Data Integration
. . . and Semistructured Data

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini

Dipartimento di Informatica e Sistemistica

Università di Roma "La Sapienza"

Dottorato in Ingegneria Informatica
Dicembre, 2000 — Roma

Why semistructured data for data integration?

- Data integration is one of the major issues in dealing with data on the Web
- The Web today: HTML documents
  - many (already most?) generated automatically
  - all intended for human consumption
  - applications cannot consume HTML – wrappers are brittle
- The new Web: structured documents and data
  - generated and consumed by applications
  - data exchange across platforms and enterprises

Database community can help

- query optimization and query processing
- view based query processing (rewriting, answering)
- mediators
- efficient access, secondary storage, indexes

Needs a paradigm shift

Web data differs from database data

- self-describing, schema-less
- dynamic (structure changes without notice)
- heterogeneous, deeply nested, irregular
- documents and data mixed together
- limitations in access
- distributed, linked

Need to extend management of structured data to management of semistructured data

Outline

1. Modeling and querying semistructured data
   - representing semistructured data
   - query languages for semistructured data
   - schemas for semistructured data
2. Data integration and semistructured data
   - ssd in data integration systems
   - view based query processing for semistructured data

Part 1 — Modeling and querying ssd

$\Rightarrow$ Representing semistructured data
  - Object Exchange Model (OEM)
  - XML
  - semistructured data vs. XML

- Query languages for semistructured data
- Schemas for semistructured data

Representing semistructured data

One de facto standard: Object Exchange Model (OEM)

- introduced in the TSIMMIS mediator system, and adopted in other systems (e.g., LORE)
- provides a format for exchanging objects between mediators
- no notion of schema
- objects are not “typed” in the traditional sense

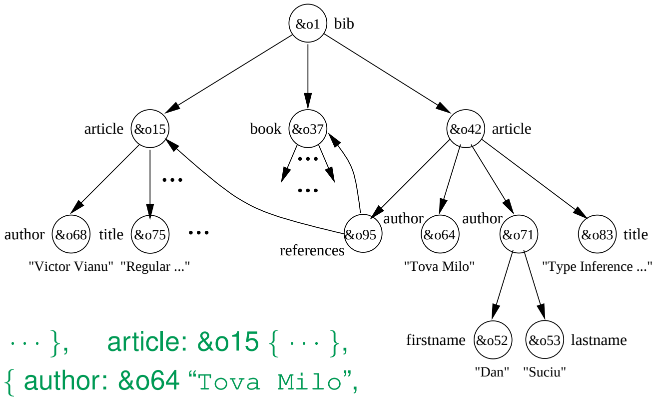
Object Exchange Model

Original version:

- an OEM object is a triple $(label : oid\ value)$, where $value$ is either
  - an atomic value (integer, string, etc.)
  - a set of OEM objects
- a database is a set of OEM objects

A database is a node-labeled graph (labels $\sim$ classes)

Object Exchange Model — Example



bib:&o1 { book: &o37 { $\cdots$ }, article: &o15 { $\cdots$ },
article: &o42 { author: &o64 "Tova Milo",
author: &o71 { firstname: &o52 "Dan",
lastname: &o53 "Suciu" },
title: &o83 "Type Inference for Queries ...",
references: &o95 { &o15, &o37 } } }

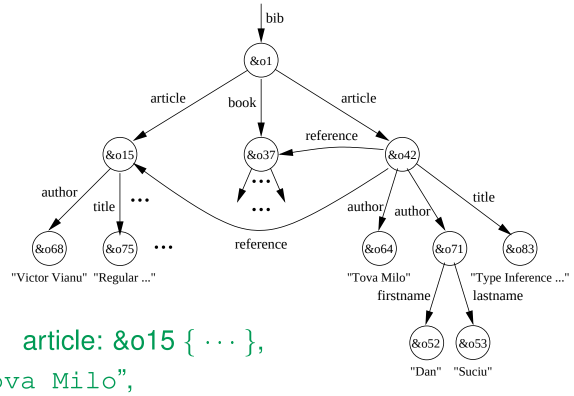
Object Exchange Model — Recent variant

Recent systems use a more convenient variant of OEM:

- an OEM object is a pair $(oid\ \ value)$, with $value$ either
  - an atomic value (integer, string, etc.)
  - a set of pairs $label : OEM\text{-}object$
- a database is a set of pairs $label : OEM\text{-}object$

A database is an edge-labeled graph (labels $\sim$ attributes)

Object Exchange Model (edge labels) — Example



bib:&o1 { book: &o37 { $\cdots$ }, article: &o15 { $\cdots$ },
article: { author: "Tova Milo",
author: &o71 { firstname: "Dan",
lastname: "Suciu" },
title: "Type Inference for Queries ...",
reference: &o15, reference: &o37 } }

Characteristics of semistructured data

Self-describing irregular data with no a-priori structure

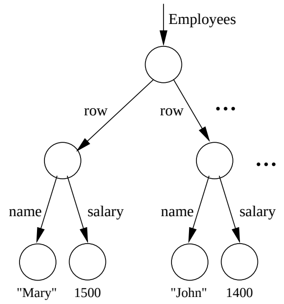
- missing or additional attributes
- multiple attributes
- attributes with different types in different objects
- heterogeneous collections

Semistructured view of relational data

Schema elements become labels in the semistructured database

Employees

name	salary
Mary	1500
John	1400
⋮	⋮



{ Employees: { row: { name: “Mary”, salary: “1500” },
row: { name: “John”, salary: “1400” },
⋮ },
⋮ }

eXtensible Markup Language (XML)

New standard approved by W3C (1998) [http://www.w3.org/]

- format for representing structured documents
- a simplified (cleaner) form of SGML
- designed with HTML in mind (simplicity of use on the Web)
- separates presentation from content (initial design goal)
- emerging as a standard for data exchange on the Web and between applications (tremendous success)

XML vs. HTML

```
<h1>Bibliography</h1>
<ul>
  <li><i>Principles of Database Systems</i>.<br>
      <b>Jeffrey D. Ullman</b>.
      Computer Science Press 1982.
  </li>
  <li><i>Type Inference for Queries over ...</i>.<br>
      <b>Tova Milo</b> and <b>Dan Suciu</b>.
      Proc. of PODS'99, 1999.
  </li>
</ul>
```

HTML describes the presentation

XML vs. HTML

```
<bibliography>
  <book><title>Principles of Database Systems</title>
        <author>Jeffrey D. Ullman</author>
        <publisher>Computer Science Press</publisher>
        <year>1982</year>
  </book>
  <article><title>Type Inference for Queries ...</title>
           <author>Tova Milo</author>
           <author>Dan Suciu</author>
           <booktitle>Proc. of PODS'99</booktitle>
           <year>1999</year>
  </article>
</bibliography>
```

XML describes the content

Basic XML terminology

- element types: bibliography, book, title, ...
- start tag: `<book>`
- end tag: `</book>`
- elements: `<book> ...</book>`, or `<title> ...</title>`
  can be arbitrarily nested
- empty elements: `<red></red>`, or `<red/>`
- XML document consists of a single root element
- XML document is well-formed if it has correctly matching tags

XML attributes

Attributes are an alternative way of representing data:
attribute-value pairs inside start tags

```
<book price="60EU" available="yes">
  <title>Principles of Database Systems</title>
  <author>Jeffrey D. Ullman</author>
  <publisher>Computer Science Press</publisher>
  <year>1982</year>
</book>
```

Note: attributes are unordered (while subelements are ordered)

XML oid's and references

Special attributes are used to identify and reference elements

```
<book id="Ullm-82">
  <title>Principles of Database Systems</title>
  ...
</book>
<article id="MiSu-99" references="Ullm-82 AbVi-97 ...">
  <title>Type Inference for Queries on ...</title>
  ...
</article>
<article id="AbVi-97" references="Abit-97 ...">
  ...
</article>
```

Oid's and references in XML are just syntax

XML data model

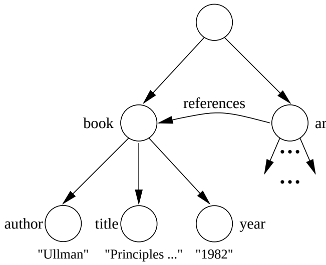
- until recently not available
- closest approximation: Document Object Model (DOM)
  - provides a class hierarchy (`node`, `element`, ...)
  - document represented as a tree
  - is an object model (and not only a data model)
    - objects have behaviour
    - defines API to manipulate the document
- XML Query Data Model is a working draft of the W3C

XML Query Data Model

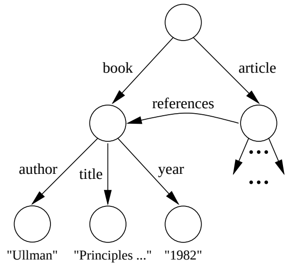
- document is viewed as a labeled tree
- eight types of nodes: document, element, value, attribute, . . .
- successors of a node may be:
  - an (ordered) sequence of nodes (e.g., for subelements)
  - an (unordered) set of nodes (e.g., for attibutes)
- compatible with XML-schema (see later)

XML vs. semistructured data

```
<book id="Ullm-82">
  <author>Ullman</author>
  <title>Principles ...</title>
  <year>1982</year>
</book>
<article references="Ullm-82 ...">
  ...
</article>
```



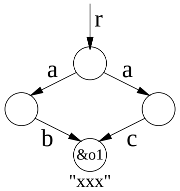

```
{ book: &o37 { author: "Ullman",
               title: "Principles ...",
               year: 1982 }
  article: { references: &o37, ... }
  ...
}
```



Mutual transformation is easy on trees

XML vs. semistructured data (cont.)

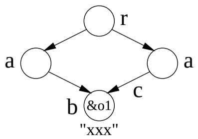
Analogy is less clean with references



can be encoded in XML in 2 ways

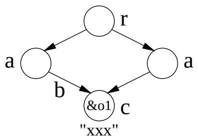
1. using c as a reference attribute

```
<r> <a> <b id="&o1"> xxx </b> </a>
    <a c="&o1"/> </r>
```



2. using b as a reference attribute

```
<r> <a b="&o1"/>
    <a> <c is="&o1"> xxx </c> </a> </r>
```



Comparison of XML and semistructured data

Similarities:

- both described best by a labeled graph
- both are schema-less, self-describing

Differences:

- XML is ordered, semistructured data is unordered
- XML can mix text and elements

```
<section><title>Introduction</title>
  In this section we deal with ...
</section>
```

- XML has lots of other features . . .

Other aspects of XML

- entities: basically macros
- external entities: allow for modularization
- CDATA sections: allow for inclusion of arbitrary character sequences in a document
- processing instructions: application specific
- comments

Interesting from the “document perspective”, not so much from the “data perspective”

Other standards related to XML

- navigation on and between documents:
  - XPath: basic mechanism for navigation
  - XLink: intra-document navigation
  - XPointer: inter-document navigation
- document presentation: XSL and XSLT (see later)
- schema specification: (see later)
  - XML Document Type Definitions (DTD) (is part of XML)
  - XML-Schema: extends DTDs with types, inheritance, constraints, . . .
  - RDF, OIL, . . .

Part 1 — Modeling and querying ssd

√ Representing semistructured data

⇒ Query languages for semistructured data
  - Lorel
  - UnQL
  - XML-QL
  - XSLT and structural recursion
  - XML Query Algebra

- Schemas for semistructured data

Required features of query languages for ssd

- expressive power
  - on relational data at least as expressive as SQL
  - ability to restructure data
  - need to navigate data with arbitrary nesting
    $\Longrightarrow$ regular path queries
- compositionality: a query must produce ssd (restructuring may require object invention)
- schemas: when structure is defined, it should be exploited for optimization, type checking, . . .

Query languages

- for semistructured data
  - Lorel (coercion, regular path expressions)
  - UnQL (patterns, constructors)
  - StruQL, MSL, W3QL, WebSQL, WebLog, . . .
- for XML
  - XML-QL
  - XSLT (structural recursion)
  - XML query algebra

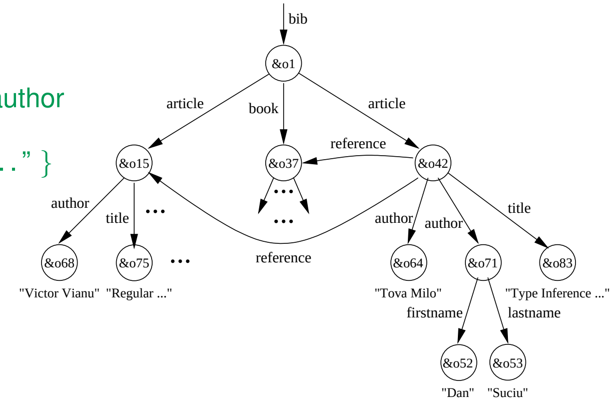
Lorel

Query language of the LORE system (Stanford)

Adapts OQL (query language of ODMG) to semistructured data

select X.title
from bib.article X
where "Tova Milo" in X.author

returns { title: "Type Inf..." }



Lorel — Semantics of queries

Evaluation of "select E from B where C" defined in three steps:

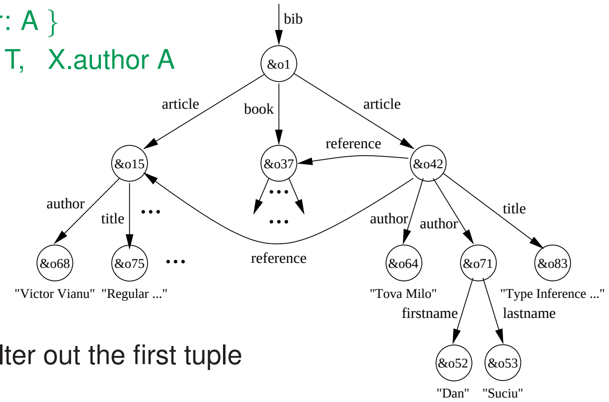
1. from B specifies a set of bindings of the vars $\vec{\mathbf{x}}$ in B.
   Each binding maps vars to oid's in the data graph
2. where C filters the bindings, obtaining a set $\{\vec{\mathbf{o}}_1, \ldots, \vec{\mathbf{o}}_k\}$
3. select E constructs an ssd expression $\{E(\vec{\mathbf{o}}_1), \ldots, E(\vec{\mathbf{o}}_k)\}$
   by substituting the vars of $\vec{\mathbf{x}}$ in E with each $\vec{\mathbf{o}}_i$

$\{E(\vec{\mathbf{o}}_1), \ldots, E(\vec{\mathbf{o}}_k)\}$ forms the result of the query

Lorel — Example of query

select row: { title: T, author: A }
from bib.article X, X.title T, X.author A
where X.year > 1998

from bounds (X,T,A) to
{(&o15, &o75, &o68),
(&o42, &o83, &o64),
(&o42, &o83, &o71) }



let X.year>1998 in where filter out the first tuple

applying select we obtain

{ row: { title: "Type I...", author: "Tova Milo" },
row: { title: "Type I...", author: { fn: "Dan", ln: "Suciu" } } }

Features of Lorel

- differences with typed query languages
  - performs implicit coercions: e.g., 1998 to "1998"
  - deals with missing attributes (empty answer instead of type error)
  - set-valued attributes:
    e.g., in X.year>1998, X may have several years
- select clause creates new nodes
- allows for nested queries
- allows for regular path expressions

Regular path expressions

Paths can be specified by means of regular expressions

```
select  A.title
from    bib.article A,   bib.(article|book) P
where   P.author.lastname? = "Ullman",   A.reference+ P
```

Useful for:

- syntactic substitute for inheritance: article|book
- navigating partially known structures: lastname?
- navigating arbitrarily nested structures: reference+

Note: $X.p.\ell$, with p a path expression, stands for the nested query

select $\ell : Y$
from $X.p.\ell\ Y$

Unstructured Query Language (UnQL)

Can be seen as an extension of basic Lorel

- where clause contains two kind of constructs
  - generators: introduce and bind variables via patterns
  - conditions: as in Lorel
- from clause is not needed (variables bound in patterns)
- does not make use of coercion

UnQL queries

```
select  title: T
where   {bib: {article: {title: T, year: Y }}} in db,
        Y > 1998
```

- root of the database is explicitly represented: db
- meaning of in is extended to perform pattern matching

Note: pattern matching is just syntactic sugar:
UnQL queries can be rewritten in Lorel

```
select  title: T
from    bib.article A,   A.title T,   A.year Y
where   Y > 1998
```

Additional features of Lorel and UnQL

- label variables
  - combining "schema" and "data" information

```
select new-person: (select L:Y  from X.L.Y
                              where not (L="salary"))
from   db.person X
```

  - turn labels into data and vice-versa

```
select publication: { type: L, title: T }
from   bib.L.X,  X.title T
where  X.year > 1998
```

  - perform group-by operations

- matching of variables with regular expressions

```
matches(".*(s|S)emistructured.*", T)
```


A query language for XML: XML-QL

- combines the features of the query languages for ssd
  - variables for binding data
  - patterns to extract data
  - templates to specify how to construct the output data
  - regular path expressions
  - skolem functions to create new oid's
- uses XML syntax: where . . . construct . . .
  (analogous to "select . . . where . . . " of UnQL)
- relies on an edge-labeled graph model (model mismatch)

XML-QL — Pattern matching

Patterns used in the where clause to bind variables

```
where <article language="English">
        <title> $T </title>
        <year> 1998 </year>
      </article> in "www.x.y/bib.xml"
construct $T
```

A bound variable can later be used to the right of "in"

Note: patterns allow for querying attributes

XML-QL — Constructors

Templates in the construct clause allow for constructing new data

```
where <book language=$L>
        <author> $A </>
      </> in "www.x.y/bib.xml"
construct <result>
            <author> $A </>
            <lang> $L </>
          </>
```

Groups titles by authors:

```
<result><author>Ullman</author><lang>English</lang></result>
<result><author>Ullman</author><lang>French</lang></result>
```

XML-QL — Order

Two different semantics for ordered and unordered data

For ordered data:

- when matching, patterns in where clause interpreted as unordered

```
pattern  <a> <b>$X</b> <c>$Y</c> </a>
matches  <a> <b>11</b> <c>22</c> </a>
    and  <a> <c>22</c> <b>11</b> </a>
```

- when constructing result, variable bindings in where clause are ordered

XML-QL — Order (cont.)

For the pattern `<a> <b>$X</b> <c>$Y</c> </a>`

on data

```
<a>
    <b> 1 </b>
    <c> 2 </c>
    <b> 3 </b>
    <c> 4 </c>
</a>
```

the binding order is

$X	$Y
1	2
1	4
3	2
3	4

For the purpose of order, it matters how we write the pattern:
`<a> <c>$Y</c> <b>$X</b> </a>` sorts first by $Y, then by $X

Can also use optional order-by clause

XML-QL — Nested queries

- for processing optional elements

```
where <book> $B </> in "www.x.y/bib.xml",
      <title> $T </> in $B,  <price> $P </> in $B
construct <result> <bt> $T </> <bp> $P </> </>
```

books without price are not included in the output

```
where <book> $B </> in "www.x.y/bib.xml"
      <title> $T </> in $B
construct <result> <booktitle> $T </>
                   where <price> $P </> in $B
                   construct <bookprice> $P </>
          </>
```

- for grouping the data

XML-QL — Regular path expressions

```
where <section*>
        <title> $T </>
        <note> optional </>
      </> in "www.x.y/bib.xml"
construct <optional> $T </>
```

Can use $ to match any tag

Note: regular expressions traverse the XML tree vertically

XML-QL — Skolem functions

Used to create new oid's that need to be referenced elsewhere

```
{ where <person>  <name> $N </>  <ssn> $S </> </>
        in  "www.x.y/employees.xml"
  construct <result id=SSNID($S)> <name> $N </> </>
}
{ where <employee>  <ssn> $S </>  <income> $I </> </>
        in  "www.x.y/salaries.xml"
  construct <result id=SSNID($S)> <income> $I </> </>
}
```

- each result produced in first subblock has a unique oid generated by Skolem function SSNID(·)
- incomes of second subblock appended to objects with same oid

Note: performs an outer join

XML-QL — Summary

- features derived from query languages for ssd (variables, patterns, templates, ...)
- adapted to XML syntax
  Note: XML-QL queries are not valid XML
- relies on an edge-labeled graph model
- regular path expressions allow for arbitrary deep selection
  but we cannot restructure along the way
  Example: replace all <a> tags with <A> and leave the rest as is

XSLT

- part of the XML Stylesheet Language (XSL)
- XSL has 2 parts
  1. XSLT: specifies XML document transformations
  2. XSL Formatting Objects: language to define XML display
- XSLT is based on a different computation model:
  structural recursion

Structural recursion

Computation model first introduced for ssd in UnQL

- data is an ordered tree (references not handled)
  and is viewed as sets with union operator
- multiple functions with a restricted form of recursion
  (simpler to implement than e.g., Datalog)
- in general more powerful than select-from-where
  (can make changes arbitrarily deep in the data)
- in XSL limited expressive power (e.g., no joins)

Structural recursion — Example

Assume the following data:

$$\{a{:}3,\ a{:}\{b{:}\text{“xxx”},\ c{:}5\},\ b{:}4\} = \{a{:}3\} \cup \{a{:}\{b{:}\text{“xxx”},\ c{:}5\}\} \cup \{b{:}4\}$$
$$= \{a{:}3\} \cup \{a{:}\{b{:}\text{“xxx”}\} \cup \{c{:}5\}\} \cup \{b{:}4\}$$

We want to retrieve all integers in the data

$$\begin{array}{rcl} f(v) & = & \text{if } isInt(v) \text{ then } \{\text{“result”} : v\} \text{ else } \{\} \\ f(\{\}) & = & \{\} \\ f(\{\ell : t\}) & = & f(t) \\ f(t_1 \cup t_2) & = & f(t_1) \cup f(t_2) \end{array}$$

Answer: {result:3, result:4, result: 5}

XSLT — Templates and rules

- a query is a collection of template rules
- a template rule has a match pattern and a template

Rule application:

- the rules are applied starting from the root
- when a pattern matches a node, the template specifies
  - what goes in the output
  - how to continue processing on selected nodes (typically the children of the matched node)
- priorities among rules used to disambiguate

XSLT — Example

Retrieve titles of books and articles in the bibliography database

```
<xsl:template>
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="/bib//(book|article)/title">
   <result>
     <xsl:value-of/>
   </result>
</xsl:template>
```

Equivalent query in XML-QL

```
where <bib.*.(book|article).title> $T </> in input
construct <result> $T </result>
```


XSLT — Path expressions

For matching and selecting nodes (defined in XPath)

bib	matches a bib element
*	matches any element
/	matches the root element
/bib	matches a bib element immediately after root
/bib/book	matches a book following a bib
/bib//book	matches a book following a bib at any depth
//book	matches a book at any depth
book|article	matches a book or an article
@price	matches a price attribute
/bib/book/@price	matches the price attribute of a book
book[year]	matches a book if it has a year subelement

XSLT — No variables

XSLT does not use variables

- concise representation
- nonlinear patterns can be expressed by means of qualifiers

  book[year and publisher/name and @language]

  matches a book, but only if it has a year subelement, a publisher subelement with a name, and a language attribute
- cannot express joins (not relationally complete)

XSLT — Deep restructuring

```
<xsl:temp match="a"> <A> <xsl:apply-temps/> </A> </xsl:temp>
<xsl:temp match="b"> <B> <xsl:value-of/> </B> </xsl:temp>
<xsl:temp match="c"> <xsl:apply-temps/> </xsl:temp>
<xsl:temp match="*"> <xsl:element name="name()">
                       <xsl:apply-temps/>
                     </xsl:element> </xsl:temp>
```

```
<a> <c> <a> <b>1</b>
            <b>2</b>
        </a>
        <f> <b>3</b> </f>
    </c>
    <b> <c>4</c> </b>
</a>
```

$\Longrightarrow$

```
<A>     <A> <B>1</B>
            <B>2</B>
        </A
        <f> <B>3</B> </f>

    <B> <c>4</c> </B>
</A>
```


XSLT is structural recursion

Equivalent to

$$
\begin{aligned}
f(v) &= v \\
f(\{\}) &= \{\} \\
f(\{\ell : t\}) &= \text{if } \ \ell = \text{“a”} \ \text{ then } \ \{\text{“A”} : f(t)\} \\
&\quad\ \text{else if } \ \ell = \text{“b”} \ \text{ then } \ \{\text{“B”} : t\} \\
&\quad\ \text{else if } \ \ell = \text{“c”} \ \text{ then } \ \{f(t)\} \\
&\quad\ \text{else } \ \{\ell : f(t)\} \\
f(t_1 \cup t_2) &= f(t_1) \cup f(t_2)
\end{aligned}
$$

XSLT query = a single structural recursion function

XSLT query with modes = multiple functions

XSLT vs. structural recursion

- applicability:
  - structural recursion: on arbitrary graphs
  - XSLT: on trees only
- termination:
  - structural recursion always terminates
  - XSLT may run forever

```
...
<xsl:template match="e">
   <xsl:apply-templates select="/"/>
</xsl:template>
```

XSLT — Summary

- perfectly integrated with XML
- not relational complete (no joins)
- complex (look at the XSL document)
- still work in progress (version 1.1 is a working draft)

Suitability:

- good for formatting (design goal)
- can be used for querying
- not suitable for data integration

XML Query Algebra

Recent proposal of the W3C (December 2000)

Advantages of having a query algebra:

- used to give a semantics for a query language
- used to support query optimizations

Features of the XML Query Algebra

- uses a simple type system (basically XSchema)
- statically typed (output type of query known at compile time)

XML Query Algebra — Operations

- projection: by means of path navigation as in XPath
- selection, quantification, join
- iteration: over elements in a document
- supports local order (among siblings) but not global order
- built in aggregation functions (avg, count, max, min, sum)
- recursive functions (structural recursion)

Summary of query languages for ssd

- studied extensively in the context of semistructured data
- powerful features: regular expressions, constructors, Skolem functions, structural recursion
- no standard query language for XML yet (XML-QL, XSLT, XML version of Lorel, . . .)
- XSLT introduced for stylesheets, but can serve as query language substitute (based on structural recursion)

Part 1 — Modeling and querying ssd

$\surd$ Representing semistructured data

$\surd$ Query languages for semistructured data

$\Rightarrow$ Schemas for semistructured data

- motivation for schemas in ssd
- lower and upper bound schemas
- simulation
- schemas in XML

Schemas in semistructured data

Data may contain

- irregular data, but also
- some structure that
  - may not be explicitly stated
  - may be confined to some portion of the data
  - may change without notice
  - may be too complex to be fully described

Structure can be captured by means of a schema
Schema should be concise (much smaller than the data)

Why schemas

Why consider structure?

- to improve data processing (primary motivation)
  - optimize query evaluation (limit search)
  - improve storage
  - construct indexes
- to describe semantics: data integration
  - specify relationships between data (in different sources)
  - useful for query formulation (over the global view)

When is a schema used?

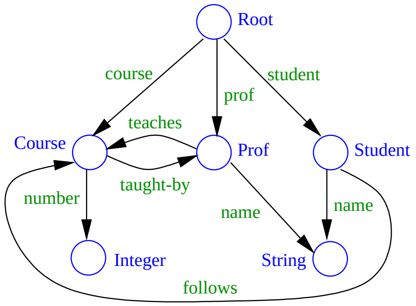
- for relational data: schema needed to interpret binary data
- for ssd: schema specified after the database is populated
  (schema is independent from the data – schema extraction)

Difference with types in databases

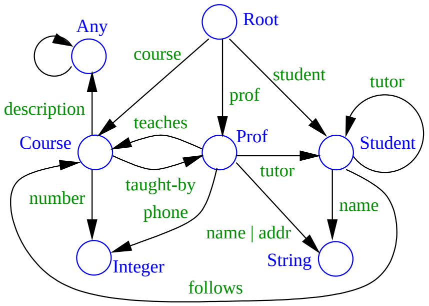
- classes defined less precisely
  $\Rightarrow$ objects may belong to several classes (play several roles)
- some objects may not belong to any class
- typing may be approximate (instances do not conform precisely to a class)

Types of schemas for ssd

Lower-bound schema
shows required edges



Upper-bound schema
shows permitted edges



Applications of schemas

- lower bound schemas: to improve secondary storage
  - one table for each node: attributes are required edges
  - store rest in overflow repository
- upper bound schemas: query optimization
  - replace wildcards with explicit edges

Still much research to be done
(how to exploit the structure when present, but not require it for effective processing?)

Questions to ask

Basic:

Conformance: does this data conform to that schema?

Classification: if yes, then which objects belong to what classes?

Additionally:

Consistency: can this class/schema be populated by some object?

Refinement: do all databases conforming to this schema also conform to that schema?

Lower bound schemas — Unary Datalog

EDB predicates: ref, String, ...
IDB predicates: Student, Prof, ...

Student(S)	$\leftarrow$	ref(S,name,N), String(N), ref(S,follows,C), Course(C)
Prof(P)	$\leftarrow$	ref(P,name,N), String(N), ref(P,teaches,C), Course(C)
Course(C)	$\leftarrow$	ref(C,number,N), Integer(N), ref(C,taught-by,P), Prof(P)
	...	

- Conformance: check Root(&r)
- Classification: check Student(John), Course(CS101), ...

Need greatest fixpoint semantics for Datalog
(we want to type the “largest set” of objects)

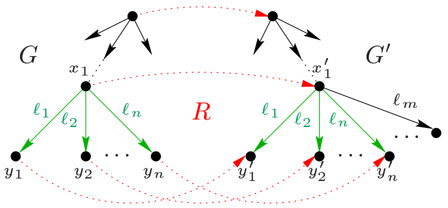
Graph simulation

A simulation between two rooted edge-labeled graphs G and G' is a relation R on their nodes s.t. the roots are in R and

> if $(x, x') \in R$, then for every edge $x \xrightarrow{\ell} y$ in G there is an edge $x' \xrightarrow{\ell} y'$ in G' (same label) s.t. $(y, y') \in R$

Notation: $G \preceq_R G'$

what G can do ...



... G' can do too

Simulation can be computed quite efficiently: $O(|G| \cdot |G'|)$

Using simulation

Given database D and schema S

- for lower bound schemas (required edges)
  - conformance: check $S \preceq_R D$
  - classification: check Student $\preceq_R$ John

  (same as unary Datalog)
- for upper bound schemas (permitted edges)
  - conformance: check $D \preceq_R S$
  - classification: check John $\preceq_R$ Student

where R is the greatest simulation

Extracting schema from data

Given a database, find the most specific schema for it

- lower bound schemas: compute the equivalence classes modulo simulation
- upper bound schemas: Data Guides
  - same set of paths as the database
  - each path is unique
  - essentially the equivalent deterministic automaton (may be exponential)
  - implemented in Lore (Data Guides from DTDs)

XML DTDs

Document Type Definition (DTD)

- basically a grammar
- XML document may contain a DTD
  - a well-formed XML document does not require a DTD
  - a valid XML document has a DTD and the document instance conforms to it

DTDs as grammars

```
<!DOCTYPE bib [ <!ELEMENT bib   (book|article|techrep)*>
                <!ELEMENT book  (title,author+,pub?,year)>
                <!ELEMENT title (#PCDATA)>
                ... ]>
```

Conforming document

```
<bib><book><title>Principles of Database Systems</title>
           <author>Jeffrey D. Ullman</author>
           <year>1982</year>
     </book>
     <article><title>Type Inference for Queries ...</title>
              <author>Tova Milo</author>
              <author>Dan Suciu</author> ...
     </article>
</bib>
```


DTDs as schemas

Not so well suited

- unwanted restrictions on order of elements
  `<!ELEMENT book (title,author+,pub?,year)>`
  means that `<title>` comes before `<author>`, . . .
- cannot restrict range of references
- cannot reuse element definitions (no inheritance)

XML Schema

Overcomes the limitations of DTDs as schema formalism

- simple and complex types
  - builtin simple types (string, integer, . . .) and intervals
  - lists, unions of simple types
  - complex types as in DTDs + unordered sequence of elements
  - types can be derived from other types (extension, restriction)
- abstract types (cannot be directly instantiated)
- constraints
  - cardinality restrictions
  - uniqueness constraints and keys (for any element and attribute, scope can be declared)

XML namespaces

name ::= [*prefix*:]*localpart*

```
<book xmlns:isbn="http://www.isbn-org.org/def">
   <title>Principles of Database Systems</title>
   ...
   <isbn:number> ... </isbn:number>
</book>
```

- avoid name clashes when using external DTD/schema
- provide URI for a DTD/schema
  element type `number` defined in `http://www...`

Summary of schemas

- in semistructured data
  - data and schema are decoupled $\Rightarrow$ schema extraction
  - emphasis is on using schema information in data processing
- in XML
  - data and schema tightly coupled (document must refer to schema)
  - emphasis is on semantics

Summary of Part 1

- ssd is self-describing and has no a priori structure
- represented best as edge/node labeled graph
- powerful features of query languages
  - regular path queries
  - restructuring
  - structural recursion
- schemas may be added (data processing)
- still a lot of work to do

Part 2 — Data integration and ssd

$\Rightarrow$ Ssd in data integration systems
  - data integration systems vs. ssd
  - TSIMMIS
  - Strudel
  - Tukwila
  - querying relations and ssd
- View based query processing for ssd

Data integration systems vs. ssd

Various possibilities:

sources \ global schema	relational	semistructured
relational	GAV, LAV IM, SIMS, DWQ	mediators GAV SilkRoute, Ozone
semistructured	wrappers, XML GAV, LAV Tukwila	wrappers, mediators GAV TSIMMIS, LORE, Strudel

Semistructured global view: more flexibility

Note: many of the systems are for managing ssd rather than proper integration systems

TSIMMIS [Stanford Univ.]

- data model is OEM
- GAV approach, although there is no notion of global schema
- based on mediators
  - mediators are linked together and to sources (multiple levels)
  - each mediator exports a set of predicates
  - each mediator answers queries about its own predicates (using wrapped sources and/or other mediators)
- automatic generation of wrappers and mediators
  - mediators specified in MSL (declarative)
  - MSL can be encoded in Datalog

Strudel [Univ. of Washington]

- actually a system for Web site implementation: separates
  - management of Web site's data ⇐ integration system
  - specification of content and structure
  - visual presentation
- based on an edge-labeled ssd model
- StruQL query language (variant of UnQL)
- GAV approach with materialized views (assumes data underlying web-site is small)

Tukwila [Univ. of Washington]

- LAV approach based on an enhanced query rewriting algorithm (conjunctive queries)
- main point: adaptivity
  - interleaves planning and execution (may send only partial plans to execution engine)
  - adaptive operators to minimize time required to obtain first answer
  - operator to efficiently union data from multiple sources
  - query plan contains conditional nodes
- recently extended with an algorithm to efficiently evaluate regular path queries over streaming XML data

Querying relations and ssd

SilkRoute [Univ. of Washington] framework to express mappings of relational data into XML that conforms to arbitrary DTDs

- query language RXL for mapping relational sources to XML views
- query composition algorithm XML-QL $+$ RXL $=$ RXL

WSQ/DSQ [Stanford Univ.] combines query facilities of DBs with Web search engines

- Web-Supported (Database) Queries: allows one to write queries that automatically execute Web searches
- Database-Supported (Web) Queries: enhances Web keyword searches with information in the DB

Summary of ssd in data integration systems

- with ssd integration follows the GAV approach
- ssd in the global view provides more flexibility
- ssd sources are usually wrapped
- recently support for XML sources builtin

What about ssd and LAV integration?

Open problem: view-based processing of full ssd queries

- results on queries involving regular paths — very hard (see later)
- what to do with the restructuring part is open

Part 2 — Data integration and semistructured data

$\surd$ Ssd in data integration systems

$\Rightarrow$ View based query processing for semistructured data

- (conjunctive) regular path queries
- view based query rewriting for RPQs
- 2-way RPQs and 2-way automata
- view based query answering for 2-way RPQs

Queries over ssd revisited

Composed of two parts:

- selection part: makes use of regular paths
- restructuring part: object creation and oid invention

Can be encoded in Datalog with function symbols

More targeted encoding of selection part:
conjunctive regular path queries

$$Q(x_1, \ldots, x_n) \leftarrow y_1\, E_1\, y_2 \wedge \cdots \wedge y_{2m-1}\, E_m\, y_{2m}$$

where each atom $y\,E\,y'$ is a regular path query

Regular path queries

Syntactically:

a regular expression E over the edge labels

bib$\cdot$_$^*\cdot$(book $+$ article)$\cdot$title

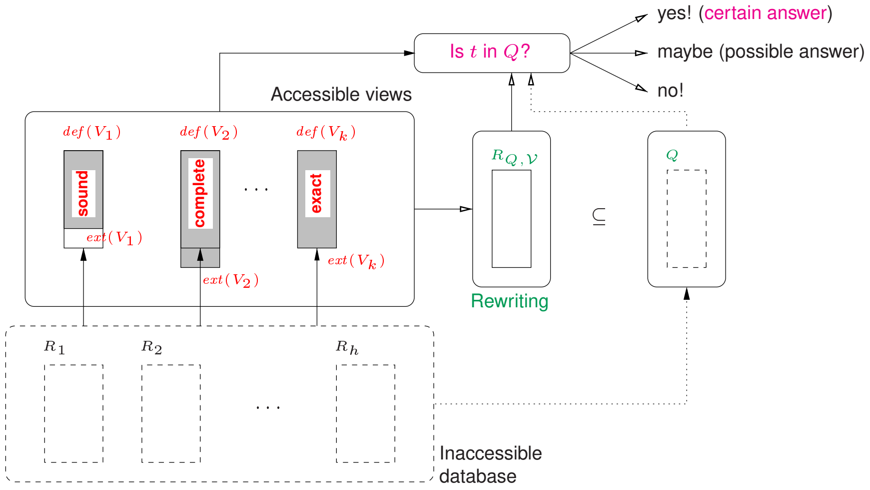
Semantically:

returns set of pairs of nodes connected by a path in $L(E)$

Are the basic constituents of ssd queries

We address rewriting of RPQs by rewriting of regular expressions

View based query rewriting



Rewriting of regular expressions

Given an alphabet Σ and

- a regular expression E_0 over Σ
- a set $\mathcal{E} = \{E_1, \ldots, E_k\}$ of regular expressions over Σ

re-express E_0 as a combination of $E_1, \ldots, E_k$.

$$
\begin{aligned}
E_0 &= a{\cdot}(b{\cdot}a + c)^* \\
\mathcal{E} &= \{a,\ \ c,\ \ a{\cdot}c^*{\cdot}b\} \\
E_0 &= a{\cdot}(b{\cdot}a + c)^* \ = \ (a{\cdot}c^*{\cdot}b)^*{\cdot}a{\cdot}c^* \ = \ (a{\cdot}c^*{\cdot}b)^* \cdot (a) \cdot (c)^*
\end{aligned}
$$

We denote the language of E by $\mathcal{L}(E)$

Rewriting of regular expressions — Formally

Let $\mathcal{V} = \{v_1, \ldots, v_k\}$, with $\mathit{def}(v_i) = E_i$

For a language L over $\mathcal{V}$, let

$$\mathit{expand}(L) = \bigcup_{v_1 \cdots v_n \in L} \{w_1 \cdots w_n \mid w_i \in \mathcal{L}(\mathit{def}(v_i))\}$$

A formalism R for defining a language $\mathcal{L}(R)$ over $\mathcal{V}$ is a rewriting of E_0 wrt $\mathcal{V}$ if $\mathit{expand}(\mathcal{L}(R)) \subseteq \mathcal{L}(E_0)$.

Regular expression: $E_0 = a{\cdot}(b{\cdot}a + c)^*$

Views: $\mathit{def}(v_1) = a$, $\mathit{def}(v_2) = c$, $\mathit{def}(v_3) = a{\cdot}c^*{\cdot}b$

$\mathit{expand}(\mathcal{L}(v_3^*{\cdot}v_1{\cdot}v_2)) = \{ac,\ abac,\ acbac,\ \ldots\} = \mathcal{L}(E_0)$

$\Longrightarrow$ $v_3^*{\cdot}v_1{\cdot}v_2$ is a rewriting of E_0 wrt $\mathcal{V}$

Maximal rewritings

$R_{E_0,\mathcal{V}}$ is $\mathcal{V}$-maximal if for each $R'_{E_0,\mathcal{V}}$ we have that

$$\mathcal{L}(R'_{E_0,\mathcal{V}}) \subseteq \mathcal{L}(R_{E_0,\mathcal{V}})$$

$R_{E_0,\mathcal{V}}$ is Σ-maximal if for each $R'_{E_0,\mathcal{V}}$ we have that

$$\mathit{expand}(\mathcal{L}(R'_{E_0,\mathcal{V}})) \subseteq \mathit{expand}(\mathcal{L}(R_{E_0,\mathcal{V}}))$$

We are interested in computing Σ-maximal rewritings

Result: Every $\mathcal{V}$-maximal rewriting is also Σ-maximal

How can we compute the $\mathcal{V}$-maximal rewriting?

Rewriting of regular expressions — Ideas

Characterize bad rewritings

a rewriting R is bad if $\mathcal{L}(R)$ contains a $\mathcal{V}$-word w s.t. $\mathit{expand}(\{w\})$ contains some Σ-word not in $\mathcal{L}(E_0)$

Regular expressions are of little help

Use automata instead !!!

- construct automaton that accepts all bad rewritings
- complement it

Computing rewriting of regular expressions

Input: Σ-regexp E_0, set $\mathcal{V} = \{v_1, \ldots, v_k\}$ of Σ-regexp views

Output: automaton $R_{E_0,\mathcal{V}}$ (maximal rewriting of E_0 wrt $\mathcal{V}$)

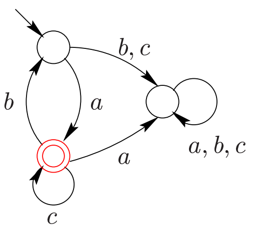
1. construct deterministic automaton $A_d = (\Sigma, S, s_0, \rho, F)$ for E_0
2. let $B = (\mathcal{V}, S, s_0, \rho', S \setminus F)$, where

   $s_j \in \rho'(s_i, v)$ iff $\exists w \in \mathcal{L}(\mathit{def}(v))$ s.t. $s_j \in \rho^*(s_i, w)$
3. $R_{E_0,\mathcal{V}}$ is $\overline{B}$, i.e., the complement of B

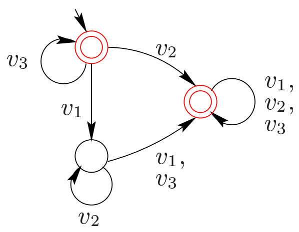
Rewriting of regular expressions – Example

$$E_0 = a{\cdot}(b{\cdot}a + c)^* \qquad \mathcal{V} = \{\ v_1 \ ,\ v_2 \ ,\ v_3 \ \}$$

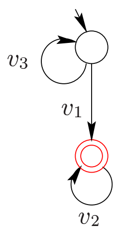
$$v_1 \downarrow a \qquad v_2 \downarrow c \qquad v_3 \downarrow a{\cdot}c^*{\cdot}b$$



A_d



B



$\overline{B} = R_{E_0,\mathcal{V}}$

Computing rewriting of regular expressions (cont.)

Observation: B accepts the bad rewritings

a $\mathcal{V}$-word $v_1 \cdots v_n$ is accepted by B
iff
there is a Σ-word in $\mathit{expand}(\{v_1 \cdots v_n\})$ rejected by A_d

Thm. $R_{E_0,\mathcal{V}} = \overline{B}$ is a $\mathcal{V}$-maximal rewriting of E_0 wrt $\mathcal{V}$.

Proof: $\overline{B}$ is a rewriting of E_0 wrt $\mathcal{V}$: By the observation above.

$\overline{B}$ is $\mathcal{V}$-maximal: By contradiction. Let R be a rewriting s.t. $\mathcal{L}(R) \not\subseteq \mathcal{L}(\overline{B})$ and let $v_1 \cdots v_n \in \mathcal{L}(R)$ but $v_1 \cdots v_n \notin \mathcal{L}(\overline{B})$. Hence $v_1 \cdots v_n$ is accepted by B.
By observation above there is a Σ-word in $\mathit{expand}(\{v_1 \cdots v_n\})$ rejected by A_d.
Contradiction to $v_1 \cdots v_n \in \mathcal{L}(R)$ and R being a rewriting.

Exact rewritings

A rewriting R of E_0 wrt $\mathcal{V}$ is exact if $expand(\mathcal{L}(R)) = \mathcal{L}(E_0)$

To verify whether $R_{E_0,\mathcal{V}}$ is an exact rewriting of E_0 wrt $\mathcal{V}$:

1. Construct an automaton C over Σ accepting $expand(\mathcal{L}(R_{E_0,\mathcal{V}}))$ i.e., replace each v_i edge in $R_{E_0,\mathcal{V}}$ by automaton A_i accepting $\mathcal{L}(def(v_i))$. Observe that $\mathcal{L}(C) \subseteq \mathcal{L}(A_d) = \mathcal{L}(E_0)$.
2. Check whether $\mathcal{L}(A_d) \subseteq \mathcal{L}(C)$, i.e., check whether $\mathcal{L}(A_d \cap \overline{C}) = \emptyset$.

Thm. $R_{E_0,\mathcal{V}}$ is an exact rewriting of E_0 wrt $\mathcal{V}$ iff $\mathcal{L}(A_d \cap \overline{C}) = \emptyset$

Complexity of rewriting regular expressions

For a regular expression E_0 and a set $\mathcal{V}$ of regular expressions:

Generating the $\mathcal{V}$-maximal rewriting: in 2EXPTIME
(two determinization steps)

Shortest nonempty rewriting may be of double-exponential size
(hence upper bound for generating $\mathcal{V}$-maximal rewriting is optimal)

Checking existence of nonempty rewriting: EXPSPACE-complete

Checking existence of exact rewriting: 2EXPSPACE-complete
($\overline{C}$ must be constructed "on the fly" while checking for emptiness)

Rewriting of regexps vs. rewriting of RPQs

We have solved rewriting of regular path queries by means of rewriting of regular expressions

Observations:

- we have not restricted in any way the language of the rewriting (it is just a formalism)

  Maximal rewriting of a regular expression is regular

- we have required the rewriting of an RPQ to be a language, i.e., a set of strings — This is already a restriction

Is the rewriting of RPQs we have obtained perfect?

Non-perfectness of RPQ rewriting

Query: $Q = R_1 \cdot R_3 + R_2 \cdot R_4$
Views: $\mathcal{V} = \{V_1, V_2, V_3\}$ with $def(V_1) = R_1$
$def(V_2) = R_2$
$def(V_3) = R_3 + R_4$

Maximal rewriting of Q wrt $\mathcal{V}$ that is an RPQ is empty.

Consider the CRPQ over $\mathcal{V}$

$$R(x,y) \leftarrow x\, V_1\, z \wedge x\, V_2\, z \wedge z\, V_3\, y$$

By expanding V_1, V_2, and V_3 with their definitions we obtain

$$R(x,y) \leftarrow x\, R_1\, z \wedge x\, R_2\, z \wedge z\, (R_3 + R_4)\, y$$

which is contained in Q.

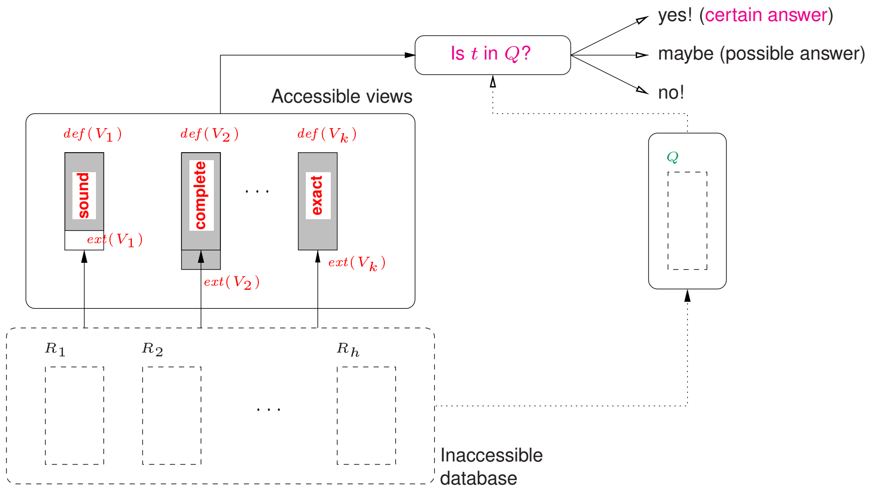
Hence R is a CRPQ that is a nonempty rewriting of Q.

Problems related to rewriting in ssd

- can we compute the perfect rewriting?
- in what query language can it be expressed?
- we would like queries and languages to be expressed in the same query language $\Rightarrow$ we need at least CRPQs
- how to apply regexps to CRPQs?

Let's look at view based query answering first

View based query answering



View based query answering for 2RPQs

Ideas:

- search for counterexample databases
- encode counterexample databases as words
- use (two-way) finite automata to check if a candidate is indeed a counterexample database

Two-way regular path queries

Expressed as finite automata over $\Sigma = \Sigma' \cup \{p^- \mid p \in \Sigma'\}$ (p^- denotes the inverse of the binary relation p)

Semipath in a DB $\mathcal{B}$: $y_1 \xrightarrow{r_1} y_2 \rightarrow \cdots \rightarrow y_q \xrightarrow{r_q} y_{q+1}$, with each $r_i \in \Sigma$

- if $r_i = p$ then $y_i \xrightarrow{p} y_{i+1}$ is in $\mathcal{B}$
- if $r_i = p^-$ then $y_{i+1} \xrightarrow{p} y_i$ is in $\mathcal{B}$

The semipath conforms to a regexp E if $r_1 \cdots r_q \in \mathcal{L}(E)$

The answer to a 2RPQ E over $\mathcal{B}$ is the set $ans(E, \mathcal{B})$ of pairs of nodes connected by a semipath that conforms to E.

A Σ-word $w = r_1 \cdots r_q$ satisfies E iff $(x, y) \in ans(E, \mathcal{B}_w)$, where $\mathcal{B}_w$ is constituted by a single semipath $x \xrightarrow{r_1} \cdots \xrightarrow{r_q} y$ labeled with w

Two-way automata (2NFA)

Similar to standard one-way automata (1NFA)

$$A = (\Gamma, S, S_0, \rho, F)$$

but with transition function $\rho : S \times \Sigma \to 2^{S\times\{-1,0,1\}}$
mapping each state to a set of pairs

- new state
- moving direction (left, don't move, or right)

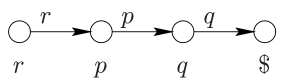
Two-way automata accept regular languages

Given a 2NFA A with n states, one can construct

- a 1NFA B_1 with $O(2^{n \log n})$ states such that $\mathcal{L}(B_1) = \mathcal{L}(A)$
- a 1NFA B_2 with $O(2^n)$ states such that $\mathcal{L}(B_2) = \Gamma^* \setminus \mathcal{L}(A)$

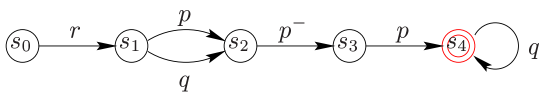
Two-way automata and 2RPQs

Database represented by a word

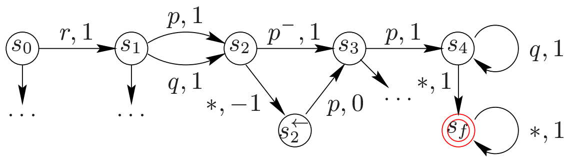


$$Q = r{\cdot}(p+q){\cdot}p^-{\cdot}p{\cdot}q^*$$

1-way automaton that accepts $\mathcal{L}(Q)$



2-way automaton that evaluates Q over a linearized DB



Two-way automata and 2RPQs

From the 2RPQ $E = (\Sigma, S, I, \delta, F)$ we construct the 2NFA

$$A_E \;=\; (\Sigma \cup \{\$\},\quad S \cup \{s_f\} \cup \{s^\leftarrow \mid s \in S\},\quad I,\quad \delta_A,\quad \{s_f\})$$

where δ_A is defined as follows:

- $(s^\leftarrow, -1) \in \delta_A(s, \ell)$, for each $s \in S$ and $\ell \in \Sigma \cup \{\$\}$
- $(s_2, 1) \in \delta_A(s_1, r)$ and $(s_2, 0) \in \delta_A(s_1^\leftarrow, r^-)$, for each transition $s_2 \in \delta(s_1, r)$ of E.
- $(s_f, 1) \in \delta_A(s, \ell)$, for each $s \in F$ and $\ell \in \Sigma \cup \{\$\}$

Result: w satisfies E iff $w\$ \in \mathcal{L}(A_E)$

View-based query answering — Basic idea

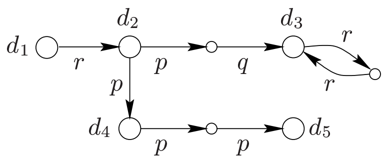
- we search for counterexample databases
- it is sufficient to restrict the attention to canonical DBs
  (a set of simple semipaths $a \stackrel{r_1}{\to} x_1 \cdots x_{n-1} \stackrel{r_n}{\to} b$, where a, b are in the extensions of the views $\mathcal{D}_V$ and $x_1, \ldots, x_{n-1}$ are not in $\mathcal{D}_V$ and do not occur in any other semipath in the set)
- each canonical DB $\mathcal{B}$ can be represented by a word $w_\mathcal{B}$ over $\Sigma_A = \Sigma \cup \mathcal{D}_V \cup \{\$\}$ of the form

  $$\$\, d_1\, w_1\, d_2\, \$\, d_3\, w_2\, d_4\, \$\, \cdots\, \$\, d_{2m-1}\, w_m\, d_{2m}\, \$$$

  where $d_1, \ldots, d_{2m}$ range over $\mathcal{D}_V$, $w_i \in \Sigma^+$, and the $\$$ acts as a separator

Two-way automata and canonical DBs

$\$\ d_1\ r\ d_2\ \$\ d_4\ pp\ d_5\ \$\ d_4\ p^-\ d_2\ \$\ d_3\ rr\ d_3\ \$\ d_2\ pq\ d_3\ \$$



$Q = r\cdot(p \cup q)\cdot p^-\cdot p\cdot q^*$

To verify whether, e.g., (d_1, d_3) satisfies Q in the above DB, we exploit the ability of 2NFA

- to move on the word both forward and backward
- to jump from one position in the word representing a node to any other position representing the same node

View-based query answering — 2NFA construction

If $E = (\Sigma, S, I, \delta, F)$, then $A_{(Q,a,b)} = (\Sigma_A, S_A, \{s_0\}, \delta_A, \{s_f\})$, where $S_A = S \cup \{s_0, s_f\} \cup \{s^{\leftarrow} \mid s \in S\} \cup S \times \mathcal{D}$, and

1. $(s^{\leftarrow}, -1) \in \delta_A(s, \ell)$, for each $s \in S$ and $\ell \in \Sigma \cup \mathcal{D}_V$
2. $(s_2, 1) \in \delta_A(s_1, r)$ and $(s_2, 0) \in \delta_A(s_1^{\leftarrow}, r^-)$, for each $s_2 \in \delta(s_1, r)$
3. $(s_0, 1) \in \delta_A(s_0, \ell)$, for each $\ell \in \Sigma_A$, $(s, 0) \in \delta_A(s_0, a)$ for each $s \in I$
4. $$\begin{array}{rcl@{\qquad}rcl} ((s,d),0) & \in & \delta_A(s,d) & ((s,d),0) & \in & \delta_A(s^{\leftarrow},d) \\ ((s,d),1) & \in & \delta_A((s,d),\ell) & ((s,d),-1) & \in & \delta_A((s,d),\ell) \\ (s,0) & \in & \delta_A((s,d),d) & (s,1) & \in & \delta_A(s,d) \end{array}$$
5. $(s_f, 0) \in \delta_A(s, b)$, for each $s \in F$ and $(s_f, 1) \in \delta_A(s_f, \ell)$.

Result: $A_{(Q,a,b)}$ accepts $w_{\mathcal{B}}$ iff $(a, b) \in ans(Q, \mathcal{B})$

View-based query answering — Technique

To check whether $(c, d) \notin ans(Q, \mathcal{B})$ for some DB $\mathcal{B}$ consistent with the views, we check for nonemptiness of the 1NFA A_{ODA}, constructed as the intersection of

- the 1NFA A_0 that accepts $(\$\cdot\mathcal{D}_V\cdot\Sigma^+\cdot\mathcal{D}_V)^*\cdot\$$
- the 1NFAs corresponding to the various $A_{(def(V_i),a,b)}$ (for each view V_i and for each pair $(a, b) \in ext(V_i)$)
- the 1NFAs corresponding to the complement of each A_{V_i} (for each view V_i, A_{V_i} checks whether a pair of objects other than those in $ext(V_i)$ is in $ans(def(V_i), \mathcal{B})$)
- the 1NFA corresponding to the complement of $A_{(Q,c,d)}$

Complexity of view based query answering

Can be measured in three different ways:

data complexity: as a function of the size of the view extensions
$ext(V_1) \cup \cdots \cup ext(V_k)$

expression complexity: as a function of the size of the query Q
and of the view definitions $def(V_1), \ldots, def(V_k)$

combined complexity: as a function of the size of both
$ext(V_1) \cup \cdots \cup ext(V_k)$ and the expressions Q,
$def(V_1), \ldots, def(V_k)$

View-based query answering — Complexity

- All 2NFAs constructed above are of linear size in the size of Q, $def(V_1), \ldots, def(V_k)$, and $ext(V_1), \ldots, ext(V_k)$. Hence, the corresponding one-way automata would be exponential
- However, we do not need to construct A_{ODA} explicitly. Instead, we can construct it on the fly while checking for nonemptiness

View-based query answering for 2RPQs is PSPACE-complete wrt expression complexity and combined complexity

View-based query answering — Complexity

Assump. on domain	Assump. on views	Complexity		
		data	expression	combined
closed	all sound	co-NP	co-NP	co-NP
	all exact	co-NP	co-NP	co-NP
	arbitrary	co-NP	co-NP	co-NP
open	all sound	co-NP	PSPACE	PSPACE
	all exact	co-NP	PSPACE	PSPACE
	arbitrary	co-NP	PSPACE	PSPACE

(all results are completeness results)

Extension of technique

The technique based on 2NFAs and linearized databases can be extended to provide optimal upper bounds for

- rewriting of 2RPQs
- containment of conjunctive 2RPQs
- view-based query processing for generalized 2RPQs (i.e., linear conjunctive 2RPQs)
- view-based query answering for branching 2RPQs
- view-based query answering for conjunctive 2RPQs

Summary of view based query processing for ssd

- selection part of queries over ssd is a conjunctive RPQ
- techniques for view based query processing for conjunctive 2RPQ are based on two-way automata
- high computational complexity
  - existence of nonempty rewriting for 2RPQs: EXPSPACE-complete
  - view based query answering for 2RPQs: PSPACE-complete wrt combined complexity and coNP-complete wrt data complexity
  - query containment and view-based query answering for conjunctive 2RPQs: EXPSPACE-complete