

A Complete Logic for Database Abstract State Machines

Qing Wang^a, Flavio Ferrarotti^b, Klaus-Dieter Schewe^b, Loredana Tec^b

^a*Research School of Computer Science, The Australian National University, Australia,
qing.wang@anu.edu.au*

^b*Software Competence Center Hagenberg, Austria,
[flavio.ferrarotti|loredana.tec]@scch.at,kdschewe@acm.org*

Abstract

In database theory, the term *database transformation* was used to refer to a unifying treatment for computable queries and updates. Recently, it was shown that non-deterministic database transformations can be captured exactly by a variant of ASMs, the so-called Database Abstract State Machines (DB-ASMs). In this article we present a logic for DB-ASMs, extending the logic of Nanchen and Stärk for ASMs. In particular, we develop a rigorous proof system for the logic for DB-ASMs, which is proven to be sound and complete. The most difficult challenge to be handled by the extension is a proper formalisation capturing non-determinism of database transformations and all its related features such as consistency, update sets or multisets associated with DB-ASM rules. As the database part of a state of database transformations is a finite structure and DB-ASMs are restricted by allowing quantifiers only over the database part of a state, we resolve this problem by taking update sets explicitly into the logic, i.e. by using an additional modal operator $[X]$, where X is interpreted as an update set Δ generated by a DB-ASM rule. The DB-ASM logic provides a powerful verification tool to study properties of database transformations.

Acknowledgements. The research reported in this paper results from the project *Behavioural Theory and Logics for Distributed Adaptive Systems* supported by the **Austrian Science Fund (FWF): [P26452-N15]**. It was further supported by the Austrian Research Promotion Agency (FFG) through the COMET funding for the Software Competence Center Hagenberg.

1. Introduction

Queries and updates are two basic types of computations in databases, which capture the capability to retrieve and update data, respectively. In database theory, database transformations refer to transforming database instances into other database instances, including both queries and updates. More formally, a database transformation was defined as a binary relation on database instances over an input schema and database instances

over an output schema which must satisfy the four criteria: well-typedness, effective computability, genericity and functionality [1, 3].

In the past 50 years, a main research topic of database transformations was to characterise different subclasses of database transformations in terms of their logical or algebraic properties. Although prior studies have yielded fruitful results for queries, e.g., computable queries [12], determinate transformations [1], semi-deterministic transformations [44], constructive transformations [42, 45], etc., extending these results to updates is by no means straightforward. In [43] Van den Bussche and Van Gucht conjectured that queries and updates may have some fundamental distinctions, and raised the question of whether there exists a theoretical framework that can unify both queries and updates.

In many real-life database applications, database queries and updates often turn out to have intimate connections. For instance, a relation may be updated by using matched tuples from another (possibly the same) relation on a join operation, tuples of a relation may be deleted based on some selection criteria, etc. On one side, such connections between queries and updates justify the fact that the embedding of queries in updates is supported as a fundamental feature in all major commercial relational database systems. On the other side, it brings up considerable concerns on the theoretical foundations for database updates. In a sharp contrast to the elegant and fruitful theory for database queries, the theoretical foundations for database updates, or more generally, for a unifying framework encompassing both queries and updates are still lacking.

Recently, the rising trend of NoSQL database applications has further increased the importance of studying database transformations as a unifying framework that encompasses both queries and updates. This is because these application areas are often complicated and their data is schema-less, heterogeneous, redundant, inconsistent and frequently modified. As a result, the query mechanisms used in NoSQL database applications are more similar to traditional programming using programming languages, such as C or Java, rather than traditional queries using SQL. The distinction between database queries and updates has been considerably lessened. Thus, in order to rigorously manage and use data in these database applications, a theoretical framework for database transformations is required.

Nevertheless, formalising a unifying framework for database transformations is challenging. As reported in [43], the non-determinism of database transformations is a difficult problem. Particularly, in the presence of unique identifiers, the representation of unique identifiers is irrelevant and only the interrelationship between objects represented by them matters [1]. Indeed, the degree of non-determinism is one of critical factors which determine the upper bound of the expressiveness of associated languages. Apart from the issue of non-determinism, it has also been perceived that there is a mismatch between the declarative semantics of query languages and the operational semantics of update languages. Unlike many query languages which can describe what a program should accomplish rather than specify how to accomplish, update languages usually require explicit specifications of operations and control flow. An obvious question is “how can different semantics be integrated within one database language for database transformations?”.

We address these challenges by developing a theoretical framework for database transformations using Abstract State Machines and studying the logical foundations of database transformations in such a theoretical framework. Abstract State Machine (ASM) is a universal model of computation introduced by Gurevich in his well-known attempts to formalise different notions of algorithms [21, 22]. Gurevich’s sequential ASM thesis has shown that sequential ASMs can exactly capture all the sequential algorithms that are stipulated by three postulates [21]. As ASMs are essentially state machines operating on states that are first-order structures, dynamics between states can be rigorously captured by the concepts of updates, update sets or update multisets, as widely acknowledged in the ASM research community [9, 22]. The sequential ASM thesis also sheds light into the way of establishing a unifying theoretical framework for database transformations that contain queries and updates. Intuitively, this is based on the observation that the class of computations described by database transformations can be formalised as a class of ASMs respecting database principles. This has led to the development of *Database Abstract State Machines* (DB-ASMs), a variant of ASMs, as a model of computation for database transformations, and the DB-ASM thesis which has proven that DB-ASMs satisfy the five postulates stipulated for database transformations, and all computations stipulated by the postulates for database transformations can also be simulated step-by-step by a *behaviourally equivalent* DB-ASM [33]. This in a sense establishes the database analogue of Gurevich’s sequential ASM thesis [21].

Contributions In this paper, we study the logical foundations of the DB-ASM thesis. Our contributions are as follows.

Firstly, we characterise states of DB-ASMs by the logic of meta-finite structures [16] which is then incorporated into a logic for DB-ASMs. In database theory, states of a database transformation are predominantly regarded as finite structures. However, when applying algorithmic operations to tackle database-related problems, the finiteness condition on states often turns out to be too restrictive for several reasons: (1) database transformations may deal with new elements from countably infinite domains, e.g. counting queries produce natural numbers even if no natural numbers occur in a finite structure; (2) finite structures may have invariant properties that possibly have infinite elements implied in satisfying them, such as, numerical invariants of geometric objects or database constraints; (3) each database transformation either implicitly or explicitly lives in a background that supplies all necessary information relating to computations and usually exists in the form of infinite structures. Thus, we consider a state of database transformation as a meta-finite structure consisting of (i) a database part, which is a finite structure, (ii) an algorithmic part, which may be an infinite structure, and (iii) a finite number of bridge functions between these two parts. Characterising states of database transformations using the logic of meta-finite structures enables us to reason about aggregate computations commonly existing in database applications.

Our second contribution is the handling of bounded non-determinism in the logic of DB-ASMs. This was also the most challenging problem we faced in this work. It is worth

to mention that non-deterministic transitions manifest themselves as a very difficult task in the logical formalisation for ASMs. Nanchen and Stärk analysed potential problems to several approaches they tried by taking non-determinism into consideration and concluded [39]:

“Unfortunately, the formalisation of consistency cannot be applied directly to non-deterministic ASMs. The formula $\text{Con}(R)$ (as defined in Sect. 8.1.2 of [9]) expresses the property that the *union of all possible* update sets of R in a given state is consistent. This is clearly not what is meant by consistency. Therefore, in a logic for ASMs with **choose** one had to add $\text{Con}(R)$ as an atomic formula to the logic.”

However, we observe that this conclusion is not necessarily true, as finite update sets can be made explicit in the formulae of a logic to capture non-deterministic transitions. In doing so, the formalisation of consistency defined in [39] can still be applied to such an explicitly specified finite update set U yielded by a rule r in the form of the formula $\text{con}(r, X)$ where the second-order variable X is interpreted by U , as will be discussed in Section 7.1. We can thus solve this problem by adding the modal operator $[X]$ for an update set generated by a DB-ASM rule. In doing so, DB-ASMs are restricted to have quantifiers only over the database part of a state which is a finite structure, and consequently update sets (or multisets) yielded by DB-ASM rules are restricted to be finite. Hence, the logic for DB-ASMs is empowered to capture non-deterministic database transformations.

Our third contribution is the development of a proof system for the logic for DB-ASMs, which extends the proof system for the logic for ASMs [39] in several aspects:

- DB-ASMs can collect updates yielded in parallel computations under the multiset semantics, i.e. update multisets, then aggregate updates in an update multiset to an update set by applying so-called *location operators*. Our proof system can capture this by incorporating the axioms for both the predicate of update multisets and the predicate of update sets. The axioms also specify the interaction between update multisets and update sets in relating to DB-ASM rules.
- A DB-ASM rule may be associated with different update sets. Applying different update sets may lead to different successor states to the current state. As the logic for DB-ASMs includes formulae denoting explicit update sets and update multisets, and second-order variables that are bound to update sets or update multisets, our proof system allows us to reason about the interpretation of a formula over all successor states or over some successor state after applying a DB-ASM rule over the current state.
- In addition to capturing the consistency of an update set yielded by a DB-ASM rule, our proof system also develops two notions of consistency for a DB-ASM rule (i.e.

weak version and strong version). When a DB-ASM rule is deterministic, these two notions coincide.

Our last contribution is a proof of the completeness of the logic for DB-ASMs. Due to the importance of non-determinism for enhancing the expressive power of database transformations and for specifying database transformations at flexible levels of abstraction, DB-ASMs take into account choice rules. Consequently, the logic for DB-ASMs has to handle all the issues related to non-determinism which have been identified as the source of problems in the completeness proof of the logic for ASMs [39, 10]. Nevertheless, we prove that we can use a Henkin semantics for the required (in our approach) second-order quantification, and thus despite of the inclusion of second-order formulae in the logic for DB-ASMs, we can establish a sound and complete proof system for the logic of DB-ASMs. Note that, the logic for DB-ASMs preserves the restriction of the logic of Nanchen and Stärk for ASMs [39] dealing only with properties of a *single step* of an ASM, *not* with properties of whole ASM runs. This restriction to a one-step logic allows us to define a Hilbert-style proof theory and to show its completeness, whereas for a logic dealing with properties of whole ASM runs (and even more so, whole DB-ASMs runs) can hardly be expected to be complete.

Outline The remainder of the article is structured as follows. Section 2 discusses related work on logical characterisations of database transformations. Then we provide a motivating example in Section 3. In Section 4 we discuss meta-finite structures and states of DB-ASMs. In Section 5 we present the definitions of DB-ASM. As states of a database transformation are meta-finite structures, in Section 6 we define a logic for DB-ASMs that is built upon the logic of meta-finite structures. Subsequently, a detailed discussion of basic properties of the logic for DB-ASMs, such as consistency, update sets and multisets, along with the formalisation of a proof system is presented in Section 7. In Section 8, we present some interesting properties of the logic for DB-ASMs which are implied by the axioms and rules of the proof system introduced in Section 7. We prove in Section 9 that the logic for DB-ASMs is complete. We conclude the article with a brief summary in Section 10.

2. Related Work

It is widely acknowledged that a logic-based perspective for database queries can provide a yardstick for measuring the expressiveness and complexity of query languages. To extend the application of mathematical logics from database queries to database updates, a number of logical formalisms have been developed providing the reasoning for both states and state changes in a computation model [8, 46]. A popular approach was to take dynamic logic as a starting point and then to define the declarative semantics of logical formulae based on Kripke structures. It led to the development of the *database dynamic logic* (DDL) and *propositional database dynamic logic* (PDDL) [36, 35, 37]. DDL has atomic updates for inserting, deleting and updating tuples in predicates and for functions,

whereas PDDL has two kinds of atomic updates: passive and active updates. Passive updates change the truth value of an atom while active updates compute derived updates using a logic program. In [38] Spruit, Wieringa and Meijer proposed *regular first-order update logic* (FUL), which generalises dynamic logic towards specification of database updates. A state of FUL is viewed as a set of non-modal formulae. Unlike standard dynamic logic, predicate and function symbols rather than variables are updatable in FUL. There are two instantiations of FUL. One is called *relational algebra update logic* (RAUL) that is an extension of relational algebra with assignments as atomic updates. Another one is DDL that parameterizes FUL by two kinds of atomic updates: bulk updates to predicates and assignment updates to functions. It was shown that DDL is also “update complete” in relational databases with respect to the update completeness criterion proposed by Abiteboul and Vianu in [2].

As we explained before ASMs turn out to be a promising approach for specifying database transformations. The logical foundations for ASMs have been well studied from several perspectives. Groenboom and Renardel de Lavalette presented in [18] a logic called *modal logic of creation and modification* (MLCM) that is a multimodal predicate logic intended to capture the ideas behind ASMs. On the basis of MLCM they developed a language called *formal language for evolving algebras* (FLEA) [19]. Instead of values of variables, states of an MLCM are represented by mathematical structures expressed in terms of dynamic functions. The work in [31] generalises MLCM and other variations from [13] to *modification and creation logic* (MCL) for which there exists a sound and complete axiomatisation. In [34] Schönegge presented an extension of dynamic logic with update of functions, extension of universes and simultaneous execution (called EDL), which allows statements about ASMs to be directly represented. In addition to these, a logic complete for *hierarchical ASMs* (i.e., ASMs that do not contain recursive rule definitions) was developed by Nanchen and Stärk in [39]. This logic for ASMs differs from other logics in two respects: (1) the consistency of updates has been accounted for; (2) modal operators are allowed to be eliminated in certain cases. As already remarked, the ASM logic of Nanchen and Stärk permits reasoning about ASM rules, but not about ASM runs, which is the price to be paid for obtaining completeness. In this article we will extend this logic for ASMs towards database transformations, in which states are regarded as meta-finite structures and a bounded form of non-determinism is captured.

It was Chandra and Harel who first observed limitations of finite structures in database theory [12]. They proposed a notion of an *extended database* that extends finite structures by adding another countable, enumerable domain containing interpreted features such as numbers, strings and so forth. The intention of their study was to provide a more general framework that can capture queries with interpreted elements. Another extension of finite structures was driven by the efforts to solve the problem of expressing cardinality properties [7, 11, 17, 26, 29, 30, 40, 41]. For example, Grädel and Otto developed a two-sorted structure that adjoins a one-sorted finite structure with an additional finite numerical domain and added the terms expressing cardinality properties [17]. They aimed at studying the expressive power of logical languages that involve induction with counting

on such structures. A promising line of work is *meta-finite model theory*. Grädel and Gurevich in [16] defined *meta-finite structures*. Based on the work presented in [16], Hella et al. in [23] studied the logical grounds of query languages with aggregation, which is closely related to our work presented in this article. However, the logic for DB-ASMs covers not only database queries with aggregation but also database updates. Put it in another way, it is a logical characterisation for database transformations including aggregate computing and sequential algorithms.

3. Motivating Example

To motivate our work, we use an example to illustrate how database transformations can be captured by DB-ASMs and how a logic for DB-ASMs can be used for verifying database transformations, i.e., one of the potential applications of the logic for DB-ASMs.

Example 3.1. Consider a relational database schema: CITY={Cid, Name} and ROUTE={FromCid, ToCid, Distance}, which store route information of cities and their distance. Assume that we have $\forall c_1, c_2, d((c_1, c_2, d) \in \text{ROUTE} \rightarrow (c_2, c_1, d) \in \text{ROUTE})$. Then a relation of ROUTE corresponds to an undirected graph in which the nodes represent cities and the edges represent direct routes, for example, the undirected graph in Fig. 2 corresponds to the relation of ROUTE in Fig. 1. Assume that such graphs are always connected. Let $Q_1(c)$ be the query “find a shortest path tree rooted at city c ”. To answer this query, we would need to find a spanning tree T with the root node c such that the path distance from c to any other node c' in T is the shortest path distance from city c to c' in the graph induced by ROUTE.

CITY		ROUTE		
Cid	Name	FromCid	ToCid	Distance
c_1	A	c_1	c_2	d_1
c_2	B	c_1	c_3	d_3
c_3	C	c_2	c_4	d_2
c_4	D	c_3	c_4	d_1
c_5	E	c_3	c_5	d_2
		c_5	c_4	d_4
	

VISITED	RESULT	
Cid	TotalCost	LastStop

Figure 1: An initial state

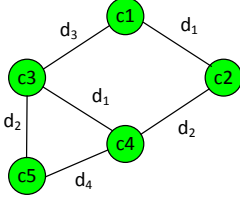


Figure 2: An undirected graph

$\text{DIST} : \text{Cid} \rightarrow \mathbb{N}$	$\text{VAL} : \text{Distance} \rightarrow \mathbb{N}$
$\text{DIST}(c_1) = 1$	$\text{VAL}(d_1) = 500.50$
$\text{DIST}(c_2) = 2$	$\text{VAL}(d_2) = 100.00$
$\text{DIST}(c_3) = 3$	$\text{VAL}(d_3) = 808.20$
$\text{DIST}(c_4) = 4$	$\text{VAL}(d_4) = 203.20$
$\text{DIST}(c_5) = 5$	

Figure 3: Two bridge functions

The DB-ASM rule in Fig. 5 (of the signature Υ_G described next), which corresponds to the famous Dijkstra's algorithm, expresses the query $Q_1(c)$. Let Υ_G be the signature in Fig. 4. Apart from CITY and ROUTE, Υ_G includes VISITED= $\{\text{Cid}\}$ to store the cities that have been visited during the computation and RESULT= $\{\text{ChildCid}, \text{ParentCid}\}$ to store the shortest path tree as a child-parent node relationship. We assume that in every initial state the relations VISITED and RESULT are empty (as shown in Fig. 1) and that INITIAL=TRUE. We also assume that the constant symbol INFINITY is interpreted by a natural number which is strictly greater than the sum of all the distances in ROUTE, that ZERO is interpreted by the value 0, and that the values interpreting the constant symbols TRUE and FALSE are different. A state in which every city has been visited, i.e., a state in which VISITED contains every city id in the database, is considered as a final state. Notice that entries in the database part of the states which correspond to the distances between adjacent nodes in ROUTE are surrogates for the actual distances which are natural numbers in the algorithmic part of the state. Thus, there are two bridge functions:

- $\text{DIST} : \text{Cid} \rightarrow \mathbb{N}$ to keep track of cities visited during a computation and their corresponding shortest distances to c , respectively;
- $\text{VAL} : \text{Distance} \rightarrow \mathbb{N}$ to map the surrogates for the actual distances to the natural numbers in the algorithmic part.

$\Upsilon_G = (\Upsilon_{db}, \Upsilon_a, \mathcal{F}_b)$, where - CITY, ROUTE, VISITED, RESULT, c, TRUE, FALSE, INITIAL $\in \Upsilon_{db}$; - INFINITY, ZERO, MDIST $\in \Upsilon_a$; - VAL, DIST $\in \mathcal{F}_b$; and - MIN is a location operator.
--

Figure 4: A signature

In general, the DB-ASM in Fig. 5 proceeds in two stages:

- The first stage is described by Lines 2-12. The DB-ASM starts with an initial state in which `INITIAL = TRUE`, then assigns (in parallel) to every city a tentative distance value (i.e., 0 for the city c and `INFINITY` for all other cities), and ends with `INITIAL=FALSE`.
- The second stage is described by Lines 13-36. The shortest paths to reach other cities from the city c are repeatedly calculated and stored in `RESULT` until a final state in which every city has been visited is reached.

At Line 15 of the DB-ASM rule in Fig. 5 the location operator `MIN` is assigned to the location `(MDIST, ())`, and thus `MDIST` is updated to the shortest distance among the collection of distances between the city in consideration and all its unvisited neighbor cities. At Line 20, we can see that the DB-ASM is non-deterministic because a city is arbitrarily chosen from the non-visited cities whose shortest paths are equally minimum at each step of the computation process. This indeed exemplifies the importance of non-determinism for specifying database transformations at a high-level of abstraction.

Now suppose that we want to know whether the properties P1 and P2 described next, are satisfied by the DB-ASM corresponding to the DB-ASM rule in Fig. 5 over certain states of signature Υ_G . Clearly, the use of a logic to specify such properties of DB-ASMs can contribute significantly to the verification of the correctness of database transformations expressed by means of DB-ASMs. Although the logic proposed for DB-ASMs in this paper can only reason about such properties within one-step of computation, it nevertheless provides a useful tool which is a first step towards developing a logic that can reason about properties of whole DB-ASMs runs.

- (P1) In every non-initial state of a run, each city in the child/parent node relationship encoded in `RESULT` has exactly one parent city, except for c which has none. In other words, `RESULT` encodes a tree with root node c .

$$\neg\text{INITIAL} \rightarrow \forall xy(\text{RESULT}(x, y) \rightarrow x \neq c \wedge \forall z(z \neq y \rightarrow \neg\text{RESULT}(x, z))) \wedge \exists x(\text{RESULT}(x, c))$$

- (P2) In every state of a run, if a city not yet visited (by the algorithm) is a neighbour city of one already visited, then the calculated (shortest so far) distance from c to that city is strictly less than `INFINITY` already.

$$\forall xy(\text{VISITED}(x) \wedge \neg\text{VISITED}(y) \wedge \exists z(\text{ROUTE}(x, y, z)) \rightarrow \text{DIST}(y) < \text{INFINITY})$$

4. Meta-finite Structures as States

Meta-finite structures were originally studied by Grädel and Gurevich in order to extend the methods of finite model theory beyond finite structures [16]. In a nutshell, a meta-finite structure consists of (a) a primary part, which is a finite structure, (b) a

```

1  par
2    if INITIAL then
3      par
4        forall x with  $\exists y(\text{CITY}(x, y))$  do
5          par
6            if  $x = c$  then  $\text{DIST}(x) := \text{ZERO}$  endif
7            if  $x \neq c$  then  $\text{DIST}(x) := \text{INFINITY}$  endif
8          endpar
9        enddo
10       INITIAL := FALSE
11     endpar
12   endif
13   if  $\neg \text{INITIAL}$  then
14     seq
15       let  $(\text{MDIST}, ()) \rightarrow \text{MIN}$  in
16         forall x with  $\exists y(\text{CITY}(x, y) \wedge \neg \text{VISITED}(x))$  do
17           MDIST :=  $\text{DIST}(x)$ 
18         enddo
19       endlet
20       choose x with  $\text{DIST}(x) = \text{MDIST} \wedge \neg \text{VISITED}(x)$  do
21         par
22           VISITED(x) := TRUE
23           forall y, z with  $\text{ROUTE}(x, y, z) \wedge \neg \text{VISITED}(y) \wedge$ 
24              $\text{MDIST} + \text{VAL}(z) < \text{DIST}(y)$  do
25             par
26                $\text{DIST}(y) := \text{MDIST} + \text{VAL}(z)$ 
27                $\text{RESULT}(y, x) := \text{TRUE}$ 
28               forall x' with  $x' \neq x \wedge \text{RESULT}(y, x')$  do
29                  $\text{RESULT}(y, x') := \text{FALSE}$ 
30               enddo
31             endpar
32           enddo
33         endpar
34       enddo
35     endseq
36   endif
37 endpar

```

Figure 5: A DB-ASM

secondary part, which is a (usually infinite) structure, and (c) a set of functions mapping from the primary part into the second part. Typical examples of meta-finite structures are finite objects arising in many areas of computer science, which usually consist of both structures and numbers. For example, graphs with weights on the edges, where a graph may be representable by a finite structure but its weights on the edges may be reals from an infinite domain, and arithmetical operations performed on these weights may not be

any a *priori* fixed finite subdomain [16]. Another example is relational databases in which each relation contains only a finite number of tuples. Although theoretically a relational database is viewed as a finite structure, attribute domains of a relation are often assumed to be countably infinite. In particular, such domains may be infinite mathematical structures, e.g., the natural numbers with arithmetic, rather than merely plain sets, and such infinite mathematical structures are widely used by aggregate queries in many real-life database applications.

In [20] Gurevich argued that ASMs provide a model of computation that is more powerful and more universal than other standard models of computation such as Turing machines, in the sense that any algorithm, however abstract, can be simulated step-for-step by an ASM. This is because a state of an ASM is abstract, which may include any real world objects and functions at a chosen level of abstraction. Let us consider for example algorithms that work with graphs. The conventional computation models require a string representation of the given graph or similar, even in those cases when the algorithm is independent of the graph representation. In particular, a same database might have different representations, but the meaning of the data should not change. Database query languages are supposed to reflect only representation-independent properties.

In ASMs, states are viewed as first-order structures, whereas in DB-ASMs we consider states as meta-finite structures [33]. Conceptually, each state of a DB-ASM has a finite *database part* and a possibly infinite *algorithmic part*, which are linked via *bridge functions* such that actual database entries are treated merely as surrogates for the real values. This permits a database to remain finite while allowing database entries to be interpreted in possibly infinite domains such as the natural numbers with arithmetic operations. A signature Υ of states comprises (i) a sub-signature Υ_{db} for the database part, (ii) a sub-signature Υ_a for the algorithmic part and (iii) a finite set \mathcal{F}_b of bridge function names. The *base set* of a state S is a nonempty set of values $B = B_{db} \cup B_a$, where B_{db} is finite, and B_a contains natural numbers, i.e., $\mathbb{N} \subseteq B_a$. Function symbols f in Υ_{db} and Υ_a , respectively, are interpreted as functions f^S over B_{db} and B_a , and the interpretation of a k -ary function symbol $f \in \mathcal{F}_b$ defines a function f^S from B_{db}^k to B_a . For every state over Υ , the restriction to Υ_{db} results in a finite structure.

Since the states of DB-ASMs are defined as meta-finite structures, we now need to define a matching logic so that it can be used in the conditional statements of DB-ASMs. That is, we need a logic of meta-finite structures as introduced in [16]. Logics of meta-finite structures distinguish among two types of terms. The first type, which we call *database terms*, denote elements of the primary (finite) part of the meta-finite structure. The second type, which we call *algorithmic terms*, denote elements of the secondary (possibly infinite) part of the meta-finite structure.

Definition 4.1. Let $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$ be a signature of meta-finite states. Fix a countable set \mathcal{X}_{db} of first-order variables, denoted with standard lowercase letters x, y, z, \dots , that range over the primary database part of the meta-finite states (i.e., the finite set B_{db}). The set of *database terms* \mathcal{T}_{db} is defined as the closure of the set \mathcal{X}_{db} of variables under

the application of function symbols in Υ_{db} . We assume that Υ_{db} always include a function symbol for equality. In turn, the set of *algorithmic terms* \mathcal{T}_a is defined inductively as follows:

- If t_1, \dots, t_n are database terms in \mathcal{T}_{db} and f is an n -ary bridge function symbol in \mathcal{F}_b , then $f(t_1, \dots, t_n)$ is an algorithmic term in \mathcal{T}_a .
- If t_1, \dots, t_n are algorithmic terms in \mathcal{T}_a and f is an n -ary function symbol in Υ_a , then $f(t_1, \dots, t_n)$ is an algorithmic term in \mathcal{T}_a .
- Nothing else is an algorithmic term in \mathcal{T}_a .

We set $\mathcal{T}_{\Upsilon, \mathcal{X}_{db}} = \mathcal{T}_{db} \cup \mathcal{T}_a$.

In this context, a *variable assignment* (or *valuation*) ζ is a function which assigns to every variable in \mathcal{X}_{db} a value in the base set of the database part B_{db} of the meta-finite state S . The value of a term $t \in \mathcal{T}_{\Upsilon, \mathcal{X}_{db}}$ in a state S under a valuation ζ , denoted $val_{S, \zeta}(t)$, is defined as usual in first-order logic, i.e., using the classical Tarski's semantics.

The *logic of meta-finite states* \mathcal{L}^{FO} which we use in the formalization DB-ASMs is defined as the first-order logic with equality which is built up from equations between terms in $\mathcal{T}_{\Upsilon, \mathcal{X}_{db}}$ by using the standard connectives and first-order quantifiers. Its semantics is defined in the standard way. The truth value of a formula of meta-finite states φ in S under the valuation ζ is denoted as $\llbracket \varphi \rrbracket_{S, \zeta}$.

5. Database Abstract State Machines

Our work in this paper concerns the model of *Database Abstract State Machine* (DB-ASM) that captures the class of database transformations defined by the postulates in the DB-ASM thesis [33, 47]. Accordingly, we assume that states of DB-ASMs are meta-finite structures which include a minimum background of computation as required by the background postulate in the axiomatization of database transformations in [33, 47] (note that this is essentially the same background that is required in the parallel ASM thesis [5, 6, 15]). That is, every state of a DB-ASM includes:

- An infinite reserve of values not used in a current state, but available to be added to the active domain in any state transition.
- Boolean values (*true* and *false*), Boolean operations (\neg , \wedge , \vee and \rightarrow), and the undefinedness value (*undef*).
- A pairing constructor and a multiset constructor together with necessary operators on tuples and multisets.

The bounded exploration postulate in the DB-ASM thesis [33, 47], as well as the bounded exploration postulates in Gurevich’s sequential ASM thesis [21] and in the new parallel ASM thesis [15] (which simplifies the parallel ASM thesis of Blass and Gurevich [5, 6]), are motivated by the *accessibility principle*, which can be defined as the prerequisite that each location of a state must be uniquely identifiable. In fact, unique identifiability also applies to databases as emphasised by Beeri and Thalheim in [4], and has to be claimed for the basic updatable units in a database, for example, objects in [32]. The accessibility principle is also a fundamental assumption used in the characterization proofs of the DB-ASM thesis as well as of the sequential and parallel ASM thesis. We therefore assume that it holds for every state of the DB-ASMs.

As explained in [21], an algorithm A can access an element a of a state by using formulae φ and $\psi(x)$ such that φ is a sentence and x is the only free variable in $\psi(x)$, and the equation $\psi(x) = true$ has a unique solution in every state S satisfying φ . If this information is available, then A can evaluate φ at a given state S , and provided that φ holds in S , point to the unique solution a of the equation $\psi(x) = true$. To bridge the gap between the formula $\psi(x)$ and the element a , a new nullary function symbol c is introduced, where c is interpreted as the unique solution of the equation $\psi(x) = true$ if φ holds and as *undef* otherwise. Using this approach, any given algorithm A (database transformation or DB-ASM for that matter) can be formalized so that it can access any location of its states, and therefore satisfies the accessibility principle. We avoid the formal details here as this is a well known fact in the ASM community. For the remainder of this paper, we simply assume that every element a of a state S can be accessed by producing an appropriate nullary function symbol c_a .

5.1. Syntax of Rules

For simplicity, we consider function arguments as tuples. That is, if f is an n -ary function and t_1, \dots, t_n are arguments for f , we write $f(t)$ where t is a term which evaluates to the tuple (t_1, \dots, t_n) . Let t and s denote terms in $\mathcal{T}_{\Upsilon, \mathcal{X}_{db}}$, f a dynamic function symbol in Υ and let φ denote an \mathcal{L}^{FO} -formula of vocabulary Υ . The set of *DB-ASM rules* over Υ is inductively defined as follows:

- *assignment rule*: update the content of f at the argument t to s ;

$$f(t) := s$$
- *conditional rule*: execute the rule r if φ is true; otherwise, do nothing;

$$\mathbf{if\ } \varphi \mathbf{\ then\ } r \mathbf{\ endif}$$
- *forall rule*: execute the rule r in parallel for each x satisfying φ ;

$$\mathbf{forall\ } x \mathbf{\ with\ } \varphi \mathbf{\ do\ } r \mathbf{\ enddo}$$
- *choice rule*: choose a value of x that satisfies φ and then execute the rule r ;

$$\mathbf{choose\ } x \mathbf{\ with\ } \varphi \mathbf{\ do\ } r \mathbf{\ enddo}$$

- *parallel rule*: execute the rules r_1 and r_2 in parallel;

par r_1 r_2 **endpar**

- *sequence rule*: first execute the rule r_1 and then execute the rule r_2 ;

seq r_1 r_2 **endseq**

- *let rule*: aggregates, using the location operator ρ , all updates to the location (f, t) yielded by r (see definition of location and location operator in Section 5.2 next);

let $(f, t) \rightarrow \rho$ **in** r **endlet**

Notice that all variables appearing in a DB-ASM rule are *database variables* that must be interpreted by values in B_{db} . A rule r is *closed* if all variables of r are bounded by forall and choice rules.

5.2. Update Sets and Multisets

In the ASM literature [9], locations, updates, update sets and update multisets are the key concepts used to formalise the dynamics of computations. Thus, similar to ASMs [9], DB-ASMs can be understood as an extension of finite state machines which proceed by transitions from states to successor states through updates. In a state of a DB-ASM, updatable dynamic functions are called locations, which are distinguished from static functions that cannot be updated. Informally, such locations represent the abstract concept of basic object containers, such as memory units. During each transition step, a DB-ASM produces a set or multiset of updates that are used to change location contents in a state, and location contents in a DB-ASM can only be changed by such updates.

Let S be a state over Υ , $f \in \Upsilon$ be a dynamic function symbol of arity n and a_1, \dots, a_n be elements in B_{db} or B_a depending on whether $f \in \Upsilon_{db} \cup \mathcal{F}_b$ or $f \in \Upsilon_a$, respectively. Then $(f, (a_1, \dots, a_n))$ is called a *location* of S . An *update* of S is a pair (ℓ, b) , where ℓ is a location and $b \in B_{db}$ or $b \in B_a$, depending on whether $f \in \Upsilon_{db}$ or $f \in \Upsilon_a \cup \mathcal{F}_b$, respectively, is the *update value* of ℓ . To simplify notations, we write $(f, (a_1, \dots, a_n), b)$ for the update (ℓ, b) with the location $\ell = (f, (a_1, \dots, a_n))$. The interpretation of ℓ in S is called the *content* of ℓ in S , denoted by $val_S(\ell)$. An *update set* U is a set of updates; an *update multiset* \ddot{U} is a multiset of updates. A *location operator* ρ is a multiset function that returns a single value from a multiset of values, e.g. AVERAGE, COUNT, SUM, MAX and MIN used in SQL. An update set U is called *consistent* if it does not contain conflicting updates, i.e., for all $(\ell, b), (\ell, b') \in U$ we have $b = b'$. Likewise, we say that an update multiset \ddot{U} is *consistent* if its corresponding update set U , obtained by setting the multiplicity of each element (update) in \ddot{U} to 1, is a consistent update set. Otherwise, we say that \ddot{U} is *inconsistent*. If U is a consistent update set, then there exists a unique state $S + U$ resulting from updating S with U . We have

$$val_{S+U}(\ell) = \begin{cases} b & \text{if } (\ell, b) \in U \\ val_S(\ell) & \text{otherwise} \end{cases}$$

If U is not consistent, then $S + U$ is undefined.

To illustrate the concepts of location operator, update sets and update multisets, we provide the following example in which parallel computations are synchronised by using a let rule.

Example 5.1. Consider the relation ROUTE in Fig. 6 and the two DB-ASMs presented in Fig. 7.

FromCid	ToCid	Distance
c_1	c_2	d_1
c_1	c_5	d_4
c_2	c_3	d_2
c_1	c_4	d_1

Figure 6: A relation ROUTE

```

let ( $\ell_{num}, ()$ )  $\rightarrow$  SUM in
  forall  $x_1, x_2$  with  $\exists x_3(\text{ROUTE}(x_1, x_2, x_3))$  do
     $\ell_{num} := 1$ 
  enddo
endlet

```

(a) First DB-ASM

```

forall  $x_1, x_2$  with  $\exists x_3(\text{ROUTE}(x_1, x_2, x_3))$  do
   $\ell_{num} := 1$ 
enddo

```

(b) Second DB-ASM

Figure 7: Two DB-ASMs

The first DB-ASM in Fig. 7.(a) computes the total number of routes in the relation ROUTE. Here SUM is a location operator assigned to the location $(\ell_{num}, ())$. In a state containing the relation ROUTE in Fig.6, the forall sub-rule yields the update multiset $\{\{(\ell_{num}, (), 1), (\ell_{num}, (), 1), (\ell_{num}, (), 1), (\ell_{num}, (), 1)\}\}$ and the update set $\{(\ell_{num}, (), 1)\}$. In turn the let rule (and thus the DB-ASM) yields the corresponding update set $\{(\ell_{num}, (), 4)\}$, which results from the aggregation produced by the location operator SUM of the four updates to the location $(\ell_{num}, ())$ that appear in the multiset produced by the forall rule. Since the second DB-ASM in Fig. 7.(b) has no location operator associated with the location $(\ell_{num}, ())$ and the forall rule yields the same update multiset and update set as before, this second DB-ASM produces the update set $\{(\ell_{num}, (), 1)\}$ instead.

5.3. Semantics of Rules

The semantics of DB-ASM rules is defined in terms of update multisets and update sets. More specifically, each DB-ASM rule is associated with a set of update multisets, which then “collapses” to a set of update sets. Thus, if r is a DB-ASM rule of signature Υ and S is a state of Υ , then we associate a set $\Delta(r, S, \zeta)$ of update sets and a set $\check{\Delta}(r, S, \zeta)$ of update multisets with r and S , respectively, where ζ is a variable assignment.

Let $\zeta[x \mapsto a]$ denote the variable assignment which coincides with ζ except that it assigns the value a to x . We formally define the sets of update sets and sets of update multisets yielded by DB-ASM rules in Fig. 8 and Fig. 9, respectively. Assignment rules create updates in update sets and multisets. Choice rules introduce non-determinism. Each choice rule generates a *set of update sets* and a corresponding set of update multisets which contain all the different update sets and multisets, respectively, corresponding to all possible choices. Let rules aggregate updates to the same location into a single update by means of location operators. All other rules only rearrange updates into different update sets and multisets.

Lemma 5.1. *For each state S , each DB-ASM rule r and each variable assignment ζ from \mathcal{X}_{db} to the base set B_{db} of the database part of S , the following holds:*

1. $\Delta(r, S, \zeta)$ and $\check{\Delta}(r, S, \zeta)$ are finite sets.
2. Each $U \in \Delta(r, S, \zeta)$ is a finite update set.
3. Each $\check{U} \in \check{\Delta}(r, S, \zeta)$ is a finite update multiset.

Proof. (Sketch). We use structural induction on r . The case of the assignment rule is obvious, as a single update will be created.

The conditional rule either produces exactly the same update sets and multisets as before or a single empty update set and multiset, respectively. For the forall rule the set $V = \{a \in B_{db} \mid \llbracket \varphi \rrbracket_{S, \zeta[x \mapsto a]} = true\}$ is finite, because x ranges over the finite set B_{db} . The stated finiteness then follows by induction, as $\Delta(r, S, \zeta)$ and $\check{\Delta}(r, S, \zeta)$ are finite sets, all $U \in \Delta(r, S, \zeta)$ and all $\check{U} \in \check{\Delta}(r, S, \zeta)$ are finite, and the new update sets and update multisets are built by set and multiset unions, respectively, that range over the finite set V .

For all other rules, the individual update sets and multisets are built by \cup , \uplus , \otimes and aggregation with location operators applied to finite update sets and multisets, which gives the statements 2 and 3. Furthermore, the sets of update sets and update multisets, respectively, are built by comprehensions that range over finite sets. Hence they are finite as well, which gives statement 1 and completes the proof. □

A *Database Abstract State Machine* (DB-ASM) M over signature Υ consists of

- a set \mathcal{S} of states over Υ , non-empty subsets $\mathcal{S}_I \subseteq \mathcal{S}$ of initial states and $\mathcal{S}_F \subseteq \mathcal{S}$ of final states,

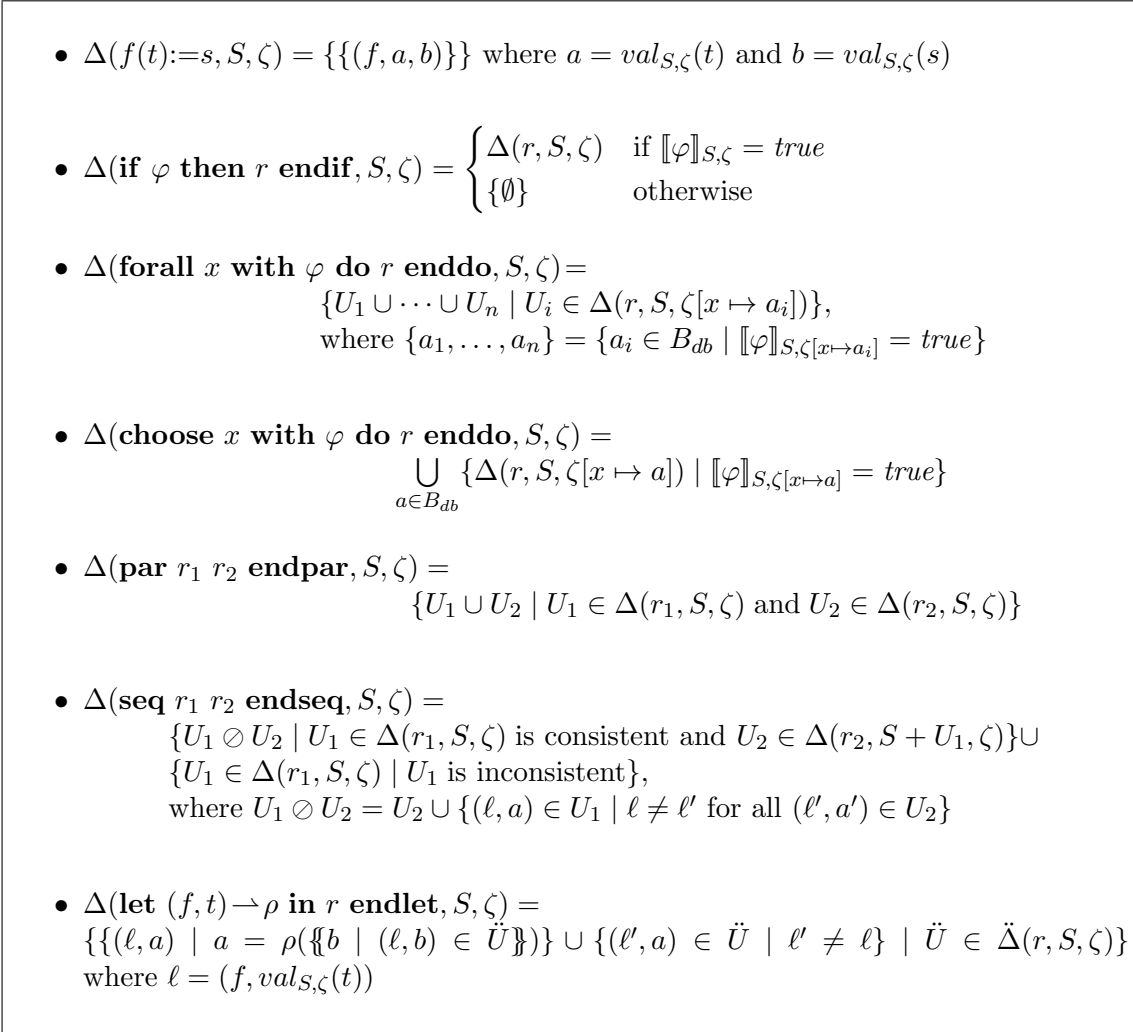


Figure 8: Update sets of DB-ASM rules

- a closed DB-ASM rule r over Υ , and
- a binary successor *relation* δ over \mathcal{S} determined by r , i.e.

$$\delta = \{(S, S + U) \mid U \in \Delta(r, S) \text{ consistent}\},$$

where the set $\Delta(r, S)$ (ζ is omitted from $\Delta(r, S, \zeta)$ since r is closed) of update sets yielded by rule r over the state S defines the successor relation δ of M . A *run* of M is a finite sequence S_0, \dots, S_n of states with $S_0 \in \mathcal{S}_I$, $S_n \in \mathcal{S}_F$, $S_i \notin \mathcal{S}_F$ for $0 < i < n$, and $(S_i, S_{i+1}) \in \delta$ for all $i = 0, \dots, n - 1$.

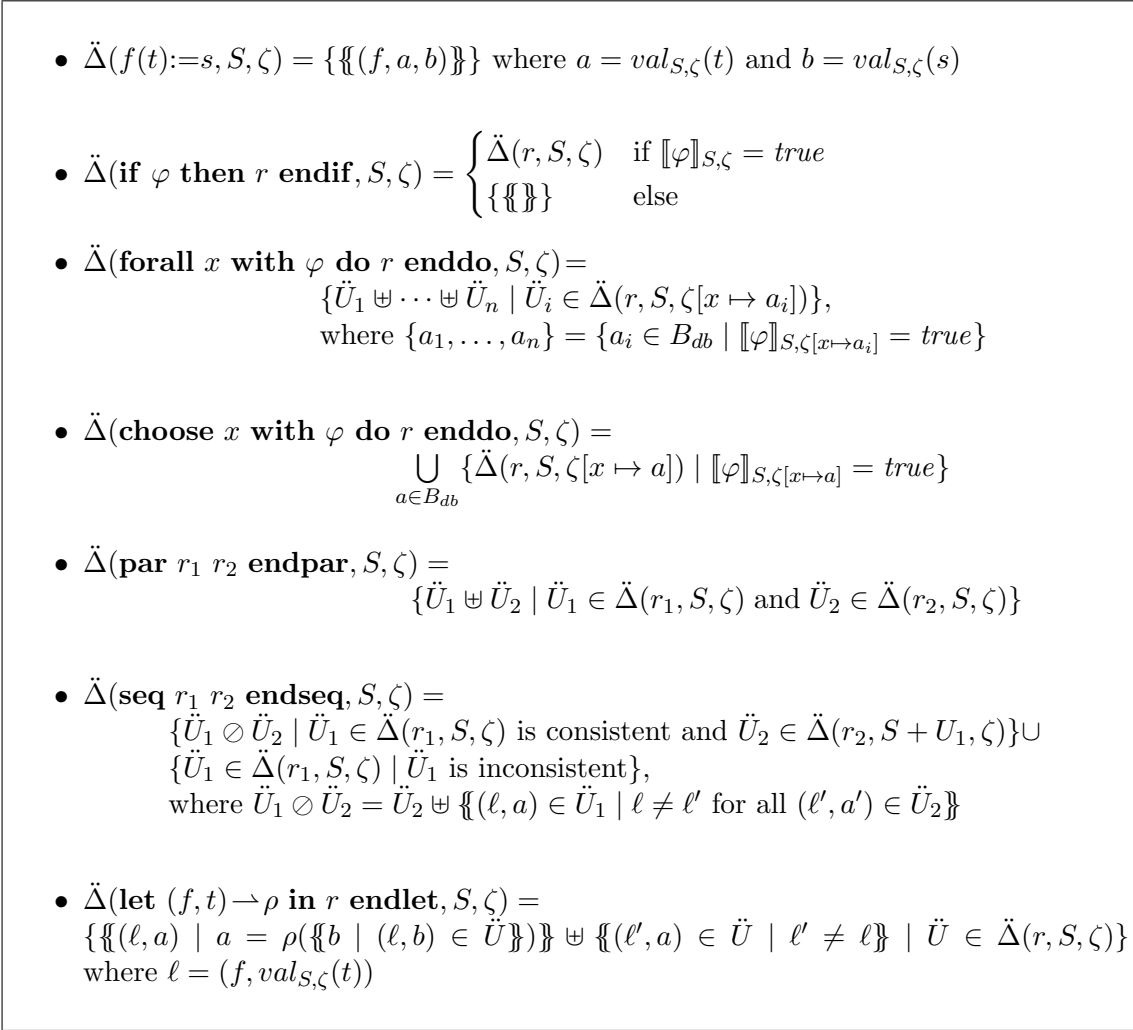


Figure 9: Update multisets of DB-ASM rules

6. A Logic for DB-ASMs

In this section we introduce a logic for DB-ASMs. This logic for DB-ASMs, which we denote as \mathcal{L}^{db} , is built as an extension of the logic \mathcal{L}^{FO} of meta-finite structures used in the formalization of DB-ASMs (see Section 4).

We start with an informal introduction which highlights the main characteristics of \mathcal{L}^{db} and provides some illustrative examples. Then we proceed to introduce its formal syntax and semantics.

Same as in the logic of meta-finite structures \mathcal{L}^{FO} , in \mathcal{L}^{db} we distinguish between *database terms* which are interpreted in the finite primary part of the states of DB-ASMs, and *algorithmic terms* which are interpreted in the possible infinite secondary part. The

set \mathcal{T}_a of algorithmic terms needs however to be extended with first-order variables which range over the secondary part of the state and with a new kind of term (the ρ -terms).

ρ -terms are terms of the form $\rho_v(t|\varphi)$ where ρ is a multiset operator, t is a term in \mathcal{T}_a , v is a variable which ranges either over the primary or secondary part of the state, and φ is a \mathcal{L}^{db} formula. They are interpreted by the value resulting of applying the multiset operator ρ to the multiset resulting of collecting the values of t under all valuations that satisfy φ . The need for ρ -terms arises from the fact that DB-ASMs are able to collect updates yielded in parallel computations under the multiset semantics, i.e., update multisets, and then aggregate updates in an update multiset to an update set by using location operators.

Example 6.1. Consider the relation ROUTE in Fig. 1. The following aggregate queries are expressible by means of ρ -terms.

- Q_1 : Calculate the total number of direct routes.

$$\text{COUNT}_x(1 \mid \exists yz(\text{ROUTE}(x, y, z)))$$

In an SQL database, Q_1 can be expressed by the following SQL statement:

```
SELECT count(*) FROM ROUTE
```

- Q_2 : Find the maximum number of direct connections of any city in the database.

$$\text{MAX}_x(\text{COUNT}_{y'}(1 \mid \exists z(\text{ROUTE}(x, y', z))) \mid \exists yz(\text{ROUTE}(x, y, z)))$$

In a similar way, Q_2 can be expressed by the following SQL statement:

```
SELECT max(NumofConnections)
FROM (SELECT Cid, count(*) as NumofConnections
      FROM ROUTE
      GROUP BY Cid)
```

Due to the importance of non-determinism for enhancing the expressive power of database transformations, DB-ASMs include a non-deterministic choice rule. Consequently, it is no longer enough to consider the individual updates associated to the *unique* update set produced by a rule of a deterministic ASM, as it is the case in the logic for ASMs [39] of Nanchen and Stärk. Instead, the logic \mathcal{L}^{db} needs to be able to describe properties of the different update sets (and multisets) which can be associated to a given DB-ASM rule. That is, the logic \mathcal{L}^{db} should allow us to handle multiple update sets, since a non-deterministic DB-ASM rule can produce a possible different update set for each of the possible choices.

A natural and concise way of handling update sets (and multisets) is by means of second-order variables and second-order quantification. We therefore include both in the language of \mathcal{L}^{db} , albeit with a Henkin's semantics instead of the standard Tarski's semantics, so that we can avoid the well known incompleteness result of second-order logic. For the same reason we additionally include the multi-modal operator $[X]$, where X is a second-order variable of arity 3. The intended meaning of a formula $[X]\varphi$ is that φ is true in the state obtained by applying the updates in X to the current state. In turn, to express that X is the update set produced by a rule r , we also include atomic formulae of the form $\text{upd}(r, X)$ to the language of \mathcal{L}^{db} .

In order to encode update sets into second-order variables, we need to make some assumptions more precise.

Definition 6.1. Given a DB-ASM of some schema Υ , we extend the sub-schema Υ_a of the algorithmic part with a new nullary and static function symbol (constant) c_{f_i} for each dynamic function $f_i \in \Upsilon$. We assume that in every state S , these new constant symbols are interpreted by arbitrary, but pairwise different values. That is, if c_{f_i} and c_{f_j} are among the new constant symbols, then $c_{f_i}^S \neq c_{f_j}^S$.

Let S be a state of this extended signature Υ , let ζ be a variable assignment into S , let X be a second-order variable of arity 3 and let $U = \{(f_i, a_1, a_2) \mid (a_0, a_1, a_2) \in \zeta(X) \text{ and } a_0 = c_{f_i}^S\}$. We say $\zeta(X)$ *represents* U if U constitutes an update set for S and $(a_0, a_1, a_2) \in \zeta(X)$ iff $a_0 = c_f^S$ for some dynamic function $f \in \Upsilon$ and $(f, a_1, a_2) \in U$.

As noted earlier, the multiset semantics allows DB-ASMs to collect updates yielded in parallel computations, i.e., update multisets. This multiset semantics is handled via the inclusion of atomic formulae of the form $\text{upm}(r, X)$. In this case, the intended meaning is that $\text{upm}(r, X)$ is true if X is a second-order variable of arity 4 which represents an update multiset yielded by the rule r . We say that X represents an update multiset \ddot{U} iff for every update $(f, a_0, a_1) \in \ddot{U}$ with multiplicity $n > 0$ there are *exactly* n distinct b_1, \dots, b_n such that $(f, a_0, a_1, b_i) \in X$ and vice versa.

Example 6.2. Consider Example 3.1 and the corresponding DB-ASM depicted in Fig. 5. Let r denote the main rule of the DB-ASM and S denote one of its states. The following formulae illustrate how the logic \mathcal{L}^{db} can be used to express desirable properties of this DB-ASM.

1. If the rule r over S yields an update set U containing an update $(\text{VISITED}, x, \text{TRUE})$, then for every neighbour city y of x , the (current) shortest distance in state $S + U$ (calculated by the algorithm) between y and c is no longer INFINITY. Representing U by the second-order variable X , we obtain:

$$\exists Xx \left(\text{upd}(r, X) \wedge X(c_{\text{VISITED}}, x, \text{TRUE}) \rightarrow [X] \forall yz (\text{ROUTE}(x, y, z) \rightarrow \text{DIST}(y) \neq \text{INFINITY}) \right)$$

2. If in the current state S , the distance between a non-visited (by the algorithm) city x and c is minimal among the non-visited cities, then there is an update set U yielded by the rule r in state S which updates the status of x to visited. Representing U by the second-order variable X , we obtain:

$$\exists x \left(\neg \text{VISITED}(x) \wedge \forall y (\neg \text{VISITED}(y) \rightarrow \text{DIST}(x) \leq \text{DIST}(y)) \rightarrow \right. \\ \left. \exists X (\text{upd}(r, X) \wedge [X] \text{VISITED}(x)) \right)$$

3. If the current state S is not an initial nor a final state and U is an update set yielded by the rule r in S , then the value of MDIST in the successor state $S + U$ equals the distance between c and the closest unvisited (by the algorithm in state S) city. Representing U by the second-order variable X and using a ρ -term with location operator MIN, we obtain:

$$\forall X \left(\neg \text{INITIAL} \wedge \exists xy (\text{CITY}(x, y) \wedge \neg \text{VISITED}(x)) \wedge \text{upd}(r, X) \rightarrow \right. \\ \left. [X] \text{MDIST} = \text{MIN}_x (\text{DIST}(x) \mid \exists y (\text{CITY}(x, y) \wedge \neg \text{VISITED}(x))) \right)$$

4. If the current state S is not an initial nor a final state and U is an update set yielded by the rule r in S , then every update multiset \ddot{U} yielded by r in S contains at least one update $(\text{MDIST}, (), a_i)$ such that a_i coincides with the value stored in the location $(\text{MDIST}, ())$ in the successor state $S + U$ and $a_i \leq a_j$ for every update $(\text{MDIST}, (), a_j)$ in \ddot{U} . Representing U and \ddot{U} by the second-order variables X and Y , respectively, we obtain:

$$\forall X \left(\neg \text{INITIAL} \wedge \exists xy (\text{CITY}(x, y) \wedge \neg \text{VISITED}(x)) \wedge \text{upd}(r, X) \rightarrow \right. \\ \forall Y (\text{upm}(r, Y) \rightarrow \exists \mathbf{x} \mathbf{y} (Y(c_{\text{MDIST}}, (), \mathbf{x}, \mathbf{y}) \wedge [X] \text{MDIST} = \mathbf{x} \wedge \\ \forall \mathbf{x}' \mathbf{y}' (Y(c_{\text{MDIST}}, (), \mathbf{x}', \mathbf{y}') \rightarrow \mathbf{x} \leq \mathbf{x}')))) \left. \right)$$

6.1. Syntax

The set of database and algorithmic terms of the logic \mathcal{L}^{db} is defined as follows.

Definition 6.2. Let $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$ be a signature of meta-finite states and let Λ denote a set of location operators. Fix a countable set $\mathcal{X} = \mathcal{X}_{db} \cup \mathcal{X}_a$ of first-order variables. Variables in \mathcal{X}_{db} , denoted with standard lowercase letters x, y, z, \dots , range over the database part of meta-finite states (i.e., the finite base set B_{db}), whereas variables in \mathcal{X}_a , denoted with typewriter-style lowercase letters $\mathbf{x}, \mathbf{y}, \mathbf{z}, \dots$, range over the algorithmic part of meta-finite states (i.e. the possible infinite base set B_a). The set of terms of \mathcal{L}^{db} is formed by the set \mathcal{T}_{db} of *database terms* and the set \mathcal{T}_a of *algorithmic terms* as defined by the following rules:

- If $x \in \mathcal{X}_{db}$, then x is a database term in \mathcal{T}_{db} .

- If $\mathbf{x} \in \mathcal{X}_a$, then \mathbf{x} is an algorithmic term in \mathcal{T}_a .
- If $f \in \Upsilon_{db}$ and $t \in \mathcal{T}_{db}$, then $f(t)$ is a database term in \mathcal{T}_{db} .
- If $f \in \mathcal{F}_b$ and $t \in \mathcal{T}_{db}$, then $f(t)$ is an algorithmic term in \mathcal{T}_a .
- If $f \in \Upsilon_a$ and $t \in \mathcal{T}_a$, then $f(t)$ is an algorithmic term in \mathcal{T}_a .
- If ρ is a location operator in Λ , φ is \mathcal{L}^{db} -formula (as in Definition 6.3 below), $t \in \mathcal{T}_a$ and $v \in \mathcal{X}_{db} \cup \mathcal{X}_a$, then $\rho_v(t \mid \varphi)$ is an algorithmic term in \mathcal{T}_a .
- Nothing else is a term in \mathcal{T}_{db} or \mathcal{T}_a .

A *pure term* is defined as a term that is *not* a ρ -term and does *not* contain any sub-term which is a ρ -term.

Next, we formally introduce the set of well formed formulae of \mathcal{L}^{db} .

Definition 6.3. Let $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$ be a signature of meta-finite states and let Λ denote a set of location operators. Let \mathcal{T}_{db} and \mathcal{T}_a be the corresponding set of database and algorithmic terms over Υ and Λ (as per Definition 6.2). Extend the set \mathcal{X} of first-order variables with a countable set of second-order (relation) variables of arity r for each $r \geq 1$. The following rules define the set of well formed formulae (wff) of \mathcal{L}^{db} .

1. If s and t are terms in \mathcal{T}_{db} , then $s = t$ is a wff.
2. If s and t are terms in \mathcal{T}_a , then $s = t$ is a wff.
3. If t_1, \dots, t_r are terms in $\mathcal{T}_{db} \cup \mathcal{T}_a$ and X is a second-order variable of arity r , then $X(t_1, \dots, t_r)$ is a wff.
4. If r is a DB-ASM rule and X is a second-order variable of arity 3, then $\text{upd}(r, X)$ is a wff.
5. If r is a DB-ASM rule and X is a second-order variable of arity 4, then $\text{upm}(r, X)$ is a wff.
6. If φ is a wff, then $(\neg\varphi)$ is a wff.
7. If φ and ψ are wff's, then $(\varphi \vee \psi)$ is a wff.
8. If φ is a wff and $x \in \mathcal{X}_{db}$, then $\forall x(\varphi)$ is a wff.
9. If φ is a wff and $\mathbf{x} \in \mathcal{X}_a$, then $\forall \mathbf{x}(\varphi)$ is a wff.
10. If φ is a wff and X is a second-order variable, then $\forall X(\varphi)$ is a wff.
11. If φ is a wff and X is a second-order variable of arity 3, then $([X]\varphi)$ is a wff.
12. Nothing else is a wff.

Formulae of the form $\varphi \wedge \psi$, $\varphi \rightarrow \psi$, $\exists x(\varphi)$, $\exists \mathbf{x}(\varphi)$ and $\exists X(\varphi)$ are considered as abbreviations of $\neg(\neg\varphi \vee \neg\psi)$, $\neg\varphi \vee \psi$, $\neg\forall x(\neg\varphi)$, $\neg\forall \mathbf{x}(\neg\varphi)$ and $\neg\forall X(\neg\varphi)$, respectively.

We say that a formula of \mathcal{L}^{db} is a *pure formula* if it can be defined using only the rules 1–2 and 6–9 in Definition 6.3 and does *not* contains any ρ -term. Notice that the formula occurring in the **if**, **forall** and **choose** rules of DB-ASMs satisfy this definition, i.e., they

are pure formulae of \mathcal{L}^{db} . We also say that a term or formula of \mathcal{L}^{db} is *static* if it does *not* contain any dynamic function symbol. Since static functions cannot be updated, it is clear that the value of a static term as well as the truth value of a static formula cannot change during the run of a DB-ASM.

The atomic formulae $\text{upd}(r, X)$ and $\text{upm}(r, X)$ are not strictly necessary. As shown latter (see Lemmas 7.1 and 7.2), they can be eliminated from the language of \mathcal{L}^{db} without affecting its expressive power.

6.2. Semantics

We use a semantics due to Henkin [24], in which the interpretation of second-order quantifiers is part of the specification of a structure (state) rather than an invariant through all models as in the case of the standard Tarski's semantics.

Definition 6.4. Let $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$ be a signature of meta-finite states. A *Henkin meta-finite Υ -prestructure* S is a meta-finite state of signature Υ and nonempty base set $B = B_{db} \cup B_a$, which is extended with a new universe D_n of n -ary relations for each $n \geq 1$, where $D_n \subseteq \mathcal{P}(\underbrace{B \times \dots \times B}_n)$.

Variable assignments into a Henkin meta-finite prestructure S are defined as usual, except that we require that every assignment ζ satisfies the following conditions:

- If x is a first-order variable in \mathcal{X}_{db} , then $\zeta(x) \in B_{db}$.
- If \mathbf{x} is a first-order variable in \mathcal{X}_a , then $\zeta(\mathbf{x}) \in B_a$.
- If X is a second-order variable of arity n , then $\zeta(X) \in D_n$.

Given a variable assignment, terms of the logic \mathcal{L}^{db} can be interpreted into a Henkin meta-finite prestructure.

Definition 6.5. Let S be a Henkin meta-finite prestructure of signature $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$ and let ζ be a variable assignment into S . If t is a term (either a database term or an algorithmic term), then the value (interpretation) of t in S under ζ (denoted $\text{val}_{S,\zeta}(t)$) is defined by the following rules:

- If t is a variable $x \in \mathcal{X}_{db}$ or $\mathbf{x} \in \mathcal{X}_a$, then $\text{val}_{S,\zeta}(t) = \zeta(t)$.
- If t is of the form $f(t')$ where $f \in \Upsilon$ and t' is a term, then $\text{val}_{S,\zeta}(t) = f^S(\text{val}_{S,\zeta}(t'))$.
- If t is of the form $\rho_v(t' \mid \varphi)$, then

$$\text{val}_{S,\zeta}(t) = \rho(\{\{\text{val}_{S,\zeta[v \mapsto a_i]}(t') \mid a_i \in D \text{ and } \llbracket \varphi \rrbracket_{S,\zeta[v \mapsto a_i]} = \text{true}\}\}),$$

where $D = B_{db}$ or $D = B_a$ depending on whether $v \in \mathcal{X}_{db}$ or $v \in \mathcal{X}_a$, respectively.

The interpretation of \mathcal{L}^{db} -formulae into Henkin meta-finite prestructures is defined as follows.

Definition 6.6. Let S be a Henkin meta-finite prestructure of signature $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$, extended as per Definition 6.1 with a new and different constant symbol c_{f_i} for each dynamic function symbol $f_i \in \Upsilon$. Let ζ be a variable assignment into S . For X a second-order variable of arity 3, we abuse the notation by writing $val_{S,\zeta}(X) \in \Delta(r, S, \zeta)$, meaning that there is a set $U \in \Delta(r, S, \zeta)$ such that $(f, a_0, a_1) \in U$ iff $(c_f^S, a_0, a_1) \in val_{S,\zeta}(X)$. Likewise, for X a second-order variable of arity 4, we write $val_{S,\zeta}(X) \in \ddot{\Delta}(r, S, \zeta)$, meaning that there is a multiset $\ddot{U} \in \Delta(r, S, \zeta)$ such that $(f, a_0, a_1) \in \ddot{U}$ with multiplicity n iff there are exactly b_1, \dots, b_n pairwise different values such that $(c_f^S, a_0, a_1, b_i) \in val_{S,\zeta}(X)$ for every $1 \leq i \leq n$.

If φ is an \mathcal{L}^{db} -formula, then the truth value of φ on S under ζ (denoted as $\llbracket \varphi \rrbracket_{S,\zeta}$) is either *true* or *false* and it is determined by the following rules:

- If φ is of the form $s = t$, then $\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } val_{S,\zeta}(s) = val_{S,\zeta}(t); \\ false & \text{otherwise.} \end{cases}$
- If φ is of the form $X(t_1, \dots, t_r)$, then $\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } (val_{S,\zeta}(t_1), \dots, val_{S,\zeta}(t_r)) \in val_{S,\zeta}(X); \\ false & \text{otherwise.} \end{cases}$
- If φ is of the form $upd(r, X)$, then $\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } val_{S,\zeta}(X) \in \Delta(r, S, \zeta); \\ false & \text{otherwise.} \end{cases}$
- If φ is of the form $upm(r, X)$, then $\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } val_{S,\zeta}(X) \in \ddot{\Delta}(r, S, \zeta); \\ false & \text{otherwise.} \end{cases}$
- If φ is of the form $(\neg\psi)$, then $\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } \llbracket \psi \rrbracket_{S,\zeta} = false; \\ false & \text{otherwise.} \end{cases}$
- If φ is of the form $(\alpha \vee \psi)$, then $\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } \llbracket \alpha \rrbracket_{S,\zeta} = true \text{ or } \llbracket \psi \rrbracket_{S,\zeta} = true; \\ false & \text{otherwise.} \end{cases}$
- If φ is of the form $\forall x(\psi)$, then $\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } \llbracket \psi \rrbracket_{S,\zeta[x \mapsto a]} = true \text{ for all } a \in B_{db}; \\ false & \text{otherwise.} \end{cases}$
- If φ is of the form $\forall \mathbf{x}(\psi)$, then $\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } \llbracket \psi \rrbracket_{S,\zeta[\mathbf{x} \mapsto a]} = true \text{ for all } a \in B_a; \\ false & \text{otherwise.} \end{cases}$

- If φ is of the form $\forall X(\psi)$, where X is a second-order variable of arity n , then
$$\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } \llbracket \psi \rrbracket_{S,\zeta[X \mapsto R]} = true \text{ for all } R \in D_n; \\ false & \text{otherwise.} \end{cases}$$
- If φ is of the form $([X]\psi)$, then
$$\llbracket \varphi \rrbracket_{S,\zeta} = \begin{cases} false & \text{if } \zeta(X) \text{ represents (as per Definition 6.1) an update set } U \\ & \text{such that } U \text{ is consistent and } \llbracket \psi \rrbracket_{S+U,\zeta} = false; \\ true & \text{otherwise.} \end{cases}$$

Remark 1. Note that if φ is of the form $([X]\psi)$, then φ is interpreted as *true* in any of the following cases:

- $\zeta(X)$ represents an update set U which is inconsistent.
- $\zeta(X)$ does *not* represent an update set.
- $\zeta(X)$ represents a consistent update set U and ψ is interpreted as *true* in $S + U$.

For a sentence φ of \mathcal{L}^{db} to be valid in the given Henkin's semantics, it must be true in all Henkin meta-finite prestructures. This is a stronger requirement than saying that φ is valid in the standard Tarski's semantics. A sentence that is valid in the standard Tarski's semantics is true in those Henkin meta-finite prestructures for which each universe D_n is interpreted as the set of all relations of arity n . But such a sentence φ might turn out to be false in some Henkin meta-finite prestructure (i.e., $\neg\varphi$ might evaluate to true in some Henkin meta-finite prestructure).

Clearly, we do *not* want the universes D_n of the Henkin meta-finite prestructures to be any arbitrary collections of n -ary relations. It is then reasonable to restrict our attention to some collections of n -ary relations that we know about, because we can define them.

Definition 6.7. A *Henkin meta-finite structure* for a second-order language is a Henkin meta-finite prestructure S that is closed under definability, i.e., for every formula φ , variable assignment ζ and arity $n \geq 1$, we have that

$$\{\bar{a} \in A^n \mid \llbracket \varphi \rrbracket_{S,\zeta[a_1 \mapsto x_1, \dots, a_n \mapsto x_n]} = true\} \in D_n.$$

In the following, we restrict our attention to Henkin meta-finite structures. Notice that, if M is a DB-ASM of some vocabulary Υ of meta-finite structures, we can use \mathcal{L}^{db} formulae of the vocabulary Υ (extended with constant symbols c_{f_i} for each dynamic function symbol $f_i \in \Upsilon$ as per Definition 6.1) to express properties of M . We can then verify these properties by evaluating the formulae over appropriate Henkin meta-finite structures of the (extended) vocabulary Υ , and use the complete proof system which we introduce next to derive logical consequences.

7. A Proof System

In this section we develop a proof system for the logic \mathcal{L}^{db} for DB-ASMs.

Definition 7.1. We say that a Henkin meta-finite structure S is a *model* of a formula φ (denoted as $S \models \varphi$) iff $\llbracket \varphi \rrbracket_{S, \zeta} = true$ holds for every variable assignment ζ . If Ψ is a set of formulae, we say that S *models* Ψ (denoted as $S \models \Psi$) iff $S \models \varphi$ for each $\varphi \in \Psi$. A formula φ is said to be a *logical consequence* of a set Ψ of formulae (denoted as $\Psi \models \varphi$) if for every Henkin meta-finite structure S , if $S \models \Psi$, then $S \models \varphi$. A formula φ is said to be *valid* (denoted as $\models \varphi$) if $\llbracket \varphi \rrbracket_{S, \zeta} = true$ in every Henkin meta-finite structure S with every variable assignment ζ . A formula φ is said to be *derivable* from a set Ψ of formulae (denoted as $\Psi \vdash_{\mathfrak{R}} \varphi$) if there is a deduction from formulae in Ψ to φ by using a set \mathfrak{R} of axioms and inference rules.

We will define such a set \mathfrak{R} of axioms and rules in Subsection 7.4. Then we simply write \vdash instead of $\vdash_{\mathfrak{R}}$. We also define equivalence between two DB-ASM rules.

Definition 7.2. Let r_1 and r_2 be two DB-ASM rules. Then r_1 and r_2 are *equivalent* (denoted as $r_1 \equiv r_2$) if for every Henkin meta-finite structure S it holds that

$$S \models \forall X(\text{upd}(r_1, X) \leftrightarrow \text{upd}(r_2, X)).$$

The substitution of a term t for a variable x in a formula φ (denoted as $\varphi[t/x]$) is defined by the rule of substitution. That is, $\varphi[t/x]$ is the result of replacing all free instances of x by t in φ provided that no free variable of t becomes bound after substitution.

7.1. Consistency

In [39] Nanchen and Stärk use a predicate $\text{Con}(r)$ as an abbreviation for the statement that the rule r is consistent. As a rule r in their work is considered to be deterministic, there is no ambiguity with the reference to the update set associated with r , i.e., each deterministic rule r generates exactly one (possibly empty) update set. Thus a deterministic rule r is consistent iff the update set generated by r is consistent. However, in the logic for DB-ASMs, the presence of non-determinism makes the situation less straightforward.

Instead, given a rule r of a signature $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$ of a DB-ASM, we can use $\text{con}(r, X)$ to express that X represents one of the possible update sets generated by the rule r (which in our setting can be non-deterministic) and that X is consistent. This can be expressed in the logic \mathcal{L}^{db} with the following formula:

$$\text{con}(r, X) \equiv \text{upd}(r, X) \wedge \text{conUSet}(X) \tag{1}$$

where

$$\begin{aligned} \text{conUSet}(X) \equiv & \bigwedge_{c_f \in \mathcal{F}_{dyn} \wedge f \in \Upsilon_{db}} \forall xyz((X(c_f, x, y) \wedge X(c_f, x, z)) \rightarrow y = z) \wedge \\ & \bigwedge_{c_f \in \mathcal{F}_{dyn} \wedge f \in \Upsilon_a} \forall xyz((X(c_f, x, y) \wedge X(c_f, x, z)) \rightarrow y = z) \wedge \\ & \bigwedge_{c_f \in \mathcal{F}_{dyn} \wedge f \in \mathcal{F}_b} \forall xyz((X(c_f, x, y) \wedge X(c_f, x, z)) \rightarrow y = z) \end{aligned} \quad (2)$$

for \mathcal{F}_{dyn} the set of constants representing the dynamic function symbols in Υ (see Definition 6.1).

As the rule r may be non-deterministic, it is possible that r yields several update sets. Thus, we develop the consistency of DB-ASM rules in two versions:

- A rule r is *weakly consistent* (denoted as $\text{wcon}(r)$) if at least one update set generated by r is consistent. This can be expressed as follows:

$$\text{wcon}(r) \equiv \exists X(\text{con}(r, X)) \quad (3)$$

- A rule r is *strongly consistent* (denoted as $\text{scon}(r)$) if every update set generated by r is consistent. This can be expressed as follows:

$$\text{scon}(r) \equiv \forall X(\text{upd}(r, X) \rightarrow \text{conUSet}(X)) \quad (4)$$

In the case that a rule r is deterministic, the weak notion of consistency coincides with the strong notion of consistency, i.e., $\text{wcon}(r) \equiv \text{scon}(r)$.

7.2. Update Sets

We present the axioms for the predicate $\text{upd}(r, X)$ of the logic \mathcal{L}^{db} . Since a DB-ASM rules may be non-deterministic, a straightforward extension of the formalisation of upd for the forall and parallel rules used in the logic for ASMs [39] is not sufficient in our case (cf. Axioms **U3** and **U4** below with the corresponding axioms in [39]).

As before, we assume that if f is a dynamic function symbol in the given signature of meta-finite states $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$, then there is a corresponding constant (static nullary function) symbol $c_f \in \Upsilon_a$ as per Definition 6.1. We use \mathcal{F}_{dyn} to denote the set of all c_f such that f is a dynamic function symbol in Υ . In the following, S denotes an arbitrary Henkin structure of signature Υ and $B = B_{db} \cup B_a$ denotes the base set (domain) of the database and algorithmic parts of S . W.l.o.g. we further assume that $B_{db} \cap B_a = \emptyset$.

Let ζ be a valuation into S . In the formulation of the axioms we use the predicate $\text{isUSet}(X)$ to denote that $\zeta(X)$ represents an update set for S (see Definition 6.1). That is, for every triple $(a_1, a_2, a_3) \in \zeta(X)$, we have that $a_1 = c_f^S$ for some dynamic function

$f \in \Upsilon$ and a_2, a_3 are values of the appropriate database or algorithmic base sets depending on whether f is a database, algorithmic or bridge function symbol.

$$\begin{aligned}
\text{isUSet}(X) &\equiv \forall \mathbf{x}_1 x_2 x_3 \left(X(\mathbf{x}_1, x_2, x_3) \rightarrow \bigvee_{c_f \in \mathcal{F}_{dyn} \wedge f \in \Upsilon_{db}} \mathbf{x}_1 = c_f \right) \wedge \\
&\quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \left(X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \rightarrow \bigvee_{c_f \in \mathcal{F}_{dyn} \wedge f \in \Upsilon_a} \mathbf{x}_1 = c_f \right) \wedge \\
&\quad \forall \mathbf{x}_1 x_2 \mathbf{x}_3 \left(X(\mathbf{x}_1, x_2, \mathbf{x}_3) \rightarrow \bigvee_{c_f \in \mathcal{F}_{dyn} \wedge f \in \mathcal{F}_b} \mathbf{x}_1 = c_f \right) \wedge \\
&\quad \forall \mathbf{x}_1 \mathbf{x}_2 x_3 (\neg X(\mathbf{x}_1, \mathbf{x}_2, x_3)) \wedge \\
&\quad \forall x_1 x_2 x_3 (\neg X(x_1, x_2, x_3)) \wedge \forall \mathbf{x}_1 \mathbf{x}_2 x_3 (\neg X(\mathbf{x}_1, \mathbf{x}_2, x_3)) \wedge \\
&\quad \forall x_1 x_2 \mathbf{x}_3 (\neg X(x_1, x_2, \mathbf{x}_3)) \wedge \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3))
\end{aligned}$$

The axioms for $\text{upd}(r, X)$ are as follows (cf. the definition of update sets in Fig. 8):

- Our first three axioms express that X represents an update set yielded by the assignment rule $f(t) := s$ iff it contains exactly one update which is (f, t, s) .

U1.1: If f is a database function symbol in Υ_{db} then

$$\begin{aligned}
\text{upd}(f(t) := s, X) &\leftrightarrow \text{isUSet}(X) \wedge X(c_f, t, s) \wedge \\
&\quad \forall \mathbf{x}_1 x_2 x_3 (X(\mathbf{x}_1, x_2, x_3) \rightarrow \mathbf{x}_1 = c_f \wedge x_2 = t \wedge x_3 = s) \wedge \\
&\quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)) \wedge \forall \mathbf{x}_1 x_2 \mathbf{x}_3 (\neg X(\mathbf{x}_1, x_2, \mathbf{x}_3))
\end{aligned}$$

U1.2: If f is an algorithmic function symbol in Υ_a then

$$\begin{aligned}
\text{upd}(f(t) := s, X) &\leftrightarrow \text{isUSet}(X) \wedge X(c_f, t, s) \wedge \\
&\quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \rightarrow \mathbf{x}_1 = c_f \wedge \mathbf{x}_2 = t \wedge \mathbf{x}_3 = s) \wedge \\
&\quad \forall \mathbf{x}_1 x_2 x_3 (\neg X(\mathbf{x}_1, x_2, x_3)) \wedge \forall \mathbf{x}_1 x_2 \mathbf{x}_3 (\neg X(\mathbf{x}_1, x_2, \mathbf{x}_3))
\end{aligned}$$

U1.3: If f is a bridge function symbol in \mathcal{F}_b then

$$\begin{aligned}
\text{upd}(f(t) := s, X) &\leftrightarrow \text{isUSet}(X) \wedge X(c_f, t, s) \wedge \\
&\quad \forall \mathbf{x}_1 x_2 \mathbf{x}_3 (X(\mathbf{x}_1, x_2, \mathbf{x}_3) \rightarrow \mathbf{x}_1 = c_f \wedge x_2 = t \wedge \mathbf{x}_3 = s) \wedge \\
&\quad \forall \mathbf{x}_1 x_2 x_3 (\neg X(\mathbf{x}_1, x_2, x_3)) \wedge \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3))
\end{aligned}$$

- Axiom **U2** asserts that, if the formula φ evaluates to *true*, then X is an update set yielded by the conditional rule **if** φ **then** r **endif** iff X is an update set yielded by the rule r . Otherwise, the conditional rule yields only an empty update set.

U2: $\text{upd}(\text{if } \varphi \text{ then } r \text{ endif}, X) \leftrightarrow (\varphi \wedge \text{upd}(r, X)) \vee$

$$\begin{aligned}
&\quad (\neg \varphi \wedge \text{isUSet}(X) \wedge \forall \mathbf{x}_1 x_2 x_3 (\neg X(\mathbf{x}_1, x_2, x_3))) \wedge \\
&\quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)) \wedge \forall \mathbf{x}_1 x_2 \mathbf{x}_3 (\neg X(\mathbf{x}_1, x_2, \mathbf{x}_3))
\end{aligned}$$

- Axiom **U3** states that X is an update set yielded by the rule **forall** x **with** φ **do** r **enddo** iff X coincides with $U_{a_1} \cup \dots \cup U_{a_n}$, where $\{a_1, \dots, a_n\} = \{a_i \in B_{db} \mid \text{val}_{S, \zeta[x \mapsto a_i]}(\varphi) = \text{true}\}$ and U_{a_i} (for $1 \leq i \leq n$) is an update set yielded by the rule r under the variable assignment $\zeta[x \mapsto a_i]$. Note that the update sets U_{a_1}, \dots, U_{a_n} are encoded into the second-order variable Y of arity four.

$$\begin{aligned}
\mathbf{U3:} \quad & \text{upd}(\text{forall } x \text{ with } \varphi \text{ do } r \text{ enddo}, X) \leftrightarrow \text{isUSet}(X) \wedge \\
& \exists Y (\forall \mathbf{z} y_1 y_2 (X(\mathbf{z}, y_1, y_2) \leftrightarrow \exists x (Y(\mathbf{z}, y_1, y_2, x)))) \wedge \\
& \quad \forall \mathbf{z} y_1 y_2 (X(\mathbf{z}, y_1, y_2) \leftrightarrow \exists x (Y(\mathbf{z}, y_1, y_2, x))) \wedge \\
& \quad \forall \mathbf{z} y_1 y_2 (X(\mathbf{z}, y_1, y_2) \leftrightarrow \exists x (Y(\mathbf{z}, y_1, y_2, x))) \wedge \\
& \quad \forall x ((\varphi \rightarrow \exists Z (\text{upd}(r, Z) \wedge \\
& \quad \quad \forall \mathbf{z} y_1 y_2 (Z(\mathbf{z}, y_1, y_2) \leftrightarrow Y(\mathbf{z}, y_1, y_2, x))) \wedge \\
& \quad \quad \forall \mathbf{z} y_1 y_2 (Z(\mathbf{z}, y_1, y_2) \leftrightarrow Y(\mathbf{z}, y_1, y_2, x))) \wedge \\
& \quad \quad \forall \mathbf{z} y_1 y_2 (Z(\mathbf{z}, y_1, y_2) \leftrightarrow Y(\mathbf{z}, y_1, y_2, x)))) \wedge \\
& (\neg \varphi \rightarrow \forall \mathbf{z} y_1 y_2 (\neg Y(\mathbf{z}, y_1, y_2, x))) \wedge \\
& \quad \forall \mathbf{z} y_1 y_2 (\neg Y(\mathbf{z}, y_1, y_2, x))) \wedge \\
& \quad \forall \mathbf{z} y_1 y_2 (\neg Y(\mathbf{z}, y_1, y_2, x))))
\end{aligned}$$

- Axiom **U4** states that X is an update set yielded by the parallel rule **par** r_1 r_2 **endpar** iff it corresponds to the union of an update set yielded by r_1 and an update set yielded by r_2 .

$$\begin{aligned}
\mathbf{U4:} \quad & \text{upd}(\text{par } r_1 \ r_2 \ \text{endpar}, X) \leftrightarrow \text{isUSet}(X) \wedge \\
& \exists Y_1 Y_2 (\text{upd}(r_1, Y_1) \wedge \text{upd}(r_2, Y_2) \wedge \\
& \quad \forall \mathbf{z} y_1 y_2 (X(\mathbf{z}, y_1, y_2) \leftrightarrow (Y_1(\mathbf{z}, y_1, y_2) \vee Y_2(\mathbf{z}, y_1, y_2)))) \wedge \\
& \quad \forall \mathbf{z} y_1 y_2 (X(\mathbf{z}, y_1, y_2) \leftrightarrow (Y_1(\mathbf{z}, y_1, y_2) \vee Y_2(\mathbf{z}, y_1, y_2))) \wedge \\
& \quad \forall \mathbf{z} y_1 y_2 (X(\mathbf{z}, y_1, y_2) \leftrightarrow (Y_1(\mathbf{z}, y_1, y_2) \vee Y_2(\mathbf{z}, y_1, y_2))))
\end{aligned}$$

- Axiom **U5** asserts that X is an update set yielded by the rule **choose** x **with** φ **do** r **enddo** iff it is an update set yielded by the rule r under a variable assignment $\zeta[x \mapsto a]$ which satisfies φ .

$$\mathbf{U5:} \quad \text{upd}(\text{choose } x \text{ with } \varphi \text{ do } r \text{ enddo}, X) \leftrightarrow \exists x (\varphi \wedge \text{upd}(r, X))$$

- Axiom **U6** asserts that X is an update set yielded by a sequence rule **seq** r_1 r_2 **endseq** iff it corresponds either to an inconsistent update set yielded by rule r_1 , or to an update set formed by the updates in an update set Y_2 yielded by rule r_2 in a successor state $S + Y_1$, where Y_1 encodes a consistent set of updates produced by

rule r_1 , plus the updates in Y_1 that correspond to locations other than the locations updated by Y_2 .

$$\begin{aligned}
\mathbf{U6:} \quad & \text{upd}(\text{seq } r_1 \ r_2 \ \text{endseq}, X) \leftrightarrow (\text{upd}(r_1, X) \wedge \neg \text{conUSet}(X)) \vee \\
& (\text{isUSet}(X) \wedge \\
& \exists Y_1 Y_2 (\text{upd}(r_1, Y_1) \wedge \text{conUSet}(Y_1) \wedge [Y_1] \text{upd}(r_2, Y_2) \wedge \\
& \quad \forall z y_1 y_2 (X(z, y_1, y_2) \leftrightarrow ((Y_1(z, y_1, y_2) \wedge \forall x (\neg Y_2(z, y_1, x))) \vee Y_2(z, y_1, y_2))) \wedge \\
& \quad \forall z y_1 y_2 (X(z, y_1, y_2) \leftrightarrow ((Y_1(z, y_1, y_2) \wedge \forall x (\neg Y_2(z, y_1, x))) \vee Y_2(z, y_1, y_2))) \wedge \\
& \quad \forall z y_1 y_2 (X(z, y_1, y_2) \leftrightarrow ((Y_1(z, y_1, y_2) \wedge \forall x (\neg Y_2(z, y_1, x))) \vee Y_2(z, y_1, y_2))))))
\end{aligned}$$

- Our next axioms assert that X is an update set yielded by the rule $\text{let } (f, t) \rightarrow \rho \ \text{in } r \ \text{endlet}$ iff there is an update *multiset* Y yielded by the rule r that collapses into X , when the update values to the location (f, t) which appear in Y are aggregated using the location operator ρ , and the multiplicity of identical updates to a same location other than (f, t) is ignored. Since ρ -terms are algorithmic terms, f can either be a bridge or an algorithmic function symbol. Thus we have two possible cases.

U7.1: If f is an algorithmic function symbol in Υ_a then

$$\begin{aligned}
& \text{upd}(\text{let } (f, t) \rightarrow \rho \ \text{in } r \ \text{endlet}, X) \leftrightarrow \text{isUSet}(X) \wedge \exists Y (\text{upm}(r, Y) \wedge \\
& \quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \leftrightarrow (((\mathbf{x}_1 \neq c_f \vee t \neq \mathbf{x}_2) \wedge \exists \mathbf{z} (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{z}))) \vee \\
& \quad \quad (\mathbf{x}_1 = c_f \wedge \mathbf{x}_2 = t \wedge \mathbf{x}_3 = \rho_{\mathbf{y}}(\mathbf{y} | \exists \mathbf{z} (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}, \mathbf{z})))))) \wedge \\
& \quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \leftrightarrow \exists \mathbf{z} (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{z}))) \wedge \\
& \quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \leftrightarrow \exists \mathbf{z} (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{z}))))
\end{aligned}$$

U7.2: If f is a bridge function symbol in \mathcal{F}_b then

$$\begin{aligned}
& \text{upd}(\text{let } (f, t) \rightarrow \rho \ \text{in } r \ \text{endlet}, X) \leftrightarrow \text{isUSet}(X) \wedge \exists Y (\text{upm}(r, Y) \wedge \\
& \quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \leftrightarrow (((\mathbf{x}_1 \neq c_f \vee t \neq \mathbf{x}_2) \wedge \exists \mathbf{z} (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{z}))) \vee \\
& \quad \quad (\mathbf{x}_1 = c_f \wedge \mathbf{x}_2 = t \wedge \mathbf{x}_3 = \rho_{\mathbf{y}}(\mathbf{y} | \exists \mathbf{z} (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}, \mathbf{z})))))) \wedge \\
& \quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \leftrightarrow \exists \mathbf{z} (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{z}))) \wedge \\
& \quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \leftrightarrow \exists \mathbf{z} (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{z}))))
\end{aligned}$$

The following lemma is a direct consequence of Axioms **U1–U7**.

Lemma 7.1. *Each formula in the DB-ASM logic \mathcal{L}^{db} can be replaced by an equivalent formula not containing any subformulae of the form $\text{upd}(r, X)$.*

7.3. Update Multisets

Each DB-ASM rule is associated with a set of update multisets as defined in Fig. 9. The axioms presented in this section assert how an update multiset is yielded by a DB-ASM rule, i.e., they define the predicate $\text{upm}(r, X)$.

Same as in the axioms for update sets, we assume that if f is a dynamic function symbol in the given signature of meta-finite states $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$, then there is a corresponding constant (static nullary function) symbol $c_f \in \Upsilon_a$ as per Definition 6.1. We use \mathcal{F}_{dyn} to denote the set of all c_f such that f is a dynamic function symbol in Υ . Again, S denotes an arbitrary Henkin structure of signature Υ , $B = B_{db} \cup B_a$ denotes the base set (domain) of the database and algorithmic parts of S , and w.l.o.g. we assume $B_{db} \cap B_a = \emptyset$.

In the formulation of the axioms we use the predicate $\text{is}\ddot{\text{U}}\text{Set}(X)$ which is analogous to the predicate $\text{isUSet}(X)$ defined in the case of update sets. Let ζ be a valuation into S , $\text{is}\ddot{\text{U}}\text{Set}(X)$ expresses that $\zeta(X)$ represents an update multiset set for S . That is, for every tuple $(a_1, a_2, a_3, a_4) \in \zeta(X)$, we have that $a_1 = c_f^S$ for some dynamic function $f \in \Upsilon$, a_4 is an arbitrary value of B_a , and a_2, a_3 are values of the appropriate database or algorithmic base sets depending on whether f is a database, algorithmic or bridge function symbol.

$$\begin{aligned} \text{is}\ddot{\text{U}}\text{Set}(X) \equiv & \forall \mathbf{x}_1 x_2 x_3 \mathbf{x}_4 \left(X(\mathbf{x}_1, x_2, x_3, \mathbf{x}_4) \rightarrow \bigvee_{c_f \in \mathcal{F}_{dyn} \wedge f \in \Upsilon_{db}} \mathbf{x}_1 = c_f \right) \wedge \\ & \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 \left(X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \rightarrow \bigvee_{c_f \in \mathcal{F}_{dyn} \wedge f \in \Upsilon_a} \mathbf{x}_1 = c_f \right) \wedge \\ & \forall \mathbf{x}_1 x_2 \mathbf{x}_3 \mathbf{x}_4 \left(X(\mathbf{x}_1, x_2, \mathbf{x}_3, \mathbf{x}_4) \rightarrow \bigvee_{c_f \in \mathcal{F}_{dyn} \wedge f \in \mathcal{F}_b} \mathbf{x}_1 = c_f \right) \wedge \\ & \forall \mathbf{x}_1 \mathbf{x}_2 x_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, x_3, \mathbf{x}_4)) \wedge \forall \mathbf{x}_1 x_2 x_3 x_4 (\neg X(\mathbf{x}_1, x_2, x_3, x_4)) \wedge \\ & \forall x_1 x_2 x_3 \mathbf{x}_4 (\neg X(x_1, x_2, x_3, \mathbf{x}_4)) \wedge \forall x_1 \mathbf{x}_2 x_3 \mathbf{x}_4 (\neg X(x_1, \mathbf{x}_2, x_3, \mathbf{x}_4)) \wedge \\ & \forall x_1 x_2 x_3 x_4 (\neg X(x_1, x_2, x_3, x_4)) \wedge \forall \mathbf{x}_1 \mathbf{x}_2 x_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, x_3, \mathbf{x}_4)) \wedge \\ & \forall x_1 x_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(x_1, x_2, \mathbf{x}_3, \mathbf{x}_4)) \wedge \forall x_1 x_2 x_3 x_4 (\neg X(x_1, x_2, x_3, x_4)) \wedge \\ & \forall x_1 x_2 x_3 x_4 (\neg X(x_1, x_2, x_3, x_4)) \wedge \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)) \end{aligned}$$

The axioms for the predicate $\text{upm}(r, X)$ are analogous to the axioms for the predicate $\text{upd}(r, X)$, except for the fact that we need to deal with multisets represented as relations.

- Axioms $\ddot{\text{U}}1.1$ – $\ddot{\text{U}}1.3$ express that X represents an update multiset yielded by the assignment rule $f(t) := s$ iff it contains exactly one update with multiplicity 1, and that update is (f, t, s) .

$\ddot{\text{U}}1.1$: If f is a database function symbol in Υ_{db} then

$$\begin{aligned} \text{upm}(f(t) := s, X) \leftrightarrow & \text{is}\ddot{\text{U}}\text{Set}(X) \wedge \exists \mathbf{z} (X(c_f, t, s, \mathbf{z}) \wedge \\ & \forall \mathbf{x}_1 x_2 x_3 \mathbf{x}_4 (X(\mathbf{x}_1, x_2, x_3, \mathbf{x}_4) \rightarrow \mathbf{x}_1 = c_f \wedge x_2 = t \wedge x_3 = s \wedge \mathbf{x}_4 = \mathbf{z})) \wedge \\ & \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)) \wedge \forall x_1 x_2 x_3 x_4 (\neg X(x_1, x_2, x_3, x_4)) \end{aligned}$$

Ü1.2: If f is an algorithmic function symbol in Υ_a then

$$\begin{aligned} \text{upm}(f(t) := s, X) &\leftrightarrow \text{isÜSet}(X) \wedge \exists \mathbf{z} (X(c_f, t, s, \mathbf{z}) \wedge \\ &\quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \rightarrow \mathbf{x}_1 = c_f \wedge \mathbf{x}_2 = t \wedge \mathbf{x}_3 = s \wedge \mathbf{x}_4 = \mathbf{z})) \wedge \\ &\quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)) \wedge \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4))) \end{aligned}$$

Ü1.3: If f is a bridge function symbol in \mathcal{F}_b then

$$\begin{aligned} \text{upm}(f(t) := s, X) &\leftrightarrow \text{isÜSet}(X) \wedge \exists \mathbf{z} (X(c_f, t, s, \mathbf{z}) \wedge \\ &\quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \rightarrow \mathbf{x}_1 = c_f \wedge \mathbf{x}_2 = t \wedge \mathbf{x}_3 = s \wedge \mathbf{x}_4 = \mathbf{z})) \wedge \\ &\quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)) \wedge \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4))) \end{aligned}$$

- Axiom **Ü2** asserts that, if the formula φ evaluates to *true*, then X is an update multiset yielded by the conditional rule **if φ then r endif** iff X is an update multiset yielded by the rule r . Otherwise, the conditional rule yields only an empty update multiset.

Ü2: $\text{upm}(\text{if } \varphi \text{ then } r \text{ endif}, X) \leftrightarrow (\varphi \wedge \text{upm}(r, X)) \vee$

$$\begin{aligned} &(\neg \varphi \wedge \text{isÜSet}(X) \wedge \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4))) \wedge \\ &\quad \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)) \wedge \forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (\neg X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4))) \end{aligned}$$

- Axiom **Ü3** states that X is an update multiset yielded by the rule **forall x with φ do r enddo** iff X coincides with $\ddot{U}_{a_1} \uplus \dots \uplus \ddot{U}_{a_n}$, where $\{a_1, \dots, a_n\} = \{a_i \in B_{db} \mid \text{val}_{S, \zeta[x \mapsto a_i]}(\varphi) = \text{true}\}$ and \ddot{U}_{a_i} (for $1 \leq i \leq n$) is an update multiset yielded by the rule r under the variable assignment $\zeta[x \mapsto a_i]$. Note that the update multisets $\ddot{U}_{a_1}, \dots, \ddot{U}_{a_n}$ are encoded into the second-order variable Y of arity five. We use the informal expression “ F is a bijection from X to Y ” to denote that there is a bijection f from $\zeta(X)$ to $\zeta(Y)$ such that $F(a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4, b_5)$ iff $f((a_1, a_2, a_3, a_4)) = (b_1, b_2, b_3, b_4, b_5)$. It is a well known fact that F can be easily defined in first-order logic (see for instance [14]).

$$\begin{aligned}
\mathbf{\ddot{U}3}: \text{upm}(\mathbf{forall } x \text{ with } \varphi \text{ do } r \text{ enddo}, X) &\leftrightarrow \text{is}\ddot{\text{U}}\text{Set}(X) \wedge \\
&\exists Y F(\text{"}F \text{ is a bijection from } X \text{ to } Y\text{"} \wedge \\
&\quad \forall z y_1 y_2 y_3 z' y'_1 y'_2 y'_3 x (F(z, y_1, y_2, y_3, z', y'_1, y'_2, y'_3, x) \rightarrow \\
&\quad \quad (z = z' \wedge y_1 = y'_1 \wedge y_2 = y'_2)) \wedge \\
&\quad \forall z y_1 y_2 y_3 z' y'_1 y'_2 y'_3 x (F(z, y_1, y_2, y_3, z', y'_1, y'_2, y'_3, x) \rightarrow \\
&\quad \quad (z = z' \wedge y_1 = y'_1 \wedge y_2 = y'_2)) \wedge \\
&\quad \forall z y_1 y_2 y_3 z' y'_1 y'_2 y'_3 x (F(z, y_1, y_2, y_3, z', y'_1, y'_2, y'_3, x) \rightarrow \\
&\quad \quad (z = z' \wedge y_1 = y'_1 \wedge y_2 = y'_2)) \wedge \\
&\quad \forall x ((\varphi \rightarrow \exists Z (\text{upm}(r, Z) \wedge \\
&\quad \quad \forall z y_1 y_2 y_3 (Z(z, y_1, y_2, y_3) \leftrightarrow Y(z, y_1, y_2, y_3, x)) \wedge \\
&\quad \quad \forall z y_1 y_2 y_3 (Z(z, y_1, y_2, y_3) \leftrightarrow Y(z, y_1, y_2, y_3, x)) \wedge \\
&\quad \quad \forall z y_1 y_2 y_3 (Z(z, y_1, y_2, y_3) \leftrightarrow Y(z, y_1, y_2, y_3, x)))) \wedge \\
&\quad (\neg \varphi \rightarrow \forall z y_1 y_2 y_3 (\neg Y(z, y_1, y_2, y_3, x)) \wedge \\
&\quad \quad \forall z y_1 y_2 y_3 (\neg Y(z, y_1, y_2, y_3, x)) \wedge \\
&\quad \quad \forall z y_1 y_2 y_3 (\neg Y(z, y_1, y_2, y_3, x))))))
\end{aligned}$$

- Axiom $\mathbf{\ddot{U}4}$ states that X represents an update multiset yielded by the rule **par** r_1 r_2 **endpar** iff X represents an update multiset $\ddot{U}_1 \uplus \ddot{U}_2$ where \ddot{U}_1 is an update multiset yielded by r_1 and \ddot{U}_2 is an update multiset yielded by r_2 .

$$\begin{aligned}
\mathbf{\ddot{U}4}: \text{upm}(\mathbf{par } r_1 \ r_2 \ \mathbf{endpar}, X) &\leftrightarrow \text{is}\ddot{\text{U}}\text{Set}(X) \wedge \\
&\exists Y_1 Y_2 (\text{upm}(r_1, Y_1) \wedge \text{upm}(r_2, Y_2) \wedge \\
&\quad \forall z y_1 y_2 y_3 z' y'_1 y'_2 y'_3 (Y_1(z, y_1, y_2, y_3) \wedge Y_2(z', y'_1, y'_2, y'_3) \rightarrow y_3 \neq y'_3) \wedge \\
&\quad \forall z y_1 y_2 y_3 z' y'_1 y'_2 y'_3 (Y_1(z, y_1, y_2, y_3) \wedge Y_2(z', y'_1, y'_2, y'_3) \rightarrow y_3 \neq y'_3) \wedge \\
&\quad \forall z y_1 y_2 y_3 z' y'_1 y'_2 y'_3 (Y_1(z, y_1, y_2, y_3) \wedge Y_2(z', y'_1, y'_2, y'_3) \rightarrow y_3 \neq y'_3) \wedge \\
&\quad \forall z y_1 y_2 y_3 (X(z, y_1, y_2, y_3) \leftrightarrow (Y_1(z, y_1, y_2, y_3) \vee Y_2(z, y_1, y_2, y_3))) \wedge \\
&\quad \forall z y_1 y_2 y_3 (X(z, y_1, y_2, y_3) \leftrightarrow (Y_1(z, y_1, y_2, y_3) \vee Y_2(z, y_1, y_2, y_3))) \wedge \\
&\quad \forall z y_1 y_2 y_3 (X(z, y_1, y_2, y_3) \leftrightarrow (Y_1(z, y_1, y_2, y_3) \vee Y_2(z, y_1, y_2, y_3))))
\end{aligned}$$

- Axiom $\mathbf{\ddot{U}5}$ asserts that X is an update multiset yielded by the rule **choose** x **with** φ **do** r **enddo** iff it is an update multiset yielded by the rule r under a variable assignment $\zeta[x \mapsto a]$ which satisfies φ .

$$\mathbf{\ddot{U}5}: \text{upm}(\mathbf{choose } x \text{ with } \varphi \text{ do } r \text{ enddo}, X) \leftrightarrow \exists x (\varphi \wedge \text{upm}(r, X))$$

- Axiom $\mathbf{\ddot{U}6}$ asserts that X is an update multiset yielded by a sequence rule **seq** r_1 r_2 **endseq** iff it corresponds either to an inconsistent update multiset Y_1 yielded by

rule r_1 , or to an update multiset formed by the updates in an update multiset Y_2 yielded by rule r_2 in a successor state $S + U$, where U is the set of updates which appear in the multiset Y_1 , plus the updates in Y_1 that correspond to locations other than the locations that appear in the updates in Y_2 .

Ü6: $\text{upm}(\text{seq } r_1 \ r_2 \ \text{endseq}, X) \leftrightarrow (\text{upm}(r_1, X) \wedge \neg \text{conÜSet}(X)) \vee$

$(\text{isÜSet}(X) \wedge$

$\exists Y_1 Y_1' Y_2 (\text{upm}(r_1, Y_1) \wedge \text{conÜSet}(Y_1) \wedge \text{isÜSet}(Y_1') \wedge [Y_1'] \text{upm}(r_2, Y_2) \wedge$

$\forall z y_1 y_2 (Y_1'(z, y_1, y_2) \leftrightarrow \exists y_3 (Y_1(z, y_1, y_2, y_3))) \wedge$

$\forall z y_1 y_2 (Y_1'(z, y_1, y_2) \leftrightarrow \exists y_3 (Y_1(z, y_1, y_2, y_3))) \wedge$

$\forall z y_1 y_2 (Y_1'(z, y_1, y_2) \leftrightarrow \exists y_3 (Y_1(z, y_1, y_2, y_3))) \wedge$

$\forall z y_1 y_2 y_3 (X(z, y_1, y_2, y_3) \leftrightarrow ((Y_1(z, y_1, y_2, y_3) \wedge \forall x_1 x_2 (\neg Y_2(z, y_1, x_1, x_2))) \vee$
 $Y_2(z, y_1, y_2, y_3))) \wedge$

$\forall z y_1 y_2 y_3 (X(z, y_1, y_2, y_3) \leftrightarrow ((Y_1(z, y_1, y_2, y_3) \wedge \forall x_1 x_2 (\neg Y_2(z, y_1, x_1, x_2))) \vee$
 $Y_2(z, y_1, y_2, y_3))) \wedge$

$\forall z y_1 y_2 y_3 (X(z, y_1, y_2, y_3) \leftrightarrow ((Y_1(z, y_1, y_2, y_3) \wedge \forall x_1 x_2 (\neg Y_2(z, y_1, x_1, x_2))) \vee$
 $Y_2(z, y_1, y_2, y_3))))))$

- Our next axioms assert that X is an update multiset yielded by a let rule $\text{let } (f, t) \rightarrow \rho \text{ in } r \ \text{endlet}$ iff it corresponds to an update multiset Y yielded by the rule r except for the updates to the location (f, t) which are collapsed into a unique update in X by aggregating their values using the operator ρ . Again notice that a ρ -term is an algorithmic term and thus we have to consider only two cases.

Ü7.1: If f is an algorithmic function symbol in Υ_a then

$\text{upm}(\text{let } (f, t) \rightarrow \rho \text{ in } r \ \text{endlet}, X) \leftrightarrow \text{isÜSet}(X) \wedge \exists Y \mathbf{z} (\text{upm}(r, Y) \wedge$

$\forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \leftrightarrow (((\mathbf{x}_1 \neq c_f \vee t \neq \mathbf{x}_2) \wedge Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)) \vee$

$(\mathbf{x}_1 = c_f \wedge \mathbf{x}_2 = t \wedge \mathbf{x}_3 = \rho_{\mathbf{y}}(\mathbf{y} | \exists \mathbf{x}_0 (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}, \mathbf{x}_0))) \wedge \mathbf{x}_4 = \mathbf{z}))) \wedge$

$\forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \leftrightarrow Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)) \wedge$

$\forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \leftrightarrow Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)))$

Ü7.2: If f is a bridge function symbol in \mathcal{F}_b then

$\text{upm}(\text{let } (f, t) \rightarrow \rho \text{ in } r \ \text{endlet}, X) \leftrightarrow \text{isÜSet}(X) \wedge \exists Y \mathbf{z} (\text{upm}(r, Y) \wedge$

$\forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \leftrightarrow (((\mathbf{x}_1 \neq c_f \vee t \neq \mathbf{x}_2) \wedge Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)) \vee$

$(\mathbf{x}_1 = c_f \wedge \mathbf{x}_2 = t \wedge \mathbf{x}_3 = \rho_{\mathbf{y}}(\mathbf{y} | \exists \mathbf{x}_0 (Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}, \mathbf{x}_0))) \wedge \mathbf{x}_4 = \mathbf{z}))) \wedge$

$\forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \leftrightarrow Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)) \wedge$

$\forall \mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_4 (X(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) \leftrightarrow Y(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)))$

Analogous to Lemma 7.1, the following result is a straightforward consequence of Axioms **Ü1–Ü7**.

Lemma 7.2. *Each formula in the DB-ASM logic \mathcal{L}^{db} can be replaced by an equivalent formula not containing any subformulae of the form $\text{upm}(r, X)$.*

7.4. Axioms and Inference Rules

We present a set of axioms and inference rules which constitute a proof system for the logic \mathcal{L}^{db} for DB-ASMs. A good starting point is the natural formalism L_2 as defined in [27] for the relational variant of second-order logic on which \mathcal{L}^{db} is based. L_2 uses the usual axioms and rules for first-order logic, with quantifier rules applying to second-order variables as well as first-order variables, and with the stipulation that the range of the second-order variables includes *at least* all the relations definable by the formulae of the language.

A deductive calculus for L_2 is obtained by augmenting the inference rules and axioms of first-order logic with the comprehension axiom schema **SO-C** (which is a form of the Comprehension Principle of Set Theory), and with the axiom schema of universal instantiation **SO-UI** and the inference rule of universal generalization **SO-UG** for second-order variables.

SO-C $\exists X \forall v_1, \dots, v_k (X(v_1, \dots, v_k) \leftrightarrow \varphi)$, where $k \geq 1$, v_1, \dots, v_k are first-order variables from $\mathcal{X}_{db} \cup \mathcal{X}_a$, and X is a k -ary second-order variable which does not occur free in the formula φ .

SO-UI $\forall X (\varphi) \rightarrow \varphi[Y/X]$, provided the arity of X and Y coincides.

SO-UG $\frac{\psi \rightarrow \varphi[Y/X]}{\psi \rightarrow \forall X (\varphi)}$, provided Y is not free in ψ .

The axioms and rules of L_2 together with the axioms for update sets and multisets form the basis of the proposed proof system for the logic \mathcal{L}^{db} for DB-ASMs. The complete list of axioms and rules is composed by:

- The Axioms **SO-C**, **SO-UI** and **SO-UG** of the deductive calculus L_2 .
- The Axioms **U1–U7** in Section 7.2 which assert the properties of $\text{upd}(r, X)$.
- The axioms **Ü1–Ü7** in Section 7.3 which assert the properties of $\text{upm}(r, X)$.
- Axiom **M1** and Rules **M2–M3** from the axiom system K of modal logic, which is the weakest normal modal logic system [25]. Axiom **M1** is called *Distribution Axiom* of K, Rule **M2** is called *Necessitation Rule* of K and Rule **M3** is the inference rule called *Modus Ponens* in the classical logic. By using these axioms and rules together, we are able to derive all modal properties that are valid in Kripke frames.

$$\mathbf{M1} \quad [X](\varphi \rightarrow \psi) \rightarrow ([X]\varphi \rightarrow [X]\psi)$$

$$\mathbf{M2} \quad \frac{\varphi}{[X]\varphi}$$

$$\mathbf{M3} \quad \frac{\varphi, \varphi \rightarrow \psi}{\psi}$$

- Axiom **M4** asserts that, if an update set X is not consistent, then there is no successor state obtained after applying X over the current state and thus $[X]\varphi$ is interpreted as true for any formula φ . As applying a consistent update set X over the current state is deterministic, Axiom **M5** describes the deterministic accessibility relation in terms of $[X]$.

$$\mathbf{M4} \quad \neg \text{conUSet}(X) \rightarrow [X]\varphi$$

$$\mathbf{M5} \quad \neg [X]\varphi \rightarrow [X]\neg\varphi$$

- Axiom **M6** is called *Barcan Axiom*. It originates from the fact that all states in a run of a DB-ASM have the same base set, and thus the quantifiers in all states always range over the same set of elements.

$$\mathbf{M6} \quad \forall v([X]\varphi \rightarrow [X]\forall v(\varphi)), \text{ where } v \text{ stands for any first-order variable in } \mathcal{X}_{db} \cup \mathcal{X}_a \text{ or any second-order variable.}$$

- Axioms **M7** and **M8** assert that the interpretation of static and pure formulae is the same in all states of a DB-ASM, which is not affected by the execution of any DB-ASM rule r .

$$\mathbf{M7} \quad \varphi \wedge \text{upd}(r, X) \rightarrow [X]\varphi, \text{ for static and pure } \varphi$$

$$\mathbf{M8} \quad \text{con}(r, X) \wedge [X]\varphi \rightarrow \varphi, \text{ for static and pure } \varphi$$

- Axioms **A1.1–A1.3** assert that, if a consistent update set X does not contain any update to a given location, then the content of that location in a successor state obtained after applying X remains unchanged. Axioms **A2.1–A2.3** assert that, if a consistent update set X contains an update (f, a, b) , then the content of the location (f, a) in the successor state obtained after applying X is equal to b . Axiom **A3** says that, if a DB-ASM rule r yields an update multiset, then the rule r also yields an update set.

A1.1 If f is a database function symbol,

$$\text{conUSet}(X) \wedge \forall z(\neg X(c_f, x, z)) \wedge f(x) = y \rightarrow [X]f(x) = y$$

A1.2 If f is an algorithmic function symbol,

$$\text{conUSet}(X) \wedge \forall \mathbf{z}(\neg X(c_f, \mathbf{x}, \mathbf{z})) \wedge f(\mathbf{x}) = \mathbf{y} \rightarrow [X]f(\mathbf{x}) = \mathbf{y}$$

A1.3 If f is a bridge function symbol,
 $\text{conUSet}(X) \wedge \forall \mathbf{z}(\neg X(c_f, x, \mathbf{z})) \wedge f(x) = \mathbf{y} \rightarrow [X]f(x) = \mathbf{y}$

A2.1 If f is a database function symbol,
 $\text{conUSet}(X) \wedge X(c_f, x, y) \rightarrow [X]f(x) = y$

A2.2 If f is an algorithmic function symbol,
 $\text{conUSet}(X) \wedge X(c_f, \mathbf{x}, \mathbf{y}) \rightarrow [X]f(\mathbf{x}) = \mathbf{y}$

A2.3 If f is a bridge function symbol,
 $\text{conUSet}(X) \wedge X(c_f, x, y) \rightarrow [X]f(x) = y$

A3 $\text{upm}(r, X) \rightarrow \exists Y(\text{upd}(r, Y))$

- The following are axiom schemes from first-order logic.

P1 $\varphi \rightarrow (\psi \rightarrow \varphi)$

P2 $(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$

P3 $(\neg\varphi \rightarrow \neg\psi) \rightarrow (\psi \rightarrow \varphi)$

- The standard axiom of universal instantiation **UI** of the classical first-order calculus needs to be restricted to static terms (which do not contain dynamic function names). Otherwise, if we substitute a term t for a variable x , then t can be evaluated in different states due to sequential composition of transition rules. The rule of universal generalization **UG** is the same as in the classical first-order calculus and applies to both types of first-order variables. An analogous axiom and inference rule can be added for \exists . This however is not necessary since in this paper \exists is viewed as an abbreviation of $\neg\forall\neg$.

UI $\forall v(\varphi(v)) \rightarrow \varphi[t/v]$ if φ is pure or t is static, t is a database term or an algorithmic term depending on whether v is a first-order variable in \mathcal{X}_{db} or \mathcal{X}_a , respectively, and t is free for v in $\varphi(v)$.

UG $\frac{\psi \rightarrow \varphi[v'/v]}{\psi \rightarrow \forall v(\varphi)}$ if v and v' are first-order variables of a same type, i.e., both belong to \mathcal{X}_{db} or both belong to \mathcal{X}_a , and v' is not free in ψ .

- The following are the equality axioms adapted from first-order logic with equality. Axiom **EQ1** asserts the reflexivity property, Axiom **EQ2** asserts the substitutions for functions, Axiom **EQ3** asserts the substitutions for second-order variables, and Axiom **EQ4** asserts the substitutions for ρ -terms. Again, terms occurring in the axioms are restricted to be static, which do not contain any dynamic function symbols.

EQ1 $t = t$ for static term t .

EQ2 $t_1 = t_{n+1} \wedge \dots \wedge t_n = t_{2n} \rightarrow f(t_1, \dots, t_n) = f(t_{n+1}, \dots, t_{2n})$ for any function f and static terms t_i ($i = 1, \dots, 2n$).

EQ3 $t_1 = t_{n+1} \wedge \dots \wedge t_n = t_{2n} \rightarrow (X(t_1, \dots, t_n) \leftrightarrow X(t_{n+1}, \dots, t_{2n}))$ for any second-order variable X and static terms t_i ($i = 1, \dots, 2n$).

EQ4 $t_1 = t_2 \wedge (\varphi_1 \leftrightarrow \varphi_2) \rightarrow \rho_v(t_1|\varphi_1) = \rho_v(t_2|\varphi_2)$ for pure formulae φ_1 and φ_2 , static terms t_1 and t_2 , and v a first-order variable.

- The following axiom is taken from dynamic logic. It asserts that that executing a sequence rule is equivalent to executing its sub-rules sequentially.

$$\mathbf{DY1} \quad \exists X(\text{upd}(\text{seq } r_1 \ r_2 \ \text{endseq}, X) \wedge [X]\varphi) \leftrightarrow \exists X_1(\text{upd}(r_1, X_1) \wedge [X_1]\exists X_2(\text{upd}(r_2, X_2) \wedge [X_2]\varphi))$$

- Axiom **E** is the extensionality axiom. Recall that $r_1 \equiv r_2$ if for every Henkin meta-finite structure S it holds that $S \models \forall X(\text{upd}(r_1, X) \leftrightarrow \text{upd}(r_2, X))$ (see Definition 7.2).

$$\mathbf{E} \quad r_1 \equiv r_2 \rightarrow (\exists X_1(\text{upd}(r_1, X_1) \wedge [X_1]\varphi) \leftrightarrow \exists X_2(\text{upd}(r_2, X_2) \wedge [X_2]\varphi))$$

The following soundness theorem for the proof system is relatively straightforward, since the non-standard axioms and rules are just a formalisation of the definitions of the semantics of rules, update sets and update multisets.

Theorem 7.3. *Let φ be a formula and let Φ be a set of formulae in the logic \mathcal{L}^{db} for DB-ASMs. If $\Phi \vdash_{L_2} \varphi$, then $\Phi \models \varphi$.*

8. Derivation

In this section we present some properties of the logic for DB-ASMs which are implied by the axioms and rules from the previous section. This includes some properties known for the logic for ASMs [39]. In particular, the logic for ASMs uses the modal expressions $[r]\varphi$ and $\langle r \rangle \varphi$ with the following semantics:

- $\llbracket [r]\varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } \llbracket \varphi \rrbracket_{S+U,\zeta} = true \text{ for all consistent } U \in \Delta(r, S, \zeta), \\ false & \text{otherwise} \end{cases}$
- $\llbracket \langle r \rangle \varphi \rrbracket_{S,\zeta} = \begin{cases} true & \text{if } \llbracket \varphi \rrbracket_{S+U,\zeta} = true \text{ for at least one consistent} \\ & U \in \Delta(r, S, \zeta), \\ false & \text{otherwise} \end{cases}$

Instead of introducing modal operators $[]$ and $\langle \rangle$ for a DB-ASM rule r , we use the modal expression $[X]\varphi$ for an update set yielded by a possibly non-deterministic rule. The modal expressions $[r]\varphi$ and $\langle r \rangle\varphi$ in the logic for ASMs can be treated as the shortcuts for the following formulae in our logic.

$$[r]\varphi \equiv \forall X(\text{upd}(r, X) \rightarrow [X]\varphi). \quad (5)$$

$$\langle r \rangle\varphi \equiv \exists X(\text{upd}(r, X) \wedge [X]\varphi). \quad (6)$$

Lemma 8.1. *The following axioms and rules used in the logic for ASMs are derivable in the logic for DB-ASMs, where the rule r in Axioms (c) and (d) is assumed to be defined and deterministic.*

- (a) $([r](\varphi \rightarrow \psi) \wedge [r]\varphi) \rightarrow [r]\psi$
- (b) $\varphi \rightarrow [r]\varphi$, for static and pure φ .
- (c) $\neg \text{wcon}(r) \rightarrow [r]\varphi$
- (d) $[r]\varphi \leftrightarrow \neg[r]\neg\varphi$

Proof. We can prove them as follows:

- (a): By Equation 5, we have that $[r](\varphi \rightarrow \psi) \wedge [r]\varphi \equiv \forall X(\text{upd}(r, X) \rightarrow [X](\varphi \rightarrow \psi)) \wedge \forall X(\text{upd}(r, X) \rightarrow [X]\varphi)$. By the axioms from classical logic, this is in turn equivalent to $\forall X(\text{upd}(r, X) \rightarrow ([X](\varphi \rightarrow \psi) \wedge [X]\varphi))$. Then by Axiom **M1**, we get $\forall X(\text{upd}(r, X) \rightarrow ([X]\varphi \rightarrow [X]\psi) \wedge [X]\varphi)$. Finally, by rule **M3** (Modus Ponens), we derive $\forall X(\text{upd}(r, X) \rightarrow [X]\psi)$, which by Equation 5 is equivalent to $[r]\psi$ in the logic for ASMs.
- (b): By Rule **M7**, we have that $\varphi \rightarrow \text{upd}(r, X) \wedge [X]\varphi$, for static and pure φ . By the universal generalization rule for second-order variables (**SO-UG**), we obtain $\varphi \rightarrow \forall X(\text{upd}(r, X) \wedge [X]\varphi)$. Finally, Equation 5 gives us $\varphi \rightarrow [r]\varphi$ for static and pure φ .
- (c): By Equation 3, we have $\neg \text{wcon}(r) \leftrightarrow \neg \exists X(\text{con}(r, X))$. In turn, by Equation 1, we get $\neg \text{wcon}(r) \leftrightarrow \neg \exists X(\text{upd}(r, X) \wedge \text{conUSet}(X))$. Since a rule r in the logic for ASMs is deterministic, we get $\neg \text{wcon}(r) \leftrightarrow \neg \text{conUSet}(X)$. By Axiom **M4**, we get $\neg \text{wcon}(r) \rightarrow [r]\varphi$.
- (d): By Equation 5, we have $\neg[r]\neg\varphi \equiv \exists X(\text{upd}(r, X) \wedge \neg[X]\neg\varphi)$. By applying Axiom **M5** to $\neg[X]\neg\varphi$, we get $\neg[r]\neg\varphi \equiv \exists X(\text{upd}(r, X) \wedge [X]\varphi)$. When the rule r is deterministic, the interpretation of $\forall X(\text{upd}(r, X) \rightarrow [X]\varphi)$ coincides the interpretation of $\exists X(\text{upd}(r, X) \wedge [X]\varphi)$ and therefore $[r]\varphi \leftrightarrow \neg[r]\neg\varphi$.

□

The logic for ASMs introduced in [39] is deterministic, i.e., it excludes nondeterministic choice rules. In contrast, our logic for DB-ASMs includes a nondeterministic choice rule. Note that the formula $\text{Con}(R)$ in **Axiom 5** in [39] (i.e., in $\neg\text{Con}(R) \rightarrow [R]\varphi$) corresponds to the weak version of consistency (i.e., $\text{wcon}(r)$) in the context of our logic for DB-ASMs.

Lemma 8.2. *The following properties are derivable in the logic for DB-ASMs.*

- (e) $\text{con}(r, X) \wedge [X]f(v_1) = v_2 \rightarrow X(c_f, v_1, v_2) \vee (\forall v_3(\neg X(c_f, v_1, v_3)) \wedge f(v_1) = v_2)$, where $v_1, v_2, v_3 \in \mathcal{X}_{db}$ if f is a database function symbol, $v_1, v_2, v_3 \in \mathcal{X}_a$ if f is an algorithmic function symbol, and $v_1 \in \mathcal{X}_{db}$ and $v_2, v_3 \in \mathcal{X}_a$ if f is a bridge function symbol
- (f) $\text{con}(r, X) \wedge [X]\varphi \rightarrow \neg[X]\neg\varphi$
- (g) $[X]\exists v(\varphi) \rightarrow \exists v([X]\varphi)$, where $v \in \mathcal{X}_{db} \cup \mathcal{X}_a$ is a first-order variable.
- (h) $[X]\varphi_1 \wedge [X]\varphi_2 \rightarrow [X](\varphi_1 \wedge \varphi_2)$

Proof. (e) is derivable by applying Axioms **A1** and **A2**. (f) is a straightforward result of Axiom **M5**. (g) can be derived by applying Axioms **M5** and **M6**. Regarding (h), it is derivable by using Axioms **M1-M3**. □

Lemma 8.3. *For arbitrary terms t, s and first-order variables v_1, v_2 of the appropriate type (depending on whether f is a database, algorithmic or bridge function symbol), the following properties in [19] are derivable in the logic for DB-ASMs.*

- $v_1 = t \rightarrow (v_2 = s \leftrightarrow [f(t) := s]f(v_1) = v_2)$
- $v_1 \neq t \rightarrow (v_2 = f(v_1) \leftrightarrow [f(t) := s]f(v_1) = v_2)$

In DB-ASMs, two parallel computations may produce an update multiset, in which there are identical updates to a location assigned with a location operator. Without an outer **let** rule, the rule **par** r **endpar** could be simplified to r . This however is no longer the case if we consider update multisets.

Example 8.1. In the DB-ASM rule below, **SUM** is a location operator assigned to the location $(\text{TNUM},())$. Two identical updates (i.e., $(\text{TNUM},(),1)$ and $(\text{TNUM},(),1)$) are first generated in an update multiset, and then aggregated into one update $(\text{TNUM},(),2)$ in an update set.

```

let (TNUM, ())  $\rightarrow$  SUM in
  par
    TNUM := 1
    TNUM := 1
  endpar
endlet

```

The update multiset is collapsed into the update set $\{(TNUM, ()), 2\}$, whereas without the **let** rule we would obtain $\{(TNUM, ()), 1\}$.

Following the approach of defining the predicate joinable in [39], we define the predicate joinable over two DB-ASM rules. As DB-ASM rules are allowed to be nondeterministic, the predicate $\text{joinable}(r_1, r_2)$ means that there exists a pair of update sets without conflicting updates, which are yielded by rules r_1 and r_2 , respectively. Then, based on the use of predicate joinable, the properties in Lemma 8.4 are all derivable.

$$\begin{aligned}
\text{joinable}(r_1, r_2) \equiv & \exists X_1 X_2 (\text{upd}(r_1, X_1) \wedge \text{upd}(r_2, X_2) \wedge \\
& \bigwedge_{c_f \in \mathcal{F}_{dyn} \wedge f \in \Upsilon_{db}} \forall xyz (X_1(c_f, x, y) \wedge X_2(c_f, x, z) \rightarrow y = z) \wedge \\
& \bigwedge_{c_f \in \mathcal{F}_{dyn} \wedge f \in \Upsilon_a} \forall xyz (X_1(c_f, \mathbf{x}, \mathbf{y}) \wedge X_2(c_f, \mathbf{x}, \mathbf{z}) \rightarrow \mathbf{y} = \mathbf{z}) \wedge \\
& \bigwedge_{c_f \in \mathcal{F}_{dyn} \wedge f \in \mathcal{F}_b} \forall xyz (X_1(c_f, x, \mathbf{y}) \wedge X_2(c_f, x, \mathbf{z}) \rightarrow \mathbf{y} = \mathbf{z}))
\end{aligned} \tag{7}$$

Lemma 8.4. *The following properties for weak consistency are derivable in the logic of DB-ASMs.*

- (i) $\text{wcon}(f(t) := s)$
- (j) $\text{wcon}(\mathbf{if} \varphi \mathbf{then} r \mathbf{endif}) \leftrightarrow \neg \varphi \vee (\varphi \wedge \text{wcon}(r))$
- (k) $\text{wcon}(\mathbf{forall} x \mathbf{with} \varphi \mathbf{do} r \mathbf{enddo}) \leftrightarrow \forall x (\varphi \rightarrow \text{wcon}(r) \wedge \forall y (\varphi[y/x] \rightarrow \text{joinable}(r, r[y/x])))$
- (l) $\text{wcon}(\mathbf{par} r_1 r_2 \mathbf{endpar}) \leftrightarrow \text{wcon}(r_1) \wedge \text{wcon}(r_2) \wedge \text{joinable}(r_1, r_2)$
- (m) $\text{wcon}(\mathbf{choose} x \mathbf{with} \varphi \mathbf{do} r \mathbf{enddo}) \leftrightarrow \exists x (\varphi \wedge \text{wcon}(r))$
- (n) $\text{wcon}(\mathbf{seq} r_1 r_2 \mathbf{endseq}) \leftrightarrow \exists X (\text{con}(r_1, X) \wedge [X] \text{wcon}(r_2))$
- (o) *If f is a bridge function symbol:*
 $\text{wcon}(\mathbf{let} (f, t) \rightarrow \rho \mathbf{in} r \mathbf{endlet}) \leftrightarrow \exists XY (\text{upd}(r, X) \wedge \text{conUSet}(Y) \wedge \forall xy (Y(c_f, x, \mathbf{y}) \leftrightarrow (t = x \vee X(c_f, x, \mathbf{y}))) \wedge \forall zxy (X(\mathbf{z}, x, y) \leftrightarrow Y(\mathbf{z}, x, y)) \wedge \forall zxy (X(\mathbf{z}, \mathbf{x}, \mathbf{y}) \leftrightarrow Y(\mathbf{z}, \mathbf{x}, \mathbf{y})))$

We omit the proof of the previous lemma as well as the proof of the remaining lemmas in this section, since they are lengthy but relatively easy exercises. Furthermore, most of them are similar to the proofs of the analogous results in Nanchen's thesis [28].

Lemma 8.5. *The following properties for the formula $[r]\varphi$ are derivable in the logic for DB-ASMs.*

$$(p) \text{ [if } \varphi \text{ then } r \text{ endif]} \psi \leftrightarrow (\varphi \wedge [r]\psi) \vee (\neg\varphi \wedge \psi)$$

$$(q) \text{ [choose } x \text{ with } \varphi \text{ do } r \text{ enddo]} \psi \leftrightarrow \forall x(\varphi \rightarrow [r]\psi)$$

Lemma 8.6 states that a parallel composition is commutative and associative while a sequential composition is associative.

Lemma 8.6. *The following properties for parallel and sequential compositions are derivable in the logic for DB-ASMs.*

$$(r) \text{ par } r_1 \ r_2 \ \text{endpar} \equiv \text{par } r_2 \ r_1 \ \text{endpar}$$

$$(s) \text{ par } (\text{par } r_1 \ r_2 \ \text{endpar}) \ r_3 \ \text{endpar} \equiv \text{par } r_1 \ (\text{par } r_2 \ r_3 \ \text{endpar}) \ \text{endpar}$$

$$(t) \text{ seq } (\text{seq } r_1 \ r_2 \ \text{endseq}) \ r_3 \ \text{endseq} \equiv \text{seq } r_1 \ (\text{seq } r_2 \ r_3 \ \text{endseq}) \ \text{endseq}$$

Lemma 8.7. *The extensionality axiom for transition rules in the logic for ASMs [39] is derivable in the logic for DB-ASMs.*

$$(u) r_1 \equiv r_2 \rightarrow ([r_1]\varphi \leftrightarrow [r_2]\varphi)$$

9. Completeness

In this section we prove the completeness of the proof system of the logic \mathcal{L}^{db} for DB-ASMs which we introduced in the previous section.

In the following, S denotes an arbitrary Henkin meta-finite structure of signature (of meta-finite states) $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b$ (recall Definition 6.7. As before, $B = B_{db} \cup B_a$ denotes the base set (domain) of individual of S , where B_{db} and B_a are the base sets of the database and algorithmic parts, respectively, and D_n the universe of n -ary relations.

Clearly, we cannot axiomatize an arbitrary set $\Lambda = \{\rho^1, \dots, \rho^m\}$ of location operators. Note that even if we just take a simple location operator such as PRODUCT and axiomatize it, that leads us outside linear arithmetic and thus to an incomplete theory. As a compromise solution for this problem, we treat location (multiset) operators as standard non-axiomatized functions as follows.

Definition 9.1. We assume that Υ further includes a subset $\Upsilon_\Lambda = \{f_{\rho^1}, \dots, f_{\rho^m}\}$ of static function symbols, where each f_{ρ^i} is interpreted in S by a corresponding function $f_{\rho^i}^S : D_2 \rightarrow B_a$ defined as follows:

$$f_{\rho^i}^S(A) = \begin{cases} \rho^i(\{\{a \mid (a, b) \in A\}\}) & \text{if } \{\{a \mid (a, b) \in A\}\} \in \text{dom}(\rho_i) \\ \text{undef} & \text{otherwise.} \end{cases}$$

We then assume that the formulae of \mathcal{L}^{db} do not include any ρ -term of the form $\rho_v(t|\varphi)$. This does not affect the expressive power of \mathcal{L}^{db} since every formula φ can be translated (under the assumption made in Definition 9.1) into an equivalent formula φ' which does not use any ρ -term. We can proceed as follows. Let $\rho_{v_1}^1(t_1|\psi_1), \dots, \rho_{v_n}^n(t_n|\psi_n)$ be the ρ -terms which appear in an atomic sub-formula α of φ . Let α' be the following formula:

$$\forall X_1 \dots X_n v_1 \dots v_n \mathbf{z}_1 \dots \mathbf{z}_n \left(\bigwedge_{1 \leq i \leq n} (X_i(\mathbf{z}_i, v_i) \leftrightarrow (\psi'_i \wedge (t_i = \mathbf{z}_i)')) \rightarrow \alpha'' \right),$$

where $X_1, \dots, X_n, v_1, \dots, v_n, \mathbf{z}_1, \dots, \mathbf{z}_n$ are variables which do not appear free in α , ψ'_i and $(t_i = \mathbf{z}_i)'$ are obtained by recursively applying this procedure to every atomic sub-formula of ψ_i and to $t_i = \mathbf{z}_i$, respectively, and α'' is obtained by replacing $\rho_{v_1}^1(t_1|\psi_1), \dots, \rho_{v_n}^n(t_n|\psi_n)$ in α by $f_{\rho^1}(X_1), \dots, f_{\rho^n}(X_n)$, respectively. Then the formula φ' can be defined as the formula obtained by replacing every atomic sub-formula α of φ by α' .

We can now proceed with proving the completeness of \mathcal{L}^{db} . The strategy is to show that \mathcal{L}^{db} (with ρ -terms conveniently replaced by functions as explained above) is a syntactic variant of a complete first-order theory of types.

Let $\Upsilon = \Upsilon_{db} \cup \Upsilon_a \cup \mathcal{F}_b \cup \Upsilon_\Lambda$ be a signature of Henkin meta-finite structures. Assume w.l.o.g. that Υ_{db} , Υ_a , \mathcal{F}_b and Υ_Λ are pairwise disjoint. Let Υ^T be the signature formed by:

- The function symbols of Υ .
- Unary relations T_{db} and T_a .
- For each $n \geq 1$ a 1-ary relation symbol T_n .
- For each $n \geq 1$ a $(n + 1)$ -ary relation symbol E_n .

$T_{db}(x)$ and $T_a(x)$ are intended to state that x is an individual belonging to the database part and to the algorithmic part, respectively. Likewise, $T_n(x)$ is intended to state that x is a relation of arity n . Finally, $T_n(y_1, \dots, y_n, x)$ is intended to state that the tuple (y_1, \dots, y_n) belongs to the relation x .

A Henkin meta-finite structure S of signature Υ *determines* a unique first-order structure S' of vocabulary Υ^T as follows:

- The domain of S' is $\text{dom}(S') = B_{db} \cup B_a \cup \bigcup_{n \geq 1} D_n$, where B_{db} and B_a denote the base sets of the database and algorithmic parts of S , respectively, and D_n denotes the universe of n -ary relations of S .

- The interpretation in S' of the function symbols in $\Upsilon^T \cap (\Upsilon_{db} \cup \mathcal{F}_b)$ is the same as their interpretation in S for arguments in $dom(S') \cap B_{db}$ and it is extended arbitrarily to arguments in $dom(S') \setminus B_{db}$. Likewise, the interpretation in S' of the function symbols in $\Upsilon^T \cap \Upsilon_a$ is the same as their interpretation in S for arguments in $dom(S') \cap B_a$ and it is extended arbitrarily to arguments in $dom(S') \setminus B_a$.
- The interpretation in S' of function symbols in $\Upsilon^T \cap \Upsilon_\Lambda$ is as per Definition 9.1 for arguments in D_2 and it is extended arbitrarily to arguments in $dom(S') \setminus D_2$.
- T_{db} is interpreted as B_{db} and T_a as B_a .
- For every $n \geq 1$, T_n is interpreted as D_n and E_n as set membership restricted to n -tuples.

Each \mathcal{L}^{db} -formula φ of signature Υ can be rewritten as a first-order formula φ^* of signature Υ^T , where φ^* is obtained from φ by applying the following steps:

1. Replace each atomic formula of the form $\text{upd}(r, X)$ and $\text{upm}(r, X)$ by their corresponding definitions using the Axioms **U1–U7** and **Ü1–Ü7**, respectively.
2. Bring all remaining atomic formulae into the form $v_1 = v_2$, $f(v_2) = v_1$ or $X(v_1, \dots, v_n)$ (where each v_i denotes an appropriate first-order variable x_i or \mathbf{x}_i depending on the context) by applying the following equivalences:

$$\begin{aligned}
s = t &\leftrightarrow \exists v_1 (s = v_1 \wedge t = v_1) \\
X(t_1, \dots, t_n) &\leftrightarrow \exists v_1 \dots v_n (t_1 = v_1 \wedge \dots \wedge t_n = v_n \wedge X(v_1, \dots, v_n)) \\
f(s) = v_1 &\leftrightarrow \exists v_2 (s = v_2 \wedge f(v_2) = v_1)
\end{aligned}$$

3. Eliminate all modal operators by applying the following equivalences (again where each v_i denotes an appropriate first-order variable x_i or \mathbf{x}_i depending on the context):

$$\begin{aligned}
[X]v_1 = v_2 &\leftrightarrow (\text{IsUSet}(X) \wedge \text{conUSet}(X) \rightarrow v_1 = v_2) \\
[X]Y(v_1, \dots, v_n) &\leftrightarrow (\text{IsUSet}(X) \wedge \text{conUSet}(X) \rightarrow Y(v_1, \dots, v_n)) \\
[X]f(v_2) = v_1 &\leftrightarrow (\text{IsUSet}(X) \wedge \text{conUSet}(X) \rightarrow \\
&\quad X(c_f, v_2, v_1) \vee \forall v_3 (\neg X(c_f, v_2, v_3) \wedge f(v_2) = v_1)) \\
[X]\neg\varphi &\leftrightarrow (\text{IsUSet}(X) \wedge \text{conUSet}(X) \rightarrow \neg[X]\varphi) \\
[X](\varphi \vee \psi) &\leftrightarrow ([X]\varphi \vee [X]\psi) \\
[X]\forall v(\varphi) &\leftrightarrow \forall v([X]\varphi) \\
[X]\forall Y(\varphi) &\leftrightarrow \forall Y([X]\varphi)
\end{aligned}$$

4. Replace each atomic formula of the form $X(t_1, \dots, t_n)$ by $E(t_1, \dots, t_n, X)$, and relativise quantifiers over individuals in B_{db} to T_{db} , quantifiers over individuals in B_a to T_a , and quantifiers over n -ary relations in D_n for some $n \geq 1$ to T_n . More precisely,

φ^* is obtained by the recurrent application of the following rules to the formula φ obtained after applying steps 1–3.

$$\begin{aligned}
(v_1 = v_2)^* &= v_1 = v_2 \\
(Y(v_1, \dots, v_n))^* &= E(v_1, \dots, v_n, Y) \\
(f(v_2) = v_1)^* &= f(v_2) = v_1 \\
(\neg\varphi)^* &= \neg(\varphi^*) \\
(\varphi \vee \psi)^* &= (\varphi^* \vee \psi^*) \\
(\forall x(\varphi))^* &= \forall x(T_{db}(x) \rightarrow \varphi^*) \\
(\forall \mathbf{x}(\varphi))^* &= \forall \mathbf{x}(T_a(\mathbf{x}) \rightarrow \varphi^*) \\
(\forall X(\varphi))^* &= \forall X(T_n(X) \rightarrow \varphi^*) \quad (\text{if } X \text{ has arity } n)
\end{aligned}$$

It is then relatively easy to prove:

Lemma 9.1. *A \mathcal{L}^{db} -formula φ is true in S iff φ^* is true in S' .*

Thus, if φ^* is valid, then φ is true in all Henkin meta-finite structures. Note that the converse does not always hold. For example, $\exists \mathbf{x}(x = \mathbf{x})$ is true in all Henkin meta-finite structure (note that by the Background Postulate B_a is not empty), but $\exists x(T_a(x) \wedge (x = x))$ is not valid. In general, *not every* Υ^T -structure is an S' structure for some Henkin meta-finite Υ -structure S . Indeed, each Υ^T -structure S' which *does* correspond to some Henkin meta-finite structure S satisfies the following properties (cf. [27]):

1. Υ -correctness:
 - $f(x_1) = x_2 \rightarrow T_{db}(x_1) \wedge T_{db}(x_2)$ for every $f \in \Upsilon_{db}$,
 - $f(x_1) = x_2 \rightarrow T_{db}(x_1) \wedge T_a(x_2)$ for every $f \in \mathcal{F}_b$,
 - $f(x_1) = x_2 \rightarrow T_a(x_1) \wedge T_a(x_2)$ for every $f \in \Upsilon_a$,
 - $f(x_1) = x_2 \rightarrow T_2(x_1) \wedge T_a(x_2)$ for every $f \in \Upsilon_\Lambda$.
2. Non-emptiness: $\exists x(T_a(x))$.
3. Disjointness: $T_i(x) \rightarrow \neg T_j(x)$ for every $i, j \in \mathbb{N} \cup \{a, db\}$ such that $i \neq j$.
4. Elementhood: $E_n(x_1, \dots, x_n, y) \rightarrow T_n(y) \wedge (T_{db}(x_1) \vee T_a(x_1)) \wedge \dots \wedge (T_{db}(x_n) \vee T_a(x_n))$ for every $n \geq 1$.
5. Extensionality: $T_n(x) \wedge T_n(y) \wedge \forall \bar{z}(E_n(\bar{z}, x) \leftrightarrow E_n(\bar{z}, y)) \rightarrow x = y$ for every $n \geq 1$.
6. Comprehension: $\exists y \forall \bar{x}(E_n(\bar{x}, y) \leftrightarrow \psi)$ for every $n \geq 1$ and y non-free in ψ .

Lemma 9.2. *If A is a first-order structure of signature Υ^T which satisfies properties 1–6 above and $\text{sub}(A)$ is the sub-structure of A generated by the elements of $(T_{db})^A \cup (T_a)^A \cup \bigcup_{n \geq 1} (T_n)^A$, then $\text{sub}(A) = S'$ for some Henkin meta-finite structure S of signature Υ and corresponding first-order structure S' of signature Υ^T determined by S .*

Proof. Given A with domain $\text{dom}(A)$, we define S as follows:

- $B_{db} = (T_{db})^A$ is the base set of the database part of S .
- $B_a = (T_a)^A$ is the base set of the algorithmic part of S .
- For each $n \geq 1$, the universe D_n of n -ary relations consists of the sets $\{\bar{a} \in (B_{db} \cup B_a)^n \mid (E_n)^A(\bar{a}, s)\}$ for all $s \in (T_n)^A$.
- The interpretation of each function symbol $f \in \Upsilon$ is the same as in A but restricted to arguments from B_{db} , B_a or D_2 depending on whether f belongs to Υ_{db} , $\Upsilon_a \cup \mathcal{F}_b$ or Υ_Λ , respectively.

By the Υ -correctness, non-emptiness and comprehension properties of A , we get that S is a Henkin meta-finite structure.

We claim that $\text{sub}(A)$ is isomorphic to S' via $g : \text{dom}(S') \rightarrow \text{dom}(\text{sub}(A))$ where

$$g(x) = \begin{cases} x & \text{if } x \in (T_{db})^{S'} \cup (T_a)^{S'} \\ \{\bar{a} \in ((T_{db})^{S'} \cup (T_a)^{S'})^n \mid (E_n)^{S'}(\bar{a}, x)\} & \text{if } x \in (T_n)^{S'} \end{cases}$$

First, we get that g is well defined by the disjointness property and by the fact that, by definition of S and S' , every element x in $\text{dom}(S')$ is in $(T_{db})^{S'} \cup (T_a)^{S'} \cup \bigcup_{n \geq 1} (T_n)^{S'}$. That g is surjective follows from the definition of S' from A and the fact that $\text{dom}(\text{sub}(A))$ is the restriction of $\text{dom}(A)$ to $\text{dom}(S')$. By the extensionality property, we get that g is injective. By definition we get that g preserves the function symbols in Υ as well as the relation symbols T_{db} , T_a and T_n for every $n \geq 1$. Finally, for every $n \geq 1$, we get that g preserves E_n by the elementhood property. \square

Let Ψ be the set of formulae listed under properties 1–6 above. We then get the following Henkin style completeness theorem.

Theorem 9.3. *A \mathcal{L}^{db} -formula φ is true in all Henkin meta-finite structures iff φ^* is derivable in first-order logic from Ψ (i.e., iff $\Psi \vdash \varphi^*$).*

Proof. Assume that $\Psi \vdash \varphi^*$, and let S be a Henkin meta-finite structure. Then $S' \models \Psi$ and therefore $S' \models \varphi^*$. By Lemma 9.1, we get that $S \models \varphi$.

Conversely, assume that φ is true in all Henkin meta-finite structures. Towards showing $\Psi \models \varphi^*$, let us assume that $A \models \Psi$, and let $\text{sub}(A)$ be its substructure generated by the elements of $(T_{db})^A \cup (T_a)^A \cup \bigcup_{n \geq 1} (T_n)^A$. Then by Lemma 9.2, $\text{sub}(A) = S'$ for some first-order structure S' determined by a Henkin meta-finite structure S . Since by assumption we have that $S \models \varphi$, it follows from Lemma 9.1 that $S' \models \varphi^*$ and therefore $\text{sub}(A) \models \varphi^*$. But each quantifier in φ^* is relativised to $(T_{db})^A$, $(T_a)^A$ or $(T_n)^A$ for some $n \geq 1$, and then we also have that $A \models \varphi^*$. We have shown that $\Psi \models \varphi^*$, and then, by the completeness theorem of first-order logic, we get that $\Psi \vdash \varphi^*$. \square

We know from Theorem 7.3 that the deductive calculus L_2 introduced in Section 7.4 is sound. Thus, if φ is a \mathcal{L}^{db} -formula derivable in L_2 , then φ is true in all Henkin meta-finite structures. It is then immediate from Theorem 9.3 that φ^* is derivable in first-order logic from Ψ . On the other hand, it can be proven by an easy but lengthy induction on the length of the derivations that if φ^* is derivable in first-order from Ψ , then φ is derivable in L_2 .

Lemma 9.4. *φ^* is derivable in first-order from Ψ iff φ is derivable in L_2 .*

Finally, Theorem 9.3 and Lemma 9.4 immediately imply that the logic \mathcal{L}^{db} is complete to reason about DB-ASMs.

Theorem 9.5. *Let φ be a \mathcal{L}^{db} -formula and Φ be a set of \mathcal{L}^{db} -formulae. If $\Phi \models \varphi$, then $\Phi \vdash_{L_2} \varphi$.*

10. Conclusions

This article presents a logic for DB-ASMs. In accordance with the result that DB-ASMs and database transformations are behaviourally equivalent, it thus represents a logical characterisation for database transformations in general.

The logic for DB-ASMs is built upon the logic of meta-finite structures. The formalisation of multiset operations is captured by the notion of ρ -term. The use of ρ -terms greatly enhances the expressive power of the logic for DB-ASMs since aggregate computing in database applications can be easily expressed by using ρ -terms. On the other hand, ρ -terms can easily lead to incompleteness if we try to axiomatize them in the proof system. We avoid this problem by considering them as non-interpreted functions. In this way, we cannot reason about properties of ρ -terms themselves, but we can still use them in the formulae of our complete proof system to express meaningful properties of DB-ASMs.

As discussed in [39] and [10], the non-determinism accompanied with the use of choice rules poses a further challenging problem. In this work, we realized that the update sets produced by non-deterministic DB-ASMs rules are definable in a variant of second-order logic in which the second-order quantifiers are interpreted using a Henkin semantics, thus becoming part of the specification of a model rather than an invariant through all models as in the case of the classical second-order semantics. Base on these definitions, we use the modal operator $[X]$ where X is a second-order variable that represents an update set U generated by a (possibly non-deterministic) DB-ASM rule r . By introducing $[X]$ into the logic for DB-ASMs, it is shown that nondeterministic database transformations can also be captured.

The use of a Henkin semantics in the definition of the logic \mathcal{L}^{db} for DB-ASMs allowed us to show that \mathcal{L}^{db} is actually a syntactic variant of a complete first-order theory of types. In turn, this allowed us to establish a sound and complete proof system for the logic for DB-ASMs, which can be turned into a tool for reasoning about database transformations. However, this is restricted to reasoning about steps, not full runs, but no complete logic

for reasoning about runs can be expected. In the future we will continue to investigate how the logic for DB-ASMs can be tailored towards different classes of database transformations such as XML data transformations and used for verifying the properties of database transformations in practice.

References

- [1] Serge Abiteboul and Paris C. Kanellakis. Object identity as a query language primitive. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 159–173. ACM Press, 1989.
- [2] Serge Abiteboul and Victor Vianu. A transaction language complete for database update and specification. In Moshe Y. Vardi, editor, *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 260–268. ACM, 1987.
- [3] Serge Abiteboul and Victor Vianu. Datalog extensions for database queries and updates. *J. Comput. Syst. Sci.*, 43(1):62–124, 1991.
- [4] Catriel Beeri and Bernhard Thalheim. Identification as a primitive of data models. In Torsten Polle, Torsten Ripke, and Klaus-Dieter Schewe, editors, *Fundamentals of Information Systems*, pages 19–36. Kluwer Academic Publishers, Boston Dordrecht London, 1999.
- [5] Andreas Blass and Yuri Gurevich. Abstract state machines capture parallel algorithms. *ACM Transactions on Computational Logic*, 4(4):578–651, October 2003.
- [6] Andreas Blass and Yuri Gurevich. Abstract state machines capture parallel algorithms: Correction and extension. *ACM Transactions on Computation Logic*, 9(3):1–32, 06 2008.
- [7] Andreas Blass, Yuri Gurevich, and Saharon Shelah. On polynomial time computation over unordered structures. *The Journal of Symbolic Logic*, 67(3):1093–1125, September 2002.
- [8] Anthony J. Bonner and Michael Kifer. The state of change: A survey. In *International Seminar on Logic Databases and the Meaning of Change, Transactions and Change in Logic Databases*, pages 1–36. Springer-Verlag, 1998.
- [9] E. Börger and Robert F. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag New York, Inc., 2003.
- [10] Egon Börger and Robert F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.

- [11] J.-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 612–617. IEEE Computer Society, 1989.
- [12] Ashok K. Chandra and David Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.
- [13] D Fensel and R Groenboom. MLPM: Defining a semantics and axiomatization for specifying the reasoning process of knowledge-based systems. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, Budapest, Hungary, 1996.
- [14] Flavio Ferrarotti, Wei Ren, and Jose Maria Turull Torres. Expressing properties in second- and third-order logic: hypercube graphs and SATQBF. *Logic Journal of the IGPL*, 22(2):355–386, 2014.
- [15] Flavio Ferrarotti, Klaus-Dieter Schewe, Loredana Tec, and Qing Wang. A new thesis concerning synchronised parallel computing - simplified parallel ASM thesis. *Theor. Comput. Sci.*, 649:25–53, 2016.
- [16] E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 1998.
- [17] Erich Grädel and Martin Otto. Inductive definability with counting on finite structures. In *Selected Papers from the Workshop on Computer Science Logic*, pages 231–247. Springer-Verlag, 1993.
- [18] R. Groenboom and G. Renardel de Lavalette. Reasoning about dynamic features in specification languages - a modal view on creation and modification. In *Proceedings of the International Workshop on Semantics of Specification Languages (SoSL)*, pages 340–355. Springer-Verlag, 1994.
- [19] R. Groenboom and G. Renardel de Lavalette. A formalization of evolving algebras. In *Proceedings of Accolade95*. Dutch Research School in Logic, 1995.
- [20] Yuri Gurevich. A new thesis (abstracts). *American Mathematical Society*, 6(4):317, August 1985.
- [21] Yuri Gurevich. Sequential abstract state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, July 2000.
- [22] Yuri Gurevich. Abstract state machines: An overview of the project. In *International Symposium on Foundations of Information and Knowledge Systems*, pages 6–13, 2004.
- [23] Lauri Hella, Leonid Libkin, Juha Nurmonen, and Limsoon Wong. Logics with aggregate operators. *Journal of the ACM*, 48(4):880–907, 2001.

- [24] Leon Henkin. Completeness in the theory of types. *J. Symbolic Logic*, 15(2):81–91, 06 1950.
- [25] G.E. Hughes and MJ Cresswell. *A new introduction to modal logic*. Burns & Oates, 1996.
- [26] N. Immerman. Expressibility as a complexity measure: Results and directions. In *Proceedings of Second Conference on Structure in Complexity Theory*, pages 194–202, 1987.
- [27] Daniel Leivant. Higher order logic. In Dov M. Gabbay, Christopher J. Hogger, J. A. Robinson, and Jörg H. Siekmann, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 2, Deduction Methodologies*, pages 229–322. Oxford University Press, 1994.
- [28] Stanislas Nanchen. *Verifying abstract state machines*. PhD thesis, ETH Zürich, 2007.
- [29] M. Otto. *Bounded variable logics and counting – A study in finite models*, volume 9. Springer-Verlag, 1997.
- [30] Martin Otto. The expressive power of fixed-point logic with counting. *Journal of Symbolic Logic*, 61:147–176, 1996.
- [31] G. Renardel de Lavalette. A logic of modification and creation. In *Logical Perspectives on Language and Information. CSLI publications*, 2001.
- [32] Klaus-Dieter Schewe and Bernhard Thalheim. Fundamental concepts of object oriented databases. *Acta Cybernetica*, 11(1-2):49–84, 1993.
- [33] Klaus-Dieter Schewe and Qing Wang. A customised ASM thesis for database transformations. *Acta Cybernetica*, 19(4):765–805, 2010.
- [34] A. Schönegge. Extending Dynamic Logic for Reasoning about Evolving Algebras. Technical Report 49/95, Universität Karlsruhe, Fakultät für Informatik, 1995.
- [35] P.A. Spruit. *Logics of Database Updates*. PhD thesis, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, 1994.
- [36] P.A. Spruit, R.J. Wieringa, and J.-J.Ch. Meyer. Dynamic database logic: The first-order case. In U.W. Lipeck and B. Thalheim, editors, *Modelling Database Dynamics*, pages 103–120. Springer, 1993.
- [37] Paul Spruit, Roel Wieringa, and John-Jules Meijer. Axiomatization, declarative semantics and operational semantics of passive and active updates in logic databases. *Journal of Logic and Computation*, 5:27–50, 1995.

- [38] Paul Spruit, Roel Wieringa, and John-Jules Meyer. Regular database update logics. *Theoretical Computer Science*, 254(1-2):591–661, 2001.
- [39] Robert Stärk and Stanislas Nanchen. A logic for abstract state machines. *Journal of Universal Computer Science*, 7(11), 2001.
- [40] Jose Maria Turull Torres. On the expressibility and the computability of untyped queries. *Annals of Pure and Applied Logic*, 108(1-3):345–371, 2001.
- [41] Jose Maria Turull Torres. Relational databases and homogeneity in logics with counting. *Acta Cybernetica*, 17(3):485–511, 2006.
- [42] J. Van den Bussche. *Formal Aspects of Object Identity in Database Manipulation*. PhD thesis, University of Antwerp, 1993.
- [43] Jan Van den Bussche and Dirk Van Gucht. 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. Springer-Verlag, 1993.
- [44] Jan van den Bussche and Dirk van Gucht. A semideterministic approach to object creation and nondeterminism in database queries. *J. Comput. Syst. Sci.*, 54(1):34–47, 1997.
- [45] Jan Van Den Bussche, Dirk Van Gucht, Marc Andries, and Marc Gyssens. On the completeness of object-creating database transformation languages. *Journal of the ACM*, 44(2):272–319, 1997.
- [46] Pascal van Eck, Joeri Engelfriet, Dieter Fensel, Frank van Harmelen, Yde Venema, and Mark Willems. A survey of languages for specifying dynamics: A knowledge engineering perspective. *IEEE Transactions on Knowledge and Data Engineering*, 13(3):462–496, 2001.
- [47] Qing Wang. *Logical Foundations of Database Transformations for Complex-Value Databases*. Berlin, Germany: Logos-Verlag, 2010.