Acta Cybernetica **19** (2010) 765–805.

A Customised ASM Thesis for Database Transformations

Klaus-Dieter Schewe[*] and Qing Wang[†]

Abstract

In order to establish a theoretical foundation for database transformations, we search for a universal computation model as an umbrella for queries and updates. As updates are fundamentally distinct from queries in many respects, computation models for queries cannot be simply extended to database transformations. This motivates the question whether Abstract State Machines (ASMs) can be used to characterise database transformations in general. In this paper we start examining the differences between database transformations and algorithms, which give rise to the formalisation of five postulates for database transformations. Then a variant of ASMs called Database Abstract State Machines (DB-ASMs) is developed, and we prove that DB-ASMs capture database transformations, i.e. the main result of the paper is that every database transformation stipulated by the postulates can be behaviourally simulated by a DB-ASM.

Keywords: Abstract State Machine, database transformation, ASM thesis

1 Introduction

According to [2] a database transformation is a binary relation on database instances that encompass queries and updates. In general, a database transformation can be non-deterministic, but it must be recursively enumerable and generic in the sense that it preserves isomorphisms. The problem addressed in this article is to completely characterise the algorithms that transform input databases into output databases.

Abstract State Machines (ASMs) provide a universal computation model that formalises the notion of (sequential or parallel) algorithm [5, 10]. In his seminal work on the sequential ASM thesis Gurevich points out the difference between a computable function in the recursion-theoretic sense and an algorithm. Strictly speaking, many algorithms in numerical mathematics, e.g. Newton's algorithm for

[*]Software Competence Center Hagenberg, Hagenberg, Austria and Johannes-Kepler-University Linz, Research Institute for Applied Knowledge Processing, Linz, Austria, E-mail: `kd.schewe@scch.at, kd.schewe@faw.at`

[†]University of Otago, Dunedin, New Zealand, E-mail: `qing.wang@otago.ac.nz`

determining zeros of a differentiable function $f : \mathbb{R} \to \mathbb{R}$, do not define computable functions, as they deal with non-denumerable sets. Only if restricted to a countable subset – such as floating-point numbers instead of real numbers – and properly encoded can we get a computable function. Thus, we are actually aiming at a complete characterisation of database transformation algorithms, but nonetheless we stick to the commonly used term of database transformation.

1.1 Contributions

Analogous to the seminal work on the ASM thesis this article contains two major contributions. The first one is a characterisation of database transformations by a set of simple and intuitive postulates. These postulates should cover all database transformations, and thus leave sufficient latitude to specify the specific characteristics of data models such as the standard relational data model, object oriented data models, and databases based on the eXtensible Markup Language (XML). In forthcoming studies we will elaborate the details for these models – for XML this has been done in [16].

The second contribution is a variant of ASMs, which we call Database Abstract State Machines (DB-ASMs). We show that DB-ASMs can capture exactly database transformations stipulated by the postulates. That is, we first show that DB-ASMs satisfy the postulates, and then that any object satisfying the postulates can be simulated by a DB-ASM.

We start with examining database transformations in the light of the postulates for sequential algorithms[1] defined in [10]. Firstly, database transformations should terminate, which implies that a run will always be finite and reach a final state. Furthermore, in order to take into consideration not only deterministic but also non-deterministic database transformations the requirement of a one-step transition function has to be relaxed. We will permit a relation instead, leading to a slightly modified sequential time postulate. The necessity for non-determinism arises among others from the creation of objects [18].

According to the abstract state postulate for ASMs states are first-order structures, and the sets of states used for an algorithm are invariant under isomorphisms. This notion of state must capture databases in general. Therefore, we will customize the abstract state postulate requesting that a state is composed out of a finite database component and an arbitrary algorithmic component that are linked via bridge functions. This picks up a fundamental idea from meta-finite model theory [9].

Same as for the parallel ASM thesis we have to refer explicitly to the background of a computation, which contains everything that is needed to perform the computation, but is not yet captured by the state. For instance, truth values and their connectives, and a value $\perp$ to denote undefinedness constitute necessary elements in a background. Furthermore, for database transformations we have to capture constructs that are determined by the used data model, so we will have to

[1]In Gurevich's theory sequential algorithms still permit bounded parallelism, whereas parallel algorithms are understood to capture even unbounded parallelism.

deal with type constructors, and with functions defined on such types. This will lead us to the background postulate for database transformations.

The fourth postulate needed for the sequential ASM thesis, the bounded exploration postulate, requires that there is a finite set of terms called bounded exploration witness, and only these terms can be updated in a one-step transformation. The generalisation of this postulate to the case of parallel algorithms in the parallel ASM thesis [5] leads to several significantly more complex postulates. As database transformations are intrinsically parallel computations, though an implementation may be sequential, we have to adopt parts of these more complex postulates. The adoption will only be partial, as the parallelism in database transformations – excluding for now the area of parallel databases – is rather limited; it merely amounts to the same computation on different data.

Regarding the restricted form of parallelism needed for database transformations we capture this by location operators, which generalise aggregation functions and cumulative updates. With these location operators we actually deal with meta-finite structures with multiset operations as defined in [9]. In doing so, we have to consider update multisets, which are reduced to update sets by means of the location operators. Furthermore, depending on the data model used and thus on the actual background signature we may use complex values, e.g. tree-structured values, which leads to the problem of partial updates [11], i.e. we have to ensure that parallel updates to different parts of a tree (or database object, in general) can be synchronised. Dealing with partial updates actually is subsumed in the notion of consistent update set. Taking these ingredients together we obtain a slightly modified bounded exploration postulate.

The fifth postulate addresses non-determinism. As we permit non-determinism, equivalence of substructures may indeed be destroyed, but the non-determinism postulate ensures that non-determinism is restricted by depending only on the database part and not on the algorithmic part. However, bridge functions need to be restricted accordingly.

We then define DB-ASMs. Naturally, the permitted non-determinism requires the presence of a choice construct, while the restricted parallelism leads to a let construct that binds locations to location operators and a forall construct that allows the creation of a finite number of parallel subcomputations. For DB-ASMs we first show that they satisfy the postulates for database transformations. Our main result then shows that DB-ASMs capture database transformations, i.e. every database transformation stipulated by the postulates can be behaviourally simulated by a DB-ASM.

1.2 Background and Previous Work

With the upcoming of query languages for object oriented databases [1] the view that a query transforms databases over an input database schema into databases over an output schema that is disjoint from the input schema had to be relaxed by considering queries as transformations from an input schema to an extended output schema that preserve the input. From here it is a very small step to consider database transformations in general [2]. Since then a lot of research has been undertaken aiming at a logical characterisation of database transformations, e.g. [1, 17, 20, 18].

As discussed in [1], database transformations should satisfy criteria such as well-typedness, effective computability, genericity and functionality. However, most research with respect to these properties was conducted only for queries. According to [19] extending these results to updates is by no means straightforward. So far, there is not yet a computation model that can serve as a theoretical foundation for database transformations in general. In this article we aim at such a general model exploiting the theory of ASMs.

This article extends and corrects a preliminary conference publication [21]. In our previous work we tried to focus on tree-based databases, and therefore described states by higher-order structures. In this article, however, we capture databases in general, and therefore stay with first-order structures, while everything needed to express the specific needs of a data model is subsumed by the inclusion of background structures, which among others provide the necessary type constructors. Furthermore, we took up the idea from [22] to exploit meta-finite states [9]. In our previous work we were not able to handle such bridge functions properly. Finally, we polished the postulate capturing non-determinism.

1.3 Organisation of the Article

The remainder of this article is organised as follows. We begin with an illustrative example in Section 2. Then in Section 3 we present in detail our five postulates for database transformations. We motivate the postulates with examples that highlight the specific problems of developing a customised ASM thesis for database transformations. In addition we discuss the differences to the postulates in the sequential and parallel ASM theses. In Section 4 we present the DB-ASM variant of ASMs, and show that DB-ASMs satisfy the postulates. While this is relatively easy to achieve, we prove the converse in Section 5, i.e. DB-ASMs capture all database transformations. We conclude in Section 6 with a summary and discussion of further research, for which our current result is only the basis.

2 Illustrative Example

In this section we will provide an example to illustrate how a database transformation can be characterised by the five postulates and to answer the question of what a simulating DB-ASM for a database transformation should look like. The intention is to help the understanding of formal definitions presented in Section 3 and Section 4.

Example 1. Let us consider the weighted graph shown in Figure 1 and the database transformation "find the cheapest costs to reach other cities from the city C", where a node in the graph denotes a city and an arrow between two nodes denotes the transportation cost from one city to another.

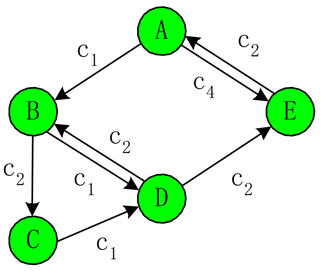

Figure 1: A weight graph with cities

Suppose that we choose the relational data model to represent the weighted graph in a database. That is, the relations City and Route in Figure 2 contain the information for cities and the transportation costs between two cities, respectively. Furthermore, we need the relation Visited to keep track of cities that have already been visited during the intermediate computation of a database transformation and the relation Result to store the final result. Now we discuss how to characterise this database transformation by using the five postulates.

- The sequential time postulate defines that this database transformation is a step-by-step computation which proceeds by one-step transitions from states to their successor states. A run of this database transformation is a finite sequence of states. If a run starts from an initial state that has the relations as shown in Figure 2, then it may terminate at a final state that has the relations as shown in Figure 3.

- The abstract state postulate defines that each state of this database transformation consists of the database part that contains four relations City, Route, Visited and Result, the algorithmic part that is an infinite structure $(\mathbb{R}, +, \cdot, max, min, \sum, \prod)$ and a bridge function $f_b : c_i \rightarrow i$ for $i = 0, ..., 5$. The purpose of the bridge function f_b is to interpret abstract elements of the attribute Cost in the relation Route of the database part with real numbers from the algorithmic part.

- The background postulate defines the background of this database transformation to reflect the choice of the relational data model and the use of relational algebra for query rewriting and optimisation at the implementation level. Therefore, each state should contain a background class defined by the background signature including at least type constructor symbols for the relational data model (i.e., finite tuple $(\cdot)$ and finite set $\{\cdot\}$) and relational algebraic operators (i.e.,σ (selection), π (projection), $\bowtie$ (join), $\cup$ (union), $-$ (difference), ϱ (renaming)), a set of base domains and a set of algebraic identities for rewriting query expressions.

- The bounded exploration postulate defines that, regardless of the chosen database language, etc, there must exist a fixed, finite set of access terms for this database transformation, which should include at least (City(x,y), true), (Route(x,y,z), true), (Visited(x), false), (Result(x,y,z), true) to access elements in the database part of a state. Furthermore, for any two different states, if they have the same interpretation for such a set of access terms, then the database transformation much create the same set of update sets over these two states at one-step transitions.

- The bounded exploration postulate defines that at any one-step transition the number of successor states to a state is finite, depending on access terms that can access elements from the database part of a state in a generic way.

City

Cid	Name
i_1	A
i_2	B
i_3	C
i_4	D
i_5	E

Route

FromCid	ToCid	Cost
i_1	i_2	c_1
i_1	i_5	c_4
i_2	i_3	c_2
i_2	i_4	c_1
...	...	...

Visited

Cid

Result

Cid	TotalCost	LastStop

Figure 2: The relations in an initial state

City

Cid	Name
i_1	A
i_2	B
i_3	C
i_4	D
i_5	E

Route

FromCid	ToCid	Cost
i_1	i_2	c_1
i_1	i_5	c_4
i_2	i_3	c_2
i_2	i_4	c_1
...	...	...

Visited

Cid
i_3
i_4
i_2
i_5
i_1

Result

Cid	TotalCost	LastStop
i_1	c_5	i_5
i_2	c_3	i_4
i_3	c_0	null
i_4	c_1	i_3
i_5	c_3	i_4

Figure 3: The relations in a final state

As one of important results in this paper is the development of DB-ASMs that can capture exactly all database transformations stipulated by the postulates, we implement Dijkstra's algorithm in a DB-ASM to simulate the database transformation discussed in Example 1.

Assume that, we have nullary function symbols @startcity, @infinity, initial, finished and mvalue in the state signature and @startcity=C, @infinity is a predefined number that should be big enough to be distinguished from the calculated cost numbers and initial=0 in every initial state of the database transformation. Then a simulating DB-ASM is presented in Figure 4.

```
par if initial=0 then seq
        forall x with CITY(x,y) do
            if x=@startcity then
                RESULT(x, 0, null):=true
            else
                RESULT(x,@infinity,null):=true
            endif enddo
        par initial:=1    finished:=0 endpar
    endseq endif
    if initial=1$\wedge$finished=0 then seq
        finished:=1
        let $\theta$(mvalue)=min in
            forall y with $\exists$ x,y. RESULT(x,y,z)$\wedge$ y< @infinity$\wedge\neg$ VISITED(x) do
                par mvalue:=$f_b$(y)    finished:=0 endpar
            enddo
        endlet
        if finished=0 then
            choose x with $\exists$y,z.RESULT(x,y,z)$\wedge f_b$(y)=mvalue do seq
                VISITED(x):=true
                forall y,y$'$,z$'$,z with ROUTE(x,y,z)$\wedge$ RESULT(y,y$'$,z$'$)$\wedge$
                                      mvalue+$f_b$(z)< $f_b$(y$'$)$\wedge\neg$ VISITED(y) do seq
                f(y):=new()
                    par
                        RESULT(y,f(y),x):=true
                        RESULT(y,y$'$,z$'$):=false
                        f$_b$(f(y)):=mvalue+f$_b$(z)
                    endpar
                endseq enddo
            endseq enddo
        endif
    endseq endif
endpar
```

Figure 4: A simulating DB-ASM

3 Postulates for Database Transformations

In this section we will formally introduce the five postulates for database transformations: the *sequential time postulate*, the *abstract state postulate*, the *background postulate*, the *bounded exploration postulate*, and the *bounded non-determinism postulate.*

Definition 1. A *database transformation* is an object satisfying the sequential time postulate, the abstract state postulate, the background postulate, the bounded exploration postulate, and the bounded non-determinism postulate.

3.1 Sequential Time

As in [10] and [5] we assume that a database transformation same as any algorithm proceeds step-wise on a set of states. It starts somewhere, which gives us a set of initial states. However, while it makes perfect sense to consider non-terminating algorithms, we want to consider only terminating database transformations, for which we add a set of final states. As discussed in [10] this is more a technicality as far as the work in this paper is concerned, but we aim at embedding our results into a theory of database systems, in which many database transformations have to co-exist.

Furthermore, we deviate from the sequential time postulate in the sequential and parallel ASM theses by using a one-step transition relation on states as in bounded-choice sequential algorithms [12] instead of a transformation function. This introduces non-determinism into database transformations, which will be limited by further postulates. In fact we will only permit non-deterministic choice among the finite answer to a query. The major reason for the non-determinism in database transformations is the need for creating objects in some data models as discussed intensively in [19, 17, 18].

Postulate 1 (sequential time postulate)**.** *A database transformation t is associated with a non-empty set of states $\mathcal{S}_t$ together with non-empty subsets $\mathcal{I}_t$ and $\mathcal{F}_t$ of initial and final states, respectively, and a one-step transition relation τ_t over $\mathcal{S}_t$, i.e. $\tau_t \subseteq \mathcal{S}_t \times \mathcal{S}_t$.*

The sequential time postulate allows us to define the notion of a *run* in analogy to sequential and parallel algorithms. As we require termination, a run must be finite ending in a final state, which should be the first final state that is reached. Nevertheless, we permit the initial state in a run to be also a final state. The motivation behind this is that database transformations, though treated in isolation in this paper, are associated with a database system, in which each run of a database transformation produces a state transition, and the final state of that transition becomes the initial state for another transition, etc. leading to an infinite sequence of states that results from running a set of database transformations in a serial (or serialisable) way. This view of database systems has been stressed in [14].

Definition 2. A *run* of a database transformation t is a finite sequence $S_0, \dots, S_f$ of states with $S_0 \in \mathcal{I}_t$, $S_f \in \mathcal{F}_t$, $S_i \notin \mathcal{F}_t$ for $0 < i < f$, and $(S_i, S_{i+1}) \in \tau_t$ for all $i = 0, \dots, f-1$.

We will consider database transformations only up to behavioural equivalence.

Definition 3. Database transformations t_1 and t_2 are *behaviourally equivalent* iff $\mathcal{S}_{t_1} = \mathcal{S}_{t_2}$, $\mathcal{I}_{t_1} = \mathcal{I}_{t_2}$, $\mathcal{F}_{t_1} = \mathcal{F}_{t_2}$ and $\tau_{t_1} = \tau_{t_2}$ hold.

Obviously, behaviourally equivalent database transformations have the same runs.

3.2 Abstract States

The abstract state postulate is an adaptation of the corresponding postulate for Abstract State Machines [10], according to which states are first-order structures, i.e. sets of (partial) functions, some of which may be marked as relational. These functions are interpretations of function symbols given by some signature. Following [10] it is assumed that each signature contains the equality sign, and nullary names *true*, *false*, *undef*, a unary name *Bool*, and the names of the usual Boolean operations. With the exception of *undef* all these logic names are relational. Equality, truth values and Boolean operations are interpreted in a fixed way in all states. In particular, partial functions are captured by the undefinedness value $\perp$ associated with *undef*.

Definition 4. A *signature* Σ is a set of function symbols, each associated with a fixed arity. A *structure* over Σ consists of a set B, called the *base set* of the structure together with interpretations of all function symbols in Σ, i.e. if $f \in \Sigma$ has arity k, then it will be interpreted by a function from B^k to B.

Let X be a structure over Σ. For each term $t \in \Sigma$ we use $val_X(t)$ to denote the interpretation of t in the structure X.

Definition 5. An *isomorphism* from structure X to structure Y is defined by a bijection $\sigma : B_X \to B_Y$ between the base sets that extends to functions by $\sigma(val_X(f(b_1, \dots, b_k))) = val_Y(f(\sigma(b_1), \dots, \sigma(b_k)))$. A Z*-isomorphism* for $Z \subseteq B_X \cap B_Y$ is an isomorphism σ from X to Y that fixes Z, i.e. $\sigma(b) = b$ for all $b \in Z$.

As the base set B contains a value $\perp$ representing undefinedness, partial functions are captured in the usual way. Furthermore, relations are captured by letting $val_X(f(a_1, \dots, a_k)) = true$ mean that $(a_1, \dots, a_k)$ is in the relation f of X, and $val_X(f(a_1, \dots, a_k)) = false$ mean that it is not.

Z-isomorphisms are needed when dealing with constants in the base set that are represented by 0-ary function symbols. However, an automorphism σ of a structure X fixes all values $a \in B$ that are represented by ground terms, i.e. if $a = val_X(t)$ holds for some ground term t, then $\sigma(a) = a$ holds for each automorphism σ of

X. Thus, Z-isomorphisms can be neglected, as we could always add syntactic surrogates, i.e. 0-ary function symbols for all elements of Z to the signature Σ.

Taking structures as states reflects common practice in mathematics, where almost all theories are based on first-order structures. Variables are special cases of function symbols of arity 0, and constants are the same, but unchangeable.

In the case of databases we have to take care of two specific problems that affect the definition of states. The first problem is the intrinsic finiteness of databases. For this it is tempting to adopt Finite Model Theory [8] and consequently require that states for database transformations are finite structures. This would, however, not capture the full picture. For instance, a simple query counting the number of tuples in a relation would require natural numbers in the base set, and any restriction to a finite set would already rule out some database transformations. Fortunately, in order to deal with this problem Grädel and Gurevich proposed the use of *meta-finite* structures [9], henceforth *meta-finite states*, in which we may consider actual database entries as being merely surrogates for real values. This permits the database to remain finite while adding functions that interpret database entries in possibly infinite domains, and at the same time generalises most of the result achieved in Finite Model Theory to meta-finite models. We adopt this model, and consequently there will be three kinds of function symbols, those representing the database, those representing everything outside the database, and bridge functions that map surrogates in the database to values outside the database in possibly infinite domains.

There is one little sublety here that we also have to take care of. In object-oriented databases we may make use of object identifiers, in tree-based databases we may require identifiers for tree nodes, and in both cases there is a need to create new identifiers. As discussed in [10] the creation of new values in principle is no problem, as we can assume an infinite set of reserve values existing in the background of a database transformation, from which such new values are taken, but it is not a priori clear how many such values will be needed. In the subsection on backgrounds we will further clarify this matter. Therefore, this problem can be circumvented by requiring that only the active database domain is finite, i.e. the set of values of the base set appearing in the database part of the structure.

The second database-specific problem is the presence of a data model that prescribes how a database should look like. In case of the relational model we would have to deal simply with relations, while in case of object-oriented and XML-based databases we need constructors for complex values such as finite sets, multisets, maps, arrays, union, trees, etc. We will deal with the consequences for states separately in the subsection on backgrounds.

Definition 6. The *signature* Σ of a meta-finite structure is composed as a disjoint union consisting of a sub-signature Σ_{db} (called *database part*), a sub-signature Σ_a (called *algorithmic part*) and a finite set of bridge function symbols each with a fixed arity, i.e. $\Sigma_t = \Sigma_{db} \cup \Sigma_a \cup \{f_1, \dots, f_\ell\}$. The *base set* of a meta-finite strcuture S is $B = B_{db}^{ext} \cup B_a$ with interpretation of function symbols in Σ_{db} and Σ_a over $B_{db} \subseteq B_{db}^{ext}$ and B_a, respectively, with B_{db} depending on S. The interpretation of a bridge function symbol of arity k defines a function from B_{db}^k to B_a. With respect to such states S the restriction to Σ_{db} is a finite structure, i.e. B_{db} is finite.

Postulate 2 (abstract state postulate)**.** *All states $S \in \mathcal{S}_t$ of a database transformation t are meta-finite structures over the same signature Σ_t, and whenever $(S, S') \in \tau_t$ holds, the states S and S' have the same base set B. The sets $\mathcal{S}_t$, $\mathcal{I}_t$ and $\mathcal{F}_t$ are closed under isomorphisms, and for $(S_1, S_1') \in \tau_t$ each isomorphism σ from S_1 to S_2 is also an isomorphism from S_1' to $S_2' = \sigma(S_1')$ with $(S_2, S_2') \in \tau_t$.*

The abstract state postulate is an adaptation of the analogous postulate from [10, 5] to further consider states as being meta-finite structures, the presence of final states and the fact that one-step transition is a binary relation.

Example 2. Consider a database, in which we represent persons with their name and age. In the sub-signature Σ_{db} we would thus have a binary function symbol *person*. In addition, Σ_a could contain a unary function symbol *even*, and we would have two bridge functions f_{name} and f_{age} for the interpretation of the two components of persons.

In a structure, we would get $B_{db} = D_{\text{name}} \cup D_{\text{age}} \cup \{true, false\}$ as a union of three disjoint, finite sets, and $B_a = \mathbb{N} \cup A^*$ as the union of the set of non-negative integers and the set of character strings over some alphabet A, both infinite.

The function symbol *person* would be interpreted by a function $D_{\text{name}} \times D_{\text{age}} \to \{\text{true, false}\}$, and *even* would be interpreted by a function $\mathbb{N} \to \{\text{true, false}\}$. The bridge function symbols would be interpreted as functions $D_{\text{name}} \to A^*$ and $D_{\text{age}} \to \mathbb{N}$, respectively.

Using this representation of a finite relation of persons with name and age, a query such as "List the names of all persons with even age" would be possible, provided we add a unary function symbol *names* to the signature to pick up the result.

Similarly, a query such as "List the names of all persons who are older than average" would require 0-ary function symbols *age_sum*, *count*, and *average_age* as well as a binary function symbol $>$ in the algorithmic part of the signature.

In the abstract state postulate above we adopt the idea of meta-finite states, but we do not restrict the database part of the state to be relational, so we can capture data models other than the relational one as well.

Let us finally look at genericity as expressed by the preservation of isomorphisms in successor states. In the sequential ASM thesis sequential algorithms are deterministic, so a state S has a unique successor state $\tau(S)$. Then the abstract state postulate implies that an automorphism σ of S is also an automorphism of $\tau(S)$. In the abstract state postulate for database transformations (i.e., Postulate 2), however, there can be more than one successor state of S, as τ_t is a relation. Now, if σ is an automorphism of S, and $(S, S') \in \tau_t$ holds, we obtain an isomorphism σ from S' to $S'' = \sigma(S')$ with $(S, S'') \in \tau_t$. Thus, an automorphism of S induces a permutation of the successor states of S.

3.3 Updates

The definitions of locations, updates, update sets and update multisets are the same as for ASMs [6].

Definition 7. For a database transformation t let S be a state of t, f a dynamic function symbol of arity n in the state signature of t, and $a_1, ..., a_n, v$ be elements in the base set of S. Then $f(a_1, ..., a_n)$ is called a *location* of t. The interpretation of a location ℓ in S is called the *content* of ℓ in S, denoted by $val_S(\ell)$. An *update* of t is a pair (ℓ, v), where ℓ is a location and v is an update value. An *update set* is a set of updates; an *update multiset* is a multiset of updates.

An update is *trivial* in a state S if its location content in S is the same with its update value, while an update set is *trivial* if all of its updates are trivial.

An update set Δ is *consistent* if it does not contain conflicting updates, i.e. for all $(\ell, v), (\ell, v') \in \Delta$ we have $v = v'$.

Using a *location function* (denoted by θ) that assigns a location operator or $\perp$ to each location, an update multiset can be reduced to an update set. It is further possible to construct for each $(S, S') \in \tau_t$ a minimal update set $\Delta(t, S, S')$ such that applying this update set to the state S will produce the state S'. More precisely, if S is a state of the database transformation t and Δ is a consistent update set for the signature of t, then there exists a unique state $S' = S + \Delta$ resulting from updating S with Δ: we simply have

$$val_{S+\Delta}(\ell) \quad = \quad \begin{cases} v & \text{if } (\ell, v) \in \Delta \\ val_S(\ell) & \text{else} \end{cases}$$

If Δ is not consistent, we let $S + \Delta$ be undefined. Note that this last point is different from the treatment of inconsistent update sets in [10], but as discussed there the difference is a mere technicality as long as we concentrate on a single database transformation. Same as with final states the distinction only becomes necessary when placed into the context of persistence with several concurrent database transformations, and a serialisability request. In that case a computation that gets stuck and thus has to be aborted (this is the case, when $S + \Delta$ is undefined) has to be distinguished from a computation that produces the same state S over and over again (this is the case, if $S + \Delta$ is defined as S in case of Δ being inconsistent as in [10]).

Lemma 1. *Let $S, S' \in \mathcal{S}_t$ be states of the database transformation t with the same base set. Then there exists a unique, minimal consistent update set $\Delta(t, S, S')$ with $S' = S + \Delta(t, S, S')$.*

Note that the minimality of the update set implies the absence of trivial updates.

Proof. Let $Loc_\Delta = \{\ell \mid val_S(\ell) \neq val_{S'}(\ell)\}$ be the set of locations, on which the two states differ. Then the update set $\Delta(t, S, S') = \{(\ell, val_{S'}(\ell) \mid \ell \in Loc_\Delta\}$ is the one needed. $\square$

Let us now look at the one-step transition relation τ_t of a database transformation t. As we permit non-determinism, i.e. there may be more than one successor state of a state S, we need a set of update sets. Therefore, define

$$\Delta(t, S) = \{\Delta(t, S, S') \mid (S, S') \in \tau_t\}$$

for a database transformation t and a state $S \in \mathcal{S}_t$.

Let us take a brief look at the effect of isomorphisms on update sets and sets of update sets. For this, any isomorphism σ can be extended to updates $(f(a_1, \dots, a_n), b)$ by defining $\sigma((f(a_1, \dots, a_n), b)) = (f(\sigma(a_1), \dots, \sigma(a_n)), \sigma(b))$, and to sets by defining $\sigma(\{u_1, \dots, u_k\}) = \{\sigma(u_1), \dots, \sigma(u_k)\}$.

Lemma 2. *Let S_1 be a state of a database transformation t and σ be an isomorphism from S_1 to S_2. Then $\Delta(t, S_2, \sigma(S_1')) = \sigma(\Delta(t, S_1, S_1'))$ for all $(S_1, S_1') \in \tau_t$, and consequently $\Delta(t, S_2) = \sigma(\Delta(t, S_1))$.*

Proof. According to the abstract state postulate all $(S_2, S_2') \in \tau_t$ have the form $(\sigma(S_1), \sigma(S_1'))$ with $(S_1, S_1') \in \tau_t$. Then $val_{\sigma(S_1)}\sigma(\ell) = \sigma(val_{S_1}(\ell))$ and analogously for S_1'. So

$$Loc_{\Delta_2} = \{\ell \mid val_{S_2}(\ell) \neq val_{S_2'}(\ell)\} = \{\sigma(\ell) \mid val_{S_1}(\ell) \neq val_{S_1'}(\ell)\} = \sigma(Loc_{\Delta_1}),$$

and

$$\begin{aligned}\Delta(t, S_2, S_2') =& \{(\ell, val_{S_2'}(\ell)) \mid \ell \in Loc_{\Delta_2}\} = \\ & \{(\sigma(\ell^{'}), \sigma(val_{S_1'}(\ell^{'}))) \mid \sigma(\ell^{'}) \in \sigma(Loc_{\Delta_1})\} = \sigma(\Delta(t, S_1, S_1')).\end{aligned}$$

□

3.4 Backgrounds

The postulates 1 and 2 are in line with the sequential and parallel ASM theses [10, 5], and with the exception of allowing non-determinism in the sequential time postulate and the reference to meta-finite structures in the abstract state postulate there is nothing in these postulates that makes a big difference to postulates for sequential algorithms. The next postulate, however, is less obvious, as it refers to the background of a computation, which contains everything that is needed to perform the computation, but is not yet captured by the state. For instance, truth values and their connectives, and a value $\perp$ to denote undefinedness constitute necessary elements in a background.

For database transformations, in particular, we have to capture constructs that are determined by the used data model, e.g. relational, object-oriented, object-relational or semi-structured, i.e. we will have to deal with type constructors, and with functions defined on such types. Furthermore, when we allow values, e.g. identifiers to be created non-deterministically, we would like to take these values out of an infinite set of reserve values. Once created, these values become active, and we can assume they can never be used again for this purpose.

Let us take the following example, which was used in [1] to illustrate a data model that generalises most of known complex object data models. In this model a distinction is made between abstract identifiers and constants. These elements stem from disjoint base domains, which together constitute the *base set* for database transformations. With the addition of constructors for records, sets, multisets, lists, etc. domains of arbitrarily nested complex values can be built upon the base domains. For instance, a domain $D_{(Int,\{String\})}$ over base domains *Int* and *String* would represent complex record values consisting of an integer and a set of strings.

Example 3. Suppose that the state of a database has a universe containing abstract identifiers from domains $I_1 = \{i_{eve}, i_{adam}\}$, $I_2 = \{i_{cain}, i_{abel}, i_{seth}, i_{other}\}$, $I_3 = \{i_{n_k} | k = 1, ..., 5\}$, $I_4 = \{i_{o_k} | k = 1, ..., 3\}$, $I_5 = \{i_{d_1}, i_{d_2}\}$ and constants from domains *String* and *Bool*. Furthermore, assume that we have the following constructors: finite sets $\{\cdot\}$ with unfixed arity, records $(\cdot)$ with arity up to 3, and union $\cup$ with arity 2.

Let the state signature contain function names *1st-generation*, *2nd-generation*, *name*, *occupation*, *descendant*, and relation names *founded-lineage*, *ancestor-of-celebrity* such that

- *1st-generation*: $I_1 \rightarrow \mathrm{D}_{(nam:I_3, spou:I_1, children:\{I_2\})}$,
- *2nd-generation*: $I_2 \rightarrow \mathrm{D}_{(nam:I_3, occu:I_4)}$,
- *founded-lineage*: 2nd-gen:$I_2 \rightarrow$ *Bool*,
- *ancestor-of-celebrity*: anc:$I_2 \times$ desc:$I_5 \rightarrow$ *Bool*,
- *name*: $I_3 \rightarrow \mathrm{D}_{String}$,
- *occupation*: $I_4 \rightarrow \mathrm{D}_{\{String\}}$,
- *descendant*: $I_5 \rightarrow \mathrm{D}_{String \cup (spou:String)}$.

The interpretation of function and relation names in the state signature is as follows:

- for *1st-generation*, $i_{eve} \mapsto$ (nam: i_{n_1}, spou: i_{adam}, children: $\{i_{cain}, i_{abel}, i_{seth}, i_{other}\}$) and $i_{adam} \mapsto$ (nam:i_{n_2}, spou: i_{eve}, children:$\{i_{cain}, i_{abel}, i_{seth}, i_{other}\}$),
- for *2nd-generation*, $i_{cain} \mapsto$ (nam: i_{n_3}, occu: i_{o_1}), $i_{seth} \mapsto$ (nam: i_{n_4}, occu: i_{o_2}) and $i_{abel} \mapsto$ (nam: i_{n_5}, occu: i_{o_3}),
- for *founded-lineage*, it is {(2nd-gen: i_{cain}),(2nd-gen: i_{seth}),(2nd-gen: i_{other})},
- for *ancestor-of-celebrity*, it is {(anc: i_{seth}, desc: i_{d_1}), (anc: i_{cain}, desc: i_{d_2})},
- for *name*, $i_{n_1} \mapsto$ Eve, $i_{n_2} \mapsto$ Adam, $i_{n_3} \mapsto$ Cain, $i_{n_4} \mapsto$ Seth and $i_{n_5} \mapsto$ Abel,
- for *occupation*, $i_{o_1} \mapsto$ {Farmer, Nomad, Artisan}, $i_{o_2} \mapsto$ {} and $i_{o_3} \mapsto$ {Shepherd},

- for *descendant*, $i_{d_1} \mapsto$ Noah and $i_{d_2} \mapsto$ (spou: Ada).

That is, objects of *1st-generation* are described by a name, a reference to a spouse, and a set of references to children. Objects of *2nd-generation* are described by a name and a set of professions, and objects of *descendant* are described by a name only or a record with a name. The *founded-lineage* defines a subset of the second generation, and *ancestor-of-celebrity* is a simple binary relation.

Suppose, we want to create a new object with a new identifier in I_3. For this, we obtain a new identifier $i_3 \in I_3$, from the set of reserve values. We then set $\mathit{name}(i_3) :=$ Isaac and $\mathit{ancestor\text{-}of\text{-}celebrity}(i_{seth}, i_3) := \mathit{true}$ to update *name* and *ancestor-of-celebrity* (taking $false$ as the default value for all other cases). Similarly, with $\mathit{founded\text{-}lineage}(i_{other}) := false$ we would delete i_{other} from the *founded-lineage* relation.

Following [5] we use background classes to define backgrounds, which will then become part of states. Background classes themselves are determined by background signatures that consist of constructor symbols and function symbols. Function symbols are associated with a fixed arity as in Definition 4, but for constructor symbols we permit the arity to be unfixed or bounded.

Definition 8. Let $\mathcal{D}$ be a set of base domains and V_K a background signature, then a *background class* $\mathcal{K}$ with V_K over $\mathcal{D}$ is constituted by

- the universe $\mathcal{U} = \bigcup_{D \in \mathfrak{D}} D$ of elements, where $\mathfrak{D}$ is the smallest set with $\mathcal{D} \subseteq \mathfrak{D}$ satisfying the following properties for each constructor symbol $\llcorner\lrcorner \in V_K$:
  - If $\llcorner\lrcorner \in V_K$ has unfixed arity, then $\llcorner D \lrcorner \in \mathfrak{D}$ for all $D \in \mathfrak{D}$, and $\llcorner a_1, \dots, a_m \lrcorner \in \llcorner D \lrcorner$ for every $m \in \mathbb{N}$ and $a_1, \dots, a_m \in D$.
  - If $\llcorner\lrcorner \in V_K$ has unfixed arity, then $A_{\llcorner\lrcorner} \in \mathfrak{D}$ with $A_{\llcorner\lrcorner} = \bigcup_{\llcorner D \lrcorner \in \mathfrak{D}} \llcorner D \lrcorner$.
  - If $\llcorner\lrcorner \in V_K$ has bounded arity n, then $\llcorner D_1, \dots, D_m \lrcorner \in \mathfrak{D}$ for all $m \leq n$ and $D_i \in \mathfrak{D}$ $(1 \leq i \leq m)$, and $\llcorner a_1, \dots, a_m \lrcorner \in \llcorner D_1, \dots, D_m \lrcorner$ for every $m \in \mathbb{N}$ and $a_1, \dots, a_m \in D$.
  - If $\llcorner\lrcorner \in V_K$ has fixed arity n, then $\llcorner D_1, \dots, D_n \lrcorner \in \mathfrak{D}$ for all $D_i \in \mathfrak{D}$ $(1 \leq i \leq n)$, and $\llcorner a_1, \dots, a_n \lrcorner \in \llcorner D_1, \dots, D_n \lrcorner$ for all $a_1, \dots, a_n \in D$.
- and an interpretation of function symbols in V_K over $\mathcal{U}$.

Example 4. Let us consider the type system used in [1] with some slight modifications. Type expressions are defined as follows:

$$\tau = \lambda \mid D \mid P \mid (A_1 : \tau_1, \dots, A_k : \tau_k) \mid \{\tau\} \mid \tau_1 \sqcup \tau_2 \mid \tau_1 \sqcap \tau_2$$

The semantics of these type expressions, denoted as $[\![\tau]\!]$, is formally defined as follows:

$$\begin{aligned}
[\![\lambda]\!] &= \emptyset \\
[\![D]\!] &= \xi_1(D) \\
[\![P]\!] &= \xi_2(P) \\
[\![(A_1 : \tau_1, ..., A_k : \tau_k)]\!] &= \{(A_1 : v_1, ..., A_k : v_k) \mid v_i \in [\![\tau_i]\!], i = 1, ..., k\} \\
[\![\{\tau\}]\!] &= \{\{v_1, ..., v_j\} \mid j \geq 0 \text{ and } v_i \in [\![\tau]\!], i = 1, ..., j\} \\
[\![\tau_1 \sqcup \tau_2]\!] &= [\![\tau_1]\!] \cup [\![\tau_2]\!] \\
[\![\tau_1 \sqcap \tau_2]\!] &= [\![\tau_1]\!] \cap [\![\tau_2]\!]
\end{aligned}$$

So λ is a trivial type denoting the empty set $\emptyset$. D and P represent a base type for constants and a class type for objects, respectively, and ξ_1 and ξ_2 are functions mapping each base type to a possibly infinite set of constants, and each class type to a finite set of objects, respectively. In addition to these, there are constructor symbols – records $(\cdot)$ with bounded arity k, finite sets $\{\cdot\}$ with unfixed arity, as well as unions $\sqcup$ and intersections $\sqcap$, both of arity 2.

The types are associated with function symbols $\in$ of arity 2 denoting set membership, π_i $(1 \leq i \leq k)$ of arity k denoting projection functions on records, and $\cup$ and $\cap$, both of arity 2 denoting union and intersection, respectively.

For every database transformation, a binary tuple constructor $(,)$ is indispensable. This is due to the formalisation of update that is a pair of a location and an update value as defined in Definition 7. The type constructor for finite multisets also plays a critical role in database transformations since a database transformation may have many subcomputations running in parallel, which yield possibly identical updates. As we will introduce later, by assigning location operators to locations, identical updates yielded during a computation can be aggregated to form a final update in an update set yielded by a one-step transition. Therefore, we need the type constructor for finite multisets to collect all updates generated during parallel computations.

The following are several multiset operations [5]. We use the constructor symbol $\langle\cdot\rangle$ for finite multisets with unfixed arity. Let x and y be two multisets, and M be a set of multisets.

- $x \uplus y$ returns a multiset that has members from x and y, and the occurrence of each member is the sum of the occurrences of such a member in x and in y.
- $\biguplus M$ returns a multiset that has members from all elements of M, and the occurrence of each member is the sum of the occurrences of such a member in all elements of M.
- $AsSet(x)$ returns a set that has the same members as x, such that

$$AsSet(x) \quad = \quad \{a| \ a \in x\}$$

- $\mathbf{I}x$ is defined by

$$\mathbf{I}x \quad = \quad \begin{cases} a & \text{if } x = \langle a \rangle \\ \bot & \text{otherwise} \end{cases}$$

Postulate 3 (background postulate). *Each state of a database transformation t must contain*

- *an infinite set of reserve values,*
- *truth values and their connectives, the equality predicate, the undefinedness value $\bot$, and*
- *a background class $\mathcal{K}$ defined by a background signature V_K that contains at least a binary tuple constructor $(\cdot)$, a finite multiset constructor $\langle \cdot \rangle$, and function symbols for operations such as pairing and projection for pairs, and empty multiset $\langle\rangle$, singleton $\langle x \rangle$, binary multiset union $\uplus$, general multiset union $\biguplus$, AsSet, and $\mathbf{I}x$ on multisets.*

The minimum requirements in the background postulate are the same as for parallel algorithms [5], but we leave it open how many other constructors will be in a background class in order to capture any request in data models.

Given the base set of a state S, we can add truth values and $\bot$, and partition them into base domains. Then the background class $\mathcal{K}$ contained in S can be obtained by applying the construction provided in Definition 8 to get a much larger base set and to interpret functions symbols in V_K with respect to this enlarged base set.

3.5 Bounded Exploration

The bounded exploration postulate for sequential algorithms requests that only finitely many terms can be updated in an elementary step [10]. For parallel algorithms this postulate becomes significantly more complicated, as basic constituents not involving any parallelism (so-called "proclets") have to be considered [5].

For database transformations the problem lies somehow in between. Computations are intrinsically parallel, even though implementations may be sequential, but the parallelism is restricted in the sense that all branches execute de facto the same computation. We will capture this by means of location operators, which generalise aggregation functions as in [7] and cumulative updates.

The idea behind location operators is inspired by the synchronisation of parallel updates in [5]. First, updates generated by parallel computations define an update multiset, then all updates to the same location are merged by means of a location operator to reduce the update multiset to an update set.

Definition 9. Let $M(D)$ be the set of all non-empty multisets over a domain D, then a *location operator* ρ over $M(D)$ consists of a unary function $f_\alpha : D \to D$, a commutative and associative binary operation $\odot$ over D, and a unary function $f_\beta : D \to D$, which define $\rho(m) = f_\beta(f_\alpha(b_1) \odot \cdots \odot f_\alpha(b_n))$ for $m = \langle b_1, ..., b_n \rangle \in M(D)$.

If a database transformation uses location operators, they must be defined in the algorithmic part of states requested in Postulate 2.

Example 5. *sum* is a location operator with $f_\alpha(v) = v$, $v_1 \odot v_2 = v_1 + v_2$, and $f_\beta(v) = v$. Another example is *avg* with $f_\alpha(v) = (v, 1)$, $(v_1, w_1) \odot (v_2, w_2) = (v_1 + v_2, w_1 + w_2)$, and $f_\beta((v, w)) = v \div w$.

Location operators define operations on multisets, and as such form an important part of logics for meta-finite structures [9]. They permit to express in a simple way the restricted parallelism in many database aggregate functions such as building sums or average values over query results, selecting maximum or minimum, and even structural recursion on sets, multisets, lists or trees.

Example 6. Consider the evaluation of a Boolean formula $\forall x \in D_1 \exists y \in D_2 \varphi(x, y)$. Assume the evaluation result will be stored at location ℓ. Let the cardinalities of D_1 and D_2 be n_1 and n_2, respectively. Then there are two nested parallel computations involved in the evaluation. The inner parallel computation for a specific value $u_i \in D_1$ ($i \in [1, n_1]$) has n_2 parallel branches, each of which evaluating a term $\varphi(u_i, v_j)$ for the various values $v_j \in D_2$ ($j \in [1, n_2]$), thus producing an update multiset with n_2 updates $(\ell, true)$ or $(\ell, false)$. Using $\theta(\ell) = \bigvee$ (logical OR) as location operator – in this case f_α and f_β are the identity function, and $\odot$ is $\vee$ – evaluates the inner existentially quantified formula, thereby producing another update multiset with n_1 entries $(\ell, true)$ or $(\ell, false)$. Using the location operator $\theta(\ell) = \bigwedge$ (logical AND) reduces this update multiset to a set with a single update.

Depending on the data model used and thus on the actual background signature we may use complex values, e.g. tree-structured values. As a consequence we have to cope with the problem of partial updates [11], e.g. the synchronisation of updates to different parts of the same tree values, or more generally complex database objects. Since updates may produced at different levels of abstraction, overlapping locations can lead to clashes. However, the issues relating to inconsistent update sets are irrelevant for the proof of the characterization theorem in Section 5 as shown in [5, 10]. That is, the problem of partial updates is subsumed by the problem of providing consistent update sets, in which there cannot be pairs (ℓ, v_1) and (ℓ, v_2) with $v_1 \neq v_2$.

The bounded exploration postulate in [10] for sequential algorithms is motivated by the *sequential accessibility principle*, which could be phrased as the request that each location must be uniquely identifiable. Leaving aside the discussion how to deal logically with partially defined terms unique identifiability can be obtained by using terms of the form $\mathbf{I}x.\varphi(x)$ with a formula φ, in which x is the only free variable. Such terms have to be interpreted as "the unique x satisfying formula $\varphi(x)$", which of course may be undefined, if no such x exists or more than one exist. According to the modified abstract state postulate 2 for database transformations the sequential accessibility principle must be preserved for the algorithmic part of the structure.

In principle, the claim of unique identifiability also applies to databases, as emphasised by Beeri and Thalheim in [4]. More precisely, unique identifiability has to be claimed for the basic updatable units in a database, e.g. objects in [15]. Unique identifiability, however, does not necessarily apply to all elements in a database. Sets of logically indistinguishable locations may be updated simultaneously. Nevertheless, for databases only logical properties are relevant – this is the so-called "genericity principle" in database theory [3] – and therefore, it must still be possible to use terms to access elements and locations in the database part of a state. These terms, however, may be non-ground. If a non-ground term identifies more than one location in a state S, these locations will be called accessible in parallel.

Definition 10. Let S be a state of the database transformation t. An element a of S is *accessible* if there is a ground term α in the signature of S that is interpreted as a in S. A location $f(a_1, \dots, a_n)$ is *accessible* if the elements $a_1, \dots, a_n$ are all accessible. An update $(f(a_1, \dots, a_n), b)$ is *accessible* if the location $f(a_1, \dots, a_n)$ and the element b are accessible.

Locations $f(a_1^1, \dots, a_n^1), \dots, f(a_1^m, \dots, a_n^m)$ with $f \in \Sigma_{db}$ are *accessible in parallel* if there exists a term α and an accessible element $b^{'}$, such that the values for which α is interpreted by $b^{'}$ in S are $f(a_1^1, \dots, a_n^1), \dots, f(a_1^m, \dots, a_n^m)$.

Updates $(f(a_1^1, \dots, a_n^1), b), \dots, (f(a_1^m, \dots, a_n^m), b)$ with $f \in \Sigma_{db}$ are *accessible in parallel* iff $f(a_1^1, \dots, a_n^1), \dots, f(a_1^m, \dots, a_n^m)$ are accessible in parallel and b is accessible.

The first part of Definition 10 is exactly the same as defined in [10, Definition 5.3]. The second part formalises our discussion above.

Example 7. Take a database transformation t with a ternary predicate symbol R in its signature. Let the interpretation of R in a state S be $\{(a, a, b), (a, b, c), (b, b, a), (b, a, c)\}$. Then $R(a, a, b)$ and $R(b, b, a)$ are accessible in parallel using the term $R(x, x, y)$ and the accessible element **true**.

The bounded exploration postulate in the sequential ASM thesis in [10] uses a finite set of ground terms as bounded exploration witness in the sense that whenever states S_1 and S_2 coincide over this set of ground terms the update set produced by the sequential algorithm is the same in these states. The intuition behind the postulate is that only the part of a state that is given by means of the witness will actually be explored by the algorithm.

The fact that only finitely many locations can be explored remains the same for database transformations. However, permitting parallel accessibility within the database part of a state forces us to slightly change our view on the bounded exploration witness. For this we need access terms.

Definition 11. An *access term* is either a ground term α or a pair (β, α) of terms, the variables $x_1, \dots, x_n$ in which must be database variables, referring to the arguments of some dynamic function symbol $f \in \Sigma_{db} \cup \{f_1, ..., f_\ell\}$. The interpretation of (β, α) in a state S is the set of locations

$$\{f(a_1, \dots, a_n) \mid val_{S,\zeta}(\beta) = val_{S,\zeta}(\alpha) \text{ with } \zeta = \{x_1 \mapsto a_1, \dots, x_n \mapsto a_n\}\}.$$

Structures S_1 and S_2 *coincide* over a set T of access terms if the interpretation of each $\alpha \in T$ and each $(\beta, \alpha) \in T$ over S_1 and S_2 are equal.

Instead of writing (β, α) for an access term, we should in fact write (f, β, α), but for simplicity we drop the function symbol f and assume it is implicitly given.

Due to our request that the database part of a state is always finite there will be a maximum number m of elements that are accessible in parallel. Furthermore, there is always a number n such that n variables are sufficient to describe the updates of a database transformation, and n can be taken to be minimal. Then for each state S the upper boundary of exploration is $\mathcal{O}(m^n)$, where m depends on S. Taking these together we obtain our fourth postulate.

Postulate 4 (bounded exploration postulate)**.** *For a database transformation t there exists a fixed, finite set T of access terms of t such that $\Delta(t, S_1) = \Delta(t, S_2)$ holds whenever the states S_1 and S_2 coincide over T.*

As in the sequential ASM thesis we continue calling the set T of access terms a *bounded exploration witness.* The only difference to the bounded exploration postulate for sequential algorithms in [10] is the use of access terms (β, α), whereas in the sequential ASM thesis only ground terms are considered. Access terms of the form (β, α) are actually equivalent to closed set comprehension terms $\{f(x_1, \dots, x_n) \mid \beta = \alpha\}$, i.e. they express first-order queries to the database similar to the relational calculus, and due to the fact that the database part of a state is a finite structure the set of locations defined by an access term is always finite. However, building terms on top of the state signature does not yet capture such terms. Access terms for the algorithmic part can still only be ground terms, otherwise finiteness cannot be guaranteed. Therefore, the modified Postulate 4 still expresses the same intention as the bounded exploration postulate for sequential algorithms does, i.e. only finitely many locations can be updated at a time, and these locations are determined by finitely many terms that appear in some way in the textual description of a database transformation.

3.6 Bounded Non-determinism

The last postulate addresses the question of how non-determinism is permitted in a database transformation. To handle this, we need to further clarify the relationship between access terms and states. As defined in the abstract state postulate, every state of a database transformation is a meta-finite structure consisting of two parts: the database part and algorithmic part, which are linked via a fixed, finite number of bridge functions. To restrict non-determinism in a database transformation t, we consider that *ground access terms* of t can access only the algorithmic part of a state, while *non-ground access terms* of t can access both the database and algorithmic parts of a state. Furthermore, variables in non-ground access terms are limited to range merely over the database part.

Example 8. Let us look back Example 2 again. We would have

- 2, 7.8, $+(3,9)$ and $\text{Even}(9)$ as ground access terms, and
- $(\text{Person}(x,y,z,z^{'}),\textbf{true})$, $(f_{num}(x),8)$, $(+(f_{num}(x),f_{num}(y)),20)$ and $(\text{Even}(f_{num}(x)),\textbf{false})$ as non-ground access terms.

Given a meta-finite structure with the signature $\Sigma = \Sigma_{db} \cup \Sigma_a \cup \{f_1,...,f_\ell\}$ and the base set B, i.e., $B = B_{db} \cup B_a$, we now formally define access terms.

Definition 12. A *ground access term* is defined by the following rules:

- $\alpha \in B_a$ is a ground access term, and
- $f(\alpha_1,...,\alpha_n)$ for n-ary function symbol $f \in \Sigma_a$ and ground access terms $\alpha_1,...,\alpha_n$ is a ground access term.

A *non-ground access term* is a pair (β,α) of terms, in which at least one of them is a non-ground term inductively defined by applying function symbols from Σ over variables in accordance with the definition of a meta-finite structure as in Definition 6.

We define equivalent substructures in the following sense.

Definition 13. Given two structures S' and S of the same signature Σ, a structure S' is a *substructure* of the structure S (notation: $S' \preceq S$) if

- the base set B' of S' is a subset of the base set B of S, i.e., $B' \subseteq B$, and
- for each function symbol f of arity n in the signature Σ the restriction of $val_S(f(x_1,...,x_n))$ to B' results in $val_{S'}(f(x_1,...,x_n))$.

Substructures $S_1, S_2 \preceq S$ are *equivalent* (notation: $S_1 \equiv S_2$) if there exists an automorphism $\sigma \in Aut(S)$ with $\sigma(S_1) = S_2$. The *equivalence class* of a substructure S' in the structure S is the subset of all substructures of S which are equivalent to S'.

Example 9. Let us consider a simple ternary relation schema R. Suppose our database contains $R(a,a,b_1), R(b,b,a_1), R(c_1,c,c_2)$. Then $R(a,a,b_1)$ defines a substructure with base set $\{a,b_1\}$. This substructure is equivalent to the substructure $R(b,b,a_1)$, as the isomorphism defined by the permutation $(a,b)(b_1,a_1)$ just swaps the two substructures.

If, however, we have a second relation schema $R^{'}$, and the database contains only $R^{'}(a,b_1)$ and $R^{'}(c,c)$, then the restriction to $\{a,b_1\}$ defines a substructure containing $R(a,a,b_1)$ and $R^{'}(a,b_1)$, whereas the restriction to $\{b,a_1\}$ defines a substructure $R(b,b,a_1)$ – these substructures are no longer equivalent.

If the database contained also $R^{'}(b,a_1)$, the two restrictions would again define equivalent substructures. □

Now we need to discuss the relationship between access terms and states, and explain how non-determinism can be restricted in a database transformation via access terms. Let us start with the simple case that states have no bridge functions and thereby only elements in the database part of a state are accessible in parallel via the interpretation of non-ground access terms. As the abstract state postulate captures the genericity of database transformations (i.e., preserved under isomorphisms [3]), an isomorphism between two states gives rise to an isomorphism between their corresponding successor states. Consequently, whenever a substructure $S^{'}$ of the database part is preserved in some successor state, each substructure in the equivalence class of $S^{'}$ is preserved in some (possibly same) isomorphic successor state. This is because the automorphism that interchanges two equivalent substructures permutes successor states, according to Definition 13. Nevertheless, it is also possible that there exists a successor state, in which none of substructures in the equivalence class of $S^{'}$ is preserved.

Example 10. Let us consider the relation $R = \{(a, a, b_1), (b, b, a_1), (c_1, c, c_2)\}$ in Example 9 again.

- Suppose that we non-deterministically delete a tuple from R in a state S. Then there will be three successor states of S with $R = \{(b, b, a_1), (c_1, c, c_2)\}$, $R = \{(a, a, b_1), (c_1, c, c_2)\}$ or $R = \{(a, a, b_1), (b, b, a_1)\}$, respectively. It is clear to see that for the equivalent substructures $R(b, b, a_1)$ and $R(a, a, b_1)$, whenever one of them is preserved in some successor state, another one is preserved in some (possibly same) isomorphic successor state. Note that not all of the successor states are isomorphic.
- If we non-deterministically select two tuples from R in a state S and delete them. Then we will also get three successor states of S: $R = \{(b, b, a_1)\}$, $R = \{(a, a, b_1)\}$ or $R = \{(c_1, c, c_2)\}$. In this case, in terms of the equivalence class $\{(a, a, b_1), (b, b, a_1)\}$, none of the equivalent substructures are preserved in the successor state of S with $R = \{(c_1, c, c_2)\}$.

In the case that states have bridge functions, however, the situation becomes a bit tricky because bridge functions define substructures of the algorithmic part based on substructures of the database part. Thus non-determinism caused by non-deterministically selecting elements in the database part may also result in the non-deterministic changes on substructures of the algorithmic part. Nevertheless, the distinction between the database and algorithmic parts is that non-determinism cannot arise from the algorithmic part by selecting non-deterministically substructures in the algorithmic part of a state.

Example 11. For the relation $R = \{(a, a, b_1), (b, b, a_1), (c_1, c, c_2)\}$ in Example 9, we assume that there exists a bridge function $f_{num} = \{(a, 4), (a_1, 6), (b, 3), (b_1, 1)(c, 5), (c_1, 8), (c_2, 7)\}$ and $\{\textsc{Even}, \textsc{Odd}, \textsc{Test}\} \subseteq \Sigma_a$. First we can use the non-ground access terms $(\textsc{Odd}(f_{num}(z)), \mathbf{true})$, $(\textsc{Even}(f_{num}(x)), \mathbf{true})$ and $(R(x, y, z), \mathbf{true})$ with the formula $R(x, y, z) \wedge \textsc{Even}(f_{num}(x)) \wedge \textsc{Odd}(f_{num}(z))$ to retrieve out the tuples (a, a, b_1) and (c_1, c, c_2) in R.

Then we can non-deterministically generate updates on function $\textsc{Test}$ by non-deterministically selecting one of these two tuples. For example, two update sets $\{(\textsc{Test}(4), 1)\}$ and $\{(\textsc{Test}(8), 7)\}$ may be created by using the access term $(\textsc{Test}(f_{num}(x)), f_{num}(z))$ together with the formula $R(x, y, z) \wedge$ $\textsc{Even}(f_{num}(x))$ $\wedge \textsc{Odd}(f_{num}(z))$.

Therefore, for a state of database transformations, substructures of its algorithmic part may or may not be preserved in its successor states. This indeed can be explained under a broader view on equivalence classes, i.e., they are defined in terms of a state taking all of the database part, the algorithmic part and bridge functions into consideration. Then the rationale that whenever a substructure of a state is preserved in some successor state, each substructure in the equivalence class of that substructure is preserved in some (possibly same) isomorphic successor state can still be captured by the abstract state postulate in the same way as in the case without bridge functions.

Now we formalise the bounded non-determinism postulate to capture these ideas by properly defining the presence of non-ground access terms. In doing so, we put a severe restriction on the non-determinism in the transition relation τ_t.

Postulate 5 (bounded non-determinism postulate)**.** *For a database transformation t, if there are states S_1, S_2 and $S_3 \in \mathcal{S}_t$ with $(S_1, S_2) \in \tau_t$, $(S_1, S_3) \in \tau_t$ and $S_2 \neq S_3$, then there exists a non-ground access term of the form (β, α) in the bounded exploration witness of t.*

According to this bounded non-determinism postulate, if a database transformation t over some state S_1 has non-determinism (i.e., $\Delta(t, S_1)$ contains more than one update set), then we must have a non-ground access term in the bounded exploration witness of t. Alternatively, if the bounded exploration witness of t contains only ground access terms, then t can access only the algorithmic part of a state and cannot have non-determinism. The bounded non-determinism postulate is motivated by the necessity of non-determinism in database queries and updates to permit identifier creation. This will become clear in the proof of our main result in Section 5, according to which the bounded non-determinism postulate enforces that only the bounded choice among database elements can be the source of non-determinism.

Remark 1. In [17, 18] Van den Bussche defined the notions of *determinacy* and *semi-determinism* - a determinate transformation preserves the input database, whereas a semi-deterministic transformation produces isomorphic outputs and thus preserves the input database up to an automorphism. In the sense of Van den Bussche an input database would define a substructure, and by the bounded non-determinism postulate preserving this substructure implies that each automorphism of the input database defines an isomorphism between the possible successor states, but not all of successor states will be isomorphic. Hence, database transformations characterised by five postulates subsume semi-deterministic transformations. In the same way they captures the insertion of new objects with a choice of identifiers as worked out for generic updates in [15].

3.7 Final Remarks

Naturally, for a database transformation the decisive part is the progression of the database part of states, whereas the algorithmic part could be understood as playing only a supporting role. Nonetheless, the postulates for database transformations in this section permit transformations, in which the major computation happens on the algorithmic part. In the extreme case we could even only manipulate the algorithmic part. This implies that our model actually subsumes all sequential algorithms. Furthermore, all extensions such as bounded non-determinism, meta-finite states, location operators and bounded exploration with non-ground terms only affect the database part. This will become more apparent in the next two sections, when we present a variant of ASMs capturing exactly database transformations as stipulated by the five postulates. On the other hand, our model of database transformations does not capture parallel algorithms, as the bounded exploration postulate excludes unbounded parallelism.

4 Database Abstract State Machines

In this section we define a variant of Abstract State Machines, called *Database Abstract State Machines*, and show that DB-ASMs satisfy the postulates of a database transformation. In the next section we will address the more challenging problem showing the converse of this result.

4.1 DB-ASM Rules and Update Sets Generated by Them

First we define DB-ASM rules r, and if S is a state, i.e. a Σ-structure for the signature Σ of r, we associate a set $\Delta(r, S)$ of update sets with r and S. For convenience, we also use the notation $\ddot{\Delta}(r, S)$ for a set of update multisets defined by r and S.

For the signature Σ we adopt the requirements of the abstract state postulate, i.e. it comprises a sub-signature Σ_{db} for the database part, a sub-signature Σ_a for the algorithmic part, and bridge functions $\{f_1, \dots, f_\ell\}$. For states we assume that the requirement in the abstract state postulate, according to which the restriction to Σ_{db} results in a finite structure, is satisfied. Furthermore, we assume a background in the sense of the background postulate being defined.

DB-ASM rules may involve variables, so in the following definition we also use the notations $\Delta(r, S, \zeta)$ for a set of update sets that depends on a variable assignment ζ, and analogously $\ddot{\Delta}(r, S, \zeta)$ for a set of update multisets. If ζ is a variable assignment, then $\zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]$ is another variable assignment defined by

$$\zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k](x) = \begin{cases} b_i & \text{if } x = x_i (i = 1, \dots, k) \\ \zeta(x) & \text{else} \end{cases}$$

We refer to *database variables* as variables that must be interpreted by values in B_{db}. The notation $var(t)$ is used to denote the set of variables occurring in a term

t. Similar to free variables occurring in formulae we can define the set $fr(r)$ of free variables appearing in a DB-ASM rule r. A rule r is called *closed* if $fr(r) = \emptyset$.

Definition 14. The set $\mathcal{R}$ of *DB-ASM rules* over a signature $\Sigma = \Sigma_{db} \cup \Sigma_a \cup \{f_1, \dots, f_\ell\}$ and associated sets of update sets (with respect to states as in Postulate 2 with a background as in Postulate 3) are defined as follows:

- If $t_0, \dots, t_n$ are terms over Σ, and f is a n-ary dynamic function symbol in Σ, then $f(t_1, \dots, t_n) := t_0$ is a rule r in $\mathcal{R}$ called *assignment rule* with $fr(r) = \bigcup\limits_{i=0}^{n} var(t_i)$. For a state S over Σ and a variable assignment ζ for $fr(r)$ we obtain

$$\Delta(r, S, \zeta) = \{\{(f(a_1, \dots, a_n), a_0)\}\}$$

with $a_i = val_{S,\zeta}(t_i)$ $(i = 0, \dots, n)$, and

$$\ddot{\Delta}(r, S, \zeta) = \{\langle (f(a_1, \dots, a_n), a_0) \rangle\}$$

- If φ is a Boolean term and $r' \in \mathcal{R}$ is a DB-ASM rule, then **if** φ **then** r' **endif** is a rule r in $\mathcal{R}$ called *conditional rule* with $fr(r) = fr(\varphi) \cup fr(r')$. For a state S over Σ and a variable assignment ζ for the variables in $fr(r)$, we obtain

$$\ddot{\Delta}(r, S, \zeta) = \begin{cases} \ddot{\Delta}(r', S, \zeta) & \text{if } val_{S,\zeta}(\varphi) = true \\ \emptyset & \text{else} \end{cases}$$

and

$$\Delta(r, S, \zeta) = \begin{cases} \Delta(r', S, \zeta) & \text{if } val_{S,\zeta}(\varphi) = true \\ \emptyset & \text{else} \end{cases}$$

- If φ is a Boolean term with only database variables, $\{x_1, \dots, x_k\} \subseteq fr(\varphi)$ and $r' \in \mathcal{R}$ is a DB-ASM rule, then **forall** $x_1, \dots, x_k$ **with** φ **do** r' **enddo** is a rule r in $\mathcal{R}$ called *forall rule* with $fr(r) = (fr(r') \cup fr(\varphi)) - \{x_1, \dots, x_k\}$. For a state S over Σ and a variable assignment ζ for the variables in $fr(r)$ let $\mathcal{B} = \{(b_1, \dots, b_k) \mid val_{S,\zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]}(\varphi) = true\}$ and $\mathcal{W}$ denote the set of mappings η from $\mathcal{B}$ to $\bigcup\{\Delta(r', S, \zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]) \mid (b_1, \dots, b_k) \in \mathcal{B}\}$ with $\eta(b_1, \dots, b_k) \in \Delta(r', S, \zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k])$. Then each $\eta \in \mathcal{W}$ defines an update set $\Delta_\eta = \bigcup\{\eta(b_1, \dots, b_k) \mid (b_1, \dots, b_k) \in \mathcal{B}\}$, from which we obtain

$$\Delta(r, S, \zeta) = \{\Delta_\eta \mid \eta \in \mathcal{W}\}.$$

Analogously, let $\ddot{\mathcal{W}}$ denote the set of mappings $\ddot{\eta}$ from $\mathcal{B}$ to $\bigcup\{\ddot{\Delta}(r', S, \zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]) \mid (b_1, \dots, b_k) \in \mathcal{B}\}$ with $\ddot{\eta}(b_1, \dots, b_k) \in \ddot{\Delta}(r', S, \zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k])$. Then each $\ddot{\eta} \in \ddot{\mathcal{W}}$ defines an update multiset $\ddot{\Delta}_{\ddot{\eta}} = \biguplus\{\ddot{\eta}(b_1, \dots, b_k) \mid (b_1, \dots, b_k) \in \mathcal{B}\}$, which finally gives

$$\ddot{\Delta}(r, S, \zeta) = \{\ddot{\Delta}_{\ddot{\eta}} \mid \ddot{\eta} \in \ddot{\mathcal{W}}\}.$$

- If $r_1, \dots, r_n$ are rules in $\mathcal{R}$, then the rule r defined as **par** $r_1 \dots r_n$ **endpar** is a rule in $\mathcal{R}$, called *parallel rule* with $fr(r) = \bigcup_{i=1}^{n} fr(r_i)$. For a state S over Σ and a variable assignment ζ for the variables in $fr(r)$ we obtain

$$\Delta(r, S, \zeta) = \{\Delta_1 \cup \dots \cup \Delta_n \mid \Delta_i \in \Delta(r_i, S, \zeta) \text{ for } i = 1, \dots, n\}$$

  and

$$\ddot{\Delta}(r, S, \zeta) = \{\ddot{\Delta}_1 \uplus \dots \uplus \ddot{\Delta}_n \mid \ddot{\Delta}_i \in \ddot{\Delta}(r_i, S, \zeta) \text{ for } i = 1, \dots, n\}.$$

- If φ is a Boolean term with only database variables, $\{x_1, \dots, x_k\} \subseteq fr(\varphi)$ and $r' \in \mathcal{R}$ is a DB-ASM rule, then **choose** $x_1, \dots, x_k$ **with** φ **do** r' **enddo** is a rule r in $\mathcal{R}$ called *choice rule* with $fr(r) = (fr(r') \cup fr(\varphi)) - \{x_1, \dots, x_k\}$. For a state S over Σ and a variable assignment ζ for the variables in $fr(r)$ let $\mathcal{B} = \{(b_1, \dots, b_k) \mid val_{S,\zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]}(\varphi) = true\}$. Then we obtain

$$\Delta(r, S, \zeta) = \bigcup \{\Delta(r', S, \zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]) \mid (b_1, \dots, b_k) \in \mathcal{B}\}.$$

  and

$$\ddot{\Delta}(r, S, \zeta) = \bigcup \{\ddot{\Delta}(r', S, \zeta[x_1 \mapsto b_1, \dots, x_k \mapsto b_k]) \mid (b_1, \dots, b_k) \in \mathcal{B}\}.$$

- If r_1, r_2 are rules in $\mathcal{R}$, then the rule r defined as **seq** r_1 r_2 **endseq** is a rule in $\mathcal{R}$, called *sequence rule* with $fr(r) = fr(r_1) \cup fr(r_2)$. For a state S over Σ and a variable assignment ζ for the variables in $fr(r)$ we obtain

$$\Delta(r, S, \zeta) = \{\Delta_1 \oslash \Delta_2 \mid \Delta_1 \in \Delta(r_1, S, \zeta) \text{ and } \Delta_2 \in \Delta(r_2, S + \Delta_1, \zeta)\}$$

  with update sets defined as

$$\Delta_1 \oslash \Delta_2 = \Delta_2 \cup \{(\ell, v) \in \Delta_1 \mid \neg \exists v'.(\ell, v') \in \Delta_2 \text{ and } v \neq v'\}$$

  and

$$\ddot{\Delta}(r, S, \zeta) = \{\ddot{\Delta}_1 \oslash \ddot{\Delta}_2 \mid \ddot{\Delta}_1 \in \ddot{\Delta}(r_1, S, \zeta) \text{ and } \ddot{\Delta}_2 \in \ddot{\Delta}(r_2, S + AsSet(\ddot{\Delta}_1), \zeta)\}$$

  with update multisets defined as

$$\ddot{\Delta}_1 \oslash \ddot{\Delta}_2 = \ddot{\Delta}_2 \uplus \langle (\ell, v) \in \ddot{\Delta}_1 \mid \neg \exists v'.(\ell, v') \in \ddot{\Delta}_2 \text{ and } v \neq v' \rangle.$$

- If $r' \in \mathcal{R}$ is a DB-ASM rule and θ is a location function that assigns location operators ρ to terms t with $var(t) \subseteq fr(r')$, then **let** $\theta(t) = \rho$ **in** r' **endlet** is a rule $r \in \mathcal{R}$ called *let rule* with $fr(r) = fr(r')$. For a state S over Σ and a variable assignment ζ for the variables in $fr(r)$ let $\ddot{\Delta}(r', S, \zeta) =$

$\{\ddot{\Delta}_1, \dots, \ddot{\Delta}_n\}$ with update multisets $\ddot{\Delta}_i = \ddot{\Delta}_i^{(t)} \uplus \ddot{\Delta}_i^-$ such that the first of these two multisubsets contains the updates to locations $val_{S,\zeta}(t)$, while the second one contains updates to all other locations. Define

$$\ddot{\Delta}_i^{(r)} = \langle (\ell, v) \mid \ell = val_{S,\zeta}(t), v = \rho(\langle v_1, \dots, v_k \mid (\ell, v_i) \in \ddot{\Delta}_i^{(t)} \rangle) \rangle \uplus \ddot{\Delta}_i^-$$

and

$$\Delta_i^{(r)} = \{(\ell, v) \mid \ell = val_{S,\zeta}(t), v = \rho(\langle v_1, \dots, v_k \mid (\ell, v_i) \in \ddot{\Delta}_i^{(t)} \rangle)\} \cup \Delta_i^-,$$

with $\Delta_i^- = \{(\ell, v) \mid (\ell, v) \in \ddot{\Delta}_i^-\}$. This finally gives

$$\ddot{\Delta}(r, S, \zeta) = \{\ddot{\Delta}_1^{(r)}, \dots, \ddot{\Delta}_n^{(r)}\} \quad \text{and} \quad \Delta(r, S, \zeta) = \{\Delta_1^{(r)}, \dots, \Delta_n^{(r)}\}.$$

Note that only assignment rules "create" updates in update sets and multisets, only choice rules introduce non-determinism, let rules reduce update multisets to update sets by letting updates to the same location collapse to a single update using assigned location operators, whereas all other rules just rearrange these updates into different sets and multisets, respectively. The sequence operator **seq** is associative, so we can also use more complex sequence rules **seq** $r_1 \dots r_n$ **endseq**.

Example 12. Consider the following DB-ASM rule

```
forall $x$ with $\exists z.R(x,x,z)$
do
    let $\theta(f(x)) = \mathtt{sum}$ in
        forall $y$ with $R(x,x,y)$
        do
            $f(x) := 1$
        enddo
    endlet
enddo
```

using `sum` as a shortcut for the location operator $(id, +, id)$. If the state contains the tuples $R(a,a,b), R(a,a,b^{'}), R(c,c,a^{'}), R(b,b,c), R(b,b,a^{'}), R(b,b,b), R(b^{'},a^{'},a)$, then first the update multisets $\langle (a,1),(a,1)\rangle, \langle (c,1)\rangle, \langle (b,1),(b,1),(b,1)\rangle$ are produced by means of the forall rules, which are then collapsed to the update set $\{(a,2),(c,1),(b,3)\}$ using the `sum`-operator in the let rule. Thus, for x such that there are tuples $R(x,x,y)$ in the database, then number of such tuples is counted and assigned to $f(x)$.

Let us denote the inner and outer forall rules as r_1 and r_2, respectively. Then we have

- $fr(r_1) = (\{x\} \cup fr(R(x,x,y))) - \{y\} = \{x\}$ and
- $fr(r_2) = (\{x\} \cup fr(\exists z.R(x,x,z))) - \{x\} = \emptyset$.

Hence this DB-ASM rule is closed.

Lemma 3. *Let r be a DB-ASM rule and $\sigma : S_1 \to S_2$ be an isomorphism between states S_1 and S_2. Let $S_1' = S_1 + \Delta$ be a successor state of S_1 for some $\Delta \in \Delta(r, S_1)$. Then we have $\sigma(\Delta) \in \Delta(r, S_2)$, and $\sigma : S_1' \to S_2' = S_2 + \sigma(\Delta)$ is an isomorphism between the successor states S_1' and S_2'.*

Proof. We proceed by structural induction on the rule r. So we start with an assignment rule $f(t_1, \dots, t_n) := t_0$. Then we must take $\Delta = \{(f(a_1, \dots, a_n), a_0)\}$ with $a_i = val_{S_1}(t_i)$ for $i = 0, \dots, n$. Then we have

$$val_{S_1'}(\ell) = \begin{cases} a_0 & \text{if } \ell = f(a_1, \dots, a_n) \\ val_{S_1}(\ell) & \text{else} \end{cases}$$

for any location ℓ. With $\sigma(\Delta) = \{(f(\sigma(a_1), \dots, \sigma(a_n)), \sigma(a_0))\}$ we obtain $val_{S_2'}(\sigma(\ell)) = \begin{cases} \sigma(a_0) & \text{if } \ell = f(a_1, \dots, a_n) \\ val_{S_2}(\sigma(\ell)) & \text{else} \end{cases} = \sigma(val_{S_1'}(\ell))$, which gives $S_2' = \sigma(S_1')$ as desired. The same argument applies to update multisets.

For a conditional rule $r =$ **if** φ **then** r' **endif** let $\zeta_2 = \sigma(\zeta_1)$. Then $val_{S_1,\zeta_1}(\varphi) = true$ iff $val_{S_2,\zeta_2}(\varphi) = true$. This implies

$$\begin{aligned} & S_2 + \Delta(r, S_2, \zeta_2) \\ = & \begin{cases} S_2 + \sigma(\Delta(r', S_1, \zeta_1)) & \text{if } val_{S_2,\zeta_2}(\varphi) = true \\ S_2 & \text{else} \end{cases} \\ = & \begin{cases} \sigma(S_1 + \Delta(r', S_1, \zeta_1)) & \text{if } val_{S_1,\zeta_1}(\varphi) = true \\ \sigma(S_1) & \text{else} \end{cases} \\ = & \ \sigma(S_1 + \Delta(r, S_1, \zeta_1)). \end{aligned}$$

The other cases are proven analogously.

□

4.2 Database Abstract State Machines

We are now prepared to define DB-ASMs and show that they satisfy the five postulates for database transformations from the previous section.

Definition 15. A *Database Abstract State Machine* (DB-ASM) $\mathcal{M}$ over signature Σ as in Postulate 2 and with a background as in Postulate 3 consists of

- a set $\mathcal{S}_{\mathcal{M}}$ of states over Σ, non-empty subsets $\mathcal{I}_{\mathcal{M}} \subseteq \mathcal{S}_{\mathcal{M}}$ of initial states and $\mathcal{F}_{\mathcal{M}} \subseteq \mathcal{S}_{\mathcal{M}}$ of final states, satisfying the requirements in Postulate 2,
- a closed DB-ASM rule $r_{\mathcal{M}}$ over Σ, and
- a binary relation $\tau_{\mathcal{M}}$ over $\mathcal{S}_{\mathcal{M}}$ determined by $r_{\mathcal{M}}$ such that

  $$\{S_{i+1} \mid (S_i, S_{i+1}) \in \tau_{\mathcal{M}}\} = \{S_i + \Delta \mid \Delta \in \Delta(r_{\mathcal{M}}, S_i)\}$$

  holds.

Theorem 6. *Each DB-ASM $\mathcal{M}$ defines a database transformation t with the same signature and background as $\mathcal{M}$.*

Proof. We have to show that the five postulates for database transformations are satisfied. As for the sequential time and background postulates 1 and 3, these are already built into the definition of a DB-ASM. The same holds for the abstract state postulate 2 as far as the definition of states is concerned, and the preservation of isomorphisms follows from Lemma 3. Thus, we have to concentrate on the bounded exploration and bounded non-determinism postulates 4 and 5.

Regarding bounded exploration we noted above that assignment rules within a DB-ASM rule r that defines $\tau_{\mathcal{M}}$ are decisive for a set $\Delta(r, S)$ of update sets over any state S. Hence, if $f(t_1, \dots, t_n) := t_0$ is an assignment rule occurring within r, and $val_{S,\zeta}(t_i) = val_{S',\zeta}(t_i)$ holds for all $i = 0, \dots, n$ and all variable assignments ζ that have to be considered, then we obtain $\Delta(r, S) = \Delta(r, S')$.

We use this to define a bounded exploration witness T. If t_i is ground, we add the access term $\alpha = t_i$ to T. If t_i is not ground, then the corresponding assignment rule must appear within the scope of forall and choice rules introducing the database variables in t_i, as r is closed. Thus, variables in t_i are bound by a Boolean term φ, i.e. for $fr(t_i) = \{x_1, \dots, x_k\}$ the relevant variable assignments are $\zeta = \{x_1 \mapsto b_1, \dots, x_k \mapsto b_k\}$ with $val_{S,\zeta}(\varphi) = true$. Bringing φ into a form that only uses conjunction, negation and existential quantification with atoms $\beta_i = \alpha_i$ $(i = 1, \dots, \ell)$, we can extract a set of access terms $\{(\beta_1, \alpha_1), \dots, (\beta_\ell, \alpha_\ell)\}$ such that if S and S' coincide on these access terms, they will also coincide on the formula φ. This is possible, as we evaluate access terms by sets, so conjunction corresponds to union, existential quantification to projection, and negation to building the (finite) complement. We add all the access terms $(\beta_1, \alpha_1), \dots, (\beta_\ell, \alpha_\ell)$ to T.

More precisely, if φ is a conjunction $\varphi_1 \wedge \varphi_2$, then $\Delta(r, S_1) = \Delta(r, S_2)$ will hold, if $\{(b_1, \dots, b_k) \mid val_{S_1,\zeta}(\varphi) = true\} = \{(b_1, \dots, b_k) \mid val_{S_2,\zeta}(\varphi) = true\}$ holds (with $\zeta = \{x_1 \mapsto b_1, \dots, x_k \mapsto b_k\}$). If T_i is a set of access terms such that whenever S_1 and S_2 coincide on T_i, then $\{(b_1, \dots, b_k) \mid val_{S_1,\zeta}(\varphi_i) = true\} = \{(b_1, \dots, b_k) \mid val_{S_2,\zeta}(\varphi_i) = true\}$ will hold $(i = 1, 2)$, then $T_1 \cup T_2$ is a set of access terms such that whenever S_1 and S_2 coincide on $T_1 \cup T_2$, then $\{(b_1, \dots, b_k) \mid val_{S_1,\zeta}(\varphi) = true\} = \{(b_1, \dots, b_k) \mid val_{S_2,\zeta}(\varphi) = true\}$ will hold.

Similarly, a set of access terms for ψ with the desired property will also be a witness for $\varphi = \neg\psi$, and $\bigcup_{b_{k+1} \in B_{db}} T_{b_{k+1}}$ with sets of access terms $T_{b_{k+1}}$ for $\psi[x_{k+1}/t_{k+1}]$ with $val_S(t_{k+1}) = b_{k+1}$ defines a finite set of access terms for $\varphi = \exists x_{k+1}\psi$. In this way, we can restrict ourselves to atomic formulae, which are equations and thus give rise to canonical access terms.

Then by construction, if S and S' coincide on T, we obtain $\Delta(r, S) = \Delta(r, S')$. As there are only finitely many assignment rules within r and only finitely many choice and forall rules defining the variables in such assignment rules, the set T of access terms must be finite, i.e. r satisfies the bounded exploration postulate.

Regarding bounded non-determinism, assuming that $\mathcal{M}$ does not satisfy the bounded non-determinism postulate. It means that there does not exist any non-ground access term (β, α) in T even when $\Delta(r, S)$ contains more than one update sets. However, according to our remark above r must contain a choice rule **choose** $x_1, \dots, x_k$ **with** φ **do** r' **enddo**. Hence, it implies that there exist at least one non-ground access term in T contradicting our assumption. $\square$

5 A Characterisation Theorem

In this section we want to show that DB-ASMs capture all database transformations. This constitutes the converse of Theorem 6, i.e. that every database transformation can be behaviouraly simulated by a DB-ASM. We start with some preliminaries that are only slight adaptations of corresponding definitions and results for the sequential ASM-thesis, except that the term *critical value* has to be defined differently due to the use of variables in access terms. We then show first that a one-step transition from a state to successor states can be expressed by a DB-ASM rule. Here we rely heavily on the abstract state postulate and the bounded non-determinism postulate, which allows us to deal with the restricted non-determinism appropriately.

In a second step we generalise the proof to the complete database transformation using a construction that is similar to the one used in the sequential ASM-thesis. Again, the fact that our bounded exploration witness contains non-ground terms makes up most of the difficulty.

5.1 Critical Terms and Critical Elements

Throughout this section we only deal with consistent update sets, which define the progression of states in a run. We now start providing the key link from updates as implied by the state transitions to DB-ASM rules. Same as the previous subsection this is only a slight extension to the work done for the sequential ASM thesis.

Definition 16. Let T be a bounded exploration witness for the database transformation t. A term that is constructed out of the subterms of $\alpha \in T$ and variables $x_1, \dots, x_k$, for which there are access terms $(\beta_1, \alpha_1), \dots, (\beta_\ell, \alpha_\ell) \in T$ such that $\bigcup_{i=1}^{\ell} fr(\beta_i) \cup fr(\alpha_i) = \{x_1, \dots, x_k\}$ holds is called a *critical term.*

This definition differs from the one given in [10] in that we consider also non-ground terms. For access terms in T we cannot simply require closure under subterms, as coincidence of structures on T does not carry over to subterms of "associative" access terms (β, α). Therefore, we need a different approach to define critical values.

If γ is a critical term, let $(\beta_1, \alpha_1), \dots, (\beta_\ell, \alpha_\ell)$ be the access terms used in its definition. For a state S choose $b_1, \dots, b_k \in B_{db}$ with $val_{S,\zeta}(\beta_i) = val_{S,\zeta}(\alpha_i)$ with $\zeta = \{x_1 \mapsto b_1, \dots, x_k \mapsto b_k\}$ for $i = 1, \dots, \ell$, and let $a = val_{S,\{x_1 \mapsto b_1, \dots, x_k \mapsto b_k\}}(\gamma)$.

Definition 17. For each state S of a database transformation t, let $C_S = \{val_S(\alpha) \mid \alpha \in T\} \cup \{true, false, \bot\}$ and $B_S = \{a_i \mid f(a_1, \dots, a_n) \in val_S(\beta, \alpha)$ for some access term $(\beta, \alpha) \in T\}$. Then $\bar{C}_S$ is the *background closure* of $C_S \cup B_S$ containing all complex values that can be constructed out of $C_S \cup B_S$ using the constructors and function symbols (interpreted in S) in V_K. The elements of $\bar{C}_S$ are called the *critical elements* of S.

The following lemma and its proof are analogous to the result in [10, Lemma 6.2].

Lemma 4. *For all updates $(f(a_1, \dots, a_n), a_0) \in \Delta(t, S, S')$ for $(S, S') \in \tau_t$ the values $a_0, \dots, a_n$ are critical elements of S.*

Proof. Assume one of the a_i is not critical. Then choose a structure S_1 by replacing a_i with a fresh value b without changing anything else. Thus, S_1 is a state isomorphic to S by the abstract state postulate.

Let (β, α) be an access term in T. Then we must have $val_S(\beta, \alpha) = val_{S_1}(\beta, \alpha)$, so S and S_1 coincide on T. From the bounded exploration postulate we obtain $\Delta(t, S) = \Delta(t, S_1)$ and thus $(f(a_1, \dots, a_n), a_0) \in \Delta(t, S_1, S_1')$ for some $(S_1, S_1') \in \tau_t$.

However, a_i does not appear in the structure S_1, and hence cannot appear in S_1' either, nor in $\Delta(t, S_1, S_1')$, which gives a contradiction. $\square$

5.2 Rules for One-Step Updates

In [10] it is a straighforward consequence of Lemma 6.2 that individual updates can be represented by assignments rules, and consistent update sets by par-blocks of assignment rules. In our case showing that $\Delta(t, S)$ can be represented by a DB-ASM rule requires a bit more work, which relies heavily on the abstract state postulate and the bounded non-determinism postulate. We address this in the next lemma.

Lemma 5. *Let t be a database transformation. For every state $S \in \mathcal{S}_t$ there exists a rule r_S such that $\Delta(t, S) = \Delta(r_S, S)$, and r_S only uses critical terms.*

Proof. $\Delta(t, S)$ is a set of update sets. Let $\{S_1, \dots, S_m\} = \{S' \mid (S, S') \in \tau_t\}$. Then $\Delta(t, S) = \{\Delta(t, S, S_i) \mid 1 \leq i \leq m\}$.

Now consider any update $u = (f(a_1, \dots, a_n), a_0) \in \Delta(t, S, S_i)$ for some $i \in \{1, \dots, m\}$. According to Lemma 4 the values $a_0, \dots, a_n$ are critical and hence representable by terms involving variables from access terms in T, i.e. $a_i = val_{S,\zeta}(t_i)$ with either $fr(t_i) \subseteq \{x_1, \dots, x_k\}, \zeta = \{x_1 \mapsto b_1, \dots, x_k \mapsto b_k\}$ and

$$(b_1, \dots, b_k) \in \mathcal{B}_u = \{(b_1, \dots, b_k) \in B_{db}^k \mid \bigwedge_{1 \leq i \leq \ell} val_{S,\zeta}(\beta_i) = val_{S,\zeta}(\alpha_i)\}$$

with access terms $(\beta_i, \alpha_i) \in T$ $(i = 1, \dots, \ell)$ and $fr(\beta_i) \subseteq \{x_1, \dots, x_k\}$, or t_i is a ground critical term.

Therefore, we distinguish two cases:

I. At least one of the terms $t_0, \dots, t_n$ is not a ground term.

II. All terms $t_0, \dots, t_n$ are ground terms.

Case I. We first assume that none of terms $t_0, \dots, t_n$ contain location operators. The access terms (β_i, α_i) define a finite set of locations

$$\begin{aligned} L \quad = \quad & \{f(a_1, \dots, a_n) \mid a_i = val_{S,\zeta}(t_i) \text{ for } i = 1, \dots, n, \text{ and} \\ & \zeta = \{x_1 \mapsto b_1, \dots, x_k \mapsto b_k\} \text{ for } (b_1, \dots, b_k) \in \mathcal{B}_u\}. \end{aligned}$$

However, instead of looking at updates at these locations we switch to a relational perspective, i.e. we replace $f \in \Sigma$ with arity n by a relation symbol Rf of arity $n+1$, so $f_S(a_1, \dots, a_n) = a_0$ holds iff $Rf_S(a_1, \dots, a_n, a_0) = true$. A non-trivial update $u = (f(a_1, \dots, a_n), a_0)$ is accordingly represented by two relational updates

$$u_d = (Rf(a_1, \dots, a_n, f_S(a_1, \dots, a_n)), false) \text{ and } u_i = (Rf(a_1, \dots, a_n, a_0), true).$$

So, instead of locations in L we consider locations in $L_{pre} \cup L_{post}$ with

$$\begin{aligned} L_{pre} = & \{Rf(a_1, \dots, a_n, a_0) \mid a_i = val_{S,\zeta}(t_i) \text{ for } 1 \leq i \leq n, \\ & a_0 = val_{S,\zeta}(f(t_1, \dots, t_n)) \text{ for } \zeta = \{x_1 \mapsto b_1, \dots, x_k \mapsto b_k\} \\ & \text{and } (b_1, \dots, b_k) \in \mathcal{B}_u\} \end{aligned}$$

and

$$\begin{aligned} L_{post} = & \{Rf(a_1, \dots, a_n, a_0) \mid a_i = val_{S,\zeta}(t_i) \text{ for } 0 \leq i \leq n \\ & \text{for } \zeta = \{x_1 \mapsto b_1, \dots, x_k \mapsto b_k\} \text{ and } (b_1, \dots, b_k) \in \mathcal{B}_u\}. \end{aligned}$$

Furthermore, we may assume that the set $\mathcal{B}_u$ is minimal in the sense that we may not find additional access terms that would define a subset $\mathcal{B}'_u \subsetneq \mathcal{B}_u$ still containing the value tuple $(b_1, \dots, b_k)$ that is needed to define the update u.

Then each tuple $(a_1, \dots, a_n, a_0) \in L_{pre} \cup L_{post}$ defines a substructure of S with base set $B' = \{a_0, \dots, a_n, true, false\}$ and all functions (in fact: relations) restricted to this base set. In doing so all substructures defined by L_{pre} (and analogously by L_{post}) are pairwise equivalent, and the induced isomorphisms are defined by permutations of tuples in $\mathcal{B}_u$. If they were not equivalent, we could find a distinguishing access structure $(\beta_{\ell+1}, \alpha_{\ell+1}) \in T$ that would define a subset $\mathcal{B}'_u \subsetneq \mathcal{B}_u$ thereby violating the minimality assumption for $\mathcal{B}_u$. Let E_{pre} and E_{post} denote these equivalence classes of substructures, respectively.

If $\ell' \in L$ is not updated in S_i – hence, corresponding locations in L_{pre} and L_{post} are neither updated – then the substructures $S_{\ell'}$ in E_{pre} (and E_{post}, respectively) that are defined by ℓ' are preserved in S_i, i.e. $S_{\ell'} \preceq S_i$. From Lemma 2 we can get that $\sigma(\Delta(t, S)) = \Delta(t, \sigma(S)) = \Delta(t, S)$ for the case that σ is an automorphism of S. It means that for an update in $\Delta(t, S, S_i)$ there is a translated update (by means of σ) in $\Delta(t, S, \sigma(S_i))$. Then we conclude that for every $\ell'' \in L$ there is some successor state $S_j = \sigma'(S_i)$ of S with $S_{\ell''} \preceq S_j$ for an automorphism σ' of S that maps $S_{\ell'}$ to $S_{\ell''}$, and each $S_{\ell''} \in E_{pre}$ (and $S_{\ell''} \in E_{post}$, respectively) is preserved in some S_j. Thus, there is some successor state S_j of S with $S_\ell \preceq S_j$ for $u = (\ell, a_0)$.

Then, we obtain two subcases:

1) If ℓ is updated in all $S_1, \dots, S_m$, i.e. there exist values $a_0^1, \dots, a_0^m$ with $(\ell, a_0^i) \in \Delta(t, S, S_i)$ for all $i = 1, \dots, m$, then all $\ell' \in L$ are also updated in all S_i. If (ℓ, a_0^i) is represented by the assignment rule $f(t_1, \dots, t_n) := t_0^i$ with $x_1, \dots, x_k$ interpreted by $(b_1, \dots, b_k) \in \mathcal{B}_u$, then the fact that almost all substructures defined by these interpretations of $t_1, \dots, t_n$ and any value other than a_0^i are preserved – and hence by virtue of Lemma 2 as we explained before equivalent substructures are preserved in the other S_j – implies that each instantiation of the rule $f(t_1, \dots, t_n) := t_0^i$ with values from $\mathcal{B}_u$ defines an update in one of the update sets $\Delta(t, S, S_j)$. Hence these updates can be collectively represented by the rule

```
**choose** $x_1, \dots, x_k$ **with** $\beta_1(\vec{x}_1) = \alpha_1(\vec{x}_1) \wedge \dots \wedge \beta_\ell(\vec{x}_\ell) = \alpha_\ell(\vec{x}_\ell)$
**do** $f(t_1, \dots, t_n) := t_0^i$ **enddo**
```

Here the $\vec{x}_1, \dots, \vec{x}_\ell$ denote vectors of variables among $x_1, \dots, x_k$ appearing in $\beta_1, \dots, \beta_\ell$, respectively. In case all the terms t_0^i for $i = 1, \dots, m$ are identical (to say t_0), we obtain in fact the rule $r_S^{(u)}$ as

```
**choose** $x_1, \dots, x_k$ **with** $\beta_1(\vec{x}_1) = \alpha_1(\vec{x}_1) \wedge \dots \wedge \beta_\ell(\vec{x}_\ell) = \alpha_\ell(\vec{x}_\ell)$
**do** $f(t_1, \dots, t_n) := t_0$ **enddo**
```

In general, however, it is possible that different terms t_0^i must be chosen, so all updates to locations in L are represented by the rule $r_S^{(u)}$, which becomes

```
**choose** $x_1^{(1)}, \dots, x_k^{(1)}, \dots, x_1^{(m)}, \dots, x_k^{(m)}$
**with** $\bigwedge_{1 \le j_1 < j_2 \le m} (x_1^{(j_1)}, \dots, x_k^{(j_1)}) \neq (x_1^{(j_2)}, \dots, x_k^{(j_2)})$
            $\wedge \bigwedge_{1 \le j \le m} \beta_1(\vec{x}_1^{(j)}) = \alpha_1(\vec{x}_1^{(j)}) \wedge \dots \wedge \beta_\ell(\vec{x}_\ell^{(j)}) = \alpha_\ell(\vec{x}_\ell^{(j)})$
**do par**
   $f(t_1, \dots, t_n)[x_1^{(1)}/x_1, \dots, x_k^{(1)}/x_k] := t_0^1[x_1^{(1)}/x_1, \dots, x_k^{(1)}/x_k]$
          $\vdots$          $\vdots$
   $f(t_1, \dots, t_n)[x_1^{(m)}/x_1, \dots, x_k^{(m)}/x_k] := t_0^m[x_1^{(m)}/x_1, \dots, x_k^{(m)}/x_k]$
**endpar enddo**
```

2) If only ℓ is updated in S_i, but no other $\ell'' \in L$ is, then, for each $\ell' \in L$, only ℓ' is updated in some S_j for $j \in [1, m]$, which is isomorphic to S_i. Analogously, if only i locations in L are updated in S_i, then for any $\{\ell_1, ..., \ell_i\} \subseteq L$ there is some state S_j for $j \in [1, m]$, in which only locations $\ell_1 \dots \ell_i$ are updated. Using exactly the same arguments as in case 1), we now can represent these updates collectively by the rule $r_S^{(u)}$, which now becomes

```
choose $x_1^{(1)}, \dots, x_k^{(1)}, \dots, x_1^{(i)}, \dots, x_k^{(i)}$
with $\bigwedge_{1 \le j_1 < j_2 \le i} (x_1^{(j_1)}, \dots, x_k^{(j_1)}) \neq (x_1^{(j_2)}, \dots, x_k^{(j_2)})$
        $\wedge \bigwedge_{1 \le j \le i} \beta_1(\vec{x}_1^{(j)}) = \alpha_1(\vec{x}_1^{(j)}) \wedge \dots \wedge \beta_\ell(\vec{x}_\ell^{(j)}) = \alpha_\ell(\vec{x}_\ell^{(j)})$
do par
  $f(t_1, \dots, t_n)[x_1^{(1)}/x_1, \dots, x_k^{(1)}/x_k] := t_0^1[x_1^{(1)}/x_1, \dots, x_k^{(1)}/x_k]$
        $\vdots$                    $\vdots$
  $f(t_1, \dots, t_n)[x_1^{(i)}/x_1, \dots, x_k^{(i)}/x_k] := t_0^i[x_1^{(i)}/x_1, \dots, x_k^{(i)}/x_k]$
endpar enddo
```

By exploiting Lemma 2, we showed how to create a proper choice rule with respect to $\mathcal{B}_u$ that is minimal. However, the created choice rule does not capture all update sets in $\Delta(t, S)$. If there exists another update in an update set that is not in the orbit of $\Delta(t, S, S_i)$ $(1 \le i \le m)$ under σ, we can use the same argument to obtain another choice rule. As the orbits are disjoint, we end up with a choice of choice rules, which can be combined into a single choice rule. The underlying condition for constructing such a single choice rule is the finiteness of $\Delta(t, S)$, i.e., there are only finitely many update sets created by t over state S. This can be assured by Lemma 4 and the bounded non-determinism postulate. Consequently there can only be finitely many successor states for each state S, depending on the database part of S that is a finite structure.

Now we revise the previous assumption that none of terms $t_0, \dots, t_n$ contain location operators to a general case, i.e., location operators may appear in the terms $t_0, ..., t_n$ of an assignment rule $f(t_1, ..., t_n) := t_0$. Let f_ℓ be a unary function symbol such that $x_{t_i} = f_\ell(i)$, then, without loss of generality, we can replace the terms $t_1, ..., t_n$ of an assignment rule $f(t_1, ..., t_n) := t_0$ with the variables $x_{t_1}, ..., x_{t_n}$, such that

```
seq
  par
    $x_{t_1} := t_1$
        $\vdots$
    $x_{t_n} := t_n$
  endpar
  $f(x_{t_1}, ..., x_{t_n}) := t_0$
endseq
```

It means that we can simplify the construction of rules for updates which may correspond to terms with location operators by only considering the case that location operators appear at the right hand side of an assignment rule. If a term t_i $(i \in [1, n])$ at the left hand side contains a location operator, by the above translation, we may treat it as being a term at the right hand side of another assignment rule again.

Suppose that the outermost function symbol of term t_0 is a location operator ρ, e.g.,$t_0 = \rho(m)$ where $m = \langle t_0^{'} |$ for all values $\overline{a} = (a_1, ..., a_p)$ in $\overline{y} = (y_1, ..., y_p)$ such that $val_{S,\zeta[x_1 \mapsto b_1,...,x_k \mapsto b_k]}(\varphi(\overline{x}, \overline{y})) = true\rangle$, and $\overline{x}$ denotes a tuple of variables among $x_1, ..., x_k$. Then for each assignment rule $f(x_{t_1}, ..., x_{t_n}) := t_0$ in which t_0 contains a location operator as described before, we can construct the following rule to remove the location operator ρ by a let rule and a forall rule:

```
**let** $\theta(f(x_{t_1}, ..., x_{t_n})) = \rho$ **in**
    **forall** $y_1$,...,$y_p$ **with** $\varphi(\overline{x}, \overline{y})$
    **do**
        $f(x_{t_1}, ..., x_{t_n}) := t_0^{'}$
    **enddo**;
**endlet**
```

This construction can be conducted iteratively. If the outermost function symbol of the above term $t_0^{'}$ is a location operator, then we need to construct a rule in a similar way to replace the assignment rule $f(x_{t_1}, ..., x_{t_n}) := t_0^{'}$. This procedure continues until the right hand side of an assignment rule is a term without any location operator.

Case II. In case of a simple update $f(t_1, \dots, t_n) := t_0$ without free variables we consider the substructure defined by $\{a_1, \dots, a_n, val_S(f(t_1, \dots, t_n)), true, false\}$ as before. However, in this case it is the only substructure in its equivalence class. Furthermore, the substructure can be represented by ground access terms. According to the bounded non-determinism postulate, we know that, ground access terms can access only the algorithmic part of a state and there is no non-determinism. Consequently, $(f(a_1, \dots, a_n), a_0) \in \Delta(t, S, S_i)$ for all $i = 1, \dots, m$, and these updates can be collectively represented by the simple assignment rule $r_S^{(u)}$, which now becomes

$$f(t_1, \dots, t_n) := t_0.$$

Finally, we construct r_S by using the **par**-construct:

$$r_S = \textbf{par}\ r_S^{(u_1)} \dots r_S^{(u_p)}\ \textbf{endpar}$$

for $\{u_1, \dots, u_p\} = \bigcup_{i=1}^{m} \Delta(t, S, S_i)$. □

5.3 Rules for Multiple-Steps Updates

Let us now extend Lemma 5 to the construction of a DB-ASM rule that captures the complete behaviour of a database transformation t. According to Definition 16, a set of critical terms can be obtained from a bounded exploration witness. For this fix a bounded exploration witness T and the set CT of critical terms derived from it. Furthermore, for a state S of t fix the rule r_S as in Lemma 5.

For $\gamma \in CT$ let $(\beta_1, \alpha_1), \dots, (\beta_\ell, \alpha_\ell)$ be the access terms in T defining $fr(\gamma) = \{x_1, \dots, x_k\}$. For a state $S \in \mathcal{S}_t$ define

$$val_S(\gamma) = \{val_{S,\zeta}(\gamma) \mid \zeta = (x_1 \mapsto b_1, \dots, x_k \mapsto b_k) \text{ and } \bigwedge_{1 \le i \le \ell} val_{S,\zeta}(\beta_i) = val_{S,\zeta}(\alpha_i)\}.$$

The following two lemmata extend Lemma 5 first to state that coincide with S on critical terms, then to isomorphic states.

Lemma 6. *Let $S, S' \in \mathcal{S}_t$ be states that coincide on the set CT of critical terms. Then $\Delta(r_S, S') = \Delta(t, S')$ holds.*

Proof. As S and S' coincide on CT, they also coincide on T, which gives $\Delta(t, S) = \Delta(t, S')$ by the bounded exploration postulate. Furthermore, we have $\Delta(r_S, S) = \Delta(t, S)$ by Lemma 5. As r_S uses only critical terms, the updates produced in state S must be the same as those produced in state S', i.e. $\Delta(r_S, S) = \Delta(r_S, S')$, which proves the lemma. □

Lemma 7. *Let S, S_1, S_2 be states with S_1 isomorphic to S_2 and $\Delta(r_S, S_2) = \Delta(t, S_2)$. Then also $\Delta(r_S, S_1) = \Delta(t, S_1)$ holds.*

Proof. Let σ denote an isomorphism from S_1 to S_2. Then $\Delta(r_S, S_2) = \sigma(\Delta(r_S, S_1))$ holds by Lemma 2, and the same applies to $\Delta(t, S_2) = \sigma(\Delta(t, S_1))$. As we presume $\Delta(r_S, S_2) = \Delta(t, S_2)$, we obtain $\sigma(\Delta(r_S, S_1)) = \sigma(\Delta(t, S_1))$ and hence $\Delta(r_S, S_1) = \Delta(t, S_1)$, as σ is an isomorphism. □

Next, in the spirit of [10] we want to extend the equality of sets of update sets for t and r_S to a larger class of states by exploiting the finiteness of the bounded exploration witness T. For this we define the notion of T-equivalence similar to the corresponding notion for the sequential ASM thesis, with the difference that in our case we cannot take T, but must base our definition and the following lemma on CT.

Definition 18. States $S, S' \in \mathcal{S}_t$ are called *T-similar* iff $E_S = E_{S'}$ holds, where E_S is an equivalence relation on CT defined by

$$E_S(\gamma_1, \gamma_2) \Leftrightarrow val_S(\gamma_1) = val_S(\gamma_2).$$

Lemma 8. *We have $\Delta(r_S, S') = \Delta(t, S')$ for every state S' that is T-similar to S.*

Proof. Replace every element in S' that also belongs to S by a fresh element. This defines a structure S_1 isomorphic to S' and disjoint from S. By the abstract state postulate S_1 is a state of t. Furthermore, by construction S_1 is also T-similar to S' and hence also to S.

Now define a structure S_2 isomorphic to S_1 such that $val_{S_2}(\gamma) = val_S(\gamma)$ holds for all critical terms $\gamma \in CT$. This is possible, as S and S_1 are T-similar, i.e. we have $val_S(\gamma_1) = val_S(\gamma_2)$ iff $val_{S_1}(\gamma_1) = val_{S_1}(\gamma_2)$ for all critical terms γ_1, γ_2. By the abstract state postulate S_2 is also a state of t.

Using Lemma 6 we conclude $\Delta(r_S, S_2) = \Delta(t, S_2)$, and by Lemma 7 we obtain $\Delta(r_S, S') = \Delta(t, S')$ as claimed. □

We are now able to prove our main result, first generalising Lemma 5 to multiple-steps updates in the next lemma, from which the proof of the main characterisation theorem is straightforward.

Lemma 9. *Let t be a database transformation with signature Σ. Then there exists a DB-ASM rule r over Σ, with same background as t such that $\Delta(r, S) = \Delta(t, S)$ holds for all states $S \in \mathcal{S}_t$.*

Proof. In order to decide whether equivalence relations E_S and $E_{S'}$ coincide for states $S, S' \in \mathcal{S}_t$ it is sufficient to consider the subset $CT' \subseteq CT$ defined by the bounded exploration witness T as in Definition 16. Hence, as T is finite, CT' is also finite, and consequently there can only be finitely many such equivalence relations. Let these be $E_{S_1}, \dots, E_{S_n}$ for states $S_1, \dots, S_n \in \mathcal{S}_t$.

For $i = 1, \dots, n$ construct Boolean terms φ_i such that $val_S(\varphi_i) = true$ holds iff S is T-similar to S_i. For this let $CT' = \{\gamma_1, \dots, \gamma_m\}$, and define terms

$$\bar{\gamma}_j = \begin{cases} \gamma_j & \text{if } \gamma_j \text{ is closed} \\ \langle (x_1, \dots, x_k) \mid \bigwedge\limits_{1 \le i \le \ell} \beta_i = \alpha_i \rangle & \text{if } \gamma_j = (x_1, \dots, x_k) \text{ with variables taken} \\ & \text{from } (\beta_1, \alpha_1), \dots, (\beta_\ell, \alpha_\ell) \end{cases}$$

exploiting the fact that the background structures provide constructors for multisets and pairs (and thus also tuples). Then

$$\varphi_i = \bigwedge_{\substack{1 \le j_1, j_2 \le m \\ E_{S_i}(\gamma_{j_1}, \gamma_{j_2})}} \bar{\gamma}_{j_1} = \bar{\gamma}_{j_2} \quad \wedge \bigwedge_{\substack{1 \le j_1, j_2 \le m \\ \neg E_{S_i}(\gamma_{j_1}, \gamma_{j_2})}} \bar{\gamma}_{j_1} \neq \bar{\gamma}_{j_2}$$

asserts that $E_S = E_{S_i}$ holds. Now define the rule r by

```
par  if $\varphi_1$ then $r_{S_1}$ endif
     if $\varphi_2$ then $r_{S_2}$ endif
          ⋮
     if $\varphi_n$ then $r_{S_n}$ endif endpar
```

If $S \in \mathcal{S}_t$ is any state of t, then S is T-equivalent to exactly one S_i $(1 \le i \le n)$, which implies $val_S(\varphi_j) = true$ iff $j = i$, and hence $\Delta(r, S) = \Delta(r_{S_i}, S) = \Delta(t, S)$ by Lemma 8. □

Theorem 7. *For every database transformation t there exists an equivalent DB-ASM $\mathcal{M}$.*

Proof. By Lemma 9 there is a DB-ASM rule r with $\Delta(r, S) = \Delta(t, S)$ for all $S \in \mathcal{S}_t$. Define $\mathcal{M}$ with the same signature and background as t (and hence $\mathcal{S}_{\mathcal{M}} = \mathcal{S}_t$), $\mathcal{I}_{\mathcal{M}} = \mathcal{I}_t$, $\mathcal{F}_{\mathcal{M}} = \mathcal{F}_t$, and program $\pi_{\mathcal{M}} = r$. □

Note that for the proof of Theorem 7 we constructed a DB-ASM rule that does not use sequence rules, so by Theorem 6 this construction can be considered to be merely "syntactic sugar". As discussed before the let rules capture aggregate updates that exploit parallelism. Whether this can be extended to capture various aspects of parallelism, thus looking deeper inside database transformations, is an open problem.

6 Discussion and Conclusions

In this article we presented a variant of Gurevich's sequential ASM-thesis [10] dealing with database transformations in general. In analogy to Gurevich's seminal work we formulated five intuitive postulates for database transformations, and discussed why database transformations should satisfy these postulates. We then defined a variant of Abstract State Machines, which we called Database Abstract State Machines (DB-ASMs), and showed that DB-ASMs capture exactly all database transformations.

Despite many little technical differences of minor importance – such as final states, finite runs, and undefined successor state in case of an inconsistent update set – we stayed rather close to Gurevich's seminal work, but added as much as we felt is necessary to capture the essentials of database transformations as opposed to sequential algorithms. The important differences to Gurevich's sequential time postulate are the permission of non-determinism in a limited form with limitations enforced by the bounded non-determinism postulate, the exploitation of meta-finite states to capture the finiteness of databases, and an extended bounded exploration postulates, in which non-ground terms can appear in the bounded exploration witness.

Let us first summarise and discuss these important differences again. Regarding states we stayed with Gurevich's fundamental idea that states are first-order structures. We only adopted the notion of meta-finiteness [9]. As for the sequential ASM-thesis closure under isomorphisms is requested. So, apart from the incorporation of meta-finite states that are composed of a finite database part and a usually infinite algorithmic part with bridge functions linking them we more or less kept Gurevich's abstract state postulate. The idea of using meta-finite states was already expressed in our previous work in [22], but we were not yet able to handle the bridge functions in a satisfactory way. This gap is now closed.

Though for database transformations the projection of a run to the database part of states is most decisive, we did not build such a restriction into our model. This in turn implies that all sequential algorithms are also captured by DB-ASMs. Parallel algorithms, however, are not captured, as we only permit bounded parallelism as in the sequential ASM thesis. Combining our insights on database transformations with the parallel ASM thesis, i.e. permitting unrestricted parallelism on the algorithmic part of states, is a problem left for continued research.

Due to the permission of non-determinism that is restricted to choice among query results we capture more than sequential algorithms. Even without the non-determinism this is still the case because of the more general bounded exploration witnesses and the exploitation of location operators, which permit a limited form of parallelism by aggregating multisets of values.

In order to capture different data models, we originally thought of manipulating the notion of state, e.g. in our first attempt in [21] we tried to employ higher-order structures to capture tree-structured databases such as object-oriented and XML-based databases. In this article, we formulated that different data models should be captured by means of different background structures, which led us to formulate a background postulate. We are currently investigating this approach in more detail showing which particular constructors have to be present to capture data models such as the relational, nested-relational, complex values, object-oriented and XML models. The results achieved so far show that shifting specific data model requirements to background structures does indeed do the trick; we hope to be able to publish results shortly.

Nevertheless, we believe it is a promising idea for future research to consider "playing" with the notion of states, maybe even not only within the context of databases. For instance, for tree-based databases we may think of structures that can be recognised by certain tree-automata, or we could investigate automatic structures that are recognised by finite automata, etc. While these define restrictions to the general computational model, they may on one side provide interesting links to various logics, and on the other side define the challenge to integrate these automata into the formalism of DB-ASMs in order to capture exactly a particular class of database transformations.

The logical links would be of particular interest for queries, for which declarative approaches are preferred. In this paper we only touched the surface of queries in our extension of Gurevich's bounded exploration postulate. Instead of requesting the existence of a finite set of ground terms that determines update sets (or sets of update sets due to the assumed non-determinism) we widen this to a set of access terms, which in a sense capture associative access that is considered to be essential for databases. Nevertheless, there is still a big gap between access terms and similarly the access conditions in choice and forall rules in DB-ASMs on one side and highly declarative query languages. Bringing these different aspects together is a challenge for future research.

Finally, the notion of database transformation developed in this paper is limited to sequential database transformations over centralised databases, i.e. aspects of parallel and distributed databases have been neglected. With respect to parallelism we only considered aggregate update by means of location operators, i.e. we accumulate a multiset of updates on a location and then let them collapse to a single update. This form of parallelism is limited to the same computation on different data. Capturing parallelism and distribution as used in the architecture in [13] would require to investigate a more elaborate DB-ASM thesis picking up ideas from the parallel ASM thesis [5].

References

[1] Abiteboul, Serge and Kanellakis, Paris C. Object identity as a query language primitive. *Journal of the ACM*, 45(5):798–842, 1998.

[2] Abiteboul, Serge and Vianu, Victor. Datalog extensions for database queries and updates. *Journal of Computer and Systems Science*, 43(1):62–124, 1991.

[3] Beeri, Catriel, Milo, Tova, and Ta-Shma, Paula. On genericity and parametricity (extended abstract). In *PODS '96: Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 104–116, New York, NY, USA, 1996. ACM.

[4] Beeri, Catriel and Thalheim, Bernhard. Identification as a primitive of data models. In Polle, Torsten, Ripke, Torsten, and Schewe, Klaus-Dieter, editors, *Fundamentals of Information Systems*, pages 19–36. Kluwer Academic Publishers, Boston Dordrecht London, 1999.

[5] Blass, Andreas and Gurevich, Jury. Abstract state machines capture parallel algorithms. *ACM Transactions on Computational Logic*, 4(4):578–651, 2003.

[6] Börger, Egon and Stärk, Robert. *Abstract State Machines: A Method for High-Level System Design and Analysis.* Springer-Verlag, Berlin Heidelberg New York, 2003.

[7] Cohen, Sara. User-defined aggregate functions: bridging theory and practice. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on the Management of Data*, pages 49–60, New York, NY, USA, 2006. ACM Press.

[8] Ebbinghaus, H.-D. and Flum, J. *Finite Model Theory.* Springer-Verlag, 2 edition, 1999.

[9] Grädel, Erich and Gurevich, Yuri. Metafinite model theory. *Information and Computation*, 140(1), 1998.

[10] Gurevich, Jury. Sequential abstract state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, 2000.

[11] Gurevich, Yuri and Tillmann, Nikolai. Partial updates. *Theoretical Computer Science*, 336(2-3):311–342, 2005.

[12] Gurevich, Yuri and Yavorskaya, Tanya. On bounded exploration and bounded nondeterminism. Technical Report MSR-TR-2006-07, Microsoft Research, January 2006.

[13] Kirchberg, M., Schewe, K.-D., Tretiakov, A., and Wang, R. A multi-level architecture for distributed object bases. *Data and Knowledge Engineering*, 60(1):150–184, 2007.

[14] Ma, Hui, Schewe, Klaus-Dieter, Thalheim, Bernhard, and Wang, Qing. A theory of data-intensive software services. *Service Oriented Computing and Applications*, 3(4):263–283, 2009.

[15] Schewe, Klaus-Dieter and Thalheim, Bernhard. Fundamental concepts of object oriented databases. *Acta Cybernetica*, 11(4):49–84, 1993.

[16] Schewe, Klaus-Dieter and Wang, Qing. XML database transformations. Journal of Universal Computer Science (to appear).

[17] Van den Bussche, J. *Formal Aspects of Object Identity in Database Manipulation.* PhD thesis, University of Antwerp, 1993.

[18] Van den Bussche, Jan and Van Gucht, Dirk. Semi-determinism (extended abstract). In *PODS '92: Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 191–201, New York, NY, USA, 1992. ACM Press.

[19] Van den Bussche, Jan and Van Gucht, Dirk. Non-deterministic aspects of object-creating database transformations. In *Selected Papers from the Fourth International Workshop on Foundations of Models and Languages for Data and Objects*, pages 3–16, London, UK, 1993. Springer-Verlag.

[20] Van Den Bussche, Jan, Van Gucht, Dirk, Andries, Marc, and Gyssens, Marc. On the completeness of object-creating database transformation languages. *J. ACM*, 44(2):272–319, 1997.

[21] Wang, Qing and Schewe, Klaus-Dieter. Axiomatization of database transformations. In *Proceedings of the 14th International ASM Workshop (ASM 2007)*, University of Agder, Norway, 2007.

[22] Wang, Qing and Schewe, Klaus-Dieter. Towards a logic for abstract metafinite state machines. In Hartmann, Sven and Kern-Isberner, Gabriele, editors, *Foundations of Information and Knowledge Systems – 5th International Symposium (FoIKS 2008)*, volume 4932 of *Lecture Notes in Computer Science*, pages 365–380. Springer-Verlag, 2008.

Received 11th September 2009