Advanced XML / Data on the Web
Lecture 5

Lars Birkedal [birkedal@it-c.dk]

The IT University of Copenhagen

Outline of this lecture

- Types I
  - Motivation
  - Datalog rules and relations (min and max fixed points)
  - Schema Graphs
- Readings:
  - ABS Sections 7.1–7.5
  - Nestorov et. al.: “Extracting Schema from Semistructured Data”
  - Optional: for more rigorous treatment of Datalog, see, e.g., Abiteboul, Hull, Vianu: Foundations of Databases, Addison-Wesley.

Typing Semistructured Data

- Fairly new and controversial field
- Active research area
- Types may be specified *after* the database is populated
- Thus: type inference or schema extraction is central

Motivations

- To optimize query evaluation
- To facilitate the task of integrating several data sources
- To improve storage
- To construct indexes
- To describe the database content to users and facilitate query formulation (*data guides*)
- To proscribe certain updates

Analyzing the problem

Assume we know what a type is. Then basic questions are, as usual:

- Does the database conform to this type ?
- Which objects belong to each class ?

However, situation here different from standard typing for object databases:

- less clear definition of classes, so objects may belong several classes
- some objects may not belong to any class
- typing may be approximate (will be ignored in the sequel)

Schema Formalisms

Overview:

- We will use the ssd model (edge-labelled graphs)
- We will consider two approaches:
  1. Datalog rules (fragment of first-order logic) to describe data, fixed point semantics
  2. Graphs to describe data, simulation semantics

Datalog

Definition A *(datalog) rule* is an expression of the form

$$R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n)$$

where $n \geq 1$, R_1, . . . , R_n are relation names, and u_1, . . . , u_n are tuples of appropriate arities. Each variable occuring in u_1 must occur in at least one of u_2, . . . , u_n. A *datalog program* is a finite set of datalog rules.

- The *head* of the rule is $R_1(u_1)$
- The *body* of the rule is $R_2(u_2), \ldots, R_n(u_n)$

Datalog

Definition Let v be a valuation (a function from variables to constants). An *instantiation* of

$$R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n)$$

is an expression

$$R_1(v(u_1)) \leftarrow R_2(v(u_2)), \ldots, R_n(v(u_n))$$

(where each variables x in the rule is replaced by $v(x)$).

Datalog

Let P be a datalog program.

- An *extensional relation* is a relation ocurring only in the body of the rules
- An *intensional relation* is a relation ocurring in the head of some rule
- The extensional (database) schema, denoted $edb(P)$ consists of all the extensional relation names
- The intensional (database) schema, denoted $idb(P)$ consists of all the intensional relation names.
- The schema $sch(P) = edb(P) \cup idb(P)$
- The semantics of P is a mapping from database instances over $edb(P)$ to database instances over $idb(P)$.

Datalog Example

```
r(X) :- ref(X,person,Y), p(Y), ref(X,company,Z),
        c(Z)
p(X) :- c(Y), ref(Y,manager,X), c(Z),
        ref(Z,employee,X), ref(X,worksfor,U),
        c(U), ref(X,name,N), string(N),
        ref(X,position,P), string(P)
c(X) :- p(Z), ref(Z,worksfor, X), p(Z),
        ref(Z,worksfor,X), ref(X,manager,M), p(M
        ref(X,employee,E), p(E),
        ref(X,name,N), string(N),
        ref(X,address,A), string(A)
```

(root, person, company)

- Intensional: `r`, `p`, `c`
- Extensional: `ref`, `string`,

Datalog Example

Note:

- Idea: the datalog program describes ssd graphs with root nodes, person nodes, and company nodes having edges as described by the datalog program
- So, datalog serves here as a *schema formalism*

Fixed Point Semantics

Let P be a datalog program and K an instance over $sch(P)$. A fact A is an *immediate consequence* for K and P if either $A \in K(R)$ for some edb relation R, or $A \leftarrow A_1 \ldots, A_n$ is an instantiation of a rule in P and each A_i is in K. The *immediate consequence operator* $T_P : inst(sch(P)) \rightarrow inst(sch(P))$ is defined by: $T_P(K)$ is the set of all facts A that are immediate consequences for K and P.

Thus T_P is an operator (function) between sets of instances.

Fixed Point Semantics

Let T be an operator. Then recall that

- T is *monotone* if, for each I, J, if $I \subseteq J$ then $T(I) \subseteq T(J)$.
- K is a fixed point of T if $T(K) = K$.
- A monotone operator has a least and a maximal fixed point (Knaster-Tarski).

Lemma Let P be a datalog program.

- The operator T_P is monotone.

Min Fixed Point Semantics

Standard datalog interpreation: minimal fixed point.
Let P be a datalog program and I an instance over $edb(P)$ (think of I as the input data graph).
Then

- $minfix(I) = I \cup T_P(I) \cup T_P^2(I) \cup T_P^3(I) \cup \ldots$ is the minimal fixed point containing I.

Doesn't work for us: every rule in P has at least one intensional predicate, so for all $R \in idb(P)$, $T_P(R) = \emptyset$, so we don't classify any data.

In the example, $T_P(\mathsf{r}) = T_P(\mathsf{c}) = T_P(\mathsf{p}) = \emptyset$.

Max Fixed Point Semantics

Instead we use maximal fixed point.
Let P be a datalog program and I an instance over $sch(P)$, such that, for each $R \in idb(P)$, $I(R)(o)$ always holds. (The input data graphs is represented by I on the extensional predicates). Then

- $maxfix(I) = I \cap T_P(I) \cap T_P^2(I) \cap T_P^3(I) \cap \ldots$ is the maximal fixed point

Example

Input data as ssd expression

```
&o1 {company: &o2{name: &o5 "O2",
                  address: &o6 "Versailles",
                  manager: &o3,
                  employee: &o3,
                  employee: &o4},
     person:  &o3{name: &o7 "Francois",
                  position: &o8 "CEO",
                  worksfor: &o2},
     person:  &o4{name: &o9 "Lucien",
                  position: &o10 "Programmer",
                  worksfor: &o2}
    }
```

Example

Input data graph represented as extensional relations:

```
ref(&o1,company,&o2), ref(&o2,name,&o5),...
string(&o5), ...
```

Denote this set of facts by D. Let $I_0 = I$ be the initial instance (defined as above).

$$
\begin{aligned}
I_0 = D \quad & \cup \quad \{\texttt{r(\&o1)}, \texttt{r(\&o1)}, \texttt{r(\&o2)}, \texttt{r(\&o3)}, \texttt{p(\&o1)}\} \\
& \cup \quad \{\texttt{p(\&o2)p(\&o3)}, \texttt{p(\&o4)}, \texttt{c(\&o1)}\} \\
& \cup \quad \{\texttt{c(\&o2)}, \texttt{c(\&o3)}, \texttt{c(\&o4)}\}
\end{aligned}
$$

Example

Then

$$maxfix(I) = D \cup \{\texttt{r(\&o1)}, \texttt{p(\&o3)}, \texttt{p(\&o4)}, \texttt{c(\&o2)}\}$$

So we managed to type all objects.

Break

20 minutes

Schema Graphs

Idea: use a graph to describe schema and use *simulation* to answer the key questions:

- Conformance: does the data conform to the schema ?
- Classification: if so, which objects belong to what classes ?

Graph Simulation

Recall from Lecture 1:
Definition Let G_1 and G_2 be two edge-labelled graphs. A **simulation** is a relation R between the nodes such that

$$\forall (x_1, x_2) \in R. \forall (x_1, a, y_1) \in G_1. \exists (x_2, a, y_2) \in G_2. (y_1, y_2) \in R$$

Example

On the board

Schema Graph

Definition A schema graph is a graph such that

- nodes are called classes
- edges are labelled with unary predicates, $p(x)$

Examples of unary predicates

- person — $\text{person}(x)$
- $\text{name} \mid \text{address}$ — $\text{name}(x) \vee \text{address}(x)$
- $*$ — true
- int — $x \in Z$
- $\neg\text{name}$ — $\neg(\text{name}(x))$

Using Simulation

Given data graph D and schema graph S

- conformance: find maximal simulation R from D to S, notation: $D \leq S$.
- classification: check if (x, c) in R, notation: $x \leq c$.

Examples of Schema Graphs

On the board.

Note: schema graphs describe those edges that are allowed.

Using Simulation

Any data graph conforms to the “universal schema graph” (with one node and one looping edge labelled true”).

Schemas in SS data vs. relational data:

- relational data:
  - each data instance has exactly one schema
- semistructured data:
  - each data instance has several schemas

The classification problem

- Schema is nondeterministic: creates ambiguous classifications.
- Example on board.
- **Definition** A schema S is *deterministic* if for every class c and every label a, there is at most one outgoing edge labelled a from c.
- **Fact** If S is deterministic and D is a tree, then each node is uniquely classified. (when D is not a tree, then it is not true).

Deterministic Schemas

- Given a schema S, we can always construct a deterministic approximation S_d.
- In general, S_d is obtained by the powerset construction (as also used for transforming NFAs to DFAs), so computationally expensive.

Review of Schemas so far

Datalog programs:

- define a class by saying what incoming and outgoing edges are *required*

Schema graphs:

- upper bound schema: tells us what labels are allowed

Datalog vs. schema graphs

- To compare, restrict datalog programs to check only outgoing edges.
- Then the datalog program correspons to a graph, called the *dual schema graph*

```
r(X) :- ref(X,person,Y), p(Y),
        ref(X,company,Z), c(Z)
p(X) :- ref(X,worksfor,U), c(U),
        ref(X,name,N), string(N),
        ref(X,position,P), string(P)
c(X) :- ref(X,manager,M), p(M),
        ref(X,employee,E), p(E),
        ref(X,name,N), string(N),
        ref(X,address,A), string(A)
```

Datalog vs. schema graphs

- Conformance testing using dual schema graph S: $S \leq D$
- Conformance testing with schema graph S: $D \leq S$

Schema Extraction

Problem:

- given data graph D
- find the "most specific" schema S for S

In practice: S is too large, need to relax.

Schema Extraction

Lower Bound Schema Extraction:

- Compute the maximal simulation $D \leq D$
- Two nodes p and q are *equivalent* iff $p \leq q$ and $q \leq p$
- Schema consists of equivalence classes.

Remark: see book for alternative equivalent presentations.