Querying Composite Objects in Semistructured Data

Keishi Tajima

Department of Computer and Systems Engineering
Kobe University
Japan

partly supported by

Ministry of Education Grant-in-Aid: "Advanced Databases"

Japan Society for the Promotion of Science: "Advanced Multimedia Contents Processing"

Background (1/4)

Semistructured Data

Semistructured data is:

- schemaless, and
- self-describing

data used for:

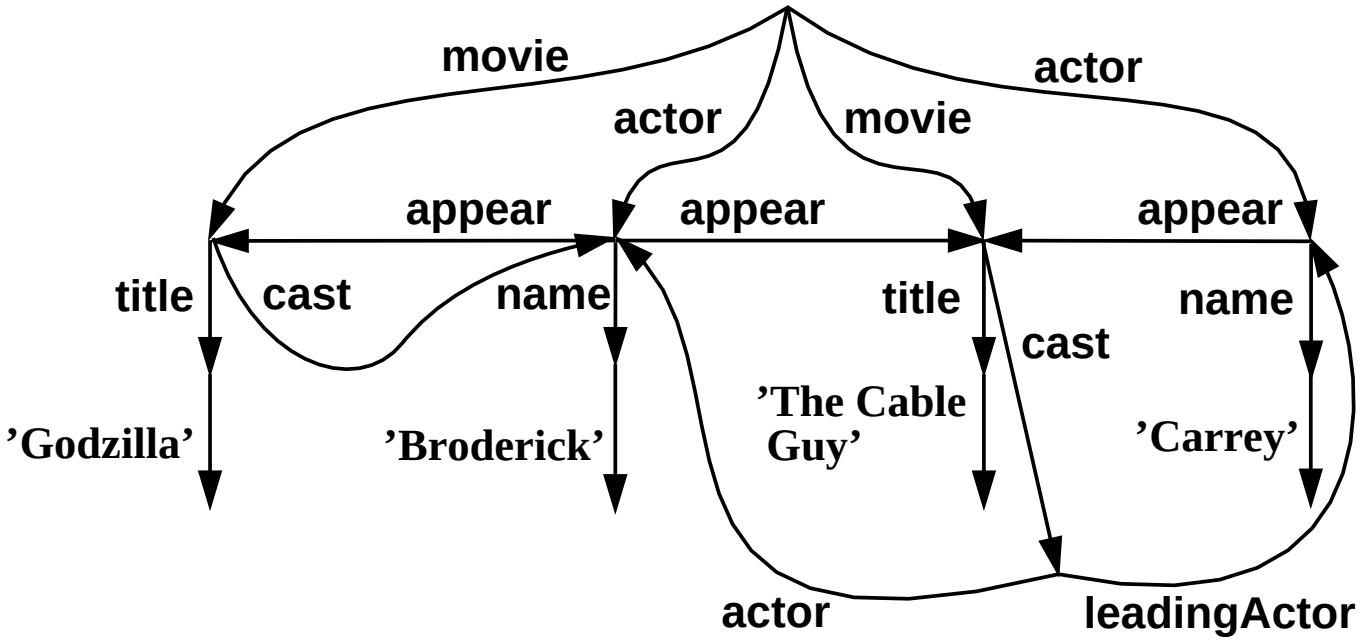
- heterogeneous data integration,
- Web DB,
- etc...

Background (2/4)

Data Models for Semistructured Data

Most researches have proposed edge-labeled graphs.

e.g.: edge-labeled tree model [Buneman *et al.* 96]



Attribute names are described in the data itself.

Background (3/4)

Query Languages for Semistructured Data

Most researches proposed path expressions with wild cards.

e.g.: UnQL [Buneman *et al.* 96]

- list the titles of all the movies:

  select l
  where movie⇒title⇒ $\backslash l \Rightarrow \{\}$

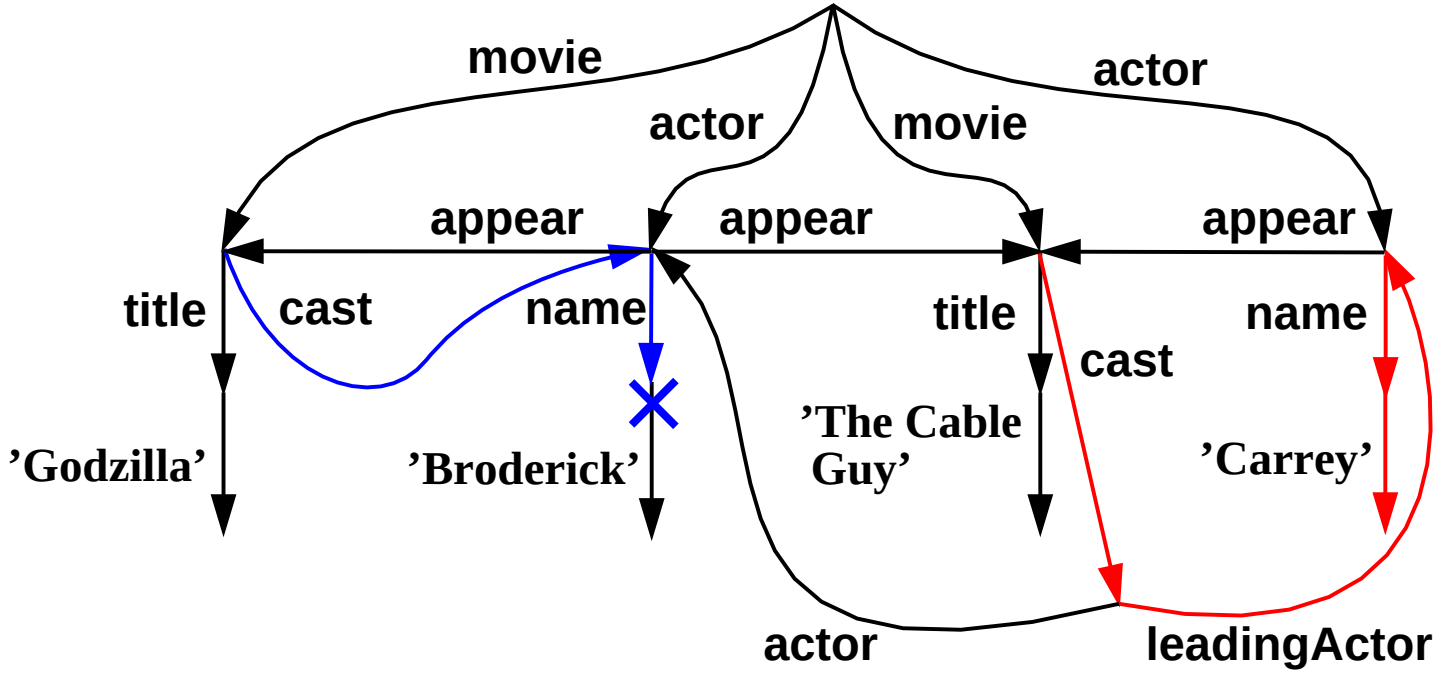
- list the titles of the movies featuring an actor named "Carrey":

  select l
  where movie⇒ {title⇒ $\backslash l \Rightarrow \{\}$,
  cast⇒[_⇒]*name⇒'Carrey'⇒ {}}

Background (5/5)

Wild cards are used to deal with heterogenity

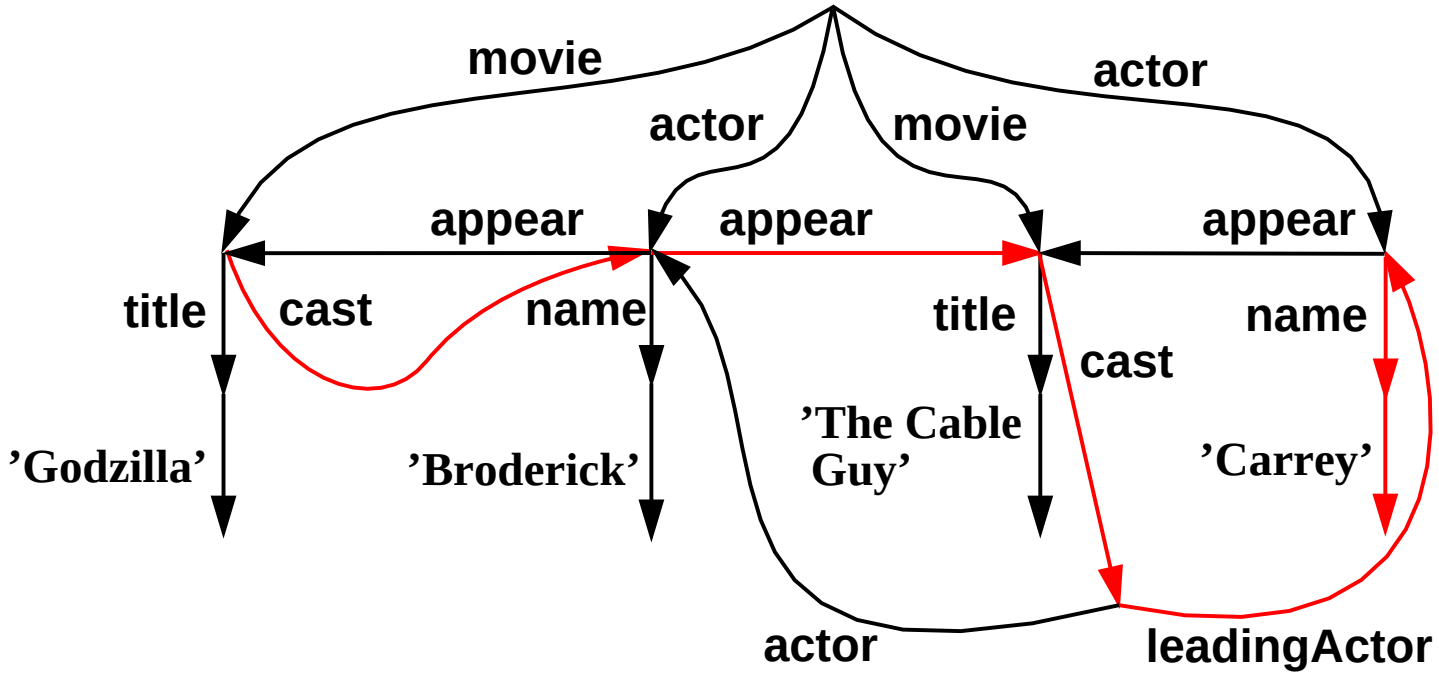
cast⇒[_⇒]*name⇒'Carrey'⇒ {}



Problem

Wild cards [_⇒]* often matches with unexpected paths

cast⇒[_⇒]*name⇒'Carrey'⇒ {}



The previous query answers {'Godzilla', 'Tha Cable Guy'}.

Analysis of the Problem (1/2)

Wild cards are ...

- One solution is to explicitly exclude all "unexpected" cases.

```
select l
where movie⇒ {title⇒ \l ⇒ {},
              cast⇒[^appear⇒]*name⇒'Carrey'⇒ {}}
```

But,

- when wild cards reach everywhere in a graph, it is difficult to anticipate all the "unexpected" cases.

Wild cards for paths of arbitrary length are essentially dangerous.

Analysis of the Problem (1/2)

What we actually need is ...

- The example database consists of entities (movies and actors) and references between them.
- Our intention was to skip the heterogeneous structure within the movie entries, and not to reach to everywhere in the graph.
- The previous example query, which are entity-conscious, are the most typical queries on semistructured data.

In many cases, we want wild cards to match in more restricted way, especially in entity-conscious way.

Goal of this Research

Based on two observations:

- wild cards are dangerous, and
- we usually need more restricted entity-conscious use,

our goal is to design a framework for enabling:

wild cards that are entity-conscious and easier to use.

What To Do

We need to do two things:

- to develop a method of dividing semistructured data into entities, and
- to design query language constructs for wild cards with restricted, entity-conscious matching.

Detection of Entities in Semistructured Data (1/3)

Analogy to *composite objects* in OODBs

Composite objects in OODBs [Kim *et al.* 89] are:

- representing independent real-world entities, and
- rooted trees consisting of only composite links

where composite links are:

- exclusive references representing is-part-of relationship.

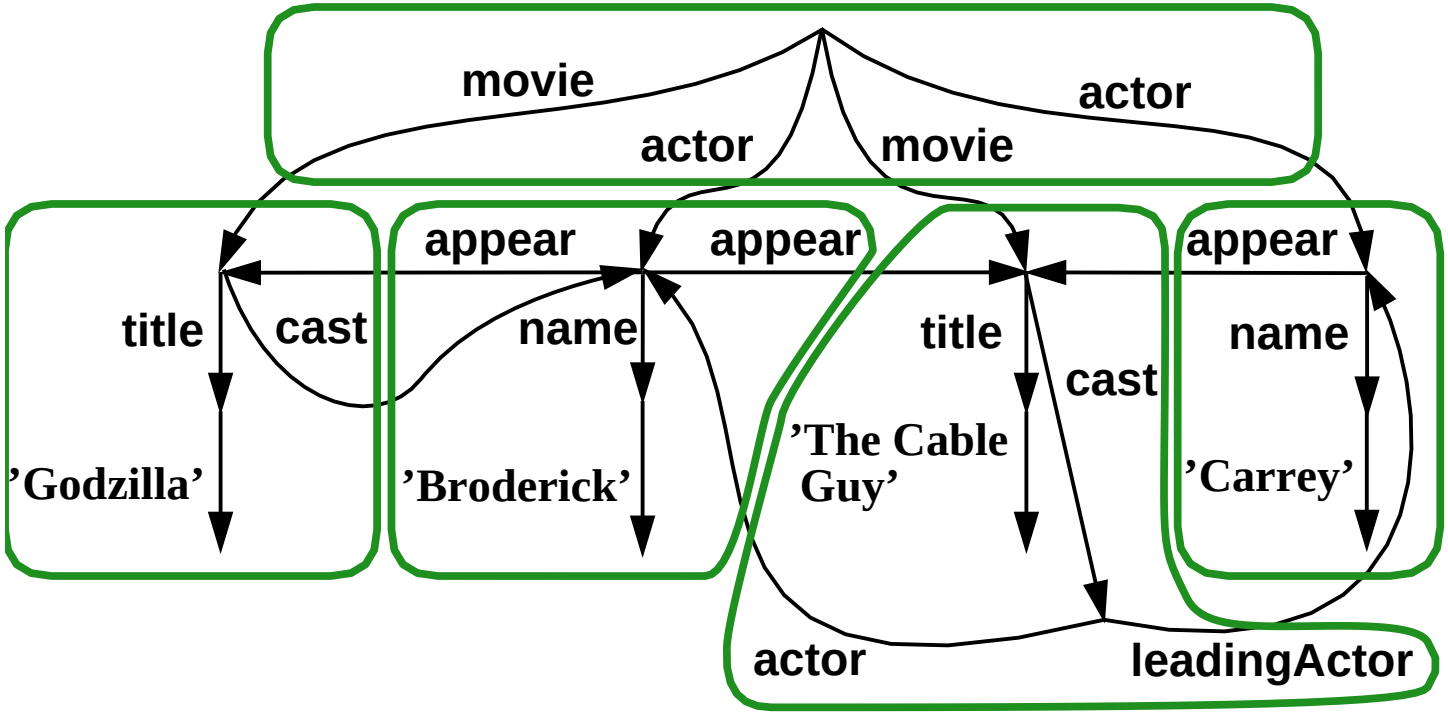
By the analogy to these, we consider entities in semistructured data are:

- connected subgraphs consisting of only exclusive references.

Detection of Entities in Semistructured Data (2/3)

Example of Entity Detection

This method works well for the previous example data.

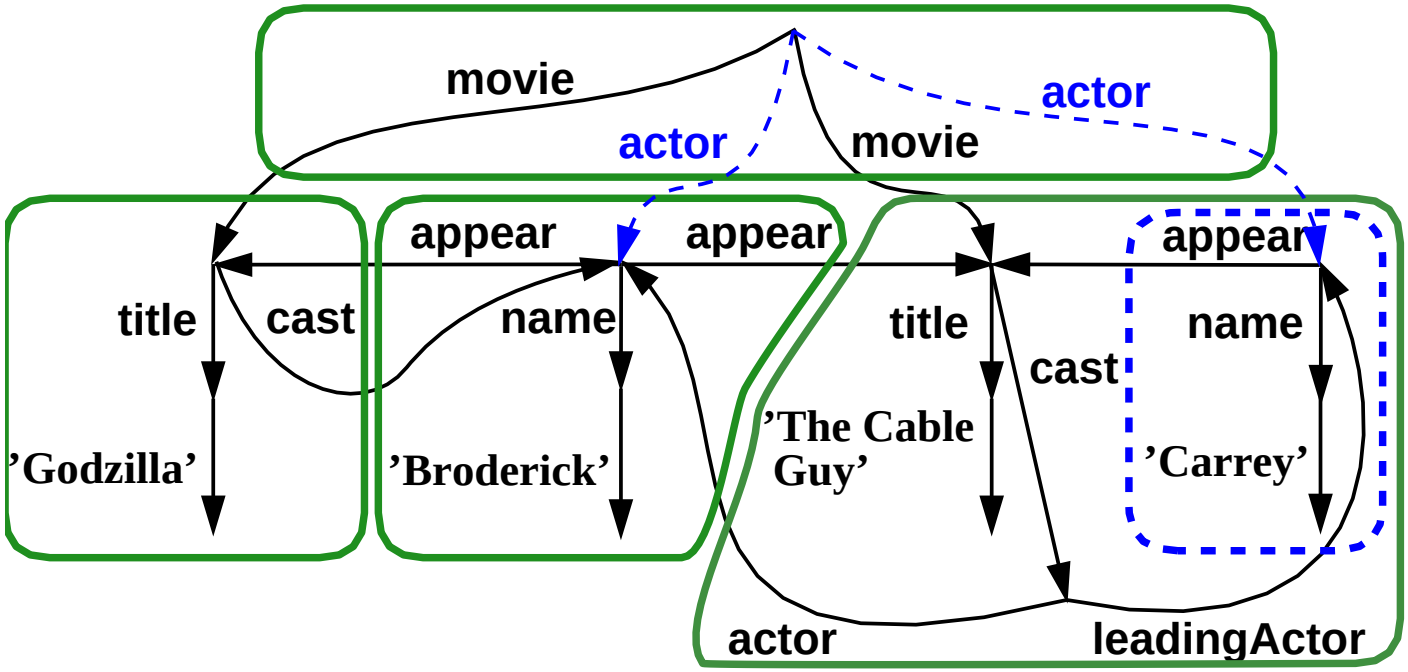


5 composite objects (2 movies, 2 actors, and the root)

Detection of Entities in Semistructured Data (2/3)

Example of Entity Detection

Our method is not always perfect.



Approximate typing of nodes [Nestorov *et al.* 98] is useful to improve the correctness.

Constructs to Restricted Wild Cards (1/3)

Distinguishing Composite Links and Non-composite Links

1. $\langle l \Rightarrow \rangle^{\circ}$ matches only with composite links.
   - retrieve the value of **age** attribute at someplace in the entry of "Carrey."

   select l
   where actor⇒{name⇒'Carrey'⇒{},
   [⟨_⇒⟩°]*age⇒ \l ⇒ {}}

2. $\langle l \Rightarrow \rangle^{\times}$ matches only with non-composite links.

Constructs to Restricted Wild Cards (2/3)

Using these constructs, the query "the titles of all movies featuring an actor Carrey" is rewritten as follows:

$$
\begin{array}{l}
\mathsf{select}\ l \\
\mathsf{where\ movie}\Rightarrow \{\mathsf{title}\Rightarrow \backslash l \Rightarrow \{\},\\
\qquad\qquad [\quad \langle \mathsf{cast}\Rightarrow\rangle^{\times}[\langle \_\Rightarrow\rangle^{\circ}]^{*} \quad | \\
\qquad\qquad\quad \langle \mathsf{cast}\Rightarrow\rangle^{\circ}[\langle \_\Rightarrow\rangle^{\circ}]^{*}\langle \_\Rightarrow\rangle^{\times}[\langle \_\Rightarrow\rangle^{\circ}]^{*} \quad] \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{name}\Rightarrow \text{'Carrey'}\Rightarrow \{\}\}
\end{array}
$$

- for the cases where **cast** is composite link
- for the cases where **cast** is non-composite link

Constructs to Restricted Wild Cards (3/3)

A macro expression for more concise description

- $\langle \mathit{path\ pattern} \rangle^{\circ n_1}_{\times n_2}$ — which matches only with paths including n_1 composite links and n_2 non-composite links.

With this macro, the previous query is rewritten as follows:

select l
where movie⇒ {title⇒ $\backslash l \Rightarrow \{\}$,
$\langle$cast⇒[_⇒]*$\rangle^{\circ}_{\times 1}$name⇒'Carrey'⇒ {}}

Entity-Based Style Queries (1/2)

A query style taking full advantage of our constructs

- retrieve the value of **year** of the movie "Godzilla":

$$\begin{array}{l}\textsf{select } l\\ \textsf{where } [\_\Rightarrow]^*\{[\langle\_\Rightarrow\rangle^\circ]^*\text{'Godzilla'}\Rightarrow\{\},\\ \qquad\qquad [\langle\_\Rightarrow\rangle^\circ]^*\textsf{year}\Rightarrow\backslash l\Rightarrow\{\}\}\end{array}$$

- list the name of the movies featuring the actor "Carrey":

$$\begin{array}{l}\textsf{select } l\\ \textsf{where } [\_\Rightarrow]^*\{[\langle\_\Rightarrow\rangle^\circ]^*\textsf{title}\Rightarrow\backslash l\Rightarrow\{\},\\ \qquad\qquad [\langle\_\Rightarrow\rangle^\circ]^*\langle\textsf{cast}\Rightarrow[\_\Rightarrow]^*\rangle^\circ_{\times 1}\textsf{name}\Rightarrow\text{'Carrey'}\Rightarrow\{\}\}\end{array}$$

They assume as least knowledge on the schema as possible.

Entity-Based Style Queries (1/2)

A macro expression for entity-based style queries

$$E\{t_1, \ldots, t_n\} \quad \equiv \quad [\_\Rightarrow]^*\{[\langle\_\Rightarrow\rangle\circ]^* t_1, \ldots, [\langle\_\Rightarrow\rangle\circ]^* t_n\}$$

- list the name of the movies featuring the actor "Carrey":

  select l
  where $E\{$'Godzilla', year$\Rightarrow \backslash l \Rightarrow \{\}\} \leftarrow$ DB

- list the name of the movies featuring the actor "Carrey":

  select l
  where $E\{$title$\Rightarrow \backslash l \Rightarrow \{\}$,
  $\langle$cast$\Rightarrow[_\Rightarrow]^*\rangle^{\circ}_{\times 1}$name$\Rightarrow$'Carrey'$\Rightarrow \{\}\}$

Conclusion

- We develop a method of detecting composite objects, i.e. entities in semistructured data.
- We design constructs for restricted matching of edges. They make wild cards easier to use.

Being able to use wild cards more often implies that we can specify queries with less knowledge.

Future Work

- Better method of entity detection.