Instance Independent Concurrency Control for Semistructured Databases

Stijn Dekeyser, Jan Hidders, Jan Paredaens

University of Antwerp

Abstract. Semistructured databases require tailor-made concurrency control mechanisms since traditional solutions for the relational model have been shown to be inadequate. Such mechanisms need to take full advantage of the hierarchical structure of semistructured data, for instance allowing concurrent updates of subtrees of, or even individual elements in XML documents. In earlier work, we presented two equivalent path locking schemes and two schedulers which guarantee serializability of schedules on XML documents. However, these methods are dependent on the instance of the XML database. In this paper we take a more general, higher level approach by characterizing and deciding equivalence of schedules without looking at the instance.

1 Introduction

In previous work [3–5] we have shown that traditional concurrency control [9] mechanisms for the relational and object-oriented model [2, 6–8] are inadequate to capture the complicated update behavior that is possible for semistructured databases. Indeed, when XML documents are stored in relational databases, their hierarchical structure becomes invisible to the locking strategy used by the database management system. Therefore, in previous work mentioned above, we have investigated the use of path locks to remedy this situation. We introduced two equivalent path locking protocols. For both systems we introduced conflict rules and analyzed their complexity. We also indicated that the conflict rules are necessary. In addition, we introduced a *commit* scheduler and a *conflict* scheduler which guarantee serializability of schedules involving queries and updates.

The differences between the work presented in this paper and our earlier work are as follows. First, in our earlier work the definition of equivalence of schedules is tightly bound to the document over which the schedule is defined. This has the advantage that relatively more schedules are serializable than in the work we present here. However, the clear disadvantage is that involving the instance of the database is much less general. The second difference is that in earlier work we require schedules to be node-correct, meaning that any identifier used in a transaction must have been obtained through an earlier operation in the same transaction. In this work we drop this requirement, again making this work more general.

Contribution In this paper we first characterize the *correctness* of transactions involving only update operations (queryless transactions); i.e. whether there exists at least one document for which the transformation represented by the transaction is defined. We then show decidability of equivalence and of serializability of queryless schedules. Finally, we characterize equivalence of schedules including both queries and update operations.

2 Data Model and Operations

The data model is derived from the classical data model for semistructured data [1]; we consider directed, unordered trees in which the edges are labelled. Consider a fixed universal set of nodes $\mathcal{N}$ and a fixed universal set of edge labels $\mathcal{L}$ not containing the symbol /.

```
<document id="0">
<person id="1",age="55",
 spouse="2">
  <name>Peter</name>
  <addr>Parklane 7</addr>
  <child>
   <person id="3", age="22">
     <name>John</name>
     <addr>Unistreet 1</addr>
     <hobby>swimming</hobby>
     <hobby>cycling</hobby>
   </person>
  </child>
</person>
<person id="2", age="43">
  <name>Mary</name>
  <addr>Parklane 7</addr>
  <hobby>painting</hobby>
</person>
</document>
```

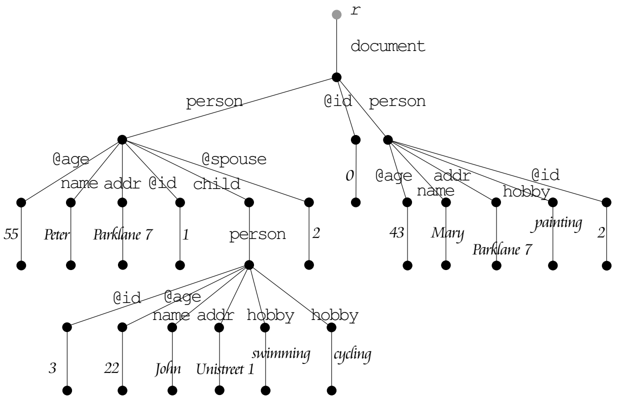

Fig. 1. A fragment of an XML document and its DT representation.

Definition 1. *A* graph *is a tuple* (N, E)[1] *with* $N \subseteq \mathcal{N}$ *and* $E \subseteq N \times \mathcal{L} \times N$. *A* document tree *(*DT*)* T *is a tuple* (N, E, r) *such that* (N, E) *is a graph that represents a tree with root* r*. The edges are directed from the parent to the child.*

This data model closely mimics the XML data model as illustrated next.

Example 1. Consider the XML document D of Figure 1. Figure 1 presents a representation of D into a DT. Note that the document order is not preserved, and that there is no distinction between elements, attributes and text nodes.

[1] Because every set of edges $E \subseteq \mathcal{N} \times \mathcal{L} \times \mathcal{N}$ uniquely defines a graph, we will call such sets also graphs.

Definition 2. *Let G be a graph and m and n two nodes in G then $path_G(m, n)$ is the set of simple paths*[2] *in G from m to n.*
A label path *is a string of the form $l_1/\ldots/l_m$ with $m \geq 0$ and every l_i an edge label in $\mathcal{L}$. Given a path $p = ((n_1, l_1, n_2), \ldots, (n_m, l_m, n_{m+1}))$ in a graph G, the* label path of *p, denoted $\bar{\lambda}_T(p)$ (or $\bar{\lambda}(p)$ when T is subsumed) is the string $l_1/\ldots/l_m$.*

Processes working on document trees do so in the context of a general programming language that includes an interface to a document server which manages transactions on documents. The process generates a list of operations that will access the document. In general there are three types of operations: the query, the addition and the deletion. The input to a query-operation will be a node and a path expression, while the result of the invocation of a query-operation will be a set of nodes. The programming language includes the concepts of sets, and has constructs to iterate over their entire contents. The input to an addition or a deletion will be an edge. The result of an addition or a deletion will be a simple transformation of the original tree into a new tree. If the result would not be a tree anymore it is not defined.
We now define path expressions and query-operations, subsuming a given DT T. The syntax of path expressions[3] is given by $\mathcal{P}$:

$$\mathcal{P} ::= pe_\epsilon \mid \mathcal{P}^+ \quad \mathcal{P}^+ ::= \mathcal{F} \mid \mathcal{P}^+/\mathcal{F} \mid \mathcal{P}^+//\mathcal{F} \quad \mathcal{F} ::= * \mid \mathcal{L}$$

The set $\mathbf{L}(pe)$ of label paths represented by a path expression pe is defined as:

$$\begin{array}{ll} \mathbf{L}(pe_\epsilon) = \{\epsilon\} \quad \mathbf{L}(*) = \mathcal{L} & \mathbf{L}(l) = \{l\} \\ \mathbf{L}(pe/f) = \mathbf{L}(pe) \cdot \{/\} \cdot \mathbf{L}(f) & \mathbf{L}(pe//f) = \mathbf{L}(pe) \cdot \{/\} \cdot (\mathcal{L} \cdot \{/\})^* \cdot \mathbf{L}(f) \end{array}$$

Let n be an arbitrary node of T and pe a path expression. The query-operation query(n, pe) returns a set of nodes, and is defined as follows:

Definition 3. query(n, pe) *with $n \in \mathcal{N}$ and $pe \in \mathcal{P}$. The result of* query(n, pe) *on a* DT *T is defined as* $\text{query}_T(n, pe) = \{n' \in N \mid \exists p \in path_T(n, n') : \bar{\lambda}(p) \in \mathbf{L}(pe)\}$.

There are two update-operations; the addition and the deletion of an edge:

Definition 4. *The update operations return no value but transform a* DT *$T = (N, E, r)$ into a new* DT *$T' = (N', E', r)$.*

- add(n, l, n') *with $n, n' \in \mathcal{N}$ and $l \in \mathcal{L}$. The resulting T' is defined by $E' = E \cup \{(n, l, n')\}$ and $N' = N \cup \{n'\}$. If the resulting T is not a document tree anymore or (n, l, n') was already in the document tree then the operation is undefined.*
- del(n, l, n') *with $n, n' \in \mathcal{N}$ and $l \in \mathcal{L}$. The resulting T' is defined by $E' = E - \{(n, l, n')\}$ and $N' = N - \{n'\}$. If the resulting T is not a document tree anymore or (n, l, n') was not in the document tree then the operation is undefined.*

[2] This includes empty paths that contain no edges and start and end in the same node.
[3] Remark that path expressions form a subset of XPath expressions.

3 Schedules

We first give some straightforward definitions of schedules and their semantics. An *action* is a pair (o, t), where o is one of the three operations $\text{query}(n, pe)$, $\text{add}(n, l, n')$ and $\text{del}(n, l, n')$ and t is a transaction identifier. A *transaction* is a sequence of actions with the same transaction identifier. A *schedule* over a set of transactions is an interleaving of these transactions.
We can *apply* a schedule S on a DT T. The result of such application is (1) the DT that results from the sequential application of the actions of S; this DT is denoted by $S[T]$, and (2) for each query in S the result of this query.
If some of these actions are undefined the application is undefined. Two schedules are equivalent on a DT T iff their application on T has the same result. Two schedules are equivalent iff they are defined on the same non-empty set of DT's and they are equivalent on these DT's. The definition of serial and serializable schedules is straightforward.

Example 2. Let $T_1 = (\{n_1, n_2\}, \{(n_1, b, n_2)\}, n_1)$, $T_2 = (\{n_1, n_2\}, \{(n_1, a, n_2)\}, n_1)$ and $T_3 = (\{n_1\}, \emptyset, n_1)$ be three DT's and let $S_1 = (\text{add}(n_2, b, n_3), t_1), (\text{query}(n_1, a/b), t_2)$ and $S_2 = (\text{query}(n_1, a/b), t_2), (\text{add}(n_2, b, n_3), t_1)$ be two schedules. Schedules S_1 and S_2 are equivalent on T_1, they are not equivalent on T_2 and their application is undefined on T_3.

Example 3. Let $S_1 = (\text{add}(n_1, l_1, n_2), t_1), (\text{del}(n_1, l_1, n_2), t_2)$ and S_2 be the empty schedule. Schedules S_1 and S_2 are not equivalent although they are equivalent on many DT's.

4 Correctness of Queryless Schedules

We begin by considering only transactions and schedules that contain no queries. Later in this paper we will look at general schedules.

Definition 5. *A transaction (a schedule) is called* queryless *(QL) iff it contains no queries.*

Because of the way that operations can fail it is possible that the application of a certain transaction is not defined for any document tree. However, this is a necessary property for all transactions in a schedule if the schedule is to be serializable. Therefore we call such transactions (schedules) correct transactions (schedules).

Definition 6. *A QL schedule S is called* correct *iff there is a* DT *T with $S[T]$ defined.*

It will be clear that we are only interested in correct transactions.

Example 4. The next transaction is correct: $(\text{add}(r, l_1, n_1), t_1)$, $(\text{del}(r, l_1, n_1), t_1)$, $(\text{add}(r, l_2, n_2), t_1)$, $(\text{del}(r, l_2, n_2), t_1)$, $(\text{add}(r, l_2, n_2), t_1)$, $(\text{del}(r, l_2, n_2), t_1)$.
The next transaction is not correct: $(\text{add}(r, l_1, n_1), t_1)$, $(\text{del}(r, l_2, n_1), t_1)$, $(\text{add}(n_1, l_3, n_2), t_1)$.

Remark that if two QL schedules are equivalent then they are both correct. This equivalence relation is defined on the set of correct QL schedules.

Example 5. Here is a QL schedule S that is defined on $T = (\{r\}, \emptyset, r)$ but that is not serializable because of only one transaction t_1 that is not correct. Every equivalent serial QL schedule would be undefined! Transaction t_1 has the property that all QL schedules over a set of transactions that contain t_1 are non-serializable, independent of T.
$S = (\text{add}(r, l_1, n_1), t_1), (\text{del}(r, l_1, n_1), t_2), (\text{add}(r, l_1, n_1), t_1)$.

Since a transaction is a special case of a schedule all the definitions on QL schedules also apply on transactions.

We will characterize correct QL schedules and prove that this property is decidable. For this purpose we will first attempt to characterize for which document trees a given correct QL schedule S is defined, and what the properties for the document trees in the result of the QL schedule. We do this by defining the sets $N_I^+(S)$, $N_I^-(S)$, $E_I^+(S)$ and $E_I^-(S)$, which informal meaning is respectively the nodes that are required in the input document tree, the nodes that are not allowed, the edges that are required and the edges that are not allowed. In the same fashion we define the sets $N_O^+(S)$, $N_O^-(S)$, $E_O^+(S)$ and $E_O^-(S)$, which informal meaning is respectively the nodes that are always present in the output document tree, the nodes that are never present, the edges that are always present and the edges that are never present.

Definition 7. *Let S be a QL schedule. $\phi_S(n, o)$ ($\phi_S((m, l, n), o)$) indicates that the first occurrence of the node n (the edge (m, l, n)) in the schedule S has the form of the operator o. For example, $\phi_S(n_2, \text{add}(r, l_2, n_2))$ in the correct QL schedule in Example 4 above. $\lambda_S(n, o)$ ($\lambda_S((m, l, n), o)$) indicates that the last occurrence of the node n (the edge (m, l, n)) in the QL schedule S has the form of the operation o. We then define the sets $N_I^+(S)$, $N_I^-(S)$, $E_I^+(S)$ and $E_I^-(S)$, and the sets $N_O^+(S)$, $N_O^-(S)$, $E_O^+(S)$ and $E_O^-(S)$ as in Figure 2.*
A DT T is called a basic-input-tree (basic-output-tree) of S iff it contains all the nodes of $N_I^+(S)$ ($N_O^+(S)$), no nodes of $N_I^-(S)$ ($N_O^-(S)$), all the edges of $E_I^+(S)$ ($E_O^+(S)$) and no edges of $E_O^-(S)$ ($E_O^-(S)$).

Example 6. Let $S = (\text{add}(n_1, l_1, n_2), t_1)$, $(\text{del}(n_4, l_2, n_3), t_2)$, $(\text{del}(n_1, l_1, n_4), t_3)$ then

$$\begin{aligned}
&N_I^+(S) = \{n_1, n_3, n_4\};\ N_I^-(S) = \{n_2\};\ E_I^+(S) = \{(n_4, l_2, n_3), (n_1, l_1, n_4)\} \\
&E_I^-(S) = \{\ (m, l, n_2),\ (n_2, l, m),\ (m', l', n_3),\ (m'', l'', n_4),\ (n_3, l, m), \\
&\qquad (n_4, l''', m''') \mid l, l', l'', l''' \in \mathcal{L},\ m, m', m'', m''' \in \mathcal{N}, \\
&\qquad (m', l') \neq (n_4, l_2),\ (m'', l'') \neq (n_1, l_1),\ (l''', m''') \neq (l_2, n_3)\} \\
&N_O^+(S) = \{n_1, n_2\};\ N_O^-(S) = \{n_3, n_4\};\ E_O^+(S) = \{(n_1, l_1, n_2)\} \\
&E_O^-(S) = \{\ (n_2, l, m),\ (m', l', n_2),\ (n_3, l, m),\ (m, l, n_3),\ (n_4, l, m),\ (m, l, n_4) \mid \\
&\qquad l, l' \in \mathcal{L},\ m, m' \in \mathcal{N},\ (m', l') \neq (n_1, l_1)\}
\end{aligned}$$

$$\begin{array}{rl}
N_I^+(S) = & \{m|\phi_S(m, \mathrm{add}(m,l,n))\} \cup \{m|\phi_S(m, \mathrm{del}(m,l,n))\} \ \cup \{n|\phi_S(n, \mathrm{del}(m,l,n))\} \\
N_I^-(S) = & \{n|\phi_S(n, \mathrm{add}(m,l,n))\} \\
E_I^+(S) = & \{(m,l,n)|\ \phi_S((m,l,n), \mathrm{del}(m,l,n))\} \\
E_I^-(S) = & \{(m,l,n)|\ \phi_S((m,l,n), \mathrm{add}(m,l,n))\} \cup \\
& \{(m,l,n)|\ \exists(m_1,l_1,n) \text{ that occurs in } S, (m_1,l_1) \neq (m,l) \text{ and} \\
& \qquad (m,l,n) \text{ does not occur before}\} \cup \\
& \{(m,l,n)|\exists(m_1,l_1,m) \text{ that occurs in } S \text{ and } (m,l,n) \text{ does not occur before}\} \\
N_O^+(S) = & \{m|\lambda_S(m, \mathrm{del}(m,l,n))\} \cup \{m|\lambda_S(m, \mathrm{add}(m,l,n))\} \cup \{n|\lambda_S(n, \mathrm{add}(m,l,n))\} \\
N_O^-(S) = & \{n|\lambda_S(n, \mathrm{del}(m,l,n))\} \\
E_O^+(S) = & \{(m,l,n)|\lambda_S((m,l,n), \mathrm{add}(m,l,n))\} \\
E_O^-(S) = & \{(m,l,n)|\lambda_S((m,l,n), \mathrm{del}(m,l,n))\} \cup \\
& \{(m,l,n)|\exists(m_1,l_1,n) \text{ that occurs in } S, (m_1,l_1) \neq (m,l) \text{ and} \\
& \qquad (m,l,n) \text{ does not occur afterwards }\} \cup \\
& \{(m,l,n)|\exists(m_1,l_1,m) \text{ that occurs in } S \text{ and } (m,l,n) \text{ does't occur afterwards}\}
\end{array}$$

Fig. 2. The Definition of the Basic Input and Output Sets.

When a QL schedule is not correct this is always because two operations in the QL schedule conflict, as for example the first two operations in the incorrect transaction of Example 4: add(r, l_1, n_1) and del(r, l_2, n_1). If these two operations immediately follow each other then at least one of them will always fail. However, if between them we find the action del(r, l_1, n_1) then this is no longer true. The following definition attempts to identify such pairs of conflicting operations and states which operations we should find between them to remove the conflict.

Definition 8. *A QL schedule fulfills the* C-condition *iff actions a_1 and a_2 appear in that order in S and action a_3 appears between them, where a_1, a_2, a_3 are as follows.*

a_1	a_2	a_3
$(\mathrm{add}(n,l_1,n_1),t_1)$	$(\mathrm{add}(n_2,l_2,n),t_2)$	$(\mathrm{del}(n,l_1,n_1),t_3)$
$(\mathrm{add}(n_1,l_1,n),t_1)$	$(\mathrm{add}(n_2,l_2,n),t_2)$	$(\mathrm{del}(n_1,l_1,n),t_3)$
$(\mathrm{add}(n,l_1,n_1),t_1)$	$(\mathrm{del}(n_2,l_2,n),t_2)$	$(\mathrm{del}(n,l_1,n_1),t_3)$
$(\mathrm{add}(n_1,l_1,n),t_1)$	$(\mathrm{del}(n,l_2,n_2),t_2)$	$(\mathrm{add}(n,l_2,n_2),t_3)$
$(\mathrm{add}(n_1,l_1,n),t_1)$	$(\mathrm{del}(n_2,l_2,n),t_2)$	$(\mathrm{del}(n_1,l_1,n),t_3)$ *if*$(n_1,l_1) \neq (n_2,l_2)$
$(\mathrm{del}(n,l_1,n_1),t_1)$	$(\mathrm{add}(n_2,l_2,n),t_2)$	$(\mathrm{del}(n_3,l_3,n),t_3)$
$(\mathrm{del}(n_1,l_1,n),t_1)$	$(\mathrm{add}(n,l_2,n_2),t_2)$	$(\mathrm{add}(n_3,l_3,n),t_3)$
$(\mathrm{del}(n_1,l_1,n),t_1)$	$(\mathrm{del}(n,l_2,n_2),t_2)$	$(\mathrm{add}(n_3,l_3,n),t_3)$
$(\mathrm{del}(n_1,l_1,n),t_1)$	$(\mathrm{del}(n_2,l_2,n),t_2)$	$(\mathrm{add}(n_2,l_2,n),t_3)$

The following theorem establishes the relationship between correctness, basic-input trees and the C-condition.

Theorem 1. *The following conditions are equivalent for a QL schedules S: (1) there is a basic-input-tree of S and the application of S is defined on each basic-input-tree of S; (2) there is a basic-input-tree of S on which the application of S*

is defined; (3) S is correct; (4) S fulfills the C-condition; (5) there is a tree on which the application of S is defined and all trees on which the application of S is defined are basic-input-trees of S.

Proof (Sketch) Clearly $1 \to 2 \to 3 \to 4$ and $5 \to 3$. We prove that 4 implies 1. First we proof that there is a basic-input-tree for which S is defined. This tree consists of all edges (m, l, n) for which $\phi_S((m, l, n), \text{del}(m, l, n))$ holds and all nodes m for which $\phi_S(m, \text{add}(m, l, n))$ augmented with edges from the root to the nodes in which no edge arrives. This is a basic-input-tree. Then we prove that the application of S is defined on each basic-input-tree of S. By induction on the length of S. In general we prove this property for the QL schedule $o.S$, supposing it holds for S. Finally 3 implies 5. Indeed, let S be defined on T, where T is not a basic-input-tree of S. T does not satisfy one of the four conditions of Definition 7. In each case this yields a contradiction. $\square$

Corollary 1. *It is decidable whether a QL schedule or a transaction is correct.*

For the sets that define the basic-input-trees and basic-output-trees we can derive the following properties.

Property 1. Let S be a correct QL schedule.

1. If $(m, l, n) \in E_I^+(S)(E_O^+(S))$ then $m, n \in N_I^+(S)(N_O^+(S))$. If $n \in N_I^-(S)$ $(N_O^-(S))$ then for all m, l holds $(n, l, m), (m, l, n) \in E_I^-(S)(E_O^-(S))$;
2. $E_I^+(S) \cap E_I^-(S) = \emptyset$, $E_O^+(S) \cap E_O^-(S) = \emptyset$, $N_I^+(S) \cap N_I^-(S) = \emptyset$, $N_O^+(S) \cap N_O^-(S) = \emptyset$;
3. $E_I^+(S)$, $E_O^+(S)$ do not contain two different edges ending in the same node;
4. $E_I^+(S)$, $E_O^+(S)$ do not contain a cycle. $\square$

The symmetry between basic-input-trees and basic-output-trees can be made even more clear by looking at reverse QL schedules.

Definition 9. *Let S be a QL schedule. S^σ, the reverse of S where every addition of an edge is substituted by the deletion of the edge and vice versa.*

Theorem 2. *Let S be a correct QL schedule and T_{in} be a basic-input-tree of S, and let $T_{out} = S[T_{in}]$. Then S^σ is a correct QL schedule and $T_{in} = S^\sigma[T_{out}]$.*

By ADD(S) we denote the set of edges that is really added by the QL schedule S, i.e., they are added without being removed again afterwards, and by DEL(S) we denote the set of edges that is really deleted by the QL schedule S, i.e., the are deleted without being added again afterwards.

Definition 10. *Let S be a correct QL schedule. We denote* $\text{ADD}(S) = \{(m, l, n) | \lambda_S((m, l, n), \text{add}(m, l, n))\}$ *and* $\text{DEL}(S) = \{(m, l, n) | \lambda_S((m, l, n), \text{del}(m, l, n))\}$.

Remark that two correct QL schedules with the same ADD and DEL are not necessarily equivalent. Cf. Example 3.

Definition 11. *Let $T = (N, E, r)$ be a* DT *and $E_1 = \{(m_i, a_i, n_i)\}$ be a set of edges. $T \cup E_1 = (N \cup \{n_i\}, E \cup E_1, r)$ and $T - E_1 = (N - \{n_i\}, E - E_1, r)$. Note that $T \cup E_1$ nor $T - E_1$ are necessarily* DT*'s.*

Theorem 3. *Let S be a correct QL schedule and T_{in} be a basic-input-tree of S. $S[T_{in}] = T_{in} \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$ is a basic-output-tree.*

Proof (Sketch) Clearly $T_{in} \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$ is the result of the application of S on T_{in}. We verify that $T_{in} \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$ is a basic-output-tree. $\square$

Property 2. Let S be a correct QL schedule. $\mathrm{ADD}(S)$ and $\mathrm{DEL}(S)$ fulfill the following conditions: (1) $\mathrm{ADD}(S) \cap \mathrm{DEL}(S) = \emptyset$; (2) if $(m_1, l_1, n), (m_2, l_2, n) \in \mathrm{ADD}(S)$ then $(m_1, l_1) = (m_2, l_2)$; (3) $\mathrm{ADD}(S)$ does not contain a cycle; (4) if $(m_1, l_1, n) \in \mathrm{ADD}(S)$ and $(n, l_2, n_2) \in \mathrm{DEL}(S)$ then $\exists (m_3, l_3, n) \in \mathrm{DEL}(S)$; and (5) if $(m_1, l_1, n) \in \mathrm{DEL}(S)$ and $(n, l_2, n_2) \in \mathrm{ADD}(S)$ then $\exists (m_3, l_3, n) \in \mathrm{ADD}(S)$.

The following theorem establishes the relationships between the addition and deletion sets, and the basic input and output sets.

Theorem 4. *Let S be a correct QL schedule. Then*

$N_O^+ = (N_I^+ - \{n | \exists m, l : (m, l, n) \in \mathrm{DEL}(S)\}) \cup \{n | \exists m, l : (m, l, n) \in \mathrm{ADD}(S)\}$
$N_O^- = (N_I^- \cup \{n | \exists m, l : (m, l, n) \in \mathrm{DEL}(S)\}) - \{n | \exists m, l : (m, l, n) \in \mathrm{ADD}(S)\}$
$E_O^+ = (E_I^+ - \mathrm{DEL}(S)) \cup \mathrm{ADD}(S)$, *and* $E_O^- = (E_I^- \cup \mathrm{DEL}(S)) - \mathrm{ADD}(S)$.

5 Equivalence and Serializability of QL schedules

The purpose of a scheduler is to schedule requests by users such that the resulting schedule is serializable. In this section we discuss the problem of deciding whether a schedule is serializable and, as a subproblem, whether two schedules are equivalent.
One possible approach for a scheduler is to introduce a locking mechanism such that operations of a certain user are only allowed if they do not conflict with previous operations of other users. Because non-conflicting operations can be commuted any schedule that is allowed by such a scheduler can be serialized. The following example shows that such an approach will be too strict in this case.

Example 7. Here is a QL schedule S that is defined on $T = (\{r\}, \emptyset, r)$, that is serializable, that has only one equivalent serial QL schedule S' but we cannot go from S to S' only by swapping:

$S = (\mathrm{add}(r, l_1, n_1), t_1)$, $(\mathrm{del}(r, l_1, n_1), t_2)$, $(\mathrm{add}(r, l_2, n_2), t_2)$, $(\mathrm{del}(r, l_2, n_2), t_2)$,
$(\mathrm{add}(r, l_2, n_2), t_1)$, $(\mathrm{del}(r, l_2, n_2), t_1)$.

However, it can also be shown that that any schedule over two given transactions will always result in the same DT if its result is defined.

Theorem 5. *Let S and S' be two QL schedules over the same set of transactions. If their applications on the* DT *T are both defined then the resulting* DT*'s are equal.*

As a consequence the problem of deciding whether two correct schedules over two given transactions are equivalent reduces to the problem of deciding whether their result is defined for the same DTs, which in turn can be decided with the help of the basic input and output sets.

Theorem 6. *Two (correct) QL schedules over the same set of transactions are equivalent iff they have the same set of basic-input-trees.*

Note that this theorem does not hold for two arbitrary QL schedules. Indeed $S_1 = (\text{add}(m, l, n), t)$ and $S_2 = (\text{add}(m, l, n), t), (\text{del}(m, l, n), t)$ have the same basic-input-trees and are not equivalent.

Theorem 7. *Two (correct) QL schedules S_1, S_2 over the same set of transactions are equivalent iff $N_I^+(S_1) = N_I^+(S_2)$, $N_I^-(S_1) = N_I^-(S_2)$, $E_I^+(S_1) = E_I^+(S_2)$ and $E_I^-(S_1) = E_I^-(S_2)$.*

We can use the basic input and output sets to decide whether one correct schedule can directly follow another correct schedule without resulting an an incorrect schedule.

Theorem 8. *Let S_1 and S_2 be two correct QL schedules. $S_1.S_2$ is correct iff $N_O^-(S_1) \cap N_I^+(S_2) = \emptyset$, $E_O^-(S_1) \cap E_I^+(S_2) = \emptyset$, $N_O^+(S_1) \cap N_I^-(S_2) = \emptyset$ and $E_O^+(S_1) \cap E_I^-(S_2) = \emptyset$*

The following theorems show how the basic input and output sets can be computed for a concatenation of schedules if we know these sets for the concatenated schedules.

Theorem 9. *Let $S_1, S_2, ..., S_n$ and $S_1.S_2...S_n$ be $(n + 1)$ correct QL schedules. Then*
$N_I^+(S_1...S_n) = \bigcup_{i=1}^n (N_I^+(S_i) - \bigcup_{k<i} N_O^+(S_k))$; $N_I^-(S_1...S_n) = \bigcup_{i=1}^n (N_I^-(S_i) - \bigcup_{k<i} N_O^-(S_k))$
$E_I^+(S_1...S_n) = \bigcup_{i=1}^n (E_I^+(S_i) - \bigcup_{k<i} E_O^+(S_k))$; $E_I^-(S_1...S_n) = \bigcup_{i=1}^n (E_I^-(S_i) - \bigcup_{k<i} E_O^-(S_k))$

Theorem 10. *Let $S_1, S_2, ..., S_n$ and $S_1.S_2...S_n$ be $(n+1)$ correct QL schedules. Then*
$N_I^+(S_1...S_n) = \bigcup_{i=1}^n (N_I^+(S_i) - \bigcup_{k<i} N_I^-(S_k))$; $N_I^-(S_1...S_n) = \bigcup_{i=1}^n (N_I^-(S_i) - \bigcup_{k<i} N_I^+(S_k))$
$E_I^+(S_1...S_n) = \bigcup_{i=1}^n (E_I^+(S_i) - \bigcup_{k<i} E_I^-(S_k))$; $E_I^-(S_1...S_n) = \bigcup_{i=1}^n (E_I^-(S_i) - \bigcup_{k<i} E_I^+(S_k))$

These previous theorems can be used to show that serializability is decidable.

Theorem 11. *It is decidable whether a given QL schedule is serializable.*

Proof (Sketch) There is a backtracking algorithm to decide whether a QL schedule is serializable. Indeed, we verify whether each transaction is correct (Corollary 1); we draw a graph that indicates which transactions can follow directly another transaction (Theorem 8); there is a Hamiltonian path, that fulfills Theorem 10 iff the QL schedule is serializable. $\square$

6 Schedules over the Same Set of Transactions

In the previous section we only considered queryless schedules, but in this section we consider all schedules. We start with generalizing the notions that were introduce for QL schedules.
A schedule S is called correct iff its corresponding QL schedule is correct. $\mathrm{ADD}(S) = \mathrm{ADD}(S')$ where S' is the QL schedule of S. Analogously for DEL, E_I^+, E_I^-, E_O^+, E_O^-, N_I^+, N_I^-, N_O^+, N_O^-.
To verify whether two correct schedules over the same set of transactions are equivalent, we first eliminate the queries and verify whether the resulting QL schedules are equivalent. (Cfr. Theorem 7). In this section we investigate the equivalence of two correct schedules over the same set of transactions and whose QL schedules are equivalent. Such schedules can sometimes be equivalent on all the DTs they are defined on, on only some of them or on none.
In order to show decidability of equivalence we introduce the following definitions.

Definition 12. *We define $ADD^N(S) = \{n|\exists m, l : (m, l, n) \in \mathrm{ADD}(S)\}$. Let m be a node of $\mathrm{ADD}^N(S)$. We denote by $Aroot(S, m)$ the node n that is an ascendant of m in $\mathrm{ADD}(S)$ and such that n has no parent in $\mathrm{ADD}(S)$. $Aroot(S, n)$ is uniquely defined by Property 2. The label path of the path from $Aroot(S, n)$ to n in $\mathrm{ADD}(S)$ is denoted by $Alabel(S, n)$.*

Remark that $\mathrm{ADD}(S)$ is always a forest and that $Aroot(S, m)$ is simply the root of the tree that contains m. Given a path expression pe and a label path lp we define the prefix of lp in pe, denoted as $\delta_{lp}(pe)$ as a set of path expressions that together represent the prefixes of the label paths represented by pe that end with lp. For instance $\delta_a(b//*) = \{b, b//*\}$

Definition 13. *Let pe be a path expression, lp be a label path and $l \in \mathcal{L}$. The prefix of lp in pe, denoted as $\delta_{lp}(pe)$ is defined by*

$$\begin{array}{ll}
\delta_\epsilon(pe) = \{pe\} & \delta_l(pe//*) = \delta_l(pe//l) = \{pe, pe//*\} \\
\delta_l(*) = \delta_l(l) = \{pe_\epsilon\} & \delta_{lp/l}(pe/*) = \delta_{lp/l}(pe/l) = \delta_{lp}(pe) \\
\delta_l(pe/*) = \delta_l(pe/l) = \{pe\} & \delta_{lp/l}(pe//*) = \delta_{lp/l}(pe//l) = \delta_{lp}(pe) \cup \delta_{lp}(pe//*)
\end{array}$$

Otherwise $\delta_{lp}(pe) = \emptyset$. Furthermore we define $\mathbf{L}(\delta_{lp}(pe)) = \bigcup_{pe_i \in \delta_{lp}(pe)} \mathbf{L}(pe_i)$.

Example 8. We now turn to a brief example.

- $\delta_{a/b}(a/*/*/b) = \delta_a(a/*/*) = \{a/*\}$
- $\delta_{a/b/c}(a//*/c) = \delta_{a/b}(a//*) = \delta_a(a) \cup \delta_a(a//*) = \{pe_\epsilon, a, a//*\}$
- $\delta_{a/b/c}(*//*) = \delta_{a/b}(*) \cup \delta_{a/b}(*//*) =$
  $\emptyset \cup \delta_a(*) \cup \delta_a(*//*) = \emptyset \cup \{pe_\epsilon\} \cup \{*, *//*\} = \{pe_\epsilon, *, *//*\}$
- $\delta_{c/d}(a/b/c/d) = \{a/b\}$; $\delta_{c/d}(a/*/*/d) = \{a/*\}$; $\delta_{c/d}(a//d) = \{a, a//*\}$
- $\delta_{c/d}(c//d) = \{p_\epsilon, c, c//*\}$ and $\delta_{b/c/d}(a//b//d) = \{a, a//*, a//b, a//b//*\}$

Theorem 12. *Let pe be a path expression and lp be a label path. $\delta_{lp}(pe)$ is uniquely defined, finite and is computable.* $\mathbf{L}(\delta_{lp}(pe)) = \{lp_1 | lp_1/lp \in \mathbf{L}(pe)\}$[4].

Given a graph G that defines a forest and two nodes n and m in $\mathcal{N}$ we define the function $\theta_G(n,m)$ such that $\theta_G(n,m) = \bar{\lambda}(p)$ if $p \in path_G(n,m)$ and $\theta_G(n,m) = \perp$ otherwise, where $\perp$ is a special string unequal to every label path.
With these definitions we can now characterize when exactly a node is in the result of a query after a certain schedule has been applied to a DT.

Theorem 13. *Let T be a* DT *and S a schedule such that $S[T]$ is defined, then $m \in \text{query}_{S[T]}(n,pe)$ iff at least one of the following holds*

- $n, m \notin \text{ADD}^N(S) \wedge\ \theta_{T-\text{DEL}(S)}(n,m) \in \mathbf{L}(pe)$
- $n \notin \text{ADD}^N(S) \wedge m \in \text{ADD}^N(S) \wedge \theta_{T-\text{DEL}(S)}(n, Aroot(S,m)) \in \mathbf{L}(\delta_{Alabel(S,m)}(pe))$
- $n, m \in \text{ADD}^N(S)\ \wedge\ \theta_{\text{ADD}(S)}(n,m) \in \mathbf{L}(pe)$

From the characterization above we can derive a characterization of when a node is in the result of a query after a certain schedule but not in the result after another schedule.

Theorem 14. *Let S_1 and S_2 be two schedules and T a* DT *such that $S_1[T]$ and $S_2[T]$ are defined. It then holds for every query $\text{query}(n,pe)$ and node $m \in \mathcal{N}$ that $m \in (\text{query}_{S_1[T]}(n,pe) - \text{query}_{S_2[T]}(n,pe))$ iff at least one of the following holds: (1) $n, m \notin \text{ADD}^N(S_1)\ \wedge\ \theta_{T-\text{DEL}(S_1)}(n,m) \in \mathbf{L}(pe)$; (2) $n \notin \text{ADD}^N(S_1) \wedge m \in \text{ADD}^N(S)\ \wedge \theta_{T-\text{DEL}(S_1)}(n, Aroot(S_1,m)) \in \mathbf{L}(\delta_{Alabel(S_1,m)}(pe))$; (3) $n, m \in \text{ADD}^N(S_1)\ \wedge\ \theta_{\text{ADD}(S_1)}(n,m) \in \mathbf{L}(pe)$, and at least one of the following holds: (1) $n, m \notin \text{ADD}^N(S_2)\ \wedge\ \theta_{T-\text{DEL}(S_2)}(n,m) \notin \mathbf{L}(pe)$; (2) $n \notin \text{ADD}^N(S_2) \wedge m \in \text{ADD}^N(S_2) \wedge \theta_{T-\text{DEL}(S_2)}(n, Aroot(S,m)) \notin \mathbf{L}(\delta_{Alabel(S_2,m)}(pe))$; (3) $n, m \in \text{ADD}^N(S_2) \wedge \theta_{\text{ADD}(S_2)}(n,m) \notin \mathbf{L}(pe)$.*

The purpose of the previous theorem is to obtain a necessary and sufficient condition for when such a DT and node m exist. To approximate T we define the sets $E^{min}(S)$ and $E^{max}(S)$ which can informally be described as the set of edges that should at least be in T and the the set of edges that can at most be in T, respectively.
We denote $E^{min}(S) = E_I^+(S)$ and $E^{max}(S) = (\mathcal{N} \times \mathcal{L} \times \mathcal{N}) - E_I^-(S)$. Remark that if S_1 and S_2 are two schedules over the same set of transactions and the QL schedules of S_1 and S_2 are equivalent then $E^{min}(S_1) = E^{min}(S_2)$ and $E^{max}(S_1) = E^{max}(S_2)$.
With these sets we can characterize when for two schedules over the same transactions and a certain query in those schedules there is a basic-input-tree such that a certain node is in the result of the query in the first schedule but not in the result of the query in the second schedule.

Theorem 15. *Let S_1 and S_2 be two correct schedules over the same set of transactions with equivalent QL schedules that both contain $Q = (\text{query}(n,pe), t)$ and*

[4] We identify lp_1/ϵ with lp_1

let $m \in \mathcal{N}$. We denote that part of the schedule S_i that comes before Q by S_i^Q. There is a basic-input-tree T of S_1 and S_2 for which $m \in (\mathrm{query}_{S_1^Q[T]}(n, pe) - \mathrm{query}_{S_2^Q[T]}(n, pe))$ iff there exists a path $p \in path_{E^{max}(S_1)}(n, m)$ such that (1) $\bar{\lambda}(p) \in \mathbf{L}(pe))$, (2) $E^{min}(S_1) \cup \{p\}$ is a forest and (3) the condition in Theorem 14 holds with T replaced with $E^{min}(S_1) \cup \{p\}$ and S_1 and S_2 replaced with S_1^Q and S_2^Q, respectively.

This final theorem suggests a decidable characterization since the paths p that are quantified over are paths in the graph $E^{max}(S_1)$ and this graph can be described by giving for a finite set of nodes either finite set of incident edges that are either all allowed or all not allowed. If this is decidable then we can decide the equivalence of two schedules over the same transactions by deciding for each query if there is a basic-input-tree such that the result of the query contains in one schedule a node that it does not contain in the other schedule or vice versa. This leads to the following conjecture.

Conjecture 1. It is decidable whether two schedules are equivalent.

7 Further Research

We have conjectured that it is decidable whether two schedules are equivalent. In future work we intend to prove this conjecture. Secondly, we will also look to incorporate the information contained in the Schema or DTD of an XML document to enhance the results in this paper. Using the additional information of DTDs is expected to allow more schedules to be serializable. Finally, we will also analyse the computational complexity of the decision problems.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan-Kaufmann, San Francisco, 1999.
2. P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison Wesley, Reading, Mass., 1987.
3. S. Dekeyser and J. Hidders. Path locks for XML document collaboration. In *Proceedings of the Third WISE Conference*, 2002.
4. S. Dekeyser and J. Hidders. A commit scheduler for XML databases. In *Proceedings of the Fifth Asia Pacific Web Conference*, Xi'an, China, 2003.
5. S. Dekeyser, J. Hidders, and J. Paredaens. A transaction model for XML databases. *World Wide Web Journal*, 2003. To appear.
6. J. Gray, G. Putzolo, and I. Traiger. Granularity of locks and degrees of consistency in a shared data base. In *Modeling in Data Base Management Systems.* North Holland, Amsterdam, 1976.
7. C. Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, Rockville, MD, 1986.
8. A. Silberschatz and Z. Kedem. Consistency in hierarchical database systems. *Journal of the ACM*, 27(1):72–80, 1980.
9. G. Weikum and G. Vossen. *Transactional Information Systems.* Morgan Kaufmann, 2002. ISBN: 1-55860-508-8.