ELSEVIER

Theoretical Computer Science 254 (2001) 591–661

Theoretical Computer Science

www.elsevier.com/locate/tcs

Regular database update logics

Paul Spruit [1], Roel Wieringa*,[2], John-Jules Meyer [3]

Faculty of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081a, 1081 HV, Amsterdam, Netherlands

Received 18 April 1998; revised 13 March 2000; accepted 27 June 2000
Communicated by M. Nivat

Abstract

We study regular first-order update logic (FUL), which is a variant of regular dynamic logic in which updates to function symbols as well as to predicate symbols are possible. We first study FUL without making assumptions about atomic updates. Second, we look at relational algebra update logic (RAUL), which can be viewed as an extension of relational algebra with assignment. RAUL is an instantiation of FUL. Third, we study dynamic database logic (DDL), which is another version of FUL, in which the atomic updates can be "bulk updates" of predicates and updates of updateable functions. In all three cases, we define syntax, declarative semantics, axiomatizations, and operational semantics of the logic. All axiom systems are shown to be sound. Assuming the domain closure and unique naming assumptions, we also give a proof sketch of completeness of the axiomatization of DDL. The operational semantics presented in the paper are shown to be equivalent to the declarative semantics for certain classes of databases. We give examples of correctness proofs in RAUL and in DDL. Finally, we compare our approach to that of others and show how DDL can be used as a logic in which to specify and reason about updates to an object-oriented database system. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Dynamic logic; Relational algebra; Database updates

1. Introduction

In this paper, we define a first-order logic for regular programs (FUL), that is used to define two languages for database updates, relational algebra update logic (RAUL)

* Correspondence address: Department of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands.

E-mail addresses: roelw@cs.utwente.nl (R. Wieringa), jj@cs.ruu.nl (J. Meyer).

[1] Now at DSE, Heemskerkweg 70, 1944 GW Beverwijk, Netherlands.

[2] http://www.cs.utwente.nl/ ∼roelw.

[3] Now at the Department of Computer Science, Utrecht University, Padualaan 14, De Uithof, P.O. Box 80089, 3508 TB Utrecht, Netherlands.

0304-3975/01/$ - see front matter © 2001 Elsevier Science B.V. All rights reserved.
PII: S0304-3975(00)00289-9

and dynamic database logic (DDL). FUL and its instantiations RAUL and DDL can be used to *specify* the requirements for a database update program declaratively and to specify an implementation of such a program, as well as to *reason* about the correctness of this implementation with respect to the requirements. We do this in a way that clearly separates actions from states in the semantics, and update programs from queries in the languages. FUL is similar to classical dynamic logic, but allows updates of function and predicate symbols rather than of variables. No commitment is made in FUL as to what the atomic updates of the language are. RAUL choses as atomic updates assignments of the form $p := e$, where p is a predicate and e is a relational algebra expression. DDL choses as atomic updates actions *bulk insertions*, *updates or deletions* or *function updates*. For example, a bulk insertion has the form $\&X\,\mathit{I}pT$ **where** ϕ, where p is a predicate with argument tuple T, X is a finite set of variable declarations and ϕ a formula. A function update has the form $fT := t$, where f is a function symbol applied to argument tuple T and t is a term. The idea of parameterizing an update logic by its atomic actions appeared earlier in the literature in transaction logic [7, 5, 6, 8]. It is also present in classical dynamic logic, where the axioms for programs are separate from the axioms of atomic actions.

1.1. Motivation and overview

To set the stage, a general picture of (first-order) modal logic as a logic for database updates is given in Fig. 1. The left-hand side of this picture concerns the syntax, the right-hand side concerns the declarative semantics. The *logic database schema* defines the predicate and function symbols used (the signature of the logic) and contains the constraints. A *possible world* interprets the predicate and function symbols in such a way that the constraints are satisfied. Some of the function and predicate symbols have a fixed interpretation over all possible worlds (like addition and less than for natural numbers); these symbols are used for the abstract data types. Because we allow the definition of abstract data types, possible worlds may contain infinite sets. In addition, in the case of disjunctive information database, one database state would be represented by a possibly infinite set of possible worlds. As we want a database state to be finite and still allow disjunctive information, we define a *database state* to be a closed first-order formula (or equivalently, a finite set of closed first-order formulas). The declarative semantics of an *update* is given as a relation on the possible worlds.

The atomic updates on the predicate symbols (relations of the relational database model) are *bulk updates* in which a set of tuples can be inserted, deleted or updated in a relation. Insertions, updates and deletions of single tuples are special cases of this. As atomic updates on function symbols we have *assignment* with which an updateable function can be given a new value for an argument. These atomic updates may be combined into update *programs* with the standard regular operators choice, sequential composition and iteration.

In Section 2, we look at the syntax, declarative semantics and axiomatization of regular first-order update logic (FUL), without yet giving a syntax or semantics for

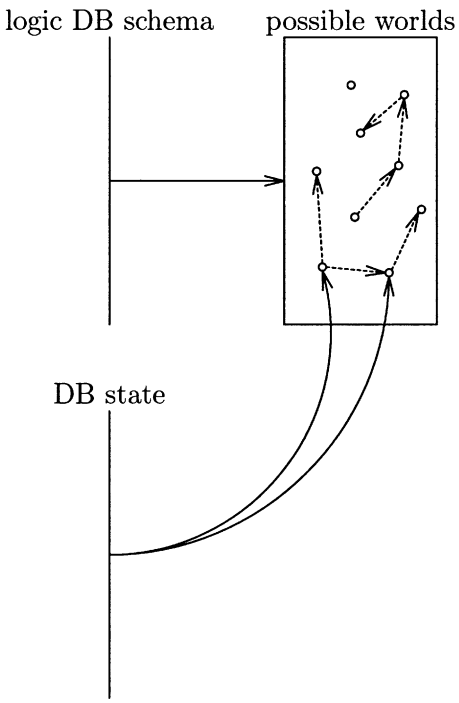

Fig. 1. Syntax and semantics of modal logic databases.

the atomic updates. This logic differs from standard dynamic logic because we take predicate and function symbols rather than variables to be updateable. In Section 3 we introduce RAUL as an instantiation of FUL with relational algebra assignments as atomic updates. Because the choice of atomic updates is orthogonal to the choice of process combinators, we ignore the *regular* update programs of FUL in this section. We define syntax, axioms, declarative and operational semantics, and give an example of a correctness proof in FUL. In Section 4 we then define dynamic database logic (DDL) as regular first-order update logic with two kinds of atomic updates, bulk updates to predicates, and assignment as an update operator for functions. We illustrate the use of DDL with an example correctness proof of an update program for a logic database.

Sections 3 and 4 are largely independent from each other, since they describe two different ways of instantiating FUL with classes of atomic updates. When we defined an operational semantics of DDL updates, we discovered that we needed constructive expressions by which to update the extension of predicates. For this purpose, we developed RAUL and inserted Section 3 on RAUL before the DDL Section 4. However, since the definition of the declarative and operational semantics of RAUL is rather intricate, the reader may want to skip those definitions on first reading. The only place where Section 4 depends upon Section 3 is in the definition of an operational semantics for DDL.

Section 5 contains a discussion of the way in which DDL can be used for the specification of updates to an object-oriented database. It also compares our approach with other work, including transaction logic [7], in which the idea of factoring out atomic updates first appeared. Section 6 concludes the paper and discusses some topics for further research. The paper is based on earlier research into regular update languages [34, 33].

The languages defined in this paper are all based upon order-sorted first-order logic with equality. This is a language design choice that could have been made differently. For example, sorts can be replaced by unary predicates and the partial ordering on sorts can be replaced by implications between sort predicates. Furthermore, functions can be replaced by binary predicates with axioms that enforce a functional relationship. Conversely, predicates can be replaced by Boolean functions. We offer the following as motivation for the choices made in this paper. Inclusion of sort names in the languages allows a natural blend with algebraic specification of abstract data types, which we use in our specifications. It also permits simpler specifications, because it allows concise specification of a sort ordering and of function declarations. The price to pay for this is a slightly more complex semantics and proof system for the languages. The desire to assume algebraic specification of abstract data types also motivates the inclusion of function symbols in the languages. Again, this makes specifications simpler, because we do not have to include axioms that enforce functional relationships. In addition, a language that includes function symbols (common in programming languages) as well as predicate symbols (common in database languages) reduces the "impedance mismatch" between programming languages and database languages.

2. Regular first-order update logic (FUL)

In this section, we define regular first-order dynamic logic, which is a version of dynamic logic that will be used to specify database updates. The logic is parametrized by a set of atomic updates. DDL is an instantiation of regular first-order update logic with a particular choice for atomic updates.

2.1. Syntax

In regular first-order update logic, we distinguish *updateable* from *non-updateable* predicate and function symbols. As illustrated in Section 3, updateable predicate symbols can be used to represent relations in a relational database, and as illustrated in Section 5, updateable function symbols can be used to represent attributes in an object-oriented database. Non-updateable predicate and function symbols are used to represent relations and functions in abstract data types (for instance the less than relation and addition function on natural numbers). The distinction between updateable and non-updateable function and predicate symbols is a refinement of standard modal logic, where all predicate symbols (except equality) are updateable and all function symbols are non-updateable [22]. We now make this precise.

Definition 2.1. Let *Sorts* be a (finite) set of sort symbols. A *function declaration* over *Sorts* has the form $f : \langle s_1, \ldots, s_n \rangle \to s$, with $s_1, \ldots, s_n, s \in Sorts$. The symbol f is the *name*, $s_1, \ldots, s_n$ are the *argument sorts* and s is the *result sort* of the function declaration. Of course, we allow the case $n = 0$ (so we have constants).

Definition 2.2. Let *Sorts* be a (finite) set of sort symbols. A *predicate declaration* over *Sorts* has the form p: $\langle s_1, \ldots, s_n \rangle$, with $s_1, \ldots, s_n \in Sorts$. The symbol p is the *name* and $s_1, \ldots, s_n$ are the *argument sorts* of the predicate declaration.

Definition 2.3. A (first-order) *update logic signature* Σ is a five-tuple $(Sorts, UFun, NUFun, UPred, NUPred)$ with *Sorts* a finite set of sort symbols, *UFun* and *NUFun* finite, disjoint sets of function declarations over *Sorts* and *UPred* and *NUPred* finite, disjoint sets of predicate declarations over *Sorts*.

The intention is that *UFun* and *UPred* are declarations of updateable function and predicate symbols and *NUFun* and *NUPred* are declarations of non-updateable function and predicate symbols. For convenience, we make the assumption that no two (predicate or function) symbols may have the same name; this is the *non-overloading* restriction. Under this restriction, we can without confusion write $p \in UPred$ instead of p: $\langle s_1, \ldots, s_n \rangle \in UPred$, etc. The non-overloading restriction is just made to ease various definitions, but is not essential.

Definition 2.4. Let $\Sigma = (Sorts, UFun, NUFun, UPred, NUPred)$ be an update logic signature (satisfying the non-overloading restriction). Then $Fun_\Sigma = UFun \cup NUFun$ and $Pred_\Sigma = UPred \cup NUPred$. When no confusion can arise, we drop the subscript Σ from Fun_Σ and $Pred_\Sigma$.

From now on, we just assume we have a single given update logic signature. We also assume we have a set of typed variables *Var* and we assume that we always have enough variables of every sort. (We usually do not state explicitly what sort a variable has, but this should not lead to confusion.) Next, we define the *terms*, the *update programs* and the *formulas*.

Definition 2.5. The set of *update programs* is defined by induction:
1. all atomic updates are update programs;
2. $\phi?$ is an update program, for any formula ϕ;
3. $\alpha;\beta$, $\alpha + \beta$ and α^* are update programs, for any update programs α and β;
4. the only update programs are those given by 1–3.

The atomic updates mentioned in Definition 2.5 are defined in Section 4 as bulk updates and assignment.

Definition 2.6. The set of terms of every sort $s \in Sorts$ is defined inductively as the minimal set that satisfies

1. Every variable of sort s is a term of sort s.
2. For every function declaration $f : \langle s_1, \ldots, s_n \rangle \to s \in Fun$ and terms $t_1, \ldots, t_n$ of sorts $s_1, \ldots, s_n$ respectively, $f(t_1, \ldots, t_n)$ is a term of sort s. If $n = 0$, then f is called a *constant* and we write f instead of $f()$.

Closed terms are terms that do not contain variables. *Non-updateable* terms are terms that do not contain updateable function symbols. *Immutable* terms are closed, non-updateable terms. We write $s \in t$ if the symbol (variable, function symbol or predicate symbol) s occurs in the term t and we write $s \in T$ if the symbol s occurs in one of the terms in the tuple of terms T. (Note that for a predicate symbol p, we always have $p \notin t$.)

Note that an immutable term (non-updateable, closed term) is not necessarily the same things as a constant (0-ary function symbol)!

Definition 2.7. The set of (*update logic*) *formulas* is defined inductively as

1. pT is a formula, for any p: $\langle s_1, \ldots, s_n \rangle \in Pred$ and any tuple $T = (t_1, \ldots, t_n)$ of terms $t_1, \ldots, t_n$ of sorts $s_1, \ldots, s_n$ respectively;
2. $t = t'$ is a formula, for any terms t and t' of the same sort;
3. $\phi \vee \psi$ is a formula, for any formulas ϕ and ψ;
4. $\neg\phi$ is a formula, for any formula ϕ;
5. $\exists x\phi$ is a formula, for any variable x and formula ϕ;
6. $[\alpha]\phi$ is a formula, for any update program α, and any formula ϕ;
7. the only formulas are those, given by 1–6.

First-order formulas are formulas that are built inductively only from clauses 1–5. We write $s \in \phi$ if the (updateable or non-updateable predicate or function) symbol s occurs in the formula ϕ.

All clauses in the definition of formulas are the same as in first-order modal logic [22], except for clause 6. Instead of a single modal operator "$\Box$", we have a modality for every update program α. The formula $[\alpha]\phi$ should be read in a dynamic logic way; it states that after every possible execution of α we are in a state where ϕ holds. The other formulas are just the well-known propositional connectives and first-order logic quantification. We assume that the other common propositional connectives and universal quantification are defined in terms of $\vee$ and $\neg$ and that modal possibility $\langle\alpha\rangle$ is defined as $\neg[\alpha]\neg$.

For later reference, we need the familiar definitions of a variable being free for a term and of substitution.

Definition 2.8. An occurrence of a variable x in a formula ϕ is called *bound* iff there is a subformula of ϕ of the form $\exists x\psi$ and the occurrence of x is within this subformula. If an occurrence of a variable in a formula is not bound, then it is called *free*. If there

is a free occurrence of a variable x in a formula ϕ then x is called a free variable in ϕ. The set of all free variables in ϕ is denoted $FV(\phi)$. We also use this notation for (tuples of) terms: $FV(t)$ ($FV(T)$) is the set of all variables occurring in the term t (the tuple of terms T).

Since we view $\forall$ as an abbreviation for $\neg\exists\neg$, the above definition also handles the $\forall$ quantifier.

Definition 2.9. Let t be a term, let x be variable and let ϕ be a formula. Then x is *free for t in ϕ* iff for all variables $y \neq x$ that occur in t, there is no subformula $\exists y\psi$ of ϕ such that x is free in ψ.

If x is free for t in ϕ, then substituting t for all free occurrences of x in ϕ does not introduce (unwanted) bindings on variables occurring in t.

Definition 2.10. The *substitution* of a term t (of sort s) for a variable x (of the same sort s) in a formula ϕ, denoted $\phi[t/x]$, is the formula ϕ with all free occurrences of x in ϕ replaced by t. The substitution of a term t for a variable x in a term t' is defined similarly (in a term, all occurrences of variables are free).

2.2. Declarative semantics

The semantics of formulas and update programs is given in Kripke structures. We have an algebra that gives the interpretation of the non-updateable symbols, and we have possible worlds that give the interpretation of the updateable symbols. A structure is then a set of possible worlds; updates can be evaluated in structures as relations on the possible worlds.

Definition 2.11. Let $\Sigma=(Sorts, Fun, Pred)$ be a many sorted first-order signature. (This means that *Sorts* is a set of sort symbols, *Fun* is a set of function declarations over *Sorts* and *Pred* is a set of predicate declarations over *Sorts*.) A *Σ-algebra* (first-order interpretation) $\mathcal{A}$ assigns to each sort symbol $s \in Sorts$ a set $s^{\mathcal{A}}$, to every function declaration $f : \langle s_1, \ldots, s_n\rangle \rightarrow s \in Fun$ a function $f^{\mathcal{A}}$ with domain $s_1^{\mathcal{A}} \times \cdots \times s_n^{\mathcal{A}}$ and range $s^{\mathcal{A}}$, and to every predicate declaration p: $\langle s_1, \ldots, s_n\rangle \in Pred$ a subset $p^{\mathcal{A}}$ of $s_1^{\mathcal{A}} \times \cdots \times s_n^{\mathcal{A}}$.

From now on, we write just "algebra", instead of $(Sorts, NUFun, NUPred)$-algebra. As the non-updateable symbols are used for the specification of abstract data types, they usually have an intended interpretation (for instance with respect to an initial algebra of an equational specification). For the moment, we ignore the intended semantics of the non-updateable part of a specification.

We define two special structures that play an important role for update logics. In the relational database model, a table (the extension of a predicate symbol) is always finite. This motivates the definition of the *relational* structure. Relatively easy to handle

from a theoretical viewpoint is the structure in which all possible worlds are present: the *full* structure.

Definition 2.12. Let $\mathscr{A}$ be an algebra.

- A *possible world* on $\mathscr{A}$ is a function w which assigns to every function declaration $f:\langle s_1,\ldots,s_n\rangle \to s \in UFun$ a function $w(f)$ with domain $s_1^{\mathscr{A}}\times\cdots\times s_n^{\mathscr{A}}$ and range $s^{\mathscr{A}}$ and to every predicate declaration p: $\langle s_1,\ldots,s_n\rangle \in UPred$ a subset $w(p)$ of $s_1^{\mathscr{A}}\times\cdots\times s_n^{\mathscr{A}}$.
- A *structure* on $\mathscr{A}$ is a set of possible worlds on $\mathscr{A}$.
- The *full structure* on $\mathscr{A}$ (written as $\mathscr{F}_{\mathscr{A}}$) consists of all possible worlds on $\mathscr{A}$.
- A *relational world* on $\mathscr{A}$ is a possible world on $\mathscr{A}$ that assigns a finite number of tuples to every predicate symbol declaration.
- The *relational structure* on $\mathscr{A}$ (written as $\mathscr{R}_{\mathscr{A}}$) consists of all relational worlds on $\mathscr{A}$.

When we consider relational structures, we usually assume that $UFun=\emptyset$.

We have chosen to identify a possible world with its valuation (the way the world interprets the function and predicate symbols), so different possible worlds have different valuations (cannot have the same valuations). By contrast, in standard modal logic, different possible worlds *can* have the same valuation. In general, such worlds can be distinguished in modal logic with the modal operator. In database update logic, as a result of the semantics of the update actions, worlds with the same valuations cannot be distinguished with the update actions (see also [35]). Therefore, our choice to identify a possible world with its valuation is no real restriction on the semantics.

Definition 2.13. A *variable interpretation* on an algebra $\mathscr{A}$ is a function I_V that assigns to each variable x of sort $s\in Sorts$ an element $I_V(x)$ of $s^{\mathscr{A}}$. We sometimes use I_V as a function on tuples of variables; this function is just the pointwise extension of I_V.

We use variants and restrictions of variable interpretations, which are defined in the standard way.

Definition 2.14. Let $f:A\to B$ be a function, let a be an element of A and let b be an element of B. Then the *variant* of f that maps a to b is the function $f\{a\mapsto b\}:A\to B$ which is defined for all $a'\in A$ as

$$f\{a\mapsto b\}(a')=\begin{cases} b & \text{if } a'=a,\\ f(a') & \text{if } a'\neq a.\end{cases}$$

Instead of $f\{a_1\mapsto b_1\}\cdots\{a_n\mapsto b_n\}$, we usually write $f\{a_1\mapsto b_1,\ldots,a_n\mapsto b_n\}$.

Note that the order of the variants in $f\{a_1 \mapsto b_1, \ldots, a_n \mapsto b_n\}$ is in general important, as we do not exclude $a_i = a_j$ for $i \neq j$.

Definition 2.15. Let $f : A \to B$ be a function and let $A' \subseteq A$. Then the *restriction* of f to A' is a function $f \restriction_{A'} : A' \to B$ which is defined for all $a \in A'$ as: $f \restriction_{A'}(a) = f(a)$.

In the above two definitions, A may also be a Cartesian product of some sets, so the definition handles the general case of n-ary functions.

A term is interpreted as a domain element of a domain of an algebra.

Definition 2.16. Let $\mathscr{A}$ be an algebra, let w be a possible world on $\mathscr{A}$ and let I_V be a variable interpretation on $\mathscr{A}$. The interpretation $t^{\mathscr{A},w,I_V}$ of a term t in possible world w over $\mathscr{A}$ under variable interpretation I_V is defined inductively as:

- $x^{\mathscr{A},w,I_V} = I_V(x)$
- $f(t_1, \ldots, t_n)^{\mathscr{A},w,I_V} = \begin{cases} f^{\mathscr{A}}(t_1^{\mathscr{A},w,I_V}, \ldots, t_n^{\mathscr{A},w,I_V}) & \text{if } f \in NUFun, \\ w(f)(t_1^{\mathscr{A},w,I_V}, \ldots, t_n^{\mathscr{A},w,I_V}) & \text{if } f \in UFun. \end{cases}$

We define $T^{\mathscr{A},w,I_V}$ for T a tuple of terms as the pointwise extension of the interpretation of single terms.

The interpretation of a closed term t does not depend on the variable interpretation. Without confusion, we can therefore write $t^{\mathscr{A},w}$ for closed terms t. Similarly, we can write $t^{\mathscr{A},I_V}$ for non-updateable terms and $t^{\mathscr{A}}$ for immutable terms. The next lemma states formally that the interpretation of terms is independent from the interpretation of symbols not occurring in them.

Lemma 2.17. *Let $\mathscr{A}$ be an algebra, let w and w' be possible worlds on $\mathscr{A}$, let I_V and I'_V be variable interpretations on $\mathscr{A}$ and let t be a term such that for all $f \in UFun$ with $f \in t$: $w'(f) = w(f)$ and for all variables x with $x \in t$: $I'_V(x) = I_V(x)$. Then $t^{\mathscr{A},w,I_V} = t^{\mathscr{A},w',I'_V}$.*

Proof. The straightforward induction proof on the structure of t is omitted. □

For the interpretation of formulas, we need an interpretation of update programs in a structure S. In general, when an update language is chosen, its semantics will be defined as a function m_{S,I_V} which, given an update program α, gives the interpretation of α as a relation $m_{S,I_V}(\alpha)$ on the possible worlds of S. In this section, we assume that m_{S,I_V} is given for atomic updates. The next definition shows how this semantics can be extended to update programs. In next Section 4, we then define m_{S,I_V} for atomic updates.

Definition 2.18. Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$ and let I_V be a variable interpretation on $\mathscr{A}$. The extension of m_{S,I_V} from atomic updates to update programs, is defined by structural induction (simultaneously with the semantics of formulas given

in Definition 2.20) as

$$
\begin{aligned}
&m_{S,I_V}(\phi?) = \{(w,w) \in S^2 \mid S,w,I_V \models \phi\},\\
&m_{S,I_V}(\alpha;\beta) = \{(w,w') \in S^2 \mid \exists v \in S\colon\ (w,v) \in m_{S,I_V}(\alpha) \text{ and}\\
&\qquad\qquad (v,w') \in m_{S,I_V}(\beta)\},\\
&m_{S,I_V}(\alpha+\beta) = m_{S,I_V}(\alpha) \cup m_{S,I_V}(\beta),\\
&m_{S,I_V}(\alpha^*) = \{(w,w') \in S^2 \mid \exists n \geqslant 1, w_1,\ldots,w_n \in S\colon\ w_1 = w \text{ and}\\
&\qquad\qquad w_n = w' \text{ and } \forall i\ (1 \leqslant i < n)\colon\ (w_i,w_{i+1}) \in m_{S,I_V}(\alpha)\}.
\end{aligned}
$$

The successor worlds function which is defined next, is often more convenient than the function m_{S,I_V}.

Definition 2.19. Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a world in S, let I_V be a variable interpretation and let α be an update. Then a world w' in S is an α-*successor world* of w (with respect to I_V) iff $(w,w') \in m_{S,I_V}(\alpha)$. The set of all α-successor worlds of w in structure S (with respect to I_V) is written $succ_{S,I_V}(\alpha)(w)$ (so $succ_{S,I_V}(\alpha)(w) = \{w' \in S \mid (w,w') \in m_{S,I_V}(\alpha)\}$).

Definition 2.20. Let S be a structure on the algebra $\mathscr{A}$, let w be a world in S and let I_V be a variable interpretation. We write $S,w,I_V \models \phi$ for a formula ϕ if ϕ is true in world w of structure S under variable interpretation I_V. The relation $\models$ is defined by structural induction (simultaneously with the semantics of update programs given in Definition 2.18) as

$$
\begin{aligned}
S,w,I_V \models pT &\quad\Leftrightarrow T^{\mathscr{A},w,I_V} \in p^{\mathscr{A}},\ \text{for}\ p \in NUPred\\
S,w,I_V \models pT &\quad\Leftrightarrow T^{\mathscr{A},w,I_V} \in w(p),\ \text{for}\ p \in UPred\\
S,w,I_V \models t = t' &\quad\Leftrightarrow t^{\mathscr{A},w,I_V} = t'^{\mathscr{A},w,I_V}\\
S,w,I_V \models \phi_1 \vee \phi_2 &\quad\Leftrightarrow S,w,I_V \models \phi_1 \text{ or } S,w,I_V \models \phi_2\\
S,w,I_V \models \neg\phi &\quad\Leftrightarrow S,w,I_V \not\models \phi\\
S,w,I_V \models \exists x\phi &\quad\Leftrightarrow \exists d \in s^{\mathscr{A}}\colon\ S,w,I_V\{x \mapsto d\} \models \phi,\\
&\qquad\qquad \text{for } x \text{ a variable of sort } s\\
S,w,I_V \models [\alpha]\phi &\quad\Leftrightarrow \forall w' \in S \text{ with } (w,w') \in m_{S,I_V}(\alpha)\colon\ S,w',I_V \models \phi
\end{aligned}
$$

We write $S,w \models \phi$ if for all I_V, $S,w,I_V \models \phi$, we write $S \models \phi$ if for all $w \in S$: $S,w \models \phi$ and we write $\models \phi$ (ϕ is *valid*) if for all structures S: $S \models \phi$. For a set of formulas Φ, we write $S,w,I_V \models \Phi$ iff for all $\phi \in \Phi$: $S,w,I_V \models \phi$. All other cases ($S,w \models \Phi$, etc.) are derived from this definition, similar as done for single formulas. A formula ϕ is called *satisfiable* if there is a structure S, world w in S and variable interpretation I_V such that $S,w,I_V \models \phi$.

First-order formulas do not contain modalities, so they can already be evaluated in a possible world under some variable interpretation, without needing a structure (as there is no "reference" to other possible worlds in first-order formulas). For a first-order

formula ϕ, we therefore write $w, I_V \models \phi$ and this relation $\models$ is defined by the first six clauses in Definition 2.20. By a completely straightforward induction proof on the structure of ϕ, we can then prove for all structures S on $\mathscr{A}$ which contain w that $S, w, I_V \models \phi \Leftrightarrow w, I_V \models \phi$; this proof is omitted.

For the modal "diamond", the semantics can be derived from the semantics of the "box" and the definition of the diamond in terms of the box. This yields

$$S, w, I_V \models \langle\alpha\rangle\phi \Leftrightarrow \exists w' \in S\colon\ (w, w') \in m_{S,I_V}(\alpha) \text{ and } S, w', I_V \models \phi.$$

By a straightforward induction on the structure of ϕ it can be proven that changing the interpretation of free variables does not alter the truth value of a formula. More precisely, we have the following lemma.

Lemma 2.21. *$\mathscr{A}$ is an algebra, S be a structure on $\mathscr{A}$, w be a world in S, I_V and I'_V are variable interpretations on $\mathscr{A}$ then for any formula ϕ and for all variables $x \in FV(\phi)$ with $I'_V(x) = I_V(x)$ we have*

$$S, w, I_V \models \phi \Leftrightarrow S, w, I'_V \models \phi.$$

Proof. The straightforward induction proof on the structure of ϕ is omitted. □

We list a number of standard properties which the semantics of first-order update logic shares with the semantics of first-order modal logic. The straightforward proofs of these properties are omitted.

Proposition 2.22. *Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a world in S and let I_V be a variable interpretation on $\mathscr{A}$. Then*

$$\begin{aligned}
&S, w, I_V \models \mathit{true} \\
&S, w, I_V \not\models \mathit{false} \\
&S, w, I_V \models t \neq t' && \Leftrightarrow t^{\mathscr{A},w,I_V} \neq t'^{\mathscr{A},w,I_V} \\
&S, w, I_V \models T = T' && \Leftrightarrow T^{\mathscr{A},w,I_V} = T'^{\mathscr{A},w,I_V} \\
&S, w, I_V \models T \neq T' && \Leftrightarrow T^{\mathscr{A},w,I_V} \neq T'^{\mathscr{A},w,I_V} \\
&S, w, I_V \models \phi \wedge \psi && \Leftrightarrow S, w, I_V \models \phi \text{ and } S, w, I_V \models \psi \\
&S, w, I_V \models \phi \rightarrow \psi && \Leftrightarrow \text{if } S, w, I_V \models \phi \text{ then } S, w, I_V \models \psi \\
&S, w, I_V \models \phi \leftrightarrow \psi && \Leftrightarrow S, w, I_V \models \phi \text{ iff } S, w, I_V \models \psi \\
&S, w, I_V \models \exists x_1, \ldots, x_n \phi && \Leftrightarrow \text{there is a variable int.} I'_V \text{ with } S, w, I'_V \models \phi \\
&\qquad && \qquad \text{and} I'_V \restriction_{Var\setminus\{x_1,\ldots,x_n\}} = I_V \restriction_{Var\setminus\{x_1,\ldots,x_n\}} \\
&S, w, I_V \models \forall x_1, \ldots, x_n \phi && \Leftrightarrow \text{for all variable int.} I'_V \text{ with} I'_V \restriction_{Var\setminus\{x_1,\ldots,x_n\}} = \\
&\qquad && \qquad I_V \restriction_{Var\setminus\{x_1,\ldots,x_n\}}\colon\ S, w, I'_V \models \phi \\
&S, w, I_V \models \langle\alpha\rangle\phi && \Leftrightarrow \exists w' \in \mathit{succ}_{S,I_V}(\alpha)(w)\colon\ S, w', I_V \models \phi
\end{aligned}$$

2.3. Axioms

To get an axiomatization for regular first-order update logic, we adapt a standard axiomatization for dynamic logic. Fig. 2 gives the axioms and inference rules of regular

Axioms:

(Prop)	all (instances of) axioms of propositional logic
(Inst)	$\forall x\phi \to \phi[t/x]$, with x free for t in ϕ
(Refl)	$t = t$
(K)	$[\alpha](\phi \to \psi) \to ([\alpha]\phi \to [\alpha]\psi)$
(Barcan)	$\forall x[\alpha]\phi \to [\alpha]\forall x\phi$, for $x \notin FV(\alpha)$
(Choice)	$[\alpha + \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$
(Seq)	$[\alpha\,;\beta]\phi \leftrightarrow [\alpha][\beta]\phi$
(Test)	$[\phi?]\psi \leftrightarrow (\phi \to \psi)$
(Star)	$[\alpha^*]\phi \to (\phi \wedge [\alpha][\alpha^*]\phi)$
(Ind)	$(\phi \wedge [\alpha^*](\phi \to [\alpha]\phi)) \to [\alpha^*]\phi$
(FrNUTerm1)	$t = t' \to [\alpha]t = t'$, for t and t' non-updateable
(FrNUTerm2)	$t \neq t' \to [\alpha]t \neq t'$, for t and t' non-updateable
(FrNUPred1)	$pT \to [\alpha]pT$, for $p \in NUPred$ and T non-updateable
(FrNUPred2)	$\neg pT \to [\alpha]\neg pT$, for $p \in NUPred$ and T non-updateable

Inference rules:

(MP)	$\dfrac{\phi \to \psi,\ \phi}{\psi}$
(ModGen)	$\dfrac{\phi}{[\alpha]\phi}$
(UnGen)	$\dfrac{\phi \to \psi}{\phi \to \forall x\psi}$ for $x \notin FV(\phi)$
(Sym)	$\dfrac{t_1 = t_2}{t_2 = t_1}$
(Tran)	$\dfrac{t_1 = t_2, t_2 = t_3}{t_2 = t_3}$
(Sub)	$\dfrac{t_1 = t_2}{t_1[t/x] = t_2[t/x]}$
(ConFun)	$\dfrac{t_1 = t_2}{t[t_1/x] = t[t_2/x]}$
(ConPred)	$\dfrac{t_1 = t'_1, \ldots, t_n = t'_n}{p(t_1, \ldots, t_n) \leftrightarrow p(t'_1, \ldots, t'_n)}$

Fig. 2. Axioms and inference rules of regular first-order update logic.

first-order update logic. All axioms and rules except the frame axioms are standard axioms and rules for a normal first-order modal (dynamic) logic with constant domains. Frame axioms (FrNUTerm1), (FrNUTerm2), (FrNUPred1) and (FrNUPred2) just state that the interpretation of non-updateable symbols does not change under updates. The terms used in axioms (FrNUPred1) and (FRNUPred2) must be non-updateable because otherwise, the update α could change these terms and then the axiom would actually state that the interpretation of the non-updateable symbol changes.

Soundness of all axioms but the frame axioms is standard. For the frame axioms, soundness almost immediately follows with Lemma 2.17; details are omitted.

Completeness with respect to truth in all structures should be provable along the same lines as the standard Henkin completeness proof for first-order modal logic. Note that such a completeness proof for FUL does not help us much in proving completeness when we have specified the update language. The axiomatization given above is (just like the syntax and declarative semantics) parametrized by the atomic updates that we choose. When particular alphabet of atomic updates is given, we need to provide additional axioms for these updates. Completeness is still a subject of study (see Section 4.4).

2.4. Representation of database states

The declarative semantics of first-order update logic interprets an update program as an accessibility relation on possible worlds. An operational semantics should define for each update program an operation on database states (which are sets of formulas) that, in a sense to be explained, corresponds to the accessibility relation on possible worlds. When we give an operational semantics for the (atomic) updates, we usually need to make some assumptions on the format of the database states. An important choice we must make is the amount of *indefinite information* we allow in the database state. A fairly strong restriction is, not to allow indefinite information at all; this choice results in *definite database states*.

Before we define definite database states, we need an operational semantics for the non-updateable function and predicate symbols. We have chosen to use term-rewriting. For the precise definitions of *term rewriting system*, *strong normalization*, *confluence* (or Church-Rosser) and *normal form*, we refer to [13, 25]. We assume that $NUPred = \emptyset$ and that we have a set E only containing equations that, read from left to right, form a strongly normalizing and confluent term rewriting system. In such a term rewriting system, every term has a unique normal form, which can be computed in a finite amount of time. How the normal form computation takes place is irrelevant here, we just assume that we have a function *normalize* that performs this computation (so *normalize* is a function from the immutable terms to the immutable terms).

We now give the definition of a definite database state. For the predicate symbols, we use a relational interpretation. For the updateable function symbols in a database state, we consider formulas that give the same result on their whole domain, except for some finite number of domain values on which an explicit other function result is defined.

Definition 2.23. A *definite database state* is a set of formulas, containing

- for every $p \in UPred$ a formula, the *extension formula* of p, of the form

$$\forall \bar{x}\ \ p\bar{x} \leftrightarrow (\bar{x} = T_1 \vee \cdots \vee \bar{x} = T_m),$$

  where $\bar{x}$ is a tuple of different variables and $T_1, \ldots, T_m$ are tuples of immutable terms that are in normal form;
- for every $f \in UFun$ a formula, the *extension formula* of f, of the form

$$fT_1 = t_1 \wedge \cdots \wedge fT_m = t_m \wedge \forall \bar{x}(\bar{x} \neq T_1 \wedge \cdots \wedge \bar{x} \neq T_m \rightarrow f\bar{x} = t),$$

where $\bar{x}$ is a tuple of different variables, $T_1,\ldots,T_m$ are tuples of immutable terms that are in normal form such that if $i \neq j$, then $T_i \neq T_j$, and $t_1,\ldots,t_m,t$ are immutable terms that are in normal form.

Intuitively, the extension formula for a predicate symbol defines the interpretation of the predicate symbol as a specific finite set of tuples. The extension formula for a function symbol defines the value of the function on all its arguments. This formula contains a *default* part end an *exception* part. The exception part $fT_1 = t_1 \wedge \cdots \wedge fT_m = t_m$ explicitly defines the value of the function on some specific (finite number of) arguments. The default part $\forall \bar{x}(\bar{x} \neq T_1 \wedge \cdots \wedge \bar{x} \neq T_m \rightarrow f\bar{x} = t)$ defines the value of the function on all other arguments.

The terms that are used in a definite database state are constant terms in normal form, because this is the way databases are practically used: for instance you do not store the fact that the age of a certain person is $2*5+10$ in a database, but you store the fact that the age of that person is 20.

It is convenient to view the update actions as set operations. We therefore assume that definite database states are given as functions that, given an updateable predicate symbol, produce a set $\{T_1,\ldots,T_m\}$ and given a function symbol, produce a tuple $(t,\{(T_1,t_1),\ldots,(T_m,t_m)\})$ (using the notation of the above definition). If f is such a representation of a function, then we just write the standard fT (for T a tuple of closed terms in normal form) for the result of applying the function. In this representation of definite database states, the specific choice of variables used in the database state is lost, but that is irrelevant information anyway.

In Appendix A.4 it is proven that for every algebra $\mathscr{A}$ and every definite database state DB, there is a unique possible world on $\mathscr{A}$ (written as $pw_{\mathscr{A}}(DB)$) in which DB is true. (A definite database state is a first-order logic formula, so for its evaluation, we do not need a structure, just a world.) Next, we give the formal definition of $pw_{\mathscr{A}}(DB)$.

Definition 2.24. Let $\mathscr{A}$ be an algebra and let DB be a definite database state. Then $pw_{\mathscr{A}}(DB)$ is the possible world on $\mathscr{A}$, which is defined for every $p \in UPred$ as: $\{T^{\mathscr{A}} \mid T \in DB(p)\}$ and for every $f : \langle s_1,\ldots,s_n\rangle \in UFun$ with $DB(f) = (t,\{(T_1,t_1),\ldots,(T_n,t_n)\}$ as the function that assigns $t_i^{\mathscr{A}}$ to tuples of arguments $T_i^{\mathscr{A}}$ (for all i with $1 \leqslant i \leqslant n$) and $t^{\mathscr{A}}$ to other tuples of arguments.

Database states contain only immutable terms in normal form. Therefore, if we want to store terms we have computed, we must first make sure that they are immutable terms in normal form.

Definition 2.25. Let t be a closed term and DB be a database state. The term $mc(t,DB)$ (make a immutable term of t with DB giving the interpretation of the updateable function symbols) is defined inductively as

$$mc(fT,DB) = \begin{cases} normalize(f(mc(T,DB))) & \text{if } f \in NUFun, \\ DB(f)(mc(T,DB)) & \text{if } f \in UFun. \end{cases}$$

Here, $mc(T, DB)$ for a tuple of closed terms $T = (t_1, \ldots, t_n)$ is defined as the tuple of (closed) terms (in normal form): $(mc(t_1, DB), \ldots, mc(t_n, DB))$.

The next lemma "links" the functions $pw_{\mathscr{A}}$ and mc.

Lemma 2.26. *Let DB be a definite database state, let t be a closed term and let $\mathscr{A}$ be a model of E. Then*

$$mc(t, DB)^{\mathscr{A}} = t^{\mathscr{A}, pw_{\mathscr{A}}(DB)}.$$

The proof is given in Appendix A.4.

The definite operational semantics is fairly restrictive and therefore the advantages of logic over the relational model (the possibilities to include explicit negative information and disjunctive information in the database state) have vanished. We discuss what options we have to include (some forms of) explicit negative information and disjunctive information in the database states. We concentrate on indefinite information for the predicate symbols.

More general definite databases: The definite database states described in this section only contain a finite amount of explicit positive information for every predicate symbol (just like in the relational model). We can generalize this to also allow a finite amount of explicit negative information about predicate symbols in database states. A database state then contains for every predicate symbol, a formula of the form $\forall \bar{x}\ p\bar{x} \leftrightarrow (\bar{x} = T_1 \vee \cdots \vee \bar{x} = T_m)$ or a formula of the form $\forall \bar{x}\ \neg p\bar{x} \leftrightarrow (\bar{x} = T_1 \vee \cdots \vee \bar{x} = T_m)$ (see Definition 2.23).

Local finite indefinite information databases: We can include indefinite information by allowing for every predicate symbol instead of a formula of the format of Definition 2.23, disjunctions of such formulas. This approach can also be combined with the explicit negative information approach. The kind of indefiniteness described here is called local, because we can only specify disjunctive information for single predicate symbols at a time; disjunctions over more than one predicate symbol cannot be specified. (Of course, we may have indefinite information on both p and q separately, but we can, for instance, not have the information that $p(a) \vee q(b)$ must be true.) The indefiniteness is finite, as there are still only finitely many interpretations of the predicate symbols that satisfy the formulas in the database state (as opposed to exactly one interpretation for relational database states).

Local infinite indefinite information databases: A local finite indefinite information database cannot contain the information that the value of one of the attributes in some tuple of some relation can be anything (is "unknown"), as the domain of the attribute can in general be infinite. A very general way to include such indefiniteness, is to have for every predicate symbol, a formula of the form $\forall \bar{x}\ p\bar{x} \leftrightarrow \phi$, where ϕ is a boolean combination of equations containing no variables other than those in $\bar{x}$.

Global indefinite information databases: Here we want to lift the restriction that a single "chunk" of indefinite information only concerns one predicate symbol. How to do this in a general way, while still being able to define the interpretation of atomic

update actions, is not clear to us. Just as for local indefinite information databases, we can make the distinction between finite and infinite indefiniteness.

Of all the options described above, we only consider the local finite indefinite database states in some more detail, as they are still fairly easy to handle.

Definition 2.27. A *local finite indefinite database state* is a set of closed formulas, containing for every $p \in UPred$ the *extension formula* of p, which has the form

$$\begin{aligned} &\forall \bar{x}\ (p\bar{x} \leftrightarrow (\bar{x} = T_{1,1} \vee \cdots \vee \bar{x} = T_{1,m_1})) \vee \\ &\qquad\qquad \cdots \qquad\qquad\qquad\qquad \vee \\ &\forall \bar{x}\ (p\bar{x} \leftrightarrow (\bar{x} = T_{n,1} \vee \cdots \vee \bar{x} = T_{n,m_n})) \end{aligned}$$

and for every $f \in UFun$ the *extension formula* of f, which has the same form as in Definition 2.23. Here $\bar{x}$ is a tuple of different variables of the arity of p and $T_{1,1}, \ldots, T_{n,m_n}$ are tuples of constant terms that are in normal form. We require $n \geqslant 1$ (we want a database state to be satisfiable), but m_i may be zero; the formula $\forall \bar{x}\ (p\bar{x} \leftrightarrow (\bar{x} = T_{i,1} \vee \cdots \vee \bar{x} = T_{i,m_i}))$ then is equivalent to $\forall \bar{x}\ (p\bar{x} \leftrightarrow false)$.

The very simple kind of indefinite information introduced in the previous definition is the only kind of indefinite information we will use. Without possible confusion, we can therefore write "indefinite database state" instead of "local finite indefinite database state". To give the operational semantics of atomic updates as set operations, we identify an indefinite database state DB with a function that assigns to every predicate symbol, a set of sets of tuples of immutable terms. Using the notation of Definition 2.27, we get $DB(p) = \{\{T_{1,1}, \ldots, T_{1,m_1}\}, \ldots, \{T_{n,1}, \ldots, T_{n,m_n}\}\}$. As the choice of variables in an indefinite database state is not relevant, we do not lose any information by using this alternative representation.

2.5. Operational semantics for regular operators

We consider a Plotkin-style operational semantics [31] for the update language of FUL. A Plotkin-style operational semantics consists of a proof system for *transitions* between *configurations*.

Definition 2.28. A *configuration* is either a database state σ, or a pair $\langle \sigma, \alpha \rangle$ with σ a database state and α an update program. A *transition* has the form $\langle \sigma, \alpha \rangle \to \sigma'$ (for the final step of the computation) or $\langle \sigma, \alpha \rangle \to \langle \sigma', \alpha' \rangle$ (for an intermediate step in the computation).

Remember that in our approach, a database state is a finite set of closed, non-modal formulas. A transition relation is defined by means of transition rules of the form

$$\frac{\langle DB_1, \alpha_1 \rangle \to conf_1, \langle DB_2, \alpha_2 \rangle \to conf_2, \ldots, \langle DB_n, \alpha_n \rangle \to conf_n}{\langle DB, \alpha \rangle \to conf}.$$

$$\text{(TCh1)} \quad \langle DB, \alpha + \beta \rangle \to \langle DB, \alpha \rangle$$
$$\text{(TCh2)} \quad \langle DB, \alpha + \beta \rangle \to \langle DB, \beta \rangle$$

$$\text{(TSeq1)} \quad \frac{\langle DB, \alpha \rangle \to DB'}{\langle DB, \alpha; \beta \rangle \to \langle DB', \beta \rangle}$$
$$\text{(TSeq2)} \quad \frac{\langle DB, \alpha \rangle \to \langle DB', \alpha' \rangle}{\langle DB, \alpha; \beta \rangle \to \langle DB', \alpha'; \beta \rangle}$$

$$\text{(TIter1)} \quad \langle DB, \alpha^* \rangle \to DB$$
$$\text{(TIter2)} \quad \langle DB, \alpha^* \rangle \to \langle DB, \alpha; \alpha^* \rangle$$

Fig. 3. The transition rules for the regular operators.

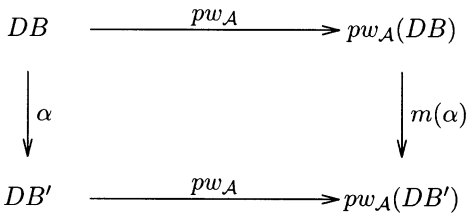

Fig. 4. Relation between operational and declarative semantics.

In this rule, $conf, conf_1, \ldots, conf_n$ are configurations. The operational semantics is given by structural induction, so all α_i in a transition rule will be subprograms of α. The rule tells us that the transition $\langle DB, \alpha \rangle \to conf$ can be made if all transitions $\langle DB_i, \alpha_i \rangle \to conf_i$ are possible. The transitions $\langle DB_i, \alpha_i \rangle \to conf_i$ are therefore called the *premises* of the transition rule, and $\langle DB, \alpha \rangle \to conf$ is called the *conclusion* of the transition rule. Of course, if we want to have any transitions at all, we need rules with an empty list of conditions ($n = 0$). We present such a rule, by only writing down the part below the line.

Fig. 3 gives the operational semantics for the regular operators by means of a transition system. The transition rules define the transition relation $\to$ as the minimal relation that satisfies (all instances of) the rules. Let $\to^*$ be the transitive closure of this relation. Then $\langle DB, \alpha \rangle \to^* DB'$ means that there is an execution of α in DB that leads to DB'.

An operational semantics for database updates must be sound and complete with respect to the declarative semantics in the following sense. Fig. 4 illustrates the idea that if we start with a database state DB, the following two paths should lead to the same possible world:

1. compute DB' with the operational semantics of α and then take the declarative semantics of the resulting relational database state DB';
2. take the declarative semantics of DB (a unique possible world) and then determine the α successor world in the declarative semantics.

Soundness is the statement that, given a path 1, a path 2 exists which makes the diagram commute and completeness is the statement that, given a path 2, a path 1 exists which makes the diagram commute. We will prove soundness and completeness for two choices of atomic updates, relational algebra assignment (Section 3) and logic database updates (Section 4).

3. Relational algebra update logic (RAUL)

Regular first-order update logic is an update logic parametrized by a set of atomic updates. In this section we choose relational algebra assignments as our atomic updates. To keep the treatment simple, we only consider atomic updates and leave regular update programs out of consideration. This is not a restriction, because all results of this section remain valid when we add regular operators to the update language. The choice of atomic updates and the choice of process combinators are orthogonal to each other.

We call the version of FUL with relational algebra updates and without regular process combinators RAUL, for relational algebra update language. We define the syntax, axioms and declarative semantics for RAUL and give an operational semantics. RAUL can be viewed as an extension of relational algebra with assignment. Syntax, axioms and declarative semantics for extensions of domain calculus and tuple calculus with assignment are studied by Spruit [33]. The operational semantics of RAUL will be used to define the operational semantics of dynamic database logic, introduced later.

Function symbols are not part of the usual definitions of relational algebra [12, 36]. In FUL, we included non-updateable function symbols right from the start to reduce the mismatch between database languages (based on predicates) and programming languages (in which functions play a central role). Function symbols are easily integrated in the syntax of relational algebra: where relational algebra uses constants and variables, we may now use *terms*. We build upon the general framework for regular first-order update logics that is given in Section 2. Although relational algebra has no construction to update function symbols, the inclusion of updateable function symbols causes no extra complexity in the definition of the declarative semantics and axiomatization of the languages, so we do not bother to throw the updateable function symbols out.

In the relational database model, attributes are referred to by name [12, 36], whereas in logic, "attributes" are referred to by position (in the list of arguments of a predicate symbol). We need to refer to these positions syntactically (for instance for the projection operator in relational algebra), so we need some syntactical representation for the positions. We assume we have a set $\mathcal{N}$ of such representations. In examples, we will use boldface natural numbers in decimal notation for the elements of $\mathcal{N}$. Note that FUL (and hence RAUL) allow the specification of abstract data types. In order to make sure that notations for positions can always be distinguished from constants and variables

Table 1
Relational databases in logic

Relational database	Logic
Relation name	Updateable predicate symbol
Table	Extension of updateable predicate symbol
Attribute	Position

introduced in a specification, we assume that the sets of constants and variables do not contain elements of $\mathcal{N}$. Note that a constant may still be a representation of a natural number; it just has to be a different representation than the one in $\mathcal{N}$. For instance we could use $succ(succ(zero))$ as a constant that denotes 2 and **2** as the element of $\mathcal{N}$ that denotes the second argument position of a predicate. We use i, j, n, m etc., for natural numbers, and k, l, etc., for syntactical representations of positions (elements of $\mathcal{N}$). Moreover, we write $[\![k]\!]$ for the interpretation of the syntactic object $k \in \mathcal{N}$ as a natural number. The definition of relational algebra below uses the positional notation for attributes and therefore differs slightly from the common definitions used in the relational model. This difference is, however, not important. Using positions, we roughly translate relational database theory concepts to logic as given in Table 1.

The relational algebra consists of the following five basic operators:

Product: Cartesian product of two relations.

Selection: Select some rows of a relation (given a boolean combination of comparisons).

Union: Set union of two relations (that must be "union compatible").

Difference: Set difference of two relations (that must be "union compatible").

Projection: Select some columns of a relation.

Other well-known relational operators, like for instance the "join" can be expressed in terms of the five basic operators above.

To illustrate how these operators can be used to define a relation in terms of existing relations, we give one (informal) example which uses the Cartesian product, selection and projection operators (for a more detailed introduction to the relational algebra, we refer to [12, 36]). Assume we have two relations $person(id, name)$ and $parent(id_1, id_2)$. Furthermore, assume $person'$ and $parent'$ are copies of person and parent, respectively. Then the following relational algebra expression gives the pairs of names of persons that are in the "grandparent" relation to each other:

$$\begin{aligned}((&person \times parent \times parent' \times person') \ \textbf{where} \\ &(person.id = parent.id_1 \wedge \\ &\ parent.id_2 = parent'.id_1 \wedge \\ &\ parent'.id_2 = person'.id \\ &)) \ [person.name, person'.name]\end{aligned}$$

In the next sections, we introduce the syntax, the declarative semantics, a proof system and an operational semantics for RAUL.

3.1. Syntax of RAUL

We define four syntactic classes: the relational algebra tests, the relational algebra expressions, the relational algebra updates and the relational algebra formulas. Relational algebra tests are just boolean combinations of attribute–constant or attribute–attribute comparisons. They are used as the condition in the relational algebra selection operator. Relational algebra expressions are well known from relational database theory: they define a relation in terms of existing relations, using the five relational algebra operators. Relational algebra updates are used to make the computed result of a relational algebra expression "permanent", by assigning it to a predicate symbol. Finally, the relational algebra formulas are plain first-order update logic formulas, where we use the relational algebra updates as atomic updates.

Definition 3.1. The *relational algebra tests* are parametrized by a tuple of sort symbols $(s_1,\ldots,s_n)$ and defined inductively as

1. $k=t$ is an $(s_1,\ldots,s_n)$ relational algebra test, for any $k\in\mathcal{N}$ with $1\leqslant[\![k]\!]\leqslant n$ and any immutable term t of sort $s_{[\![k]\!]}$;
2. $k=l$ is an $(s_1,\ldots,s_n)$ relational algebra test, for any $k,l\in\mathcal{N}$ with $1\leqslant[\![k]\!],[\![l]\!]\leqslant n$ and $s_{[\![k]\!]}=s_{[\![l]\!]}$;
3. $\phi_1\vee\phi_2$ is an $(s_1,\ldots,s_n)$ relational algebra test, for any $(s_1,\ldots,s_n)$ relational algebra tests ϕ_1 and ϕ_2;
4. $\neg\phi$ is an $(s_1,\ldots,s_n)$ relational algebra test, for any $(s_1,\ldots,s_n)$ relational algebra test ϕ;
5. the only $(s_1,\ldots,s_n)$ relational algebra tests are those, given by 1–4.

The relational algebra tests $k=t$ and l are the *atomic* tests; they test if an attribute (identified by its place) is equal to a term or to an other attribute. Clauses 3 and 4 state that relational algebra tests are formed by taking boolean combinations of the atomic tests. (We just assume other boolean connectives, like $\wedge$ and $\rightarrow$ are defined in the standard way in terms of $\vee$ and $\neg$.) Note that if ϕ is an $(s_1,\ldots,s_n)$ relational algebra test, then it is also an $(s_1,\ldots,s_n,s_{n+1},\ldots,s_{n+m})$ relational algebra test, for any sort symbols $s_{n+1},\ldots,s_{n+m}$.

Definition 3.2. The *relational algebra expressions* are parametrized by a tuple of sort symbols $(s_1,\ldots,s_n)$ and defined inductively as

1. p is an $(s_1,\ldots,s_n)$ relational algebra expression, for any p: $\langle s_1,\ldots,s_n\rangle\in UPred$;
2. $e_1\cup e_2$ and $e_1\backslash e_2$ are $(s_1,\ldots,s_n)$ relational algebra expressions, for any $(s_1,\ldots,s_n)$ relational algebra expressions e_1 and e_2;
3. $e_1\times e_2$ is an $(s_1,\ldots,s_n,s_{n+1},\ldots,s_{n+m})$ relational algebra expression, for any $(s_1,\ldots,s_n)$ relational algebra expression e_1 and $(s_{n+1},\ldots s_{n+m})$ relational algebra expression e_2;
4. $e[k_1,\ldots,k_n]$ is an $\mathrm{m}(s_{[\![k_1]\!]},\ldots,s_{[\![k_n]\!]})$ relational algebra expression, for any $(s_1,\ldots,s_m)$ relational algebra expression e and $k_1,\ldots,k_n\in\mathcal{N}$ with $1\leqslant[\![k_1]\!]<\cdots<[\![k_n]\!]\leqslant m$;

5. *e where* ϕ is an $(s_1,\ldots,s_n)$ relational algebra expression, for any $(s_1,\ldots,s_n)$ relational algebra expression e and $(s_1,\ldots,s_n)$ relational algebra test ϕ;
6. The only relational algebra expressions, are those given by 1–5.

The above definition gives a specific notation for the relational algebra operators. Of course, $\cup$ is union, $\setminus$ is difference, $\times$ is product, $[\cdots]$ is projection and **where** is selection. Note that in clause 4. we must have $m \geqslant n$. Also note that in clause 1. we do not allow non-updateable predicate symbols, as they in general may have an infinite extension.

Definition 3.3. A *relational algebra update* has the form $p := e$, for $p: \langle s_1,\ldots,s_n\rangle \in UPred$ and e an $(s_1,\ldots,s_n)$ relational algebra expression.

Intuitively, the assignment $p := e$ is an action that computes the value of e (a relation, which is just a set of tuples) and assigns the result to the updateable predicate symbol p.

Definition 3.4. We use the relational algebra updates as update actions in Definition 2.5 and we call the resulting set of formulas the *relational algebra formulas*.

All actions α we consider, are of the form $p := e$. These actions are *deterministic*; if we know exactly what is true before the update, then we know exactly what holds after the update. Moreover, $p := e$ actions are always successful; there always is a next state of this update action. Therefore, the relational algebra formulas $[p := e]\phi$ and $\langle p := e\rangle\phi$ are equivalent (true in the same worlds of the same structures). The reader may check this equivalence as an exercise for the declarative semantics that is given in the next section.

3.2. *Declarative semantics of* RAUL

Intuitively, the four syntactic classes defined in the previous section are given a semantics in the following way:

- A relational test, evaluated for a single tuple of domain elements, yields either true or false. (This evaluation presupposes some algebra.)
- A relational algebra expression is evaluated in a possible world and yields a set of tuples.
- A relational algebra update is evaluated in a structure and yields a relation on possible worlds. This relation gives which world results if we execute a relational algebra update in a world.
- A relational algebra formula is evaluated in a possible world of a structure under a variable interpretation and yields true or false.

We now give the semantics of the four syntactic classes formally.

Definition 3.5. Let $\mathscr{A}$ be an algebra and let $(s_1,\ldots,s_n)$ be a tuple of sort symbols. The relation $\models_A$ between tuples from $s_1^{\mathscr{A}} \times \cdots \times s_n^{\mathscr{A}}$ and $(s_1,\ldots,s_n)$ relational tests is defined inductively as

$$
\begin{aligned}
&(d_1,\ldots,d_n) \models_{\mathscr{A}} k = t \quad &&\Leftrightarrow d_{[\![k]\!]} = t^{\mathscr{A}}\\
&(d_1,\ldots,d_n) \models_{\mathscr{A}} k = l \quad &&\Leftrightarrow d_{[\![k]\!]} = d_{[\![l]\!]}\\
&(d_1,\ldots,d_n) \models_{\mathscr{A}} \phi_1 \vee \phi_2 &&\Leftrightarrow (d_1,\ldots,d_n) \models_{\mathscr{A}} \phi_1 \text{ or } (d_1,\ldots,d_n) \models_{\mathscr{A}} \phi_2\\
&(d_1,\ldots,d_n) \models_{\mathscr{A}} \neg\phi \quad &&\Leftrightarrow (d_1,\ldots,d_n) \not\models_{\mathscr{A}} \phi.
\end{aligned}
$$

Note that in fact, we have a relation $\models_{\mathscr{A}}$ for every tuple of sorts $(s_1,\ldots,s_n)$, but as no confusion can arise, this tuple is not given as a subscript of $\models$. Moreover, the subscript $\mathscr{A}$ is usually also omitted from $\models_{\mathscr{A}}$ if no confusion can arise.

The symbols $\cup$, $\setminus$ and $\times$ we used above in the syntax of RAUL, are also used below as the notation for the standard set operations union, set difference and Cartesian product. Furthermore, we use a (semantic) projection operation, which we define explicitly as it is less standard.

Definition 3.6. Let n and m be natural numbers and let $i_1,\ldots,i_n$ be a sequence of natural numbers such that $1 \leqslant i_1 < \cdots < i_n \leqslant m$. Then for all m-tuples $\bar{d} = (d_1,\ldots,d_m)$, we define $\bar{d}[i_1,\ldots,i_n]$ to be the n-tuple $(d_{i_1},\ldots,d_{i_n})$. For a set of m-tuples D, $D[i_1,\ldots,i_n]$ is defined as the following set of n-tuples: $\{\bar{d}[i_1,\ldots,i_n] \mid \bar{d} \in D\}$.

Definition 3.7. Let $\mathscr{A}$ be an algebra and let w be a possible world on $\mathscr{A}$. We extend w inductively to relational algebra expressions as follows:

$$
\begin{aligned}
&w(e_1 \cup e_2) = w(e_1) \cup w(e_2),\\
&w(e_1 \setminus e_2) = w(e_1) \setminus w(e_2),\\
&w(e_1 \times e_2) = w(e_1) \times w(e_2),\\
&w(e[k_1,\ldots,k_n]) = w(e)[[\![k_1]\!],\ldots,[\![k_n]\!]],\\
&w(e \textbf{ where } \phi) = \{\bar{d} \in w(e) \mid \bar{d} \models \phi\}.
\end{aligned}
$$

Definition 3.8. Let S be a structure on the algebra $\mathscr{A}$. We define the function m_S which takes a relational algebra update $p := e$ and gives a relation on S as follows:

$$m_S(p := e) = \{(w, w') \in S \times S \mid w' = w\{p \mapsto w(e)\}\}.$$

Next to stating how the extension of p changes under the update $p := e$, the above definition also states that the extensions of all other predicate symbols remain the same (frame assumption). Note that as relational algebra updates do not contain free variables, we omit the subscript I_V from m_{S,I_V}. (The meaning function m_{S,I_V} was introduced in Definition 2.18.) Using the definition of m_S, Definition 2.20 gives the semantics of relational algebra formulas.

3.3. Axiomatization of RAUL

In this section, we axiomatize truth of relational algebra formulas in full structures. This is not quite satisfactory (but at the moment, it is the best we can do), as truth in full structures does *not* coincide with truth in relational structures (defined in Definition 2.12). Consider the formula

$$\forall x(x \neq inc(x)) \rightarrow \neg(p(zero) \wedge \forall x(p(x) \rightarrow p(inc(x)))).$$

This formula states that the extension of p does not contain all elements from an infinite domain. Therefore, this formula is true in relational structures, but is in general not true in full structures.

We take the axiomatization of FUL and add axioms that are specific for the relational algebra updates. For the axiomatization of the selection operator, we need simultaneous substitution of terms for positions in relational algebra tests. This way, we can "translate" relational algebra tests to first-order logic formulas. We define this simultaneous substitution rather informally.

Definition 3.9. Let ϕ be an $(s_1,\ldots,s_n)$ relational algebra test and let $T=(t_1,\ldots,t_n)$ be a tuple of terms of sorts $s_1,\ldots,s_n$, respectively. Then the *simultaneous substitution* of the terms in T for positions in ϕ, denoted $\phi[T/\mathcal{N}]$ is the first-order logic formula that we get by replacing all occurrences of l in ϕ by $t_{[\![l]\!]}$, for all $l \in \mathcal{N}$ with $1 \leqslant [\![l]\!] \leqslant n$.

Fig. 5 gives an axiom system for RAUL. Appendix A.1 contains soundness proofs of these axioms with respect to full structures. In the next paragraphs, we give intuitive explanations of the axioms.

Axioms (PosAt) and (NegAt) are evident; with these axioms we can derive that the extension of p after the update is the same as the extension of q before the update. Note that only an axiom (Atom) $[p:=q]pT \leftrightarrow qT$ is *not* sufficient. In particular, if we only have (Atom), then we cannot derive (NegAt). However, it is not too difficult to show that the axiom (Atom) in combination with the determinism axiom $\langle p:=q\rangle pT \rightarrow [p:=q]pT$ is equivalent with the combination of (NegAt) and (PosAt).

The axioms (Disj) and (Diff) are also evident. For the axiom (Diff), we note that as the "box" and "diamond" coincide for RAUL (because all update actions are functional and successor worlds always exist), we could also have written $[p:=e_2]\neg pT$ instead of $\neg[p:=e_2]pT$ in this axiom. For the axiom (Prod), we see that we need to have extra predicate symbols p_1 and p_2 of the correct sorts; for instance, we can only use the axiom for the update $p:=p[\mathbf{1}]\times p[\mathbf{2}]$ (for a binary predicate symbol p: $\langle s_1,s_2\rangle \in UPred$), if we also have unary (updateable) predicate symbols of sorts s_1 and s_2. This is also true for the axiom (Proj): p' is a way to refer to the extension of e syntactically. The axiom (Sel) "translates" a relational algebra test in a selection to a first-order logic formula (so that we can reason about it in the proof system).

The frame axioms (FrUFun1) and (FrUFun2) state that updateable function symbols do not change under relational algebra updates; frame axioms FrUPred1) and

Axioms:

(PoSAt)	$qT \rightarrow [p := q]pT$
(NegAt)	$\neg qT \rightarrow [p := q]\neg pT$
(Disj)	$[p := e_1 \cup e_2]pT \leftrightarrow ([p := e_1]pT \vee [p := e_2]pT)$
(Diff)	$[p := e_1 \setminus e_2]pT \leftrightarrow ([p := e_1]pT \wedge \neg[p := e_2]pT)$
(Prod)	$[p := e_1 \times e_2]pT_1T_2 \leftrightarrow ([p_1 := e_1]p_1T_1 \wedge [p_2 := e_2]p_2T_2)$
(Proj)	$[p := e[k_1, \ldots, k_n]]p(t_1, \ldots, t_n) \leftrightarrow \exists x_1, \ldots, x_m$ $([p' := e]p'(x_1, \ldots, x_m) \wedge t_1 = x_{[\![k_1]\!]} \wedge \cdots \wedge t_n = x_{[\![k_n]\!]})$, for $x_1, \ldots, x_m$ different variables with $\forall i, j : x_i \notin t_j$
(Sel)	$[p := e \textbf{ where } \phi]pT \leftrightarrow ([p := e]pT \wedge \phi[T/\mathcal{N}])$
(FrUFun1)	$fT = t \rightarrow [p := e]fT = t$, for $f \in UFun$
(FrUFun2)	$fT \neq t \rightarrow [p := e]fT \neq t$, for $f \in UFun$
(FrUPred1)	$qT \rightarrow [p := e]qT$, for $q \neq p \in UPred$
(FrUPred2)	$\neg qT \rightarrow [p := e]\neg qT$, for $q \neq p \in UPred$
(SuccEx)	$\langle p := e \rangle true$

Fig. 5. Axioms for RAUL.

(FrUPred2) state that all updateable predicate symbols that are not explicitly updated, have the same interpretation after the update as they had before the update. For the non-updateable predicate and function symbols, we already have the FUL axioms (FrNUTerm), (FrNUPred1) and (FrNUPred2) (see Fig. 2).

3.4. Example correctness proof in RAUL

We give an example of how the axiomatization can be used to prove that an implementation of an update is correct, relative to its specification.

Example 3.10. Assume $person \in Sorts$, $p : \langle person, person \rangle \in UPred$ and $gp : \langle person, person \rangle \in UPred$. The intended meaning of $p(a, b)$ is that a is a parent of b; the intended meaning of $gp(a, b)$ is that a is a grandparent of b. Now, suppose we know that the p relation is filled correctly and that we have to compute the gp relation. The specification of this update is as follows: we have to find a relational algebra update α such that

$$\forall x, z([\alpha]gp(x, z) \leftrightarrow \exists y(p(x, y) \wedge p(y, z)))$$

This specification states that after the update, x is a grandparent of z iff before the update there is a person y such that x is a parent of y and z is a child of y. The specification also should contain frame assumptions that state that the interpretations of all predicate symbols other than gp do not change under the update α. As the frame assumptions in a specification can always easily be proven with the frame axioms, we

have ignored the frame assumptions here. The update α can be implemented in the following way:

$$\alpha = g\,p := ((p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3})[\mathbf{1},\mathbf{4}].$$

To show that this implementation is correct, we must prove that the formula

$$\forall x,z([g\,p := ((p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3})[\mathbf{1},\mathbf{4}]]\ g\,p\ (x,z) \leftrightarrow \exists y(p(x,y) \wedge p(y,z)))$$

is a theorem in the axiomatization of RAUL. Below, we give a derivation of this formula in the proof system. We mainly want to illustrate the relational algebra axioms, and we have therefore used some shortcuts: if a formula follows from some other formulas by straightforward first-order modal logic reasoning (or is a first-order modal logic theorem), then we just write "foml" (for first-order modal logic). As a special case, we write "re" for the derivable inference rule of replacing equivalent subformulas. We assume an extra predicate symbol $p_4 : \langle person, person, person, person\rangle \in UPred$ is present (without such an extra predicate symbol, the proof is impossible). We also use the "axiom" (Atom). This is not really an axiom, but it follows easily from the axioms (PosAt) and (NegAt); see the discussion immediately following the axioms.

1. $[p := p]p(x,y) \leftrightarrow p(x,y)$ (Atom)
2. $[p := p]p(y,z) \leftrightarrow p(y,z)$ (Atom)
3. $[p_4 := p \times p]p_4(x,y,y,z) \leftrightarrow$
   $[p := p]p(x,y) \wedge [p := p]p(y,z)$ (Prod)
4. $[p_4 := p \times p]p_4(x,y,y,z) \leftrightarrow p(x,y) \wedge p(y,z)$ re, 3, 2, 1
5. $[p_4 := (p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3}]p_4(x,x_2,x_3,z) \leftrightarrow$
   $[p_4 := p \times p]p_4(x,x_2,x_3,z) \wedge x_2 = x_3$ (Sel)
6. $\exists x_2,x_3[p_4 := (p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3}]p_4(x,x_2,x_3,z) \leftrightarrow$
   $\exists y([p_4 := p \times p]p_4(x,y,y,z))$ foml, 5
7. $\exists x_2,x_3[p_4 := (p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3}]p_4(x,x_2,x_3,z) \leftrightarrow$
   $\exists y(p(x,y) \wedge p(y,z))$ re, 6, 4
8. $\exists x_1,x_2,x_3,x_4([p_4 := (p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3}]p_4(x_1,x_2,x_3,x_4)$
   $\wedge x = x_1 \wedge z = x_4) \leftrightarrow \exists y(p(x,y) \wedge p(y,z))$ foml, 7
9. $[gp := ((p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3})[\mathbf{1},\mathbf{4}]]gp(x,z) \leftrightarrow$
   $\exists x_1,x_2,x_3,x_4(x = x_1 \wedge z = x_4 \wedge$
   $[p_4 := (p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3}]p_4(x_1,x_2,x_3,x_4))$ (Proj)
10. $[gp := ((p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3})[\mathbf{1},\mathbf{4}]]gp(x,z) \leftrightarrow$
    $\exists y(p(x,y) \wedge p(y,z))$ re, 9, 8
11. $\forall x,z([gp := ((p \times p)\ \textbf{where}\ \mathbf{2} = \mathbf{3})[\mathbf{1},\mathbf{4}]]gp(x,z) \leftrightarrow$
    $\exists y(p(x,y) \wedge p(y,z)))$ (UnGen), 10

3.5. Operational definite semantics of RAUL

Using definite database states as defined in Section 2.4 (Definition 2.24), we give the transition rules for the update language of RAUL. As we only have atomic actions in RAUL, all transitions we can derive result in a database state (and not in a general

configuration). But as already noted at the start of this section, we can easily add the regular operators and their transition rules to RAUL to get an operational semantics for a regular language with relational algebra assignments as atomic actions.

In definite database states, we store both information on the updateable predicate symbols and the updateable function symbols. RAUL contains no update actions for the updateable function symbols, so for all transitions $\langle DB, \alpha\rangle \rightarrow DB'$ we can derive in the transition system, the information on the updateable function symbols in DB' will be the same as the information on the updateable function symbols in DB.

For the operational semantics of the selection operator, we first define an evaluation function that evaluates the truth value of a relational algebra test in a relational database state. This function resembles the semantics of relational algebra tests very closely, but is formulated as a function that is intended to be computed (instead of a declarative definition).

Definition 3.11. Let $\mathscr{A}$ be an algebra, let ϕ be a $(s_1,\ldots,s_n)$ relational algebra test and let s be a finite set of tuples of constant terms of sorts $s_1,\ldots,s_n$ that are in normal form. The set $cs(s,\phi)$ (the computed selection of tuples of s that satisfy ϕ) is defined inductively on the structure of ϕ as

- $cs(s, k = t) = \{(t_1,\ldots,t_n) \in s \mid t_{[\![k]\!]} = normalize(t)\}$;
- $cs(s, k = l) = \{(t_1,\ldots,t_n) \in s \mid t_{[\![k]\!]} = t_{[\![l]\!]}\}$;
- $cs(s, \phi \vee \psi) = cs(s,\phi) \cup cs(s,\psi)$;
- $cs(s, \neg\phi) = s \backslash cs(s,\phi)$.

The function cs is computable, because all sets of tuples are finite, and *normalize* is a computable function.

Next to the function cs, we also use $\cup$, $\backslash$ and $\times$ as the familiar set operations union, difference and Cartesian product in the transition rules. Moreover, we use the projection operator as given in Definition 3.6. The transition rules for the atomic updates of RAUL are given in Table 2.

In Appendix A.2 we prove that the operational definite semantics of RAUL is equivalent to the declarative semantics of atomic relational algebra updates in full structures on initial algebras. We do not have the equivalence with respect to the declarative semantics in arbitrary structures on arbitrary algebras. The reasons for this are as follows:

- For the operational semantics of the function symbols in RAUL, we use term rewriting. Therefore, we assumed in Section 2.4 the equations (of some given equational specification), read from left to right, form a strongly normalizing and confluent term rewriting system. Under these circumstances, term rewriting operationalizes the initial algebra [13, 25]. Naturally, we then can only expect to have soundness and completeness of the operational semantics with respect to the declarative semantics of updates in structures that are based on an initial algebra (and not structures based on an arbitrary algebra).
- There is no notion of restricting the allowed relational database states in the operational semantics. In other words, in the operational semantics, we assume that all

Table 2
The transition rules for definite RAUL

(TAtom)	$\langle DB, p := q\rangle \to DB\{p \mapsto DB(q)\}$
(TDisj)	$\dfrac{\langle DB, p := e_1\rangle \to DB_1, \langle DB, p := e_2\rangle \to DB_2}{\langle DB, p := e_1 \cup e_2\rangle \to DB\{p \mapsto DB_1(p) \cup DB_2(p)\}}$
(TDiff)	$\dfrac{\langle DB, p := e_1\rangle \to DB_1, \langle DB, p := e_2\rangle \to DB_2}{\langle DB, p := e_1 \setminus e_2\rangle \to DB\{p \mapsto DB_1(p) \setminus DB_2(p)\}}$
(TProd)	$\dfrac{\langle DB, p_1 := e_1\rangle \to DB_1, \langle DB, p_2 := e_2\rangle \to DB_2}{\langle DB, p := e_1 \times e_2\rangle \to DB\{p \mapsto DB_1(p_1) \times DB_2(p_2)\}}$
(TProj)	$\dfrac{\langle DB, q := e\rangle \to DB'}{\langle DB, p := e[k_1, \ldots, k_n]\rangle \to DB\{p \mapsto DB'(q)[[\![k_1]\!], \ldots, [\![k_n]\!]]\}}$
(TSel)	$\dfrac{\langle DB, p := e\rangle \to DB'}{\langle DB, p := e \textbf{ where } \phi\rangle \to DB\{p \mapsto cs(DB'(p), \phi)\}}$

relational database states exist. Naturally, we can then only expect to prove soundness and completeness of the operational semantics with respect to a declarative semantics that also assumes that all (relational) states (possible worlds) are present. We therefore use *full* structures, but we could also have used *relational* structures.

3.6. Operational indefinite semantics of RAUL

In this section, we consider an operational semantics of RAUL for indefinite database states (Definition 2.27) instead of the definite database states (Definition 2.23) that were considered above. The declarative semantics of a relational algebra update is a relation on possible worlds. In the declarative semantics, executing a relational algebra update in a set of worlds is therefore already defined: just execute the update in every world and take the set of all the worlds that results. As we want the operational semantics of updates to be equivalent to the declarative semantics, we must have a similar procedure for the operational indefinite semantics. We can define such an operational semantics for RAUL in a surprisingly simple way: we take exactly the same transition system as we had for the relational operational semantics, just using extensions of the operators $\cup$, $\setminus$, $\times$, $[\cdots]$ and the *cs* function to sets of sets of tuples. The only thing we have to do, is to define these extensions. For the standard set operators $\cup$, $\setminus$ and $\times$, we have to use a different notation, as these operators are already defined on sets of sets of tuples (but not with the semantics that we want here).

Definition 3.12 (*Extensions of standard set operators*). Let A and B be sets of sets of elements. Then

- $A \cup^{\mathrm{e}} B = \{a \cup b \mid a \in A \text{ and } b \in B\}$;

Table 3
The transition rules for indefinite RAUL

(TAtom′)	$\langle DB, p := q\rangle \to DB\{p \mapsto DB(q)\}$
(TDisj′)	$\dfrac{\langle DB, p := e_1\rangle \to DB_1, \langle DB, p := e_2\rangle \to DB_2}{\langle DB, p := e_1 \cup e_2\rangle \to DB\{p \mapsto DB_1(p) \cup^{e} DB_2(p)\}}$
(TDiff′)	$\dfrac{\langle DB, p := e_1\rangle \to DB_1, \langle DB, p := e_2\rangle \to DB_2}{\langle DB, p := e_1 \setminus e_2\rangle \to DB\{p \mapsto DB_1(p) \setminus^{e} DB_2(p)\}}$
(TProd′)	$\dfrac{\langle DB, p_1 := e_1\rangle \to DB_1, \langle DB, p_2 := e_2\rangle \to DB_2}{\langle DB, p := e_1 \times e_2\rangle \to DB\{p \mapsto DB_1(p_1) \times^{e} DB_2(p_2)\}}$
(TProj′)	$\dfrac{\langle DB, q := e\rangle \to DB'}{\langle DB, p := e[k_1, \ldots, k_n]\rangle \to DB\{p \mapsto DB'(q)[[\![k_1]\!], \ldots, [\![k_n]\!]]\}}$
(TSel′)	$\dfrac{\langle DB, p := e\rangle \to DB'}{\langle DB, p := e \textbf{ where } \phi\rangle \to DB\{p \mapsto cs(DB'(p), \phi)\}}$

- $A \setminus^{e} B = \{a \setminus b \mid a \in A \text{ and } b \in B\}$;
- $A \times^{e} B = \{a \times b \mid a \in A \text{ and } b \in B\}$.

The definition of the extension of the projection operator and the computed selection functions to sets of sets of tuples are also straightforward.

Definition 3.13. Let n and m be natural numbers and let $i_1, \ldots, i_n$ be a sequence of natural numbers such that $1 \leqslant i_1 < \cdots < i_n \leqslant m$. Then for all sets of sets of m-tuples $\mathscr{D}$, we define $\mathscr{D}[i_1, \ldots, i_n]$ as $\{D[i_1, \ldots, i_n] \mid D \in \mathscr{D}\}$.

Definition 3.14. Let ϕ be a $(s_1, \ldots, s_n)$ relational algebra test and let S be a finite set of finite sets of tuples of immutable terms of sorts $s_1, \ldots, s_n$ that are in normal form. Then $cs(S, \phi) = \{cs(s, \phi) \mid s \in S\}$.

Note that the definitions of the extensions of the relational operators do not seem very operational. But the reader may check that for finite sets of finite sets of tuples, all operations can indeed be computed in a finite amount of time.

As said before, the transition system we get contains exactly the same transition rules as the existing transition system for the definite operational semantics of RAUL, replacing the relational operators by their extended variants. These transition rules are given in Table 3.

Soundness and completeness of the operational indefinite semantics of RAUL with respect to the declarative semantics can be proven in a similar way as was done for the operational definite semantics; this proof has been omitted.

4. Dynamic database logic (DDL)

We now turn to dynamic database logic (DDL), which is an instantiation of FUL in which the atomic updates are insertions, updates and deletions of tuples in the extension of base predicates or atomic updates to functions. We define the syntax, declarative semantics and axiomatization of DDL in separate subsections. Next, we note that the axioms are sound and, for the limited case when Reiter's domain closure and unique naming assumptions are satisfied, complete. Completeness in the general case is however still an open problem. Finally, we show that the update language of DDL is "update complete".

4.1. Syntax of DDL

We define the atomic updates. The first four clauses provide the atomic insert, delete, change and assignment actions; the fifth clause provides an extra operator that can be used to model object creation.

Definition 4.1. The set of DDL update programs is defined as in Definition 2.5 where we take as atomic updates:

1. $\&X\mathcal{I}pT$ **where** ϕ is an update program (called *atomic insertion*), for any predicate declaration $p:\langle s_1,\ldots,s_n\rangle \in UPred$, tuple of terms T of sorts $s_1,\ldots,s_n$, finite set of variables X, and formula ϕ;
2. $\&X\mathcal{D}pT\phi$ **where** is an update program (called *atomic deletion*), for any predicate declaration $p:\langle s_1,\ldots,s_n\rangle \in UPred$, tuple of terms T of sorts $s_1,\ldots,s_n$, finite set of variables X, and formula ϕ;
3. $\&X\mathcal{U}pT \to T'$ **where** ϕ is an update program (called *atomic update*), for any predicate declaration $p:\langle s_1,\ldots,s_n\rangle \in UPred$, tuples of terms T and T' of sorts $s_1,\ldots,s_n$, finite set of variables X, and formula ϕ;
4. $fT := t$ is an update program (called *assignment*), for any function declaration $f:\langle s_1,\ldots,s_n\rangle \to s \in UFun$, tuple of terms T of sorts $s_1,\ldots,s_n$ and term t of sort s;
5. $+X\ \alpha$ **where** ϕ is an update program (called *conditional choice*), for any finite set of variables X, formula ϕ and update program α.

The intuitive meaning of $\&X\mathcal{I}pT$ **where** ϕ is: for all variable interpretations of X that make ϕ true, simultaneously add the tuple T to the extension of p. In database terms, one can say: for all variable interpretations of X that make ϕ true, simultaneously insert the tuple T into the table p. The action $\&X\mathcal{D}pT\phi$ **where** deletes instead of inserts tuples. If a tuple that is already present is inserted, or if a tuple that is not present is deleted, then nothing changes. The intuitive meaning of $\&X\mathcal{U}pT \to T'$ **where** ϕ is: for all variable interpretations of X that make ϕ true, simultaneously replace the tuple T in p by T'. As different tuples may be changed into the same tuple, and duplicates are not stored, an atomic update to a predicate may decrease the number of tuples in the interpretation of that predicate. The intended meaning of the atomic assignment

$fT := t$ is to change the value of the function f for the tuple of arguments T to t. The intended meaning of the conditional choice $+X\ \alpha$ **where** ϕ is to execute α for *one* of the possible assignments to the variables in X that makes ϕ true.

For notational convenience, when an atomic update has an empty set of variables, we omit $\&\emptyset$ and when the condition part of an atomic update or conditional choice is the formula *true*, we omit **where** *true*. This means we allow single tuple update programs such as $Ip(x_1,\ldots,x_n)$, containing only free variables, as well as bulk updates such as $\&x_1 : s_1,\ldots,x_n : s_n\ Ip(x_1,\ldots,x_n)$ **where** $\phi(x_1,\ldots,x_n)$, which are closed formulas.

We give a few example update programs. First of all, a simple example illustrating the atomic insertion and deletion.

Example 4.2. Suppose we have a database that stores information about persons and we have the binary predicate symbols p and gp, for the parent and grandparent relations, respectively. Then the update program

$$\&\{x,y\}\mathscr{D}gp(x,y); \&\{x,y,z\}\mathscr{I}gp(x,z)\ \textbf{where}\ \ p(x,y)\wedge p(y,z)$$

computes the grandparent relation from the parent relation (by first making the grandparent relation empty, and then filling it with all pairs of persons for which there is someone that is a child of the first and a parent of the second person).

Atomic update actions can be expressed in terms of atomic insert and delete actions, provided that we have extra predicates to store temporary information. We give an example which shows this and we argue why in general, extra predicate symbols are necessary.

Example 4.3. Assume we have sorts *id* (of employee identifiers) and *nat* (of natural numbers) and assume we have the usual (non-updateable) functions and predicates on natural numbers (like addition and less than). Furthermore, assume we have the updateable predicate symbol $sal : \langle id, nat\rangle$. We give the specification of the action $IncSal(m)$ (which increases the salaries of all employees by some fixed amount):

$$sal(i,n) \rightarrow [IncSal(m)]\ \forall n'(sal(i,n') \leftrightarrow n' = n + m)$$

(Normally, the specification should also contain frame assumptions that state that the interpretation of all other updateable symbols remains unchanged under the $IncSal(m)$ action, but this is ignored for this example.) Using an atomic update, we can easily implement $IncSal(m)$ as follows:

$$IncSal(m) = \&\{i,n\}\mathscr{U}sal(i,n) \rightarrow (i,n+m).$$

We now give an implementation of $IncSal(m)$, not using atomic updates, but only atomic insertions and deletions (and an extra temporary predicate symbol sal' of which

we do not mind how its interpretation changes under the update):

$$IncSal(m) = \&\{i,n\}\mathscr{D}sal'(i,n); \&\{i,n\}\mathscr{I}sal'(i,n) \textbf{ where } sal(i,n);$$
$$\&\{i,n\}\mathscr{D}sal(i,n); \&\{i,n\}\mathscr{I}sal(i,n+m) \textbf{ where } sal'(i,n).$$

The first line copies the extension of sal to sal' and the second line then fills sal from sal'. This formulation of the update is awkward to say the least and the formulation with atomic update is much easier.

The above example shows how to implement $IncSal(m)$ with the sequential composition of atomic inserts and deletes, using an extra predicate symbol to store temporary information. It will be clear that we can always "implement" atomic updates in such a way: first copy the contents of the updated predicate symbol to the temporary predicate symbol, delete the contents of the updated predicate symbol and finally "fill" the updated predicate symbol using the information in the temporary predicate symbol. We now show for the above example that we can also implement the $IncSal(m)$ action in DDL with only insert and delete and without an extra predicate symbol. Informally, the idea is to do the following two steps:

- As long as there is an employee with only one salary, pick such an employee and add a tuple with an increased salary.
- As long as there is an employee with one salary that is m less than an other salary that he has, pick such an employee and delete his lowest salary.

This is specified in DDL in the following way (assuming that m is not 0):

$$(+\{i,n\}\mathscr{I}sal(i,n+m) \textbf{ where } sal(i,n) \wedge \neg\exists n'(sal(i,n') \wedge n \neq n'))^*;$$
$$(\forall i,n(sal(i,n) \rightarrow \exists n'(sal(i,n') \wedge n' \neq n)))?;$$
$$(+\{i,n,n'\}\mathscr{D}sal(i,n) \textbf{ where } sal(i,n) \wedge sal(i,n') \wedge n' = n+m);$$
$$(\forall i,n,n'(sal(i,n) \wedge sal(i,n') \rightarrow n = n'))?$$

There are two reasons why the above "trick" works. First of all, the employee is the "key" of the relation and therefore we cannot have one employee with two different salaries in the initial situation. If we do not have such a constraint, then it is easy to see that the above update program does not do what we want. Second, we have a way to tell, when we have two tuples of the same employee in the relation, which one is the updated tuple and which one the original (by looking which salary is higher). Without this information we cannot do the second step of the above update program. With these considerations, it now is easy to construct an example in which for the implementation of an atomic update by atomic inserts and deletes, we cannot do without the extra predicate symbols.

Example 4.4. Assume we have the sorts *id* (again for the person identifiers) and *sex* with constants m (for male) and f (for female). Furthermore, assume we have the

updateable predicate symbol $sex : \langle id, sex \rangle$. Now suppose (not very realistically) that the sex of all employees needs to be reversed. This can be done with one atomic update:

$$\&\{i,s,s'\}\mathcal{U}sex(i,s) \to (i,s') \textbf{ where } (s = m \wedge s' = f) \vee (s = f \wedge s' = m)$$

With only insert and delete actions and without an extra predicate symbol, we cannot give an update program that reverses the sex of all employees in the above example. We give an informal argumentation for this statement. There are basically two approaches to the problem: "relation at a time" updates and "tuple at a time updates":

- The "relations at a time update" does not work, because after inserting all new tuples, we no longer know which tuples are the original tuples and which tuples are the new tuples (so we do not know what tuples we need to delete).
- The "tuple at a time update" (using a conditional choice in combination with a while loop) does not work, because in a step of the while loop, we cannot distinguish between tuples that were already updated and tuples that still have to be updated (so we cannot prevent that a tuple is updated for the second time nor can we prevent that a tuple is not updated at all).

So if we have no extra predicate symbols to store temporary information, it is not always possible to replace an update action by an equivalent update program that uses only inserts and deletes.

The next two examples show that the number of elements in the extension of p before and after an atomic update action on p need not be the same.

Example 4.5. Consider the update action $\&\{x\}\ \mathcal{U}p(x) \to p(a)$. If we execute this update action in a world where $p(b)$ and $p(c)$ are true (with $b \neq c$), then after the update, only $p(a)$ is true, so the number of tuples in the relation p has decreased.

Example 4.6. Consider the update action $\&\{x\}\ \mathcal{U}p(a) \to p(x)$. If we execute this update action in a world where only $p(a)$ is true, then after the update, $\forall x\ p(x)$ holds, so the number of tuples in the relation p has increased (assuming that the domain contains at least two elements).

The reason for this non-standard behavior of the update actions in the above two examples, can be found in the way we used the variables. In the first example, we have used a variable in the updated tuple that does not occur in the tuple that is updated, and in the second we did something similar the other way around. We can forbid such update actions by some kind of safeness requirements, but there seems to be no practical reason for doing that. (Without restrictions, the semantics and axiomatization are still easily given. For instance, we can derive both $\vdash p(b) \wedge p(c) \to [\&\{x\}\mathcal{U}p(x) \to p(a)]\forall x\ (p(x) \leftrightarrow x = a)$ and $\vdash p(a) \to [\&\{x\}\ \mathcal{U}p(a) \to p(x)]\forall x\ p(x)$ in the proof system in Section 4.3.)

The atomic insertion, deletion and update are *conditional* and the reader may wonder why we do not also have a conditional assignment. Using a syntax similar to atomic insert delete and update, a conditional assignment would look like $\&X\, fT := t$ **where** ϕ. The next example shows that it is not immediately clear what the semantics of this action would be.

Example 4.7. Consider the conditional assignment action $\&\{x\}\ f(a) := x$. Intuitively, this action states that all domain values for x are simultaneously assigned to $f(a)$, which is absurd if the domain of x contains at least two distinct elements.

We can try to find some syntactical restrictions under which the conditional assignment always has an unambiguous semantics. A simple restriction seems to be to demand that in $\&X\ f(t_1,\ldots,t_n) := t$ **where** ϕ, the term t does not contain variables in X other than those occurring in $t_1,\ldots,t_n$. This restriction indeed excludes the preceding example, but the next example shows that this condition is not strong enough.

Example 4.8. Suppose we have the sort *bit*, non-updateable function $g:\langle bit\rangle \to bit$ and updateable function $f:\langle bit\rangle \to bit$. Furthermore, assume that we have an algebra $\mathscr{A}$ such that $bit^{\mathscr{A}} = \{0,1\}$, $g^{\mathscr{A}}(0) = 0$ and $g^{\mathscr{A}}(1) = 0$. Now consider the update $\&\{x\}\ f(g(x)) := x$. Intuitively, the update states that both 0 and 1 are assigned to $f(0)$, which is of course absurd.

To also exclude this example, we can make the stronger restriction that in the update $\&X\ f(t_1,\ldots,t_n) := t$ **where** ϕ, the terms $t_1,\ldots,t_n$ are variables and t does not contain variables in X other than $t_1,\ldots,t_n$. This restriction makes sure that if we have a tuple of domain values for (the variables) $t_1,\ldots,t_n$, then there is only one corresponding value for t, so there can be no ambiguity. However, the restriction is very strong and we have not elaborated this idea.

Instead of trying to find syntactical restrictions, we can also keep the general conditional assignment action and just define its semantics to be equal to failure (no successor worlds) if the result of the conditional choice is ambiguous (as in the above examples). The declarative semantics then is relatively easy to define. However, finding axioms for this action presents a serious problem. We encounter the problem of the axiomatization of successor existence: under what conditions does the parallel assignment action succeed. We have found no solution for this problem and therefore exclude conditional assignment.

Before we define the semantics of DDL we note in passing that RAUL updates can be expressed in terms of DDL updates. We show this by giving an example for every RAUL operator. Suppose we have predicate symbols $p_1:\langle s,s\rangle$, $p_2:\langle s,s\rangle$, $q:\langle s,s\rangle$ and $r:\langle s,s,s,s\rangle$. Fig. 6 illustrates the translation. It can be easily seen that using "temporary" predicate symbols, we can translate every RAUL update to DDL by evaluation the relational operations in the RAUL update inside out, one operator at a time.

Union	$q := p_1 \cup p_2$	$\&x, y \;\; Dq(x, y)$; $\&x, y \;\; Iq(x, y)$ **where** $p_1(x, y) \vee p_2(x, y)$
Difference	$q := p_1 \setminus p_2$	$\&x, y \;\; Dq(x, y)$; $\&x, yIq(x, y)$ **where** $p_1(x, y)$; $\&x, yDq(x, y)$ **where** $\neg p_2(x, y)$
Product	$r := p_1 \times p_2$	$\&x_1, y_1, x_2, y_2 \;\; Dr(x_1, y_1, x_2, y_2)$; $\&x_1, y_1, x_2, y_2 \;\; Ir(x_1, y_1, x_2, y_2)$ **where** $p_1(x_1, y_1) \wedge p_2(x_2, y_2)$
Projection	$q := r[1, 3]$	$\&x_1, x_2 \;\; Dq(x_1, x_2)$; $\&x_1, x_2 \;\; Iq(x_1, x_2)$ **where** $\exists y_1, y_2 \;\; r(x_1, y_1, x_2, y_2)$
Selection	$r := p_1$ **where** $1 = 2$	$\&x, y \;\; Dr(x, y)$; $Ir(x, y)$ **where** $x = y$

Fig. 6. Translation of RAUL into DDL.

4.2. Declarative semantics of DDL

We need to define the function m_{S,I_V} for the constructions introduced in Definition 4.1. This function formally defines the intuitive semantics described in the previous section. In the next definition, *USym* is the set of updateable symbols $UFun \cup UPred$.

Definition 4.9. Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, and let I_V be a variable interpretation on $\mathscr{A}$. Then the defining clauses for m_{S,I_V} of the atomic insert, delete and update, assignment and conditional choice are:

$$m_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi) = \{(w, w') \in S^2 \mid w' \restriction_{USym\setminus\{p\}} = w \restriction_{USym\setminus\{p\}} \text{ and } w'(p) = w(p) \cup \{T^{\mathscr{A},w,I'_V} \mid I'_V \restriction_{Var\setminus X} = I_V \restriction_{Var\setminus X} \text{ and } S, w, I'_V \models \phi\}\}$$

$$m_{I_V}(\&X\mathscr{D}pT \textbf{ where } \phi) = \{(w, w') \in S^2 \mid w' \restriction_{USym\setminus\{p\}} = w \restriction_{USym\setminus\{p\}} \text{ and } w'(p) = w(p) \setminus \{T^{\mathscr{A},w,I'_V} \mid I'_V \restriction_{Var\setminus X} = I_V \restriction_{Var\setminus X} \text{ and } S, w, I'_V \models \phi\}\}$$

$$m_{I_V}(\&X\mathscr{U}pT \to T' \textbf{ where } \phi) = \{(w, w') \in S^2 \mid w' \restriction_{USym\setminus\{p\}} = w \restriction_{USym\setminus\{p\}} \text{ and } w'(p) = (w(p) \setminus \{T^{\mathscr{A},w,I'_V} \mid I'_V \restriction_{Var\setminus X} = I_V \restriction_{Var\setminus X} \text{ and } S, w, I'_V \models \phi\}) \cup \{T'^{\mathscr{A},w,I'_V} \mid I'_V \restriction_{Var\setminus X} = I'_V \restriction_{Var\setminus X} \text{ and } S, w, I'_V \models \phi \wedge pT\}\}$$

$$m_{I_V}(fT := t) = \{(w, w') \in S^2 \mid w' \restriction_{USym\setminus\{f\}} = w \restriction_{USym\setminus\{f\}} \text{ and } w'(f) = w(f)\{T^{\mathscr{A},w,V} \mapsto t^{\mathscr{A},w,V}\}\}$$

$$m_{I_V}(+X \;\; \alpha \textbf{ where } \phi)(I_V) = \{(w, w') \in S^2 \mid \exists I'_V : I'_V \restriction_{Var\setminus X} = I_V \restriction_{Var\setminus X} \text{ and } S, w, I'_V \models \phi \text{ and } (w, w') \in mi_{I'_V}(\alpha)\}$$

The intended meaning of $\&X\mathscr{I}pT$ **where** ϕ is to insert the tuples T in p for all values for the variables in X that make the formula ϕ true. This is reflected as follows in the formal semantics:

- The condition $w'(p) = w(p) \cup \{T^{\mathscr{A},w,I'_V} \mid I'_V \restriction_{Var\setminus X} = I_V \restriction_{Var\setminus X}$ and $S,w,I'_V \models \phi\}$ states that the interpretation of p after the update (in world w') contains all tuples present before the update (in world w) together with all tuples $T^{\mathscr{A},w,I'_V}$ for I'_V a variable interpretation that interprets the variables in X in some way that makes ϕ true.
- The condition $w' \restriction_{USym\setminus\{p\}} = w \restriction_{USym\setminus\{p\}}$ is the frame assumption: the interpretation of all other updateable symbols remains the same.

The formal semantics of the atomic delete, atomic update and assignment can be understood in a similar way.

The intended meaning of $+X\ \alpha$ **where** ϕ is to execute α for one of the variable assignments to variables in X that makes ϕ true. Formally, this means that all worlds that result from executing α for some variable interpretation for X that makes ϕ true, are successor worlds.

4.3. Axiomatization of DDL

Truth in the full structures is axiomatized by extending the axioms in Fig. 2 with those in Fig. 7. Axiom (PosIns) states that if pT' holds before the insertion or if tuple T' is inserted, then pT' holds after the insertion. Similarly, axiom (NegIns) states that if pT' does not hold before the insertion and pT' is not inserted, then pT' does not hold after the insertion. Note that we cannot replace (PosIns) and (NegIns) by the single equivalence axiom (Ins) $pT' \vee \exists X(\phi \wedge T = T') \leftrightarrow [\&X\mathscr{I}pT$ **where** $\phi]pT'$, because with only this axiom, we cannot derive (NegIns). However, if we use axiom (Ins) in combination with the determinism axiom $\langle \&X\mathscr{I}pT$ **where** $\phi\rangle pT' \rightarrow [\&X\mathscr{I}pT$ **where** $\phi]pT'$, then we can derive axiom (NegIns). (This gives us an alternative axiomatization of atomic insertion.) The requirement $FV(T') \cap X = \emptyset$ of axiom (PosIns) is explained as follows: If T' would contain free variables that occur in X, then within the $\exists X$ quantification of (PosIns), these variables would be bound whereas in the other two occurrences of T', these variables would be free. It would then be easy to construct a counterexample to the axiom.

Axiom (PosDel) states that if pT' holds before the deletion and if tuple T' is not deleted, then pT' holds after the deletion. Similarly, axiom (NegDel) states that if pT' does not hold before the update or if tuple T' is deleted, then pT' does not hold after the deletion. Just like for insertion, (PosDel) and (NegDel) can be replaced by an equivalence axiom and a determinism axiom; this yields an alternative axiomatization for atomic deletion.

Axiom (PosUpd) states that pT_1 is true after the update if either pT_1 is true before the update and the tuple T_1 is not changed by the update, or there is some tuple in p that is changed to T_1 by the update. Axiom (NegUpd) states that if the same

(PosIns) $pT' \vee \exists X(\phi \wedge T{=}T') \rightarrow [\&X\ \mathcal{I}p\,T\ \mathbf{where}\ \phi]pT'$, for $FV(T') \cap X = \emptyset$
(NegIns) $\neg(pT' \vee \exists X(\phi \wedge T{=}T')) \rightarrow [\&X\ \mathcal{I}p\,T\ \mathbf{where}\ \phi]\neg pT'$, for $FV(T') \cap X = \emptyset$

(PosDel) $pT' \wedge \neg\exists X(\phi \wedge T{=}T') \rightarrow [\&X\ \mathcal{D}p\,T\ \mathbf{where}\ \phi]pT'$, for $FV(T') \cap X = \emptyset$
(NegDel) $\neg(pT' \wedge \neg\exists X(\phi \wedge T{=}T')) \rightarrow [\&X\ \mathcal{D}p\,T\ \mathbf{where}\ \phi]\neg pT'$, for $FV(T') \cap X = \emptyset$

(PosUpd) $(pT_1 \wedge \neg\exists X(\phi \wedge T{=}T_1)) \vee \exists X(pT \wedge \phi \wedge T' = T_1) \rightarrow$
$[\&X\ \mathcal{U}pT \rightarrow T'\ \mathbf{where}\ \phi]pT_1$, for $FV(T_1) \cap X = \emptyset$
(NegUpd) $(\neg pT_1 \vee \exists X(\phi \wedge T{=}T_1)) \wedge \neg\exists X(pT \wedge \phi \wedge T' = T_1) \rightarrow$
$[\&X\ \mathcal{U}pT \rightarrow T'\ \mathbf{where}\ \phi]\neg pT_1$, for $FV(T_1) \cap X = \emptyset$

(Assign) $[fT := t]fT = t$, for $f \notin T, t$

(FrInsUPred1) $q\,T' \rightarrow [\&X\ \mathcal{I}p\,T\ \mathbf{where}\ \phi]q\,T'$, for $q \neq p \in UPred$
(FrInsUPred2) $\neg q\,T' \rightarrow [\&X\ \mathcal{I}p\,T\ \mathbf{where}\ \phi]\neg q\,T'$, ""
(FrDelUPred1) $q\,T' \rightarrow [\&X\ \mathcal{D}p\,T\ \mathbf{where}\ \phi]q\,T'$, ""
(FrDelUPred2) $\neg q\,T' \rightarrow [\&X\ \mathcal{D}p\,T\ \mathbf{where}\ \phi]\neg q\,T'$, ""
(FrUpdUPred1) $q\,T' \rightarrow [\&X\ \mathcal{U}pT \rightarrow T'\ \mathbf{where}\ \phi]q\,T'$, ""
(FrUpdUPred2) $\neg q\,T' \rightarrow [\&X\ \mathcal{U}pT \rightarrow T'\ \mathbf{where}\ \phi]\neg q\,T'$, ""
(FrAssUPred1) $pT' \rightarrow [fT := t]pT'$, for $p \in UPred$, $f \notin T'$
(FrAssUPred2) $\neg pT' \rightarrow [fT := t]\neg pT'$, ""
(FrInsUFun) $fT_1 = t \rightarrow [\&X\ \mathcal{I}p\,T\ \mathbf{where}\ \phi]fT_1 = t$, for $f \in UFun$
(FrDelUFun) $fT_1 = t \rightarrow [\&X\ \mathcal{D}p\,T\ \mathbf{where}\ \phi]fT_1 = t$, ""
(FrUpdUFun) $fT_1 = t \rightarrow [\&X\ \mathcal{U}pT \rightarrow T'\ \mathbf{where}\ \phi]fT_1 = t$, ""
(FrAssUFun1) $gT' = t' \rightarrow [fT := t]gT' = t'$, for $g \neq f \in UFun$, $f \notin T', t'$
(FrAssUFun2) $(fT' = t' \wedge T \neq T') \rightarrow [fT := t]fT' = t'$, for $f \notin T', t'$

(SuccIns) $\langle\&X\ \mathcal{I}p\,T\ \mathbf{where}\ \phi\rangle true$
(SuccDel) $\langle\&X\ \mathcal{D}p\,T\ \mathbf{where}\ \phi\rangle true$
(SuccUpd) $\langle\&X\ \mathcal{I}p\,T\ \mathbf{where}\ \phi\rangle true$
(SuccAss) $\langle fT := t\rangle true$

(CondC) $[+X\ \alpha\ \mathbf{where}\ \phi]\psi \leftrightarrow \forall X(\phi \rightarrow [\alpha]\psi)$,
for $FV(\psi) \cap X = \emptyset$

Fig. 7. Axioms for the atomic actions in DDL. These must be added to the axioms of regular first-order update logic.

precondition does not hold, then pT_1 does not hold after the update. Again, (PosUpd) and (NegUpd) can be replaced by an equivalence axiom and a determinism axiom; this yields an alternative axiomatization for atomic update.

All axioms of which the name starts with "Fr" are the frame axioms. Note that for the non-updateable symbols, we already have the axioms (FrNUPred1), (FrNUPred2) and (FrNUTerm) of Section 2.

The successor existence axioms (SuccIns), (SuccDel), (SuccUpd) and (SuccAss) just state that atomic updates and assignments always succeed. These axioms are sound, because we axiomatize truth in the full structures, they are (of course) not sound in all structures.

Axiom (CondC) states that all ways of executing the update $+X\ \alpha$ **where** ϕ lead to ψ iff for all values for the variables in X that make ϕ true, any execution of α makes ψ true. If ψ would contain free variables that occur in X, then left of the $\leftrightarrow$, these free variables would not be bound by $+X$, but right of the $\leftrightarrow$, they would be bound by $\forall X$. It would then be easy to construct a counterexample to the axiom, therefore we have the requirement $FV(\psi) \cap X = \emptyset$.

The condition that the updated function symbol f may not occur in T and t in axiom (Assign), in T' in axioms (FrAssUPred1) and (FrAssUPred2) and not in T' and t' in axioms (FrAssUFun1) and (FrAssUFun2) is essential. We illustrate this with one simple example for axiom (FrAssUPred1) (similar examples can be given for the other four axioms).

Example 4.10. Suppose f may occur in T' in axiom (FrAssUPred1). Then an instance of (FrAssUPred1) would be $p(f(a)) \rightarrow [f(a) := b]p(f(a))$. Now, suppose that $p(f(a))$ is true before the update $f(a) := b$ and $p(b)$ is false before the update (so we also know $f(a) \neq b$ before the update). Then $p(f(a))$ should be false after the update, because $p(b)$ is false before the update and we do not want the interpretation of the predicate symbol p to change under the update. But with the "axiom" we can derive that $p(f(a))$ is true after the update.

First-order dynamic logic contains assignments of values to variables as atomic actions. This assignment action is axiomatized by the following well-known axiom [21, 26]:

$$[x := t]\phi \leftrightarrow \phi[t/x].$$

By analogy, we might expect an axiom

$$[fT := t]\phi \leftrightarrow \phi[t/fT]$$

instead of the axioms for assignment that we have given. Unfortunately, this "axiom" is not sound.

Example 4.11. Consider the formula (which is an instance of the above "axiom")

$$[f(a) := b]f(c) = d \leftrightarrow f(c) = d.$$

Furthermore, assume that a is equal to c but b is not equal to d. This formula is clearly false in a world where $f(a) = d$, because after the update $f(c)$ is no longer equal to d, but to b.

The reason why the "axiom" is not sound is, that it uses a syntactic substitution operator, where it should use a semantic substitution operator. In order to define such a substitution operator, we need to make some assumptions on the interpretation of the non-updateable function symbols (such that we have unique "normal forms" of terms and that the semantics respects this). The axiomatization we have given above is preferable, as we do not need extra assumptions. We also prefer the axiomatization we have given above because it is similar in style to the axiomatization of the other atomic update operators.

4.4. Soundness and completeness of the proof system

Soundness of the proof system with respect to full structures is proven in Appendix A.3. To prove completeness, we can try to apply the standard Henkin

completeness proof technique for modal logic. The Henkin completeness proof consists mainly of constructing a canonical model. To make sure that every consistent formula is satisfiable in the canonical structure (an essential step in the Henkin completeness proof), the interpretation of the modal operators is chosen in a very specific way. In the case of DDL, we encounter a problem here. In update logic, we cannot *choose* the interpretation of the modal operators (updates), simply because the semantics of the updates is determined by the interpretation of the function and predicate symbols in the possible worlds. This problem is essentially the same as in the propositional case and is explained in detail in [35].

For *propositional* dynamic database logic, we were able to prove completeness with a new technique based on *complete information formulas* (cifs) [35]. A cif is just a finite conjunction of literals. When we have a complete information formula γ *over* a formula ϕ (this means that γ contains all propositional atoms occurring in ϕ), then we can prove the central proposition that $\vdash \gamma \to \phi$ or $\vdash \gamma \to \neg\phi$. So in a way, a complete information formula resembles a possible world in the sense that it contains sufficient information to evaluate a formula to true or false. The cif technique cannot be applied directly to DDL. The reason is that to define complete information formula for DDL, we would in general need infinite conjunctions (even for one predicate symbol, there may be infinitely many closed instances), so we would need infinitary logic. Finding a completeness proof using infinitary logic might be possible, or even a "trick" to keep the complete information formulas finite might be found. But this surely will be far from easy and we have not found a completeness proof along such lines yet. Completeness of DDL in the general case is therefore still an open question.

Important from the (relational) database perspective are Reiter's *domain closure* and *unique naming* assumptions [32]. These assumptions, respectively, state that all domain elements are named, and that different names denote a different domain element. To be able to formulate these assumptions, Reiter [32] assumes that the only function symbols are a finite set of constants $\{c_1, \ldots, c_n\}$. Domain closure is then the axiom $\forall x(x = c_1 \vee \cdots \vee x = c_n)$ and unique naming is the set of axioms $c_i \neq c_j$, for $i \neq j$. Under these assumptions, the domains are finite and all domain elements are named, so it is possible to define complete information formulas (as conjunctions of atoms), and the number of complete information formulas "over" a formula is finite. Including the domain closure and unique naming axioms, we can use the cif technique and prove completeness of the axiomatization (with respect to truth in full structures that satisfy the domain closure and unique naming axioms) in essentially the same way as for propositional dynamic database logic. For details of this technique, we refer to [35].

Next to the general question of completeness of DDL with respect to truth of formulas in full structures over arbitrary algebras, at least as important is the question of completeness of the axiomatization with respect to truth of formulas in the full structure over an intended algebra of the non-updateable function and predicate symbols. If the intended algebra is an initial algebra of an equational specification, then completeness is in general impossible. The simple reason is that, with the initial algebra semantics, we can specify a structure that is isomorphic to the natural numbers, but the set of all

true numerical equations is not recursively enumerable (so not axiomatizable). Details are given by Nourani [30, Section 6].

Another interesting completeness problem is the following. Assume the initial algebra semantics and use all true formulas in the initial algebra of some equational specification, instead of only the formulas that are true in all algebras of the equational specification, i.e. the formulas that are derivable from the equational specification. Can we then prove completeness of the axiomatization? This is a relative completeness question; assuming we have an "oracle" for the formulas true in the initial algebra, do we have completeness of the whole axiomatization. Just as the standard completeness problem, this question is still open.

4.5. *Update completeness of* DDL

If we have a language to update databases, then a natural question to ask is what the expressive power of this language is. Ideally, we would like some completeness result: every "reasonable" function from database states to database states can be expressed in the update language. Abiteboul and Vianu [1, 2] define such a notion of completeness of update languages for *relational* databases and show that the specific update language TL is update complete. As a relational database is a special case of a logic database, it is not too difficult to translate their definitions of database schema and instance to DDL. This section contains such a translation. (As this section is mainly concerned with relational databases, we will frequently use the word "relation" instead of the word "predicate".) The basic result of this section is that the update language of DDL is (relational) update complete.

For the translation, we restrict the semantics to one special structure $\mathscr{R}$, the *relational structure*. This structure is based on the closed term model (Con, id_{Con}) and has as worlds all functions that interpret all predicate symbols as a *finite* sets of tuples (of constants). (Note that we do not consider the problem of axiomatizing this logic here, but some infinitary axiomatization seems to be the most natural.)

Definition 4.12. A *database schema* is a finite subset of the predicate symbols. An *instance* of a relational database schema S is the restriction of some element of $\mathscr{R}$ to S (note that elements of $\mathscr{R}$ are indeed functions with the domain being the predicate symbols). The set of instances of a database schema S is denoted $Inst(S)$.

The notion of update completeness is based on the notion of C-genericity. Intuitively, a function on instances is C-generic, if it treats all constants "uniformly", except maybe for the constants of C (the constants that occur in the update).

Definition 4.13. Let S and T be database schemas. Let $C \subseteq Con$ be a finite set of constants. A function r from $Inst(S)$ to $Inst(T)$ is *C-generic* iff for every bijective function ρ on Con which is the identity on elements of C, we have that $r \circ \rho = \rho \circ r$. (Here, we use ρ as a function on instances: just change all the constants in the instance in the way indicated by ρ.)

Definition 4.14. Let S and T be database schemas. A *database mapping* is a function from $Inst(S)$ to $Inst(T)$ which is computable and C-generic, for some finite set of constants C.

In general, database updates may need relations to store temporary results. As such relations are not important in the final result, we have the notion of i–o schemas (S,T) (where S and T are database schemas). An update over an i–o schema (S,T) uses the relations of S to compute the relations of T and may also use relations not in S or T to store temporary results. All relations used in the update that are not in S are assumed to be initially empty, all relations not in T do not count in the final result.

Definition 4.15. An update language is *update complete*, if for every database mapping from $Inst(S)$ to $Inst(R)$, there is an update over the i–o schema (R,S) such that the semantics of this update is exactly the database mapping.

Abiteboul and Vianu [1] give a specific language called transaction language (TL) that is update complete. This language contains

- atomic insert and delete actions: $i_P(r)$ and $d_P(r)$, for P a relation and r a tuple;
- sequential composition: if t and t' are TL expressions, then $t;t'$ is a TL expression;
- a while construction: if Q is a conjunction of literals (including possibly the $=$ predicate) and t is a TL expression, then **while** Q **do** t is a TL expression.

Intuitively, atomic inserts and deletes just insert/delete one tuple (or do nothing if the tuple is already (not) present), sequential composition just executes the two updates, one after the other, and **while** Q **do** t repeatedly picks values for the variables in Q that make Q true and executes t with these values, until there exists no value assignment to the variables of Q that makes Q true.

The formal semantics of a TL expression is given (just as for DDL update programs) as a relation on the possible worlds of $\mathscr{R}$. The easiest way of defining the formal semantics of TL expressions, is by giving equivalent DDL update programs for the TL expressions: $i_P(r)=\mathscr{I}P(r)$, $d_P(r)=\mathscr{D}P(r)$, $t;t'=t;t'$ and **while** Q **do** $t=(+FV(Q)\ t$ **where** $Q)^*;(\neg\exists FV(Q)Q)?$. As we are able to define the semantics of TL expressions in terms of DDL update programs, we immediately get from the update completeness of TL, that the DDL update language is update complete.

4.6. Operational semantics of DDL

Using the definite database states as defined in Definition 2.23, we give an operational semantics for the update language of DDL. The transition rules for the regular operators are already given in Section 2.5 (Fig. 3). Here, we give the transition rules for the atomic updates, assignment, conditional choice and test. To define these transition rules, we need to make some restrictions on the format of the formulas that are allowed in the condition part of atomic updates and conditional choice, and in tests. Just as for the choice of the format of formulas in database states, we again have various options.

A first reasonable restriction is not to allow modalities in conditions and tests, because modal formulas state properties of possible future states, whereas we should only query the current state. We also need to restrict quantification in some way, as we do not want to allow quantification over an infinite set of objects. In general, atomic updates contain two kinds of quantification: universal quantification over the set of variables X of the update and universal and existential quantification in the condition formula. For simplicity, we do not allow quantification in the condition formula at all. As atomic updates on predicate symbols must turn definite database states into definite database states, they may only insert a finite amount of information. We therefore introduce a safeness concept for conditions used in atomic updates.

Definition 4.16. Let X be a finite set of variables. An *X-safe* formula has the form

$$p_1T_1 \wedge \cdots \wedge p_nT_n \wedge \phi$$

for $p_1, \ldots, p_n$ updateable predicate symbols, $T_1, \ldots, T_n$ tuples of terms (of the correct arities) and ϕ a first-order formula not containing quantification, such that all variables in X occur at least once as term but not as subterm of one of the tuples $T_1, \ldots, T_n$.

Note that in the above definition, the variables of X must occur as entire components in a tuple and not just somewhere in some term that is a component of a tuple. The next example shows why we need this requirement.

Example 4.17. Suppose we have a non-updateable function $zero : \langle nat \rangle \rightarrow nat$ such that $zero(x)$ always yields 0 and suppose we have two unary updateable predicate symbols p and q over the natural numbers. Then in a state where $q(0)$ is true, the update $\&\{x\}\ \mathcal{I}p(x)$ **where** $q(zero(x))$ inserts $p(x)$ for all natural numbers x, which would result in a non-definite database state. (And therefore according to the above definition, $q(zero(x))$ is not an $\{x\}$-safe formula.)

There can only be a finite number of value assignments to the variables in X that make an X-safe formula $p_1T_1 \wedge \cdots \wedge p_nT_n \wedge \phi$ true in a definite database state. The reason is, that every variable in X occurs as a component of some T_i (say as the jth component) and as p_i only contains finitely many tuples in a definite database state, the projection of p_i on the jth element gives a finite set.

In the condition of the transition rules for atomic updates, we use the operational semantics of RAUL. Given a closed atomic update $\&X\mathcal{I}pT$ **where** ϕ, the main problem is to compute all X-assignments that make ϕ true. Given these X-assignments, then by substitution of the assignments to the variables in T we can easily find the set of tuples to insert, delete or update.

Definition 4.18. Let X be a set of variables. If ϕ is a formula that only contains free variables in X, then we call ϕ *X-closed*. Similarly, if t is a term that only contains variables in X, then t is called *X-closed*.

Intuitively, X-closed means "closed except for variables in X". By taking the empty set for X, the notion X-closed collapses to the familiar notion of closed as having no free variables at all.

Definition 4.19. Let $x_1,\ldots,x_{[\![k]\!]}$ be a sequence of distinct variables, let $X=\{x_1,\ldots,x_{[\![k]\!]}\}$ and let $\phi=p_1T_1\wedge\cdots\wedge p_nT_n\wedge\psi$ be an X-safe, X-closed formula. For a variable $x\in X$, we use the notation p_x for the first predicate symbol in the sequence $p_1,\ldots,p_n$ for which x occurs as a component of its arguments in ϕ and we use the notation k_x for the argument position of x (if x occurs more than once as a component, then we just take the first occurrence). We then define $ca(X,\phi)$, the *computed assignments* to variables of X that make ϕ true, as the relational algebra expression:

$$p_{x_1}[k_{x_1}]\times\cdots\times p_{x_n}[k_{x_n}]\ \textbf{where}\ \phi[1/x_1,\ldots,k/x_{[\![k]\!]}]$$

In the above definition, some ordering on the variables was needed to formulate the computation in relational algebra. But this order is not relevant; if we take a permutation of the variables, then the result is the same permutation of the old result.

As definite database states do not contain information on variables, the update programs that are used in configurations are always closed.

For the assignments, we just introduce a bit of notation. This is nothing more than an explicit formulation of function variants for representations of functions that are used in database states.

Definition 4.20. Let $(t,\{(T_1,t_1),\ldots,(T_n,t_n)\})$ be the representation of a function f, then

$$f\{T'\mapsto t'\}=\begin{cases}(t,\{(T_1,t_1),\ldots,(T_{i-1},t_{i-1}),(T',t'),\\ \quad(T_{i+1},t_{i+1}),\ldots,(T_n,t_n)\}) & \text{if } T_i=T',\\ (t,\{(T_1,t_1),\ldots,(T_n,t_n),(T',t')\}) & \text{if } \forall i(1\leqslant i\leqslant n):\ T_i\neq T'.\end{cases}$$

As for $i\neq j$ we cannot have $T_i=T_j$, the result is always well defined. We could make the result "shorter" if $t'=t$, because then the function result would not have to be listed as an explicit exception. For simplicity, we have not made this optimization.

The transition rules for the atomic updates and assignment in DDL are given in Table 4. The function *mc* used in the conclusion of these rules computes the value of a closed term in normal form ("make constant") and has been defined in Definition 2.25. Although the transition rules look horrendous, most of it is bookkeeping. For example, the rule (Tins) defines a predicate q whose extension contains exactly the tuples that make ϕ true (the computed assignments of Definition 4.19). The conclusion of this rule then says that the extension of p can be extended with all the tuples to which T is mapped by these assignments. The rule for deletion is similar to the rule for insertion and the rule for updates just combines a deletion and an insertion. Note that to instantiate the rules for atomic updates, we need a predicate

Table 4
Transition rules for atomic update and assignment in DDL

(TIns)	$\dfrac{\langle DB, q := ca(\bar{x}, \phi)\rangle \to DB'}{\langle DB, \&X\,\mathscr{I}pT \textbf{ where } \phi\rangle \to}$ $DB\{p \mapsto DB(p) \cup \{mc(T[\bar{t}/\bar{x}], DB) \mid \bar{t} \in DB'(q)\}\}$
(TDel)	$\dfrac{\langle DB, q := ca(\bar{x}, \phi)\rangle \to DB'}{\langle DB, \&X\,\mathscr{D}pT\phi \textbf{ where } \rangle \to}$ $DB\{p \mapsto DB(p) \setminus \{mc(T[\bar{t}/\bar{x}], DB) \mid \bar{t} \in DB'(q)\}\}$
(TUpd)	$\dfrac{\langle DB, q := ca(\bar{x}, \phi)\rangle \to DB'}{\langle DB, \&X\,\mathscr{U}pT \to T' \textbf{ where } \phi\rangle \to}$ $DB\{p \mapsto (DB(p) \setminus \{mc(T[\bar{t}/\bar{x}], DB) \mid \bar{t} \in DB'(q)\}) \cup \{mc(T'[\bar{t}/\bar{x}], DB) \mid \bar{t} \in DB'(q)\}$
(TAss)	$\langle DB, fT := t\rangle \to DB\{f \mapsto DB(f)\{mc(T, DB) \mapsto mc(t, DB)\}\}$
	for (TIns), (TDel), and (TUpd): $X = \{x_1, \ldots, x_n\}$ and $\bar{x} = (x_1, \ldots, x_n)$ and ϕ is X-safe and X-closed

symbol q of the right arity. The situation is similar as for the axiomatization; there we also needed predicate symbols to store "temporary" information.

We now turn our attention to the transition rules for the conditional choice and test. For the formulas in the condition of conditional choices we just use the same restrictions as for formulas used in the conditions of atomic updates (so we assume these formulas are safe). For the tests, we assume they have the format $\forall X(\phi \to \psi)$, with ϕ an X-safe formula and ψ a first-order formula not containing modalities. Again, only finitely many assignments to X can make ϕ true (and we can compute these assignments), so we only have to evaluate ψ for finitely many assignments. Note that the choice of format of the formulas for tests is rather arbitrary, but because we already have the notion of safe formulas and we want to be able to evaluate tests in finite time, it seems to be a useful format.

With the conditional choice rule, we can introduce variables. The next two definitions are used for this purpose.

Definition 4.21. Let X be a finite set of variables. An *X-assignment* is a function with domain X and range the set of immutable terms.

Definition 4.22. Let A be an X-assignment. For a formula ϕ, $A(\phi)$ is the formula ϕ with all free occurrences of variables x in ϕ that are in X replaced by $A(x)$. In the same way, we can define $A(\alpha)$ and $A(t)$ for an update program α and term t.

Note that we can define $A(\phi)$, $A(\alpha)$ and $A(t)$ formally by simultaneous induction on the structure of formulas and update programs, but this definition is completely straightforward and has therefore been omitted. Of course, when A is an X-assignment

Table 5
The transition rule for conditional choice in DDL

(TCondC)	$\langle DB, +X\ \alpha\ \textbf{where}\ \phi\rangle \rightarrow \langle DB, A(\phi?;\alpha)\rangle$ (for A an X-assignment)

and ϕ is X-closed, then $A(\phi)$ is a closed formula (and similarly for update programs and terms).

The transition rule for the conditional choice in DDL is given in Table 5.

For the conditional choice rule, we note that intuitively, there is a difference between choosing values for the variables in X that satisfy ϕ and executing α for these values, and choosing values for the variables in X and then if ϕ holds for these values, executing α (in the second case, we may get a failing execution). But as we are only interested in executions that succeed (so we look for database states DB' such that $\langle DB, \alpha\rangle \rightarrow DB'$), the transition rule is sound.

If a database state would contain indefinite information, then tests could reduce the amount of "indefiniteness" by ruling out (some of the) uncertainties. As we only consider definite database states, a test just either fails or succeeds (and in the case of success, the database state remains unchanged).

4.7. Example correctness proof in DDL

In this section we give a detailed example, showing how we can use the proof system of DDL to prove correctness of an implementation of an update program, given a specification for it.

Consider a "copy" update program, which copies the extension of the unary predicate symbol q to p. The specification of the copy program α consists of two formulas:

1. $\forall y(q(y) \leftrightarrow [\alpha]q(y))$, the extension of q remains unchanged under the update.
2. $[\alpha]\forall y(p(y) \leftrightarrow q(y))$, after the update, p has the same extension as q.

(Note that we assume that the only updateable symbols are p and q, otherwise we should also specify that none of the other predicate symbols is changed by the update.) We prove that the following implementation of the update program satisfies the specification:

$$\alpha = \&\{x\}\mathscr{D}\,p(x); \&\{x\}\mathscr{I}\,p(x)\ \textbf{where}\ q(x).$$

We must prove $\vdash \forall y(q(y) \leftrightarrow [\alpha]q(y))$ and $\vdash [\alpha]\forall y(p(y) \leftrightarrow q(y))$. Proving this using only the axioms and derivation rules given in the proof system of Section 4.3 is extremely tedious. We therefore use a number of abbreviations in constructing the proofs:

- When we need a first-order theorem, we just write it down without proof. When we can derive a formula from another formula in a few steps, using only first-order properties of $\exists$ and $\forall$, then we do not give all steps of the derivation, but we just say the second formula follows from the first by "first-order reasoning".

- When we can derive a formula from another formula in a few steps, only using the properties of the propositional connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$ and $\leftrightarrow$, then we do not give all steps involved in this derivation, but we just say the second formula follows from the first by "propositional reasoning". For some specific cases of propositional reasoning, we use special names. Below, we use: transitivity of $\leftrightarrow$, symmetry of $\leftrightarrow$, transitivity of $\rightarrow$, definition of $\rightarrow$ (in terms of $\neg$ and $\vee$) and definition of $\leftrightarrow$ (in terms of $\rightarrow$ and $\wedge$).
- When we can derive a formula from another formula, using only standard properties of (propositional) modal logic (so using the K axiom and modal generalization), then we do not give all steps of the derivation, but we just say the second formula follows from the first by "modal reasoning". One special case of modal reasoning is the derivation rule monotonicity of [ ] (from $\vdash \phi \rightarrow \psi$ derive $\vdash [\beta]\phi \rightarrow [\beta]\psi$ and from $\vdash \phi \leftrightarrow \psi$ derive $\vdash [\beta]\phi \leftrightarrow [\beta]\psi$).
- When we can derive a formula from another formula in a few steps, using one of the specific dynamic logic axioms, then we say the second formula follows from the first by "dynamic reasoning (no. of axiom)". Dynamic logic reasoning is usually used to replace a (sub)formula by an equivalent formula. We also use one specific inference rule, called the sequential composition rule (from $\vdash \phi \rightarrow [\beta_1]\phi'$ and $\vdash \phi' \rightarrow [\beta_2]\phi''$ derive $\vdash \phi \rightarrow [\beta_1;\beta_2]\phi''$ with as special case for $\phi = true$: from $\vdash [\beta_1]\phi'$ and $\vdash \phi' \rightarrow [\beta_2]\phi''$ derive $\vdash [\beta_1;\beta_2]\phi''$).
- From the frame axioms, it is easy to derive the frame theorems, which are the same as the axioms, only having bi-implications instead of implications (so $qT' \leftrightarrow [\&X\mathscr{I}pT$ **where** $\phi]qT'$, etc.).

There are, of course, many other interesting derivable inference rules and theorems than the ones listed above, we have only given the rules and theorems that we will use in the proof below.

We now prove that the implementation of α satisfies the specification. In the proof, we use $\alpha_1 = \&\{x\}\ \mathscr{D}p(x)$ and $\alpha_2 = \&\{x\}\mathscr{I}p(x)$ **where** $q(x)$ as abbreviations (so $\alpha = \alpha_1;\alpha_2$).

First, we prove $\vdash \forall y(q(y) \leftrightarrow [\alpha_1;\alpha_2]q(y))$:

1. $q(y) \leftrightarrow [\alpha_2]q(y)$ frame theorem
2. $[\alpha_1]q(y) \leftrightarrow [\alpha_1][\alpha_2]q(y)$ monotonicity of [ ], 1
3. $q(y) \leftrightarrow [\alpha_1]q(y)$ frame theorem
4. $q(y) \leftrightarrow [\alpha_1][\alpha_2]q(y)$ transitivity of $\leftrightarrow$, 3, 2
5. $q(y) \leftrightarrow [\alpha_1;\alpha_2]q(y)$ (Seq), 4
6. $\forall y(q(y) \leftrightarrow [\alpha_1;\alpha_2]q(y))$ (UnGen), 5

Next, we prove $\vdash [\alpha_1;\alpha_2]\forall y(p(y) \leftrightarrow q(y))$:

1. $\neg(p(y) \wedge \neg\exists x(true \wedge x = y)) \rightarrow [\alpha_1]\neg p(y)$ (NegDel)
2. $\neg(p(y) \wedge \neg\exists x(true \wedge x = y))$ first-order logic
3. $[\alpha_1]\neg p(y)$ (MP), 1, 2
4. $\forall y[\alpha_1]\neg p(y)$ (UnGen), 3

5. $\forall y[\alpha_1]\neg p(y) \rightarrow [\alpha_1]\forall y \neg p(y)$ (Barcan)
6. $[\alpha_1]\forall y \neg p(y)$ (MP), 5, 4
7. $p(y) \vee \exists x(q(x) \wedge x = y) \rightarrow [\alpha_2]p(y)$ (PosIns)
8. $q(y) \leftrightarrow \exists x(q(x) \wedge x = y)$ first-order logic
9. $p(y) \vee q(y) \rightarrow [\alpha_2]p(y)$ replace equivalents, 7, 8
10. $\neg p(y) \rightarrow (\neg q(y) \vee [\alpha_2]p(y))$ propositional reasoning, 9
11. $\neg q(y) \leftrightarrow [\alpha_2]\neg q(y)$ frame theorem
12. $\neg p(y) \rightarrow ([\alpha_2]\neg q(y) \vee [\alpha_2]p(y))$ replace equivalents, 10, 11
13. $\neg p(y) \rightarrow [\alpha_2](\neg q(y) \vee p(y))$ modal reasoning, 12
14. $\neg p(y) \rightarrow [\alpha_2](q(y) \rightarrow p(y))$ definition of $\rightarrow$, 13
15. $\neg(p(y) \vee \exists x(q(x) \wedge x = y)) \rightarrow [\alpha_2]\neg p(y)$ (NegIns)

$\cdots$ (similar to 8–13)

22. $\neg p(y) \rightarrow [\alpha_2](p(y) \rightarrow q(y))$ definition of $\rightarrow$, 21
23. $\neg p(y) \rightarrow [\alpha_2]((q(y) \rightarrow p(y)) \wedge (p(y) \rightarrow q(y)))$ modal reasoning, 14, 22
24. $\neg p(y) \rightarrow [\alpha_2](p(y) \leftrightarrow q(y))$ definition of $\leftrightarrow$, 23
25. $\forall y(\neg p(y) \rightarrow [\alpha_2](p(y) \leftrightarrow q(y)))$ (UnGen), 24
26. $\forall y \neg p(y) \rightarrow \forall y[\alpha_2](p(y) \leftrightarrow q(y))$ first-order reasoning, 25
27. $\forall y[\alpha_2](p(y) \leftrightarrow q(y)) \rightarrow [\alpha_2]\forall y(p(y) \leftrightarrow q(y))$ (Barcan)
28. $\forall y \neg p(y) \rightarrow [\alpha_2]\forall y(p(y) \leftrightarrow q(y))$ transitivity of $\rightarrow$, 26, 27
29. $[\alpha_1; \alpha_2]\forall y(p(y) \leftrightarrow q(y))$ seq. composition rule, 6, 28

This concludes the proof that the update program α satisfies the specification of a "copy" update program.

5. Discussion

5.1. Updating an object-oriented database

By an *object-oriented system* we mean a system of dynamic interacting objects; and by an *object* we mean, informally, an entity with a local state and behavior. Objects are divided into *classes*, where each class is a set of objects with similar external behavior. This view of object-oriented systems includes object-oriented database systems but is so general that it also includes embedded systems. For example, an object-oriented database can be viewed as a collection of *surrogates*, each of which represents a real-world entity. Each surrogate can then be modeled as an object. An embedded system can be viewed as a collection of objects such as sensors, interface objects, actuators and controllers. We restrict ourselves in this paper to object-oriented database systems. In this section, we give a rough sketch of how these general and vague ideas can be made more precise in DDL. More details about the syntax, semantics and inference rules of this way of specifying object-oriented systems are given by Wieringa et al. [39, 38].

The abstract data type specification of a DDL specification defines the types used in a specification, such as the natural numbers. For each class of objects that we want

to be able to talk about we specify an abstract data type in DDL called an *identifier type*. The identifier type of a class defines an infinite supply of closed terms that can be used as *object identifiers* (oids). Each oid can be used to refer to an object in the system. The intention is that once an oid is used to refer to an object, it always refers to that object and not to any other object. This intention concerns a relationship between formal entities (closed terms) and entities in the system (objects) and cannot be expressed in the specification itself. We assume that the specification of identifier types is sufficiently well structured that an oid can always be rewritten to a term that does not contain updateable function symbols.

Changeable properties of objects are represented by means of unary updateable function symbols and unary predicate symbols whose argument sort is an identifier sort. These symbols are now called *attributes*. There are no other updateable symbols than attributes. An example should make this clear. Suppose a specification of an extremely simple library database system contains a specification of the abstract data types *STRING* and in addition the following schema:

Identifier sorts: $BOOK$, $PERSON$,
Non-updateable functions:
$b_0 : BOOK$,
$next_b : \langle BOOK \rangle \to BOOK$,
$p_0 : PERSON$,
$next_p : \langle PERSON \rangle \to PERSON$.

This declares two identifier types, both containing an infinite set of oids. The updateable part of the specification declares the following attributes:

$author : \langle BOOK \rangle \to PERSON$
$title : \langle BOOK \rangle \to STRING$
$Borrowed : BOOK$.

Now, suppose we want to define the effect of a *borrow* action declaratively. We would like to do this with the axiom

(1) $\forall b : BOOK([borrow(b)]Borrowed(b))$.

This formula does not obey the syntax of DDL as defined in this paper, for $borrow(b)$ is an atomic update not part of DDL. It is however simple to extend the syntax of DDL to allow other atomic updates besides the standard ones treated so far. We proceed as if this had been done.

The above axiom must be supplemented with a frame axiom that says that the author of a book does not change when it is borrowed:

(2) $\forall b : BOOK, p : PERSON(author(b) = p \to [borrow(b)]author(b) = p)$.

To specify that a book can only be borrowed when it is not already borrowed, we can write the enabling axiom

(3) $\forall b : BOOK(\langle borrow(b) \rangle true \to \neg Borrowed(b))$.

This says that $borrow(b)$ leads to a next state only if currently, it is not borrowed.

A naive guess at an implementation of the *borrow* action would be the following DDL update program:

$IBorrowed(b)$ **where** $\neg Borrowed(b)$.

This can be shown to be an incorrect implementation of the specification. By using the PosIns axiom of Fig. 7, it can be shown that this satisfies (1) and from the axiom FrInsUFun it immediately follows that this satisfies (2). However, the action $IBorrowed(b)$ **where** $\neg Borrowed(p)$ leads to a world in which $Borrowed(b)$ holds if in the current world, $\neg Borrowed(b)$ holds. Nothing is said about the case where in the current world, $Borrowed(b)$ holds. Thus, there are models of the implementation in which the implementation leads from a world in which $Borrowed(b)$ holds to a world in which $Borrowed(b)$ holds and this violates constraint (3). Thus, the proposed update program does not implement the specification.

Without spelling out the proof, we note that a correct implementation is the program

$\neg Borrowed(b)?; IBorrowed(b)$.

Note however that another correct implementation is the program *false*?. This program does not terminate and satisfies specification (1)–(3). To rule out this implementation, we should add an axiom that guarantees the existence of a successor world, i.e.

(4) $\forall b : BOOK(\neg Borrowed(b) \rightarrow \langle borrow(b)\rangle true)$.

Specifications (3) and (4) give necessary and sufficient conditions for the applicability of $borrow(b)$.

In order to make DDL easier to use, we could extend it with the facility to define updates, e.g.

(5) $borrow(p) = \neg Borrowed(b)?; IBorrowed(b)$.

In such an extension of DDL, instead of giving a declarative specification of an update (which requires the addition of frame axioms), we could instead define updates by means of regular update programs such as (5) and then derive formulas such as (1)–(4) that specify postconditions, frame axioms and necessary and sufficient enabling conditions.

5.2. Comparison with other approaches

While there is considerable agreement on how to describe database states in logic [17, 32, 9], there are different schools of thought about the description of database *updates* in logic. One school of thought views a database update as a *belief revision*, and various authors define operators to add a formula (a new belief) to a set of formulas (representing the current belief state) [10, 11, 14, 37, 40]. Winslett gives a survey of approaches in this class [41]. In belief revision, new information about one state of the world must be incorporated with existing information about the same state of the world (the existing and new information may be inconsistent). This is different from

database updates, where information about a *change* in the state of the world must be incorporated with information about the current state of the world. We therefore agree with Katsuno and Mendelzohn [23] that database updates require a different semantics from that of belief revision and that for example incomplete information updates should satisfy different requirements than the well-known Gärdenfors postulates [3, 18].

In both the belief revision approach and the approach of Katsuno and Mendelsohn, the current state and the new information are written down in the same logic; usually propositional logic. One of the rationality postulates for database updates, given by Katsuno and Mendelsohn [23] is then that the revision of (current state) ϕ with (new information) ψ must imply ψ. In DDL, there is a clear separation between the state description language (the formulas) and the state change (revision) language (the update programs), and there is no notion of implication between formulas and update programs. Therefore, the application of both the Gärdenfors and the Katsuno and Mendelsohn postulates to our update languages is meaningless.

We view database updates as updates of the world, not of our beliefs of the world, and we separate the update language from the state description language. A very similar approach is taken by Golshani et al. [19] and Khosla et al. [24], who study updates to relational databases in a modal logic with multiple modalities; they assume a modal operator $[u]$ for every update action u. However, the interpretation of the update action is not fixed (as it is in our semantics) but must be specified by the user by giving some defining axioms. This means that the writer of the specification must solve the frame problem, i.e. she must indicate, for every atomic update, what part of the state remains unchanged. A second difference is that the only way they allow atomic updates to be combined is by sequential composition, where we allow a regular update language that also contains operators for choice, test and iteration. A further (minor) difference is that they view a database state as a model of a set of non-modal formulas, where we view a database state as a set of non-modal formulas.

Manchanda and Warren [27] also use modal logic with multiple modalities. Just like we do, they assume for every base atom an insert and a delete action with a fixed interpretation. They combine such updates into programs by means of *update rules* of the form $\langle E\rangle \leftarrow C_1 \wedge \langle E_1\rangle(C_2 \wedge \langle E_2\rangle(\ldots))$ (the update E is defined in terms of updates E_i, every C_i is a conjunction of atoms). Update rules perform the same function as our update programs, but they also allow recursion. If we exclude recursion, our approach has at least the same expressive power as theirs. An important part of their paper is concerned with view updates, which we do not consider in this paper.

Naqvi and Krishnamurty [28] and Naqvi and Tsur [29] introduce an update operator in LDL (Logical Language for Data and Knowledge Bases). They add control to a logical query language in such a way that the language can be used for queries as well as updates (which resemble queries with side effects). This approach is opposite to ours, for we separate control from logical rules by restricting control to the update program α in $[\alpha]\phi$.

Bonner and Kifer present an extension of logic programming called *transaction logic programming* [7, 5, 6, 8]. In transaction logic, some predicates represent states

(as usual) and some predicates represent state changes. The formulas of transaction logic are built from the standard (unsorted) first-order logic operators and quantifiers, with an additional operator *serial conjunction* (written as $\otimes$). Intuitively, if $ins : p(a)$ is an atom that inserts a in the relation p in a relational database state and $ins : p(b)$ similarly inserts b in p, then $ins : p(a) \otimes ins : p(b)$ first inserts a and then b in p. If insertion of b fails for some reason, then the whole transaction fails and the database state remains unchanged. (Note that $ins : p$ is a predicate symbol just like p itself, it only obtains its special meaning as an update through the transition base.)

In a transaction logic program atomic updates are defined in a *transition base* and query and update programs are defined in a *transaction base*. The transition base is a finite set of atomic updates of the form $\langle D_1, D_2 \rangle u$, meaning that the atom u may transform state D_1 into state D_2. The transaction base is a finite set of *serial Horn clauses*, which are formulas $p \leftarrow p_1 \otimes \cdots \otimes p_n$ (where p, $p_1, \ldots, p_n$ are atoms). Intuitively, this means that after the state changes p_1 to p_n are executed successfully (in that order), p is true.

This approach differs from ours, because in DDL we separate actions from states. In transaction logic, an update is a query with a side effect, but in DDL, an update is a program that is syntactically different from queries (which are formulas). The transition base is a way of defining the effect of all possible atomic updates by enumerating the ways in which an update atom can change a database. By contrast, we have a small set of atomic updates, for which we give axioms. Other updates can be constructed as regular programs from these atomic updates. In a technical report, Bonner and Kifer [4] use the transition base in the definition of the semantics, as well as in the definition of an inference system for transaction logic. The report as well as the published version [7] give a large number of interesting applications of transaction logic.

Groenboom and Renardel [20] define a language called modal logic of creation and modification (MLCM) which is similar to DDL. Fensel and Groenboom [15, 16] extend this to a language called modal logic of predicate modification (MLPM) which contains operators similar to our $\&X\mathcal{I}pT$ **where** ϕ and $\&X\mathcal{D}pT\phi$ **where** . Groenboom and Renardel give an axiomatization and a Henkin-like completeness proof for the $*$-free fragment of MLCM. Due to differences in syntax and semantics, this result is not immediately transferrable to DDL but such a transfer is an interesting possibility that is subject of further research. We are currently investigating this possibility.

6. Conclusions

We define first-order regular update logic (FUL) as a variant of dynamic logic in order to have a framework within which different database update logics can be defined. RAUL is an instantiation of FUL with relational assignment but without regular process combinators. These process combinators can be added without additional work, because the choice of atomic updates is orthogonal to the choice of process combinators. RAUL has a declarative semantics in terms of a Kripke structure. An operational semantics

has been given for definite RAUL in the form of a transition system, which restricts itself to database states without any indefinite information, and for indefinite RAUL, in which for each predicate there is a finite indefiniteness as to the tuples for which it holds. The operational semantics were proven to be sound and complete with respect to the declarative semantics.

DDL is an instantiation of FUL in which we use updates on both predicate symbols (bulk updates) and function symbols (assignment) as atomic updates. DDL has a declarative semantics in terms of a Kripke structure. We gave an operational semantics for definite databases in the form of a transition system and showed it to be sound and complete with respect to the declarative semantics of DDL.

The axiom systems of RAUL and DDL are sound but have not been shown to be complete with respect to full structures [35, 33]. However, under the assumptions of domain closure and unique names, completeness follows trivially from an earlier result about propositional database update logic. We also showed that the update language of DDL is update complete. Completeness with respect to full structures remains a topic for further study. Other topics for further research include the extension of DDL to an update language for object-oriented systems along the lines indicated in Section 5.1 and the use of regular programs to define object-oriented system transactions.

Owing are due to the anonymous referees, whose many suggestions improved the presentation and content of the paper.

Appendix A. Proofs of theorems

A.1. Soundness of RAUL *axioms*

In this section we prove soundness of the axioms (PosAt), (NegAt), (Disj), (Diff), (Prod), (Proj), (Sel), (FrUFun1), (FrUFun2), (FrUPred1), (FrUPred2) and (SuccEx) of Fig. 5. For the five axioms concerning the relational algebra operators ((Disj), (Diff), (Prod), (Proj) and (Sel)) and for the successor existence axiom (SuccEx), we need the fact that we are axiomatizing truth in the *full* structure.

Proposition A.1 (Soundness of (PosAt)). *Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a possible world in S, let I_V be a variable interpretation on $\mathscr{A}$, let p and q be updateable predicate symbols of the same arity and let T be a tuple of terms of the same arity as p. Then*

$$S, w, I_V \models qT \rightarrow [p := q]pT.$$

Proof. Assume $S, w, I_V \models qT$ and let w' be a world in $succ(p := q)(w)$. Then the $\rightarrow$ case of Proposition 2.22, and Definition 2.20 give that it is sufficient to prove $S, w', I_V \models pT$. From $S, w, I_V \models qT$, and the fact that q is updateable, we get by Definition 2.20 that $T^{\mathscr{A},w,I_V} \in w(q)$. From $w' \in succ(p := q)(w)$, we get by Definition 2.19 that $(w, w') \in m(p := q)$, so Definition 3.8 gives $w' = w\{p \mapsto w(q)\}$. Then $w \restriction_{NUFun}$

$= w' \restriction_{NUFun}$, so Lemma 2.17 gives $T^{\mathscr{A},w,I_V} = T^{\mathscr{A},w',I_V}$. We derive that $T^{\mathscr{A},w',I_V} \in w(q) = w'(p)$. Definition 2.20 then gives $S,w',I_V \models pT$, as had to be proven. □

The soundness proof of axiom (NegAt) is omitted; it is similar to the above soundness proof for (PosAt). Soundness of the axioms (Disj), (Diff), (Prod), (Proj) and (Sel) is proven in a very similar way, using the following lemma.

Lemma A.2. *Let $\mathscr{A}$ be an algebra, let w be a possible world on $\mathscr{A}$, let I_V be a variable interpretation on $\mathscr{A}$, let $p \in UPred$, let e be a relational algebra expression of the same arity as p and let T be a tuple of terms of the same arity as p. Then*

$$\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e]pT \Leftrightarrow T^{\mathscr{A},w,I_V} \in w(e).$$

Proof. $S,w,I_V \models [p := e]pT \Leftrightarrow$ (Definition 2.20) $\forall w' \in succ(p := e) : S,w',I_V \models pT$ $\Leftrightarrow$ (Definitions 2.19, 3.8 and 2.20) $\forall w' \in S$ with $w' = w\{p \mapsto w(e)\}$: $T^{\mathscr{A},w',I_V} \in w'(p)$ $\Leftrightarrow$ (there is one world $w\{p \mapsto w(e)\}$ in $\mathscr{F}_{\mathscr{A}}$) $T^{\mathscr{A},w\{p\mapsto w(e)\},I_V} \in w\{p \mapsto w(e)\}(p) \Leftrightarrow T^{\mathscr{A},w\{p\mapsto w(e)\},I_V} \in w(e) \Leftrightarrow$ (Lemma 2.17, as $w \restriction_{UFun} = w\{p \mapsto w(e)\} \restriction_{UFun}$) $T^{\mathscr{A},w,I_V} \in w(e)$. □

To prove soundness of axiom (Sel), we need an additional lemma that links truth of relational algebra tests in tuples of domain elements, with truth of formulas in worlds of structures.

Lemma A.3. *Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a world in S, let I_V be a variable interpretation on $\mathscr{A}$, let ϕ be an $(s_1,\ldots,s_{[\![k]\!]})$ relational algebra test and let $T = (t_1,\ldots,t_{[\![k]\!]})$ be a tuple of terms of sorts $s_1,\ldots,s_{[\![k]\!]}$, respectively. Then*

$$T^{\mathscr{A},w,I_V} \models \phi \Leftrightarrow S,w,I_V \models \phi[T/\mathscr{N}].$$

Proof. By induction on the structure of ϕ. We only prove the $\phi = (l = t)$ case; the $\phi = (l_1 = l_2)$ case can be proven similarly and the $\phi = \phi_1 \vee \phi_2$ and $\phi = \neg\psi$ cases follow in a straightforward way from the induction hypothesis. So consider the case $\phi = (l = t)$, for $l \in \mathscr{N}$ with $1 \leqslant [\![l]\!] \leqslant [\![k]\!]$ and t an immutable term of sort $s_{[\![l]\!]}$. Now $T^{\mathscr{A},w,I_V} \models l = t \Leftrightarrow$ (Definition 3.5) $t^{\mathscr{A},w,I_V}_{[\![l]\!]} = t^{\mathscr{A}} \Leftrightarrow t^{\mathscr{A},w,I_V}_{[\![l]\!]} = t^{\mathscr{A},w,I_V} \Leftrightarrow$ (Definition 2.20) $S,w,I_V \models t_{[\![l]\!]} = t \Leftrightarrow$ (Definition 3.9) $S,w,I_V \models (l = t)[T/\mathscr{N}]$. □

Proposition A.4 (Soundness of (Disj), (Diff), (Prod), (Proj) and (Sel)). *Let $\mathscr{A}$ be an algebra, let w be a possible world on $\mathscr{A}$ and let I_V be a variable interpretation on $\mathscr{A}$. Then all axioms* (Disj), (Diff), (Prod), (Proj) *and* (Sel) *are true in world w of structure $\mathscr{F}_{\mathscr{A}}$ under variable interpretation I_V.*

Proof. All axioms (Disj), (Diff), (Prod), (Proj) and (Sel) are equivalences (formulas of the form $\phi \leftrightarrow \psi$). We then use the $\leftrightarrow$ case of Proposition 2.22, which states that to prove $\mathscr{F}_{\mathscr{A}},w,I_V \models \phi \leftrightarrow \psi$, it is sufficient to prove $\mathscr{F}_{\mathscr{A}},w,I_V \models \phi \Leftrightarrow \mathscr{F}_{\mathscr{A}},w,I_V \models \psi$. Using this, we give the proofs for the individual axioms.

- $\mathscr{F}_{\mathscr{A}},w,I_V \models [p := e_1 \cup e_2]pT \Leftrightarrow$ (Lemma A.2) $T^{\mathscr{A},w,I_V} \in w(e_1 \cup e_2) \Leftrightarrow$ (Definition 3.7) $T^{\mathscr{A},w,I_V} \in w(e_1) \cup w(e_2) \Leftrightarrow T^{\mathscr{A},w,I_V} \in w(e_1)$ or $T^{\mathscr{A},w,I_V} \in w(e_2) \Leftrightarrow$ (Lemma A.2)

$\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e_1]pT$ or $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e_2]pT \Leftrightarrow$ ($\vee$ case of Proposition 2.22) $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e_1]pT \vee [p := e_2]pT$.

- $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e_1 \setminus e_2]pT \Leftrightarrow$ (Lemma A.2) $T^{\mathscr{A},w,I_V} \in w(e_1 \setminus e_2) \Leftrightarrow$ (Definition 3.7) $T^{\mathscr{A},w,I_V} \in w(e_1) \setminus w(e_2) \Leftrightarrow$ $T^{\mathscr{A},w,I_V} \in w(e_1)$ and $T^{\mathscr{A},w,I_V} \notin w(e_2) \Leftrightarrow$ (Lemma A.2) $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e_1]pT$ and $\mathscr{F}_{\mathscr{A}}, w, I_V \not\models [p := e_2]pT \Leftrightarrow$ ($\neg$ and $\wedge$ cases of Proposition 2.22) $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e_1]pT \wedge \neg[p := e_2]pT$.
- $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e_1 \times e_2]pT_1T_2 \Leftrightarrow$ (Lemma A.2) $(T_1T_2)^{\mathscr{A},w,I_V} \in w(e_1 \times e_2) \Leftrightarrow T_1^{\mathscr{A},w,I_V}$ $T_2^{\mathscr{A},w,I_V} \in w(e_1 \times e_2) \Leftrightarrow$ (Definition 3.7) $T_1^{\mathscr{A},w,I_V} T_2^{\mathscr{A},w,I_V} \in w(e_1) \times w(e_2) \Leftrightarrow$ $T_1^{\mathscr{A},w,I_V}$ $\in w(e_1)$ and $T_2^{\mathscr{A},w,I_V} \in w(e_2) \Leftrightarrow$ (Lemma A.2) $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p_1 := e_1]pT_1$ and $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p_2 := e_2]pT_2 \Leftrightarrow$ ($\wedge$ case of Proposition 2.22) $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p_1 := e_1]pT_1 \wedge$ $[p_2 := e_2]pT_2$.
- $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e[k_1, \ldots, k_n]p(t_1, \ldots, t_n) \Leftrightarrow$ (A.2) $(t_1, \ldots, t_n)^{\mathscr{A},w,I_V} \in w(e[k_1, \ldots, k_n])$ $\Leftrightarrow$ (def. 3.7) $(t_1, \ldots, t_n)^{\mathscr{A},w,I_V} \in w(e)[[\![k_1]\!], \ldots, [\![k_n]\!]] \Leftrightarrow$ (def. 3.6) $\exists(d_1, \ldots, d_m) \in$ $w(e) : \forall i(1 \leqslant i \leqslant n) : t_i^{\mathscr{A},w,I_V} = d_{[\![k_i]\!]} \Leftrightarrow$ (all the variables $x_1, \ldots, x_m$ are different) there is a variable interpretation I'_V with $I'_V \restriction_{Var \setminus \{x_1,\ldots,x_m\}} = I_V \restriction_{Var \setminus \{x_1,\ldots,x_m\}}$ and $I'_V((x_1, \ldots,$ $x_m)) \in w(e)$ and $\forall i(1 \leqslant i \leqslant n) : t_i^{\mathscr{A},w,I_V} = I'_V(x_{[\![k_i]\!]}) \Leftrightarrow$ (Definition 2.16 and Lemma 2.17) $\exists I'_V : I'_V \restriction_{Var \setminus \{x_1,\ldots,x_m\}} = I_V \restriction_{Var \setminus \{x_1,\ldots,x_m\}}$ and $(x_1, \ldots, x_m)^{\mathscr{A},w,I'_V} \in w(e)$ and $\forall i(1 \leqslant i \leqslant n)$: $t_i^{\mathscr{A},w,I'_V} = x_{[\![k_i]\!]}^{\mathscr{A},w,I'_V} \Leftrightarrow$ (Lemma A.2 and Definition 2.20) $\exists I'_V : I'_V \restriction_{Var \setminus \{x_1,\ldots,x_m\}} = I_V$ $\restriction_{Var \setminus \{x_1,\ldots,x_m\}}$ and $\mathscr{F}_{\mathscr{A}}, w, I'_V \models [p' := e]p'(x_1, \ldots, x_m)$ and $\forall i$ $(1 \leqslant i \leqslant n) : \mathscr{F}_{\mathscr{A}}, w, I'_V \models$ $t_1 = x_{[\![k_1]\!]} \Leftrightarrow$ (cases $\wedge$ and $\exists$ of Proposition 2.22) $\mathscr{F}_{\mathscr{A}}, w, I_V \models \exists x_1, \ldots, x_m([p' := e]$ $p'(x_1, \ldots, x_m) \wedge t_1 = x_{[\![k_1]\!]} \wedge \cdots \wedge t_n = x_{[\![k_n]\!]})$.
- $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p := e$ **where** $\phi]pT \Leftrightarrow$ (Lemma A.2) $T^{\mathscr{A},w,I_V} \in w(e$ **where** $\phi) \Leftrightarrow$ (Definition 3.7) $T^{\mathscr{A},w,I_V} \in w(e)$ and $T^{\mathscr{A},w,I_V} \models \phi \Leftrightarrow$ (Lemmas A.2 and A.3) $\mathscr{F}_{\mathscr{A}}, w, I_V \models [p$ $:= e]pT$ and $\mathscr{F}_{\mathscr{A}}, w, I_V \models \phi[T/\mathscr{N}] \Leftrightarrow$ ($\wedge$ case of Proposition 2.22) $\mathscr{F}_{\mathscr{A}}, w, I_V \models$ $[p := e]pT \wedge \phi[T/\mathscr{N}]$. □

Proposition A.5 (Soundness of (FrUFun1)). *Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a possible world in S, let I_V be a variable interpretation on $\mathscr{A}$, let $p := e$ be a relational algebra update, let $f \in UFun$ and let T be a tuple of terms of the arity of f. Then*

$$S, w, I_V \models fT = t \rightarrow [p := e]fT = t.$$

Proof. Assume $S, w, I_V \models fT = t$ and let w' be a world in $succ(p := e)(w)$. Then the $\rightarrow$ case of Proposition 2.22, and Definition 2.20 give that it is sufficient to prove $S, w', I_V \models fT = t$. From $S, w, I_V \models fT = t$, we get by Definition 2.19 that $(fT)^{\mathscr{A},w,I_V} = t^{\mathscr{A},w,I_V}$. From $w' \in succ(p := q)(w)$, we get by Definition 2.19 that $(w, w') \in m(p := q)$, so Definition 3.8 gives $w' = w\{p \mapsto w(q)\}$. Then $w \restriction_{NUFun} = w' \restriction_{NUFun}$, so Lemma 2.17 gives $(fT)^{\mathscr{A},w',I_V} = t^{\mathscr{A},w',I_V}$. Definition 2.20 finally gives $S, w', I_V \models fT$, which had to be proven. □

Soundness of the axioms (FrUFun2), (FrUPred1) and (FrUPred2) can be proven in a similar way as as soundness of (FrUFun1); these soundness proofs are omitted.

Proposition A.6 (Soundness of (SuccEx)). *Let $\mathscr{A}$ be an algebra, let w be a possible world on $\mathscr{A}$, let I_V be a variable interpretation and let $p := e$ be a relational algebra update. Then*

$$\mathscr{F}_{\mathscr{A}}, w, I_V \models \langle p := e \rangle true.$$

Proof. Let $w' = w\{p \mapsto w(e)\}$. Then w' is a possible world on $\mathscr{A}$ and as $\mathscr{F}_{\mathscr{A}}$ contains all possible worlds on $\mathscr{A}$, we have that $w' \in \mathscr{F}_{\mathscr{A}}$. Definition 3.8 gives $(w, w') \in m(p := e)$, so by Definition 2.19, we have $w' \in succ(p := e)(w)$. The *true* case of Proposition 2.22 gives $\mathscr{F}_{\mathscr{A}}, w', I_V \models true$. The $\langle\rangle$ case of Proposition 2.22 finally gives $\mathscr{F}_{\mathscr{A}}, w, I_V \models \langle p := e \rangle true$. □

This concludes the proof that all RAUL axioms are sound in the full structure. As all FUL axioms are also sound in the full structure and all FUL inference rules preserve soundness in the full structure, we get that the axiomatization of RAUL is sound with respect to truth in the full structure.

A.2. Equivalence of the operational and declarative semantics of RAUL

All relational algebra updates are of the form $p := e$, for p a predicate symbol and e a matching relational expression. We prove that the operational semantics is sound and complete with respect to the declarative semantics. Before we prove the soundness and completeness propositions, we make two remarks:

- The transition rules we use in the transition system have a restricted format: they are all of the form $\langle DB, \alpha \rangle \to DB'$. (We do not have transitions of the more general format $\langle DB, \alpha \rangle \to \langle DB', \alpha' \rangle$.) So if we take the transitive closure of the $\to$ relation, we just end up with the same relation. Therefore we only consider the $\to$ relation, and not the $\to^*$ relation. (The reflexive part of $\to^*$ is not very interesting anyway.)
- With every relational database state, a unique possible world is associated. This means that the notions of soundness and completeness are restricted versions of the notions of soundness and completeness that were used for the *indefinite* operational semantics for PDDL [35].

Lemma A.7. *Let DB be a relational database state, let $p : \langle s_1, \ldots, s_n \rangle \in UPred$, let ϕ be an $(s_1, \ldots, s_n)$ relational algebra test, let $\mathscr{A}$ be an initial algebra of E and let $T = (t_1, \ldots, t_n)$ be a tuple of terms in normal form of sorts $s_1, \ldots, s_n$. Then*

$$T \in cs(DB(p), \phi) \Leftrightarrow T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p \textbf{ where } \phi).$$

Proof. We prove the lemma by induction on the structure of ϕ:

1. $\phi = (k = t)$, for $k \in \mathscr{N}$ with $1 \leqslant [\![k]\!] \leqslant n$ and t a closed term of sort $s_{[\![k]\!]}$. Now $T \in cs(DB(p), k = t) \Leftrightarrow$ (Definition 3.11) $T \in DB(p)$ and $t_{[\![k]\!]} = normalize(t) \Leftrightarrow$ (for closed terms in normal form, equality in the initial algebra coincides with syntactic equality) $T \in DB(p)$ and $t^{\mathscr{A}}_{[\![k]\!]} = normalize(t)^{\mathscr{A}} \Leftrightarrow$ (semantically, a term is equal to its

normal form) $T \in DB(p)$ and $t^{\mathscr{A}}_{[\![k]\!]} = t^{\mathscr{A}} \Leftrightarrow$ (Definitions 2.24 and 3.5) $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p)$ and $T^{\mathscr{A}} \models k = t \Leftrightarrow$ (Definition 3.7) $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p$ **where** $k = t)$.

2. $\phi = (k = l)$, for $k, l \in \mathscr{N}$ with $1 \leqslant [\![k]\!], [\![l]\!] \leqslant n$ and $s_{[\![k]\!]} = s_{[\![l]\!]}$. The proof of this case is similar to the proof of the first case and has been omitted.
3. $\phi = \phi_1 \vee \phi_2$, for ϕ_1, ϕ_2 relational algebra tests. Now $T \in cs(DB(p), \phi_1 \vee \phi_2) \Leftrightarrow$ (Definition 3.11) $T \in cs(DB(p), \phi_1)$ or $T \in cs(DB(p)\phi_2) \Leftrightarrow$ (by the induction hypothesis on ϕ_1 and ϕ_2) $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p$ **where** $\phi_1)$ or $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p$ **where** $\phi_2)$ (Definition 3.7) ($T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p)$ and $T^{\mathscr{A}} \models \phi_1$) or ($T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p)$ and $T^{\mathscr{A}} \models \phi_2$) $\Leftrightarrow T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p)$ and ($T^{\mathscr{A}} \models \phi_1$ or $T^{\mathscr{A}} \models \phi_2$) $\Leftrightarrow$ (Definition 3.5) $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p)$ and $T^{\mathscr{A}} \models \phi_1 \vee \phi_2 \Leftrightarrow$ (Definition 3.7) $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p$ **where** $\phi_1 \vee \phi_2)$.
4. $\phi = \neg\psi$, for relational algebra test ψ. Now $T \in cs(DB(p), \neg\psi) \Leftrightarrow$ (Definition 3.11) $T \in DB(p) \setminus cs(DB(p), \psi) \Leftrightarrow T \in DB(p)$ and $T \notin cs(DB(p), \psi) \Leftrightarrow$ (Definition 2.24 and the induction hypothesis on ψ) $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p)$ and $T^{\mathscr{A}} \notin pw_{\mathscr{A}}(DB)(p$ **where** $\psi) \Leftrightarrow$ (Definition 3.7) $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p)$ and ($T^{\mathscr{A}} \notin pw_{\mathscr{A}}(DB)(p)$ or $T^{\mathscr{A}} \not\models \psi) \Leftrightarrow$ $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p)$ and $T^{\mathscr{A}} \not\models \psi \Leftrightarrow$ (Definition 3.5) $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p)$ and $T^{\mathscr{A}} \models \neg\psi \Leftrightarrow$ (Definition 3.7) $T^{\mathscr{A}} \in pw_{\mathscr{A}}(DB)(p$ **where** $\neg\psi)$. □

Proposition A.8 (Soundness of the operational semantics of RAUL). *Let $\mathscr{A}$ be an initial algebra of E and let α be a relational algebra update. Then for all relational database states DB and DB′ with $\langle DB, \alpha\rangle \to DB'$, we have*

$$(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha).$$

Proof. The relational algebra update α has the format $p := e$. We prove the proposition by induction on the structure of e. We prove the $e = q$, $e = e_1 \cup e_2$ and $e = e'$ **where** ϕ cases in complete detail; the proofs of the $e = e_1 \setminus e_2$, $e = e_1 \times e_2$ and $e = e'[k_1, \ldots, k_n]$ cases have the same structure as the proof of the $e = e_1 \cup e_2$ case, and are therefore only sketched/omitted. For all cases, let DB and DB' be relational database states such that $\langle DB, \alpha\rangle \to DB'$.

1. $e = q$, for some updateable predicate symbol q. The only transition rule with as conclusion the format $\langle DB, p := q\rangle \to DB'$ is (TAtom). From (TAtom), we get that $DB' = DB\{p \mapsto DB(q)\}$. For all predicate symbols $r \neq p$ we then have $pw_{\mathscr{A}}(DB')(r) =$ (Definition 2.24) $\{T^{\mathscr{A}} \mid T \in DB'(r)\} = \{T^{\mathscr{A}} \mid T \in DB(r)\} =$ (Definition 2.24) $pw_{\mathscr{A}}(DB)(r)$. Furthermore, $pw_{\mathscr{A}}(DB')(p) =$ (Definition 2.24) $\{T^{\mathscr{A}} \mid T \in DB'(p)\} = \{T^{\mathscr{A}} \mid T \in DB(q)\} =$ (Definition 2.24) $pw_{\mathscr{A}}(DB)(q)$. So $pw_{\mathscr{A}}(DB') = pw_{\mathscr{A}}(DB)\{p \mapsto pw_{\mathscr{A}}(DB)(q)\}$ and Definition 3.8 gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(p := q)$.
2. $e = e_1 \cup e_2$, for some relational algebra expressions e_1 and e_2. The only transition rule with as conclusion the format $\langle DB, p := e_1 \cup e_2\rangle \to DB'$ is (TDisj). From the premises of (TDisj), we get relational database states DB_1 and DB_2 such that $\langle DB, p := e_1\rangle \to DB_1$, $\langle DB, p := e_2\rangle \to DB_2$ and $DB' = DB\{p \to DB_1(p) \cup DB_2(p)\}$. The induction hypothesis on e_1 gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB_1)) \in m_{\mathscr{F}_{\mathscr{A}}}(p := e_1)$ and the induction hypothesis on e_2 gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB_2)) \in m_{\mathscr{F}_{\mathscr{A}}}(p := e_2)$. By Definition 3.8, we get $pw_{\mathscr{A}}(DB_1)(p) = pw_{\mathscr{A}}(DB)(e_1)$ and $pw_{\mathscr{A}}(DB_2)(p) =$

$pw_{\mathscr{A}}(DB)(e_2)$. So $pw_{\mathscr{A}}(DB')(p) =$ (Definition 2.24) $\{T^{\mathscr{A}} \mid T \in DB'(p)\} = \{T^{\mathscr{A}} \mid T \in DB_1(p) \cup DB_2(p)\} = \{T^{\mathscr{A}} \mid T \in DB_1(p)\} \cup \{T^{\mathscr{A}} \mid T \in DB_2(p)\} =$ (Definition 2.24) $pw_{\mathscr{A}}(DB_1)(p) \cup pw_{\mathscr{A}}(DB_2)(p) = pw_{\mathscr{A}}(DB)(e_1) \cup pw_{\mathscr{A}}(DB)(e_2) =$ (Definition 3.7) $pw_{\mathscr{A}}(DB)(e_1 \cup e_2)$. Moreover, for all predicate symbols $q \neq p$, we have $DB'(q) = DB(q)$, so by Definition 2.24 we get for all $p \neq q$ that $pw_{\mathscr{A}}(DB')(q) = pw_{\mathscr{A}}(DB)(q)$. So $pw_{\mathscr{A}}(DB') = pw_{\mathscr{A}}(DB)\{p \mapsto pw_{\mathscr{A}}(DB)(e_1 \cup e_2)\}$ and Definition 3.8 gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(p := e_1 \cup e_2)$.

3. $e = e_1 \setminus e_2$, for some relational algebra expressions e_1 and e_2. The proof of this case is a direct transformation of the proof of the previous case: just replace $\cup$ by $\setminus$ and (TDisj) by TDiff). We just note that in the step $\{T^{\mathscr{A}} \mid T \in DB_1(p) \setminus DB_2(p)\} = \{T^{\mathscr{A}} \mid T \in DB_1(p)\} \setminus \{T^{\mathscr{A}} \mid T \in DB_2(p)\}$, we use the facts that all terms in $DB_1(p)$ and $DB_2(p)$ are immutable terms in normal form and $\mathscr{A}$ is the initial algebra. These two facts make sure that two tuples of terms are syntactically the same iff their interpretations are the same, so that the equality indeed holds. Note that we did not need this property for the $\cup$ case.
4. $e = e_1 \times e_2$, for some relational algebra expressions e_1 and e_2. The proof of this case is similar to the proof of the $\cup$ case and has therefore been omitted. (Note that the proof contains the step $\{T^{\mathscr{A}} \mid T \in DB_1(p_1) \times DB_2(p_2)\} = \{T^{\mathscr{A}} \mid T \in DB_1(p_1)\} \times \{T^{\mathscr{A}} \mid T \in DB_2(p_2)\}$. Unlike the corresponding step in the $\setminus$ case, here we do not need the facts that $DB_1(p_1)$ and $DB_2(p_2)$ are in normal form and $\mathscr{A}$ is an initial algebra.)
5. $e = e'[k_1, \ldots, k_n]$, for some relational algebra expression e' and positions $k_1, \ldots, k_n \in \mathscr{N}$. The proof of this case is similar to the proof of the $\cup$ case and has therefore been omitted. (Note that the proof contains the step $\{T^{\mathscr{A}} \mid T \in DB''(q)[[\![k_1]\!], \ldots, [\![k_n]\!]]\} = \{T^{\mathscr{A}} \mid T \in DB''(q)\}[[\![k_1]\!], \ldots, [\![k_n]\!]]\}$. Unlike the corresponding step in the $\setminus$ case, here we do not need the facts that $DB''(q)$ is in normal form and $\mathscr{A}$ is an initial algebra.)
6. $e = e'$ **where** ϕ, for some relational algebra expression e' and some relational algebra test ϕ. The only transition rule with as conclusion the format $\langle DB, p := e' \textbf{ where } \phi\rangle \to DB'$ is (TSel). From the premises of (TSel), we get that there is a relational database state DB'' such that $\langle DB, p := e'\rangle \to DB''$ and $DB' = DB\{p \to cs(DB''(p), \phi)\}$. The induction hypothesis on e' gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB'')) \in m_{\mathscr{F}_{\mathscr{A}}}(p := e')$. By Definition 3.8, we get $pw_{\mathscr{A}}(DB'')(p) = pw_{\mathscr{A}}(DB)(e')$. So $pw_{\mathscr{A}}(DB')(p) =$ (Definition 2.24) $\{T^{\mathscr{A}} \mid T \in DB'(p)\} = \{T^{\mathscr{A}} \mid T \in cs(DB''(p), \phi)\} =$ (Lemma A.7, using the fact that all domain elements in an initial algebra are named, "no junk") $pw_{\mathscr{A}}(DB'')(p \textbf{ where } \phi) =$ (Definition 3.7) $\{\bar{d} \in pw_{\mathscr{A}}(DB'')(p) \mid \bar{d} \models \phi\} = \{\bar{d} \in pw_{\mathscr{A}}(DB)(e') \mid \bar{d} \models \phi\} =$ (Definition 3.7) $pw_{\mathscr{A}}(DB)(e' \textbf{ where } \phi)$. Moreover, for all predicate symbols $q \neq p$, we have $DB'(q) = DB(q)$, so Definition 2.24 gives for all $q \neq p$: $pw_{\mathscr{A}}(DB')(q) = pw_{\mathscr{A}}(DB)(q)$. Definition 3.8 then gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(p := e' \textbf{ where } \phi)$. □

Proposition A.9 (Completeness of the operational semantics of RAUL). *Let $\mathscr{A}$ be an initial algebra of E and let α be a relational algebra update. Then for all relational*

database states DB and possible worlds w' on $\mathscr{A}$ with $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha)$, there is a relational database state DB' such that $\langle DB, \alpha\rangle \to DB'$ and $w' = pw_{\mathscr{A}}(DB')$.

Proof. The relational algebra update α has the format $p := e$. We prove the proposition by induction on the structure of e. For all cases, let DB be a relational database state and let w' be a possible world on $\mathscr{A}$ such that $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha)$:

1. $e = q$, for some updateable predicate symbol q. We define $DB' = DB\{p \mapsto DB(q)\}$. The transition rule (TAtom) then gives $\langle DB, p := q\rangle \to DB'$. From $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(p := q)$, Definition 3.8 gives $w'(p) = pw_{\mathscr{A}}(DB)(q)$ and for all predicate symbols $r \neq p$: $w'(r) = pw_{\mathscr{A}}(DB)(r)$. So we have $pw_{\mathscr{A}}(DB')(p) =$ (Definition 2.24) $\{T^{\mathscr{A}} \mid T \in DB'(p)\} = \{T^{\mathscr{A}} \mid T \in DB(q)\} =$ (Definition 2.24) $pw_{\mathscr{A}}(DB)(q) = w'(p)$. Moreover, for all predicate symbols $r \neq p$: $pw_{\mathscr{A}}(DB')(r) =$ (Definition 2.24) $\{T^{\mathscr{A}} \mid T \in DB'(r)\} = \{T^{\mathscr{A}} \mid T \in DB(r)\} =$ (Definition 2.24) $pw_{\mathscr{A}}(DB)(r) = w'(r)$. So $pw_{\mathscr{A}}(DB')$ and w' are the same functions, so $pw_{\mathscr{A}}(DB') = w'$.
2. $e = e_1 \cup e_2$, for some relational algebra expressions e_1 and e_2. We define the possible worlds w_1 and w_2 on $\mathscr{A}$ as $w_1 = pw_{\mathscr{A}}(DB)\{p \mapsto pw_{\mathscr{A}}(DB)(e_1)\}$ and $w_2 = pw_{\mathscr{A}}(DB)\{p \mapsto pw_{\mathscr{A}}(DB)(e_2)\}$. By Definition 3.8, $(pw_{\mathscr{A}}(DB), w_1) \in m_{\mathscr{F}_{\mathscr{A}}}(p := e_1)$ and $(pw_{\mathscr{A}}(DB), w_2) \in m_{\mathscr{F}_{\mathscr{A}}}(p := e_2)$. The induction hypothesis on e_1 then gives a relational database state DB_1 such that $\langle DB, p := e_1\rangle \to DB_1$ and $w_1 = pw_{\mathscr{A}}(DB_1)$. Similarly, the induction hypothesis on e_2 gives a relational database state DB_2 such that $\langle DB, p := e_1\rangle \to DB_2$ and $w_2 = pw_{\mathscr{A}}(DB_2)$. We define $DB' = DB\{p \mapsto DB_1 \cup DB_2\}$. The transition rule (TDisj) then gives $\langle DB, p := e_1 \cup e_2\rangle \to DB'$. As $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(p := e_1 \cup e_2)$, Definition 3.8 gives $w' = pw_{\mathscr{A}}(DB)\{p \mapsto pw_{\mathscr{A}}(DB)(e_1 \cup e_2)\}$. So for all predicate symbols $q \neq p$, we have $w'(q) = pw_{\mathscr{A}}(DB)(q) =$ (Definition 2.24) $\{T^{\mathscr{A}} \mid T \in DB(q)\} =$ (Definition 2.24) $pw_{\mathscr{A}}(DB')(q)$. Furthermore, we have $w'(p) = pw_{\mathscr{A}}(DB)(e_1 \cup e_2) =$ (Definition 3.7) $pw_{\mathscr{A}}(DB)(e_1) \cup pw_{\mathscr{A}}(DB)(e_2) = w_1(p) \cup w_2(p) = pw_{\mathscr{A}}(DB_1)(p) \cup pw_{\mathscr{A}}(DB_2)(p) =$ (Definition 2.24) $\{T^{\mathscr{A}} \mid T \in DB_1(p)\} \cup \{T^{\mathscr{A}} \mid T \in DB_2(p)\} = \{T^{\mathscr{A}} \mid T \in DB_1(p) \cup DB_2(p)\} = \{T^{\mathscr{A}} \mid T \in DB'(p)\} =$ (Definition 2.24) $pw_{\mathscr{A}}(DB')(p)$. So w' and $pw_{\mathscr{A}}(DB')$ are the same functions, so $w' = pw_{\mathscr{A}}(DB')$.
3. $e = e_1 \backslash e_2$, for some relational algebra expressions e_1 and e_2. The proof of this case is similar to the proof of the $\cup$ case. Note that just like in the $\backslash$ case of the soundness proof, we again need the equality $\{T^{\mathscr{A}} \mid T \in DB_1(p)\} \backslash \{T^{\mathscr{A}} \mid T \in DB_2(p)\} = \{T^{\mathscr{A}} \mid T \in DB_1(p) \backslash DB_2(p)\}$, so we again need the facts that $\mathscr{A}$ is an initial algebra and that terms in database states are in normal form.
4. $e = e_1 \times e_2$, for some relational algebra expressions e_1 and e_2. The proof of this case is similar to the proof of the $\cup$ case, and is omitted.
5. $e = e'[k_1, \ldots, k_n]$, for some relational algebra expression e' and positions $k_1, \ldots, k_n \in \mathscr{N}$. The proof of this case is similar to the proof of the $\cup$ case, and is omitted.
6. $e = e'$ **where** ϕ, for some relational algebra expression e' and some relational algebra test ϕ. The proof of this case is similar to the proof of the $\cup$ case, and is omitted. (In this case, we need Lemma A.7.) $\square$

A.3. Soundness of the axiomatization of DDL

In this section, we prove that the DDL specific axioms are sound. For all axioms except the successor existence axioms, we prove soundness in all structures; for the successor existence axioms we prove soundness in the full structure.

Proposition A.10 (Soundness of (PosIns)). *Let $p: \langle s_1,\ldots,s_n\rangle \in UPred$, let T and T' be tuples of terms of sorts $s_1,\ldots,s_n$, let ϕ be a formula and let X be a finite set of variables such that $FV(T')\cap X=\emptyset$. We then have*

$$\models pT' \vee \exists X(\phi \wedge T = T') \rightarrow [\&X\mathscr{I}pT \textbf{ where } \phi]pT'.$$

Proof. Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a world in S and let I_V be a variable interpretation on $\mathscr{A}$. We must prove $S,w,I_V \models pT' \vee \exists X(\phi \wedge T = T') \rightarrow [\&X\mathscr{I}pT \textbf{ where } \phi]pT'$. Assume $S,w,I_V \models pT' \vee \exists X(\phi \wedge T = T')$ and let w' be some world in $succ_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)(w)$. By the $\rightarrow$ case of Proposition 2.22, and by Definition 2.20, it then is sufficient to prove $S,w',I_V \models pT'$. From $w' \in succ_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)(w)$, we get by Definition 2.19 that $(w,w') \in m_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)$. From $S,w,I_V \models pT' \vee \exists X(\phi \wedge T = T')$ we derive by Definition 2.20 that (at least) one of the following two cases is true:

- $S,w,I_V \models pT'$. Definition 2.20 now gives $T^{\mathscr{A}',w,I_V} \in w(p)$. As $(w,w') \in m_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)$, we now get with Definition 4.9 that $T'^{\mathscr{A},w,I_V} \in w'(p)$.
- $S,w,I_V \models \exists X(\phi \wedge T = T')$. The $\exists$ case of Proposition 2.22 gives a variable interpretation I'_V such that $S,w,I'_V \models \phi \wedge T = T'$ and $I'_V \restriction_{Var\setminus X} = I_V \restriction_{Var\setminus X}$. The $\wedge$ case of Proposition 2.22 gives $S,w,I'_V \models \phi$ and $S,w,I'_V \models T = T'$. As $(w,w') \in m_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)$, we now get with Definition 4.9 that $T^{\mathscr{A},w,I'_V} \in w'(p)$. From $S,w,I'_V \models T = T'$, we get by Proposition 2.22 that $T^{\mathscr{A},w,I'_V} = T'^{\mathscr{A},w,I'_V}$. We know that $FV(T)\cap X=\emptyset$ and $I'_V \restriction_{Var\setminus X} = I_V \restriction_{Var\setminus X}$, so Lemma 2.17 gives $T'^{\mathscr{A},w,I_V} = T'^{\mathscr{A},w,I'_V}$. We get $T'^{\mathscr{A},w,I_V} = T^{\mathscr{A},w,I'_V}$, so $T'^{\mathscr{A},w,I_V} \in w'(p)$.

In both cases, $T'^{\mathscr{A},w,I_V} \in w'(p)$. From $(w,w') \in m_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)$, we derive with Definition 4.9 that $w' \restriction_{USym\setminus\{p\}} = w \restriction_{USym\setminus\{p\}}$. Lemma 2.17 then gives $T'^{\mathscr{A},w',I_V} = T'^{\mathscr{A},w,I_V}$. We derive $T'^{\mathscr{A},w',I_V} \in w'(p)$, and Definition 2.20 gives $S,w',I_V \models pT'$. □

Soundness of axioms (NegIns), (PosDel) and (NegDel) can be proven in a similar way as the above soundness proof for (PosIns); these soundness proofs are omitted. Proving soundness of the axioms (PosUpd) and (NegUpd) requires some more work, but the basic ideas are the same as the ones used in the soundness proof for (PosIns); these soundness proofs are also omitted.

Proposition A.11 (Soundness of (Assign)). *Let $f: \langle s_1,\ldots,s_n\rangle \rightarrow s \in UFun$, let T be a tuple of terms of sorts $s_1,\ldots,s_n$ and let t be a term of sort s such that $f \notin T,t$. We then have*

$$\models [fT := t]fT = t.$$

Proof. Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a world in S and let I_V be a variable interpretation on $\mathscr{A}$. We must prove $S,w,I_V \models [fT := t]fT = t$. Let w' be a world in $succ_{I_V}(fT := t)(w)$, by Definition 2.20, it then is sufficient to prove $S,w',I_V \models fT = t$. From $w' \in succ_{I_V}(fT := t)(w)$, we get by Definition 2.19 that $(w,w') \in m_{I_V}(fT := t)$, so Definition 4.9 gives $w'(f) = w(f)\{T^{\mathscr{A},w,I_V} \mapsto t^{\mathscr{A},w,I_V}\}$ and $w' \restriction_{USym\setminus\{f\}} = w \restriction_{USym\setminus\{f\}}$. As $f \notin T,t$, Lemma 2.17 then gives $T^{\mathscr{A},w',I_V} = T^{\mathscr{A},w,I_V}$ and $t^{\mathscr{A},w',I_V} = t^{\mathscr{A},w,I_V}$. So we have $(fT)^{\mathscr{A},w',I_V} =$ (Definition 2.16) $w'(f)(T^{\mathscr{A},w',I_V}) = w'(f)(T^{\mathscr{A},w,I_V}) = t^{\mathscr{A},w,I_V} = t^{\mathscr{A},w',I_V}$. Definition 2.20 then gives $S,w',I_V \models fT = t$. □

Proposition A.12 (Soundness of (FrInsUPred1)). *Let $p : \langle s_1,\ldots,s_n\rangle \in UPred$, let T be a tuple of terms of sorts $s_1,\ldots,s_n$, let $q : \langle s'_1,\ldots,s'_{n'}\rangle \neq p \in UPred$, let T' be a tuple of terms of sorts $s'_1,\ldots,s'_{n'}$, let X be a finite set of variables and let ϕ be a formula. We then have*

$$\models qT' \to [\&X\mathscr{I}pT \textbf{ where } \phi]qT'.$$

Proof. Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a world in S and let I_V be a variable interpretation on $\mathscr{A}$. We must prove $S,w,I_V \models qT' \to [\&X\mathscr{I}pT \textbf{ where } \phi]qT'$. Assume $S,w,I_V \models qT'$ and let w' be some world in $succ_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)(w)$. By the $\to$ case of Proposition 2.22, and by Definition 2.20, it then is sufficient to prove $S,w',I_V \models qT'$. From $w' \in succ_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)(w)$, Definition 2.19 gives $(w,w') \in m_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)$ so by Definition 4.9, we derive $w(p) \subseteq w'(p)$ and $w' \restriction_{USym\setminus\{p\}} = w \restriction_{USym\setminus\{p\}}$. From $S,w,I_V \models qT'$, we get by Definition 2.20 that $T'^{\mathscr{A},w,I_V} \in w(q)$. We get $T'^{\mathscr{A},w,I_V} \in w'(q)$. Lemma 2.17 gives $T'^{\mathscr{A},w',I_V} = T'^{\mathscr{A},w,I_V}$, so we have $T'^{\mathscr{A},w',I_V} \in w'(q)$. Definition 2.20 finally gives $S,w',I_V \models qT'$. □

Soundness of the frame axioms (FrInsUPred2), (FrDelUPred1), (FrDelUPred2), (FrUpdUPred1), (FrUpdUPred2), (FrAssUPred1) and (FrAssUPred2) is proven in a similar way as the above soundness proof for (FrInsUPred1); these soundness proofs are omitted.

Proposition A.13 (Soundness of (FrInsUFun)). *Let $f : \langle s_1,\ldots,s_n\rangle \to s \in UFun$, let T_1 be a tuple of terms of sorts $s_1,\ldots,s_n$, let $p : \langle s'_1,\ldots,s'_{n'}\rangle \in UPred$, let T be a tuple of terms of sorts $s'_1,\ldots,s'_{n'}$, let X be a finite set of variables and let ϕ be a formula. We then have*

$$\models fT_1 = t \to [\&X\mathscr{I}pT \textbf{ where } \phi]fT_1 = t.$$

Proof. Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a world in S and let I_V be a variable interpretation on $\mathscr{A}$. We must prove $S,w,I_V \models fT_1 = t \to [\&X\mathscr{I}pT \textbf{ where } \phi]fT_1 = t$. Assume $S,w,I_V \models fT_1 = t$ and let w' be some world in $succ_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)(w)$. By the $\to$ case of Proposition 2.22, and by Definition 2.20, it then is sufficient to prove $S,w',I_V \models fT_1 = t$. From $w' \in succ_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)(w)$, we get by Definition 2.19 that $(w,w') \in m_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)$

so by Definition 4.9, we derive $w'\restriction_{USym\setminus\{p\}} = w\restriction_{USym\setminus\{p\}}$. Lemma 2.17 then gives $(fT_1)^{\mathscr{A},w',I_V} = (fT_1)^{\mathscr{A},w,I_V}$ and $t^{\mathscr{A},w',I_V} = t^{\mathscr{A},w,I_V}$. From $S,w,I_V \models fT_1 = t$, we get by Definition 2.20 that $(fT_1)^{\mathscr{A},w,I_V} = t^{\mathscr{A},w,I_V}$. We derive $(fT_1)^{\mathscr{A},w',I_V} = t^{\mathscr{A},w',I_V}$, so by Definition 2.20, we have $S,w',I_V \models fT_1 = t$. □

Soundness of axioms (FrDelUFun), (FrUpdUFun) and (FrAssUFun1) can be proven in a similar way as the above soundness proof for (FrInsUFun); these soundness proofs are omitted.

Proposition A.14 (Soundness of (FrAssUFun2)). *Let* $f\colon \langle s_1,\ldots,s_n\rangle \to s \in UFun$, *let* T *and* T' *tuples of terms of sorts* $s_1,\ldots,s_n$ *and let* t *and* t' *be terms of sort* s *such that* $f \notin T',t'$. *We then have*

$$\models (fT' = t' \wedge T \neq T') \to [fT := t]fT' = t'.$$

Proof. Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a world in S and let I_V be a variable interpretation on $\mathscr{A}$. We must prove $S,w,I_V \models (fT' = t' \wedge T \neq T') \to [fT := t]fT' = t'$. Assume $S,w,I_V \models fT' = t' \wedge T \neq T'$ and let w' be some world in $succ_{I_V}(fT := t)(w)$. By the $\to$ case of Proposition 2.22, and by Definition 2.20, it then is sufficient to prove $S,w',I_V \models fT' = t'$. From $w' \in succ_{I_V}(fT := t)(w)$, we get by Definition 2.19 that $(w,w') \in m_{I_V}(fT := t)$ so by Definition 4.9, we derive $w'(f) = w(f)\{T^{\mathscr{A},w,I_V} \mapsto t^{\mathscr{A},w,I_V}\}$ and $w'\restriction_{USym\setminus\{f\}} = w\restriction_{USym\setminus\{f\}}$. As $f \notin T',t'$, Lemma 2.17 gives $T'^{\mathscr{A},w',I_V} = T'^{\mathscr{A},w,I_V}$ and $t'^{\mathscr{A},w',I_V} = t'^{\mathscr{A},w,I_V}$. From $S,w,I_V \models fT' = t' \wedge T \neq T'$, we get by Proposition 2.22 and Definition 2.20 that $(fT')^{\mathscr{A},w,I_V} = t'^{\mathscr{A},w,I_V}$ and $T^{\mathscr{A},w,I_V} \neq T'^{\mathscr{A},w,I_V}$. We get $(fT')^{\mathscr{A},w',I_V} =$ (Definition 2.16) $w'(f)(T'^{\mathscr{A},w',I_V}) = w'(f)(T'^{\mathscr{A},w,I_V})$ = (as $T'^{\mathscr{A},w,I_V} \neq T^{\mathscr{A},w,I_V}$)$w(f)(T'^{\mathscr{A},w,I_V})$ = (Definition 2.16) $(fT')^{\mathscr{A},w,I_V} = t'^{\mathscr{A},w,I_V} = t'^{\mathscr{A},w',I_V}$. Definition 2.20 finally gives $S,w',I_V \models fT' = t'$. □

Proposition A.15. *Let* $p\colon \langle s_1,\ldots,s_n\rangle \in UPred$, *let* T *be a tuple of terms of sorts* $s_1,\ldots,s_n$, *let* X *be a finite set of variables, let* ϕ *be a formula and let* $\mathscr{A}$ *be an algebra. We then have*

$$\mathscr{F}_{\mathscr{A}} \models \langle \&X\mathscr{I}pT \textbf{ where } \phi\rangle true.$$

Proof. Let w be a possible world on $\mathscr{A}$ and let I_V be a variable interpretation on $\mathscr{A}$. We must prove $\mathscr{F}_{\mathscr{A}},w,I_V \models \langle \&X\mathscr{I}pT \textbf{ where } \phi\rangle true$. By Definition 2.20, we get that we must find a world $w' \in succ_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)(w)$ such that $\mathscr{F}_{\mathscr{A}},w',I_V \models true$. Define $w' = w\{p \mapsto w(p) \cup \{T^{\mathscr{A},w,I'_V} \mid I'_V \restriction_{Var\setminus X} = I_V \restriction_{Var\setminus X} \text{ and } S,w,I'_V \models \phi\}\}$. Then w' is a possible world on $\mathscr{A}$, so $w' \in \mathscr{F}_{\mathscr{A}}$ and Definition 4.9 gives $(w,w') \in m_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)$, so by Definition 2.19, we have $w' \in succ_{I_V}(\&X\mathscr{I}pT \textbf{ where } \phi)(w)$. Furthermore, the *true* case of Proposition 2.22 gives $\mathscr{F}_{\mathscr{A}},w',I_V \models true$. □

Soundness of axioms (SuccDel), (SuccUpd) and (SuccAss) can be proven in a similar way as the above soundness proof for (SuccIns); these soundness proofs are omitted.

Proposition A.16. *Let X be a finite set of variables, let α be an update program and let ϕ and ψ be formulas such that $FV(\psi) \cap X = \emptyset$. We then have*

$$\models [+X\ \alpha\ \textbf{where}\ \phi]\psi \leftrightarrow \forall X(\phi \rightarrow [\alpha]\psi).$$

Proof. Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w be a world in S and let I_V be a variable interpretation on $\mathscr{A}$. We must prove $S, w, I_V \models [+X\ \alpha\ \textbf{where}\ \phi]\psi \leftrightarrow \forall X(\phi \rightarrow [\alpha]\psi)$. By the $\leftrightarrow$ case of Proposition 2.22, this is equivalent to proving $S, w, I_V \models [+X\ \alpha\ \textbf{where}\ \phi]\psi \Leftrightarrow S, w, I_V \models \forall X(\phi \rightarrow [\alpha]\psi)$. We prove these two directions separately:

$\Rightarrow$: Assume $S, w, I_V \models [+X\ \alpha\ \textbf{where}\ \phi]\psi$. Let I'_V be some variable interpretation with $I'_V \restriction_{Var \setminus X} = I_V \restriction_{Var \setminus X}$. By the $\forall$ case of Proposition 2.22, it now is sufficient to prove $S, w, I'_V \models \phi \rightarrow [\alpha]\psi$. Assume $S, w, I'_V \models \phi$ and let w' be some world in $succ_{I'_V}(\alpha)(w)$. By the $\rightarrow$ case of Proposition 2.22, and by Definition 2.20, it then is sufficient to prove $S, w', I'_V \models \psi$. From $w' \in succ_{I'_V}(\alpha)(w)$, we get by Definition 2.19 that $(w, w') \in m_{I'_V}(\alpha)$. Definition 4.9 gives $(w, w') \in m_{I'_V}(+X\ \alpha\ \textbf{where}\ \phi)$, so $w' \in succ_{I'_V}(+X\ \alpha\ \textbf{where}\ \phi)(w)$. As $S, w, I_V \models [+X\ \alpha\ \textbf{where}\ \phi]\psi$, we derive with Definition 2.20 that $S, w', I_V \models \psi$. Lemma 2.21 finally gives $S, w', I'_V \models \psi$.

$\Leftarrow$: Assume $S, w, I_V \models \forall X(\phi \rightarrow [\alpha]\psi)$. Let $w' \in succ_{I_V}(+X\ \alpha\ \textbf{where}\ \phi)(w)$, Definition 2.20 then gives that it is sufficient to prove $S, w', I_V \models \psi$. From $w' \in succ_{I_V}(+X\ \alpha\ \textbf{where}\ \phi)(w)$, we get by Definition 2.19 that $(w, w') \in m_{I_V}(+X\ \alpha\ \textbf{where}\ \phi)$ and Definition 4.9 gives a variable interpretation I'_V on $\mathscr{A}$ with $I'_V \restriction_{Var \setminus X} = I_V \restriction_{Var \setminus X}$ and $S, w, I'_V \models \phi$ and $(w, w') \in m_{I'_V}(\alpha)$. As $S, w, I_V \models \forall X(\phi \rightarrow [\alpha]\psi)$, the $\forall$ case of Proposition 2.22 gives $S, w, I'_V \models \phi \rightarrow [\alpha]\psi$. With the $\rightarrow$ case of Proposition 2.22, we derive $S, w, I'_V \models [\alpha]\psi$. As $(w, w') \in m_{I'_V}(\alpha)$, Definition 2.19 gives $w' \in succ_{I_V}(\alpha)(w)$ and by Definition 2.20, we derive $S, w', I'_V \models \psi$. Finally, Lemma 2.21 gives $S, w', I_V \models \psi$. □

A.4. Definite database states

Definition 2.24 assumes that for every algebra $\mathscr{A}$ and every definite database state DB, there is a unique possible world on $\mathscr{A}$ (written as $pw_{\mathscr{A}}(DB)$) in which DB is true. We show that this is sound by two lemmas. Lemma A.17 shows that DB is true in $pw_{\mathscr{A}}(DB)$ and Lemma A.18 shows that there are no other worlds that make DB true.

Lemma A.17. *Let $\mathscr{A}$ be an algebra and let DB be a definite database state. Then $pw_{\mathscr{A}}(DB) \models DB$.*

Proof. By Definition 2.20, we must prove $\forall \phi \in DB$: $pw_{\mathscr{A}}(DB) \models \phi$. Let I_V be an arbitrary variable interpretation on $\mathscr{A}$, Definition 2.20 gives that we need to prove $\forall \phi \in DB$: $pw_{\mathscr{A}}(DB), I_V \models \phi$.

First, let us consider the extension formulas of predicate symbols in DB. Let p be an arbitrary element of $UPred$ and let $\forall \vec{x}(p\vec{x} \leftrightarrow (\vec{x} = T_1 \vee \cdots \vee \vec{x} = T_n))$ be the

extension formula of p in DB. Let I'_V be an arbitrary variable interpretation such that $I'_V\restriction_{Var\setminus\bar{x}} = I_V\restriction_{Var\setminus\bar{x}}$ (for convenience, we view the tuple of variables $\bar{x}$ as a set here). By the $\forall$ and $\leftrightarrow$ cases of Proposition 2.22, it then is sufficient to prove $pw_{\mathscr{A}}(DB), I'_V \models p\bar{x} \Leftrightarrow pw_{\mathscr{A}}(DB), I'_V \models \bar{x} = T_1 \vee \cdots \vee \bar{x} = T_n$. This is easily shown as follows.

$pw_{\mathscr{A}}(DB), I'_V \models p\bar{x} \Leftrightarrow$ (Definition 2.20) $\bar{x}^{\mathscr{A},w,I'_V} \in pw_{\mathscr{A}}(DB)(p) \Leftrightarrow$ (Definition 2.24) $\bar{x}^{\mathscr{A},w,I'_V} \in \{T_1^{\mathscr{A}},\ldots,T_n^{\mathscr{A}}\} \Leftrightarrow$ (as $T_1,\ldots,T_n$ are tuples of *constant* terms) $\bar{x}^{\mathscr{A},w,I'_V} \in \{T_1^{\mathscr{A},w,I_{V'}},\ldots,T_n^{\mathscr{A},w,I'_V}\} \Leftrightarrow \exists i\ (1\leqslant i\leqslant n)$: $\bar{x}^{\mathscr{A},w,I'_V} = T_i^{\mathscr{A},w,I_{V'}} \Leftrightarrow$ (Definition 2.20) $\exists i\ (1\leqslant i\leqslant n)$: $pw_{\mathscr{A}}(DB), I'_V \models \bar{x} = T_i \Leftrightarrow$ (Definition 2.20) $pw_{\mathscr{A}}(DB), I'_V \models \bar{x} = T_1 \vee \cdots \vee \bar{x} = T_n$.

Now, consider the extension formulas of function symbols in DB. Let f be an arbitrary element of $UFun$ and let $fT_1 = t_1 \wedge \cdots \wedge fT_m = t_m \wedge \forall\bar{x}\ (\bar{x} \neq T_1 \wedge \cdots \wedge \bar{x} \neq T_m \rightarrow f\bar{x} = t)$ be the extension formula of f in DB. By the $\wedge$ case of Proposition 2.22, it is sufficient to prove for all i with $1\leqslant i\leqslant m$ that $pw_{\mathscr{A}}(DB), I_V \models fT_i = t_i$, and $pw_{\mathscr{A}}(DB), I_V \models \forall\bar{x}\ (\bar{x} \neq T_1 \wedge \cdots \wedge \bar{x} \neq T_m \rightarrow f\bar{x} = t)$.

Take some i with $1\leqslant i\leqslant m$. Definition 2.24 gives $pw(DB)(f)(T_i^{\mathscr{A}}) = t_i^{\mathscr{A}}$. Therefore $pw(DB)(f)(T_i^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V}) = t_i^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V}$ and Definition 2.16 gives $(fT_i)^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V} = t_i^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V}$. By Definition 2.20, we derive $pw_{\mathscr{A}}(DB), I'_V \models fT_i = t_i$.

Let I'_V be some variable interpretation such that $I'_V\restriction_{Var\setminus\bar{x}} = I_V\restriction_{Var\setminus\bar{x}}$ and assume $pw_{\mathscr{A}}(DB), I'_V \models \bar{x} \neq T_1 \wedge \cdots \wedge \bar{x} \neq T_m$. By the $\forall$ and $\rightarrow$ cases of Proposition 2.22, it then is sufficient to prove $pw_{\mathscr{A}}(DB), I'_V \models f\bar{x} = t$. From the assumption, we get with the $\wedge$ and $\neq$ cases of Proposition 2.22 for all i with $1\leqslant i\leqslant m$ that $\bar{x}^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V} \neq T_i^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V}$. Definition 2.24 gives $pw_{\mathscr{A}}(DB)(f)(\bar{x}^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V}) = t^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V}$ and by Definition 2.16 we get $(f\bar{x})^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V} = t^{\mathscr{A},pw_{\mathscr{A}}(DB),I'_V}$. Finally, with Definition 2.20, we derive $pw_{\mathscr{A}}(DB), I'_V \models f\bar{x} = t$. □

Lemma A.18. *Let $\mathscr{A}$ be an algebra, let DB be a definite database state and let w be a possible world on $\mathscr{A}$ such that $w \neq pw_{\mathscr{A}}(DB)$. Then $w \not\models DB$.*

Proof. By Definition 2.20, it is sufficient to prove that there exists a variable interpretation I_V on $\mathscr{A}$ such that $w, I_V \not\models DB$. As DB is a closed set of formulas, we can in fact prove this for any variable interpretation I_V. So let I_V be an arbitrary variable interpretation on $\mathscr{A}$. As $w \neq pw_{\mathscr{A}}(DB)$, we know that there is an updateable predicate symbol p such that $w(p) \neq pw_{\mathscr{A}}(DB)(p)$ or there is an updateable function symbol f such that $w(f) \neq pw_{\mathscr{A}}(DB)(f)$.

First, consider the case that we have an updateable predicate symbol p with $w(p) \neq pw_{\mathscr{A}}(DB)(p)$. Let $\forall\bar{x}(p\bar{x} \leftrightarrow (\bar{x} = T_1 \vee \cdots \vee \bar{x} = T_n))$ be the extension formula of p in DB. Both $w(p)$ and $pw_{\mathscr{A}}(DB)(p)$ are sets of tuples of domain elements. As they are different sets, we know that there is a $\bar{d} \in w(p)$ such that $\bar{d} \notin pw_{\mathscr{A}}(DB)(p)$ or there is a $\bar{d} \in pw_{\mathscr{A}}(DB)(p)$ such that $\bar{d} \notin w(p)$. In both cases, we define the variable interpretation I'_V as $I'_V(\bar{x}) = \bar{d}$ and $I'_V\restriction_{Var\setminus\bar{x}} = I_V\restriction_{Var\setminus\bar{x}}$ (for convenience, we view the sequence $\bar{x}$ as a set in this last condition). With Definitions 2.16 and 2.20, and Proposition 2.22, it then is not difficult to derive that either $w, I_V \models p\bar{x}$ or $w, I'_V \models \bar{x} = T_1 \vee \cdots \vee \bar{x} = T_n$, but

not both. The $\leftrightarrow$ case of Proposition 2.22 gives $w, I'_V \not\models p\bar{x} \leftrightarrow \bar{x} = T_1 \vee \cdots \vee \bar{x} = T_n$; the $\forall$ case of Proposition 2.22 gives $w, I_V \not\models \forall \bar{x}\ (p\bar{x} \leftrightarrow \bar{x} = T_1 \vee \cdots \vee \bar{x} = T_n)$ and therefore, $w, I_V \not\models DB$.

Now, consider the case that we have an updateable function symbol f with $w(f) \neq pw_{\mathscr{A}}(DB)(f)$. Let $fT_1 = t_1 \wedge \cdots \wedge fT_m = t_m \wedge \forall \bar{x}\ (\bar{x} \neq T_1 \wedge \cdots \wedge \bar{x} \neq T_m \rightarrow f\bar{x} = t)$ be the extension formula of f in DB. As $w(f) \neq pw_{\mathscr{A}}(DB)(f)$, we know that there is a tuple of domain elements $\bar{d}$ such that $w(f)(\bar{d}) \neq pw_{\mathscr{A}}(DB)(f)(\bar{d})$. We distinguish two cases:

- There is an i with $1 \leqslant i \leqslant m$ such that $T_i^{\mathscr{A}} = \bar{d}$. We have $w(f)(\bar{d}) = w(f)(T_i^{\mathscr{A}}) = w(f)(T_i^{\mathscr{A}, w, I_V}) =$(Definition 2.16) $(fT_i)^{\mathscr{A}, w, I_V}$. We also have $pw_{\mathscr{A}}(DB)(f)(\bar{d}) = pw_{\mathscr{A}}(DB)(f)(T_i^{\mathscr{A}}) =$(Definition 2.24) $t_i^{\mathscr{A}, w, I_V}$. We get $(fT_i)^{\mathscr{A}, w, I_V} \neq t_i^{\mathscr{A}, w, I_V}$. The $\neq$ case of Proposition 2.22, then gives $w, I_V \not\models fT_i = t_i$.
- For all i with $1 \leqslant i \leqslant m$: $T_i^{\mathscr{A}} \neq \bar{d}$. Define the variable interpretation I'_V as $I'_V(\bar{x}) = \bar{d}$ and $I'_V \restriction_{Var \setminus \bar{x}} = I_V \restriction_{Var \setminus \bar{x}}$. Then Definition 2.16 gives $\bar{x}^{\mathscr{A}, w, I'_V} = \bar{d}$. So for all i with $1 \leqslant i \leqslant m$, we have $\bar{x}^{\mathscr{A}, w, I_V} \neq T_i^{\mathscr{A}} = T_i^{\mathscr{A}, w, I'_V}$. By the $\neq$ and $\wedge$ cases of Proposition 2.22, we get $w, I'_V \models \bar{x} \neq T_1 \wedge \cdots \wedge \bar{x} \neq T_m$. By Definition 2.24, we know that $pw_{\mathscr{A}}(DB)(f)(\bar{d}) = t^{\mathscr{A}}$, so $w(f)(\bar{d}) \neq t^{\mathscr{A}}$. As $I'_V(\bar{x}) = \bar{d}$, Definition 2.16 gives $\bar{x}^{\mathscr{A}, w, I'_V} = \bar{d}$. We get $w(f)(\bar{x}^{\mathscr{A}, w, I'_V}) \neq t^{\mathscr{A}} = t^{\mathscr{A}, w, I'_V}$. Definition 2.16 gives $(f(\bar{x}))^{\mathscr{A}, w, I'_V} \neq t^{\mathscr{A}, w, I'_V}$. With Proposition 2.22, we then get $w, I'_V \not\models f\bar{x} = t$. The $\rightarrow$ and $\forall$ cases of Proposition 2.22 finally give $w, I_V \not\models \forall \bar{x}\ (\bar{x} \neq T_1 \wedge \cdots \wedge \bar{x} \neq T_m \rightarrow f\bar{x} = t)$.

In both cases, $w, I_V \not\models fT_1 = t_1 \wedge \cdots \wedge fT_m = t_m \wedge \forall \bar{x}\ (\bar{x} \neq T_1 \wedge \cdots \wedge \bar{x} \neq T_m \rightarrow f\bar{x} = t)$ immediately follows with the $\wedge$ case of Proposition 2.22, so $w, I_V \not\models DB$. □

The next lemma "links" the functions $pw_{\mathscr{A}}$ (which selects the unique world that corresponds with a database state and *mc* (which denotes the normal form of a closed term occurring in a database state).

Lemma A.19 (Lemma 2.26). *Let DB be a definite database state, let t be a closed term and let $\mathscr{A}$ be a model of E. Then*

$$mc(t, DB)^{\mathscr{A}} = t^{\mathscr{A}, pw_{\mathscr{A}}(DB)}.$$

Proof. By induction on the structure of t:

1. $t = fT$, for f an updateable function symbol. Now $mc(fT, DB)^{\mathscr{A}} =$(Definition 2.25) $(DB(f)(mc(T, DB)))^{\mathscr{A}} =$ (Definition 2.24) $pw_{\mathscr{A}}(DB)(f)(mc(T, DB)^{\mathscr{A}}) =$(by the induction hypothesis) $pw_{\mathscr{A}}(DB)(f)(T^{\mathscr{A}, pw_{\mathscr{A}}(DB)}) =$(Definition 2.16) $(fT)^{\mathscr{A}, pw_{\mathscr{A}}(DB)}$.
2. $t = fT$, for f a non-updateable function symbol. Now $mc(fT, DB)^{\mathscr{A}} =$ (Definition 2.25) $normalize(f(mc(T, DB)))^{\mathscr{A}} =$(in a model of E, a term is equal to its normal form) $(f(mc(T, DB)))^{\mathscr{A}} =$(Definition 2.16) $f^{\mathscr{A}}(mc(T, DB)^{\mathscr{A}}) =$ (induction hypothesis) $f^{\mathscr{A}}(T^{\mathscr{A}, pw_{\mathscr{A}}(DB)}) =$ (Definition 2.16) $(fT)^{\mathscr{A}, pw_{\mathscr{A}}(DB)}$.

As t is a closed term, these are the only two cases we need to consider. □

A.5. Equivalence of the operational and declarative semantics of DDL

Just as we did for RAUL, we prove that the operational semantics of DDL is equivalent to the declarative semantics of relational algebra updates in full structures on initial algebras. As we have the regular operators in DDL, the structure of the soundness and completeness proofs we give here, is similar to the structure of the soundness and completeness proofs of the operational semantics of PDDL.

Lemma A.20. *Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$, let w and w' be elements of S and let I_V be a variable interpretation on $\mathscr{A}$. We then have for every formula ϕ and update program α that*

$$(w,w') \in m_{I_V}(\phi?;\alpha) \Leftrightarrow (w,w) \in m_{I_V}(\phi?) \text{ and } (w,w') \in m_{I_V}(\alpha).$$

Proof. $\Rightarrow$: By semantics of sequential composition, there exists a world $v \in S$ such that $(w,v) \in m_{I_V}(\phi?)$ and $(v,w') \in m_{I_V}(\alpha)$. The semantics of test then gives that v must be equal to w, which proves this case.

$\Leftarrow$: Immediate by the semantics of sequential composition. □

Lemma A.21. *Let $\mathscr{A}$ be an algebra, let S be a structure on $\mathscr{A}$ and I_V be a variable interpretation on $\mathscr{A}$. Then*

$$m_{I_V}(+X\ \alpha \textbf{ where } \phi) = m_{I_V}(+X\phi?;\alpha \textbf{ where } \mathit{true}).$$

Proof. Let w and w' be two arbitrary elements of S, then

$$\begin{aligned}
&(w,w') \in m_{I_V}(+X\ \alpha \textbf{ where } \phi)\\
&\quad\Leftrightarrow\ \exists I'_V : I'_V{\restriction}_{Var\setminus X} = I_V{\restriction}_{Var\setminus X} \text{ and } S,w,I'_V \models \phi \text{ and } (w,w') \in m_{I'_V}(\alpha)\\
&\quad\Leftrightarrow\ \exists I'_V : I'_V{\restriction}_{Var\setminus X} = I_V{\restriction}_{Var\setminus X} \text{ and } (w,w) \in m_{I'_V}(\phi?)\\
&\qquad\text{and } (w,w') \in m_{I'_V}(\alpha)\\
&\quad\Leftrightarrow\ \exists I'_V : I'_V{\restriction}_{Var\setminus X} = I_V{\restriction}_{Var\setminus X} \text{ and } (w,w') \in m_{I'_V}(\phi?;\alpha)\\
&\quad\Leftrightarrow\ \exists I'_V : I'_V{\restriction}_{Var\setminus X} = I_V{\restriction}_{Var\setminus X} \text{ and } S,w,I'_V \models \mathit{true}\\
&\qquad\text{and } (w,w') \in m_{I'_V}(\phi?;\alpha)\\
&\quad\Leftrightarrow\ (w,w') \in m_{I_V}(+X\phi?;\alpha \textbf{ where } \mathit{true}).
\end{aligned}$$

The first equivalence is by the semantics of conditional choice, the second by the semantics of test, the third by Lemma A.20, the fourth by the *true* case of Proposition 2.22 and the fifth again by the semantics of conditional choice. □

Lemma A.22. *Let $\mathscr{A}$ be an algebra, let I_V be a variable interpretation on $\mathscr{A}$, let w be a possible world on $\mathscr{A}$, let $\bar{x}$ be a tuple of variables, let t be an $\bar{x}$-closed term and let $\bar{t}$ be a tuple of closed terms of the same sorts as $\bar{x}$ such that $I_V(\bar{x}) = \bar{t}^{\mathscr{A},w}$. Then*

$$(t[\bar{t}/\bar{x}])^{\mathscr{A},w} = t^{\mathscr{A},w,I_V}.$$

Proof. The lemma can be proven in a completely straightforward way by induction on the structure of t; details of the proof are omitted. □

Definition A.23. A DDL update program is called **safe** iff for all atomic updates $\&X\mathscr{I}\ pT$ **where** ϕ, $\&X\mathscr{D}\ pT$ **where** ϕ and $\&X\mathscr{U}\ pT \to T'$ **where** ϕ and conditional choices $+X\ \alpha$ **where** ϕ that occur in the update program, we have that ϕ is an X-safe formula.

Lemma A.24. *Let $\mathscr{A}$ be an initial algebra of E, let S be a structure on $\mathscr{A}$ let w be a possible world in S, let I_V be a variable interpretation on $\mathscr{A}$, let DB be a definite database state and let α be a safe DDL update program such that $(pw_{\mathscr{A}}(DB), w) \in m_S(\alpha)(I_V)$. Then there is a definite database state DB' such that $pw_{\mathscr{A}}(DB') = w$.*

Proof. By induction on the structure of α. The $\alpha = \alpha_1 + \alpha_2$, $\alpha = \alpha_1; \alpha_2$, $\alpha = \beta^*$ and $\alpha = +X\ \beta$ **where** ϕ cases follow in a completely straightforward way from the induction hypothesis; details of the proofs of these cases are omitted. We also omit the proofs for the $\alpha = \&X\mathscr{D}pT\phi$ **where** and $\alpha = \&X\mathscr{U}\,pt \to T'$ **where** ϕ cases, as these cases can be proven in a similar way as the $\&X\mathscr{I}pT$ **where** ϕ case. The only case we prove explicitly is the $\alpha = \&X\mathscr{I}pT$ **where** ϕ case. From $(pw_{\mathscr{A}}(DB), w) \in m_S(\&X\mathscr{I}pT$ **where** $\phi)(I_V)$, we get by Definition 4.9 that $w\restriction_{USym\setminus\{p\}} = pw_{\mathscr{A}}(DB)\restriction_{USym\setminus\{p\}}$ and $w(p) = pw_{\mathscr{A}}(DB)(p) \cup \{I'_V(T) \mid I'_V\restriction_{Var\setminus X} = I_V\restriction_{Var\setminus X}$ and $S, pw_{\mathscr{A}}(DB), I'_V \models \phi\}$. As α is safe, we know that ϕ is an X-safe formula. So $\phi = p_1T_1 \wedge \cdots \wedge p_nT_n \wedge \phi'$. We show that $w(p)$ is finite, by showing that there are only finitely many variable interpretations I'_V that satisfy the requirements $I'_V\restriction_{Var\setminus X} = I_V\restriction_{Var\setminus X}$ and $S, pw_{\mathscr{A}}(DB), I'_V \models \phi$. Consider some variable $x \in X$. As ϕ is X-safe, we know that there exist i and j such that x occurs as the jth component of p_i in $\phi = p_1T_1 \wedge \cdots \wedge p_nT_n \wedge \phi'$. Now suppose $I'_V(x) \notin pw_{\mathscr{A}}(DB)(p_i)[j]$. With Definition 2.16 and Proposition 2.22, we then easily derive $S, pw_{\mathscr{A}}(DB), I'_V \not\models \phi$. So to satisfy the requirements, $I'_V(x)$ must be an element of $pw_{\mathscr{A}}(DB)(p_i)[j]$. By Definition 2.23, we know that $DB(p_i)$ is finite and with Definition 2.24, we get that $pw_{\mathscr{A}}(DB)(p_i)[j]$ is also finite. This is true for all $x \in X$ and as X is finite, there are only finitely many variable interpretations I'_V that satisfy all requirements.

Define $D = \{T \mid T$ is a tuple of immutable terms in normal form of the arity of p such that $T^{\mathscr{A}} \in w(p)\}$. As $\mathscr{A}$ is an initial algebra of E, we have that every element of $w(p)$ is "named" by a tuple of immutable terms in normal form, so $w(p) = \{T^{\mathscr{A}} \mid T \in D\}$. As different immutable terms in normal form have a different semantics in the initial algebra, we get that D contains exactly the same number of elements as $w(p)$ (so D is finite). We define $DB' = DB\{p \mapsto D\}$. We then have $DB'\restriction_{USym\setminus\{p\}} = DB\restriction_{USym\setminus\{p\}}$, so Definition 2.24 gives that $pw_{\mathscr{A}}(DB')\restriction_{USym\setminus\{p\}} = pw_{\mathscr{A}}(DB)\restriction_{USym\setminus\{p\}}$ and we derive $pw_{\mathscr{A}}(DB')\restriction_{USym\setminus\{p\}} = w\restriction_{USym\setminus\{p\}}$. By Definition 2.24, we get $pw_{\mathscr{A}}(DB')(p) = \{T^{\mathscr{A}} \mid T \in DB(p)\} = \{T^{\mathscr{A}} \mid T \in D\} = w(p)$. So we indeed have $pw_{\mathscr{A}}(DB) = w$. □

Lemma A.25. *Let $\mathscr{A}$ be an initial algebra of E, let I_V be a variable interpretation on $\mathscr{A}$, let $\bar{x} = (x_1, \ldots, x_n)$ be a finite tuple of different variables, let ϕ be an $\bar{x}$-safe, $\bar{x}$-closed formula and let DB and DB' be definite database states such that $\langle DB, q := ca(\bar{x}, \phi)\rangle \to DB'$. Then*

$$pw_{\mathscr{A}}(DB), I_V \models \phi \Leftrightarrow I_V(\bar{x}) \in pw_{\mathscr{A}}(DB')(q).$$

Proof. As ϕ is an $\bar{x}$-safe formula, we know that it has the form $p_1T_1 \wedge \cdots \wedge p_nT_n \wedge \phi'$, and for every variable $x \in X$ there is a predicate symbol p_x in the sequence $p_1, \ldots, p_n$ and a position k_x such that x occurs as the k_xth argument of p_x. Definition 4.19 then gives that $ca(\bar{x}, \phi) = (p_{x_1}[k_{x_1}] \times \cdots \times p_{x_n}[k_{x_n}])$ **where** $\phi[\mathcal{N}/\bar{x}]$. From $\langle DB, q := ca(\bar{x}, \phi)\rangle \rightarrow DB'$, we get by soundness of the operational semantics op RAUL that $(pw_{\mathcal{A}}(DB), pw_{\mathcal{A}}(DB')) \in m_{\mathcal{F}_{\mathcal{A}}}(q := ca(\bar{x}, \phi))$. So by Definition 3.8, $pw_{cal.\mathcal{A}}(DB')(q) = pw_{\mathcal{A}}(DB)(ca(\bar{x}, \phi))$. We now prove both directions of the equivalence separately.

$\Rightarrow$: Suppose $pw_{\mathcal{A}}(DB), I_V \models \phi$. Lemma A.3 gives $\bar{x}^{\mathcal{A}, pw_{\mathcal{A}}(DB), I_V} \models \phi[\mathcal{N}/\bar{x}]$ and Definition 2.16 then gives $I_V(\bar{x}) \models \phi[\mathcal{N}/\bar{x}]$.

Now consider some variable x_i $(1 \leqslant i \leqslant n)$. Let T_j contain the first occurrence of x_i as a component in the sequence $T_1, \ldots, T_n$. The $\wedge$ case of Proposition 2.22 then gives $pw_{\mathcal{A}}(DB), I_V \models p_{x_i}T_j$. Definition 2.20 gives $T_j^{\mathcal{A}, pw_{\mathcal{A}}(DB), I_V} \in pw_{\mathcal{A}}(DB)(p_{x_i})$. As the k_{x_i}th component of T_j is x_i, we derive $x_i^{\mathcal{A}, pw_{\mathcal{A}}(DB), I_V} \in pw_{\mathcal{A}}(DB)(p_{x_i})[k_{x_i}]$. Using Definitions 2.16 and 3.7, we get $I_V(x_i) \in pw_{\mathcal{A}}(DB)(p_{x_i}[k_{x_i}])$. But this is true for all i with $1 \leqslant i \leqslant n$, so we derive $I_V(\bar{x}) \in pw_{\mathcal{A}}(DB)(p_{x_1}[k_{x_1}]) \times \cdots \times pw_{\mathcal{A}}(DB)(p_{x_n}[k_{x_n}])$. Definition 3.7 then gives $I_V(\bar{x}) \in pw_{\mathcal{A}}(DB)(p_{x_1}[k_{x_1}] \times \cdots \times p_{x_n}[k_{x_n}])$.

By Definition 3.7, we get that $I_V(\bar{x}) \in pw_{\mathcal{A}}(DB)((p_{x_1}[k_{x_1}] \times \cdots \times p_{x_n}[k_{x_n}])$ **where** $\phi[\mathcal{N}/\bar{x}])$. Therefore, we get $I_V(\bar{x}) \in pw_{\mathcal{A}}(DB)(ca(\bar{x}, \phi))$, so $I_V(\bar{x}) \in pw_{\mathcal{A}}(DB')(q)$.

$\Leftarrow$: Suppose $I_V(\bar{x}) \in pw_{\mathcal{A}}(DB')(q)$. Then $I_V(\bar{x}) \in pw_{\mathcal{A}}(DB)(ca(\bar{x}, \phi))$, so $I_V(\bar{x}) \in pw_{\mathcal{A}}(DB)((p_{x_1}[k_{x_1}] \times \cdots \times p_{x_n}[k_{x_n}])$ **where** $\phi[\mathcal{N}/\bar{x}])$. With Definition 3.7, we derive $I_V(\bar{x}) \models \phi[\mathcal{N}/\bar{x}]$. Definition 2.16 then gives $\bar{x}^{\mathcal{A}, pw_{\mathcal{A}}(DB), I_V} \models \phi[\mathcal{N}/\bar{x}]$. Lemma A.3 finally gives $pw_{\mathcal{A}}(DB), I_V \models \phi$. □

Lemma A.26. *Let $\mathcal{A}$ be an initial algebra of E and let α be a closed update program. Then for all definite database states DB and DB' with $\langle DB, \alpha\rangle \rightarrow DB'$, we have $(pw_{\mathcal{A}}(DB), pw_{\mathcal{A}}(DB')) \in m_{\mathcal{F}_{\mathcal{A}}}(\alpha)$.*

Proof. By induction on the length of the derivation of $\langle DB, \alpha\rangle \rightarrow DB'$. Below, we consider the five rules that allow us to conclude transitions of the format $\langle DB, \alpha\rangle \rightarrow DB'$. The rules (TIt1) and (TAss) form the basic step of the induction, as they do not have premises. The rules (TIns), (TDel) and (TUpd) form the inductive step, as they do have premises. In all cases, we must prove $(pw_{\mathcal{A}}(DB), pw_{\mathcal{A}}(DB')) \in m_{\mathcal{F}_{\mathcal{A}}}(\alpha)$. So we take an arbitrary variable interpretation I_V and we must prove $(pw_{\mathcal{A}}(DB), pw_{\mathcal{A}}(DB') \in m_{\mathcal{F}_{\mathcal{A}}, I_V}(\alpha)$.

(TIt1) Now $\alpha = \beta^*$ and $DB' = DB$. As $m_{\mathcal{F}_{\mathcal{A}}, I_V}(\beta^*)$ is a reflexive relation, we immediately get $(pw_{\mathcal{A}}(DB), pw_{\mathcal{A}}(DB')) \in m_{\mathcal{F}_{\mathcal{A}}, I_V}(\beta^*)$.

(TAss) Now we have $\alpha = fT := t$ and $DB' = DB\{f \mapsto DB(f)\{mc(T, DB) \mapsto mc(t, DB)\}\}$. We get $DB' \restriction_{USym\setminus\{f\}} = DB \restriction_{USym\setminus\{f\}}$. Definition 2.24 gives $pw_{\mathcal{A}}(DB') \restriction_{USym\setminus\{f\}} = pw_{\mathcal{A}}(DB) \restriction_{USym\setminus\{f\}}$. Below, we prove $pw_{\mathcal{A}}(DB')(f) = pw_{\mathcal{A}}(DB)(f)\{T^{\mathcal{A}, pw_{\mathcal{A}}(DB), I_V} \mapsto t^{\mathcal{A}, pw_{\mathcal{A}}(DB), I_V}\}$. Definition 4.9 then gives $(pw_{\mathcal{A}}(DB), pw_{\mathcal{A}}(DB')) \in m(fT := t)(I_V)$. We now prove that the functions $pw_{\mathcal{A}}(DB')(f)$ and $pw_{\mathcal{A}}(DB)(f)\{T^{\mathcal{A}, pw_{\mathcal{A}}(DB), I_V} \mapsto t^{\mathcal{A}, pw_{\mathcal{A}}(DB), I_V}\}$ are the same. As $\mathcal{A}$ is an

initial algebra of E, we know that we get all arguments for the functions by considering all $T'^{\mathscr{A}}$, for T' a tuple of immutable terms in normal form (of the argument sorts of f). So let T' be a tuple of immutable terms in normal form of the argument sorts of f. We consider two cases:

- $T' = mc(T, DB)$. Now $T'^{\mathscr{A}} = mc(T, DB)^{\mathscr{A}}$=(Lemma A.19) $T^{\mathscr{A}, pw_{\mathscr{A}}(DB)} = T^{\mathscr{A}, pw_{\mathscr{A}}(DB), I_V}$. Then $pw_{\mathscr{A}}(DB')(f)(T'^{\mathscr{A}}) =$ (Definition 2.24) $(DB'(f)(T'))^{\mathscr{A}} = (DB'(f)(mc(T, DB))^{\mathscr{A}} = (mc(t, DB))^{\mathscr{A}} =$(Lemma A.19) $t^{\mathscr{A}, pw_{\mathscr{A}}(DB)} = t^{\mathscr{A}, pw_{\mathscr{A}}(DB), I'_V} = pw_{\mathscr{A}}(DB)(f)\{T^{\mathscr{A}, pw_{\mathscr{A}}(DB), I'_V} \mapsto t^{\mathscr{A}, pw_{\mathscr{A}}(DB), I'_V}\}(T'^{\mathscr{A}})$.
- $T' \neq mc(T, DB)$. As $\mathscr{A}$ is an initial algebra of E and both T' and $mc(T, DB)$ are tuples of immutable terms in normal form, we get $T'^{\mathscr{A}} \neq mc(T, DB)^{\mathscr{A}}$. Furthermore, $mc(T, DB)^{\mathscr{A}}$=(Lemma A.19) $T^{\mathscr{A}, pw_{\mathscr{A}}(DB)} = T^{\mathscr{A}, pw_{\mathscr{A}}(DB), I_V}$, so $T'^{\mathscr{A}} \neq T^{\mathscr{A}, pw_{\mathscr{A}}(DB), I_V}$. Then $pw_{\mathscr{A}}(DB')(f)(T'^{\mathscr{A}}) =$ (Definition 2.24) $(DB'(f)(T'))^{\mathscr{A}} = (DB(f)(T'))^{\mathscr{A}}$=(Definition 2.24) $pw_{\mathscr{A}}(DB)(f)(T'^{\mathscr{A}}) = pw_{\mathscr{A}}(DB)(f)\{T^{\mathscr{A}, pw_{\mathscr{A}}(DB), I_V} \mapsto t^{\mathscr{A}, pw_{\mathscr{A}}(DB), I_V}\}(T'^{\mathscr{A}})$.

(TIns) Now $\alpha = \&X\mathscr{I}pT$ **where** ϕ, and there is a definite database state DB'' such that $\langle DB, q := ca(\bar{x}, \phi)\rangle \to DB''$ and $DB' = DB\{p \mapsto DB(p) \cup \{mc(T[\bar{t}/\bar{x}], DB) \mid \bar{t} \in DB''(q)\}\}$. Now $DB' \restriction_{USym\setminus\{p\}} = DB \restriction_{USym\setminus\{p\}}$ and with Definition 2.24, we get $pw_{\mathscr{A}}(DB') \restriction_{USym\setminus\{p\}} = pw_{\mathscr{A}}(DB) \restriction_{USym\setminus\{p\}}$. We want to prove that $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}, I_V}(\&X\mathscr{I}pT$ **where** $\phi)$, by Definition 4.9, it is sufficient to prove $pw_{\mathscr{A}}(DB')(p) = pw_{\mathscr{A}}(DB)(p) \cup \{T^{\mathscr{A}, pw_{\mathscr{A}}(DB), I_V} \mid pw_{\mathscr{A}}(DB), I_V \models \phi\}$. Proven as follows: $pw_{\mathscr{A}}(DB')(p)$=(Definition 2.24) $\{T'^{\mathscr{A}} \mid T' \in DB'(p)\} = \{T'^{\mathscr{A}} \mid T' \in DB(p) \cup \{mc(T[\bar{t}/\bar{x}], DB) \mid \bar{t} \in DB''(q)\}\} = \{T'^{\mathscr{A}} \mid T' \in DB(p)\} \cup \{T'^{\mathscr{A}} \mid T' \in \{mc(T[\bar{t}/\bar{x}], DB) \mid \bar{t} \in DB''(q)\}\}$=(Definition 2.24) $pw_{\mathscr{A}}(DB)(p) \cup \{(mc(T[\bar{t}/\bar{x}], DB))^{\mathscr{A}} \mid \bar{t} \in DB''(q)\} =$ (Lemma A.19) $pw_{\mathscr{A}}(DB)(p) \cup \{(T[\bar{t}/\bar{x}])^{\mathscr{A}, pw_{\mathscr{A}}(DB)} \mid \bar{t} \in DB''(q)\} =$ (Lemma A.22) $pw_{\mathscr{A}}(DB)(p) \cup \{T^{\mathscr{A}, pw_{\mathscr{A}}(DB), I_V} \mid I_V(\bar{x}) = \bar{t}^{\mathscr{A}, pw_{\mathscr{A}}(DB)}$ and $\bar{t} \in DB''(q)\} =$ (2.24) $pw_{\mathscr{A}}(DB)(p) \cup \{T^{\mathscr{A}, pw_{\mathscr{A}}(DB), I_V} \mid I_V(\bar{x}) \in pw_{\mathscr{A}}(DB'')(q)\}$=(Lemma A.25) $pw_{\mathscr{A}}(DB)(p) \cup \{T^{\mathscr{A}, pw_{\mathscr{A}}(DB), I_V} \mid pw_{\mathscr{A}}(DB), I_V \models \phi\}$.

(TDel) The proof of this case is very similar to the proof of the (TIns) case and has been omitted.

(TUpd) The proof of this case is very similar to the proof of the (TIns) case and has been omitted. □

Lemma A.27. *Let $\mathscr{A}$ be an initial algebra of E and let α and α' be closed update programs. Then for all definite database states DB, DB' and DB'' with $\langle DB, \alpha\rangle \to \langle DB'', \alpha'\rangle$ and $(pw_{\mathscr{A}}(DB''), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha')$, we have*

$$(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha).$$

Proof. By induction on the length of the derivation of $\langle DB, \alpha\rangle \to \langle DB'', \alpha'\rangle$. Below, we distinguish six cases; corresponding to the transition rules with which we can derive transitions of the form $\langle DB, \alpha\rangle \to \langle DB'', \alpha'\rangle$. Rules (TCh1), (TCh2), (TSeq1) and (TIt2)

are form the basic step of the inductive proof and rules (TSeq2) and (TCondC) form the induction step:

(TCh1) Now $DB'' = DB$ and there are closed update programs α_1 and α_2 such that $\alpha = \alpha_1 + \alpha_2$ and $\alpha' = \alpha_1$. So we have $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1)$. We derive $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1) \cup m_{\mathscr{F}_{\mathscr{A}}}(\alpha_2)$ and Definition 4.9 then gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1 + \alpha_2)$.

(TCh2) The proof of this case is omitted, as it is almost the same as the proof of the (TCh1) case.

(TSeq1) Now, there are closed update programs α_1 and α_2 such that $\alpha = \alpha_1; \alpha_2$, $\alpha' = \alpha_2$ and $\langle DB, \alpha_1 \rangle \to DB''$. Using this last transition, Lemma A.26 gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB'')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1)$. We also have $(pw_{\mathscr{A}}(DB''), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_2)$, so Definition 4.9 gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1; \alpha_2)$.

(TSeq2) Now, there are closed update programs α_1, α_2 and α_1' with $\alpha = \alpha_1; \alpha_2$, $\alpha' = \alpha_1'; \alpha_2$ and $\langle DB, \alpha_1 \rangle \to \langle DB'', \alpha_1' \rangle$. From $(pw_{\mathscr{A}}(DB''), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1'; \alpha_2)$, Definition 4.9 gives us a world w such that $(pw_{\mathscr{A}}(DB''), w) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1')$ and $(w, pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_2)$. With world w, there corresponds a definite database state DB''' such that $pw_{\mathscr{A}}(DB''') = w$. So we have $\langle DB, \alpha_1 \rangle \to \langle DB'', \alpha_1' \rangle$ and $(pw_{\mathscr{A}}(DB''), pw_{\mathscr{A}}(DB''')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1')$. The induction hypothesis gives $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB''')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1)$. We also have $(pw_{\mathscr{A}}(DB'''), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_2)$, so by Definition 4.9, we get $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1; \alpha_2)$.

(TIt2) Now $DB'' = DB$ and there is a closed update program β such that $\alpha = \beta^*$ and $\alpha' = \beta; \beta^*$. So we have $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\beta; \beta^*)$. From Definition 4.9, it is easy to derive $m_{\mathscr{F}_{\mathscr{A}}}(\beta; \beta^*) \subseteq m_{\mathscr{F}_{\mathscr{A}}}(\beta^*)$ and we get $(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\beta^*)$.

(TCondC) Now $DB'' = DB$ and there is an X-closed update program α, an X-closed formula ϕ and an X-assignment A such that $\alpha = +X\ \alpha$ **where** ϕ and $\alpha' = A(\phi?; \alpha)$. This case now immediately follows from Lemmas A.21. □

Proposition A.28 (Soundness of the operational semantics of DDL). *Let $\mathscr{A}$ be an initial algebra of E and let α be a closed* DDL *update program. Then for all definite database states DB and DB' with $\langle DB, \alpha \rangle \to^* DB'$, we have*

$$(pw_{\mathscr{A}}(DB), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha).$$

Proof. By induction on the number of steps in $\langle DB, \alpha \rangle \to DB'$.

Basic step: $\langle DB, \alpha \rangle \to DB'$. The desired result now immediately follows from Lemma A.26.

Induction step: $\langle DB, \alpha \rangle \to \langle DB'', \alpha' \rangle \to^* DB'$. By the induction hypothesis on $\langle DB'', \alpha' \rangle \to^* DB'$, we get $(pw_{\mathscr{A}}(DB''), pw_{\mathscr{A}}(DB')) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha')$. With Lemma A.27, we immediately get the desired result. □

Lemma A.29. *Let DB' and DB'' be definite database states and let α be a closed* DDL *update program with $\langle DB'', \alpha\rangle \to^* DB'$. Then for all closed* DDL *update programs β and definite database states DB with $\langle DB, \beta\rangle \to^* DB''$ we have $\langle DB, \beta; \alpha\rangle \to^* DB'$.*

Proof. By induction on the number of steps in $\langle DB, \beta\rangle \to^* DB''$.

Basic step: $\langle DB, \beta\rangle \to DB''$. Using transition rule (TSeq1) we get $\langle DB, \beta; \alpha\rangle \to \langle DB'', \alpha\rangle$. Then with $\langle DB'', \alpha\rangle \to^* DB'$, we get $\langle DB, \beta; \alpha\rangle \to^* DB'$.

Induction step: $\langle DB, \beta\rangle \to \langle DB''', \gamma\rangle \to^* DB''$. The induction hypothesis on $\langle DB''', \gamma\rangle \to^* DB''$ gives $\langle DB''', \gamma; \alpha\rangle \to^* DB'$. Transition rule (TSeq2) applied to $\langle DB, \beta\rangle \to \langle DB''', \gamma\rangle$ gives us $\langle DB, \beta; \alpha\rangle \to \langle DB''', \gamma; \alpha\rangle$, so we get $\langle DB, \beta; \alpha\rangle \to^* DB'$. □

Proposition A.30 (Completeness of the operational semantics of DDL). *Let $\mathscr{A}$ be an initial algebra of E and let α be a closed* DDL *update program. Then for all definite database states DB and possible worlds w' on $\mathscr{A}$ with $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha)$, there is a definite database state DB' such that $\langle DB, \alpha\rangle \to^* DB'$ and $w' = pw_{\mathscr{A}}(DB')$.*

Proof. By induction on the structure of α:

1. $\alpha = \&X\mathscr{I}pT$ **where** ϕ Take $DB' = DB\{p \mapsto DB(p) \cup mc(T[\bar{t}/\bar{x}], DB) \mid \bar{t} \in ca(\bar{x}, \phi)$. With (TIns), we then derive $\langle DB, \&X\mathscr{I}pT$ **where** $\phi\rangle \to^* DB'$.
2. $\alpha = \&X\mathscr{D}pT\phi$ **where** . The proof of this case is similar to case 1 and omitted.
3. $\alpha = \&X\mathscr{U}pT \to T'$ **where** ϕ. The proof of this case is similar to case 1 and omitted.
4. $\alpha = \alpha_1; \alpha_2$. We have $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1; \alpha_2)$, so Definition 2.18 gives a world v with $(pw_{\mathscr{A}}(DB), v) \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1)$ and $(v, w') \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_2)$. The induction hypothesis on α_1 gives a definite database state DB'' such that $\langle DB, \alpha_1\rangle \to^* DB''$ and $v = pw_{\mathscr{A}}(DB'')$. The induction hypothesis on α_2 then gives definite database state DB' such that $\langle DB'', \alpha_2\rangle \to^* DB'$ and $w' = pw_{\mathscr{A}}(DB')$. By Lemma A.29, we get $\langle DB, \alpha_1; \alpha_2\rangle \to^* DB'$, so DB' satisfied both the desired properties.
5. $\alpha = \alpha_1 + \alpha_2$. We have $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1 + \alpha_2)$, so Definition 2.18 gives $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1) \cup m_{\mathscr{F}_{\mathscr{A}}}(\alpha_2)$. We only consider the case $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_1)$ (the case $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(\alpha_2)$ can be proven similarly). By the induction hypothesis on α_1 we get a definite database state DB' with $\langle DB, \alpha_1\rangle \to^* DB'$ and $w' = pw_{\mathscr{A}}(DB')$. Transition rule (TCh1) gives $\langle DB, \alpha_1 + \alpha_2\rangle \to \langle DB, \alpha_1\rangle$, so $\langle DB, \alpha_1 + \alpha_2\rangle \to DB'$, so DB' satisfied both the desired properties.
6. $\alpha = \beta^*$. We have $(pw_{\mathscr{A}}(DB), w') \in m_{\mathscr{F}_{\mathscr{A}}}(\beta^*)$, so Definition 2.18 gives an $n \geqslant 1$ and worlds $w_1, w_2, \ldots, w_n$ such that $w_1 = pw_{\mathscr{A}}(DB)$ and $w_n = w'$ and for all i $(1 \leqslant i < n)$: $(w_1, w_{i+1}) \in m_{\mathscr{F}_{\mathscr{A}}}(\beta)$. By induction on n we now prove that there is a definite database state DB_n such that $\langle DB, \beta^*\rangle \to^* DB_n$ and $w_n = pw_{\mathscr{A}}(DB_n)$ (and this then proves the $\alpha = \beta^*$ case, because DB_n satisfied both desired properties). Basic step $n = 1$. Take $DB_1 = DB$, then transition rule (TIter1) gives $\langle DB, \beta^*\rangle \to DB_1$ and $w_n = w_1 = pw_{\mathscr{A}}(DB) = pw_{\mathscr{A}}(DB_1)$. Induction step $n > 1$. We have $w_1 = pw_{\mathscr{A}}(DB_1)$, so $(pw_{\mathscr{A}}(DB_1), w_2) \in m_{\mathscr{F}_{\mathscr{A}}}(\beta)$, so the induction hypothesis on β gives a definite database state DB_2 with $\langle DB_1, \beta\rangle \to^* DB_2$ and $w_2 = pw_{\mathscr{A}}(DB_2)$. The induction hypothesis on n then gives a definite database state DB_n such that $\langle DB_2, \beta^*\rangle \to^* DB_n$

and $w_n = pw_{\mathscr{A}}(DB_n)$. Lemma A.29 then gives us $\langle DB_1, \beta; \beta^* \rangle \rightarrow^* DB_n$. Transition rule (TIter2) gives $\langle DB_1, \beta^* \rangle \rightarrow \langle DB_1, \beta; \beta^* \rangle$. We get $\langle DB_1, \beta^* \rangle \rightarrow DB_n$, so DB_n satisfies both desired properties. □

References

[1] S. Abiteboul, V. Vianu, A transaction language complete for database update and specification, in: Proc. 6th ACM SIGACT-SIGMOD Symp. on the Principles of Database Systems, San Diego, CA, March 23–25 1987, pp. 260–268.
[2] S. Abiteboul, V. Vianu, Procedural and declarative database update languages, in: Proc. 7th ACM SIGACT-SIGMOD Symp. on the Principles of Database Systems, Austin, TX, ACM, New York, March 21–23 1988, pp. 240–250.
[3] C.E. Alchourrón, P. Gärdenfors, D. Makinson, On the logic theory of change: partial meet contraction and revision functions, J. Symbolic Logic 50 (1985) 510–530.
[4] A.J. Bonner, M. Kifer, Transaction logic programming, Tech. Report CSRI-270, University of Toronto, April 1992.
[5] A.J. Bonner, M. Kifer, Transaction logic: an (early) exposé, in: Proc. Workshop on Formal Methods in Databases and Software Engineering, Workshops in Computing Series, Springer, Berlin, 1993.
[6] A.J. Bonner, M. Kifer, Transaction logic programming, in: Proc. 10th Internat. Conf. on Logic Programming (ICLP) Budapest, Hungary, June 1993, pp. 257–279.
[7] A.J. Bonner, M. Kifer, An overview of transaction logic, Theoret. Comput. Sci. 133 (1994) 205–265.
[8] A.J. Bonner, M. Kifer, M. Consens, Database programming in transaction logic, in: Proc. 4th Internat. Workshop on Database Programming Languages (DBPL), New York City, Aug/Sep 1993, Workshops in Computing, Springer, Berlin, 1994, pp. 309–337.
[9] S. Ceri, G. Gottlob, L. Tanca, Logic Programming and Databases, Springer, Berlin, 1990.
[10] L. Cholvy, Update semantics under the domain closure assumption, in: M. Gyssens, J. Paredaens, D. van Gucht (Eds.), Internat. Conf. on Database Theory, Lecture Notes in Computer Science, vol. 326, Springer, Berlin, 1988, pp. 123–140.
[11] M. Dalal, Investigations into a theory of knowledge base revision: preliminary report, Proc. AAAI, vol. 2, 1988, pp. 475–479.
[12] C.J. Date, An Introduction to Database Systems, 5th ed., vol. 1, Addison-Wesley, Reading, MA, 1990.
[13] N. Dershowitz, J.-P. Jouannaud, Rewrite systems, in: Jan van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Formal Models and Semantics, vol. B, North-Holland, Amsterdam, 1990, pp. 243–320.
[14] F. Dignum, R.P. van de Riet, Addition and removal of information for a knowledge base with incomplete information, Data Knowledge Eng. 8(4) (1992) 293–307.
[15] D. Fensel, R. Groenboom, A formal semantics and axiomatization for specifying the dynamics of knowledge-based systems, Tech. Report, University of Amsterdam and University of Groningen, 1996.
[16] D. Fensel, R. Groenboom, MLPM: defining a semantics and axiomatization for specifying the reasoning process of knowledge-based systems, in: W. Wahlster (Ed.), 12th European Conf. on Artificial Intelligence, 1996.
[17] H. Gallaire, J. Minker (Eds.), Logic and Databases, Plenum Press, New York, 1978.
[18] P. Gärdenfors, D. Makinson, Revisions of knowledge system using epistemic entrenchment, in: M. Vardi (Ed.), Proc. 2nd Conf. on Theoretical Aspects of Reasoning about Knowledge, Morgan Kaufmann, Los Altos, CA, 1988, pp. 83–95.
[19] F. Golshani, T.S.E. Maibaum, M.R. Sadler, A modal system of algebras for database specification and query/update support, Proc. Nineth Internat. Conf. on Very Large Databases, 1983, pp. 331–359.
[20] R. Groenboom, G.R. Renardel de Lavalette, Reasoning about dynamic features in specification languages, in: D.J. Andrews, J.F. Groote, C.A. Middelburg (Eds.), Proc. Workshop in Semantics of Specification Languages, Springer, Berlin, 1994.
[21] D. Harel, in: First Order Dynamic Logic, Lecture Notes in Computer Science, vol. 68, Springer, Berlin, 1979.
[22] G.E. Hughes, M.J. Cresswell, An Introduction to Modal Logic, Methuen, London, 1968, reprint 1985.

[23] H. Katsuno, A.O. Mendelzon, On the difference between updating a knowledge base and revising it, in: J. Allen, R. Fikes, E. Sandewall (Eds.), Principles of Knowledge Representation and Reasoning, Massachusetts, Proceedings of the Second Internat. Conf., Morgan Kaufmann, Los Altos, CA, April 22–25 1991, pp. 387–394.
[24] S. Khosla, T.S.E. Maibaum, M. Sadler, Database specification, in: T.B. Steel Jr., R. Meersman (Eds.), Database Semantics (DS-1), North-Holland, Amsterdam, 1986, pp. 141–158.
[25] J.W. Klop, Term rewriting systems, in: S. Abramsky, Dov. M. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science, Background – Computational Structures, vol. 2, Oxford University Press, Oxford, 1992, pp. 1–116.
[26] D. Kozen, J. Tiuryn, Logics of programs, in: Jan van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier, Amsterdam, 1990, pp. 789–840.
[27] S. Manchanda, D.S. Warren, A logic-based language for database updates, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, CA, 1988, pp. 363–394.
[28] S. Naqvi, R. Krishnamurthy, Database updates in logic programming, in: Proc 7th ACM SIGACT-SIGMOD Symp. on the Principles of Database Systems, Austin, TX, ACM, New York, March 21–23 1988, pp. 251–262.
[29] S. Naqvi, S. Tsur, A Logical Language for Data and Knowledge Bases, Computer Science Press, Rockville, MD, 1989.
[30] F. Nourani, On induction for programming logic: syntax, semantics, and inductive closure, Bull. EATCS 13 (1981) 51–64.
[31] G.D. Plotkin, A structural approach to operational semantics, Tech. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981, reprinted 1991.
[32] R. Reiter, Towards a logical reconstruction of relational database theory, in: M.L. Brodie, J. Mylopoulos, J.W. Schmidt (Eds.), On Conceptual Modelling, Springer, Berlin, 1984, pp. 191–233.
[33] P.A. Spruit, Logics of Database Updates, Ph.D. Thesis, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1994.
[34] P.A. Spruit, R.J. Wieringa, J.-J.Ch. Meyer, Dynamic database logic: the first-order case, in: U.W. Lipeck, B. Thalheim (Eds.), Modelling Database Dynamics, Springer, Berlin, 1993, pp. 103–120.
[35] P.A. Spruit, R.J. Wieringa, J.-J.Ch. Meyer, Axiomatization, declarative semantics and operational semantics of passive and active updates in logic databases, J. Logic Comput. 5(1) (1995) 27–50.
[36] J.D. Ullman, Principles of Database and Knowledge-base Systems, vol. 1, Computer Science Press, Rockville, MD, 1989.
[37] A. Weber, Updating propositional formulas, First Internat. Conf. on Expert Database Systems, 1986, pp. 373–386.
[38] R.J. Wieringa, W. de Jonge, P.A. Spruit, Using dynamic classes and role classes to model object migration, Theory Practice Object Systems 1(1) (1995) 61–83.
[39] R.J. Wieringa, J.-J.Ch. Meyer, Actors, actions, and initiative in normative system specification, Ann. Math. Artif. Intell. 7 (1993) 289–346.
[40] L. Willard, L.Y. Yuan, The revised Gärdenfors postulates and update semantics, in: S. Abiteboul, P.C. Kanellakis (Eds.), Internat. Conf. on Database Theory, Lecture Notes in Computer Science, vol. 470, Springer, Berlin, 1990, pp. 409–421.
[41] M. Winslett, Updating Logical Databases, Cambridge University Press, Cambridge, 1990.