Journal of Universal Computer Science, vol. 16, no. 20 (2010), 3043-3072
submitted: 1/7/10, accepted: 13/9/10, appeared: 1/11/10 © *J.UCS*

XML Database Transformations

Klaus-Dieter Schewe
(Software Competence Centre Hagenberg, Austria
kd.schewe@scch.at)

Qing Wang
(University of Otago, Dunedin, New Zealand
qing.wang@otago.ac.nz)

Abstract: Database transformations provide a unifying umbrella for queries and updates. In general, they can be characterised by five postulates, which constitute the database analogue of Gurevich's sequential ASM thesis. Among these postulates the background postulate supposedly captures the particularities of data models and schemata. For the characterisation of XML database transformations the natural first step is therefore to define the appropriate tree-based backgrounds, which draw on hereditarily finite trees, tree algebra operations, and extended document type definitions. This defines a computational model for XML database transformation using a variant of Abstract State Machines. Then the incorporation of weak monadic second-order logic provides an alternative computational model called XML machines. The main result is that these two computational models for XML database transformations are equivalent.
Key Words: Abstract State Machine, Computation Background, Database Transformation, eXtensible Markup Language, Monadic Second-order Logic, Tree Algebra
Category: F.4.1, F.1.1, E.1, H.2.3, I.7.2

1 Introduction

For a long time database transformations as a unifying umbrella for queries and updates have been the focus of the database research community. The logical foundations of queries have always been a central focus of interest for database theoreticians. In particular, queries are considered as transformations from an input schema to an extended output schema that preserves the input. From here it is a very small step to formalising binary relations on database instances that encompass queries and updates [Abiteboul and Vianu, 1988]. Since then a lot of research has been undertaken aiming at a logical characterisation of database transformations (e.g. [Abiteboul and Kanellakis, 1998; Van den Bussche, 1993; Van Den Bussche et al., 1997; Van den Bussche and Van Gucht, 1992]).

As discussed in [Abiteboul and Kanellakis, 1998], database transformations should satisfy criteria such as well-typedness, effective computability, genericity and functionality. However, the results of these investigations were only fully satisfactory in the case of queries. Extending these results to updates is by no means straightforward [Van den Bussche and Van Gucht, 1993]. A computation model that can serve as a theoretical foundation for database transformations in general exploiting the theory of Abstract State Machines (ASMs) has only recently been proposed [Schewe and Wang, 2010].

1.1 Previous Work

The sequential and parallel ASM theses [Gurevich, 2000; Blass and Gurevich, 2003] define sequential and parallel algorithms by a set of intuitive postulates. Blass and Gurevich show that sequential and parallel Abstract State Machines [Börger and Stärk, 2003], respectively, capture these classes of algorithms. In our previous work [Schewe and Wang, 2010] we picked up on this line of thought characterising database transformations in general. Similar to the ASM theses we formulated five postulates that define database transformations, and proved that these are exactly captured by a variant of ASMs called Database Abstract State Machines (DB-ASMs).

One key issue in these postulates is the adoption of meta-finite states [Grädel and Gurevich, 1998], i.e. as for ASMs states are defined by first-order structures, but for database transformations we require that these structures are composed of a finite database part and an unrestricted algorithmic part with bridge functions linking them. Other differences to the general postulates in the sequential and parallel ASM theses concern the dealing with partial updates [Gurevich and Tillmann, 2005], the use of a limited form of parallelism by means of aggregate updates [Cohen, 2006], and the incorporation of a restricted form of non-determinism originating from choices among query results. The non-determinism is further regulated by the bounded non-determinism postulate, which states that non-determinism is always bounded to a choice among substructures of the database part of a state. A major difference is further that a bounded exploration witness in the bounded exploration postulate is not restricted to a finite set of closed terms, but may involve also so-called access terms, which capture associative access to a database. This modified bounded exploration postulate is the major source of difficulty for the proof of the characterisation theorem [Schewe and Wang, 2010].

While the characterisation of database transformations is done without any reference to a particular data model, the presence of backgrounds [Blass and Gurevich, 2000; Blass and Gurevich, 2007] supposedly enables tailoring the characterisation to any data model of interest. That is, the equivalence between the postulates and DB-ASMs holds with respect to a fixed background. The corresponding background postulate only defines the minimum requirements for backgrounds such as the availability of truth values, records and multisets with the necessary operations in each state. For instance, when dealing with relational databases finite relational signatures must be adopted [Abiteboul et al., 1995;

Ebbinghaus and Flum, 1999], and when dealing with XML tree-based structures have to become part of the background.

1.2 Contributions

In this article we build upon the general characterisation of database transformations to develop a theoretical framework for XML database transformations. This leads to two major contributions: First of all, we provide tree-based backgrounds that are necessary for XML, which define a subclass of DB-ASMs that can exactly capture XML database transformations. Secondly, we define a more elegant computational model, called XML machines, which mainly differs from DB-ASMs with tree-based backgrounds by the use of weak monadic second-order logic in forall and choice rules. We then show that this model of XML machines is behaviourally equivalent to the DB-ASM model with the same tree-based background.

For the first of these contributions we define unranked trees using child and sibling relations. This is tailored towards XML by labelling and value functions. Furthermore, we define an XML tree algebra on hedges (i.e. list of trees), contexts and labels, extending several known tree algebras [Bojanczyk and Walukiewicz, 2007; Wilke, 1996]. Thus, hereditarily finite trees together with the functions required for trees and hedge algebra operations, provide the major ingredients for tree-based backgrounds.

Furthermore, in order to capture schemata for XML documents, we adopt extended document type definitions (EDTDs) [Papakonstantinou and Vianu, 2000], which according to [Murata et al., 2005] subsume many of other XML schema definition languages such as DTDs or XML Schema. This leads to the addition of tree typing schemes into a tree-based background. For XML database transformations we only require that initial and final states adhere to given typing schemes. Adopting the results in this paper to any other XML schema formalism is straightforward.

The disadvantage of the ASM-based characterisation of XML database transformations is its lack of linkages to other work on theoretical foundations of XML databases. As XML is intrinsically connected with regular languages, a lot of research has been done to link XML with automata and defining logics. Weak monadic second-order logics (MSO) are linked to regular tree languages [Doner, 1970; Thatcher and Wright, 1968] in the sense that a set of trees is regular iff it is in weak MSO with k successors.

Therefore, we define an alternative model of computation for XML database transformations, which exploits weak MSO [Wang and Ferrarotti, 2009]. Pragmatically speaking, the use of weak MSO formulae in forall and choice rules permits more flexible access to the database. As weak MSO subsumes first-order logic, it is straightforward to see that the model of XML machines captures all transformations that can be expressed by the DB-ASM model with the same tree-based background. As our main result in [Schewe and Wang, 2010] states that DB-ASMs capture all database transformations as defined by the intuitive postulates, it should also not come as a surprise that XML machines are in fact equivalent to DB-ASMs with tree-based backgrounds. For the proof we show that XML machines satisfy the postulates in a sense the hard part of the proof is already captured by the main characterisation theorem in [Schewe and Wang, 2010].

1.3 Related Work

Our research is devoted to XML database transformations exploiting the theory of Abstract State Machines, so naturally ASMs, database tansformations in general, and XML database theory broadly define related research areas.

The seminal work by Gurevich on the sequential ASM thesis [Gurevich, 2000] is most inspiring for our work. From it we borrowed the idea to formalise database transformations by intuitive postulates, and to define an abstract machine model capturing them. ASMs have been used in all kinds of application areas, in particular also in database systems. This includes among others concurrency control [Kirchberg et al., 2009], recovery [Gurevich et al., 1997], and even the development of data warehouses [Zhao et al., 2009]. ASMs have also been studied as a means to express complete query languages [Blass et al., 2002]. All this research has the common denominator that ASMs are applied to particular database problems, whereas our research actually aims at adopting the fundamental theoretical ideas underlying ASMs to study database theory. To our knowledge this has not been done elsewhere.

With respect to database transformations in general our research also draws on ideas from the work of van den Bussche et al. in the context of object-oriented databases [Van den Bussche, 1993; Van Den Bussche et al., 1997; Van den Bussche and Van Gucht, 1992]. One significant difference, however, is that we did not yet pay particular attention to the characterisation of queries.

Over the last decade, a lot of research effort has been put in the area of XML databases, but the emphasis is usually on querying, whereas updates are neglected. Several extensions of XQuery to encompass updates that we are aware of [Chamberlin et al., 2006; Don Chamberlin and Siméon, 2008; Ghelli et al., 2006; Sur et al., 2004] generally use explicit *snapshot semantics* to control the evaluation order of updates at certain level of snapshot granularity, which are often accompanied with some restrictions on the expression usage and the design of error handling. It turns out that a well-founded theory with high-level specification for XML database transformations including updates is still missing. Our research draws on some approaches in this area.

To manipulate over tree structures, we need operators on trees or hedges. This motivates us to take into account tree algebras that have, however, been developed for various different purposes. For instance, the forest algebra in [Bojanczyk and Walukiewicz, 2007] aims at providing an algebraic framework for studying logical definability of different classes of tree languages, while the tree algebra for binary ranked trees in [Wilke, 1996] was defined for characterising the class of frontier-testable tree languages satisfying conditions such as "there exists a natural number k such that any two trees with the same set of subtrees of depth at most k either belong both to the language or both not". In this paper, we develop an XML tree algebra that adapts features from these existing tree algebras towards the setting of XML trees.

Similarly, the research on XML schema languages, from which we adopt EDTDs [Papakonstantinou and Vianu, 2000]. As MSO has intrinsic connection with regular languages, which dates back to the observation by Büchi [Büchi, 1960], MSO has become important for XML database theory, see e.g. [Comon et al., 2007]. Here, we mainly exploit weak MSO as a means to define an alternative, more elegant way to express XML database transformations without actually increasing expressiveness, in comparison with the DB ASM models with the same tree-based backgrounds.

1.4 Organisation of the Article

The rest of the article is organised as follows. In Section 2 we give a brief presentation of the five postulates for database transformations from [Schewe and Wang, 2010]. We also present the variant of ASMs called Database Abstract State Machines (DB-ASMs) that have been proven to capture database transformations. Sections 3–5 are devoted to the development of tree-based backgrounds that will customize the background postulate for XML database transformations. We start in Section 3 with formally defining trees and a tree algebra, which give rise to constructors and functions that must be part of a background class. Section 4 introduces a weak monadic second-order logic, which adds further requirements to tree-based backgrounds. Section 5 contains the final definition of tree-based backgrounds, which in addition comprise tree type schemes in order to capture schema information based on EDTDs. Then we link tree-based backgrounds with DB-ASMs thereby obtaining the first computational model for XML database transformations. The alternative computational model of XML machines, which exploit weak MSO logic is introduced in Section 6. We then prove our main result that the model of XML machines is equivalent to the DB-ASM model with tree-based backgrounds. Finally, we draw conclusions in Section 7.

2 Database Transformations

In this section we summarize our previous work on database transformations [Schewe and Wang, 2010]. We start with the five postulates: the *sequential time*, *abstract state*, *background*, *bounded exploration* and *bounded non-determinism* postulates. An object satisfying these postulates is a *database transformation*.

2.1 Postulates Defining Database Transformations

The sequential time postulate defines a database transformation as a computation that proceeds step-wise on a set of states.

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

A *run* of a database transformation t is a finite sequence $S_0, \dots, S_f$ of states with $S_0 \in \mathcal{I}_t$, $S_f \in \mathcal{F}_t$, $S_i \notin \mathcal{F}_t$ for $0 < i < f$, and $(S_i, S_{i+1}) \in \tau_t$ for all $i = 0, \dots, f-1$. Database transformations t_1 and t_2 are *behaviourally equivalent* if $\mathcal{S}_{t_1} = \mathcal{S}_{t_2}$, $\mathcal{I}_{t_1} = \mathcal{I}_{t_2}$, $\mathcal{F}_{t_1} = \mathcal{F}_{t_2}$ and $\tau_{t_1} = \tau_{t_2}$ hold.

The abstract state postulate generalises the corresponding postulate for sequential algorithms [Gurevich, 2000], according to which states are first-order structures, i.e. sets of functions. These functions are interpretations of function symbols given by some signature.

Definition 1. A *signature* Σ is a set of function symbols, each associated with a fixed arity. A *structure* over Σ consists of a set B, called the *base set* of the structure together with interpretations of all function symbols in Σ, i.e. if $f \in \Sigma$ has arity k, then it is interpreted by a function from B^k to B. An *isomorphism* from structure X to structure Y is defined by a bijection $\sigma : B_X \to B_Y$ between the base sets that extends to functions by $\sigma(f_X(b_1, \dots, b_k)) = f_Y(\sigma(b_1), \dots, \sigma(b_k))$, where $f_X(b_1, \dots, b_k)$ denotes the interpretation of function $f(b_1, \dots, b_k)$ in structure X.

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

Furthermore, the signature Σ_t is composed as a disjoint union out of a database signature Σ_{db}, an algorithmic signature Σ_a, and a finite set of unary bridge function symbols, i.e. $\Sigma_t = \Sigma_{db} \cup \Sigma_a \cup \{f_1, \dots, f_\ell\}$. The base set of a state is $B = B_{db} \cup B_a$ with interpretation of function symbols in Σ_{db} and Σ_a over B_{db} and B_a, respectively. The interpretation of a bridge function symbol defines a function from B_{db} to B_a. With respect to such states the restriction to Σ_{db} is a finite structure, i.e. B_{db} is finite.

The background postulate describes the background of a computation (i.e. everything that is needed for the computation, but not captured by the notion of state). For database transformations, in particular, we have to capture constructs that are determined by the used data model, for example, relational, object-oriented, object-relational or semi-structured. That means, we will have to deal with type constructors and with functions defined on such types. Furthermore, when we allow values (e.g. identifiers) to be created non-deterministically, we would like to take these values out of an infinite set of reserve values. Once created, these values become active, and we can assume they can never be used again for this purpose.

Following [Blass and Gurevich, 2003] we use background classes to define backgrounds. A background class is determined by a background signature consisting of constructor and function symbols. Function symbols are associated with a fixed arity as in Definition 1, while for constructor symbols we permit the arity to be unfixed or bounded.

Definition 2. Let $\mathcal{D}$ be a set of base domains and V_K a background signature, then a *background class* $\mathcal{K}$ with V_K over $\mathcal{D}$ is constituted by the universe $U = \bigcup \mathcal{D}^{'}$ of elements, where $\mathcal{D}^{'}$ is the smallest set with $\mathcal{D} \subseteq \mathcal{D}^{'}$ satisfying the following properties for each constructor symbol $\llcorner \lrcorner \in V_K$:

- If $\llcorner \lrcorner \in V_K$ has unfixed arity, then $\llcorner D \lrcorner \in \mathcal{D}^{'}$ for all $D \in \mathcal{D}^{'}$, and $\llcorner a_1, \ldots, a_m \lrcorner \in \llcorner D \lrcorner$ for every $m \in \mathbb{N}$ and $a_1, \ldots, a_m \in D$, and $A_{\llcorner\lrcorner} \in \mathcal{D}^{'}$ with $A_{\llcorner\lrcorner} = \bigcup\limits_{\llcorner D \lrcorner \in \mathcal{D}^{'}} \llcorner D \lrcorner$;
- If $\llcorner \lrcorner \in V_K$ has bounded arity n, then $\llcorner D_1, \ldots, D_m \lrcorner \in \mathcal{D}^{'}$ for all $m \leq n$ and $D_i \in \mathcal{D}^{'}$ $(1 \leq i \leq m)$, and $\llcorner a_1, \ldots, a_m \lrcorner \in \llcorner D_1, \ldots, D_m \lrcorner$ for every $m \in \mathbb{N}$ and $a_1, \ldots, a_m \in D$;
- If $\llcorner \lrcorner \in V_K$ has fixed arity n, then $\llcorner D_1, \ldots, D_n \lrcorner \in \mathcal{D}^{'}$ for all $D_i \in \mathcal{D}^{'}$ and $\llcorner a_1, \ldots, a_n \lrcorner \in \llcorner D_1, \ldots, D_n \lrcorner$ for all $a_i \in D_i$ $(1 \leq i \leq n)$,

and an interpretation of function symbols in V_K over U.

Postulate 3 (background). Each state of a database transformation t must contain an infinite set of reserve values, truth values and their connectives, the equality predicate, the undefinedness value $\perp$, and a background class $\mathcal{K}$ defined by a background signature V_K that contains at least a binary tuple constructor

$(\cdot)$, a multiset constructor $\{\!\{\cdot\}\!\}$, and function symbols for operations on pairs such as pairing and projection, and on multisets such as empty multiset $\{\!\{\,\}\!\}$, singleton $\{\!\{x\}\!\}$, and multiset union $\uplus$.

Given a state, we can add the required Booleans and $\perp$ into the base domains, then apply the construction in Definition 2 to obtain a much larger base set and interpret functions symbols with respect to this enlarged base set.

The exploration boundary postulate for sequential algorithms requests that only finitely many terms can be updated in an elementary step [Gurevich, 2000]. For parallel algorithms this postulate becomes significantly more complicated, as basic constituents not involving any parallelism (so-called "proclets") have to be considered [Blass and Gurevich, 2003]. For database transformations the problem lies somehow in between. Computations are intrinsically parallel, even though implementations may be sequential, but the parallelism is restricted in the sense that all branches execute de facto the same computation. We capture this by means of location operators, which generalise aggregation functions as in [Cohen, 2006]. Let $\mathcal{M}(D)$ be the set of all non-empty multisets over a domain D, then a *location operator* ρ over $\mathcal{M}(D)$ consists of a unary function $\alpha : D \to D^{'}$, a commutative and associative binary operation $\odot$ over $D^{'}$, and a unary function $\beta : D^{'} \to D^{''}$, which define $\rho(m) = \beta(\alpha(b_1) \odot \cdots \odot \alpha(b_n))$ for $m = \{\!\{b_1, ..., b_n\}\!\} \in \mathcal{M}(D)$.

The definitions of updates, update sets and update multisets are the same as for ASMs [Börger and Stärk, 2003]. For a database transformation t, let S be a state of t, f a function symbol of arity n in the state signature of t, and $a_1, ..., a_n, v$ be elements in the base set of S, then an *update* of t is a pair (ℓ, v), where ℓ is a location $f(a_1, ..., a_n)$. An *update set* is a set of updates; an *update multiset* is a multiset of updates. An update is *trivial* in a state s if its location content in s is the same as its update value, while an update set is *trivial* if all of its updates are trivial. An update set Δ is *consistent* if it does not contain conflicting updates, i.e. for all $(\ell, v), (\ell, v') \in \Delta$ we have $v = v'$.

Using location operators that are assigned to locations, an update multiset can be reduced to an update set. It is further possible to construct for each $(S, S') \in \tau_t$ a minimal, consistent update set $\Delta(t, S, S')$ such that applying this update set to the state S will produce the state S'. For a database transformation t and a state $S \in \mathcal{S}_t$ define $\Delta(t, S) = \{\Delta(t, S, S') \mid (S, S') \in \tau_t\}$.

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

Definition 3. An *access term* is either a ground term α or a pair (β, α) of terms, the variables $x_1, \ldots, x_n$ in which refer to the arguments of some $f \in \Sigma_{db}$. The interpretation of (β, α) in a state S is the set $\{f(x_1, \dots, x_n)[a_1/x_1, \dots, a_n/x_n] \mid val_{S,\{x_1 \mapsto a_1, \dots, x_n \mapsto a_n\}}(\beta) = val_{S,\{x_1 \mapsto a_1, \dots, x_n \mapsto a_n\}}(\alpha)\}$ of locations. Structures S_1 and S_2 *coincide* over a set T of access terms if the interpretation of each $(\beta, \alpha) \in T$ over S_1 and S_2 are equal.

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

The last postulate addresses the boundedness of non-determinism by putting a severe restriction on the non-determinism in the transition relation τ_t.

Postulate 5 (bounded non-determinism). If there are states $S_1, S_2, S_3 \in \mathcal{S}_t$ with $(S_1, S_2) \in \tau_t$, $(S_1, S_3) \in \tau_t$ and $S_2 \neq S_3$, then there exists an access term of the form (β, α) in $T_{witness}$.

This postulate states that if in a one-step transition over state S we have non-determinism (equivalently: $\Delta(t, S)$ contains more than one update set), then there must exist an access term that is not a ground term in the bounded exploration witness of t. In other words, if we only have ground access terms in the bounded exploration witness of t, then we cannot have non-determinism in the transitions.

2.2 Database Abstract State Machines

For the signature Σ we adopt the requirements of the abstract state postulate, i.e. it comprises a sub-signature Σ_{db} for the database part, a sub-signature Σ_a for the algorithmic part, and bridge functions $\{f_1, \dots, f_\ell\}$. For states we assume that the requirement in the abstract state postulate, according to which the restriction to Σ_{db} results in a finite structure, is satisfied. Furthermore, we assume a background in the sense of the background postulate being defined. We refer to *database variables* as variables that must be interpreted by elements in B_{db} and $fr(r)$ as the set of free variables appearing in a DB-ASM rule r. A rule r is called *closed* if $fr(r) = \emptyset$.

Definition 4. The set $\mathcal{R}$ of *DB-ASM rules* over a signature $\Sigma = \Sigma_{db} \cup \Sigma_a \cup \{f_1, \dots, f_\ell\}$ is defined as follows:

– If $t_0, \dots, t_n$ are terms over Σ, and f is an n-ary function symbol in Σ, then $f(t_1, \dots, t_n) := t_0$ is a rule r in $\mathcal{R}$ called *assignment rule* with $fr(r) = \bigcup_{i=0}^{n} var(t_i)$, where $var(t_i)$ is the set of variables occurring in the terms t_i $(i = 0, \dots, n)$.

– If φ is a Boolean term and $r' \in \mathcal{R}$ is a DB-ASM rule, then **if** φ **then** r' **endif** is a rule r in $\mathcal{R}$ called *conditional rule* with $fr(r) = fr(\varphi) \cup fr(r')$.

– If φ is a Boolean term with only database variables $fr(\varphi) \supseteq \{x_1, \ldots, x_k\}$ and $r' \in \mathcal{R}$ is a DB-ASM rule, then **forall** $x_1, \ldots, x_k$ **with** φ **do** r' **enddo** is a rule r in $\mathcal{R}$ called *forall rule* with $fr(r) = fr(r') \cup fr(\varphi) - \{x_1, \ldots, x_k\}$.

– If $r_1, \ldots, r_n$ are rules in $\mathcal{R}$, then also the rule r defined as **par** $r_1 \ldots r_n$ **par** is a rule in $\mathcal{R}$, called *parallel rule* with $fr(r) = \bigcup_{i=0}^{n} fr(r_i)$.

– If φ is a Boolean term with only database variables $fr(\varphi) \supseteq \{x_1, \ldots, x_k\}$ and $r' \in \mathcal{R}$ is a DB-ASM rule, then **choose** $x_1, \ldots, x_k$ **with** φ **do** r' **enddo** is a rule r in $\mathcal{R}$ called *choice rule* with $fr(r) = fr(r') \cup fr(\varphi) - \{x_1, \ldots, x_k\}$.

– If r_1, r_2 are rules in $\mathcal{R}$, then also the rule r defined as **seq** r_1 r_2 **seq** is a rule in $\mathcal{R}$, called *sequence rule* with $fr(r) = fr(r_1) \cup fr(r_2)$.

– If $r' \in \mathcal{R}$ is a DB-ASM rule and ϑ is a location function that assigns location operators ϱ to terms γ with $var(\gamma) \subseteq fr(r')$, then **let** $\vartheta(\gamma) = \varrho$ **in** r' **endlet** defines another DB-ASM rule $r \in \mathcal{R}$ called *let rule* with $fr(r) = fr(r')$.

The definition of sets of update sets $\Delta(r, S)$ for a closed DB-ASM rule r with respect to states S (defined with a background as in Postulate 3 and a finiteness condition as in Postulate 2) is straightforward [Schewe and Wang, 2010].

Definition 5. A *Database Abstract State Machine* (DB-ASM) M over signature Σ as in Postulate 2 and with a background as in Postulate 3 consists of a set $\mathcal{S}_M$ of states over Σ satisfying the requirements in Postulate 2 and closed under isomorphisms, non-empty subsets $\mathcal{I}_M \subseteq \mathcal{S}_M$ of initial states, and $\mathcal{F}_M \subseteq \mathcal{S}_M$ of final states, both also closed under isomorphisms, a program π_M defined by a closed DB-ASM rule r over Σ, and a binary relation τ_M over $\mathcal{S}_M$ determined by π_M such that $\{S_{i+1} \mid (S_i, S_{i+1}) \in \tau_M\} = \{S_i + \Delta \mid \Delta \in \Delta(\pi_M, S_i)\}$ holds.

The following two theorems are the main results in [Schewe and Wang, 2010].

Theorem 6. *Each DB-ASM M defines a database transformation t_M with the same signature and background as M.*

Theorem 7. *For each database transformation t there is a DB-ASM M such that t and t_M are behaviourally equivalent.*

3 Trees and Tree Algebra

As XML documents are trees, XML database transformations have to perform computations on trees. Thus tree values have to be available in the base set, and operations on such trees must be available for the definition of terms. Furthermore, tree values and operations have to be defined in the background. In this section we provide these necessary constituents of the background, which leads to the definition of *tree background classes* in Section 5.

3.1 Trees, Contexts and Selector Constructs

It is common to regard an XML document as an unranked tree, in which a node may have an unbounded but finite number of children nodes.

Definition 8. An *unranked tree* is a structure $(\mathcal{O}, \prec_c, \prec_s)$ consisting of a finite, non-empty set $\mathcal{O}$ of node identifiers, called *tree domain*, ordering relations $\prec_c$ and $\prec_s$ over $\mathcal{O}$ called *child relation* and *sibling relation*, respectively, satisfying the following conditions:

- there exists a unique, distinguished node $o_r \in \mathcal{O}$ (called the *root* of the tree) such that for all $o \in \mathcal{O} - \{o_r\}$ there is exactly one $o' \in \mathcal{O}$ with $o' \prec_c o$, and
- whenever $o_1 \prec_s o_2$ holds, then there is some $o \in \mathcal{O}$ with $o \prec_c o_i$ for $i = 1, 2$.

The relations $\prec_c$ and $\prec_s$ are irreflexive ($x \not\prec x$).

For $x_1 \prec_c x_2$ we say that x_2 is a *child* of x_1; for $x_1 \prec_s x_2$ we say that x_2 is the *next sibling* to the right of x_1. In order to obtain XML trees from this, we require the nodes of an unranked tree to be labelled, and the leaves, i.e. nodes without children, to be associated with values. Therefore, we fix a finite, non-empty set Σ of *labels*, and a finite family $\{\tau_i\}_{i\in i}$ of data types. Each data type τ_i is associated with a *value domain* $dom(\tau_i)$. The corresponding *universe* U contains all possible values of these data types, i.e. $U = \bigcup_{i\in I} dom(\tau_i)$.

Definition 9. An *XML tree* t (over the set of labels Σ with values in the universe U) is a triple $(\gamma_t, \omega_t, \upsilon_t)$ consisting of an unranked tree $\gamma_t = (\mathcal{O}_t, \prec_c, \prec_s)$, a total *label function* $\omega_t\colon \mathcal{O}_t \to \Sigma$, and a partial *value function* $\upsilon_t\colon \mathcal{O}_t \to U$ such that whenever υ_t is defined on the argument o, o is a leaf in γ_t.

We use $root(t)$ to denote the root node of an XML tree t. Given two XML trees t_1 and t_2, t_1 is the *subtree* of t_2 if the following properties are satisfied: (1) $\mathcal{O}_{t_1} \subseteq \mathcal{O}_{t_2}$, (2) $o_1 \prec_c o_2$ holds in t_1 iff it holds in t_2, (3) $o_1 \prec_s o_2$ holds in t_1 iff it holds in t_2, (4) $\omega_{t_1}(o') = \omega_{t_2}(o')$ holds for all $o' \in \mathcal{O}_{t_1}$, and (5) either

$\upsilon_{t_1}(o') = \upsilon_{t_2}(o')$ holds or otherwise both sides are undefined for all $o' \in \mathcal{O}_{t_1}$. t_1 is said to be the *largest subtree* of t_2 at node o, denoted as $\widehat{o}$, iff (1) t_1 is the subtree of t_2 with $root(t_1) = o$ and (2) there does not exist an XML tree t_3 with $t_3 \neq t_1$ and $t_3 \neq t_2$ such that t_1 is the subtree of t_3 and t_3 is the subtree of t_2. The set of all XML trees over Σ – neglecting the universe U – is denoted as T_Σ.

A sequence $t_1, ..., t_k$ of XML trees is called an *XML hedge* or simply a *hedge*, and a multiset $\{\!\{t_1, ..., t_k\}\!\}$ of XML trees is called an *XML forest* or simply a *forest*. ε denotes the *empty hedge*.

Definition 10. The set of *XML contexts* over an alphabet Σ ($\xi \notin \Sigma$) is the set $T_{\Sigma\cup\{\xi\}}$ of unranked trees over $\Sigma \cup \{\xi\}$ such that for each tree $t \in T_{\Sigma\cup\{\xi\}}$ exactly one leaf node is labelled with the symbol ξ and has undefined value, and all other nodes in a tree are labelled and valued in the same way as an XML tree defined in Definition 9.

The context with a single node labelled ξ is called the *trivial context* and also denoted as ξ. With contexts we can now define substitution operations that replace a subtree of a tree or context by a new XML tree or context. This leads to the following distinction between four kinds of substitutions:

Tree-to-tree substitution For an XML tree $t_1 \in T_{\Sigma_1}$ with a node $o \in \mathcal{O}_{t_1}$ and an XML tree $t_2 \in T_{\Sigma_2}$ the result $t_1[\widehat{o} \mapsto t_2]$ of substituting t_2 for the subtree rooted at o is an XML tree in $T_{\Sigma_1\cup\Sigma_2}$.

Tree-to-context substitution For an XML tree $t_1 \in T_{\Sigma_1}$ with a node $o \in \mathcal{O}_{t_1}$ the result $t_1[\widehat{o} \mapsto \xi]$ of substituting the trivial context for the subtree rooted at o is an XML context in $T_{\Sigma_1\cup\{\xi\}}$.

Context-to-context substitution For an XML context $c_1 \in T_{\Sigma_1\cup\{\xi\}}$ and an XML context $c_2 \in T_{\Sigma_2\cup\{\xi\}}$ the result $c_1[\xi \mapsto c_2]$ of substituting c_2 for the node labelled by ξ in c_1 is an XML context in $T_{\Sigma_1\cup\Sigma_2\cup\{\xi\}}$.

Context-to-tree substitution For an XML context $c_1 \in T_{\Sigma_1\cup\{\xi\}}$ and an XML tree $t_2 \in T_{\Sigma_2}$ the result $c_1[\xi \mapsto t_2]$ of substituting t_2 for the node labelled by ξ in c_1 is an XML tree in $T_{\Sigma_1\cup\Sigma_2}$.

The correspondence between an XML document and an XML tree is straightforward. Each element of an XML document corresponds to a node of the XML tree, and the subelements of an element define the children nodes of the node corresponding to the element. The nodes for elements are labeled by element names, and character data of an XML document correspond to values of leaves in an XML tree. As our main focus is on structural properties of an XML document, attributes are handled, as if they were subelements, to simplify the discussion.

To provide manipulation operations over XML trees at a level higher than individual nodes and edges, we need some tree constructs to select arbitrary tree portions of interest. For this we provide two selector constructs, which result in subtrees and contexts, respectively. For an XML tree $t = (\gamma_t, \omega_t, \upsilon_t)$, these constructs are defined as follows:

– *context* is a binary, partial function defined on pairs (o_1, o_2) of nodes with $o_i \in \mathcal{O}_t$ $(i = 1, 2)$ such that o_1 is an ancestor of o_2, i.e. $o_1 \prec_c^* o_2$ holds for the transitive closure $\prec_c^*$ of $\prec_c$. We have $context(o_1, o_2) = \widehat{o}_1[\widehat{o}_2 \mapsto \xi]$.

– *subtree* is a unary function defined on $\mathcal{O}_t$. We have $subtree(o) = \widehat{o}$.

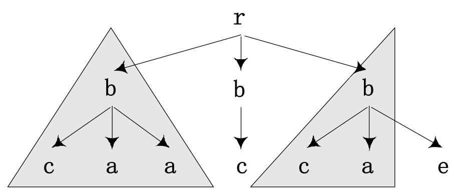

Figure 1: XML tree portions

Example 1. Consider the XML tree shown in Figure 1. Suppose we want to select (1) a subtree, rooted at a node labelled by b, which has exactly two children nodes labelled by a, and (2) a context, defined by a subtree that is rooted at a node labelled by b, in which a subtree with a root labelled by e is substituted by the trivial context ξ.

For (1) using DB-ASM rules we could build the following fragment:

$$\begin{array}{ll} \textbf{forall } x \textbf{ with} & \exists x_1, x_2.((x \prec_c x_1) \wedge (x \prec_c x_2) \wedge \omega_b(x) \wedge \omega_a(x_1) \wedge \omega_a(x_2) \wedge \\ & x_1 \neq x_2 \wedge \forall x_3((x \prec_c x_3) \wedge \omega_a(x_3) \Rightarrow (x_3 = x_1 \vee x_3 = x_2))) \\ \quad\quad\quad\quad \textbf{do} & \\ & t_1 := subtree(x) \; ; \ldots \\ \quad\quad\quad\quad \textbf{enddo} & \end{array}$$

Of course, in doing so the formula in the **with**-clause has to be interpreted using the XML tree t and binding variables to identifiers in the tree domain $\mathcal{O}_t$. Furthermore, $\prec_c$ has to be interpreted by the children relation, and predicants ω_a by the labelling function with label $a \in \Sigma$. Such formulae and their interpretations will be discussed later.

Similarly, for (2) we could use the following fragment of a DB-ASM rule:

$$\begin{array}{lll}\textbf{forall } x & \textbf{with} & (x \prec_c^* x_1) \wedge \omega_b(x) \wedge \omega_e(x_1) \\ & \textbf{do} & \\ & & \quad t_2 := \mathit{context}(x, x_1)\ ;\ \ldots \\ & \textbf{enddo} & \end{array}$$

The resulting tree portions of interest are highlighted in Figure 1. The left shadowed one corresponds to the subtree requested in (1), while the right shadowed one corresponds to the context requested in (2).

The use of the tree selector constructs $\mathit{context}(x_1, x_2)$ and $\mathit{subtree}(x)$ along with algebra operations, which we will introduce in the next subsection allow us to extract and then recombine portions of an existing XML tree to form a new XML tree. This enables the desired level of abstraction for XML database transformations beyond manipulation of nodes and edges.

3.2 Tree Algebra

A many-sorted tree algebra has three sorts: **L** for labels, **H** for hedges, and **C** for contexts, along with a set $\{\iota, \delta, \varsigma, \rho, \kappa, \eta, \sigma\}$ of function symbols with the following signatures:

$$\begin{array}{lll} \iota : \mathbf{L} \times \mathbf{H} \to \mathbf{H} & \delta : \mathbf{L} \times \mathbf{C} \to \mathbf{C} & \varsigma : \mathbf{H} \times \mathbf{C} \to \mathbf{C} \\ \rho : \mathbf{H} \times \mathbf{C} \to \mathbf{C} & \kappa : \mathbf{H} \times \mathbf{H} \to \mathbf{H} & \eta : \mathbf{C} \times \mathbf{H} \to \mathbf{H} \\ \sigma : \mathbf{C} \times \mathbf{C} \to \mathbf{C} & & \end{array}$$

Given a fixed alphabet Σ and two special symbols ε and ξ, the set $\mathcal{T}$ of terms over $\Sigma \cup \{\varepsilon, \xi\}$ comprises label terms, hedge terms, and context terms. That is, $\mathcal{T} = T_{\mathcal{L}} \cup T_{\mathcal{H}} \cup T_{\mathcal{C}}$, where $T_{\mathcal{L}}$, $T_{\mathcal{H}}$ and $T_{\mathcal{C}}$ stand for the sets of terms over sorts **L**, **H** and **C**, respectively. The set of label terms $T_{\mathcal{L}}$ is simply the set of labels, i.e. $T_{\mathcal{L}} = \Sigma$. The set $T_{\mathcal{H}}$ contains the subset $T_{\mathcal{H}}^s$ of tree terms, i.e. we identify trees with hedges of length 1, and is defined by: (1) $\varepsilon \in T_{\mathcal{H}}^s$, (2) $t\langle h\rangle \in T_{\mathcal{H}}^s$ for $t \in \Sigma$ and $h \in T_{\mathcal{H}}$, and (3) $t_1, ..., t_n \in T_{\mathcal{H}}^m$ for $t_i \in T_{\mathcal{H}}^s$ $(i = 1, ..., n)$. The set of context terms $T_{\mathcal{C}}$ is the smallest set with $\xi \in T_{\mathcal{C}}$ and $t\langle t_1, ..., t_n\rangle \in T_{\mathcal{C}}$ for a label $t \in \Sigma$ and terms $t_1, ..., t_n \in T_{\mathcal{H}}^s \cup T_{\mathcal{C}}$ such that exactly one t_i $(i = 1, \ldots, n)$ is a context term in $T_{\mathcal{C}}$.

Trees and contexts have a root, but hedges do not (unless they can be identified with a tree). For hedges of the form of $t\langle\varepsilon\rangle$ we use t as a notational shortcut, if we can avoid confusion with the label term t. The notation $\#t$ denotes the sort of a term t.

Example 2. Let $\Sigma = \{a, b, c\}$, then a, b and c are terms of sort **L**, $a\langle b\langle b\rangle, c\langle a\rangle\rangle$ and $b\langle c\rangle, a\langle a\langle b\rangle, b\rangle$ are terms of sort **H**, and $a\langle a\langle b\rangle, \xi, c\rangle$ is a term of sort **C**.

Intuitively speaking, the functions ι and δ extend hedges and contexts upwards with labels, and ς and ρ incorporate hedges into non-trivial contexts from left or right, respectively, which takes care of the order of subtrees arising in XML as illustrated in Example 3. The function κ denotes hedge juxtaposition, and likewise σ is context composition. The function η denotes context substitution, i.e. substituting ξ in a context with a hedge, which leads to a tree. These functions are illustrated in Figure 2 and formally defined as follows:

$$\iota(a,(t_1,\dots,t_n)) = a\langle t_1,\dots,t_n\rangle \tag{1}$$

(The case $n = 0$ leads to $a\langle\varepsilon\rangle$ on the right hand side.)

$$\delta(a,c) = a\langle c\rangle \tag{2}$$
$$\varsigma((t_1,\dots,t_n),a\langle t'_1,\dots,t'_m\rangle) = a\langle t_1,\dots,t_n,t'_1,\dots,t'_m\rangle \tag{3}$$
$$\rho((t_1,\dots,t_n),a\langle t'_1,\dots,t'_m\rangle) = a\langle t'_1,\dots,t'_m,t_1,\dots,t_n\rangle \tag{4}$$
$$\kappa((t_1,\dots,t_n),(t'_1,\dots,t'_m)) = t_1,\dots,t_n,t'_1,\dots,t'_m \tag{5}$$
$$\eta(c,(t_1,\dots,t_n)) = c[\xi\mapsto t_1,\dots,t_n] \tag{6}$$
$$\sigma(c_1,c_2) = c_1[\xi\mapsto c_2] \tag{7}$$

Example 3. Let us have a look at Figure 3. Given a context term $t_2 = a\langle b,\xi\rangle$ and a hedge term $t_1 = b\langle c\rangle, b\langle c\rangle$, we obtain the context in (i) by $\varsigma(t_1,t_2) = a\langle b\langle c\rangle, b\langle c\rangle, b, \xi\rangle$ and the context in (ii) by $\rho(t_1,t_2) = a\langle b,\xi,b\langle c\rangle,b\langle c\rangle\rangle$.

Proposition 11. *The algebra defined above satisfies the following equations for $t_1, t_2, t_3 \in \mathcal{T}$ (i.e., whenever one of the terms in the equation is defined, the other one is defined, too, and equality holds):*

$$\eta(\sigma(t_1,t_2),t_3) = \eta(t_1,\eta(t_2,t_3)) \tag{8}$$
$$\sigma(\sigma(t_1,t_2),t_3) = \sigma(t_1,\sigma(t_2,t_3)) \tag{9}$$
$$\kappa(\kappa(t_1,t_2),t_3) = \kappa(t_1,\kappa(t_2,t_3)) \tag{10}$$
$$\eta(\delta(t_1,t_2),t_3) = \iota(t_1,\eta(t_2,t_3)) \tag{11}$$
$$\varsigma(t_1,\varsigma(t_2,t_3)) = \varsigma(\kappa(t_1,t_2),t_3) \tag{12}$$
$$\rho(t_1,\rho(t_2,t_3)) = \rho(\kappa(t_2,t_1),t_3) \tag{13}$$
$$\rho(t_3,\varsigma(t_1,t_2)) = \varsigma(t_1,\rho(t_3,t_2)) \tag{14}$$

The tree algebra provides an algebraic approach to manipulate portions of a tree structure at a highly flexible abstraction level, such as subtrees and contexts. In doing so, individual nodes and edges are considered to be special cases of subtrees and contexts. In terms of manipulating tree structures, our tree algebra is powerful enough to express operations provided by the forest algebra

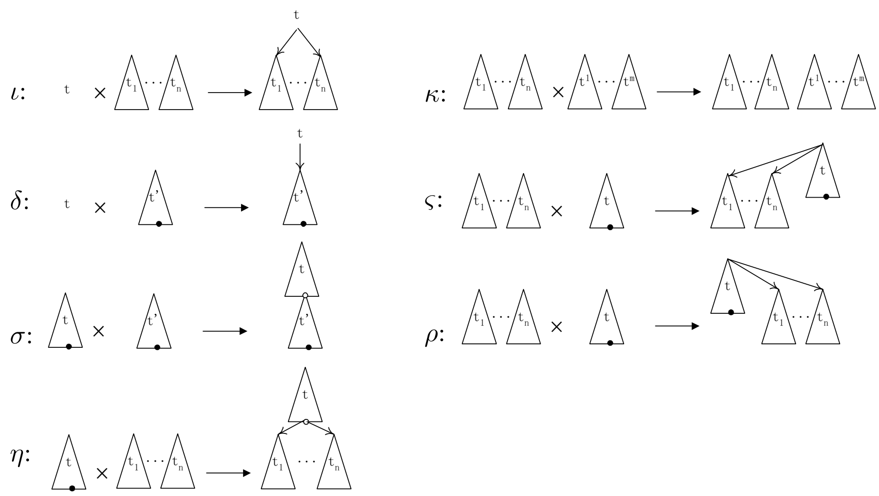

Figure 2: Tree algebra

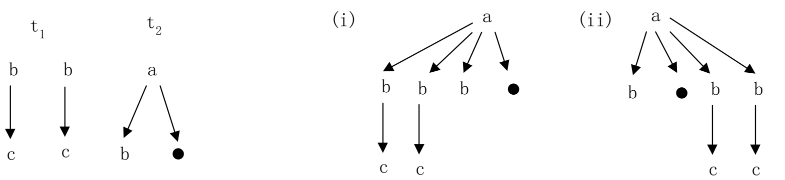

Figure 3: An illustration of functions $\varsigma(t_1, t_2)$ and $\rho(t_1, t_2)$

[Bojanczyk and Walukiewicz, 2007] and Wilke's tree algebra [Wilke, 1996]. For example, the operations $\iota^A(a)$, $\lambda^A(a, t)$ and $\rho^A(a, t)$ in [Wilke, 1996] can be expressed as $\iota(a, \varepsilon)$, $\rho(\delta(a, \xi))$ and $\varsigma(\delta(a, \xi))$, respectively, in our tree algebra.

4 Weak Monadic Second-Order Logic

We first provide a weak MSO logic to navigate within an XML tree by adopting the logic from [Comon et al., 2007] with the restriction that second-order variables can only be quantified over finite sets. The use of MSO logic is motivated by its close correspondence to regular languages, which is known from early work of Büchi [Büchi, 1960]. For XML the navigational core of XPath2.0 capture first-order logic, and some extensions on XPath have been shown to be expressively complete for MSO, or strictly less expressive than MSO such as "regular XPath" [ten Cate and Segoufin, 2008]. In a second step we incorporate the logic into the framework of DB-ASMs.

4.1 Tree Formulae

Let V_{FO} and V_{SO} denote the sets of first- and second-order variables, respectively. Using abstract syntax the formulae of MSO_X are defined by

$$\begin{aligned}\varphi \;\equiv\; & x_1 = x_2 \mid \upsilon(x_1) = \upsilon(x_2) \mid \omega_a(x_1) \mid x \in X \mid x_1 \prec_c x_2 \mid x_1 \prec_s x_2 \mid \\ & \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists x.\varphi \mid \exists X.\varphi \end{aligned} \tag{15}$$

with $x, x_1, ..., x_k \in V_{FO}$, $X \in V_{SO}$, unary function symbols υ and ω_a for all $a \in \Sigma$, and binary predicate symbols $\prec_c$ and $\prec_s$. We interpret formulae of MSO_X for a given XML tree $t = (\gamma_t, \omega_t, \upsilon_t)$ over the set Σ of labels with $\gamma_t = (\mathcal{O}_t, \prec_c^t, \prec_s^t)$. Naturally, the function symbols ω_a and υ are interpreted by the labelling and value functions ω_t and υ_t, respectively, and the predicate symbols $\prec_c$ and $\prec_s$ are interpreted using the children and sibling relations $\prec_c^t$ and $\prec_s^t$, respectively.

Furthermore, we need variable assignments $\zeta : V_{FO} \cup V_{SO} \to \mathcal{O}_t \cup \mathcal{P}(\mathcal{O}_t)$ taking first-order variables x to node identifiers $\zeta(x) \in \mathcal{O}_t$, and second-order variables X to sets of node identifiers $\zeta(X) \subseteq \mathcal{O}_t$. As usual $\zeta[x \mapsto o]$ (and $\zeta[X \mapsto O]$, respectively) denote the modified variable assignment, which equals ζ on all variables except the first-order variable x (or the second-order variable X, respectively), for which we have $\zeta[x \mapsto o](x) = o$ (and $\zeta[X \mapsto O](X) = O$, respectively). For the XML tree t and a variable assignment ζ we obtain the interpretation $val_{t,\zeta}$ on terms and formulae as follows. Terms are either variables x, X or have the form $\upsilon(x)$, thus are interpreted as $val_{t,\zeta}(x) = \zeta(x)$, $val_{t,\zeta}(X) = \zeta(X)$, and $val_{t,\zeta}(\upsilon(x)) = \upsilon_t(\zeta(x))$. For formulae φ we use $[\![\varphi]\!]_{S,\zeta}$ to denote its interpretation by a truth value, and obtain:

– $[\![\tau_1 = \tau_2]\!]_{t,\zeta} = true$ iff $val_{t,\zeta}(\tau_1) = val_{t,\zeta}(\tau_2)$ holds for the terms τ_1 and τ_2,

– $[\![\omega_a(x)]\!]_{t,\zeta} = true$ holds iff $\omega_t(val_{t,\zeta}(x)) = a$,

– $[\![x \in X]\!]_{t,\zeta} = true$ iff $val_{t,\zeta}(x) \in val_{t,\zeta}(X)$,

– $[\![\neg\varphi]\!]_{t,\zeta} = true$ iff $[\![\varphi]\!]_{t,\zeta} = false$,

– $[\![\varphi_1 \wedge \varphi_2]\!]_{t,\zeta} = true$ iff $[\![\varphi_1]\!]_{t,\zeta} = true$ and $[\![\varphi_2]\!]_{t,\zeta} = true$,

– $[\![\exists x.\varphi]\!]_{t,\zeta} = true$ iff $[\![\varphi]\!]_{t,\zeta[x\mapsto o]} = true$ holds for some $o \in \mathcal{O}$,

– $[\![\exists X.\varphi]\!]_{t,\zeta} = true$ iff $[\![\varphi]\!]_{t,\zeta[X \mapsto O]} = true$ for some finite $O \subseteq \mathcal{O}$,

– $[\![x_1 \prec_c x_2]\!]_{t,\zeta} = true$ iff $val_{t,\zeta}(x_2)$ is a child node of $val_{t,\zeta}(x_1)$ in t, i.e. $val_{t,\zeta}(x_1) \prec_c^t val_{t,\zeta}(x_2)$ holds, and

– $[\![x_1 \prec_s x_2]\!]_{t,\zeta} = true$ iff $val_{t,\zeta}(x_2)$ is the next sibling to the right of $val_{t,\zeta}(x_1)$ in t, i.e. $val_{t,\zeta}(x_1) \prec_s^t val_{t,\zeta}(x_2)$ holds.

The syntax of MSO_X can be enriched by adding $\varphi_1 \vee \varphi_2$, $\forall x.\varphi$, $\forall X.\varphi$, $\varphi_1 \Rightarrow \varphi_2$, $\varphi_1 \Leftrightarrow \varphi_2$ as abbreviations as usual. Likewise, the definition of bound and free variables of MSO_X formulae is also standard. The notation $fr(\varphi)$ refers to the set of free variables of the formula φ.

4.2 Formulae in DB-ASMs with Trees

When using XML trees as values in the base set of a state of DB-ASMs the logic MSO_X is not sufficient, as we have to take more than one such tree into account. Fortunately, this only requires an extension for the *atomic formulae*, i.e. those formulae in the first line of (15) above.

Regarding the function and predicate symbols for XML trees (i.e. υ, ω_a, $\prec_c$ and $\prec_s$) we have to add the XML tree as an additional argument, respectively, and omitting the index of ω_a in order to cope with different sets of labels. Hence, we obtain function symbols υ, ω, $\prec_c$ and $\prec_s$ of arity 2, 2, 3 and 3, respectively. Informally, $\upsilon(x, y)$ denotes the value at leaf node y in the tree x, $\omega(x, y)$ denotes the label of node y in the tree x, $y \prec_c (x)z$ denotes the (truth) value of $y \prec_c z$ in the tree x, and $y \prec_s (x)z$ denotes the (truth) value of $y \prec_s z$ in the tree x.

Together with other function symbols defined as part of the background signature and variables we can build the set of terms. For this, variables now have to be sorted including sorts for node identifiers and for sets of node identifiers above, plus labels, hedges, contexts, etc. The set of atomic formulae then contains formulae of the form $\tau_1 = \tau_2$ and $x \in X$ with terms τ_i $(i = 1, 2)$ of the same sort, x of sort "node identifier", and X is of sort "set of node identifier". The set of formulae can be built in the usual way using negation, conjunction and existential quantification plus the usual shortcuts.

5 XML Database Transformations

In this section we develop a general notion of XML database transformation. As outlined in Section 2, the customisation for a particular data model requires the definition of appropriate backgrounds. We approach this in two steps: first defining tree background classes that build upon the tree algebra and the weak

MSO logic, and secondly adding tree type schemes to capture schema information. Any object satisfying Postulates 1 - 5 with a tree-based background as customised in this section is defined to be an *XML database transformation*.

Using the same tree-based backgrounds we obtain a computational model for XML database transformations on the basis of DB-ASMs. Then following Theorems 6 and 7 we obtain that DB-ASMs with tree-based backgrounds capture exactly all XML database transformations.

5.1 Tree Background Classes

The background for an XML database transformation describes all available tree structures that may be used within an XML database transformation. Such backgrounds are called *tree-based backgrounds*. To define tree-based backgrounds, we first require a background signature specific to XML database transformations.

Definition 12. A *tree-based background signature* V contains at least

- constructor symbols for finite tuple $[\cdot]$, set $\{\cdot\}$, hedge $, \dots,$ and tree $\langle\cdot\rangle$,
- function symbols *root*, $\prec_c, \prec_s, \omega$ and υ to define XML trees,
- function symbols *context* and *subtree* for the selection of tree portions,
- function symbols ε (empty hedge) and ξ (trivial context),
- function symbols ι, δ, ς, ρ, κ, η and σ as defined by the tree algebra, and
- all constructor and function symbols defined in the background postulate.

In the sense of [Börger and Stärk, 2003] all function symbols in a background signature are *static*, i.e. their interpretation is fixed and does not permit updates. On the other hand, the interpretation of functions symbols defined in the signature of underlying states may be updated. Next we need a universe of elements, for which we require a set $\mathcal{D}$ of *base domains* as in Definition 2. Let us fix a set Σ of labels and a tree domain $\mathcal{O}$ (i.e. a set of node identifiers). Then Σ defines one of the base domains, and $\mathcal{O}$ defines a *tree universe* as well as a set of *contexts*.

Definition 13. The *tree universe* over $\mathcal{O}$ is the smallest set $HFT(\mathcal{O})$ of all finite trees over $\mathcal{O}$ with $t\langle t1, \dots, t_n\rangle \in HFT(\mathcal{O})$ for $t \in \mathcal{O}$ and finite trees $t_1, \dots, t_n \in HFT(\mathcal{O})$.

For $n = 0$ in this definition we obtain trivial trees $t\langle\rangle \in HFT(\mathcal{O})$. If we identify $t\langle\rangle$ with t, we have in fact $\mathcal{O} \subseteq HFT(\mathcal{O})$. Each finite tree in a tree universe is indeed a tree skeleton, i.e. none of nodes are labelled or assigned with values. By the Kuratowski encoding [Van Den Bussche et al., 1997], hereditarily finite trees in $HFT(\mathcal{O})$ can be viewed as a special kind of hereditarily finite lists and thus can be interpreted as hereditarily finite sets. Similarly, we can define a set of *contexts* over $\mathcal{O}$.

Definition 14. The *set of contexts* over $\mathcal{O}$ is the smallest set $HFT(\mathcal{O}, \xi)$ with $\xi \in HFT(\mathcal{O}, \xi)$ and $t\langle t_1, \ldots, t_n\rangle \in HFT(\mathcal{O}, \xi)$, where $t \in \mathcal{O}$, exactly one of $t_1, \ldots, t_n$ is a context (i.e., $t_i \in HFT(\mathcal{O}, \xi)$) and the others are finite trees (i.e., $t_j \in HFT(\mathcal{O})$ for $j = 1, ..., n$ and $j \neq i$).

In addition to the set Σ of labels, the tree universe $HFT(\mathcal{O})$ and the set of contexts $HFT(\mathcal{O}, \xi)$ define two more base domains.

Definition 15. A *tree background class* consists of a background signature V, a set $\mathcal{D}$ of base domains with $\Sigma \in \mathcal{D}$, $HFT(\mathcal{O}) \in \mathcal{D}$ and $HFT(\mathcal{O}, \xi) \in \mathcal{D}$, and a *tree background structure* over V and $\mathcal{D}$, which is a structure consisting of a universe $U = \bigcup \mathcal{D}'$ with $\mathcal{D}'$ as defined in Definition 2 and the interpretation of function symbols in V over U.

Remark. The set of algebraic equations discussed in Proposition 11 holds in every tree background class.

5.2 Tree Type Schemes

XML documents may be associated with a schema. Following the discussion in [Murata et al., 2005] we concentrate on (Extended Document Type Definitions) EDTDs, as they subsume many XML schema formalisms.

Given an alphabet Σ, the set of regular languages over Σ is denoted as $reg(\Sigma)$. We first recall the definition of DTDs (as *labelled ordered tree object type definition*) in [Papakonstantinou and Vianu, 2000].

Definition 16. A *document type definition* (DTD) consists of an alphabet Σ, a root $r \in \Sigma$ and a mapping $\beta : \Sigma \to reg(\Sigma)$ assigning to each $a \in \Sigma$ a regular language over Σ.

While such DTDs provide some schema information, they cannot express all desirable properties of XML documents. For instance, the DTD would not allow us to separate the type of an element from its label name. Extended DTDs as introduced in [Papakonstantinou and Vianu, 2000] (as *specialised labelled ordered tree object type definition*) take care of this problem.

Definition 17. An *Extended Document Type Definition* (EDTD) consists of a DTD (Σ', r, β) and a mapping $\mu : \Sigma' \to \Sigma$ with another alphabet $\Sigma \subseteq \Sigma'$.

We can use the elements in Σ' to fine-tune the desired structure of XML document adhering to a given EDTD, while $\mu(a)$ defines the actual tag that is to be used. We adopt the notational convention to write a^b for elements in Σ' with $\mu(a^b) = a \in \Sigma$. Normally, the superscript b of a^b is then called the *type* of the element, but to simplify the development in the next subsection we refer to $a^b \in \Sigma'$ as the type. If $\mu^{-1}(a)$ contains only one element, we omit the superscript and assume that μ maps a to itself.

Let t be an XML tree over Σ', $d_1 = (\Sigma', r, \beta)$ a DTD and $d_2 = (\Sigma, d_1, \mu)$ an EDTD, then t is said to *satisfy* d_1 if the root of t is labelled as r, and $a_1 \dots a_n \in \beta(o)$ for each $o \in \mathcal{O}_t$ labelled by a with children nodes labelled by $a_1, \dots, a_n$. The tree t is said to *satisfy* d_2 if there exists a tree t' over Σ' satisfying d_1 such that $\mu(t') = t$. Here we applied μ to a whole XML tree, which must be understood as the canonical extension from node labels to trees.

As the labels of nodes in an XML tree are insufficient to express typing, we have to provide *types* in addition. For an EDTD $d_2 = (\Sigma, d_1, \mu)$ with $d_1 = (\Sigma', r, \beta)$ the set of *type names* associated with XML trees satisfying d_2 is Σ'. Each type name $a \in \Sigma'$ is associated with a regular expression over Σ'.

Definition 18. Let $\mathcal{D}$ be a set of base domains with $\Sigma \in \mathcal{D}$, $HFT(\mathcal{O}) \in \mathcal{D}$ and $HFT(\mathcal{O}, \xi) \in \mathcal{D}$. A *tree type scheme* over $\mathcal{D}$ is a triple $(\Sigma', \Gamma, \tilde{\Gamma})$ consisting of a finite, non-empty set Σ' of type names, a *type name assignment* $\Gamma : \mathcal{O} \to \Sigma'$ that associates each $o \in \mathcal{O}$ with a type name $\Gamma(o)$ in $\Sigma^{'}$ and a *type expression assignment* $\tilde{\Gamma}$ that associates each $a \in \Sigma'$ with a regular expression $\tilde{\Gamma}(a)$ over Σ'.

We write $a(\tau)$ for a type name $a \in \Sigma'$ together with its type expression $\tau = \tilde{\Gamma}(a)$. We interpret $a(\tau)$ by the set of trees $\{o\langle h\rangle \mid o \in \mathcal{O}, \omega(o) = \mu(a), h \in [\![\tau]\!]\}$, in which $[\![\tau]\!]$ denotes the interpretation of the τ by a set of hedges defined by:

$$[\![\emptyset]\!] = \emptyset \tag{16}$$

$$[\![\epsilon]\!] = \{\varepsilon\} \tag{17}$$

$$[\![a]\!] = \{o\langle\rangle \mid o \in \mathcal{O} \text{ and } \omega_t(o) = \mu(a)\} \tag{18}$$

$$[\![\tau_1 \mid \tau_2]\!] = [\![\tau_1]\!] \cup [\![\tau_2]\!] \tag{19}$$

$$[\![\tau_1\tau_2]\!] = \{\kappa(h_1, h_2) \mid h_i \in [\tau_i]\!] \text{ for } i = 1, 2\} \tag{20}$$

$$[\![\tau^*]\!] = \{\kappa(h_1, \kappa(h_2, \dots, \kappa(h_{n-1}, h_n) \dots)) \mid n \in \mathbb{N}, h_i \in [\tau]\!]\} \tag{21}$$

Most database queries and updates require a pair of database schemata to restrain input and output databases, respectively. Therefore, a tree-based background should provide two sets of tree type schemes that are associated with initial and final states, respectively.

Definition 19. A *tree-based background* is a pair $(\mathfrak{K}, \mathfrak{T})$ consisting of a *tree background class* $\mathfrak{K}$ and a set $\mathfrak{T}$ of *tree type schemes*.

5.3 DB-ASMs for XML Database Transformations

Following common practice we treat XML database schemata and instances separately.

Definition 20. An *XML database schema* is a finite, non-empty set $\mathbf{S}$ of EDTDs, while an *XML database instance* over $\mathbf{S}$ is a finite, non-empty set $\mathbf{I}$ of XML trees such that the following two conditions must be both satisfied: (1) each XML tree $t \in \mathbf{I}$ is associated with an EDTD $d \in \mathbf{S}$ such that $t \in sat(d)$; (2) each EDTD $d \in \mathbf{S}$ has at least one XML tree $t \in \mathbf{I}$ such that $t \in sat(d)$.

Informally speaking, an XML database is a finite, unordered collection of XML trees, each of which should be associated with a tree name uniquely identifiable in a state, corresponding to the name of an XML document.

Definition 21. An *XML database transformation* is a database transformation with a tree-based background $(\mathfrak{K}, \mathfrak{T})$, satisfying the conditions:

– the sub-signature of the database part of a state contains a finite, non-empty set of unary (and dynamic) function symbols representing tree names, and

– the database part of each initial and final state is an XML database instance in which an EDTD is defined by the tree type schemes in $\mathfrak{T}$.

In essence, an XML database transformation describes a process starting from some XML trees constrained by an input schema, processing them in accordance with database postulates defined in Section 2 and terminating with some XML trees that are constrained by an output schema. The following theorem is a direct consequence of Theorems 6 and 7.

Theorem 22. *DB-ASMs with tree-based backgrounds* $(\mathfrak{K}, \mathfrak{T})$*, in which initial and final states have their database parts as being XML database instances over XML database schemata defined by the tree type schemes in* $\mathfrak{T}$*, capture exactly all XML database transformations.*

6 XML Machines

In this section we present an alternative computational model for XML, which we call *XML machines*. The reason is that DB-ASMs with tree-based backgrounds do not exploit the weak MSO logic from Section 4. The fact that it nonetheless captures all XML database transformation is mainly due to the power of the DB-ASM rules. Using MSO_X, however, permits more sophisticated navigation over XML trees. Therefore, compared with DB-ASMs with tree-based backgrounds, XML machines have two extensions: (1) the incorporation of MSO_X formulae in forall- and choice-rules; (2) an added partial update rule that is added for convenience, although it does not add any additional expressive power.

6.1 Extended Rules

As XML trees are unranked, a node may have an unbounded number of children nodes. To access all of them we have two choices. One possibility is to process the children nodes sequentially one by one using an unbounded loop. The alternative is to execute in parallel an unbounded number of processes, one for each child. The former one requires only standard total updates, whereas the latter one involves partial updates. For the sake of simplicity and naturalness of the computation model the latter becomes our choice.

Using an unbounded number of parallel processes, we need an update operator to merge two hedges into one. Using hedge juxtaposition by means of the algebra operation κ is one possibility, but the order may not be the desired one. Therefore, we consider hedges as forests by ignoring the order and using simply forest union $\cup$. The non-determinism provided by the choice-rules can be exploited for this, i.e. choose any order for the resulting forest to turn it back into a hedge. In doing so, $\cup$ is defined over sort $\mathbf{H}$ such that $\cup : \mathbf{H}^2 \to \mathbf{H}$, and becomes part of the background.

With these preliminary remarks we can now define *MSO*-rules, in which the formula φ always refers to a MSO_X formula.

Definition 23. The set $\mathbf{R}$ of *MSO-rules* over a signature $\Sigma = \Sigma_{db} \cup \Sigma_a \cup \{f_1, \dots, f_\ell\}$ and a tree-based background is defined as follows:

– If t_1 is a term over Σ, and t_2 is a location in Σ such that $\#t_2 = \#t_1$, then $t_2 := t_1$ is a rule r in $\mathbf{R}$ called *assignment rule* with $fr(r) = var(t_1) \cup var(t_2)$, where $var(t_i)$ is the set of variables occurring in the term t_i $(i = 1, 2)$.

– If t_1 is a term over Σ, t_2 is a location in Σ and $\cup$ is a binary operator such that $\#t_2 = \#t_1$ and $\cup : \#t_1^2 \to \#t_2$, then $t_2 \Leftarrow^{\cup} t_1$ is a rule r in $\mathbf{R}$ called *partial assignment rule* with $fr(r) = var(t_1) \cup var(t_2)$, where $var(t_i)$ is the set of variables occurring in the term t_i $(i = 1, 2)$.

– If φ is a formula and $r' \in \mathbf{R}$ is an MSO-rule, then **if** φ **then** r' **endif** is a rule r in $\mathbf{R}$ called *conditional rule* with $fr(r) = fr(\varphi) \cup fr(r')$.

– If φ is a formula with only database variables, $\{x_1, \dots, x_k, X_1, \dots, X_m\} \subseteq fr(\varphi)$ and $r' \in \mathbf{R}$ is an MSO-rule, then **forall** $x_1, \dots, x_k, X_1, \dots, X_m$ **with** φ **do** r' **enddo** is a rule r in $\mathbf{R}$ called *forall rule* with $fr(r) = fr(r') \cup fr(\varphi) - \{x_1, \dots, x_k, X_1, \dots, X_m\}$.

– If r_1, r_2 are rules in $\mathbf{R}$, then **par** r_1 r_2 **par** is a rule r in $\mathbf{R}$, called *parallel rule* with $fr(r) = fr(r_1) \cup fr(r_2)$.

– If φ is a formula with only database variables, $\{x_1, \dots, x_k, X_1, \dots, X_m\} \subseteq fr(\varphi)$ and $r' \in \mathbf{R}$ is an MSO-rule, then **choose** $x_1, \dots, x_k, X_1, \dots, X_m$ **with**

φ **do** r' **enddo** is an MSO-rule r in $\mathbf{R}$ called *choice rule* with $fr(r) = fr(r') \cup fr(\varphi) - \{x_1, \dots, x_k, X_1, \dots, X_m\}$.

– If r_1, r_2 are rules in $\mathbf{R}$, then **seq** r_1 r_2 **seq** is a rule r in $\mathbf{R}$, called *sequence rule* with $fr(r) = fr(r_1) \cup fr(r_2)$.

– If r' is a rule in $\mathbf{R}$ and ϑ is a location function that assigns location operators ϱ to terms t with $var(t) \subseteq fr(r')$, then **let** $\vartheta(t) = \varrho$ **in** r' **endlet** is a rule r in $\mathbf{R}$ called *let rule* with $fr(r) = fr(r')$.

The definition of sets of update sets $\Delta(r, S)$ for a closed MSO-rule r with respect to a state S is again straightforward [Schewe and Wang, 2010]. We only explain the non-standard case of the partial assignment rule. Let r be a partial assignment rule $t_2 \leftleftharpoons^{\cup} t_1$, S be a state over Σ and ζ a variable assignment for $fr(r)$. We then obtain $\Delta(r, S, \zeta) = \{\{(\ell, a, \cup)\}\}$ with $\ell = val_{S,\zeta}(t_2)$ and $a = val_{S,\zeta}(t_1)$, i.e. we obtain a single update set with a single partial assignment to the location ℓ. As the rule r will appear as part of a complex MSO-rule without free variables, the variable assignment ζ will be determined by the context, and the partial update will become an element of larger update sets Δ. Then, for a state S, the value of location ℓ in the successor state $S + \Delta$ becomes $val_{S+\Delta}(\ell) = val_S(\ell) \cup \bigcup_{(\ell,v,\cup)\in\Delta} v$, if the value on the right hand side is defined unambiguously, otherwise $val_{S+\Delta}(\ell)$ is undefined.

Example 4. Consider the XML tree in Figure 4 and assume it is assigned to the variable (tree name) t_{exa}. The following MSO-rule constructs the XML tree in (ii) from subtrees of the given XML tree in (i), each of which is rooted at a node labeled as b with at least two descendant nodes labeled as a and c, respectively.

$$
\begin{array}{l}
t_1 := \epsilon \;; \\
\textbf{forall} \quad x \quad \textbf{with} \; \omega(t_{exa}, x) = b \wedge \exists X.(\forall x_1, x_2.((x_1 \in X \wedge \prec_c (t_{exa}, x_1, x_2) \\
\qquad\qquad\qquad\qquad \Rightarrow x_2 \in X) \wedge \forall x_1.(\prec_c (t_{exa}, x, x_1) \Rightarrow x_1 \in X))) \\
\qquad\qquad\qquad\qquad \wedge \exists y, z.(\omega(t_{exa}, y) = c \wedge \omega(t_{exa}, z) = a \wedge y \in X \wedge z \in X) \\
\qquad \textbf{do} \quad t_1 \leftleftharpoons^{\cup} subtree(t_{exa}, x) \quad \textbf{enddo}; \\
output := \iota(d, t_1)
\end{array}
$$

Definition 24. An *XML Machine* (XMLM) M over signature Σ with a tree-based background $(\mathfrak{K}, \mathfrak{T})$ consists of

– a set $\mathcal{S}_M$ of states over Σ satisfying the abstract state postulate and closed under isomorphisms,

– non-empty subsets $\mathcal{I}_M \subseteq \mathcal{S}_M$ of initial states, and $\mathcal{F}_M \subseteq \mathcal{S}_M$ of final states, both also closed under isomorphisms and satisfying the conditions of Definition 20 with EDTDs defined by the tree type schemes in $\mathfrak{T}$,

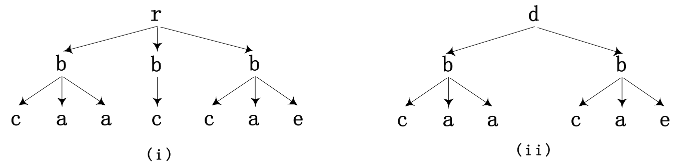

Figure 4: An XML tree and the result of tree operations

– a program π_M defined by a closed MSO-rule r over Σ, and

– a binary relation τ_M over $\mathcal{S}_M$ determined by π_M such that $\{S_{i+1} \mid (S_i, S_{i+1}) \in \tau_M\} = \{S_i + \Delta \mid \Delta \in \Delta(\pi_M, S_i)\}$ holds.

6.2 Behavioural Equivalence

In this subsection we will show the behavioural equivalence between DB-ASMs with tree-based backgrounds and XMLMs.

Theorem 25. *The XML Machines with a tree-based background* $(\mathfrak{K}, \mathfrak{T})$ *capture exactly all XML database transformations with the same background.*

Proof. According to Theorem 22 each XML database transformation can be represented by a behaviourally equivalent DB-ASM with the same tree-based background. As DB-ASMs differ from XMLMs only by the fact that DB-ASM rules are more restrictive than MSO-rules (they do not permit MSO_X formulae in forall- and choice-rules), such a DB-ASM is in fact also an XMLM.

Thus, it suffices to show that XMLMs satisfy Postulates 1 - 5 for XML database transformations. The first three of these postulates are already captured by the definitions of XMLMs and tree-based background, so we have to consider only the bounded exploration and bounded non-determinism postulates.

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

We use this to define a bounded exploration witness $T_{witness}$. If t_i is ground, we add the ground access term t_i to $T_{witness}$. If t_i is not ground, then the corresponding assignment rule must appear within the scope of forall and choice rules introducing the database variables in t_i, as r is closed. Thus, variables in t_i are bound by a formula φ, i.e. for $fr(t_i) = \{x_1, \dots, x_k\}$ the relevant variable assignments are $\zeta = \{x_1 \mapsto b_1, \dots, x_k \mapsto b_k\}$ with $val_{S,\zeta}(\varphi) = true$. Bringing φ

into a form that only uses conjunction, negation and existential quantification, we can extract a set of access terms $\{(\beta_1, \alpha_1), \dots, (\beta_\ell, \alpha_\ell)\}$ such that if S and S' coincide on these access terms, they will also coincide on the formula φ. This is possible, as we evaluate access terms by sets, so conjunction corresponds to union, existential quantification to projection, and negation to building the (finite) complement. We add all the access terms $(\beta_1, \alpha_1), \dots, (\beta_\ell, \alpha_\ell)$ to $T_{witness}$.

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

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

Then by construction, if S and S' coincide on $T_{witness}$, we obtain $\Delta(r, S) = \Delta(r, S')$. As there are only finitely many assignments rules within r and only finitely many choice and forall rules defining the variables in such assignments, the set $T_{witness}$ of access terms must be finite, i.e. r satisfies the bounded exploration postulate.

Regarding bounded non-determinism, as a choice rule **choose** $x_1, \dots, x_k$ **with** φ **do** r' **enddo** only allows the use of database variables $x_1, \dots, x_k$ and these variables are restricted to range only over the database part of a state, the presence of non-determinism in one-step transitions can always correspond to a non-empty set of access terms from the choice rule containing these database variables.

7 Conclusions and Future Work

In this article we continued our research on foundations of database transformations exploiting the theory of Abstract State Machines. In [Schewe and Wang, 2010] we developed a theoretical framework for database transformations in general, which are defined by five intuitive postulates and exactly charactericterised by DB-ASMs, a variant of Abstract State Machines. We argued that specific data model requirements are captured by background classes, while in general only minimum requirements for such backgrounds are postulated.

We now made the backgrounds explicit for the case of XML database transformations. For this we had to adopt hereditarily finite trees, operators of a hedge algebra, weak monadic second-order logic, and extended document type definitions. Consequently, DB-ASMs with backgrounds defined in this particular way capture XML database transformations. In doing so, we actually proved that capturing specific characteristics of data models by background classes is the appropriate way to develop the theory, whereas in earlier work [Wang and Schewe, 2007] we tried to approach the problem by direct manipulation of the notion of state dealing with higher-order structures instead of first-order ones.

We did, however, go one step further, and defined an alternative and more elegant computational model for XML database transformations, which directly incorporates weak MSO formulae in forall and choice rules. This leads to so-called XML machines. Due to the intuition behind the postulates it should come as no surprise that the two computation models are in fact equivalent. We mainly had to show that XML machines satisfy the postulates of database transformations with tree-based backgrounds.

This research is part of a larger research agenda devoted to studying logical foundations of database transformations, in particular in connection with tree-based databases. The next obvious step is to define a logic that permits reasoning about database transformations that are specified by DB-ASMs or in the case of XML by equivalent XML machines. First steps in this direction have been made in [Wang and Schewe, 2008]. Another line of research we intend to explore is the connection between DB-ASMs with specific backgrounds such as the tree-based backgrounds used in the context of XML and restrictions to the notion of state. In particular, we are interested in states that can be recognised by certain types of automata. This may lead to establishing links between ASMs, database theory and particular structures, e.g. automatic structures.

References

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases.* Addison-Wesley, Reading, Massachusetts.

[Abiteboul and Kanellakis, 1998] Abiteboul, S. and Kanellakis, P. C. (1998). Object identity as a query language primitive. *J. ACM*, 45(5):798–842.

[Abiteboul and Vianu, 1988] Abiteboul, S. and Vianu, V. (1988). Datalog extensions for database queries and updates. Technical Report RR-0900.

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

[Blass and Gurevich, 2000] Blass, A. and Gurevich, Y. (2000). Background, reserve, and gandy machines. volume 1862 of *LNCS*, pages 1–17. Springer-Verlag.

[Blass and Gurevich, 2007] Blass, A. and Gurevich, Y. (2007). Background of computation. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 92.

[Blass et al., 2002] Blass, A., Gurevich, Y., and Van den Bussche, J. (2002). Abstract state machines and computationally complete query languages. *Information and Computation*, 174(1):20–36.

[Bojanczyk and Walukiewicz, 2007] Bojanczyk, M. and Walukiewicz, I. (2007). Forest algebras. In *Automata and Logic: History and Perspectives*, pages 107–132. Amsterdam University Press.

[Börger and Stärk, 2003] Börger, E. and Stärk, R. (2003). *Abstract State Machines.* Springer-Verlag, Berlin Heidelberg New York.

[Büchi, 1960] Büchi, J. R. (1960). Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6(1-6):66–92.

[Chamberlin et al., 2006] Chamberlin, D., Carey, M., Florescu, D., Kossmann, D., and Robie, J. (2006). XQueryP: Programming with XQuery. In *Proceedings of the third International Workshop on XQuery Implementation, Experience, and Perspectives.*

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

[Comon et al., 2007] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (2007). Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata, release October 2007.

[Don Chamberlin and Siméon, 2008] Don Chamberlin, Daniela Florescu, J. M. J. R. and Siméon, J., editors (2008). *XQuery Update Facility 1.0.* W3C Recommendation. W3C.

[Doner, 1970] Doner, J. (1970). Tree acceptors and some of their applications. *Journal of Computer and Systems Science*, 4(5):406–451.

[Ebbinghaus and Flum, 1999] Ebbinghaus, H.-D. and Flum, J. (1999). *Finite Model Theory.* Springer-Verlag.

[Ghelli et al., 2006] Ghelli, G., Re, C., and Siméon, J. (2006). XQuery!: An XML query language with side effects. In *EDBT Workshops*, pages 178–191.

[Grädel and Gurevich, 1998] Grädel, E. and Gurevich, Y. (1998). Metafinite model theory. *Information and Computation*, 140(1):26–81.

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

[Gurevich et al., 1997] Gurevich, J., Sopokar, N., and Wallace, C. (1997). Formalizing database recovery. *JUCS*, 3(4):320–340.

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

[Kirchberg et al., 2009] Kirchberg, M., Schewe, K.-D., and Zhao, J. (2009). Using abstract state machines for the design of multi-level transaction schedulers. volume 5115 of *LNCS Festschrift*. Springer-Verlag. to appear.

[Murata et al., 2005] Murata, M., Lee, D., Mani, M., and Kawaguchi, K. (2005). Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology*, 5(4):660–704.

[Papakonstantinou and Vianu, 2000] Papakonstantinou, Y. and Vianu, V. (2000). DTD inference for views of XML data. In *Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2000)*, pages 35–46. ACM.

[Schewe and Wang, 2010] Schewe, K.-D. and Wang, Q. (2010). A customised ASM thesis for database transformations. *Acta Cybernetica*. to appear.

[Sur et al., 2004] Sur, G., Hammer, J., and Siméon, J. (2004). An XQuery-based language for processing updates in XML. *Proceedings of Programming Language Technologies for XML*.

[ten Cate and Segoufin, 2008] ten Cate, B. and Segoufin, L. (2008). Xpath, transitive closure logic, and nested tree walking automata. In *PODS '08: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 251–260, New York, NY, USA. ACM.

[Thatcher and Wright, 1968] Thatcher, J. W. and Wright, J. B. (1968). Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81.

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

[Van den Bussche and Van Gucht, 1992] Van den Bussche, J. and Van Gucht, D. (1992). Semi-determinism (extended abstract). In *PODS '92: Proceedings* *of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 191–201, New York, NY, USA. ACM Press.

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

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

[Wang and Ferrarotti, 2009] Wang, Q. and Ferrarotti, F. A. (2009). Xml machines. In *ER '09: Proceedings of the ER 2009 Workshops (CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS) on Advances in Conceptual Modeling - Challenging Perspectives*, pages 95–104, Berlin, Heidelberg. Springer-Verlag.

[Wang and Schewe, 2007] Wang, Q. and Schewe, K.-D. (2007). Axiomatization of database transformations. In *Proceedings of the 14th International ASM Workshop (ASM 2007)*, University of Agder, Norway.

[Wang and Schewe, 2008] Wang, Q. and Schewe, K.-D. (2008). Towards a logic for abstract metafinite state machines. volume 4932 of *LNCS*, pages 365–380.

[Wilke, 1996] Wilke, T. (1996). An algebraic characterization of frontier testable tree languages. *Theoretical Computer Science*, 154(1):85–106.

[Zhao et al., 2009] Zhao, J., Schewe, K.-D., and Köhler, H. (2009). Dynamic data warehouse design with abstract state machines. *JUCS*, 15(1):355–397.