XML Machines

Qing Wang[1] and Flavio A. Ferrarotti[2]

[1] University of Otago, Dunedin, New Zealand
qing.wang@otago.ac.nz
[2] University of Santiago of Chile and Yahoo! Research Latin America, Santiago of Chile
flavio@dcc.uchile.cl

Abstract. In order to capture the dynamics of XML databases a general model of tree-based database transformations is required. In this paper such an abstract computational model is presented, which brings together ideas from Abstract State Machines and monadic second-order logic. The model captures all XML database transformations.

1 Introduction

For a long time already database transformations as a unifying umbrella for queries and updates have been the focus of the database research community [11]. The logical foundations of queries have always been a central focus of interest for database theoreticians.

The sequential ASM thesis [6] defines sequential algorithms by a set of intuitive postulates. Gurevich shows that sequential Abstract State Machines capture these classes of algorithms. In our previous work [8] 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 Abstract Database Transformation Machines (ADTMs).

While the characterisation of database transformations is done without any reference to a particular data model, the presence of backgrounds [1] supposedly enables tailoring the characterisation to any data model of interest. That is, the equivalence between the postulates and ADTMs holds with respect to a fixed background. In [9] we defined tree-based backgrounds and in doing so demonstrated how to adapt our general results to XML database transformations.

The disadvantage of the ADTM-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 logics [7]. Weak monadic second-order logics (MSO) are linked to regular tree languages [5,10] in the sense that a set of trees is regular iff it is in weak MSO with k successors.

Therefore, in this paper we define an alternative model of computation for XML database transformations, which exploits weak MSO. Pragmatically speaking the use of weak MSO formulae in forall and choice rules permits more flexible

C.A. Heuser and G. Pernul (Eds.): ER 2009 Workshops, LNCS 5833, pp. 95–104, 2009.

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 ADTM model with tree-based backgrounds. As our main result in [8] states that already ADTMs 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 ADTMs with tree-based backgrounds. For the proof we simply have to show that XML machines satisfy the postulates, i.e. in a sense the hard part of the proof is already captured by the main characterisation theorem in [8].

2 XML Trees

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

Definition 1. 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: (i) 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 (ii) whenever $o_1 \prec_s o_2$ holds, then there is some $o \in \mathcal{O}$ with $o \prec_c o_i$ for $i = 1, 2$.

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 2. An *XML tree* t (over the set of labels Σ with values in the universe U corresponding to the family $\{\tau_i\}_{i \in i}$ of data types) is a triple $(t_t, \omega_t, \upsilon_t)$ consisting of an unranked tree $t_t = (\mathcal{O}_t, \prec_c, \prec_s)$, a total *label function* ω_t: $\mathcal{O}_t \to \Sigma$, and a partial *value function* υ_t: $\mathcal{O}_t \to U$ defined for all leaves in t_t.

The set of all XML trees over Σ (neglecting the universe U) is denoted as T_Σ.

Definition 3. An XML tree t_1 is said to be the *subtree* of an XML tree t_2 at node $o \in \mathcal{O}_{t_2}$ iff there exists an embedding h: $\mathcal{O}_{t_1} \hookrightarrow \mathcal{O}_{t_2}$ satisfying the following properties: (i) the root of tree t_1 is o; (ii) whenever $o_1 \prec_c o_2$ holds in t_1, then $h(o_1) \prec_c h(o_2)$ holds in t_2; (iii) whenever $o_1 \prec_c o_2$ holds in t_2 with $o_1 \in \mathcal{O}_{t_1}$, then also $o_1 \prec_c o_2$ holds in t_1; (iv) whenever $o_1 \prec_s o_2$ holds in t_1, then $h(o_1) \prec_s h(o_2)$ holds in t_2; (v) $\omega_{t_1}(o') = \omega_{t_2}(o')$ holds for all $o' \in \mathcal{O}_{t_1}$; (vi) for all $o' \in \mathcal{O}_{t_1}$ either $\upsilon_{t_1}(o') = \upsilon_{t_2}(o')$ holds or otherwise both sides are undefined.

We use the notation $\widehat{o}$ to denote the subtree of an XML tree t rooted at node o for $o \in \mathcal{O}_t$, and $root(t)$ to denote the root node of an XML tree t. 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*. The notion of forest is indispensable in situations where order is irrelevant, e.g. when representing attributes of a node, but duplicates are desirable, e.g. for computations in parallel on identical subtrees. ε denotes the *empty hedge*.

In order to define flexible operations on XML trees it will be necessary to select tree portions of interest. Such portions can be subtrees, but occasionally we will need more general structures. This will be supported by XML contexts.

Definition 4. An *XML context* over an alphabet Σ ($\xi \notin \Sigma$) is an unranked tree t over $\Sigma \cup \{\xi\}$, i.e., $t \in T_{\Sigma \cup \{\xi\}}$, such that for each tree t 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 2.

The context with a single node labelled ξ is called the *trivial context* and 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. To ensure that the special label ξ occurs at most once in the result, we distinguish four kinds of substitutions, where $[\widehat{o} \mapsto t]$ indicates substituting t for the subtree rooted at o.

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 $t_1[\widehat{o} \mapsto t_2]$ 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 $t_1[\widehat{o} \mapsto \xi]$ is an XML context in $T_{\Sigma_1 \cup \{\xi\}}$.

Context-to-context substitution: For an XML context $c_1 \in T_{\Sigma_1 \cup \{\xi\}}$ with a node $o \in \mathcal{O}_{c_1}$ and an XML tree $t_2 \in T_{\Sigma_2}$, the result $c_1[\widehat{o} \mapsto t_2]$ 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 $c_1[\xi \mapsto t_2]$ 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.

3 Tree Algebra

Our objective is to provide manipulation operations on XML trees at a higher level than individual nodes and edges. So we need some tree constructs to extract arbitrary tree portions of interest. For this we provide two selector constructs,

which will result in subtrees and contexts, respectively. For an XML tree $t = (t_t, \omega_t, \upsilon_t)$ these constructs are defined: (i) *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_1, i.e. $o_1 \prec_c^* o_2$ holds for the transitive closure $\prec_c^*$ of $\prec_c$. We have $context(o_1, o_2) = \widehat{o}_1[\widehat{o}_2 \mapsto \xi]$. (ii) *subtree* is a unary function defined on $\mathcal{O}_t$. We have $subtree(o) = \widehat{o}$.

We now need some algebra operations to recombine tree portions to form a new XML tree. We define a many-sorted algebra using three sorts: **L** for labels, **H** for hedges, and **C** for contexts, along with a set $\mathcal{F}=\{\iota, \delta, \varsigma, \rho, \kappa, \eta, \sigma\}$ of function symbols with the following signatures: $\iota : \mathbf{L} \times \mathbf{H} \to \mathbf{H}$, $\delta : \mathbf{L} \times \mathbf{C} \to \mathbf{C}$, $\varsigma : \mathbf{H} \times \mathbf{C} \to \mathbf{C}$, $\rho : \mathbf{H} \times \mathbf{C} \to \mathbf{C}$, $\kappa : \mathbf{H} \times \mathbf{H} \to \mathbf{H}$, $\eta : \mathbf{C} \times \mathbf{H} \to \mathbf{H}$, $\sigma : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$. Given that 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 $\varepsilon \in T_\mathcal{H}^s$, $t\langle h\rangle \in T_\mathcal{H}^s$ for $t \in \Sigma$ and $h \in T_\mathcal{H}$, and $t_1, ..., t_n \in T_\mathcal{H}^m$ 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. Furthermore, we use $\#t$ to denote the sort of a term t.

Example 1. Let $\Sigma = \{a, b, c, \tau_1, \tau_2\}$, then a, b, τ_1 and τ_2 are terms of sort **L**, $a\langle b\langle\tau_1\rangle, c\langle\tau_2\rangle\rangle$ and $b\langle\tau_1\rangle, a\langle a\langle\tau_1\rangle, \tau_2\rangle$ are terms of sort **H**, and $a\langle a\langle\tau_1\rangle, \xi, \tau_1\rangle$ 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 2. The function κ denotes hedge juxtaposition, and likewise σ is context composition. The function η denotes context substitution, i.e. substituting the variable ξ in a context with a hedge, which leads to a tree. A further illustration of these functions is provided in Figure 1, which can be formally defined as follows (the case $n = 0$ for $a\langle t_1, \ldots, t_n\rangle$ leads to $a\langle\xi\rangle$): (1) $\iota(a, (t_1, \ldots, t_n)) = a\langle t_1, \ldots, t_n\rangle$, (2) $\delta(a, c) = a\langle c\rangle$, (3) $\varsigma((t_1, \ldots, t_n), a\langle t'_1, \ldots, t'_m\rangle) = a\langle t_1, \ldots, t_n, t'_1, \ldots, t'_m\rangle$, (4) $\rho((t_1, \ldots, t_n), a\langle t'_1, \ldots, t'_m\rangle) = a\langle t'_1, \ldots, t'_m, t_1, \ldots, t_n\rangle$, (5) $\kappa((t_1, \ldots, t_n), (t'_1, \ldots, t'_m)) = t_1, \ldots, t_n), t'_1, \ldots, t'_m$, (6) $\eta(c, (t_1, \ldots, t_n)) = c[\xi \mapsto t_1], \ldots, c[\xi \mapsto t_n]$ and (7) $\sigma(c_1, c_2) = c_1[\xi \mapsto c_2]$.

Example 2. As shown in Figure 2, given that a context term $c_1 = a\langle b, \xi\rangle$ and a hedge term $t_1 = b\langle e\rangle, b\langle d\rangle$, we obtain $\varsigma(t_1, c_1) = a\langle b\langle e\rangle, b\langle d\rangle, b, \xi\rangle$ and $\eta(t_1, c_1) = a\langle b, \xi, b\langle\rangle, b\langle d\rangle\rangle$.

The proof of the following proposition is a straightforward exercise.

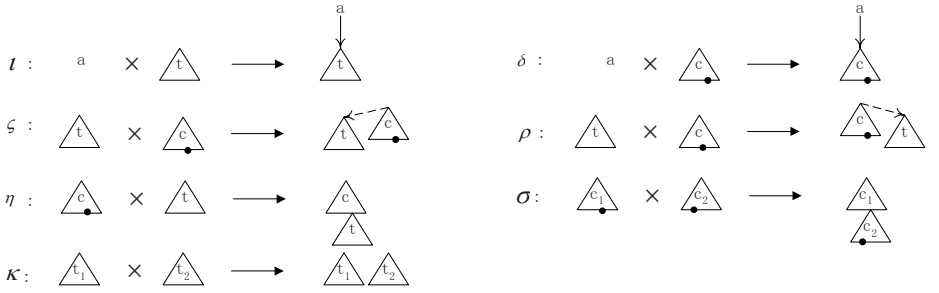

Fig. 1. XML tree algebra

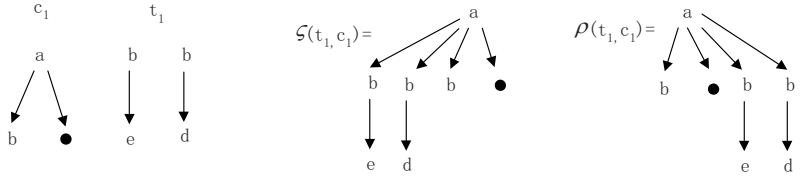

Fig. 2. An illustration of functions $\varsigma(t_1, c_1)$ and $\eta(t_1, c_1)$

Proposition 1. *The algebra defined above satisfying 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): (i) $\eta(\sigma(t_1, t_2), t_3) = \eta(t_1, \sigma(t_2, t_3))$; (ii) $\sigma(\sigma(t_1, t_2), t_3) = \sigma(t_1, \sigma(t_2, t_3))$; (iii) $\kappa(\kappa(t_1, t_2), t_3) = \kappa(t_1, \kappa(t_2, t_3))$; (iv) $\eta(\delta(t_1, t_2), t_3) = \iota(t_1, \eta(t_2, t_3))$; (v) $\varsigma(t_1, \varsigma(t_2, t_3)) = \varsigma(\kappa(t_1, t_2), t_3)$; (vi) $\rho(t_1, \rho(t_2, t_3)) = \rho(\kappa(t_1, t_2), t_3)$; (vii) $\rho(t_3, \varsigma(t_1, t_2)) = \varsigma(t_1, \rho(t_3, t_2))$.*

Example 3. To illustrate equation $\eta(\delta(t_1, t_2), t_3) = \iota(t_1, \eta(t_2, t_3))$ let us take $t_1 = b$, $t_2 = a\langle b, \xi\rangle$ and $t_3 = b\langle e\rangle$. Then the left hand side becomes $\eta(\delta(t_1, t_2), t_3) = \eta(\delta(b, a\langle b, \xi\rangle), b\langle e\rangle) = \eta(b\langle a\langle b, \xi\rangle\rangle, b\langle e\rangle) = b\langle a\langle b, b\langle e\rangle\rangle\rangle$ and the right hand side becomes $\iota(t_1, \eta(t_2, t_3)) = \iota(b, \eta(a\langle b, \xi\rangle, b\langle e\rangle)) = \iota(b, a\langle b, b\langle e\rangle\rangle) = b\langle a\langle b, b\langle e\rangle\rangle\rangle$.

4 Weak Monadic Second-Order Logic

In this section we concentrate on a logic, which permits to to navigate within an XML tree. This can be combined with the use of the tree algebra defined before to manipulate tree structures. We first provide a weak MSO logic over finite and unranked trees adopting the logic from [4] 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 already from early work of Büchi [3]. For XML the navigational part of XPath2.0 can capture first-order logic, and some extensions on XPath have been shown

to be expressively complete for MSO, e.g. using fixed-point operators. Several people have independently proposed an extension called "regular XPath" with a Kleene star operator for transitive closure, which can also be captured by MSO.

We first define a logic MSO_X with interpretations in an XML tree. For this let V_{FO} and V_{SO} denote the sets of first- and second-order variables, respectively. We denote the former ones by lower-case letters and the latter ones by upper-case letters, respectively. Using abstract syntax the formulae of MSO_X are defined by: $\varphi \equiv x_1 = x_2 \mid \upsilon(x_1) = \upsilon(x_2) \mid \omega_a(x_1) \mid x \in X \mid x_1 \prec_c x_2 \mid x_1 \prec_s x_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists x.\varphi \mid \exists X.\varphi$, with $x, x_1, x_2 \in V_{FO}$, $X \in V_{SO}$, 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 $t = (t_t, \omega_t, \upsilon_t)$ over the set Σ of labels with $t_t = (\mathcal{O}_t, \prec_c^t, \prec_s^t)$. Naturally, the function symbols ω_a and υ should be interpreted by the labelling and value functions ω_t and υ_t, respectively, and the predicate symbols $\prec_c$ and $\prec_s$ should receive interpretations 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:

- $[\![\tau_1 = \tau_2]\!]_{t,\zeta} = \mathit{true}$ iff $val_{t,\zeta}(\tau_1) = val_{t,\zeta}(\tau_2)$ holds for the terms τ_1 and τ_2,
- $[\![\omega_a(x)]\!]_{t,\zeta} = \mathit{true}$ holds iff $\omega_t(val_{t,\zeta}(x)) = a$,
- $[\![x \in X]\!]_{t,\zeta} = \mathit{true}$ iff $val_{t,\zeta}(x) \in val_{t,\zeta}(X)$,
- $[\![\neg\varphi]\!]_{t,\zeta} = \mathit{true}$ iff $[\![\varphi]\!]_{t,\zeta} = \mathit{false}$,
- $[\![\varphi_1 \wedge \varphi_2]\!]_{t,\zeta} = \mathit{true}$ iff $[\![\varphi_1]\!]_{t,\zeta} = \mathit{true}$ and $[\![\varphi_2]\!]_{t,\zeta} = \mathit{true}$,
- $[\![\exists x.\varphi]\!]_{t,\zeta} = \mathit{true}$ iff $[\![\varphi]\!]_{t,\zeta[x\mapsto o]} = \mathit{true}$ holds for some $o \in \mathcal{O}$,
- $[\![\exists X.\varphi]\!]_{t,\zeta} = \mathit{true}$ iff $[\![\varphi]\!]_{t,\zeta[X\mapsto O]} = \mathit{true}$ for some $O \subseteq \mathcal{O}$,
- $[\![x_1 \prec_c x_2]\!]_{t,\zeta} = \mathit{true}$ 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
- $[\![x_1 \prec_s x_2]\!]_{t,\zeta} = \mathit{true}$ 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 for other MSO_X formulae in the usual way. Likewise, the definition of bound and free variables of MSO_X formulae is also standard. We use the notation $fr(\varphi)$ for the set of free variables of the formula φ. Given that an XML tree t and a MSO_X formula φ with $fr(\varphi) = \{x_1, \ldots, x_n\}$, then φ is said to be *satisfiable* in t with respect to the variable assignment ζ iff $[\![\varphi]\!]_{S,\zeta}(\varphi) = \mathit{true}$.

5 XML Machines

In this section we present *XML machines* (XMLMs), a computational model for XML that adopts Abstract State Machines [2] for the purpose of dealing with XML database transformations. The most important extension of XML machines is the incorporation of MSO_X formulae in forall- and choice-rules and the use of terms from the tree algebra. The other rules used by XML machines are more or less the same except for an added partial update rule that is added for convenience, although it does not add any additional expressive power.

Using an unbounded number of parallel processes, an update operator $\cup$ merging two hedges into one is needed. With these preliminary remarks we can now define *MSO*-rules in analogy to rules in ASMs. In the following definition the formulae φ always refer to MSO_X formulae as discussed in Section 4.

Definition 5. The set $\mathbf{R}$ of *MSO-rules* over a signature $\Sigma = \Sigma_{db} \cup \Sigma_a \cup \{f_1, \ldots, f_\ell\}$ is defined as follows ($var(t)$ is the set of variables occurring in t):

- If t is a term over Σ, and f is a location in Σ such that $\#f = \#t$, then $f := t$ is a rule r in $\mathbf{R}$ called *assignment rule* with $fr(r) = var(t)$.
- If t is a term over Σ, f is a location in Σ and $\cup$ is a binary operator such that $\#f = \#t$ and $\cup : \#t^2 \to \#f$, then $f \Leftarrow^{\cup} t$ is a rule r in $\mathbf{R}$ called *partial assignment rule* with $fr(r) = var(t)$.
- 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 $fr(\varphi) = \{x_1, \ldots, x_k, X_1, \ldots, X_m\}$ and $r' \in \mathbf{R}$ is an MSO-rule, then **forall** $x_1, \ldots, x_k, X_1, \ldots, X_m$ **with** φ **do** r' **enddo** is a rule r in $\mathbf{R}$ called *forall rule* with $fr(r) = fr(r') - fr(\varphi)$.
- 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 $fr(\varphi) = \{x_1, \ldots, x_k, X_1, \ldots, X_m\}$ and $r' \in \mathbf{R}$ is an MSO-rule, then **choose** $x_1, \ldots, x_k, X_1, \ldots, X_m$ **with** φ **do** r' **enddo** is an MSO-rule r in $\mathbf{R}$ called *choice rule* with $fr(r) = fr(r') - fr(\varphi)$.
- 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 associated sets of update sets $\Delta(r, S)$ for a closed MSO-rule r with respect to a state S is again straightforward [8]. We only explain the non-standard case of the partial assignment rule. For this let r denote partial assignment rule $f \Leftarrow^{\cup} t$, and let 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 $\ell = val_{S,\zeta}(f)$ and $a = val_{S,\zeta}(t)$, 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 undate 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 $val_{S+\Delta}(\ell)$ will be undefined.

Example 4. Assume that the XML tree in Figure 3 is assigned to the variable (tree name) t_{exa}. The following MSO-rule will construct the XML tree in (i) from subtrees of the given XML tree using operators of the tree algebra:

$t_1 := \epsilon$;
forall x,y,z **with** $\prec_c (t_{exa}, root(t_{exa}), x) \wedge \prec_c (t_{exa}, x, y) \wedge \prec_c (t_{exa}, x, z)$
$\wedge \omega(t_{exa}, x) = b \wedge \omega(t_{exa}, y) = c \wedge \omega(t_{exa}, z) = a$
do
$t_1 \leftleftarrows^{\cup} \iota(d, \kappa(subtree(t_{exa}, y), subtree(t_{exa}, z)))$;
enddo ;
$output := \iota(r, t_1)$

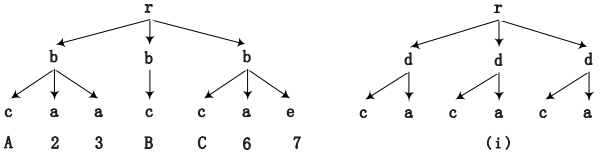

Fig. 3. An XML tree and the result of tree operations

Definition 6. An *XML Machine* (XMLM) $\mathcal{M}$ consists of a set $\mathcal{S}_\mathcal{M}$ of states over Σ closed under isomorphisms, non-empty subsets $\mathcal{I}_\mathcal{M} \subseteq \mathcal{S}_\mathcal{M}$ of initial states, and $\mathcal{F}_\mathcal{M} \subseteq \mathcal{S}_\mathcal{M}$ of final states, both also closed under isomorphisms, a program $\pi_\mathcal{M}$ defined by a closed MSO-rule r over Σ, and a binary relation $\tau_\mathcal{M}$ over $\mathcal{S}_\mathcal{M}$ determined by $\pi_\mathcal{M}$ such that the following holds:

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

Theorem 1. *The XMLMs capture exactly all XML database transformations.*

Proof. According to [8,9] each XML database transformation can be represented by a behaviourally equivalent ADTM with the same tree-based background, and vice verse. As ADTMs differ from XMLMs only by the fact that ADTM-rules are more restrictive than MSO-rules (they do not permit MSO_X formulae in forall- and choice-rules), such an ADTM is in fact also an XMLM.

Thus, it suffices to show that XMLMs satisfy the postulates for XML database transformations in [8]. 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 genericity postulates.

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

We use this to define an exploration boundary witness T. If t_i is ground, we add the access term $(-, t_i)$ to T. If t_i is not ground, then the corresponding assignment rule must appear within the scope of forall and choice rules introducing the database variables in t_i, as r is closed. Thus, variables in t_i are bound by a formula φ, i.e. for $fr(t_i) = \{x_1, \ldots, x_k\}$ the relevant variable assignments are $\zeta = \{x_1 \mapsto b_1, \ldots, x_k \mapsto b_k\}$ 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 $\{(\beta_1, \alpha_1), \ldots, (\beta_\ell, \alpha_\ell)\}$ 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 $(\beta_1, \alpha_1), \ldots, (\beta_\ell, \alpha_\ell)$ to T.

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

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

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

Regarding genericity assume that $\mathcal{M}$ does not satisfy the genericity postulate. Then there must be a state S and equivalent substructures $S_1, S_2 \preceq S$ such that S_1 is preserved by $\Delta(r, S)$, i.e. $S_1 \preceq S + \Delta_1$ for some $\Delta_1 \in \Delta(r, S)$, but S_2 is not, i.e. $S_2 \not\preceq S + \Delta_2$ for all $\Delta_2 \in \Delta(r, S)$. According to our remark above r must contain a choice rule **choose** $x_1, \ldots, x_k$ **with** φ **do** r' **enddo**. For a state S' let $\mathcal{B}_{S'} = \{(b_1, \ldots, b_k) \mid val_{S',[x_1 \mapsto b_1, \ldots, x_k \mapsto b_k]}(\varphi) = true\}$. Then the automorphism $\sigma : S \to S$ induced by $S_1 \equiv S_2$ is defined by a permutation on $\mathcal{B}$. From this we obtain $\sigma(\Delta(r, S, S + \Delta_1) = \Delta(r, S, \sigma(S + \Delta_1)) = \Delta(r, S, S + \Delta_2)$ for some $\Delta_2 \in \Delta(r, S)$. Thus, $S_2 \preceq S + \Delta_2$ for this Δ_2, which contradicts our assumption. □

6 Conclusion

In this paper we continued our research on foundations of database transformations exploiting the theory of Abstract State Machines. In [8] we developed a theoretical framework for database transformations in general, which are defined by five intuitive postulates and exactly characericterised by ADTMs, 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 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.

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 XML machines. First steps in this direction have been made in [12].

References

1. Blass, A., Gurevich, Y.: Background, reserve, and gandy machines. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 1–17. Springer, Heidelberg (2000)
2. Börger, E., Stärk, R.: Abstract State Machines. Springer, Heidelberg (2003)
3. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 6(1-6), 66–92 (1960)
4. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007), `http://www.grappa.univ-lille3.fr/tata` (release, October 2007)
5. Doner, J.: Tree acceptors and some of their applications. Journal of Computer and Systems Science 4(5), 406–451 (1970)
6. Gurevich, J.: Sequential abstract state machines capture sequential algorithms. ACM Transactions on Computational Logic 1(1), 77–111 (2000)
7. Kumar, V., Madhusudan, P., Viswanathan, M.: Visibly pushdown automata for streaming XML. In: Proceedings of WW 2007, pp. 1053–1062. ACM, New York (2007)
8. Schewe, K.-D., Wang, Q.: A customised ASM thesis for database transformations (2009) (submitted for publication)
9. Schewe, K.-D., Wang, Q.: XML database transformations (2009) (submitted for publication)
10. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. MST 2(1), 57–81 (1968)
11. Van Den Bussche, J., Van Gucht, D., Andries, M., Gyssens, M.: On the completeness of object-creating database transformation languages. J. ACM 44(2), 272–319 (1997)
12. Wang, Q., Schewe, K.-D.: Towards a logic for abstract metafinite state machines. In: Hartmann, S., Kern-Isberner, G. (eds.) FoIKS 2008. LNCS, vol. 4932, pp. 365–380. Springer, Heidelberg (2008)