P_{UP}}$ at most to the first limit ordinal. In the general case, however, $T_{P_{UP}}$ may be not continuous. □

Finally, we introduce some notation for the minimal model of an update program in an extended deductive database.

Definition 3.31 Let $P = (P_{IDB}, P_{UP})$ be a database program, and 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}. □

4 Transactions and Serializability

In this section we show how the logical semantics can be used to support isolation of independent top-level transactions. A new top-level transaction is invoked by submitting a top-level update query to the ULTRA system. We first define the possible transitions of a top-level query asked against the extended deductive database.

Definition 4.1 [Possible Transitions] Let P be a database program and let 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.

All update request sets $\Delta \in \mathcal{D}$ such that $(\emptyset, \Delta) \in M_{UP}[P, DB_0](U)$ are called *possible transitions* for Q. □

To commit a transaction invoked by a top-level query, one possible transition has to be chosen non-deterministically and all its basic updates have to be physically applied to the EDB. Note, that at this time, the *physical* EDB instance and thus the DB interpretation and the minimal model of the update program change. A transaction will logically fail, if there exists no possible transition. Whenever multiple independent transactions are processed simultaneously, also read/write conflicts have to be considered. The read tags contained in a possible transition refer to information which must not be changed by concurrent transactions.

Example 4.2 [Personal Calendar (Cont.)] Our personal calendar features a predicate $do_insert_on_day(D, L, T)$ which can be used to insert an appointment having a description T and a duration of L slots on a day D without specifying the starting time. Instead, all those time slots an appointment of the requested duration can be assigned to are looked up, then a non-deterministic choice is performed among the possibilities and the corresponding database entry is inserted. The defining rule reads as follows:

$$do_insert_on_day(D, L, T) \quad \leftarrow \quad free(D, S, L) \ : \ do_insert(D, S, L, T)$$

Now, suppose the update query

$$Q \equiv \leftarrow do_insert_on_day(mon, 1, \text{“Call Mr. Martin”})$$

is issued in database state DB_0 of Example 2.5. Using the IDB predicate $free$ (see Appendix B), it is deduced that the time slots on Monday at 10am, 11am, or 2pm can hold an appointment of one hour. Therefore, the following three possible transitions for query Q are generated, assuming the assigned identifier is 28:

$$\begin{aligned}
\Delta^{(1)} &= \{?date, -date(mon, 10, 0), +date(mon, 10, 28), \\
&\quad + description(28, \text{“Call Mr. Martin”})\} \\
\Delta^{(2)} &= \{?date, -date(mon, 11, 0), +date(mon, 11, 28), \\
&\quad + description(28, \text{“Call Mr. Martin”})\} \\
\Delta^{(3)} &= \{?date, -date(mon, 14, 0), +date(mon, 14, 28), \\
&\quad + description(28, \text{“Call Mr. Martin”})\}
\end{aligned}$$

□

As free variables in DATALOG queries usually have a set-oriented interpretation, which is different from the provided non-deterministic update semantics, we do not allow free variables in top-level update queries. This seems a little odd at a first glance, because in conventional database applications pure retrieval transactions are of course also necessary. However, the latter can be easily integrated into the overall approach provided they produce update request sets which contain read tags according to the queried relations.

Definition 4.3 [Retrieval Queries] Let Q be a retrieval query of the form $\leftarrow q(\vec{t})$ about an EDB or an IDB atom $q(\vec{t})$ which may contain free variables. Then we assign to Q the “possible transition” $\Delta := \{?r \mid r \in Def_E[P_{IDB}](q)\}$. □

Like update transactions, a pure retrieval transaction also has to materialize its (unique) possible transition. However, this materialization only checks the validity of the read access, but does not perform any updates on the EDB.

Example 4.4 [Personal Calendar (Cont.)] $browse(D, S, T)$ (see Appendix B) is an IDB predicate that relates a description T to the appointment on day D beginning at time slot S. $browse$ is defined as the natural join of the $date$ and the $description$ relation. The following retrieval query Q inquires the agenda for entries on Monday at 12pm:

$$Q \equiv \leftarrow browse(mon, 12, T)$$

This retrieval query produces the binding “Meeting Mr. Dean” for the free variable T. In addition, the update request set

$$\Delta = \{?date,\ ?description\}$$

is generated. Both read tags contained in Δ have to be checked before the query result is returned to the top-level. □

In the remainder of this section we show how the read tags can be actually used to validate all read accesses to the EDB or IDB. We define a sufficient condition to ensure the isolation of two independent transactions that are evaluated w.r.t. the same initial EDB instance.

Definition 4.5 [Read-Isolation] A consistent update request set $\Delta \in \mathcal{D}$ is called *read-isolated* from another consistent update request set $\Delta_0 \in \mathcal{D}$, if there exists no update request $+r(\vec{t}) \in \Delta_0$ or $-r(\vec{t}) \in \Delta_0$ such that the read tag $?r$ is contained in Δ. □

Informally speaking, Δ is read-isolated from Δ_0, iff the basic update requests of Δ_0 cannot change the relations that were read during the computation of Δ.

Example 4.6 [Personal Calendar (Cont.)] Given the two transactions

$$\begin{aligned} T_1 &\equiv \leftarrow do_insert(mon, 10, 1, \text{“Presentation”}) \\ T_2 &\equiv \leftarrow browse(mon, S, T) \end{aligned}$$

we get the following update request sets Δ_i for T_i, assuming the assigned identifier is 23:

$$\begin{aligned} \Delta_1 &= \{?date, -date(mon, 10, 0), +date(mon, 10, 23), \\ &\quad + description(23, \text{“Presentation”})\} \\ \Delta_2 &= \{?date,\ ?description\} \end{aligned}$$

Observe, that Δ_1 is read-isolated from Δ_2, as the latter does not write anything, while Δ_2 is not read-isolated from Δ_1, which actually contains update requests for the relations *date* and *description*. □

The following property results from the read-isolation. Under the given conditions, the updates of Δ_0 do not have a noticeable effect on the predicate q.

Lemma 4.7 [Independence] Let DB be an EDB instance and P_{IDB} be an IDB program. Let q be an EDB or an IDB predicate, and let $\Delta_0, \Delta \in \mathcal{D}$ be update request sets such that both $\{?r \mid r \in Def_E[P_{IDB}](q)\} \subseteq \Delta$ and Δ is read-isolated from Δ_0. Let $\Delta_C \in \mathcal{D}$ be another update request set.

Then the restriction of $WFM(P_{IDB} \cup ((DB \oplus_E \Delta_0) \oplus_E \Delta_C))$ to the predicate q equals the restriction of $WFM(P_{IDB} \cup (DB \oplus_E \Delta_C))$ to q.

Proof: See Appendix A. □

Definition 4.8 Let $I \in \mathcal{I}$ be an interpretation of update formulas and $\Delta_0 \in \mathcal{D}$ be a fixed update request set. Let φ be a ground update formula. The property (*) w.r.t. I, Δ_0, and φ is defined by the following condition:

For arbitrary $\Delta_C, \Delta \in \mathcal{D}$ such that Δ is read-isolated from Δ_0,
$(\Delta_0 \oplus \Delta_C, \Delta) \in I(\varphi) \Longleftrightarrow (\Delta_C, \Delta) \in I(\varphi)$ holds.

□

Our objective is to show that the minimal model $M_{UP}[P, DB_0]$ of a given database program P in a physical database state DB_0 satisfies (*) for arbitrary update request sets Δ_0 and arbitrary ground update formulas φ. From this assertion we can derive serializability properties and even an optimistic transaction processing method that guarantees full isolation of concurrent transactions.

Lemma 4.9 Let $I \in \mathcal{I}$ be an interpretation of update formulas and $\Delta_0 \in \mathcal{D}$ be a fixed update request set.

If (*) holds for every ground definable update atom $p(\vec{t}) \in \mathcal{B}_{DU}$, then (*) holds for every ground update formula φ.

Proof: A proof based on structural induction can be found in Appendix A. □

Lemma 4.10 The unique minimal model of an update program P_{UP} satisfies the property (*) for every fixed update request set $\Delta_0 \in \mathcal{D}$ and for every ground update formula φ.

Proof: The assertion can be proved using transfinite induction (see Appendix A for details). □

Now we are able to prove the following theorem expressing some form of transaction isolation. We consider two already computed update request sets Δ_1 and Δ_2 which have to be materialized in order to complete the corresponding transactions. Although it is not necessary for the proof of Theorem 4.11, the property we desire becomes more obvious when looking at Remark 4.12, which says that Δ_1 is computed w.r.t. the same state (and database program) as Δ_2.

Theorem 4.11 [Isolation] Let P be a database program and DB_0 be a physical EDB instance. Let U_2 be an update goal that does not contain any free variables.

Let $\Delta_1, \Delta_2 \in \mathcal{D}$ be update request sets such that:

1. Δ_2 is read-isolated from Δ_1.
2. $(\emptyset, \Delta_2) \in M_{UP}[P, DB_0](U_2)$

Then the following holds:

$$(\Delta_1, \Delta_2) \in M_{UP}[P, DB_0](U_2)$$

Proof: The assertion follows from Lemma 4.10 with $\Delta_0 := \Delta_1$ and $\varphi := U_2$. Apply (*) backwards with $\Delta_C := \emptyset$ and $\Delta := \Delta_2$. □

Remark 4.12 Let U_1 be another ground update goal such that

$$(\emptyset, \Delta_1) \in M_{UP}[P, DB_0](U_1).$$

Under this additional condition, Theorem 4.11 states that the computed possible transition Δ_2 for goal U_2 is still valid in a (hypothetical) state where the update requests of the other goal U_1 have already been executed, although the possible transitions have

been computed w.r.t. the same physical EDB instance DB_0. Thus, if first Δ_1 and then Δ_2 is materialized, the observable effect is equal to a serial, isolated execution of the transactions invoked by U_1 and U_2, respectively. This is illustrated in Figure 2. Note, that serializability is not guaranteed, if Δ_2 is materialized before Δ_1. This would require that Δ_1 is also read-isolated from Δ_2. We do not have to care about write/write conflicts, as the materialization is assumed to be performed in a strictly serial way, i.e. Δ_2 after Δ_1. □

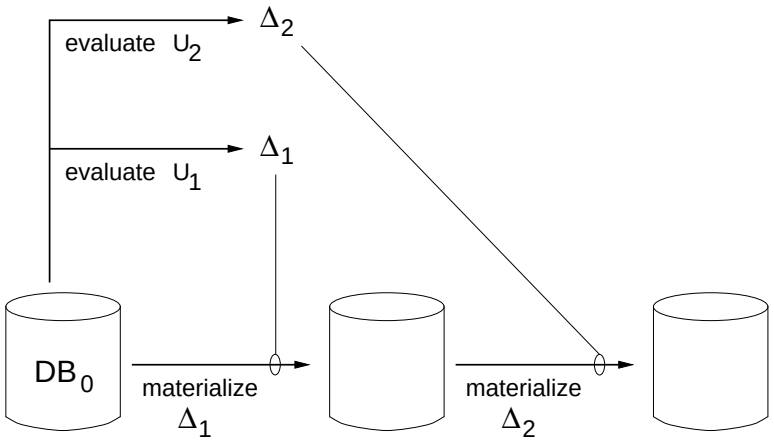

Figure 2: Serializability of two transactions

Corollary 4.13 Let P, DB_0, U_1, Δ_1, U_2, and Δ_2 be given as in Theorem 4.11 and Remark 4.12. Recall, that Δ_2 is assumed to be read-isolated from Δ_1. Then the following holds:

$$(\emptyset, \Delta_1 \oplus \Delta_2) \in M_{UP}[P, DB_0](U_1 : U_2)$$

Proof: The assertion follows directly from Theorem 4.11 using case (8) of Definition 3.12. □

Like in Remark 4.12 the update request sets Δ_1 and Δ_2 can be computed in parallel, because they both refer to the same EDB instance DB_0. Nevertheless, the sequential composition $\Delta_1 \oplus \Delta_2$ of Δ_1 and Δ_2 is a possible transition for the serial goal $U_1 : U_2$. In other words, Corollary 4.13 reformulates serializability in terms of the underlying logical concepts, i.e. the sequential conjunction.

Remark 4.14 [Preconditions for a Certifier] The reader should keep in mind the fact that we are talking about *top-level transactions* that are invoked independently from each other. This is not to be mistaken with the sequential or concurrent composition of actions within one single update query. In the latter case, the semantics has full control over the correct combination of update request sets. In the former case this is not true. In this sense, by Theorem 4.11 and Remark 4.12 we have characterized the preconditions that have to be checked by a certifier before the materialization of concurrent top-level transactions. □

Until now, we have only considered a simultaneous *evaluation* of multiple transactions, but have required that the *materialization* is performed in a strictly serial way. By the following remark, we show how to make the materialization phase more flexible.

Remark 4.15 [Materialization of Independent Transactions] Let us have a closer look at case 4 of Proposition 3.10 in Section 3.1. In the light of Theorem 4.11, it can be interpreted as offering two alternative ways to deal with the update request sets. Instead of materializing Δ_1 and Δ_2 in a serial fashion, one can compute and materialize a final update request set $\Delta := \Delta_1 \oplus \Delta_2$. If Δ_1 and Δ_2 are also write-compatible, i.e. there are no write/write conflicts between Δ_1 and Δ_2, one can even use the concurrent composition $\sqcup$. This allows us to apply the update requests of both transactions simultaneously (e.g. in an interleaved fashion). The computation of Δ can be interpreted as a merging of two local logs into one global log. The algebraic properties can be exploited for optimization purposes (e.g. sorting of update requests, writing of contiguous blocks, etc.) during the materialization phase. □

Finally, there is the question what to do, when there exist no possible transitions that are read-isolated. As serializability is not guaranteed in this case, one has to abort one of the transactions and restart (recompute) it, after the update requests of the other transaction are completely applied to the EDB. This can happen with any optimistic protocol. Note, that a transaction restarted in a new state may logically fail.

The results of this section will be illustrated by our running example.

Example 4.16 [Personal Calendar (Cont.)] Let us consider some sample transactions issued against the database instance DB_0 given in Example 2.5. The transactions could be invoked by two different users, e.g. the owner of the calendar and her secretary.

First, look at transactions T_1 and T_2 of Example 4.6. As stated there, the update request set Δ_1 is read-isolated from Δ_2, so it makes sense to evaluate both transactions in parallel w.r.t. the same database state DB_0. According to Remark 4.12, the materialization of Δ_1 after Δ_2 corresponds to the serial execution of T_1 after T_2.

If two transactions read and write the same objects, a conflict arises and must be handled, as the following example shows:

$$\begin{aligned} T_3 &\equiv \leftarrow do_insert(mon, 10, 1, \text{“Presentation”}) \\ T_4 &\equiv \leftarrow do_insert_on_day(mon, 1, \text{“Call Mr. Martin”}) \end{aligned}$$

Transaction T_3 is the same as T_1 above and thus produces the same update request set. T_4 corresponds to query Q of Example 4.2. So we get the update request set

$$\begin{aligned} \Delta_3 = \{&?date, -date(mon, 10, 0), +date(mon, 10, 23), \\ &+description(23, \text{“Presentation”})\} \end{aligned}$$

for transaction T_3 and three alternatives for T_4, namely:

$$\begin{aligned}
\Delta_4^{(1)} &= \{?date, -date(mon, 10, 0), +date(mon, 10, 28), \\
&\quad + description(28, \text{“Call Mr. Martin”})\} \\
\Delta_4^{(2)} &= \{?date, -date(mon, 11, 0), +date(mon, 11, 28), \\
&\quad + description(28, \text{“Call Mr. Martin”})\} \\
\Delta_4^{(3)} &= \{?date, -date(mon, 14, 0), +date(mon, 14, 28), \\
&\quad + description(28, \text{“Call Mr. Martin”})\}
\end{aligned}$$

None of the computed sets $\Delta_4^{(1)}$ to $\Delta_4^{(3)}$ is read-isolated from Δ_3, and Δ_3 is not read-isolated from any of the sets $\Delta_4^{(1)}$ to $\Delta_4^{(3)}$, so the two transactions cannot be serialized anymore when they both have been evaluated in database state DB_0. Therefore, the system has to choose one transaction, while restarting the other after the materialization of the first.

Observe that if the system chooses to restart T_4 while materializing T_3, in the new database state it is no longer possible to insert the entry at 10am, as this time slot now is occupied by the “Presentation”. So, for the restarted T_4, the only possible transitions will be $\Delta_4^{(2)}$ and $\Delta_4^{(3)}$.

On the other hand, if the system decides to materialize T_4 and to restart T_3, the success of T_3 depends on the non-deterministic choice in transaction T_4. If the second or the third update request set is chosen, the restarted transaction T_3 will succeed, as the time slot at 10am remains free. But if the system chooses to schedule the call for 10am, the restarted T_3 will fail, because the time slot at 10am will be occupied.

It should be kept in mind, that the conflict between T_3 and T_4 is not caused by the fact that both *write* to a common relation, but that both *read* a relation that *the other one writes*. Two write operations on the same relation can of course be serializable, as the transactions

$$\begin{aligned}
T_5 &\equiv \leftarrow do_insert_on_day(mon, 1, \text{“Call Mr. Martin”}) \\
T_6 &\equiv \leftarrow do_insert_priority(mon, 14, \text{“Dentist”})
\end{aligned}$$

show. The predicate *do_insert_priority* (see Appendix B) enters appointments without checking whether or not the requested time slot is free. When evaluated in parallel w.r.t. state DB_0, transaction T_6 creates the update request set

$$\begin{aligned}
\Delta_6 &= \{-date(mon, 14, 0), +date(mon, 14, 25), \\
&\quad + description(25, \text{“Dentist”})\}
\end{aligned}$$

assuming the assigned identifier is 25. For T_5, the same possible transitions as for T_4 are used in this example, i.e. $\Delta_5^{(1)} = \Delta_4^{(1)}$, $\Delta_5^{(2)} = \Delta_4^{(2)}$, and $\Delta_5^{(3)} = \Delta_4^{(3)}$. Although both transactions write to the same EDB relation and none of the sets $\Delta_5^{(1)}$ to $\Delta_5^{(3)}$ is read-isolated from Δ_6, Δ_6 is read-isolated from $\Delta_5^{(1)}$, $\Delta_5^{(2)}$, and $\Delta_5^{(3)}$. So, the two transactions can be evaluated in parallel in the same database state DB_0, but one of the three possible transitions for T_5 must be chosen and materialized before Δ_6 is

applied to the database.

Note, that it is no error to have both $date(mon, 14, 25)$ and $date(mon, 14, 28)$ in the database, as the former is inserted by *do_insert_priority*, which does not explicitly check the slot and thus allows overlapping entries. For the sake of the example, we do not remove the old entry, as this would require a read access, which in turn destroys the read-isolation property.

We conclude our running example with a reference to Remark 4.15. As write/write conflicts are defined on a finer granularity than read/write conflicts and in the calendar example most write operations are accompanied by read operations on the same relation, write/write conflicts between independent transactions which have already been shown to be serializable are unlikely. Indeed, in the example cases investigated above the certified possible transitions are always write-compatible with each other, i.e. there are no requests to insert and delete the same tuple. Remark 4.15 states, that also the materialization of the certified possible transitions can be performed simultaneously. In this case, the materialization does not correspond to a serial execution of the transactions, as the intermediate state between the transactions needs not to be reached during the materialization. However, the final state will be the same state as if the materialization was done in a serial fashion. □

5 Architecture

In this section we give an outline of our prototype system ULTRA which is built on top of the deductive database system *LOLA* [ZF97]. As the transaction processing is not integrated into the *LOLA* deduction engine, the *LOLA* system can be easily replaced by other systems, e.g. XSB [SSW94].

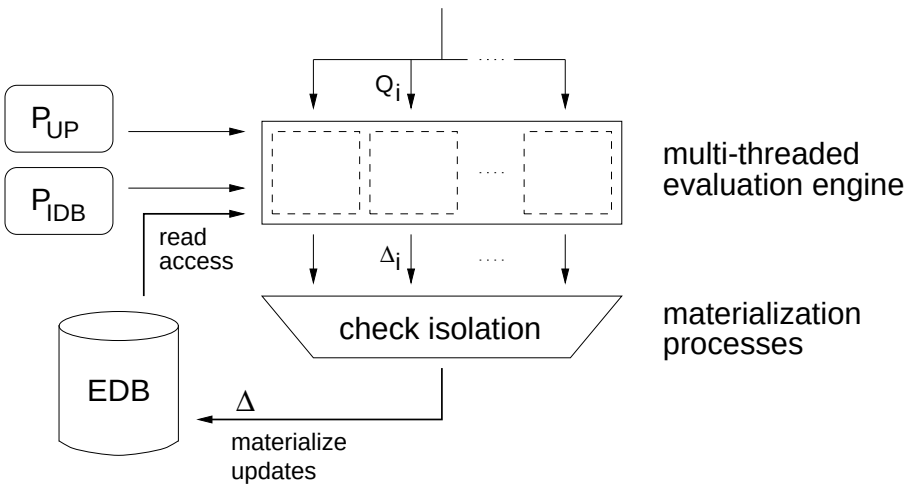

Figure 3: Architecture of the ULTRA system

The transactions invoked by top-level update queries are processed in two phases: an *evaluation phase* followed by a *materialization phase*. Figure 3 illustrates the system

architecture. Several top-level update queries Q_i are evaluated by multiple *LOLA* evaluation threads. Synchronization between different evaluation threads is not necessary, as all queries are evaluated w.r.t. the same physical database state and no updates on the EDB are actually performed. Consequently, no blocking can arise. When a transaction is ready to enter the materialization phase, i.e. when a corresponding possible transition Δ_i has been computed, the materialization processes must check if Δ_i is read-isolated from all updates that have been materialized before and that belong to transactions evaluated in the same physical database state. If the check is successful, the update requests of Δ_i are finally carried out by the external database system in an atomic write-only transaction. If the condition of read-isolation is not satisfied, the *LOLA* thread may compute an alternative possible transition or the transaction is aborted and re-executed in a new database state. Note, that update request sets can be represented by a suitable index structure to reduce the complexity of isolation checking. As already mentioned in Section 1, the evaluation engine must always access a single database state, i.e. the relevant data must not change suddenly, which would render the computed results dirty. Until each active evaluation thread has terminated, the changes of simultaneously materialized transitions must not become visible to the evaluation engine. Thus, the evaluation engine has to be synchronized with the materialization processes, or the EDB must be kept in two versions – the initial physical state and the current physical state after some materializations. It should be recalled that we assume that the extensional database system does not allow *local* transactions [BHG87], i.e. transactions not executed under control of the ULTRA system.

6 Conclusion

In this paper we have presented the syntax and the logical semantics of our rule-based update language ULTRA which can serve as a foundation of transactions in logic databases. In the second part we have investigated the problem of how to achieve isolation (in terms of serializability) of independent transactions that are evaluated in the same physical database state. We have formalized the property of read-isolation of one update request set from another and have shown that it is a sufficient condition to guarantee serializability of the corresponding transactions. It should be mentioned that checking read-isolation is the counterpart of read (lock) certification in optimistic transaction protocols [GR93], e.g. ODL [HD91]. We presented a formal criterion for checking isolation of transactions that have been evaluated simultaneously, i.e. referring to the same database state. This distinguishes our approach from that taken in Concurrent Transaction Logic [BK96], where the question is left open how isolated transactions are executed in parallel. We also have pointed out how algebraic properties can be exploited to add more flexibility to the materialization of certified transactions. In particular, the write-compatibility of independent possible transitions allows a simultaneous materialization without affecting the isolation property.

Whenever an external database system provides persistence and recoverability for the

materialized updates, it is possible to construct a logic database system that processes transactions in two independent phases – *evaluation* and *materialization* – and guarantees the ACID properties [BHG87]. On this foundation we designed the ULTRA prototype. The results of this paper have been illustrated by a non-trivial database example.

One topic for future work is the development of a more sophisticated version of the update request sets and the hypothetical states. Note, that in our semantics we currently represent a hypothetical state DB_{Curr} by an update request set Δ_C w.r.t. a fixed state DB_0. This emphasizes the hypothetical character of the intermediate states, the logging idea, and the possibility that in a forthcoming implementation the IDB might be (partially) materialized w.r.t. DB_0 but not w.r.t. the other intermediate EDB instances. A central issue, which requires a more flexible version of the update request sets, is the generalization of the ULTRA concept to arbitrary basic operations, e.g. calls to stored database procedures. To improve performance in the database context, a dedicated operational semantics which interleaves evaluation and materialization is being developed. Another research topic is to take finer-grained read information into account. A more concise isolation check will allow more freedom in the evaluation, but will also cause a considerable overhead. Note, that only approximative solutions are possible, as the problem of determining independence between IDB queries and basic updates is undecidable [Elk90].

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.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publishing Company, 1987.

[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.

[BK98] A. J. Bonner and M. Kifer. The state of change: A survey. In B. Freitag, H. Decker, M. Kifer, and A. Voronkov, editors, *Transactions and Change in Logic Databases*, volume 1472 of *LNCS*. Springer-Verlag, Berlin, Germany, 1998. to appear.

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

[Che97] W. Chen. Programming with logical queries, bulk updates, and hypothetical reasoning. *IEEE Transactions on Knowledge and Data Engineering*, 9(4):587–599, 1997.

[Elk90] C. Elkan. Independence of logic database queries and updates. In *Proc. 9th ACM Symp. on Principles of Database Systems*, pages 154–160, 1990.

[GHJ96] S. Ghandeharizadeh, R. Hull, and D. Jacobs. Heraclitus: Elevating deltas to be first-class citizens in a database programming language. *ACM Transactions on Database Systems*, 21(3):370–426, 1996.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[HD91] U. Halici and A. Dogac. An optimistic locking technique for concurrency control in distributed databases. *IEEE Transactions on Software Engineering*, 17(7):712–724, 1991.

[Lef94] A. Lefebvre. Towards an efficient evaluation of recursive aggregates in deductive databases. *New Generation Computing*, 12(2):131–160, 1994.

[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, Germany, 2nd edition, 1987.

[MBM97] D. Montesi, E. Bertino, and M. Martelli. Transactions and updates in deductive databases. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):784–797, 1997.

[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.

[SSW94] K. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 442–453, 1994.

[vG92] A. van Gelder. The well-founded semantics of aggregation. In *Proc. 11th ACM Symp. on Principles of Database Systems*, pages 127–138, 1992.

[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.

[WF97a] C. A. Wichert and B. Freitag. Capturing database dynamics by deferred updates. In *Proc. Int. Conf. on Logic Programming (ICLP '97), Leuven, Belgium*, pages 226–240, 1997.

[WF97b] C. A. Wichert and B. Freitag. Capturing database dynamics by deferred updates. Technical Report MIP-9704, Universität Passau (FMI), 1997. Available in the WWW: `http://www.uni-passau.de/~freitag/papers/`.

[WFF98] C. A. Wichert, B. Freitag, and A. Fent. Logical transactions and serializability. In B. Freitag, H. Decker, M. Kifer, and A. Voronkov, editors, *Transactions and Change in Logic Databases*, volume 1472 of *LNCS*. Springer-Verlag, Berlin, Germany, 1998. to appear.

[ZF97] U. Zukowski and B. Freitag. The deductive database system *LOLA*. In *Proc. 4th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR '97), Dagstuhl, Germany*, pages 375–386, 1997.

Appendix A

Here we will give the detailed proofs that have been omitted in Sections 3 and 4.

Proposition 3.10

1. The concurrent composition $\sqcup$ is commutative and associative.
2. The sequential composition $\oplus$ is associative.
3. Let Δ_1 and Δ_2 be not necessarily consistent update request sets. Then the following holds:

   (a) $\Delta_1 \in \mathcal{D} \ \wedge \ \Delta_2 \in \mathcal{D} \iff \Delta_1 \sqcup \Delta_2 \in \mathcal{D}$
   (b) $\Delta_1 \in \mathcal{D} \ \wedge \ \Delta_2 \in \mathcal{D} \implies \Delta_1 \oplus \Delta_2 \in \mathcal{D}$

4. Let DB be an EDB instance and $\Delta_1, \Delta_2 \in \mathcal{D}$ be consistent update request sets. Then the following holds:

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

   If Δ_1 and Δ_2 are write-compatible, then even the following equalities hold:

   (b) $(DB \oplus_E \Delta_1) \oplus_E \Delta_2 = DB \oplus_E (\Delta_1 \sqcup \Delta_2) = (DB \oplus_E \Delta_2) \oplus_E \Delta_1$

Proof: The proof essentially relies on the definitions of $\sqcup$, $\oplus$ and $\oplus_E$.

1. Recall, that the concurrent composition $\sqcup$ is defined as the set union. Assertion 1 follows from the properties of the set union.

2. Next, we show the associativity of $\oplus$ (assertion 2).

Let arbitrary update request sets Δ_1, Δ_2 and Δ_3 be given.

The following equivalences hold for arbitrary insertion requests $+r(\vec{t})$:

$$
\begin{array}{ll}
 & +r(\vec{t}) \in (\Delta_1 \oplus \Delta_2) \oplus \Delta_3 \\
\iff & (+r(\vec{t}) \in \Delta_1 \oplus \Delta_2 \ \wedge \ -r(\vec{t}) \notin \Delta_3) \ \vee \ +r(\vec{t}) \in \Delta_3 \\
\iff & [\ (\ (+r(\vec{t}) \in \Delta_1 \wedge -r(\vec{t}) \notin \Delta_2) \vee +r(\vec{t}) \in \Delta_2) \ \wedge \\
 & \ -r(\vec{t}) \notin \Delta_3\ ] \ \vee \ +r(\vec{t}) \in \Delta_3 \\
\iff & (+r(\vec{t}) \in \Delta_1 \wedge -r(\vec{t}) \notin \Delta_2 \wedge -r(\vec{t}) \notin \Delta_3) \vee \\
 & (+r(\vec{t}) \in \Delta_2 \wedge -r(\vec{t}) \notin \Delta_3) \ \vee \ +r(\vec{t}) \in \Delta_3 \\
\iff & (+r(\vec{t}) \in \Delta_1 \wedge -r(\vec{t}) \notin \Delta_2 \wedge -r(\vec{t}) \notin \Delta_3) \vee \\
(subsmpt.) & (+r(\vec{t}) \in \Delta_1 \wedge +r(\vec{t}) \in \Delta_3 \wedge -r(\vec{t}) \notin \Delta_3) \vee \\
 & (+r(\vec{t}) \in \Delta_2 \wedge -r(\vec{t}) \notin \Delta_3) \ \vee \ +r(\vec{t}) \in \Delta_3 \\
\iff & (+r(\vec{t}) \in \Delta_1 \wedge (-r(\vec{t}) \notin \Delta_2 \vee +r(\vec{t}) \in \Delta_3) \wedge -r(\vec{t}) \notin \Delta_3) \vee \\
 & (+r(\vec{t}) \in \Delta_2 \wedge -r(\vec{t}) \notin \Delta_3) \ \vee \ +r(\vec{t}) \in \Delta_3 \\
\iff & (+r(\vec{t}) \in \Delta_1 \wedge \neg\ [\ (-r(\vec{t}) \in \Delta_2 \wedge +r(\vec{t}) \notin \Delta_3) \vee -r(\vec{t}) \in \Delta_3\ ]\ ) \vee \\
 & (+r(\vec{t}) \in \Delta_2 \wedge -r(\vec{t}) \notin \Delta_3) \ \vee \ +r(\vec{t}) \in \Delta_3 \\
\iff & (+r(\vec{t}) \in \Delta_1 \ \wedge \ -r(\vec{t}) \notin \Delta_2 \oplus \Delta_3) \ \vee \ +r(\vec{t}) \in \Delta_2 \oplus \Delta_3 \\
\iff & +r(\vec{t}) \in \Delta_1 \oplus (\Delta_2 \oplus \Delta_3)
\end{array}
$$

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 $-r(\vec{t})$.

The sequential composition $\oplus$ merges the read tags exactly like the usual set union. Thus the equivalences

$$?r \in (\Delta_1 \oplus \Delta_2) \oplus \Delta_3 \iff ?r \in \Delta_1 \cup \Delta_2 \cup \Delta_3 \iff ?r \in \Delta_1 \oplus (\Delta_2 \oplus \Delta_3)$$

hold for arbitrary read tags $?r$. This completes the proof of assertion 2.

3. Next, we show the consistency properties of assertion 3.

(a) The first implication holds, since $\sqcup$ is defined as the set union: if Δ_1 or Δ_2 were not consistent, then $\Delta_1 \sqcup \Delta_2$ would not be consistent, too.

(b) Now, 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:

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

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.

4. Finally, we show assertion 4.

(a) The assertion concerning the exchange of execution $\oplus_E$ and sequential composition $\oplus$ can be proved in a similar fashion as the associativity of $\oplus$ (see assertion 2). Note, that $DB \oplus_E (\Delta_1 \oplus \Delta_2)$ is well-defined, since $\Delta_1 \oplus \Delta_2$ is consistent (see assertion 3).

(b) For the other assertion, we only have to show that the concurrent composition $\sqcup$ is a special case of the sequential composition $\oplus$ for write-compatible update request sets. Then the desired equalities can be easily proved using assertion 1.

Let $\Delta_1, \Delta_2 \in \mathcal{D}$ be consistent update request sets which are write-compatible, such that the concurrent composition $\Delta_1 \sqcup \Delta_2$ is consistent. We show the equality of $\Delta_1 \oplus \Delta_2$ and $\Delta_1 \sqcup \Delta_2$.

The following equivalences hold for arbitrary insertion requests $+r(\vec{t})$:

$$
\begin{array}{ll}
 & +r(\vec{t}) \in \Delta_1 \oplus \Delta_2 \\
\Longleftrightarrow & (+r(\vec{t}) \in \Delta_1 \wedge -r(\vec{t}) \notin \Delta_2) \ \vee \ +r(\vec{t}) \in \Delta_2 \\
\Longleftrightarrow & +r(\vec{t}) \in \Delta_1 \ \vee \ +r(\vec{t}) \in \Delta_2 \\
\scriptstyle (subsmpt.) & \\
\Longleftrightarrow & +r(\vec{t}) \in \Delta_1 \sqcup \Delta_2
\end{array}
$$

Have a closer look at the equivalence marked "subsmpt.". '$\Rightarrow$' is trivial, as the condition on the left hand side is stronger than the condition on the right. However, due to the assumption of write-compatibility, the converse '$\Leftarrow$' also holds, as the deletion request $-r(\vec{t})$ cannot be contained in Δ_2, if $+r(\vec{t}) \in \Delta_1 \sqcup \Delta_2$ holds.

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

In both compositions, the read tags are merged like in the usual set union. Thus

$$?r \in \Delta_1 \oplus \Delta_2 \iff ?r \in \Delta_1 \cup \Delta_2 \iff ?r \in \Delta_1 \sqcup \Delta_2$$

holds for arbitrary read tags $?r$. This completes the proof of the equality $\Delta_1 \oplus \Delta_2$ and $\Delta_1 \sqcup \Delta_2$.

Now, let an arbitrary EDB instance DB be given. We can reason as follows:

$$\begin{array}{rlcl} & (DB \oplus_E \Delta_1) \oplus_E \Delta_2 & = & DB \oplus_E (\Delta_1 \oplus \Delta_2) \\ = & DB \oplus_E (\Delta_1 \sqcup \Delta_2) & = & DB \oplus_E (\Delta_2 \sqcup \Delta_1) \\ = & DB \oplus_E (\Delta_2 \oplus \Delta_1) & = & (DB \oplus_E \Delta_2) \oplus_E \Delta_1 \end{array}$$

□

Proposition 3.13 The following properties hold for the connectives "," and ":".

1. The concurrent conjunction "," is commutative.
2. The concurrent conjunction "," is associative.
3. The sequential conjunction ":" is associative.

Proof: The assertions are proved using Definition 3.12 and Proposition 3.10. For two given update formulas, we show the equality of their interpretation by comparing the elements $(\Delta_C, \Delta) \in \mathcal{D} \times \mathcal{D}$.

1. First, we show the commutativity of ",":

$$\begin{array}{rl} & (\Delta_C, \Delta) \in I(\varphi, \psi) \\ \iff & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\ & (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_C, \Delta_2) \in I(\psi) \text{ and } \Delta = \Delta_1 \sqcup \Delta_2 \\ \iff & \text{there exist } \Delta_2, \Delta_1 \in \mathcal{D} \text{ such that :} \\ \overset{(3.10)}{} & (\Delta_C, \Delta_2) \in I(\psi) \text{ and } (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } \Delta = \Delta_2 \sqcup \Delta_1 \\ \iff & (\Delta_C, \Delta) \in I(\psi, \varphi) \end{array}$$

2. Next, we show the associativity of ",":

$$
\begin{aligned}
& (\Delta_C, \Delta) \in I(\ [\varphi, \psi], \chi\ ) \\
\Longleftrightarrow\ & \text{there exist } \Delta_{1-2}, \Delta_3 \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_{1-2}) \in I(\varphi, \psi) \text{ and } (\Delta_C, \Delta_3) \in I(\chi) \text{ and } \Delta = \Delta_{1-2} \sqcup \Delta_3 \\
\Longleftrightarrow\ & \text{there exist } \Delta_{1-2}, \Delta_1, \Delta_2, \Delta_3 \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_C, \Delta_2) \in I(\psi) \text{ and } (\Delta_C, \Delta_3) \in I(\chi) \\
& \text{and } \Delta_{1-2} = \Delta_1 \sqcup \Delta_2 \text{ and } \Delta = \Delta_{1-2} \sqcup \Delta_3 \\
\Longleftrightarrow\ & \text{there exist } \Delta_1, \Delta_{2-3}, \Delta_2, \Delta_3 \in \mathcal{D} \text{ such that :} \\
\overset{(3.10)}{\Longleftrightarrow}\ & (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_C, \Delta_2) \in I(\psi) \text{ and } (\Delta_C, \Delta_3) \in I(\chi) \\
& \text{and } \Delta_{2-3} = \Delta_2 \sqcup \Delta_3 \text{ and } \Delta = \Delta_1 \sqcup \Delta_{2-3} \\
\Longleftrightarrow\ & \text{there exist } \Delta_1, \Delta_{2-3} \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_C, \Delta_{2-3}) \in I(\psi, \chi) \text{ and } \Delta = \Delta_1 \sqcup \Delta_{2-3} \\
\Longleftrightarrow\ & (\Delta_C, \Delta) \in I(\ \varphi, [\psi, \chi]\ )
\end{aligned}
$$

3. Finally we show the associativity of ":":

$$
\begin{aligned}
& (\Delta_C, \Delta) \in I(\ [\varphi : \psi] : \chi\ ) \\
\Longleftrightarrow\ & \text{there exist } \Delta_{1-2}, \Delta_3 \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_{1-2}) \in I(\varphi : \psi) \text{ and } (\Delta_C \oplus \Delta_{1-2}, \Delta_3) \in I(\chi) \text{ and } \Delta = \Delta_{1-2} \oplus \Delta_3 \\
\Longleftrightarrow\ & \text{there exist } \Delta_{1-2}, \Delta_1, \Delta_2, \Delta_3 \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_C \oplus \Delta_1, \Delta_2) \in I(\psi) \text{ and } (\Delta_C \oplus \Delta_{1-2}, \Delta_3) \in I(\chi) \\
& \text{and } \Delta_{1-2} = \Delta_1 \oplus \Delta_2 \text{ and } \Delta = \Delta_{1-2} \oplus \Delta_3 \\
\Longleftrightarrow\ & \text{there exist } \Delta_1, \Delta_{2-3}, \Delta_2, \Delta_3 \in \mathcal{D} \text{ such that :} \\
\overset{(3.10)}{\Longleftrightarrow}\ & (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_C \oplus \Delta_1, \Delta_2) \in I(\psi) \\
& \text{and } ((\Delta_C \oplus \Delta_1) \oplus \Delta_2, \Delta_3) \in I(\chi) \\
& \text{and } \Delta_{2-3} = \Delta_2 \oplus \Delta_3 \text{ and } \Delta = \Delta_1 \oplus \Delta_{2-3} \\
\Longleftrightarrow\ & \text{there exist } \Delta_1, \Delta_{2-3} \in \mathcal{D} \text{ such that :} \\
& (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_C \oplus \Delta_1, \Delta_{2-3}) \in I(\psi : \chi) \text{ and } \Delta = \Delta_1 \oplus \Delta_{2-3} \\
\Longleftrightarrow\ & (\Delta_C, \Delta) \in I(\ \varphi : [\psi : \chi]\ )
\end{aligned}
$$

□

Lemma 3.21 Let $\mathcal{J} \subseteq \mathcal{I}$ be a non-empty set of interpretations. Define a new interpretation I by

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

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

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

holds for every ground update goal U.

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

Base cases:

Essentially, we apply Definition 3.12.

1. DB literal

   We show the assertion only for a positive DB literal. The proof for a negative DB literal is entirely analogous.

   $$\begin{array}{ll} & (\Delta_C, \Delta) \in I(q(\vec{t})) \\ \Longrightarrow & q(\vec{t}) \in I_{DB}(\Delta_C) \\ & \text{and } \Delta = \{?r \mid r \in Def_E[P_{IDB}](q)\} \\ \Longrightarrow & (\Delta_C, \Delta) \in I'(q(\vec{t})) \end{array}$$

2. NOP

   $$\begin{array}{ll} & (\Delta_C, \Delta) \in I(NOP) \\ \Longrightarrow & \Delta = \emptyset \\ \Longrightarrow & (\Delta_C, \Delta) \in I'(NOP) \end{array}$$

3. Basic update atom

   $$\begin{array}{ll} & (\Delta_C, \Delta) \in I(INS\ r(\vec{t})) \\ \Longrightarrow & \Delta = \{+r(\vec{t})\} \\ \Longrightarrow & (\Delta_C, \Delta) \in I'(INS\ r(\vec{t})) \end{array}$$

   Similar implications hold for a basic update atom $DEL\ r(\vec{t})$.

4. Definable update atom

   $$\begin{array}{ll} & (\Delta_C, \Delta) \in I(p(\vec{t})) \\ \Longrightarrow & (\Delta_C, \Delta) \in I'(p(\vec{t})) \end{array}$$

Induction step:

By the induction hypothesis, $I(\varphi) \subseteq I'(\varphi)$ holds for any direct proper subformula φ of the composite formulas analyzed in the following. Using Definition 3.12 we can reason as follows.

1. Concurrent conjunction

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

2. Sequential conjunction

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

3. Bulk quantification

   Consider a bulk quantification $\#\,\vec{X}\,[A \mapsto \varphi]$, where q is the EDB/IDB predicate of A. By the induction hypothesis,

   $$I(\varphi[\,\vec{X}\,/\,\vec{t}\,]) \subseteq I'(\varphi[\,\vec{X}\,/\,\vec{t}\,])$$

   holds for all instances $\varphi[\,\vec{X}\,/\,\vec{t}\,]$ of the update subgoal.

   Let $\Delta_C, \Delta \in \mathcal{D}$ be arbitrarily chosen. The set

   $$T_{A,\Delta_C} = \{(\,\vec{t}\,) \mid A[\,\vec{X}\,/\,\vec{t}\,] \in I_{DB}(\Delta_C)\}$$

   does not depend on the interpretation of update formulas.

   The case $T_{A,\Delta_C} = \emptyset$ is trivial:

$$
\begin{array}{ll}
 & (\Delta_C, \Delta) \in I(\#\,\vec{X}\,[A \mapsto \varphi]) \\
\Longrightarrow & \Delta = \{?r \mid r \in Def_E[P_{IDB}](q)\} \\
\Longrightarrow & (\Delta_C, \Delta) \in I'(\#\,\vec{X}\,[A \mapsto \varphi])
\end{array}
$$

For $T_{A,\Delta_C} \neq \emptyset$ we get:

$$
\begin{array}{ll}
 & (\Delta_C, \Delta) \in I(\#\,\vec{X}\,[A \mapsto \varphi]) \\
\Longrightarrow & \text{there exists a function } f : T_{A,\Delta_C} \to \mathcal{D} \text{ such that :} \\
 & \forall(\,\vec{t}\,) \in T_{A,\Delta_C} :\ (\Delta_C, f(\,\vec{t}\,)) \in I(\varphi[\,\vec{X}\,/\,\vec{t}\,]) \\
 & \text{and } \Delta \ = \ \{?r \mid r \in Def_E[P_{IDB}](q)\} \ \sqcup \ \bigsqcup_{(\vec{t}) \in T_{A,\Delta_C}} f(\,\vec{t}\,) \\
\Longrightarrow & \text{there exists a function } f : T_{A,\Delta_C} \to \mathcal{D} \text{ such that :} \\
^{(IH)} & \forall(\,\vec{t}\,) \in T_{A,\Delta_C} :\ (\Delta_C, f(\,\vec{t}\,)) \in I'(\varphi[\,\vec{X}\,/\,\vec{t}\,]) \\
 & \text{and } \Delta \ = \ \{?r \mid r \in Def_E[P_{IDB}](q)\} \ \sqcup \ \bigsqcup_{(\vec{t}) \in T_{A,\Delta_C}} f(\,\vec{t}\,) \\
\Longrightarrow & (\Delta_C, \Delta) \in I'(\#\,\vec{X}\,[A \mapsto \varphi])
\end{array}
$$

□

Lemma 3.22 Let P_{UP} be an update program, let $\mathcal{M} \subseteq \mathcal{I}$ 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 $p(\,\vec{t}\,) \in \mathcal{B}_{DU}$. 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 3.12, cases (5) and (6), this is true for a rule r, iff for every ground instance $U \to p(\,\vec{t}\,)$ of r the set inclusion $I(U) \subseteq I(p(\,\vec{t}\,))$ holds.

Now let $U \to p(\,\vec{t}\,)$ 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.

Applying Lemma 3.21 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 3.23 Let P_{UP} be an update program. Then P_{UP} has a unique minimal model M_{UP}.

Proof: P_{UP} has at least one model due to Remark 3.19. 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 3.22 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 $M' \in \mathcal{M}$. The minimality and uniqueness of M_{UP} follow directly. □

Lemma 4.7 Let DB be an EDB instance and P_{IDB} be an IDB program. Let q be an EDB or an IDB predicate, and let $\Delta_0, \Delta \in \mathcal{D}$ be update request sets such that both $\{?r \mid r \in Def_E[P_{IDB}](q)\} \subseteq \Delta$ and Δ is read-isolated from Δ_0. Let $\Delta_C \in \mathcal{D}$ be another update request set.

Then the restriction of $WFM(P_{IDB} \cup ((DB \oplus_E \Delta_0) \oplus_E \Delta_C))$ to the predicate q equals the restriction of $WFM(P_{IDB} \cup (DB \oplus_E \Delta_C))$ to q.

Proof: Δ is read-isolated from Δ_0 by assumption and contains read tags for all EDB predicates q depends on. Thus, Δ_0 cannot contain any update request to change an EDB relation that is relevant to the interpretation of any EDB/IDB atom $q(\vec{t})$, i.e. $DB \oplus_E \Delta_0$ restricted to predicates in $Def_E[P_{IDB}](q)$ equals DB restricted to $Def_E[P_{IDB}](q)$. Hence, $(DB \oplus_E \Delta_0) \oplus_E \Delta_C$ restricted to predicates in $Def_E[P_{IDB}](q)$ equals $DB \oplus_E \Delta_C$ restricted to $Def_E[P_{IDB}](q)$.

Consequently, the restriction of $WFM(P_{IDB} \cup ((DB \oplus_E \Delta_0) \oplus_E \Delta_C))$ to the predicate q equals the restriction of $WFM(P_{IDB} \cup (DB \oplus_E \Delta_C))$ to q.

Note, that is sufficient to consider the syntactical dependencies, because the well-founded model has a fixpoint characterization which is essentially built on consequence operators derived from the given facts and rules. □

Definition 4.8 Let $I \in \mathcal{I}$ be an interpretation of update formulas and $\Delta_0 \in \mathcal{D}$ be a fixed update request set. Let φ be a ground update formula. The property (*) w.r.t. I, Δ_0, and φ is defined by the following condition:

> For arbitrary $\Delta_C, \Delta \in \mathcal{D}$ such that Δ is read-isolated from Δ_0,
> $(\Delta_0 \oplus \Delta_C, \Delta) \in I(\varphi) \Longleftrightarrow (\Delta_C, \Delta) \in I(\varphi)$ holds.

□

Lemma 4.9 Let $I \in \mathcal{I}$ be an interpretation of update formulas and $\Delta_0 \in \mathcal{D}$ be a fixed update request set.

If (*) holds for every ground definable update atom $p(\vec{t}) \in \mathcal{B}_{DU}$, then (*) holds for every ground update formula φ.

Proof: We prove the assertion by structural induction. Let (*) hold for all ground definable update atoms $p(\vec{t}) \in \mathcal{B}_{DU}$. In each case shown below we choose arbitrary $\Delta_C, \Delta \in \mathcal{D}$ such that Δ is read-isolated from Δ_0 and show the equivalence in Definition 4.8 using Definition 3.12.

Base cases:

1. DB literal

   We show the assertion only for a positive DB literal. The proof for a negative DB literal is entirely analogous.

Let A be a DB atom $q(\vec{t})$. Define $\Delta_A := \{?r \mid r \in Def_E[P_{IDB}](q)\}$. We reason as follows:

$$
\begin{aligned}
& (\Delta_0 \oplus \Delta_C, \Delta) \in I(A) \\
\iff\ & A \in I_{DB}(\Delta_0 \oplus \Delta_C) \text{ and } \Delta = \Delta_A \\
\iff\ & A \in WFM(P_{IDB} \cup (DB_0 \oplus_E (\Delta_0 \oplus \Delta_C))) \text{ and } \Delta = \Delta_A \\
& \quad (3.11) \\
\iff\ & A \in WFM(P_{IDB} \cup ((DB_0 \oplus_E \Delta_0) \oplus_E \Delta_C)) \text{ and } \Delta = \Delta_A \\
& \quad (3.10) \\
\iff\ & A \in WFM(P_{IDB} \cup (DB_0 \oplus_E \Delta_C)) \text{ and } \Delta = \Delta_A \\
& \quad (4.7) \\
\iff\ & A \in I_{DB}(\Delta_C) \text{ and } \Delta = \Delta_A \\
& \quad (3.11) \\
\iff\ & (\Delta_C, \Delta) \in I(A)
\end{aligned}
$$

Note, that Lemma 4.7 is essential for this part of the proof.

2. NOP

$$
\begin{aligned}
& (\Delta_0 \oplus \Delta_C, \Delta) \in I(NOP) \\
\iff\ & \Delta = \emptyset \\
\iff\ & (\Delta_C, \Delta) \in I(NOP)
\end{aligned}
$$

3. Basic update atom

$$
\begin{aligned}
& (\Delta_0 \oplus \Delta_C, \Delta) \in I(INS\ r(\vec{t})) \\
\iff\ & \Delta = \{+r(\vec{t})\} \\
\iff\ & (\Delta_C, \Delta) \in I(INS\ r(\vec{t}))
\end{aligned}
$$

Similar equivalences hold for a basic update atom $DEL\ r(\vec{t})$.

4. Definable update atom

The equivalence holds by the precondition.

Induction step:

By the induction hypothesis, (*) holds for any direct proper subformula φ of the composite formulas analyzed in the following.

1. Implication

$$
\begin{array}{ll}
 & (\Delta_0 \oplus \Delta_C, \Delta) \in I(\varphi \to \psi) \\
\iff & (\Delta_0 \oplus \Delta_C, \Delta) \in I(\varphi) \Longrightarrow (\Delta_0 \oplus \Delta_C, \Delta) \in I(\psi) \\
\iff & (\Delta_0 \oplus \Delta_C, \Delta) \notin I(\varphi) \text{ or } (\Delta_0 \oplus \Delta_C, \Delta) \in I(\psi) \\
\iff & (\Delta_C, \Delta) \notin I(\varphi) \text{ or } (\Delta_C, \Delta) \in I(\psi) \\
(IH) & \\
\iff & (\Delta_C, \Delta) \in I(\varphi) \Longrightarrow (\Delta_C, \Delta) \in I(\psi) \\
\iff & (\Delta_C, \Delta) \in I(\varphi \to \psi)
\end{array}
$$

2. Universal quantification

$$
\begin{array}{ll}
 & (\Delta_0 \oplus \Delta_C, \Delta) \in I(\forall \vec{X}\, \varphi) \\
\iff & \text{for all ground term tuples } \vec{t}, \\
 & (\Delta_0 \oplus \Delta_C, \Delta) \in I(\varphi[\, \vec{X} \,/\, \vec{t}\,]) \text{ holds} \\
\iff & \text{for all ground term tuples } \vec{t}, \\
(IH) & (\Delta_C, \Delta) \in I(\varphi[\, \vec{X} \,/\, \vec{t}\,]) \text{ holds} \\
\iff & (\Delta_C, \Delta) \in I(\forall \vec{X}\, \varphi)
\end{array}
$$

3. Concurrent conjunction

$$
\begin{array}{ll}
 & (\Delta_0 \oplus \Delta_C, \Delta) \in I(\varphi, \psi) \\
\iff & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
 & (\Delta_0 \oplus \Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_0 \oplus \Delta_C, \Delta_2) \in I(\psi) \\
 & \text{and } \Delta = \Delta_1 \sqcup \Delta_2 \\
\iff & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
(IH) & (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_C, \Delta_2) \in I(\psi) \\
 & \text{and } \Delta = \Delta_1 \sqcup \Delta_2 \\
\iff & (\Delta_C, \Delta) \in I(\varphi, \psi)
\end{array}
$$

   Both Δ_1 and Δ_2 are read-isolated from Δ_0, as Δ is read-isolated from Δ_0 and contains exactly the read tags of Δ_1 and Δ_2. Therefore, the induction hypothesis is applicable.

4. Sequential conjunction

$$
\begin{array}{ll}
 & (\Delta_0 \oplus \Delta_C, \Delta) \in I(\varphi : \psi) \\
\Longleftrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
 & (\Delta_0 \oplus \Delta_C, \Delta_1) \in I(\varphi) \text{ and } ((\Delta_0 \oplus \Delta_C) \oplus \Delta_1, \Delta_2) \in I(\psi) \\
 & \text{and } \Delta = \Delta_1 \oplus \Delta_2 \\
\Longleftrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
\scriptstyle (3.10) & (\Delta_0 \oplus \Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_0 \oplus (\Delta_C \oplus \Delta_1), \Delta_2) \in I(\psi) \\
 & \text{and } \Delta = \Delta_1 \oplus \Delta_2 \\
\Longleftrightarrow & \text{there exist } \Delta_1, \Delta_2 \in \mathcal{D} \text{ such that :} \\
\scriptstyle (IH) & (\Delta_C, \Delta_1) \in I(\varphi) \text{ and } (\Delta_C \oplus \Delta_1, \Delta_2) \in I(\psi) \\
 & \text{and } \Delta = \Delta_1 \oplus \Delta_2 \\
\Longleftrightarrow & (\Delta_C, \Delta) \in I(\varphi : \psi)
\end{array}
$$

   The induction hypothesis is applicable, because like in the previous case both Δ_1 and Δ_2 are read-isolated from Δ_0.

5. Bulk quantification

   Consider a bulk quantification $\#\vec{X}\,[A \mapsto \varphi]$, where q is the EDB/IDB predicate of A. By case (9) of Definition 3.12

$$\{?r \mid r \in Def_E[P_{IDB}](q)\} \subseteq \Delta$$

   must hold. As every ground instance A' of A is built upon the predicate q and Δ is read-isolated from Δ_0, we can reason as follows (note, that Lemma 4.7 is essential again):

$$
\begin{array}{ll}
 & A' \in I_{DB}(\Delta_0 \oplus \Delta_C) \\
\Longleftrightarrow & A' \in WFM(P_{IDB} \cup (DB_0 \oplus_E (\Delta_0 \oplus \Delta_C))) \\
\scriptstyle (3.11) & \\
\Longleftrightarrow & A' \in WFM(P_{IDB} \cup ((DB_0 \oplus_E \Delta_0) \oplus_E \Delta_C)) \\
\scriptstyle (3.10) & \\
\Longleftrightarrow & A' \in WFM(P_{IDB} \cup (DB_0 \oplus_E \Delta_C)) \\
\scriptstyle (4.7) & \\
\Longleftrightarrow & A' \in I_{DB}(\Delta_C) \\
\scriptstyle (3.11) &
\end{array}
$$

   Consequently, the equality

$$T_{A,\Delta_0 \oplus \Delta_C} = T_{A,\Delta_C}$$

   holds for the sets $T_{A,\Delta_0 \oplus \Delta_C}$ and T_{A,Δ_C} as defined in Definition 3.12.

The case $T_{A,\Delta_0 \oplus \Delta_C} = T_{A,\Delta_C} = \emptyset$ is trivial:

$$\begin{array}{ll} & (\Delta_0 \oplus \Delta_C, \Delta) \in I(\#\vec{X}\,[A \mapsto \varphi]) \\ \Longleftrightarrow & \Delta = \{?r \mid r \in Def_E[P_{IDB}](q)\} \\ \Longleftrightarrow & (\Delta_C, \Delta) \in I(\#\vec{X}\,[A \mapsto \varphi]) \end{array}$$

For the complementary case we get:

$$\begin{array}{ll} & (\Delta_0 \oplus \Delta_C, \Delta) \in I(\#\vec{X}\,[A \mapsto \varphi]) \\ \Longleftrightarrow & \text{there exists a function } f : T_{A,\Delta_0 \oplus \Delta_C} \to \mathcal{D} \text{ such that :} \\ & \forall(\vec{t}) \in T_{A,\Delta_0 \oplus \Delta_C} : \ (\Delta_0 \oplus \Delta_C, f(\vec{t})) \in I(\varphi[\,\vec{X}\,/\,\vec{t}\,]) \\ & \text{and } \Delta \ = \ \{?r \mid r \in Def_E[P_{IDB}](q)\} \ \sqcup \ \bigsqcup_{(\vec{t}) \in T_{A,\Delta_0 \oplus \Delta_C}} f(\vec{t}) \\ \Longleftrightarrow & \text{there exists a function } f : T_{A,\Delta_C} \to \mathcal{D} \text{ such that :} \\ & \forall(\vec{t}) \in T_{A,\Delta_C} : \ (\Delta_0 \oplus \Delta_C, f(\vec{t})) \in I(\varphi[\,\vec{X}\,/\,\vec{t}\,]) \\ & \text{and } \Delta \ = \ \{?r \mid r \in Def_E[P_{IDB}](q)\} \ \sqcup \ \bigsqcup_{(\vec{t}) \in T_{A,\Delta_C}} f(\vec{t}) \\ \Longleftrightarrow & \text{there exists a function } f : T_{A,\Delta_C} \to \mathcal{D} \text{ such that :} \\ {\scriptstyle (IH)} & \forall(\vec{t}) \in T_{A,\Delta_C} : \ (\Delta_C, f(\vec{t})) \in I(\varphi[\,\vec{X}\,/\,\vec{t}\,]) \\ & \text{and } \Delta \ = \ \{?r \mid r \in Def_E[P_{IDB}](q)\} \ \sqcup \ \bigsqcup_{(\vec{t}) \in T_{A,\Delta_C}} f(\vec{t}) \\ \Longleftrightarrow & (\Delta_C, \Delta) \in I(\#\vec{X}\,[A \mapsto \varphi]) \end{array}$$

The induction hypothesis is applicable, because $f(\vec{t}) \subseteq \Delta$ holds for all relevant ground tuples $(\vec{t})$, which implies the read-isolation of the $f(\vec{t})$ from Δ_0.

□

Definition 3.28 The *ordinal powers* of the immediate consequence operator of an update program P_{UP} are defined as follows.

$$\begin{array}{lcll} T_{P_{UP}} \uparrow 0 & := & I_\perp & \\ T_{P_{UP}} \uparrow \alpha + 1 & := & T_{P_{UP}}(T_{P_{UP}} \uparrow \alpha) & \text{for a successor ordinal } \alpha + 1 \\ T_{P_{UP}} \uparrow \beta & := & lub\ \{\ T_{P_{UP}} \uparrow \gamma \mid \gamma < \beta\ \} & \text{for a limit ordinal } \beta \end{array}$$

□

Lemma 4.10 The unique minimal model of an update program P_{UP} satisfies the property (*) for every fixed update request set $\Delta_0 \in \mathcal{D}$ and for every ground update formula φ.

Proof: It is sufficient to prove the assertion for definable update atoms. The assertion for arbitrary update formulas follows by Lemma 4.9.

After choosing an arbitrary update request set $\Delta_0 \in \mathcal{D}$, we show by transfinite induction that (*) holds w.r.t. Δ_0 and $T_{P_{UP}} \uparrow \gamma$ for arbitrary ordinals γ. That means, we

verify the following condition for all ground definable update atoms $p(\vec{t}) \in \mathcal{B}_{DU}$:

> For arbitrary $\Delta_C, \Delta \in \mathcal{D}$ such that Δ is read-isolated from Δ_0,
> $(\Delta_0 \oplus \Delta_C, \Delta) \in T_{P_{UP}} \uparrow \gamma\,(p(\vec{t})) \Longleftrightarrow (\Delta_C, \Delta) \in T_{P_{UP}} \uparrow \gamma\,(p(\vec{t}))$ holds.

Base case: $\gamma = 0$

Let $p(\vec{t}) \in \mathcal{B}_{DU}$ be a definable update atom. Because $I_\perp(p(\vec{t})) = \emptyset$, the property (*) trivially holds w.r.t. $I_\perp$ and $p(\vec{t})$.

Induction step: successor ordinal $\alpha + 1$

By the induction hypothesis, (*) holds w.r.t. $T_{P_{UP}} \uparrow \alpha$ for all ground definable update atoms $p(\vec{t}) \in \mathcal{B}_{DU}$. By Lemma 4.9, (*) even holds for all ground update formulas φ.

Now choose a definable update atom $p(\vec{t}) \in \mathcal{B}_{DU}$. For arbitrary $\Delta_C, \Delta \in \mathcal{D}$ such that Δ is read-isolated from Δ_0, the following equivalences hold.

$$
\begin{array}{ll}
 & (\Delta_0 \oplus \Delta_C, \Delta) \in T_{P_{UP}} \uparrow \alpha + 1\,(p(\vec{t})) \\
\Longleftrightarrow & (\Delta_0 \oplus \Delta_C, \Delta) \in T_{P_{UP}}(T_{P_{UP}} \uparrow \alpha)(p(\vec{t})) \\
\scriptstyle{(3.28)} & \\
\Longleftrightarrow & \text{there exists a ground instance } U \to p(\vec{t}) \text{ of a rule } r \in P_{UP}, \\
\scriptstyle{(3.25)} & \text{such that } (\Delta_0 \oplus \Delta_C, \Delta) \in T_{P_{UP}} \uparrow \alpha\,(U) \\
\Longleftrightarrow & \text{there exists a ground instance } U \to p(\vec{t}) \text{ of a rule } r \in P_{UP}, \\
\scriptstyle{(IH)} & \text{such that } (\Delta_C, \Delta) \in T_{P_{UP}} \uparrow \alpha\,(U) \\
\Longleftrightarrow & (\Delta_C, \Delta) \in T_{P_{UP}}(T_{P_{UP}} \uparrow \alpha)(p(\vec{t})) \\
\scriptstyle{(3.25)} & \\
\Longleftrightarrow & (\Delta_C, \Delta) \in T_{P_{UP}} \uparrow \alpha + 1\,(p(\vec{t})) \\
\scriptstyle{(3.28)} &
\end{array}
$$

Induction step: limit ordinal β

Using the properties of the power-set lattice, one can show that

$$lub(\mathcal{J})(p(\vec{t})) = \bigcup_{I \in \mathcal{J}} I(p(\vec{t}))$$

holds for all $p(\vec{t}) \in \mathcal{B}_{DU}$ and arbitrary non-empty subsets $\mathcal{J}$ of the lattice $\mathcal{I}$ of interpretations (see Lemma 3.18).

By the induction hypothesis, (*) holds w.r.t. each $T_{P_{UP}} \uparrow \alpha \quad (\alpha < \beta)$ for all ground definable update atoms $p(\vec{t}) \in \mathcal{B}_{DU}$.

Now choose a definable update atom $p(\vec{t}) \in \mathcal{B}_{DU}$. For arbitrary $\Delta_C, \Delta \in \mathcal{D}$ such that Δ is read-isolated from Δ_0, the following equivalences hold.

$$
\begin{array}{ll}
 & (\Delta_0 \oplus \Delta_C, \Delta) \in T_{P_{UP}} \uparrow \beta\,(p(\vec{t})) \\
\iff & (\Delta_0 \oplus \Delta_C, \Delta) \in \bigcup_{\alpha<\beta} T_{P_{UP}} \uparrow \alpha\,(p(\vec{t})) \\
\scriptstyle (see\ above) & \\
\iff & \text{there exists } \alpha < \beta \text{ such that } (\Delta_0 \oplus \Delta_C, \Delta) \in T_{P_{UP}} \uparrow \alpha\,(p(\vec{t})) \\
\iff & \text{there exists } \alpha < \beta \text{ such that } (\Delta_C, \Delta) \in T_{P_{UP}} \uparrow \alpha\,(p(\vec{t})) \\
\scriptstyle (IH) & \\
\iff & (\Delta_C, \Delta) \in \bigcup_{\alpha<\beta} T_{P_{UP}} \uparrow \alpha\,(p(\vec{t})) \\
\iff & (\Delta_C, \Delta) \in T_{P_{UP}} \uparrow \beta\,(p(\vec{t})) \\
\scriptstyle (see\ above) &
\end{array}
$$

In particular, the assertion holds w.r.t. $T_{P_{UP}} \uparrow \alpha_{P_{UP}}$, which is equal to the minimal model of P_{UP} due to Theorem 3.29. □

Appendix B

Here we show the complete running example.

EDB Schema and Instance (DB_0):

$date(Day, Slot, ID)$ associates a time slot $Slot$ on day Day with an appointment identifier ID.

$description(ID, Text)$ stores the descriptive text $Text$ for appointment ID.

$date$	Day	$Slot$	ID
	mon	9	21
	mon	10	0
	mon	11	0
	mon	12	7
	mon	13	7
	mon	14	0
	mon	15	8
	mon	16	10

$description$	ID	$Text$
	7	Meeting Mr. Dean
	8	Hairdresser
	10	Review
	21	Call Mr. Miller

IDB Program (P_{IDB}):

$free(Day, Slot, L)$ determines whether on day Day there are L free time slots starting at time slot $Slot$.

$$\begin{array}{lll} free(D,S,1) & \leftarrow & date(D,S,0) \\ free(D,S,L) & \leftarrow & L > 1,\ free(D,S,1), \\ & & S1 = S+1,\ L1 = L-1, \\ & & free(D,S1,L1) \end{array}$$

$duration_of(ID, L)$ calculates the duration L of appointment ID by counting the number of time slots that are occupied by this appointment. (The rule contains an aggregation construct [Lef94], which is provided by $LOLA$ [ZF97]. It states that the number L of slots $date(D, S, ID)$ is counted separately for each identifier ID. Of course, a more complex implementation which is based on recursion and negation could be used instead of the quite natural aggregation.)

$$duration_of(ID,L) \quad \leftarrow \quad \mathsf{group_by}\,(\,date(D,S,ID),\ [ID],\ \mathsf{count}(L)\,)$$

$browse(Day, Slot, Text)$ is the general retrieval predicate that joins the relations $date$ and $description$.

$$browse(D,S,T) \quad \leftarrow \quad date(D,S,ID),\ description(ID,T)$$

Update Program (P_{UP}):

$do_allocate(Day, Slot, L, ID)$ recursively tries to allocate L time slots on day Day, starting at time slot $Slot$, for appointment ID. The predicate fails, if one of the time slots is not free.

$$\begin{array}{lll} do_allocate(D,S,1,ID) & \leftarrow & date(D,S,0), \\ & & DEL\ date(D,S,0),\ INS\ date(D,S,ID) \\ do_allocate(D,S,L,ID) & \leftarrow & L > 1,\ do_allocate(D,S,1,ID), \\ & & S1 = S+1,\ L1 = L-1, \\ & & do_allocate(D,S1,L1,ID) \end{array}$$

$do_deallocate(ID)$ frees all time slots that are allocated for appointment ID. Here we use the bulk quantifier instead of recursion.

$$\begin{array}{lll} do_deallocate(ID) & \leftarrow & \#\ D,S \\ & & [\ date(D,S,ID)\ \mapsto \\ & & \quad [\ DEL\ date(D,S,ID),\ INS\ date(D,S,0)\ ]\ ] \end{array}$$

$do_insert(Day, Slot, L, Text)$ inserts an appointment with descriptive text $Text$ and duration L on day Day, starting at time slot $Slot$. The insertion fails, if one of the time slots is already allocated for another entry. ($\mathsf{newid}(ID)$ is a built-in predicate that generates a new, unique identifier ID every time it is called. Such an operation is provided in many database systems to avoid concurrency problems when searching for unique keys.)

$$do_insert(D, S, L, T) \leftarrow \mathsf{newid}(ID),\ do_allocate(D, S, L, ID),\ INS\ description(ID, T)$$

$do_insert_priority(Day, Slot, Text)$ inserts a high-priority appointment $Text$ on day Day at time slot $Slot$ without checking whether the time slot is free or already occupied.

$$do_insert_priority(D, S, T) \leftarrow \mathsf{newid}(ID),\ DEL\ date(D, S, 0),\ INS\ date(D, S, ID),\ INS\ description(ID, T)$$

$do_insert_on_day(Day, L, Text)$ looks for time slots on day Day that can hold an appointment of duration L and inserts the entry with description $Text$ at one of these times.

$$do_insert_on_day(D, L, T) \leftarrow free(D, S, L)\ :\ do_insert(D, S, L, T)$$

$do_delete(Day, Slot)$ deletes the appointment on day Day at time slot $Slot$ by freeing all allocated time slots and removing the description.

$$do_delete(D, S) \leftarrow date(D, S, ID),\ do_deallocate(ID),\ description(ID, T),\ DEL\ description(ID, T)$$

$do_move(ID, Day, Slot)$ moves the appointment with identifier ID to the new day Day at time slot $Slot$. The predicate fails, if the time slots at the destination are not free.

$$do_move(ID, D, S) \leftarrow [\ duration_of(ID, L),\ do_deallocate(ID)\ ]\ :\ do_allocate(D, S, L, ID)$$