QUEST: QUery-driven Exploration of Semistructured Data with Conflic$T$s and Partial Knowledge*

Yan Qi
Comp. Sci. and Eng.
Arizona State Univ.
yan.qi@asu.edu

K. Selçuk Candan
Comp. Sci. and Eng.
Arizona State Univ.
candan@asu.edu

Maria Luisa Sapino
Dip. di Informatica
Univ. of Torino
mlsapino@di.unito.it

Keith W. Kintigh
Sch.of Human Evol.&Social Change
Arizona State Univ.
kintigh@asu.edu

Abstract

An important reality when integrating scientific data is the fact that data may often be "missing", partially specified, or conflicting. Therefore, in this paper, we present an assertion-based data model that captures both *value-based* and *structure-based* "nulls" in data. We also introduce the `QUEST` system, which leverages the proposed model for *Qu*ery-driven *E*xploration of *S*emistructured data with conflic*T*s and partial knowledge. Our approach to integration lies in enabling researchers to observe and resolve conflicts in the data by considering the context provided by the data requirements of a given research question. In particular, we discuss how *path-compatibility* can be leveraged, within the context of a query, to develop a high-level understanding of conflicts and nulls in data.

1 Motivation and Related Work

Through a joint effort of archaeologists and computer scientists, we are developing an integrated framework of knowledge-based collaborative tools that will provide the foundation for a shared information infrastructure for archaeology and contribute substantially to a shared knowledge infrastructure of science [21]. Today, the incapacity to integrate data across projects cripples archaeologists' and other scientists' efforts to recognize phenomena operating on large spatio-temporal scales and to conduct crucial comparative studies [20, 21]. A major challenge with integration of data is that the meaning of an archaeological observation is rarely self-evident.

*Supported by NSF Grant "*Enabling the Study of Long-Term Human and Social Dynamics: A Cyberinfrastructure for Archaeology*"

1.1 Incompleteness and Inconsistencies

An important reality when integrating archaeological data is that entries (archaeological observations and interpretations) may often be "missing" or only partially specified. For example, one may not be able to associate a bone collected at a given site to the species and may use vague terms or references to a hierarchically higher concept in the biological taxonomy. Thus, researchers reach conflicting conclusions, not just because their primary data differ, but because they operationalize interpretive concepts differently [20].

Within the context of our efforts to determine the needs and challenges associated with archaeological information integration, a working group of domain experts selected datasets representing archaeological fauna recovered from two excavations in the western US [28]. The goal of the effort was to integrate these two datasets into one by using ontologies to map data codes to concepts shared by the datasets and to resolve the ambiguities (as much as possible) using ontologies. One outcome of this effort was the understanding that, even after a careful study of the data sets by the domain experts, there were parts of the data that could not be successfully mapped (e.g., while use of the actual taxonomic categories was consistent, investigators differed in how they dealt with bones that could not be fully identified). Nevertheless, in the context of a particular research question, archaeologists could identify reasonable means of addressing these inconsistencies.

Thus, reconciling data and classification schemes entails developing novel data integration techniques to allow query dependent integration, despite inherent inconsistencies. Our goal is to develop a tool to allow a researcher to extract sensibly integrated observations and consistent variables from potentially incomplete and inconsistent data archive. During query processing, the repository needs to integrate data from multiple sources, note and resolve conflicting and missing data. Where there are discrepancies or missing data, the system needs to allow the researcher to interpret results and resolve conflicts as she sees appropriate.

1.2 Related Work

In general, there are many different types of null values (e.g., existential, maybe, place holder, and partial), each of which reflects different knowledge or intuitions about why a particular piece of information is missing [8]. An early attempt at modeling semistructured data with missing and partial data is presented in [23]; authors used an object-based model, where *null*, *or-valued*, and *partial set* objects are used to handle partial and missing knowledge in semi-structured data. Although it is richer than standard semistructured data models, such as Object Exchange Model (OEM) [24, 7], and Document Object Model (DOM) [1], this model is more focused on *value* nulls and does not capture inconsistencies and missing knowledge in the structure of the data. In contrast, we propose a new model for semi-structured databases where different types of null values are represented uniformly. Each entry has an associated assertion; intuitively, an entry may be thought of as being in the database iff the corresponding assertion is true. Although the idea of using assertions (constraints) to handle null values in relational databases is not new (Imielinski-Lipski [15, 16], Liu [22], Candan [8]), the use of constraints for a unified way of handling different types of nulls within the context of hierarchical data and metadata is an open problem.

Knowledge integration from diverse sources involves *matching* and *integration*. There is extensive work in the area of matching schemas and data when integrating independent sources. Our focus, in this paper, however is not on the matching, but on dealing with conflicts that arise during integration. Conflict resolution has also been studied in the context of active databases and production rule systems [3, 17]. Most of these study what to do when multiple active production rules with conflicting heads request that an atom be both added and deleted simultaneously. In contrast, we attempt to evaluate queries and resolve conflicts in answers to queries spanning multiple data sources. Furthermore, unlike the related work in this area, we will explore the application of these within semi-structured data and metadata.

In their work on nondeterministic choices in logic programming languages, Zaniolo [31, 32] and his colleagues suggest that in logic database languages, one may wish to express the fact that only one of several possible ways of satisfying an atom is nondeterministically selected. They then use this to define a choice semantics for logic programs with negation. Multiple model semantics, like the *2-valued, stable model semantics*,[12], or the *3-valued finite failure stable model semantics* [13] associate multiple, equally likely, models to the given knowledge base, each one corresponding to a possible context, or a possible consistent scenario described by the knowledge base. Problem solvers interact with *truth maintenance systems* (TMSs) [9], that record and maintain the reasons for the possible context (*belief sets*) under consideration. Sentences are associated with their justifications, which indicate what assumptions need to be changed if they need to be invalidated. In this paper, we show that we can leverage the special hierarchical structure of the data and knowledge taxonomies to develop efficient and specialized algorithms, rather than having to use general purpose truth maintenance solutions. We use path query instances to provide contexts in which conflicts can be resolved. Like us, Piazza [14] and HepToX [5] also recognize that it is unrealistic to expect an independent data source entering information exchange to agree to a global mediated schema or to perform heavyweight operations to map its schema to every other schema in the group. Unlike these, however, we recognize that while collating information from multiple sources, the knowledge that is acquired may be incomplete or inconsistent either in data values, structural relationships between data elements, or both. Yet, since the base data reflect what is currently known, data and interpretations from different sources may be important to keep *as is*, even when they may be conflicting with each other. We argue that an ultimate integrated view of multiple data sets is often not possible, and in fact is often not needed. Therefore, unlike related work [27, 10] in repairing inconsistencies in XML data using available external domain knowledge, such as functional dependencies or DTDs, our aim is to *maintain* the inconsistencies in the data and allow the researcher to resolve conflicts within the context of a given query.

1.3 Contributions of this Paper

Effective use of archaeological data requires on-the-fly data integration, where discrepancies or incomplete information is properly dealt with within the context of the given query. In this paper, we first present a data model which captures not only *value*-based, but also *structure-based* nulls in semistructured data and metadata. In particular, we suggest that it is most effective to reconcile data source observations with data requirements of a query rather than attempting global reconciliation of data sources. We refer to this as *query driven ad hoc data integration and exploration [19].* This enables us to constrain the incompatibilities of the data within the context of the question itself to reduce the complexity of the problem. In this paper, we also present an overview of a system, called `QUEST`, which we are developing to leverage the proposed model for exploratory research on the incomplete and conflicting data, based on the query driven ad hoc data integration and exploration paradigm. We are currently developing efficient algorithms to process queries on (null-valued) semi-structured data in the presence of a multitude of such alternatives, without having to materialize all alternatives.

2 Assertion-based Data Representation and Basic Null Assertions

To provide a uniform treatment to value and structure-nulls, we shred the semistructured data into its object nodes. Shredding is used in relational storage of XML data, where each node is represented as a tuple of the form $\langle node_id, label, type, value, parent_id\rangle$ [11, 29]. The model we describe below is reminiscent of well accepted node-labeled semi-structured data models, such as DOM [1] and their shredding into tuples [11, 29].

2.1 Constraint-based Data Representation

Let I denote the set of object node identifiers and let D be the domain of node tags[1]. We represent hierarchical data as a set, N, of object nodes, where each object node $n \in N$ is represented as a 3-tuple (id, tag, pid):

- $n.id \in I$ is the unique id of the object node,
- $n.tag \in D \cup I$ is its tag, and
- $n.pid \in I \cup \{\top\}$ is its parent's identifier.

If $n.pid = \top$, then n is referred to as the root of the data. If $n.tag \in I$, then its value is an object reference. The object nodes in N are constrained such that they collectively form a tree structure:

C1. *No node can be its own parent:* $\forall n_i \in N, n_i.id \neq n_i.pid$.

C2. *No two distinct nodes can have the same ID:* $\forall n_i \neq n_j \in N, n_i.id \neq n_j.id$.

C3. *All non-root nodes have a parent in the document:* $\forall n_i \in N, (n_i.pid = \top) \vee (n_i.pid \in I)$.

C4. *There is only one root:* $\forall n_i \in N, (n_i.pid = \top) \rightarrow (\nexists n_j \neq n_i \;\; n_j.pid = \top)$

C5. *Parent relationship between two nodes is captured by attribute "pid":* $\forall n_i, n_j \in N, parent(n_i, n_j) \leftrightarrow (n_i.id = n_j.pid)$.

C6. *Ancestor relationship between two nodes is defined using the parent relationship:*
$\forall n_i, n_j \in N, ancestor(n_i, n_j) \leftrightarrow$
$\exists m_1, m_2, \ldots, m_K \in N, K \geq 0, \;\; s.t.$
$parent(n_i, m_1) \wedge parent(m_1, m_2) \wedge$
$\ldots \wedge parent(m_K, n_j)$.

C7. *There are no cycles in the data:* $\forall n_i, n_j \in N,$
$ancestor(n_i, n_j) \rightarrow \neg ancestor(n_j, n_i)$.

These constraints describe hierarchically structured data without nulls. Next, we discuss how to extend this constraint model with value- and structure-nulls in a uniform manner.

[1]For simplicity of the presentation, we combine label, type, and value into a single tag.

2.2 Value- and Structure-Nulls

A value-null commonly occurs when the value of a node can not be determined for certain. E.g.,

- *"Node &5's tag can be 4, 6, or 9."*

is a value null. Structure-nulls, on the other hand, occur when the structural relationship between the data nodes can not be determined in certain. For example,

- *"Node &5 is a child of node &3 or &4".*
- *"Either node &5 or &6 is a child of node &3".*

are structure nulls. When nodes suffer from both value- and structural uncertainties or inconsistencies, we refer to these as hybrid-nulls. Naturally, the object node based representation in Subsection 2.1 is not suitable to describe *disjunctions* or *non-existence requirements* that form the basis of various types of nulls [8]. Therefore, we present a basic *choice assertion* construct, which forms the basis of nulls.

2.3 Basic Choice Assertions

We refer to a triple, $\bar{a} = \langle \overline{id}, \overline{tag}, \overline{pid}\rangle$, where $\overline{id} \subseteq I$, $\overline{tag} \subseteq (D \cup I)$, and $\overline{pid} \subseteq (I \cup \{\top\})$, as a *basic choice assertion* (or *assertion* in short). The set of all assertions corresponding to a given data is denoted as A. For example, $\langle\{\&2, \&3\}, \{Cow, Bison\}, \{\&7, \&8\}\rangle$ is a basic choice assertion. Intuitively, each assertion in A declares constraints on *id*, *tag*, and *pid* related to a *single* object node in N.

Informally, a choice assertion states that "one of all possible alternatives described by the $\overline{id}$, $\overline{tag}$, and $\overline{pid}$ sets is true". If all the sets in an assertion are singular valued (e.g. of the form $\langle\{\&2\}, \{Bison\}, \{\&7\}\rangle$), then the assertion corresponds to a single object node, and vice versa: e.g., the object node $(\&2, Bison, \&7)$ could be asserted as $\langle\{\&2\}, \{Bison\}, \{\&7\}\rangle$. These types of assertions are referred to as *singular choice assertions*[2]. We classify the choice assertions into two categories: *positive* and *negative* choice assertions.

2.3.1 Positive Choice Assertions

Positive choice assertions do not contain any empty sets, but contain at least one non-singular set. For example, $\langle\{\&1, \&2\}, \{Bison, Cow\}, \{\&3\}\rangle$ is a positive choice assertion. We define the **semantics** of the positive assertion, $\bar{a}_i = \langle \overline{id}_i, \overline{tag}_i, \overline{pid}_i\rangle$, in terms of a many-to-1 mapping, $\mu : A \rightarrow N \cup \{\perp\}$, from the set, A, of assertions to nodes in N, such that

$$\mu(\bar{a}_i) = n \in N \longrightarrow (n.id \in \overline{id}_i) \wedge (n.tag \in \overline{tag}_i) \wedge (n.pid \in \overline{pid}_i).$$

The fact that the mapping, μ, is many-to-1 implies that

[2]Any data without null-values can be represented as a set of singular assertions.

- each positive assertion describes properties of a *single* object node, while
- properties of a single object node may be described by multiple assertions.

If $\mu(\bar{a}_i) = \perp$, then the assertion $\bar{a}_i$ is ignored.

Example 2.1 *Let* $\langle\{\&1\}, \{Pelvis\}, \{\&2, \&3\}\rangle$ *be a choice assertion. Informally, this assertion means that the value of the object node with id* &1 *is "Pelvis" and its parent is either* &2 *or* &3*. However, the assertion does not mean that* &1 *has two parents. In other words, this assertion is about a* single *node, whose parent we cannot ascertain without other assertions.*

2.3.2 Negative Choice Assertions

Negative choice assertions, on the other hand, contain at least one empty set. For example, $\langle\{\&1, \&2\}, \{Pelvis\}, \emptyset\rangle$ is a negative assertion. We define the **semantics** of a negative assertion in terms of the following non-existence constraints, corresponding to various empty set scenarios:

- Scenario: $[\overline{\mathbf{id_i}} = \emptyset,\ \overline{\mathbf{tag_i}} \neq \emptyset,\ \overline{\mathbf{pid_i}} \neq \emptyset]$
  Const.: $\nexists n \in N\ \ s.t.\ \ (n.tag \in \overline{tag}_i) \wedge (n.pid \in \overline{pid}_i)$.
- Scenario: $[\overline{\mathbf{id_i}} = \emptyset,\ \overline{\mathbf{tag_i}} = \emptyset,\ \overline{\mathbf{pid_i}} \neq \emptyset]$
  Const.: $\nexists n \in N\ \ s.t.\ \ (n.pid \in \overline{pid}_i)$.
- Scenario: $[\overline{\mathbf{id_i}} = \emptyset,\ \overline{\mathbf{tag_i}} \neq \emptyset,\ \overline{\mathbf{pid_i}} = \emptyset]$
  Const.: $\nexists n \in N\ \ s.t.\ \ (n.tag \in \overline{tag}_i)$.
- Scenario: $[\overline{\mathbf{id_i}} = \emptyset,\ \overline{\mathbf{tag_i}} = \emptyset,\ \overline{\mathbf{pid_i}} = \emptyset]$
  Const.: $\nexists n \in N$.
- Scenario: $[\overline{\mathbf{id_i}} \neq \emptyset,\ \overline{\mathbf{tag_i}} = \emptyset,\ \overline{\mathbf{pid_i}} = \emptyset]$
  Const.: $\nexists n \in N\ \ s.t.\ \ (n.id \in \overline{id}_i)$.
- Scenario: $[\overline{\mathbf{id_i}} \neq \emptyset,\ \overline{\mathbf{tag_i}} \neq \emptyset,\ \overline{\mathbf{pid_i}} = \emptyset]$
  Const.: $\nexists n \in N\ \ s.t.\ \ (n.tag \in \overline{tag}_i) \wedge (n.id \in \overline{id}_i)$.
- Scenario: $[\overline{\mathbf{id_i}} \neq \emptyset,\ \overline{\mathbf{tag_i}} = \emptyset,\ \overline{\mathbf{pid_i}} \neq \emptyset]$
  Const.: $\nexists n \in N\ \ s.t.\ \ (n.id \in \overline{id}_i) \wedge (n.pid \in \overline{pid}_i)$.

2.4 Interpretation of a Set of Assertions

A set, A, of basic choice assertions can be thought of being composed of a positive assertion set, A^+, and a negative assertion set, A^-. An interpretation of A is a data instance, which (a) satisfies the structural constraints, describing the hierarchy, in Section 2.1, (b) conforms to a mapping μ, which satisfies the constraints imposed by the positive assertion set, A^+, and (c) satisfies all the non-existence constraints imposed by the negative assertion set, A^-. Given an assertion set, there may be zero, one, or more interpretations. In a sense, the positive assertions *produce* candidate interpretations, while the negative assertions, A^-, prune the space of alternative conforming data instances.

2.5 Compatible Assertions

Assertions that conflict, for example $\langle\{\&1\}, \{Bison\}, \{\&2\}\rangle$ and $\langle\{\&1\}, \{Cow\}, \{\&3\}\rangle$, may coexist in the data. Thus, we introduce the concept of *compatibility* among assertions.

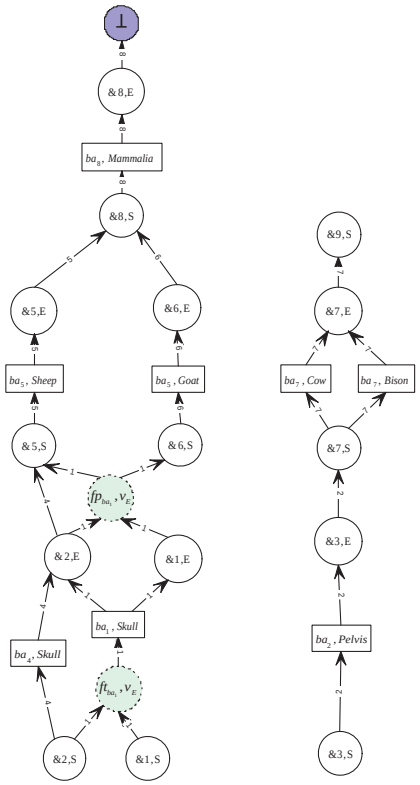


Figure 1: G^+ for a set of positive choice assertions

- A pair of positive assertions are compatible if they do neither lead to the indeterminate tags, nor imply a node with multiple parents or a cycle.
- A positive choice assertion and a negative choice assertion are compatible, if at least one choice in the positive assertion can be accepted without violating the negative constraints.

Note that although it is possible to identify consistent models (i.e., sets which consist of compatible assertions) of a given set of choice assertions, and clean the data (for instance, by choosing a *maximal* model among the alternatives), we argue that (especially in scientific data integration domain, where consistency can not be expected during research, until ultimately one model is shown to be correct) it is more meaningful to refrain from early data cleaning and resolve the conflicts within the context of the user's queries.

3 Integrated Representation of a Set of Positive Assertions

Given a set of assertions, `QUEST` integrates available positive assertions in a graph-based representation, G^+, which captures the intended structural relationships between object nodes as well as the choice semantics underlying the nulls. In this section, we provide an example of this graphical representation. The details of the model are beyond the scope of this paper.

Example 3.1 *Let us consider the assertions,*

$$ba_1 = \langle\{\&1, \&2\}, \{Skull\}, \{\&5, \&6\}\rangle$$
$$ba_2 = \langle\{\&3\}, \{Pelvis\}, \{\&7\}\rangle$$
$$ba_3 = \langle\emptyset, \{Deer\}, \emptyset\rangle$$
$$ba_4 = \langle\{\&2\}, \{Skull\}, \{\&5\}\rangle$$
$$ba_5 = \langle\{\&5\}, \{Sheep\}, \{\&8\}\rangle$$
$$ba_6 = \langle\{\&6\}, \{Goat\}, \{\&8\}\rangle$$
$$ba_7 = \langle\{\&7\}, \{Cow, Bison\}, \{\&9\}\rangle$$
$$ba_8 = \langle\{\&8\}, \{Mammalia\}, \{\top\}\rangle$$

which outline hierarchical relationships among bones and taxa. For example, "Skull" belongs to the taxon "Goat", which is a branch of "Mammalia". In detail, ba_1 is a basic choice assertion, informing that there is an object node, whose tag is "Skull", but neither its identifier nor its parent can be exactly determined (i.e., the position of the skull in the hierarchy is not exactly identified). Another poorly identified data involves basic choice assertion, ba_7, where the tag of the object node can have just one of the two alternative values. The negative assertion, ba_3, states that there is no object node in the data with "Deer" as its tag.

The directed graph G^+ based on this set of positive assertions is shown in Figure 1. We use solid-lined circles to denote the graph vertices corresponding to known object ids; for each object node there are two solid vertices (start, S, and end, E). Since each assertion needs to be mapped to a single object node, dashed vertices in the graph act as mutual exclusion constraints. The possible values for the object node tags are shown in rectangular vertices. Below, we describe the salient points of the G^+ using this example.

- *First, note that, ba_3 can not be represented in G^+ as it is not a positive assertion.*
- *Since ba_1 has a non-singleton $\overline{id}$, the mutual exclusion nodes $\langle fp_{ba_1}, v_E\rangle$ and $\langle ft_{ba_1}, v_E\rangle$ (for parent and tag respectively) are introduced. Each mutual exclusion node ensures that only one of the incoming edges supported by a given basic assertion is allowed in a given interpretation of data.*
- *Some nodes, such as &9, do not have any associated assertions; thus only the corresponding start vertices, such as $\langle\&9, S\rangle$, are included; i.e., it is impossible to determine their tags or parent with the available information. In fact, G^+ may be composed of several unconnected sub-graphs.*
- *There are two different assertions, ba_1 and ba_4, describing the parent/child relationship between nodes labeled &2 and &5.*

  These two assertions have to be seen as two non-coordinated statements. Therefore, they neither support each other nor weaken the respective claims. More specifically, the non-choice assertion $ba_4 = \langle\{\&2\}, \{Skull\}, \{\&5\}\rangle$ does not make any of the two alternative choices in the assertion $ba_1 = \langle\{\&1, \&2\}, \{Skull\}, \{\&5, \&6\}\rangle$ any more likely, until interpreted by a researcher within the appropriate context.

4 Beyond Basic Assertions

Each positive basic choice assertion describes a constraint on the relationship between a node, its tag, and its parent. Since by definition of the mapping, μ, each assertion $\bar{a}_i$ is interpreted independently from the others, there is no way to correlate the choice statements that have to hold for more than one node. Thus, any null which requires a constraint on two or more (non parent-child) nodes cannot be described using a single basic choice assertion:

- *Nodes &5 and &6 have either &8 or &9 as their common parent.*

  This statement requires a mapping, μ, where

  $$(\mu(\bar{a}_i) \in N \rightarrow (\mu(\bar{a}_i).id \in \{\&5\}) \wedge (\mu(\bar{a}_i).pid \in \{\&8, \&9\})) \wedge$$
  $$(\mu(\bar{a}_j) \in N \rightarrow (\mu(\bar{a}_j).id \in \{\&6\}) \wedge (\mu(\bar{a}_j).pid \in \{\&8, \&9\})) \wedge$$
  $$(\mu(\bar{a}_i).pid = \mu(\bar{a}_j).pid)\,.$$

  The last conjunct $(\mu(\bar{a}_i).pid = \mu(\bar{a}_j).pid)$ is a coordination requirement that can not be captured using basic choice assertions[3].

- *Node &2 has either &5 or &6 as its child; if the child is &5 the tag of the child is "Antelope" and if it is &6, the tag of the child is "Deer".*

  This statement requires a mapping μ, where

  $$(\mu(\bar{a}_i) \in N \rightarrow (\mu(\bar{a}_i).id \in \{\&5\}) \wedge$$
  $$\quad (\mu(\bar{a}_i).tag \in \{\text{“}Antelope''\}) \wedge$$
  $$\quad (\mu(\bar{a}_i).pid \in \{\&2\})) \wedge$$
  $$(\mu(\bar{a}_j) \in N \rightarrow (\mu(\bar{a}_j).id \in \{\&6\}) \wedge$$
  $$\quad (\mu(\bar{a}_j).tag \in \{\text{“}Deer''\}) \wedge$$
  $$\quad (\mu(\bar{a}_j).pid \in \{\&2\})) \wedge$$
  $$(\mu(\bar{a}_i).pid \neq \mu(\bar{a}_j).pid).$$

  Once again, last conjunct $(\mu(\bar{a}_i).pid \neq \mu(\bar{a}_j).pid)$ is a coordination requirement[4].

- *Node &2 has either the set of nodes {&5, &6} as its children or the set {&7,&8}.*

  This statement[5] requires a mapping, μ, where

  $$(\mu(\bar{a}_i) \in N \rightarrow (\mu(\bar{a}_i).id \in \{\&5\}) \wedge (\mu(\bar{a}_i).pid \in \{2\})) \wedge$$
  $$(\mu(\bar{a}_j) \in N \rightarrow (\mu(\bar{a}_j).id \in \{6\}) \wedge (\mu(\bar{a}_j).pid \in \{\&2\})) \wedge$$
  $$(\mu(\bar{a}_k) \in N \rightarrow (\mu(\bar{a}_k).id \in \{\&7\}) \wedge (\mu(\bar{a}_k).pid \in \{\&2\})) \wedge$$
  $$(\mu(\bar{a}_l) \in N \rightarrow (\mu(\bar{a}_l).id \in \{8\}) \wedge (\mu(\bar{a}_l).pid \in \{\&2\})) \wedge$$
  $$(\mu(\bar{a}_i).pid \neq \mu(\bar{a}_k).pid) \wedge$$
  $$(\mu(\bar{a}_i).pid \neq \mu(\bar{a}_l).pid) \wedge$$
  $$(\mu(\bar{a}_j).pid \neq \mu(\bar{a}_k).pid) \wedge$$
  $$(\mu(\bar{a}_j).pid \neq \mu(\bar{a}_l).pid).$$

  The last four conjuncts require coordination.

[3]Note that a simpler statement *"Node &5 has either &8 or &9 as its parent"* can be captured by a basic assertion of the form $\langle\{\&5\}, D, \{\&8, \&9\}\rangle$, **plus** the structural axiom which enforces a single parent to each node.

[4]Note that the simpler statement "*Node &2 has either &5 or &6 as its child*" can be captured by a basic assertion of the form $\langle\{\&5, \&6\}, D, \{\&2\}\rangle$.

[5]Note again that a statement *"Node &2 has either &5 or &7 as its child*" can be captured by a basic assertion of the form $\langle\{\&5, \&7\}, D, \{\&2\}\rangle$.

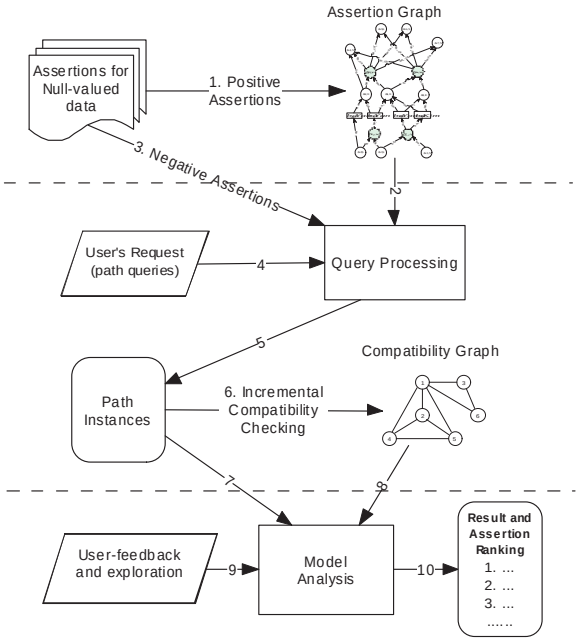


Figure 2: Overview of the query evaluation and exploration processes

- *Node &5's tag is either "Antelope" or "Deer". Node &6's tag is either "Antelope" or "Deer". Furthermore, &5 and &6 have the same tag.*

  This statement requires a mapping, μ, where

  $$(\mu(\bar{a}_i) \in N \rightarrow (\mu(\bar{a}_i).id \in \{\&5\}) \wedge$$
  $$(\mu(\bar{a}_i).tag \in \{\text{“}Antelope'', \text{“}Deer''\}) \wedge$$
  $$(\mu(\bar{a}_j) \in N \rightarrow (\mu(\bar{a}_j).id \in \{\&6\}) \wedge$$
  $$(\mu(\bar{a}_j).tag \in \{\text{“}Antelope'', \text{“}Deer''\}) \wedge$$
  $$(\mu(\bar{a}_i).tag = \mu(\bar{a}_j).tag).$$

  The last conjunct requires coordination.

Comparing the complex statement examples above with their simpler counterparts, which can be captured using basic choice assertions, illustrates that the problem arises from the need to enforce coordinated selections (among possible alternatives) for multiple nodes. The implementation of coordinated choice assertions is beyond the scope of this paper.

5 Query- and Feedback-Driven Exploration Process

Figure 2 shows the outline of the query-driven exploration process underlying the QUEST: first, based on the available positive assertions, QUEST creates an assertion graph (representing the null-valued document; see Figure 1). When the user provides a path query, matching path instances corresponding to the query and satisfying the pruning constraints imposed by negative assertions are identified. In a sense

- positive assertions *produce* path instances. In case there are conflicts among the given positive assertions, this results in alternative solution *models*, consisting of intra-compatible, but pairwise-incompatible sets of paths.
- negative assertions *delete* path instances from the result. Thus, *negative assertions* can reduce the size or collapse solution models.

Naturally, the number of these solution models can be large. Therefore, a particular challenge is to postpone the computation and visualization of these alternative solution models until absolutely necessary. Thus, QUEST helps the user explore the alternative solution spaces in an informed manner without having access to explicit materializations of the solution models: QUEST first identifies an initial subset of matches to the query and constructs (in an incremental way) an intermediary path compatibility graph during query evaluation. Once the query is evaluated and the path compatibility graph is constructed, the user can interact with QUEST to turn on and off various assertions and observe how the solution set (and the solution models) are affected. Pairwise compatibility graphs of logic rules are also used in non-monotonic reasoning systems [25]. Unlike these, however, in QUEST, the compatibility graphs are not only for the base rules (or assertions), but for the result paths obtained within the context of a query. This enables the user to explore the available data within the context of a query and *drill-down* to assertions or *zoom-out* to solution models. Once the user feedback is reflected on the assertions, the user is provided with a new subset of ranked results and the feedback-based exploration process is repeated. Below, we provide sketches of these steps.

5.1 Path Query and Results

Let us focus on path queries of type $P^{\{/,//,*\}}$ [2]. In QUEST, a path query is represented as

$$q = \theta_1 t_1 \theta_2 t_2 \ldots \theta_q t_q, \quad \theta_i \in \{/, //\}, \quad t_i \in D \cup \{*\},$$

where t_i are query tags (including "*" wildcards) and θ_i are parent/child or ancestor/descendant axes. An example of such a query is $'/Mammalia/Sheep/Skull'$. Results for a given path query are included in a set $R = \{r_1, r_2, \ldots, r_m\}$, where for each $r_i \in R$, we have

$$r_i = v_{i,1}[e_{i,1}]v_{i,2}[e_{i,2}]\ldots[e_{i,q-1}]v_{i,q}.$$

Here, $v_{i,j}$ is a label for one vertex in the assertion graph and $e_{i,j}$ is a set of labels for the assertions supporting the edge connecting the node $v_{i,j-1}$ and $v_{i,j}$. For example, the following is a result for the above query:

$\langle \&8, E\rangle[\{\langle ba_8, -\rangle\}]\langle ba_8, Mammalia\rangle[\{\langle ba_8, -\rangle\}]\langle \&8, S\rangle[\{\langle ba_5, -\rangle\}]$
$\langle \&5, E\rangle[\{\langle ba_5, -\rangle\}]\langle ba_5, Sheep\rangle[\{\langle ba_5, -\rangle\}]\langle \&5, S\rangle[\{\langle ba_1, -\rangle\}]$
$\langle fp_{ba_1}, v_E\rangle[\{\langle ba_1, -\rangle\}]\langle \&1, E\rangle[\{\langle ba_1, -\rangle\}]\langle ba_1, Skull\rangle[\{\langle ba_1, -\rangle\}]$
$\langle ft_{ba_1}, v_E\rangle[\{\langle ba_1, -\rangle\}]\langle \&1, S\rangle.$

Note that a valid path cannot contain any loops and for each data node on the path S and E vertices as well as the assertion labels need to match.

5.2 Path Compatibility Graph

Because of conflicting assertions, all results satisfying a path query might not be compatible. For example, two paths can assume that a given object node has different parents or two paths considered together may imply a loop. Furthermore, the mutual exclusion nodes introduced in Section 3 can render paths that share a given mutual exclusion node in different ways incompatible with each other. `QUEST` captures the compatibility between paths and sets of paths using a reflexive and symmetric "$\sim$" relation:

- Given two path instances p_i and p_j, $p_i \sim p_j$ iff the path instances together do not violate any structural constraints introduced in Section 2.
- Given a path instance p' and a set of path instances $P = \{p_1, p_2, ..., p_N\}$, $p' \sim P$, if and only if $\forall p_i \in P$, $p' \sim p_i$.
- Given two sets of path instances $P = \{p_1, p_2, ..., p_N\}$ and $Q = \{q_1, q_2, ..., q_M\}$, $P \sim Q$ if and only if $\forall p_i \in P, p_i \sim Q$.

Given a set of paths, P, a compatibility graph, C, captures all pairwise compatibility relationships.

5.3 Result Exploration

Let us assume that a path query q results in a set $R = \{p_1, p_2, \ldots, p_N\}$ of paths. As stated above, not all of these paths are compatible with each other. Therefore, `QUEST` provides various result exploration options to the user to enable her to get a high level understanding of the available data relative to her query:

- *Checking whether a given set, P, of paths is a model*; i.e., checking whether the given set of paths are compatible with each other. The result set, R, being a model would imply that the data does not contain any conflict relative to this query.
- Given a path p and a set of paths P, *checking whether $p \sim P$ or $p \not\sim P$.*
- *Given a path instance p and a set P, computing the number of path instances in P that are compatible with p.* This number informs the user regarding the degree of compatibility of the path p with others in P.
- *Given a path instance $p \in P$, computing the number of different models in which p occurs.* This informs the user regarding how supported each path is with the available knowledge.
- *Given a path instance $p \in P$, computing the number of models that would collapse when p is removed.* This informs the user regarding the entropy introduced by p in the integrated system.

Note that, additionally, the models themselves can be weighed based on their sizes or their compatibilities with other models. This information, then can be propagated to the weights of the paths included in these models. With these, it is possible to rank result paths and provide users with alternative exploration opportunities to observe the results, based on different definitions of likelihood (Figure 3(a)). The user can pick and choose between available result paths in an informed manner and observe the impact to the assertion and path compatibility graphs immediately. In particular, when a path is marked invalid by the user,

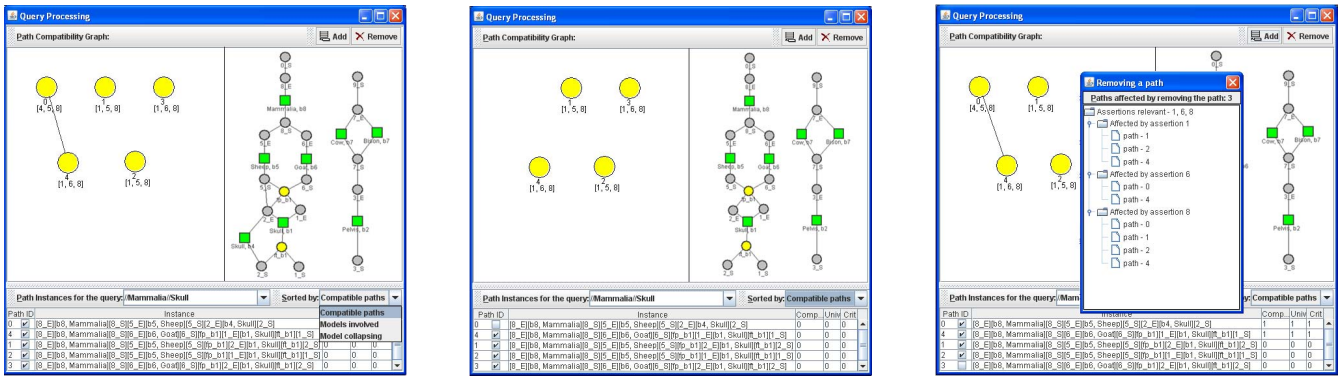
- if the path can be eliminated without affecting any other paths (by eliminating some choice in an assertion or by removing an assertion altogether), then this alternative is executed (Figure 3(b));
- if there is no way to remove it without affecting the assertions that support other paths, then the paths that might be impacted and the corresponding assertions are highlighted (Figure 3(c)).

Note that users are not always interested in ranking the result paths, but in ranking those assertions that generate and constrain the various solutions and solution models. Therefore, to support ranking of the assertions, we further propagate the various scores to the assertions on the paths. This enables the user to pick and choose between available assertions in an informed manner and observe the impacts of her actions on the solution immediately.

5.4 Computation

A model, composed of compatible result paths, corresponds to a maximal clique in the compatibility graph. Maximal cliques in a graph can be exponential in the number of vertices [26]. There are polynomial time delay algorithms for enumeration of cliques (i.e., if the graph of size n contains C cliques, the time to output all cliques is bounded by $O(n^k C)$ for some constant k) [18], but in general graphs, C can be exponential in n; for example as many as $3^{n/3}$ in Moon-Moser's graphs [26]. We, on the other hand, see that it is possible to avoid enumeration of cliques or finding of the maximal cliques in the entire compatibility graph, when supporting many of the relevant exploration tasks. For instance, the task of counting the number of maximal cliques a path occurs in can be performed by counting those maximal cliques containing only its neighbors. For sparse compatibility graphs, this can lead to significant gains in computation time. When the compatibility graph is dense, on the other hand, the number and sizes of cliques need to be estimated using alternative analysis techniques.

Thus, we are currently developing efficient algorithms to process queries on (null-valued) semi-structured data in the presence of a multitude of alternative models, without having to materialize all alternatives. In particular, we are exploring polynomial-time path and assertion ranking techniques based on structural analysis of the path and assertion compatibility graphs.



(a) (b) (c)

Figure 3: (a) Result visualization and exploration screen; (b) elimination of the path #0 changes the assertion graph accordingly; (c) elimination of path #3, on the other hand, would affect other paths in the result

5.5 Tree Queries

A tree query can be processed navigationally or split into multiple path queries and their structural joins [4, 30]. In QUEST, tree queries are handled as an extension of path query processing. After paths that satisfy the path sub-queries are identified, they need to be put together to form answers to the tree query. When paths might be incompatible, each set of paths that is put together to form an answer must be constrained to be self compatible. Therefore tree query processing involves merging of the ranked paths from multiple subqueries subject to compatibility constraints.

6 Conclusion

In this paper, we presented an *assertion*-based data model to describe hierarchical data and meta-data. We then extended this model with *basic choice assertions* which enables us to describe various types of value- and structure-based nulls in a uniform manner. We also highlighted the need for *coordinated* assertions to describe certain types of nulls. We introduced a graphical representation for hierarchical data with nulls and discussed how to enable query execution and query-driven data exploration processes using this graphical representation. We introduced the concept of path-compatibility and we highlighted how results of a query can be leveraged to have develop a high-level understanding of conflicts in the data. We also provided an overview of the QUEST system which leverages the concepts introduced in this paper to support exploratory research on incomplete and conflicting data.

References

[1] Document object model (dom) level 1 specification. http://www.w3.org/TR/REC-DOM-Level-1/.

[2] Xquery. http://www.w3.org/TR/xquery/.

[3] R. Agrawal, R. J. Cochrane, and B. G. Lindsay. On maintaining priorities in a production rule system. VLDB 1991.

[4] S. Al-Khalifa, *et al.* Structural joins: A primitive for efficient xml query pattern matching. *ICDE*, 2002.

[5] A. Bonifati, E.Q. Chang, and L.V. Lakshmanan. Heptox: Marrying xml and heterogeneity in your p2p databases. In *VLDB*, 2005. Demo.

[6] R. Boppana and M. M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. SWAT, 1990

[7] P. Buneman, W. Fan, and S. Weinstein. Query optimization for semistructured data using path constraints in a deterministic data model. *DBPL*, pages 208–223, 1999.

[8] K. Candan, J. Grant, and V. Subrahmanian. A unified treatment of null values using constraints. *Information Systems Journal*, 98(1-4):99–156, May 1997.

[9] J. Doyle. A truth maintenance system. *J.of Artificial Intel.*, 12: 231–272, 1979.

[10] S. Flesca, *et al.* Repairs and consistent answers for xml data with functional dependencies. *Xsym* pages 238–253, 2003.

[11] D. Florescu and D. Kossman. Storing and Querying XML Data using an RDBMS. *IEEE Data Eng. Bulletin*,22(3):27-34, 1999.

[12] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. *International Conference and Symposium on Logic Programming*. 1988.

[13] L. Giordano, A. Martelli and M.L. Sapino". Extending negation as failure by abduction: a 3-valued stable model semantics. *J. of Logic Programming*, 1996.

[14] A. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management. In *ICDE*, 2003.

[15] T. Imielinski and W. Lipski. On representing incomplete information in a relational data base. VLDB, 1981.

[16] T. Imielinski and W. Lipski. Incomplete information in relational databases. *JACM*, 31(4):761–791, 1984.

[17] Y. E. Ioannidis and T. K. Sellis. Conflict resolution of rules assigning values to virtual attributes. SIGMOD, 1989.

[18] D. S. Johnson and C. H. Papadimitriou. On generating all maximal independent sets. *Info. Proc. Letters*, 27, 1988.

[19] K. W. Kintigh. *et al.* Enabling the study of long-term human and social dynamics: A cyberinfrastructure for archaeology. http://cadi.asu.edu/HSDPosterSlidesLR.ppt.

[20] K. W. Kintigh *et al.* Workshop on cybertools for archaeological data integration. http://cadi.asu.edu/, December 2004.

[21] K. W. Kintigh. The promise and challenge of archaeological data integration. *American Antiquity*, 2006. in press.

[22] K.-C. Liu and R. Sunderraman. A generalized relational model for indefinite and maybe information. TKDE, 3(1), 1991.

[23] M. Liu and T. W. Ling. A data model for semistructured data with partial and inconsistent information. *LNCS*, 1777, 2000.

[24] J. McHugh, *et al.* Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[25] R. E. Mercer and V. Risch. Properties of maximal cliques of a pair-wise compatibility graph for three nonmonotonic reasoning system. In *Answer Set Programming*, 2003.

[26] J.W. Moon and L. Moser On cliques in graphs. Israel Journal of Mathematics, 3, 23–28, 1965.

[27] W. Ng. Repairing inconsistent merged xml data. In *DEXA*, pages 244–255, 2003.

[28] K. Spielmann, J. Driver, D. Grayson, E. Reitz, S. Kanza, and C. Szuter. Faunal working group. http://cadi.asu.edu.

[29] I. Tatarinov, *et al.* Storing and querying ordered XML using a relational database system. *SIGMOD*, pages 204–215, 2002.

[30] Y. Wu, J. M. Patel, and H. V. Jagadish. Structural join order selection for xml query optimization. In *ICDE*, 2003.

[31] C. Zaniolo. Design and implementation of a logic-based language for data-intensive applications. *ICLP* 1988.

[32] C. Zaniolo. *A United Semantics for Active and Deductive Databases*, chapter Rules in Database Systems. 1994.