Graph (Semi-structured) Data
The Data Model
Data Viewed As Graph

• Original intuition:
  – Entities (objects) are represented as nodes
  – Relationships are represented as edges
  – Therefore, nodes and edges have associated types, and attributes

• Many variations in circulation
  – Kind of edges?
    • Directed
    • undirected
  – Where is data?
    • Only on nodes
    • Only on edges
    • On both
  – Shape of graph?
    • Arbitrary (has cycles)
    • Directed Acyclic Graph
    • Tree
Node-labeled

Nodes are labeled with types (book, author, title) and/or data (strings)
Edge-labeled

book

author

"Abiteboul"

author

"Buneman"

author

"Suciu"

title

"Data on the Web"
Nodes labeled with Data, Edges with Type

- book
  - author: “Abiteboul”
  - author: “Buneman”
  - author: “Suciu”
  - title: “Data on the Web”
Graphs May Have Cycles
OEM (Object Exchange Model): a reference serialization format

bib: &1
  { paper: &2 { ... },
    book: &3 { ... },
    paper: &4
      { author: &10
        { firstname: &15 "Serge",
          lastname: &16 "Abiteboul" },
        author: &11 { ... }
      }
    title: &12 { ... }
    pages: &13
      { first: &17 122,
        last: &18 133 },
    references: &2,
    references: &3
  }
}
Advantages of graph data model:
• easy to discover new data and load it
• easy to integrate heterogeneous data
• easy to query without knowing data types

Disadvantages:
• loses type information
• lack of schema makes optimisation harder
Graph Schemas

• given some semi-structured data, might want to extract a schema that describes it
• useful for
  – browsing the data by types
  – optimizing queries by reducing the number of paths searched
  – improving storage of data
Schema Graph

• specifies schema as a graph itself

• schema graph specifies what edges are permitted in a data graph

• every path in the data graph occurs in the schema graph
Schema Graph Example

The diagram illustrates a schema graph with the following nodes and connections:

- bib
- book
- paper
- references
- title
- publisher
- author
- year
- pages
- firstname
- lastname
- first
- last

The graph shows the relationships between these nodes, indicating how data is structured and interconnected.
Data Graph Satisfying a Schema Graph

• given data graph $D$ and schema graph $S$

• $D$ is an instance of $S$ (or $D$ satisfies $S$) if there exists a simulation $R$ from $D$ to $S$ such that $(\text{root}(D), \text{root}(S))$ is in $R$

• a simulation is a relation $R$ between nodes:
  – if $(u, v)$ is in $R$ and $(u, x)$ labeled $l$ is in $D$
    then there exists $(v, y)$ labeled $l$ in $S$
    such that $(x, y)$ is in $R$
For Our Running Example

• node &1 in $D$ related under $R$ to node at target of $bib$ edge in $S$

• &2 and &4 related to node at target of $paper$ edge

• &3 related to node at target of $book$ edge

• note that above two cases need to satisfy requirements of edges labeled $references$ as well

• &10 and &11 related to node at target of edge labelled $author$

• ...

A Less Specific Schema Graph
Data Guide

data guide is a concise and accurate summary of a data graph
• **accurate**: every path in the data occurs in the data guide, and vice versa
• **concise**: every path in the data guide occurs exactly once

data guide is the *most specific* schema graph for a given data graph
• i.e., there is a simulation from the data guide to every other schema graph the data graph satisfies

connection to Finite State Automata:
• data graph is analogous to a *nondeterministic finite state automaton* (NFA)
• given NFA $N$, data guide is analogous to a *deterministic finite state automaton* (DFA) equivalent to NFA $N$
• conversion from NFA to DFA can result in exponential increase in size
Data Guide Example
Example Discussed

• provides a classification of nodes/objects in the data
• 2 and 4 are papers
• 5, 10 and 11 are authors of papers
• 2 and 3 are referenced by papers
• 6 and 8 are titles of objects referenced by papers
• 3 is both a book and an object that is referenced by an object that is referenced by a paper
Representative Query Paradigms
Philosophy: Patterns

• Inspired by relational QBE formalism

• Basic pattern: matches against a single edge

  \_s \rightarrow E \rightarrow \_t

\_s,\_t: node variables, E: edge type label

• Example:
  find endpoints of paths crossing an E edge, then an F edge

  Q1(\_s,\_t) \leftarrow \_s \rightarrow E \rightarrow \_x, \_x \rightarrow F \rightarrow \_t
Find nodes with an outgoing D edge whose target has an outgoing E edge and an outgoing F edge. Return source node and the two target nodes.

Q2(_s,_e,_f) :- _s − D -> _x, _x − E -> _e, _x − F -> _f

Note use of variable _x to match against the fork node
Regular Path Patterns: Concatenation

- When intermediate nodes on path do not need to be named:

\[ Q1(\_s,\_t) :\_s \rightarrow E \rightarrow _x, _x \rightarrow F \rightarrow _t \]

Expression \( E.F \) expresses edge concatenation:

\[ Q1'(\_s,\_t) :\_s \rightarrow E.F \rightarrow _t \]
Regular Path Patterns: Disjunction

• Find endpoints of E or F edges:

\[ Q3(_s,_t) :\_s \rightarrow E \lor F \rightarrow _t \]

• Find endpoints of paths starting with an E edge, followed by an F or G edge, then by an H edge:

\[ Q4(_s,_t) :\_s \rightarrow E.(F \lor G).H \rightarrow _t \]
Regular Path Patterns: Wildcard

- Find endpoints of all edges, regardless of edge type

Q5(_s,_t) :- _s - _ -_ > _t

- Endpoints of paths of length 3

Q6(_s,_t) :- _s - __.__._ -_ > _t

- Nodes involved in self-loops

Q7(_n) :- _n - _ -_ > _n
Regular Path Patterns: Kleene Star

- $p^*$
  Specifies arbitrarily many (including 0) repetitions of path pattern $p$

- Pairs of connected nodes
  
  $Q8(_s,_t) : - _s - _* - > _t$

- Pairs of nodes connected only by red or blue edges
  
  $Q9(_s,_t) : - _s - (\text{red} | \text{blue})^* - > _t$

- Connected by an alternation of red and blue edges, starting with red
  
  $Q10 (_s,_t) : - _s - (\text{red.blue})^* - > _t$
Regular Path Patterns: Syntax

- $p := T$
  - | `_`
  - | $p$ `.` $p$
  - | $p$ `|` $p$
  - | $p$ `*`
  - | `(` $p$ `)`

where $T$ is any edge type label
Node Construction

• So far, queries have only extracted sets of tuples of nodes

• What if we wish to construct new nodes?

• Need to specify
  – Node identity
  – New edges connecting newly created nodes
Generating Node Identities: Skolem functions

• A Skolem Function associates with its arguments the identity of a node.
• When called for the first time with a certain argument, the function creates a fresh node and returns its identity.
• Subsequent calls with the same argument return the identity of the previously created node.
Representative QL: StruQL

• For each author of a referenced book, create a “citedAuthor” edge, emanating from a fresh “Result” node

    from bibGraph _r

    where _r – bib.book -> _b,
        _b – author -> _a,
        _a – lastname -> _l,
        _a – ssn -> _s,
        _r -_*.*.references -> _b

    create Result (), Aut (_s), LN (_s, _l)

    link Result () – CitedAuthor -> Aut (_s),
        Aut (_s) – LastName -> LN (_s, _l)