Managing Web Data

Dan Suciu

AT&T Labs

How the Web is Today

- HTML documents
- all intended for human consumption
- many are generated automatically by applications

Paradigm Shift on the Web

- applications consuming HTML documents are out there (Junglee, Netbot): brittle technology
- the new Web standard XML makes data exchange between applications radically simpler
  - XML generated by applications
  - XML consumed by applications
- data exchange
  - across platforms (enterprise inter-operability)
  - across enterprises

Web shifts from collection of documents to data and documents

Database Community Can Help

Relevant fields:

- query optimization, query processing
- views, transformations
- data warehouses, data integration systems
- mediators, query rewriting
- secondary storage, indexes

But it Needs a Paradigm Shift Too

Web data is different from database data:

- self-describing, schema-less
- structure changes without notice
- heterogeneous, deeply nested, irregular
- documents and data mixed together
- accessed through limited capabilities
- distributed, linked

XML was designed by document experts, not database experts !

Need to extend data management to Web data management

What This Tutorial is About

- what the database community has done: *semistructured data*
  - data model: OEM
  - query languages: Lore, UnQL, StruQL, etc.
  - schemas
- what the Web community has done:
  - data format: XML
  - transformation language: XSL
  - schemas
- where they meet and where they differ

Outline

- Semistructured data and XML
- Query languages
- Schemas
- Systems issues
- Conclusions

Part 1

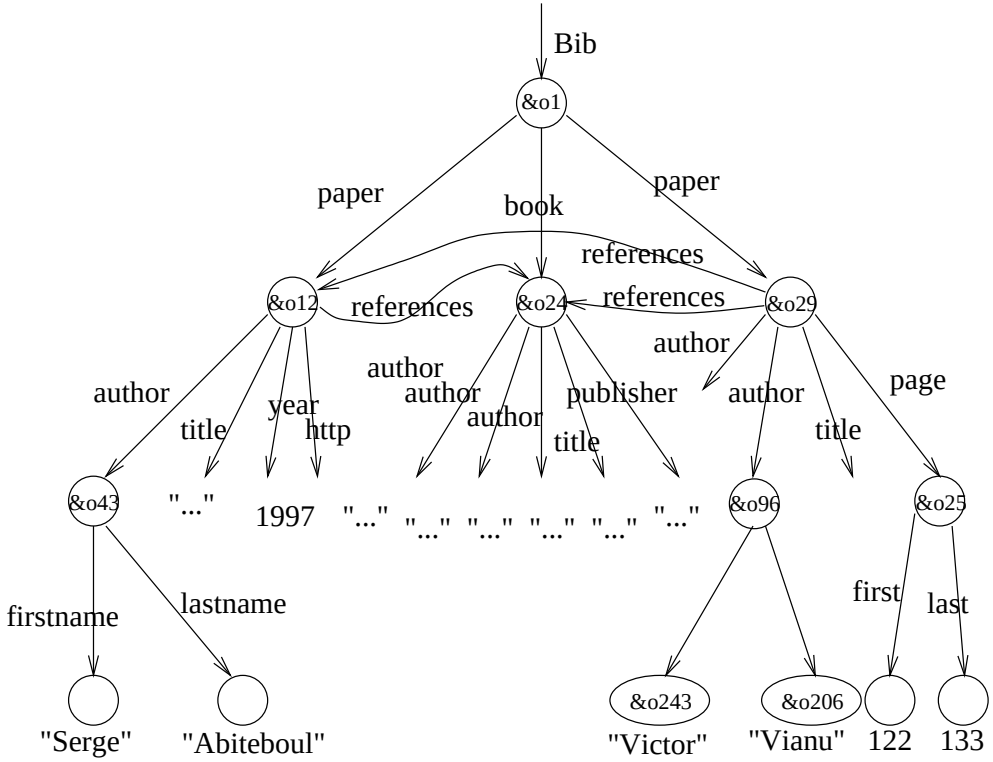
Semistructured Data and XML

Semistructured Data

Origins:

- integration of heterogeneous sources
- data sources with non-rigid structure
- biological data
- *Web data*

The Semistructured Data Model



Object Exchange Model (OEM): { - complex objects: &o1, &o12, ... ; - atomic objects: &o243, &o206, ... }

Semistructured data is an edge-labeled graph

Syntax for Semistructured Data

Serializing the data:

```
Bib:&o1 {  paper: &o12 { ... },
           book: &o24 { ... },
           paper: &o29
             {  author: &o52 "Abiteboul",
                author: &o96 { firstname: &o243 "Victor",
                               lastname: &o206 "Vianu"},
                title: &o93 "Regular path queries with constraints",
                references: &o12,
                references: &o24,
                page: &o25 { first: &o64 122, last: &o92 133}}}
```

Syntax for Semistructured Data

May omit oid's:

```
{ paper:
  { author: "Abiteboul",
    author: { firstname:  "Victor",
              lastname:   "Vianu"},
    title: "Regular path queries with constraints",
    page: {first: 122, last: 133}}}
```

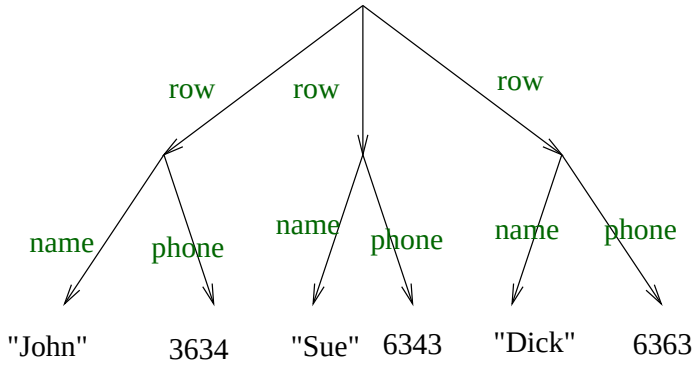
Characteristics of Semistructured Data

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

self-describing, irregular data, no a priori structure

Comparison of the Semistructured and Relational Model

name	phone
John	3634
Sue	6343
Dick	6363



{ row : {name: "John", phone: 3634},
row : {name: "Sue", phone: 6343},
row : {name: "Dick", phone: 6363} }

schema components become labels in semistructured data

XML

New standard approved by W3C, to complement HTML

Origins: structured text (SGML)

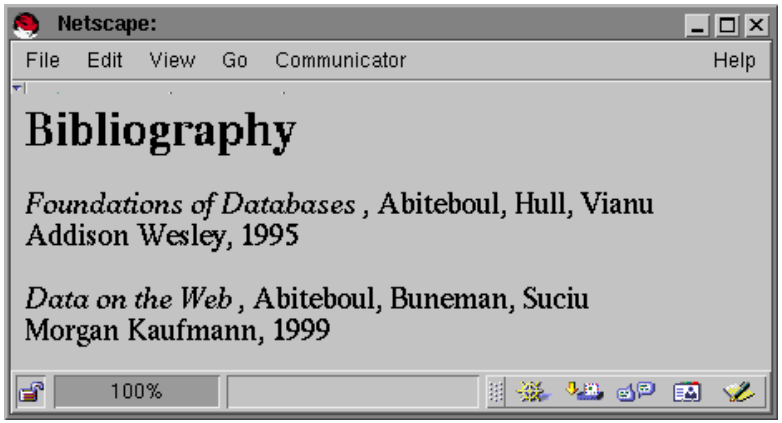
Motivation:

- HTML describes presentation
- XML describes content
- XML v.s. HTML [Bosak'97]
  - users define new tags
  - arbitrary nesting
  - validation is possible

From HTML to XML

HTML:

```
<h1> Bibliography </h1>
<p>  <i> Foundations of Databases </i>, Abiteboul, Hull, Vianu
     <br> Addison Wesley, 1995
<p>  <i> Data on the Web </i>, Abiteboul, Buneman, Suciu
     <br> Morgan Kaufmann, 1999
```



HTML describes the presentation

XML Basics

```
<bibliography>  <book>  <title> Foundations of Databases </title>
                        <author> Abiteboul </author>
                        <author> Hull </author>
                        <author> Vianu </author>
                        <publisher> Addison Wesley </publisher>
                        <year> 1995 </year>
                </book>
                ...
</bibliography>
```

XML describes the content

XML Terminology

- tags: book, title, author, ...
- start tag: <book>
- end tag: </book>
- elements: <book>...</book>, or <author>...</author>
- elements are nested
- empty elements: <red></red>, or <red/>
- an XML document consists of a single element called *root element*

a *well-formed* XML document is one with correctly matching tags

More XML: Attributes

```
<book price="55" currency="USD">
  <title> Foundations of Databases </title>
  <author> Abiteboul </author>
  <author> Hull </author>
  <author> Vianu </author>
  <publisher> Addison Wesley </publisher>
  <year> 1995 </year>
</book>
```

- attributes price, currency, values "55", "USD"

attributes are alternative ways to represent same data

More XML: Oid's and References

```
<person id="o555">  <name>Jane</name>
</person>


<person id="o456">  <name>Mary</name>
                    <children idref="o123 o555" />
</person>

<person id="o123"  mother="o456">
                    <name>John</name>
</person>
```

oids and references in XML are just syntax

XML Data Model

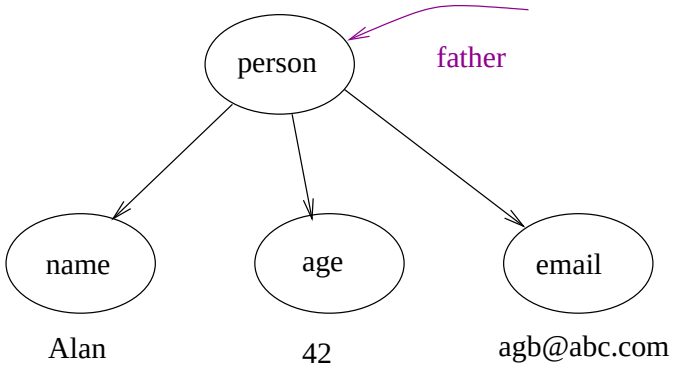
- does not exists
- closest approximation: Document Object Model (DOM):
  - class hierarchy (`node, element, attribute, ...`)
  - object model, not data model: objects have behavior
  - defines API to inspect/modify the document

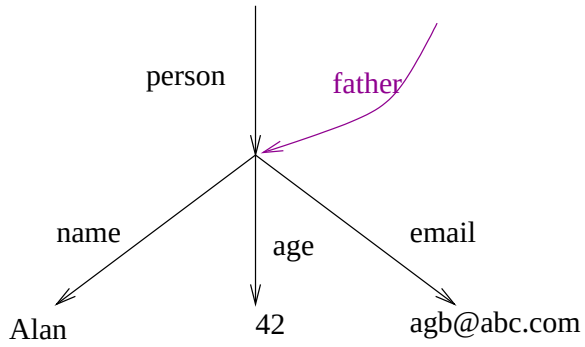
Comparison of XML and Semistructured Data

```
<person id="o123">
  <name> Alan </name>
  <age> 42 </age>
  <email> agb@abc.com </email>
<person>
<person father="o123"> ...
</person>
```

```
{ person: &o123
    { name: "Alan",
      age: 42,
      email: "agb@abc.com"}}

{ person: { father &o123 ...}
}
```





labels on nodes v.s. on edges: trees easy to convert, graphs harder

Comparison of XML and Semistructured Data

- similarities:
  - both described best by a graph
  - both are schema-less, self-describing
- differences:
  - XML is ordered, ssd is not
  - XML can mix text and elements:

```
<talk> Making Java easier to type and easier to type
        <speaker>Phil Wadler</speaker>
</talk>
```

  - XML has other stuff (processing instructions, comments)

Part 2

Query Languages

- Semistructured data and XML
- Query languages
- Schemas
- Systems issues
- Conclusions

Query Languages: Outline

- For Semistructured Data:
  - Lorel (coercions, regular path expressions)
  - UnQL (patterns, constructors)
  - Skolem Functions
- For XML: XML-QL
- A different paradigm:
  - Structural Recursion
  - XSL

Lorel

Part of the Lore system (Stanford)

Lorel adapts OQL to semistructured data

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

Abbreviated to:

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

Lorel v.s. OQL

- implicit coercions (1995 v.s. "1995")
- missing attributes (empty answer v.s. type error)
- set-valued attributes (X.year>1995: may be several years)
- regular path expressions (next)

Lorel

Regular Path Expressions

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

Useful for:

- syntactic substitute for inheritance: paper|book
- navigating partially known structures: lastname?
- transitive closure: reference+

UnQL

Pattern Matching:

```
select  T
where   Bib.paper: { title: T, year: Y, journal: "TODS" }
and     Y > 1995
```

UnQL

Simple Constructors

```
select result:{ fn: F, ln: L, pub: { title: T, year: Y}}
where Bib.paper: { title: T,
                   author: {firstname: F, lastname: L},
                   year: Y}
and Y > 2000
```

Results looks like:

```
{ result: { fn:"John", ln:"Smith",
            pub:{title:"P equals NP", year: 2005}},
  result: { fn:"Joe", ln:"Doe",
            pub:{title:"Errata to P=NP", year: 2006}},
  ...}
```

Skolem Functions

Skolem Functions in databases:

- Maier, 1986: in OO systems.
- Kifer et al, 1989: F-logic.
- Hull and Yoshikawa, 1990: in deductive systems (ILOG).
- Papakonstantinou et al., 1996: in semistructured data (MSL).

Many languages for semistructured data have Skolem functions: MSL, StruQL, Yat, Florid.

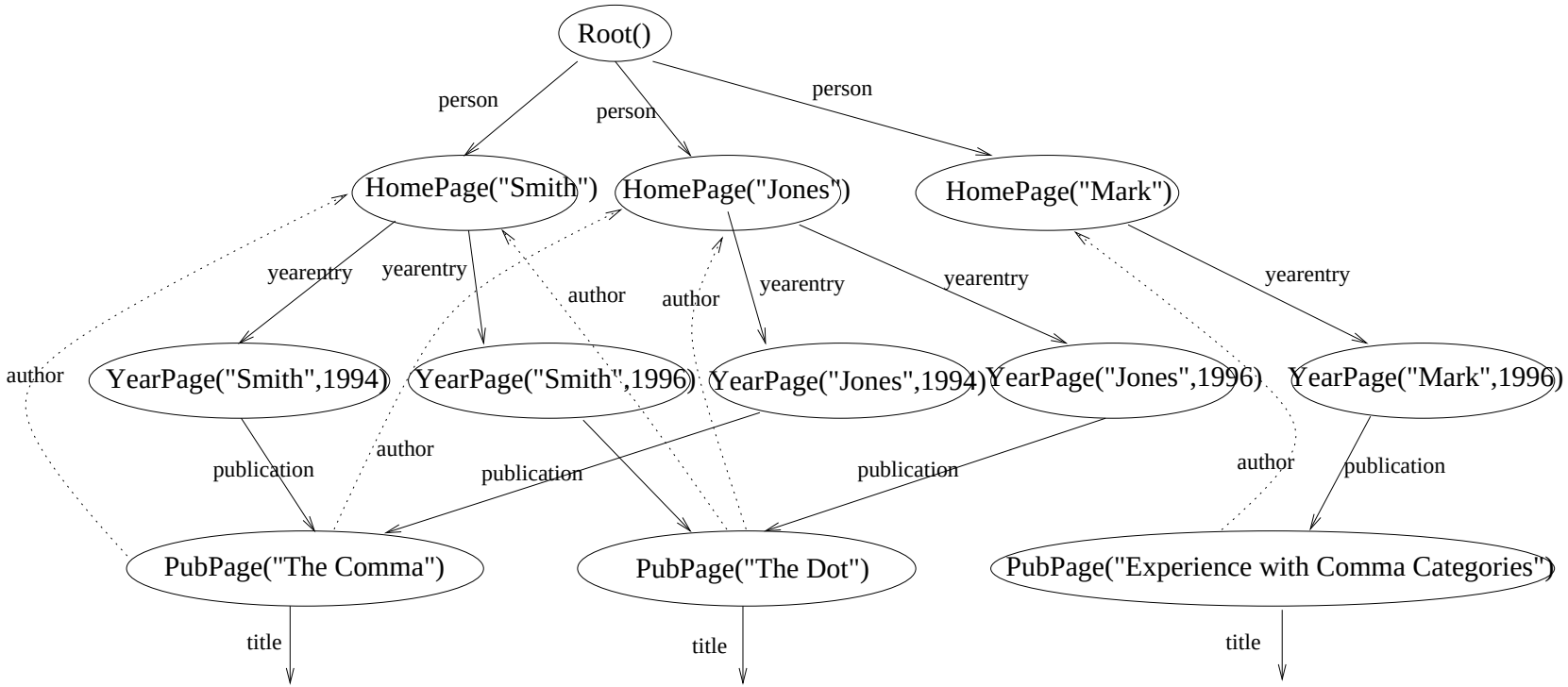
Skolem Functions

Strudel: a Web Site Management System

StruQL: it's query language

```
where   Root → "Bib" → X, X → "paper" → P, P → "author" → A,
        P → "title" → T, P → "year" → Y
create  Root(), HomePage(A), YearPage(A,Y), PubPage(P)
link    Root() → "person" → HomePage(A),
        HomePage(A) → "yearentry" → YearPage(A,Y),
        YearPage(A,Y) → "publication" → PubPage(P),
        PubPage(P) → "author" → HomePage(A)
        PubPage(P) → "title" → T
```

Skolem Functions



A query language for XML: XML-QL

- comes from the semistructured data community; features:
  - regular path expressions
  - patterns
  - simple constructors
  - Skolem Functions

  plus: XML syntax
- relies on the edge-labeled semistructured data model - suffers from model mismatch

Pattern Matching in XML-QL

```
where  <book language="french">
          <publisher><name>Morgan Kaufmann</name>
          </publisher>
          <title> $t </title>
          <author> $a </author>
       </book> in "www.a.b.c/bib.xml"
construct $a
```

Simple Constructors in XML-QL

```
where      <book language=$l>
                    <author> $a </>
           </> IN "www.a.b.c/bib.xml"
construct  <result>  <author> $a </>
                    <lang> $l </>
           </>
```

Note: </> abbreviates </book> or </result>, etc.

Groups titles by authors:

```
<result> <author>Smith</author> <lang>English</lang> </result>
<result> <author>Smith</author> <lang>Mandarin</lang> </result>
...
```

Skolem Functions in XML-QL

```
where      <paper|book>      <title> $t </>
                             <author> $a </>
           </> IN "www.a.b.c/bib.xml"
construct  <result id=F($a)> <author> $a </>
                             <t> $t </>
           </>
```

```
<result> <author>Joe</author> <t><t/> <t><t/> </result>
<result> <author>Jim</author> <t><t/> <t></t> <t></t> </result>
<result> <author>Jan</author> <t><t/> </result>
...
```

A Different Paradigm: Structural Recursion

Data as *sets*, with *union* operator:

{a:3, a:{b:"one",c:5}, b:4} = {a:3} U {a:{b:"one",c:5}, b:4}

= {a:3} U {a:{b:"one",c:5}} U {b:4}

Example: retrieve all integers in the data.

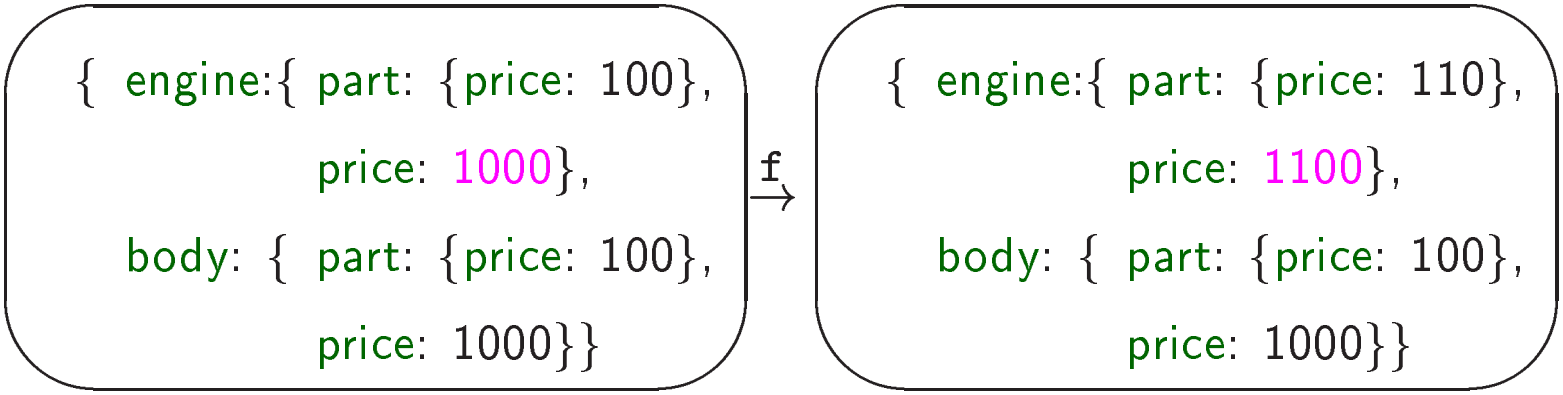
```
f(v)        =  if isInt(v) then {"result": v} else {}
f({})       =  {}
f({l: t})   =  f(t)
f(t1 U t2)  =  f(t1) U f(t2)
```

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

standard textbook programming on trees

Structural Recursion with Multiple Functions

Example: increase all engine's prices by 10%.




```
f(v)          = v
f({})         = {}
f({l: t})     = if l = "engine"
                  then {l: g(t)}
                  else {l: f(t)}
f(t1 U t2)    = f(t1) U f(t2)
```

```
g(v)          = v
g({})         = {}
g({l: t})     = if l = "price"
                  then {l: 1.1*t}
                  else {l: g(t)}
g(t1 U t2)    = g(t1) U g(t2)
```

XSL

- a proposal at W3C (soon a standard)
- implemented in IE 5.0
- purpose: stylesheet specification language:
  - stylesheet: XML $\rightarrow$ HTML
  - more general: XML $\rightarrow$ XML
- limited expressive power, e.g. no joins (but continues to be changed)
- perfectly integrated with XML
- still changing ...

XSL: Templates and Rules

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

Example: retrieve all book titles in the bibliography database:

```
<xsl:template>
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="/bib/*/title">
  <result> <xsl:value-of/>
  </result>
</xsl:template>
```

Path Expressions in XSL

bib	matches a bib element
*	matches any element
/	matches the root element
/bib	matches a bib element immediately after root
bib/paper	matches a paper following a bib
bib//paper	matches a paper following a bib at any depth
//paper	matches a paper at any depth
paper | book	matches a paper or a book
@price	matches a price attribute
bib/book/@price	the price attribute of a book

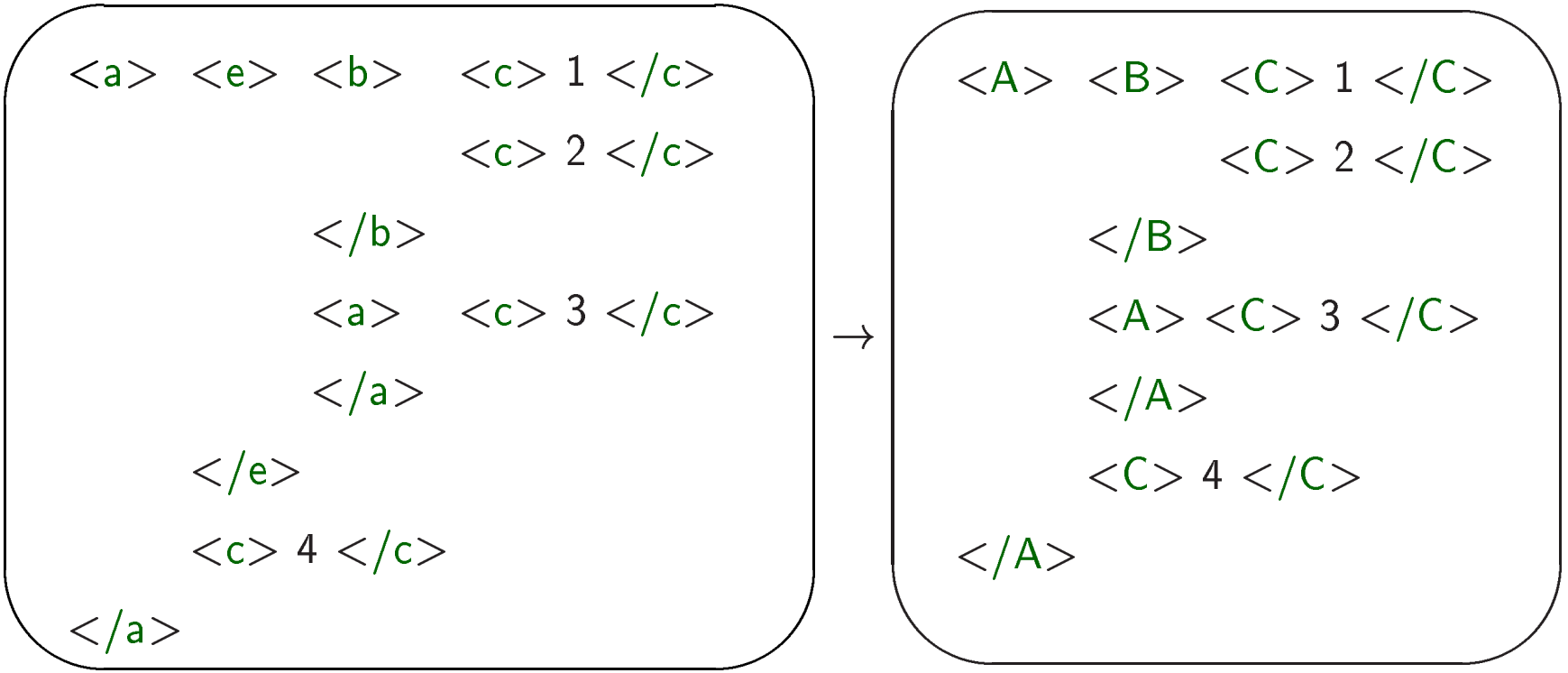
Flow Control in XSL

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

<xsl:template match="a"> <A> <xsl:apply-templates/> </A>
</xsl:template>

<xsl:template match="b"> <B> <xsl:apply-templates/> </B>
</xsl:template>

<xsl:template match="c"> <C> <xsl:value-of/> </C>
</xsl:template>
```


<a> <e> <c> 1 </c>
<c> 2 </c>

<a> <c> 3 </c>

</e>
<c> 4 </c>

→
<A> <B> <C> 1 </C>
<C> 2 </C>
</B>
<A> <C> 3 </C>
</A>
<C> 4 </C>
</A>

XSL is *Structural Recursion*

Equivalent to:

```
f(v)          =  v
f({})         =  {}
f({l:t})      =  if l = "c" then {"C":t}
                 else if l = "b" then {"B":f(t)}
                 else if l = "a" then {"A":f(t)}
                 else f(t)
f(t1 U t2)    =  f(t1) U f(t2)
```

XSL query = a single structural recursion function

XSL query with *modes* = multiple functions

Comparison of XSL and Structural Recursion

- applicability:
  - structural recursion: on arbitrary graphs
  - XSL: on trees only
- termination:
  - structural recursion: always terminates
  - XSL: may run forever. Add this to previous program

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

Stack overflow on IE 5.0

Summary of Query Languages

- studied extensively in the context of semistructured data
- powerful features: regular expressions, constructors, Skolem Functions, structural recursion
- no standard query language for XML yet
- XSL intended to be a style sheet language, can serve as query language substitute
- control flow in XSL is structural recursion

Part 3

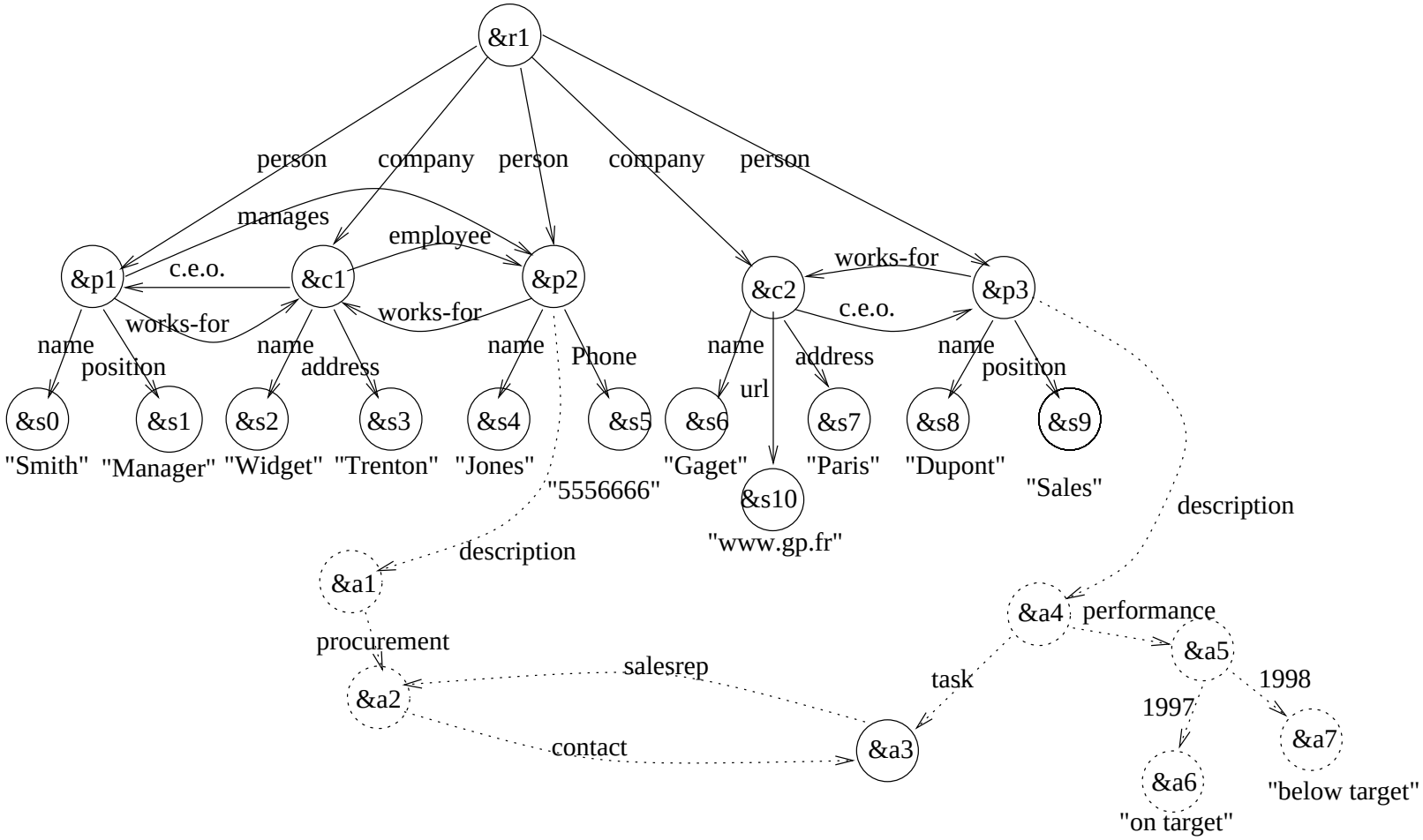
Schemas

- Semistructured data and XML
- Query languages
- Schemas
- Systems issues
- Conclusions

Highlights of Schemas in Semistructured Data

- why ?
  - to describe semantics
  - to improve data processing here lies our interest
- what ?
  - many proposals (both in ssd and in the W3C)
  - this tutorial: show two basic formalisms, and DTDs
- when ?
  - a posteriori
  - RDBMS: need schema a priori to interpret binary data
- how ?
  - independent from the data (may have several schemas !)
  - RDF, DCD: schema hardwired with the data

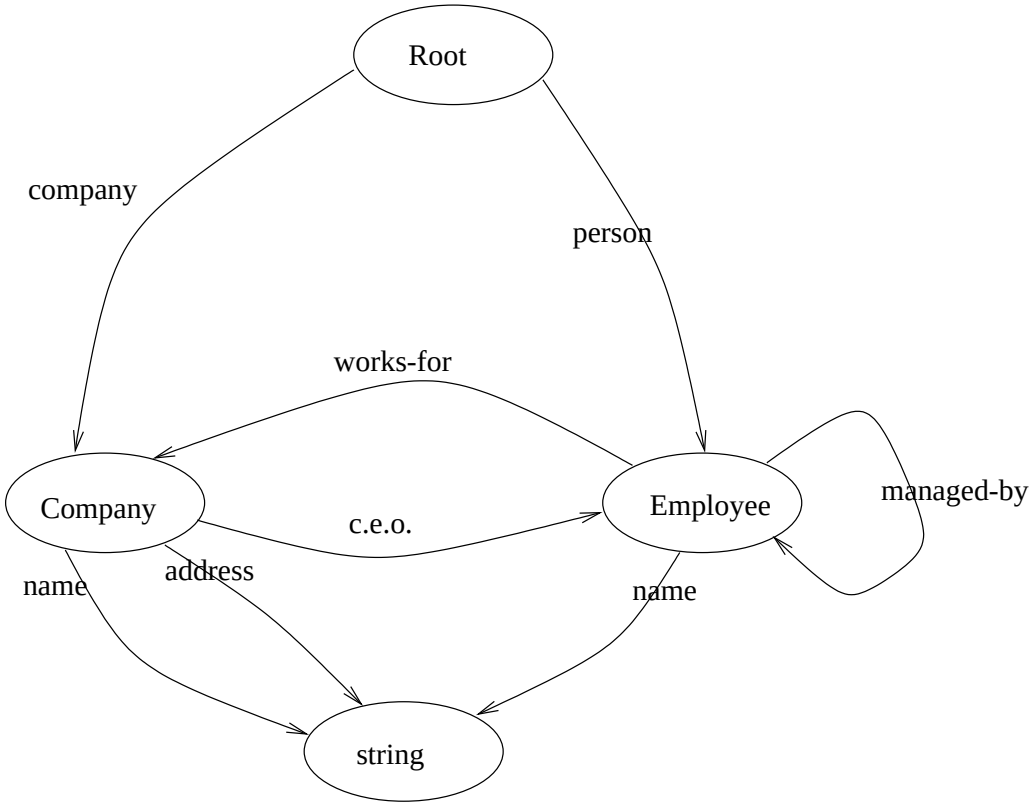
Schemas: An Example



- some well-defined classes: `companies, persons`
- some irregular data

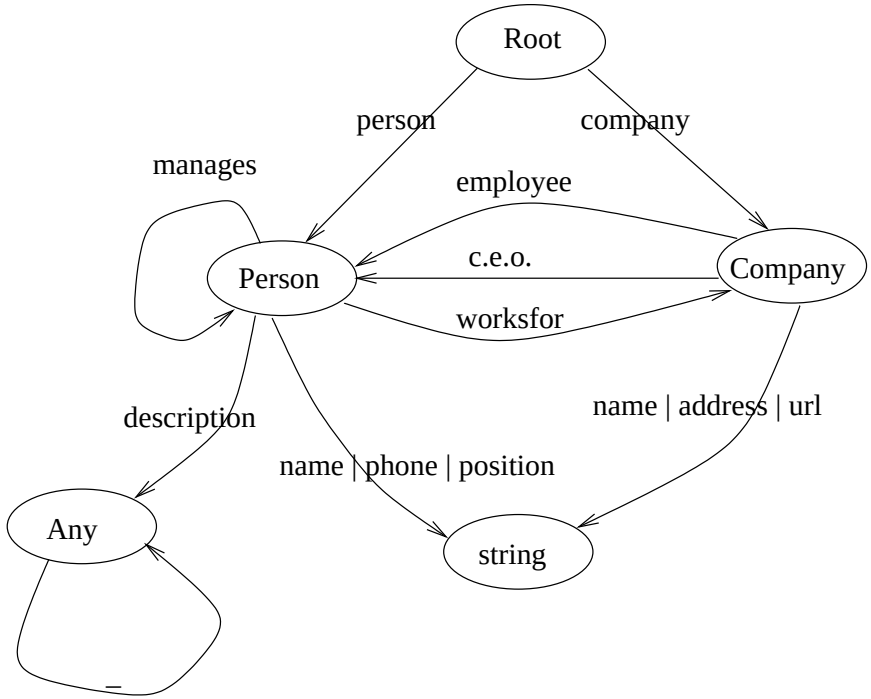
Lower-Bound Schemas

Shows required labels



Upper Bound Schemas

Shows permitted labels.



Called *Schema Graphs* in [Buenman et al'97]

The Two Questions to Ask

Conformance: does that data conform to this schema ?

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

Using Unary Datalog on Lower-Bound Schemas

[Nestorov et. al 1997] Unary Datalog Types

```
Root(X)      :- ref(X,company,C), Company(C),
                ref(X,person,M), Employee(M)
Employee(E)  :- ref(E,works-for,C), Company(C),
                ref(E,managed-by,M), Employee(M)
                ref(E,name,N), string(N)
Company(C)   :- ref(C,ceo.,E), Employee(E),
                ref(C,name,N), string(N),
                ref(C,address,A), string(A)
```

EDBs: ref, string

IDBs:

Root, Employee, Company

- Conformance: check Root(&r1)
- Classification: check Employee(&p2), Company(&c1), etc

Need greatest fixpoint semantics for Datalog !

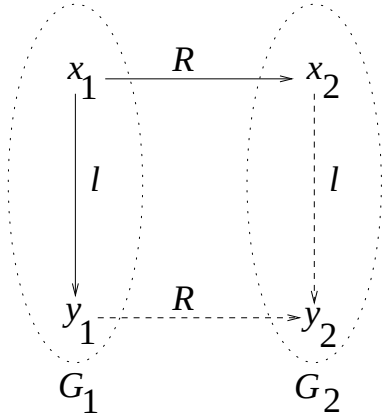
Graph Simulation

Definition Given two edge-labeled graphs G_1, G_2
a *simulation* is a relation R on their nodes s.t.:

- if $(x_1, x_2) \in R$, then for every edge $x_1 \xrightarrow{a} y_1$ in G_1 there exists an edge $x_2 \xrightarrow{a} y_2$ in G_2 (same label) s.t. $(y_1, y_2) \in R$.

Notation: $G_1 \preceq_R G_2$

what G_1 can do ...



... G_2 can do too !

can compute simulation quite efficiently: $O(\mid G_1 \mid \times \mid G_2 \mid)$.
[Henzinger, Henzinger, and Kopke'95]

Using Simulation

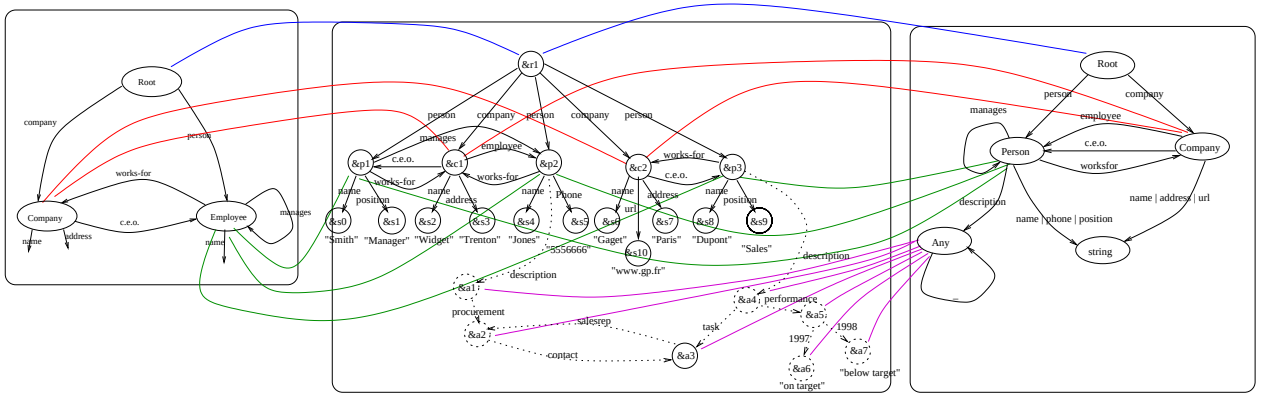
Given data instance D, schema S

- Upper bound schemas:
  - conformance: check $D \preceq_R S$
  - classification: check $(x, c) \in R$
- Lower bound schemas:
  - conformance: check $S \preceq_R D$
  - classification: check $(c, x) \in R$

where R is the largest simulation

For lower bound schemas: same as unary datalog

Conformance and Classification



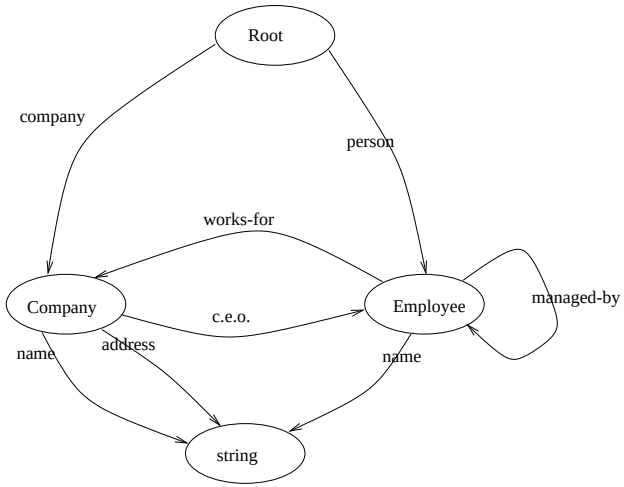
Lower Bound
(Unary Datalog)

Semistructured Data Instance

Upper Bound
(Schema Graph)

Simulation is an efficient way to decide conformance and compute object classification

Application 1: Improve Secondary Storage



Lower-Bound Schema

Company

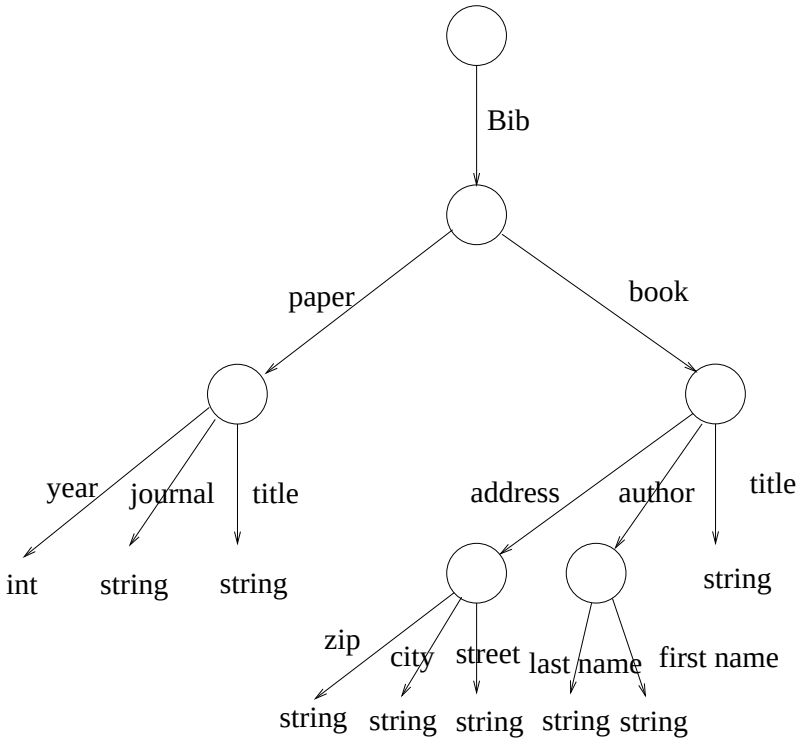
oid	name	address	c.e.o.
...	...	...	...

Employee

oid	name	managed-by	works-for
...	...	...	...

Store rest in overflow repository

Application 2: Query Optimization



Upper-Bound Schema

```
select  X.title
from    Bib._ X
where   X.*.zip = "12345"
```

⇓

```
select  X.title
from    Bib.book X
where   X.address.zip= "12345"
```

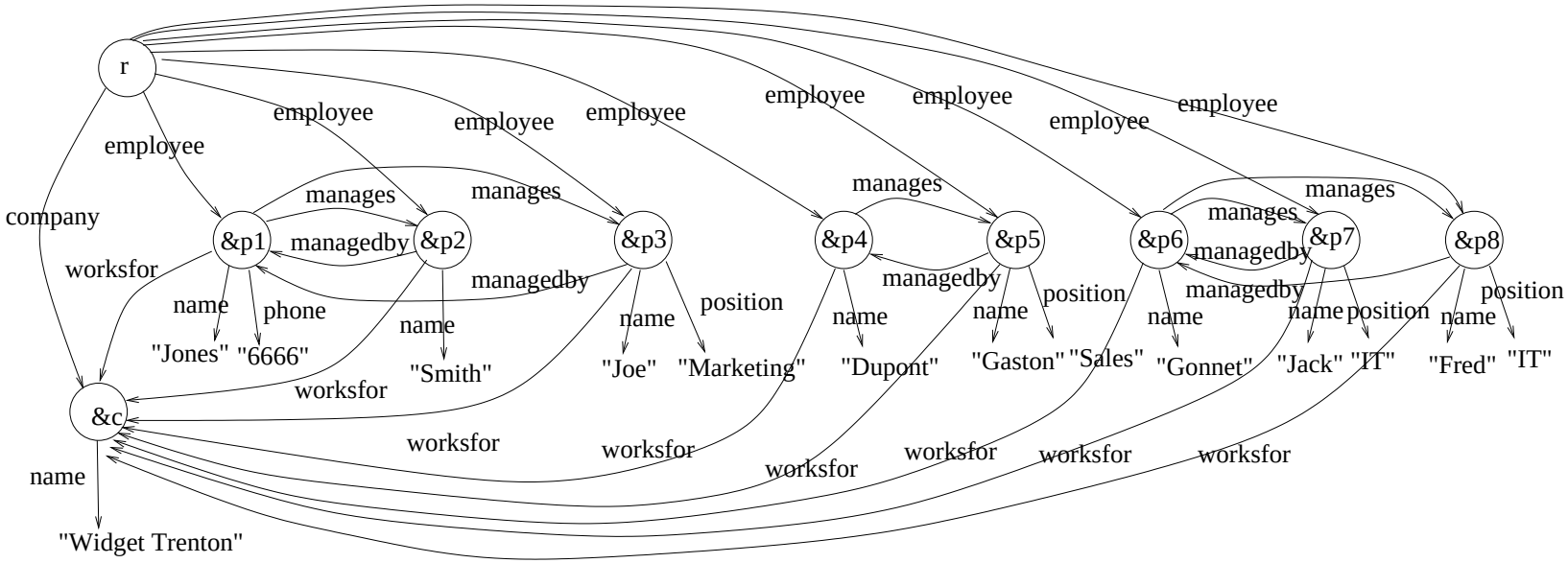
Schema Extraction from Data

Problem statement:

- given data instance D
- find the "most specific" schema S for D
- in practice: may be too large, need to relax (approximate, etc)

Schema Extraction

Example data:



Extracting Upper Bound

Data Guide [Goldman and Widom 1997]

Given data instance D:

- a data guide S for D is a graph s.t.:

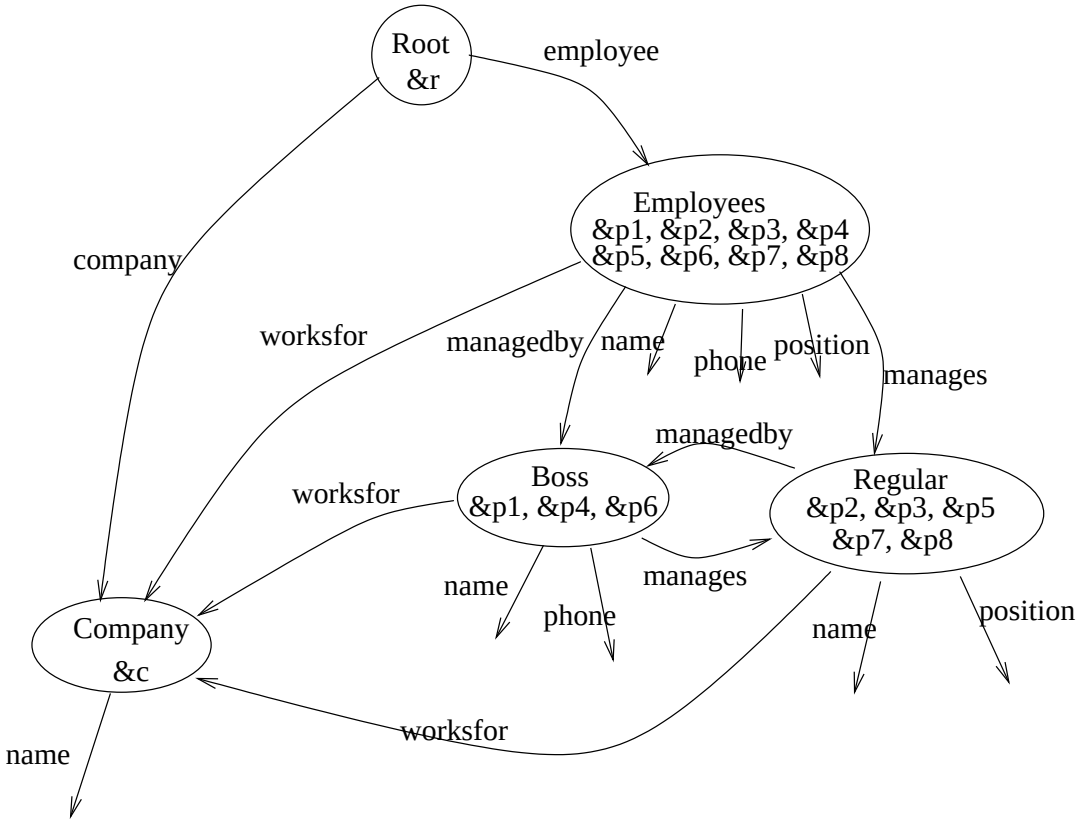
  accuracy D and S have the same sets of paths

  conciseness every path in S is unique

- construction of S: powerset automaton for D
- implemented in Lore

the data guide is precisely the most specific (deterministic) upper-bound schema

Data Guide

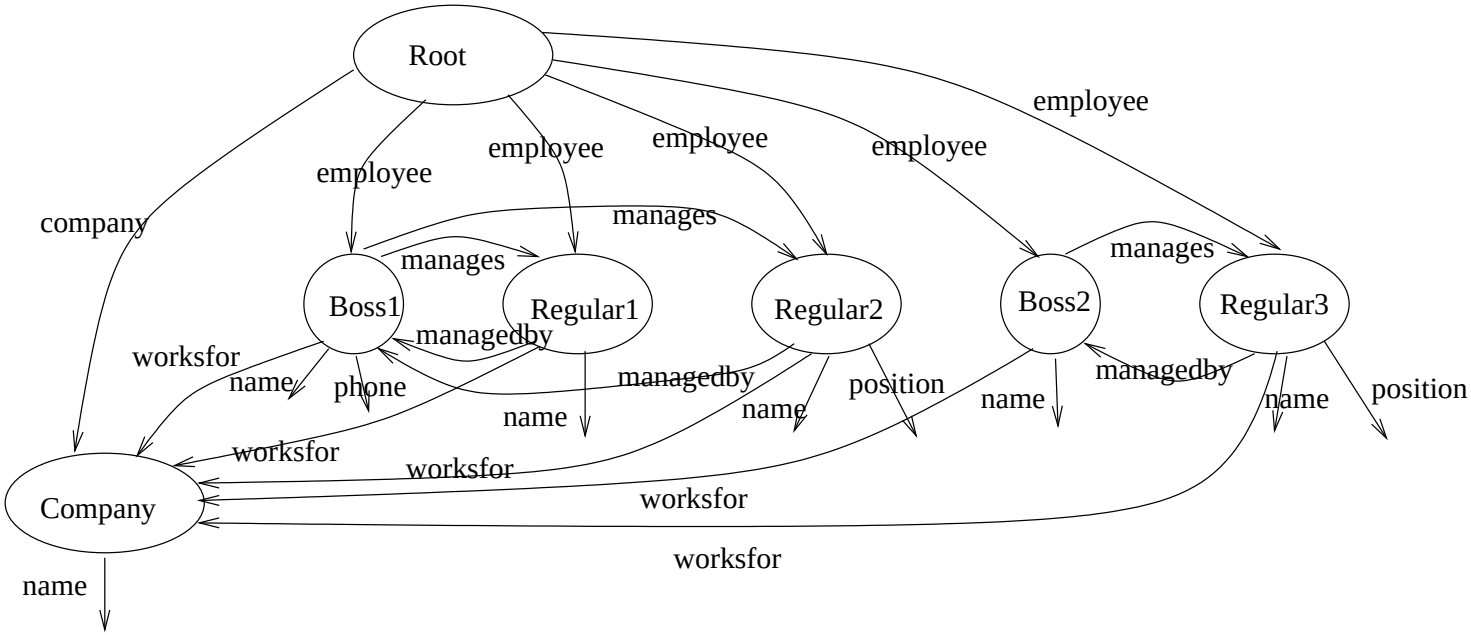


Data guides are *deterministic*

Extracting the Lower Bound

[Nestorov et al., 1998]

- construction:
  - build "most specific" datalog program: each node = a class
  - run the program
  - collapse equal classes
- alternatively: compute the simulation equivalence classes



Schema Inference

Problem statement:

- given query Q
- find most specific schema S s.t.
  - for every D, $Q(D)$ conforms to S
- related to *type inference* in functional programming languages

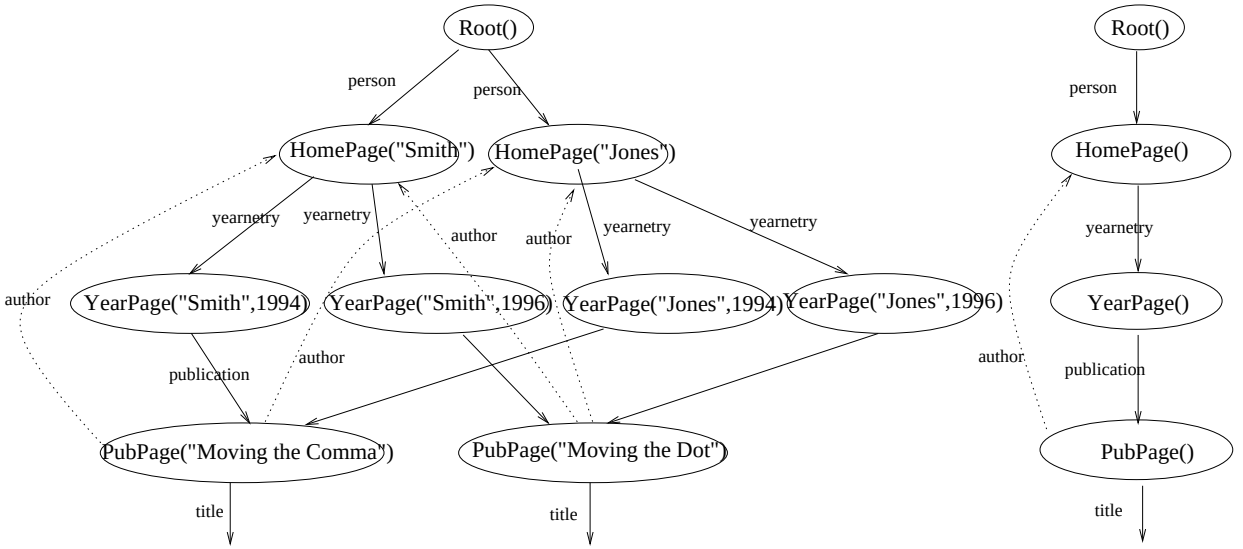
Schema Inference

where Root → "Bib" → X, X → "paper" → P, P → "author" → A,
P → "title" → T, P → "year" → Y

create Root(), HomePage(A), YearPage(A,Y), PubPage(P)

link Root() → "person" → HomePage(A),
HomePage(A) → "yearentry" → YearPage(A,Y),
YearPage(A,Y) → "publication" → PubPage(P),
PubPage(P) → "author" → HomePage(A)
PubPage(P) → "title" → T

Schema Inference



Some data instance Q(D)

Inferred schema

Schemas in XML

Document Type Descriptor (DTD)

- optionally accompanies an XML document
- specifies a *grammar*; has only limited use as a *schema*

DTD's as Grammars

```
<!DOCTYPE paper [
  <!ELEMENT paper   (section*)>
  <!ELEMENT section ((title,section*) | text)>
  <!ELEMENT title   (#PCDATA)>
  <!ELEMENT text    (#PCDATA)>
] >
```

```
<paper>  <section> <text> </text> </section>
         <section> <title> </title>  <section> . . . </section>
                                     <section> . . . </section>
         </section>
</paper>
```

DTD's as Schemas

Not so well suited:

- impose unwanted restrictions on order:

  <!ELEMENT person (name,phone)>

  means that <name> must occur before <phone>

- problems with references: cannot restrict their range
- can be too vague:

  <!ELEMENT person ((name|phone|email)*)>

  what combinations of attributes should one expect ?

Other Schemas Proposed for XML

Currently considered by the W3C (no clear winner):

- XML-Data
- RDF Schema
- DCD
- ...

Characteristics:

- object-oriented flavor (classes, subclasses)
- constraints

in all proposals the XML documents must make explicit reference to the schema

Summary of Schemas

- in SS data: data and schema are decoupled $\Rightarrow$ schema extraction from data
- in SS data: the emphasis is on using schema information in data processing (optimization, storage)
- in XML: schemas range from grammars (DTD) to object-oriented
- in XML: emphasis is on semantics

Part 4

Systems Issues

- Semistructured data and XML
- Query languages
- Schemas
- Systems issues
- Conclusions

Systems Issues: Outline

- servers for semistructured data
- mediators for semistructured data

Servers for Semistructured Data

- secondary storage:
  - text file (XML)
  - OO repository (need DTD): POQL [Christophides et al. 94], ObjectDesign's eXcelon, Bluestone, etc.
  - extend OO: Ozone
  - build object repository from scratch (Lore)
  - as ternary relation: [Florescu and Kossmann '99]
  - from data mining to relational schema [Deutsch et al. 99]
- indexes
  - deal with different atomic types: 1995, "1995" (Lore)
  - compute path expressions: Region Algebras (structured text), data guides, T-indexes, XSet
- query processing and optimization (Lore)

Mediator for Semistructured Data

- XML = a view over Relational/OO/OR data
- mediators:
  - translation between models, e.g. Relational $\leftrightarrow$ XML
  - integration
- issues:
  - query optimization in the presence of limited source capabilities (finding feasible plans) [Yerneni et al'99, Florescu et al'99]
  - query composition and rewriting [Papakonstantinou et al'96, Papakonstantinou&Vassalos'99, Deutsch et al'99, Calvanese et al'99]
  - data conversion [Cluet et al'98]

Part 5

Conclusions

- Semistructured data and XML
- Query languages
- Schemas
- Systems issues
- Conclusions

Summary

- data on the Web
  - at the intersection of semistructured data and structured text (XML)
  - highlights cultural mismatch: db community and document community
- paradigm shift, for both Web and db:
  - Web shifts from document to data
  - db shifts from structured to irregular, self-describing data
- covered by this tutorial: data models, queries, schemas
- not covered by the tutorial: transactions, limited source capabilities [Yerneni et al'99], lots of W3C standard proposals (XML-Pointer, XML-Link, RDF, RDF-Schema, DCD)

What Technologies are Needed

- Web applications possible today:
  - servers store/process data in traditional ways
  - applications exchange data in XML (and queries too !)
  - XML = view over structured databases

  needed: mediator technology, good compression (not there yet)
- Web applications in the future:
  - store/process native XML data
  - mine/analyze XML data

  needed: XML object repository, indexes, query processing/optimization, data mining

Why This Is Cool for Database Researchers

- put to work all those techniques you teach in CS101 !
  - recursive tree traversal (structural recursion, XSL)
  - automata theory (DTD's, path expressions)
  - graph theory (simulation, bisimulation, graph morphism)
- adapt old db techniques to handle the new kind of data
- make the world a better place (from FAX to XML) !

The End