Database Theory Column[1]

An Overview of Semistructured Data

Dan Suciu
AT&T Labs
suciu@research.att.com

1 Introduction

Research on semistructured data started from the observation that much of today's electronic data does not conform to traditional relational, or object oriented data models. Several applications store their data in non-standard data formats, legacy systems, structured documents, like HTML or SGML etc. Another instance is the integration of heterogeneous data sources: often these sources belong to external organizations, or partners, not under the application's control, and their structure is only partially known, and may change without notice. Data in these applications could be modeled as an object-oriented data, but it's structure is irregular: some objects may have missing attributes, others may have multiple occurrences of the same attribute, the same attribute may have different types in different objects, semantically related information may be represented differently in various objects. Data with these characteristics has been called *semistructured data*.

Recent research has aimed at extending database management techniques to semistructured data. The result of this work is a new paradigm in databases, complementing the relational and object-oriented one. The new model has been applied to a few research prototypes, like data integration [PGMW95, PAGM96], Web site management [FFK+98], general-purpose management of semistructured data [QRS+95, AQM+97, MAG+97], and data conversion [SCJK98]. Other research work has addressed query language design [AQM+97, BDHS96a, PAGM96, FFLS97a, FFLS97b] schema specification [BDFS97, MS99, DGM98, BM99], schema extraction [NUWC97, BDFS97, GW97, NAM97], optimizations [AV97b, Suc96, Suc97, FS98, MW97], indexing [MWA+98, MS99], as well as formal aspects [AV97a, AV97b, MM97, FLS98]. The interested reader may want to consult the tutorials by Abiteboul [Abi97] and Buneman [Bun97].

In the traditional *relational* paradigm a database is modeled as a finite, first order structure. The first order *vocabulary*, i.e. the names and arities of the relations of that structure models the database *schema* (the list of all table names and their attributes), and first order logic, or higher logic formulae model queries. Database theory is related to both finite model theory and descriptive complexity.

It makes no sense however to talk about finite structures or formulae without fixing the vocabulary first. In *semistructured* databases we do not have an a priory schema, hence no vocabulary. One way to model semistructured databases is to represent them as *graphs*. Thus, the vocabulary is fixed once and forever to be that of graphs. The real schema of a given database instance is now

[1]Column editor: Victor Vianu, CSE 0114, University of California San Diego, La Jolla, CA 92093-0114, vianu@cs.ucsd.edu.

moved into the graph itself, in the form of labels attached to the graph's nodes or edges. Thus, schema components, like table and attribute names, become constants in the first order structure. Formulae in the logic can now explicitly refer to the schema's components by means of constants in the formula. To retrieve the value of an attribute one traverses an edge in the graph labeled with the name of that attribute. For example to retrieve a person's `address`, we traverse an edge labeled `address` from the node corresponding to that person. To obtain a person's `zip` code we may have to follow a path of length two, labeled `address.zip`. New possibilities arise: one can query the schema's components (e.g. "find all attribute names"), or write queries which ignore part of the schema (e.g. "retrieve all attribute values, regardless of the attribute name"). More complex queries can traverse the data with only partial knowledge of the schema, by means of *regular path expressions* on the graph data. For example if we don't know whether `phone` is an attribute of the person, or of the `address` component of that person, we write the regular path expression `*.phone`, which follows any path whose last label is `phone`. The distinction between schema and the data is eliminated in semistructured databases.

Modeling semistructured data as a graph has been the preferred approach so far, and we will review it in this paper. The model however has its limitations, since it ignores some characteristics of semistructured data, like the fact that edges in the graph can be traversed only forwards, or that sometimes the order of the attributes matters.

The purpose of paper is to give an overview of the research on semistructured data and briefly illustrate some of its theoretical and formal aspects. We describe the semistructured data model in Sec. 2 and illustrate a few query languages in Sec. 3. We discuss schema formalisms for semistructured data in Sec. 4 and conclude in Sec. 5.

2 Data Model

OEM The most popular model of semistructured data is the OEM model (Object Exchange Model), originally introduced for the Tsimmis [PGMW95] data integration project. In OEM data is represented by a collection of *objects*. Each object can be atomic or complex. The value of an atomic object is of some base type (integer, string, image, sound, etc). The value of a complex object is a set of `(attribute, object)` pairs: here `attribute` is any string, usually drawn from a universe $\mathcal{A}$ of attribute names. Thus OEM data is a graph, where the nodes are the objects, the edges are labeled with attributes, and in which some leaf nodes have an associated atomic value. The graph also has a *root*, i.e. a distinguished object with all other objects are accessible from it[2]. Formally a semistructured data is $G = (V, E, r, v)$ where the set of nodes V is partitioned into complex and atomic nodes $V = V_c \cup V_a$, the edges are $E \subseteq V_c \times \mathcal{A} \times V$, $r \in V$ is the root, and $v : V_a \to \mathcal{D}$ assigns values to atomic objects; here $\mathcal{D}$ is the universe of *atomic values*.

Figure 1 illustrates an example of some OEM bibliography data. A textual representation of the same data is given in Fig. 2. Here `&o12, &o43`, etc. are object identifiers, *oids*: when the underlying graph is a tree, oid's can be omitted from the textual representation completely without any loss of information. For example the value of the object `&o24` can be written as:

```
{author: "Abiteboul",
 author: "Hull",
 author: "Vianu",
 title: "Foundations of Databases",
 publisher: "Addison Weseley"}
```

[2] OEM actually allows several named roots: we restrict to a single root to simplify the presentation.

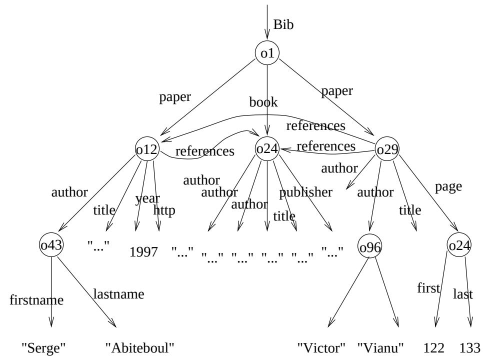

Figure 1: OEM Data

```
Bib:&o1
{paper: &o12
  {author: &o43
    {firstname: &23 "Serge",
     lastname: &76 "Abiteboul"},
   title: &o63 "Querying semi-structured data",
   year: &o42 1997,
   references: &o24,
   http: &o91
    "http://www-rocq.inria.fr/~abitebou/pub
                  /icdt97.semistructured.ps"},
 book: &o24
  {author: &o23 "Abiteboul",
   author: &o62 "Hull",
   author: &o82 "Vianu",
   title: &o97 "Foundations of Databases",
   publisher: &o22 "Addison Weseley"},
 paper: &o29
  {author: &o52 "Abiteboul",
   author: &o96 {firstname: &243 "Victor",
                 lastname: &o206 "Vianu"},
   title: &o93 "Regular path queries with constraints",
   references: &o12,
   references: &o24,
   page: &o25 {first: &o64 122, last: &o92 133}}}
```

Figure 2: Textual representation of OEM data

Name	Phone
John	3634
Sue	6343
Dick	6363

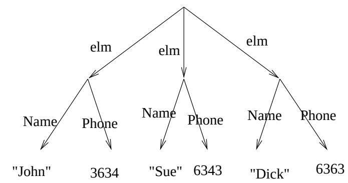

Figure 3: Relational data viewed as semistructured data

Comparison to the Relational Model A relation is a *set* of *records* [Ull89]. Both sets and records can be easily represented in OEM: the set {`5,2,9`} can be represented as, say, {`elm:5, elm:2, elm:9`}, while a record `[a=5, b="seven", c=3.14]` can be represented as {`a:5, b:"seven", c:3.14`}. Sets are OEM objects where all attributes are the same, and records are objects where all attributes are distinct. Relational data can be easily represented as a particular case of semistructured data, e.g. Fig. 3. The scope of semistructured data however is not to offer a graph view of relational data, but to enable us to deal precisely with data which is *not* relational.

Object Equality Like in object-oriented languages, there are two operators for testing object equality in OEM: *shallow equality* returns true iff the two are the same object, and *deep equality* returns true if the graphs accessible from the two objects are isomorphic. None of the two equalities would identify $\{\mathtt{a}:\{\mathtt{b}:\mathtt{1},\mathtt{c}:\mathtt{2}\},\mathtt{a}:\{\mathtt{b}:\mathtt{1},\mathtt{c}:\mathtt{2}\}\}$ with $\{\mathtt{a}:\{\mathtt{b}:\mathtt{1},\mathtt{c}:\mathtt{2},\}\}$, because of the distinct object identifiers omitted from the textual representation. This is a departure from the relational model: like in object-oriented systems, objects in OEM have an existence of their own.

The UnQL Variation on the Data Model A variant of the semistructured data model is in UnQL [BDS95, BDHS96a], in which $\{\mathtt{a}:\{\mathtt{b}:\mathtt{1},\mathtt{c}:\mathtt{2}\},\mathtt{a}:\{\mathtt{b}:\mathtt{1},\mathtt{c}:\mathtt{2}\}\}$ and $\{\mathtt{a}:\{\mathtt{b}:\mathtt{1},\mathtt{c}:\mathtt{2}\}\}$ *are* equal. Data is still a labeled graph, but the model explicitly states that two graphs G, G' are equal if they are *bisimilar*. The notion of bisimulation has bee used in non-well founded set theory [Acz88] and in process algebra [Mil89], and intuitively equates two graphs when they become equal after each graph is transformed by (1) unfolding from the root, and (2) eliminating duplicates at each node. We refer the reader to [BDS95, BDFS97] for the formal definition of bisimulation in the context of semistructured data. Paige and Tarjan have shown that a bisimulation between two unlabeled graphs can be computed in time $O(m \log n)$ [PT87] where n is the total number of nodes and m the total number of edges. Since semistructured data are multigraphs (we may have multiple edges between the same two nodes, as long as they are labeled with distinct attributes), that algorithm takes $O(m \log m)$ time on semistructured data [BDFS97].

3 Query Languages

Lorel LORE [QRS+95, MAG+97, AQM+97] (Lightweight Object REpository) is a general purpose data management system for semistructured data, and Lorel is its query language. We illustrate Lorel in a few examples. The following query returns the titles of papers written since 1995:

```
select X.title
from Bib.paper X
where X.year > 1995
```

The meaning of the query is as follows: (1) find all paths in the database which start at the root and are labeled `Bib.paper`, and bind `X` successively to each end node (referring to Fig. 1, `X` will be bound successively to `&o12, &o24, &o29`). (2) retain only those bindings where `X` has some `year` attribute whose value is `> 1995` (in the example: `&o12`), (3) return all `titles` of all these `X`'s (in the example: `"Querying semi-structured data"`).

A more complicated example is with *regular path expressions*. For example consider the query "find all papers referenced directly or indirectly by `Ullman`":

```
select X.title
from Bib.paper X, Bib.paper Y
where Y.author.(lastname)? = "Ullman",
      Y.(reference)+ X
```

Here ( )? denotes an optional path expression, and ( )+ denotes the strict Kleene closure. Other regular expressions are Kleene closure ( )∗, concatenation ( ).( ), and alternation ( ) | ( ). Regular expressions are important in the context of semistructured data because they allow users to write queries which traverse data whose structure is not fully known. For example in the query above `"Ullman"` can be either the value of `Y.author` or of `Y.author.lastname`.

UnQL The language UnQL (Unstructured Query Language) [BDS95, BDHS96a] was derived from *structural recursion*, a construct introduced in [BBW92, BLS+94, BNTW95] for programming with sets, bags, and lists. For semistructured data structural recursion is particularly attractive because it can express both *queries* and *transformations* in the same formalisms. Here *queries* just return a subset of nodes from the input data, while *transformations* may a construct a new graph. Lorel is mostly suited for expressing queries (it has special constructs for transformations, but their expressive power is limited). Structural recursion expresses both: we illustrate this next.

The first example is a query retrieving all integers in the database:

```
(1) f(v)        = if isInt(v) then {result : v}
                  else {}
(2) f({})       = {}
(3) f({l : t}) = f(t)
(4) f(t1 U t2) = f(t1) U f(t2)
```

This defines a recursive function `f` by pattern matching, with four rules. We illustrate its computation on the database `D` in Fig. 2. Here `D = {Bib : &o1}` hence rule `(3)` applies, and `f(D) = f(&o1)`. Next `&o1 = {paper:&o12, book:&o24, paper:&o29}`, which is a union `&o1 = {paper:&o12} U {book:&o24} U {paper:&o29}`: hence rule `(4)` applies twice, then rule (3) applies three times, and we get `f(&o12) U f(&o24) U f(&o29)`. Eventually we reach the leaves, for which rule `(1)` applies: the end result is `{result : 122, result : 133, result : 1997}`.

The second example illustrates a transformation. It takes some bibliography database and makes a copy of it, converting all integers to strings in `books`, and leaving integers untouched in other publications:

```
g(v)        = v
g({})       = {}
g({l : t}) = if l = book then {book : h(t)}
             else {l : g(t)}
g(t1 U t2) = g(t1) U g(t2)
```

```
h(v)         = if isInt(v) then Int2String(v)
               else v
h({})        = {}
h({l : t}) = {l : h(t)}
h(t1 U t2) = h(t1) U h(t2)
```

Here we have two mutually recursive functions: g only looks for a book label and returns everything unchanged, while h converts Ints to Strings and leaves everything else unchanged.

The general form of structural recursion consists of a definition of $m \geq 1$ mutually recursive functions, with certain restrictions. Each function f is defined by pattern matching, with four rules (as above) and must return {} on {} and f(t1) U f(t2) on t1 U t2 (rules 2 and 4). In the rule f({l:t}) it may call recursively f(t), and the other functions on t, and use their values to construct the result. We refer the reader to the references for the details.

Note that it is not immediately clear whether structural recursion has any meaning on graphs with cycles. It turns out that it has, and that its meaning admits two independent, equivalent definitions. An operational one, where recursive calls are *memoizes* to avoid infinite loops, and a declarative one, where the rule for f({l : t}) is fired in parallel, on all edges in the graph.

Formal properties of UnQL have been investigated in more detail in [BDHS96b]. Its data complexity is NLOGSPACE (due to the regular path expression), yet on relational data like that in Fig. 3 it expresses exactly the class of first order logic queries. As a consequence UnQL cannot express the transitive closure of a binary relation. Also, UnQL queries are invariant under bisimulations: if the data graphs D and D' are bisimilar, then so are $Q(D)$ and $Q(D')$ for every query Q.

Expressing Transformations with Skolem Functions A different approach to express transformations of semistructured data is with Skolem Functions. These have been introduced in object-oriented systems by Maier [Mai86], and have been extensively studied in the context of deductive query languages by Hull and Yoshikawa in [HY90]. For transforming semistructured data, they have been first used by Papakonstantinou et al. [PAGM96] in the Mediator Specification Language MSL, and later in StruQL [FFLS97a, FFLS97b], the query language of the Web site management system Strudel system [FFK+98], and also in YAT, a data conversion system [SCJK98]. In all three applications *transforming* semistructured data is a central task, and Skolem functions proved a useful technique to express such transformations. We illustrate Skolem functions next in the context of Strudel.

Assume we have a bibliography data like in Fig. 1 and would like to construct a Web site organized as follows. (a) There is a root page, (b) it has links to persons' pages (c) from each person's page there are entries for each year when that person has published, and (d) each year entry has links to the actual publications. This can be achieved with the following StruQL query:

```
where  Root -> "Bib".L -> X,  X -> "year" -> Y
       X -> "author".("lastname")? -> Z, isString(Z),
create RootPage(), PersonPage(Z), YearEntryPage(Z,Y)
link RootPage() -> "Person" -> PersonPage(Z),
     PersonPage(Z) -> "YearEntry" -> YearEntryPage(Z,Y),
     YearEntryPage(Z,Y) -> L -> X
```

The where clause is quite similar to Lorel's, with minor variation in syntax. For example the condition X -> "year" -> Y looks for an edge labeled "year" from X to Y. The variable L in Root -> "Bib".L -> X is a *label variable*, and is bound to attribute constants like "book" and "paper" (see Fig. 1). The new nodes and edges are constructed in the `create` and `link` clauses. Here `RootPage, PersonPage, YearEntryPage` are *Skolem Functions* that create new oid's. By definition a Skolem function applied to the same inputs produces the same node oid: for example if `X` is bound several times to "Abiteboul", the result of `PersonPage("Abiteboul")` will be the same, new oid. Note that `YearEntryPage(Z,Y)` needs to have two arguments since we need a new node for each author and for each year.

Formal Properties of Query Languages All query languages for semistructured data discussed here share the ability to traverse the data graph, hence to compute transitive closure. Their exact expressive power differs, however. UnQL cannot compute *any* transitive closure, but only that of the input graph. StruQL has been shown in [FFLS97b] to have the same expressive power as FO(TC) (first order logic extended with a *transitive closure* construct [Imm87]). For example, StruQL can express transitive closure of binary relations expressed as trees (which UnQL cannot). For that one composes two `where-create-link` blocks, like in the following example in which the input graph `G` is a tree encoding of a binary relation with attributes `A` and `B` (as illustrated in Fig. 3), and which tests whether the pair `("abc", "cde")` is in its transitive closure:

```
input (
 input G
 where Root -> "elm" -> X, X -> "A" -> Y, X -> "B" -> Z
 create Start(), Stop(), F(Y), F(Z)
 link Start() -> "C" -> F("abc"), F(X) -> "C" -> F(Y), F("cde") -> "C" -> Stop())
where Start() -> ("C")* -> Stop()
```

The first block constructs an intermediate data graph based which is a instantiation of the graph represented by the binary relation (all its edges are labeled `"C"`), while the outer block just searches for a path in this new graph. In Lorel one cannot construct the intermediate graph, hence Lorel cannot express transitive closure of binary relations.

Another line of research has addressed query containment and equivalence in the presence of regular expressions. Abiteboul and Vianu [AV97b] consider single path queries (i.e. given by a single regular path expression) in the presence of *path equations*. Query equivalence without equations is equivalence of regular expression, which is known to be PSPACE-complete. A path equation is illustrated by `part.(subpart)*.description = catalog.part.description`, which says says that all `descriptions` of all `subparts` can be reached directly through the catalog. Then the query `part.(subpart)*.description.section.paragraph` can be replaced with the simpler `catalog.part.description.section.paragraph`. In general, we have a set of path equations E and would like to decide whether two given path expressions $p1, p2$ are equivalent, over databases satisfying E. The authors have shown the surprising result that this problem is decidable, and moreover that its complexity is tractable under certain natural restrictions.

Queries in which regular expressions are mixed with joins are considered in [FLS98]. For *conjunctive* queries, query containment and equivalence are decidable by an algorithm which takes exponential space. When the regular expressions consist only of constant labels and $*$ (as a wildcard, meaning any path), then containment and equivalence are in NP.

4 Schemas

Semistructured data is self-describing. That is, the schema is embedded with the data, and no a priori structure is assumed. This gives us flexibility to process (store, query, etc) any data, and we can deal with changes in the data's structure seamlessly. But this extreme view has a few drawbacks:

- Data is inefficient to store, since the schema needs to be replicated with each data item.
- Queries are hard to evaluate efficiently: even a simple regular path expression may require the entire graph to be traversed.
- Queries are hard to formulate: the user has to rely on undocumented information about the data in order to formulate relevant queries.

In the applications of semistructured data one observes that the data often has some regular structure which can, and should be exploited to address the problems above. Researchers have looked for ways to describe and exploit regularities in the data's structure, without having to fully specify a rigid schema ahead of time. Proposals fall into two categories: (1) Schema formalisms described independently on the data. These formalisms offer a continuous scale of precision in describing the data's structure. In all these formalisms there exists a "universal" schema that does not impose any structure on the data, and schemas are partially ordered according to how much structure they impose on the data. Examples are graph schemas [BDFS97], description logics [DGM98], Schema Definition Language [BM99]. (2) Inferred schemas, which are inferred automatically from the data. These are rigid, and describe the data's structure accurately. They have to be recomputed, or incrementally updated, every time the data changes. Examples are data guides [NUWC97, GW97], unary datalog types [NAM97], T-indexes [MS99].

In contrast to relational data, semistructured data can exist independently of the schema. This generates a new kind of problem: given a database `G` and schema `S`, check whether the data `G` conforms to the schema `S`. The complexity of this problem ranges from PTIME for the case of graph schemas [BDFS97], to NP-complete for the case of SDL's [BM99]. One question is whether schemas are closed under set operations: e.g. given two schemas `S1`, `S2`, does there exists a schema `S` describing precisely the databases conforming to both `S1` and `S2` (intersection) ? For the case of inferred schemas, the central problem is the complexity of the inference algorithm, and the size of the inferred schema, in terms of the data size: for dataguides this is exponential, for T-indexes it is polynomial. Another problem is to simplify the inferred schema, assuming that one can tolerate deviations of the data from the schema.

The interplay between schemas and queries is still being studied. One possibility is to exploit the schema during query evaluation. For example considering the Lorel query `select X where *.subpart.*.description X` (where `*` means any sequence of labels), we normally need to traverse the entire graph data to find all bindings of `X`. This is inefficient. A schema may describe in which parts of the data one should expect a `description` attribute to follow a `subpart` one, and this can be exploited during query evaluation: this idea has been pursued in [FS98]. Another problem is type checking for queries defining data transformations. Given an input schema `S`, output schema `S'`, and query `Q`, check whether `Q(G)` conforms to `S'` for every database `G` conforming to `S`.

5 Conclusions

The semistructured data model represents a new paradigm in databases. Unlike the relational and object-oriented paradigms, in semistructured data the schema is embedded in the data, making it easier to exchange across applications, business units, or organizations. An independent but strongly relevant development is now happening in the WWW community. Recently, the World Wide Web Consortium has approved the standard for the Extensible Markup Language, XML [Con98]. A central XML application is Electronic Data Interchange, in which XML is used as a data exchange format. The data expressed in XML is strikingly similar to semistructured data: it consists of objects and attributes (called *elements* and *tags* respectively), and, like semistructured data, has the schema embedded in the data. Today, most data exchanged on the Web consists of HTML documents, and is sent from applications to humans. In the near future, many believe that even larger amounts of data will be exchanged in XML format, and will be sent from applications to applications. This will generate a whole new class of data management problems, some of which may benefit from the research done on semistructured data.

References

[Abi97] Serge Abiteboul. Querying semi-structured data. In *Proceedings of the International Conference on Database Theory*, pages 1–18, Deplhi, Greece, 1997. Springer-Verlag.

[Acz88] P. Aczel. *Non-Well-Founded Sets.* Number 14 in CSLI Lecture Notes. Stanford Univ Center for the Study, Stanford, 1988.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, April 1997.

[AV97a] Serge Abiteboul and Victor Vianu. Queries and computation on the web. In *Proceedings of the International Conference on Database Theory*, pages 262–275, Deplhi, Greece, 1997. Springer-Verlag.

[AV97b] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 122–133, 1997.

[BBW92] Val Breazu-Tannen, Peter Buneman, and Limsoon Wong. Naturally embedded query languages. In J. Biskup and R. Hull, editors, *LNCS 646: Proceedings of 4th International Conference on Database Theory, Berlin, Germany, October, 1992*, pages 140–154. Springer-Verlag, October 1992.

[BDFS97] Peter Buneman, Susan Davidson, Mary Fernandez, and Dan Suciu. Adding structure to unstructured data. In *Proceedings of the International Conference on Database Theory*, pages 336–350, Deplhi, Greece, 1997. Springer Verlag.

[BDHS96a] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 505–516, 1996.

[BDHS96b] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. Technical Report 96-09, University of Pennsylvania, Computer and Information Science Department, 1996.

[BDS95] Peter Buneman, Susan Davidson, and Dan Suciu. Programming constructs for unstructured data. In *Proceedings of the Workshop on Database Programming Languages*, Gubbio, Italy, September 1995.

[BLS+94] P. Buneman, L. Libkin, D. Suciu, V. Tannen, and L. Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, March 1994.

[BM99] Catriel Beeri and Tova Milo. Schemas for integration and translation of structured and semi-structured data. In *Proceedings of the International Conference on Database Theory*, 1999. to appear.

[BNTW95] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with collection types. *Theoretical Computer Science*, 149:3–48, 1995.

[Bun97] Peter Buneman. Tutorial: Semistructured data. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 117–121, 1997.

[Con98] World Wide Web Consortium. Extensible markup language (xml) 1.0, 1998. `http://www.w3.org/TR/REC-xml`.

[DGM98] D.Calvanese, G.Giacomo, and M.Lenzerini. What can knowledge representation do for semi-structured data ? In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[FFK+98] Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. Catching the boat with Strudel: experience with a web-site management system. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 1998.

[FFLS97a] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language and processor for a web-site management system. In *Proceedings of the Workshop on Management of Semi-structured Data*, 1997. Available from `http://www.research.att.com/~suciu/workshop-papers.html`.

[FFLS97b] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, September 1997.

[FLS98] Daniela Florescu, Alon Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 139–148, 1998.

[FS98] Mary Fernandez and Dan Suciu. Optimizing regular path expressions using graph schemas. In *Proceedings of the International Conference on Data Engineering*, pages 14–23, 1998.

[GW97] Roy Goldman and Jennifer Widom. DataGuides: enabling query formulation and optimization in semistructured databases. In *Proceedings of Very Large Data Bases*, pages 436–445, September 1997.

[HY90] R. Hull and M. Yoshikawa. ILOG: Declarative creation and manipulation of object identifiers. In *Proceedings of 16th International Conference on Very Large Data Bases*, pages 455–468, 1990.

[Imm87] Neil Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[Mai86] D. Maier. A logic for objects. In *Proceedings of Workshop on Deductive Database and Logic Programming*, Washington, D.C., August 1986.

[Mil89] Robin Milner. *Communication and concurrency*. Prentice Hall, 1989.

[MM97] A. Mendelzon and T. Milo. Formal models of web queries. In *Proceedings of ACM Symposium on Principles of Database Systems*, pages 134–143, 1997.

[MS99] Tova Milo and Dan Suciu. Index structures for path expressions. In *Proceedings of the International Conference on Database Theory*, 1999. to appear.

[MW97] Jason McHugh and Jennifer Widom. Query optimization for semistructured data. Technical report, Stanford University, 1997.

[MWA+98] J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajaraman. Indexing semistructured data. Technical report, Stanford University, 1998.

[NAM97] S. Nestorov, S. Abiteboul, and R. Motwani. Inferring structure in semistructured data. In *Proceedings of the Workshop on Management of Semi-structured Data*, 1997. Available from `http://www.research.att.com/~suciu/workshop-papers.html`.

[NUWC97] S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative objects: concise representation of semistructured, hierarchical data. In *International Conference on Data Engineering*, pages 79–90, 1997.

[PAGM96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of Very Large Data Bases*, pages 413–424, September 1996.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *IEEE International Conference on Data Engineering*, pages 251–260, March 1995.

[PT87] Robert Paige and Robert Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16:973–988, 1987.

[QRS+95] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructure heterogeneous information. In *International Conference on Deductive and Object Oriented Databases*, pages 319–344, 1995.

[SCJK98] S.Cluet, C.Delobel, J.Simeon, and K.Smaga. Your mediators need data conversion ! In *Proceedings ACM-SIGMOD International Conference on Management of Data*, pages 177–188, 1998.

[Suc96] Dan Suciu. Query decomposition and view maintenance for query languages for unstructured data. In *Proceedings of the International Conference on Very Large Data Bases*, pages 227–238, September 1996.

[Suc97] Dan Suciu. Distributed query evaluation on semistructured data, 1997. Available from `http://www.research.att.com/~suciu`.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledgebase Systems I*. Computer Science Press, Rockville, MD 20850, 1989.