Completing Queries: Rewriting of Incomplete Web Queries Under Schema Constraints

Sacha Berger[1] François Bry[1] Tim Furche[1]
Andreas J. Häusler[2]

[1]Chair for Programming and Modelling Languages, Institute for Informatics
University of Munich

[2]TUM CogBotLab, Computer Science Department
Technical University of Munich
(work done while at [1])

The First International Conference on Web Reasoning and Rule Systems

Completing Queries: Rewriting of Incomplete Web Queries

Completing Queries: Rewriting of Incomplete Web Queries Under Schema Constraints

Sacha Berger[1] François Bry[1] Tim Furche[1]
Andreas J. Häusler[2]

[1]Chair for Programming and Modelling Languages, Institute for Informatics
University of Munich

[2]TUM CogBotLab, Computer Science Department
Technical University of Munich
(work done while at [1])

The First International Conference on Web Reasoning and Rule Systems

- Incompleteness distinguishes web query languages from languages like SQL
- However, incompleteness leads to more expensive evaluation of the queries

We propose a set of equivalence rules which exploit schema information for automatic rewriting of graph-shaped web queries on graph-shaped semi-structure data. These equivalences allow the introduction or removal of all three forms of incompleteness that may be contained in the query.

Properties of web queries and schemata

- Querying the Web: Incompleteness is needed

$\Rightarrow$ Incompleteness is expensive

- (Incomplete) Queries can be optimized using schema information

$\Rightarrow$ Static optimization by rewriting

- Semi-structured data does not neccessarily need a matching schema
- Incomplete queries are needed
- This results in low evaluation performance
- Queries might be optimized using schema information
- The query author should not have to care about performance
- Optimization has to be done automatically

- The perceived strength and main contribution of the semi-structured data model is the ability to model data with little or no (a priori) information on the data's schema.
- Modern query languages languages are distinguished from previous relational query languages in providing core constructs for expressing incomplete queries, i.e., queries where only some of the sought-for data is specified but that are not affected by the presence of additional data.
- Examples are regular path expressions or the `descendant` and `following` closure axis in XPath and XQuery
- Incomplete query constructs have proved to be both essential tools for expressing Web queries and a great convenience for query authors able to focus better on the parts of the query he or she is most interested in.

- However, most approaches can handle only certain incomplete queries efficiently, and suffer from lower performance for evaluating incomplete queries.
- We also believe, that such rewritings might help the query programmer to understand the precise meaning of an incomplete query against a certain schema. Expanding incompleteness (at least on request) might help the author to decide whether the restrictions on the data specified in the query suffice to obtain what she is interested in.
- However, asking the query author to take such limitations of the evaluation engine into consideration would abandon the convenience and declarativity achieved by incomplete query constructs.
- However, though Web query languages are designed to also operate without schema information, in practice some schema information is usually available.

Types of incompleteness

Three types of incompleteness: depth, breadth and order

Example query in Xcerpt

```
paper [[
    desc var Author -> author {
        name{{ }},
        email{{ }}
    }
]]
```

Types of incompleteness

Three types of incompleteness: depth, breadth and order

Example query in Xcerpt

```
paper [[
    desc var Author -> author {
        name{{ }},
        email{{ }}
    }
]]
```

Types of incompleteness

Three types of incompleteness: depth, breadth and order

Example query in Xcerpt

```
paper [[
    desc var Author -> author {
        name{{ }},
        email{{ }}
    }
]]
```

Types of incompleteness

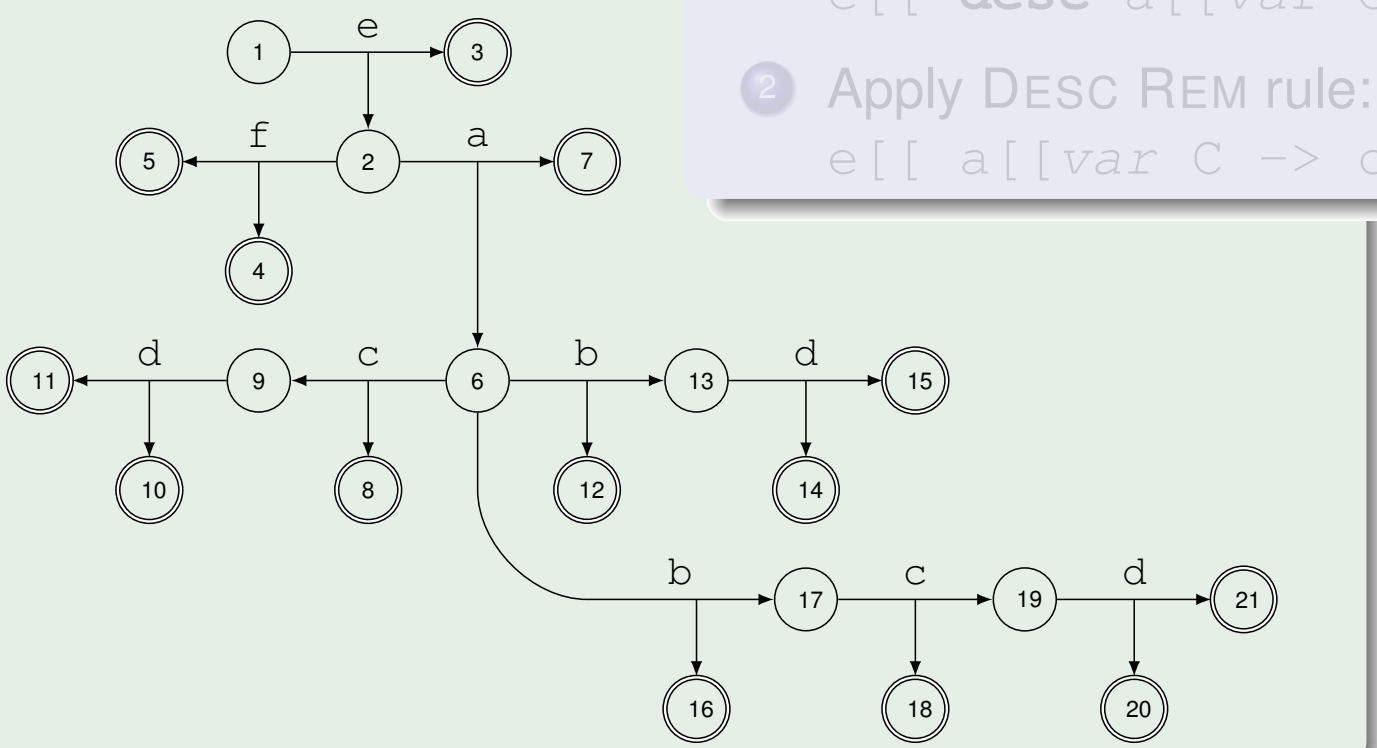
Three types of incompleteness: depth, breadth and order

Example query in Xcerpt

```
paper [[
    desc var Author -> author {
        name{{ }},
        email{{ }}
    }
]]
```

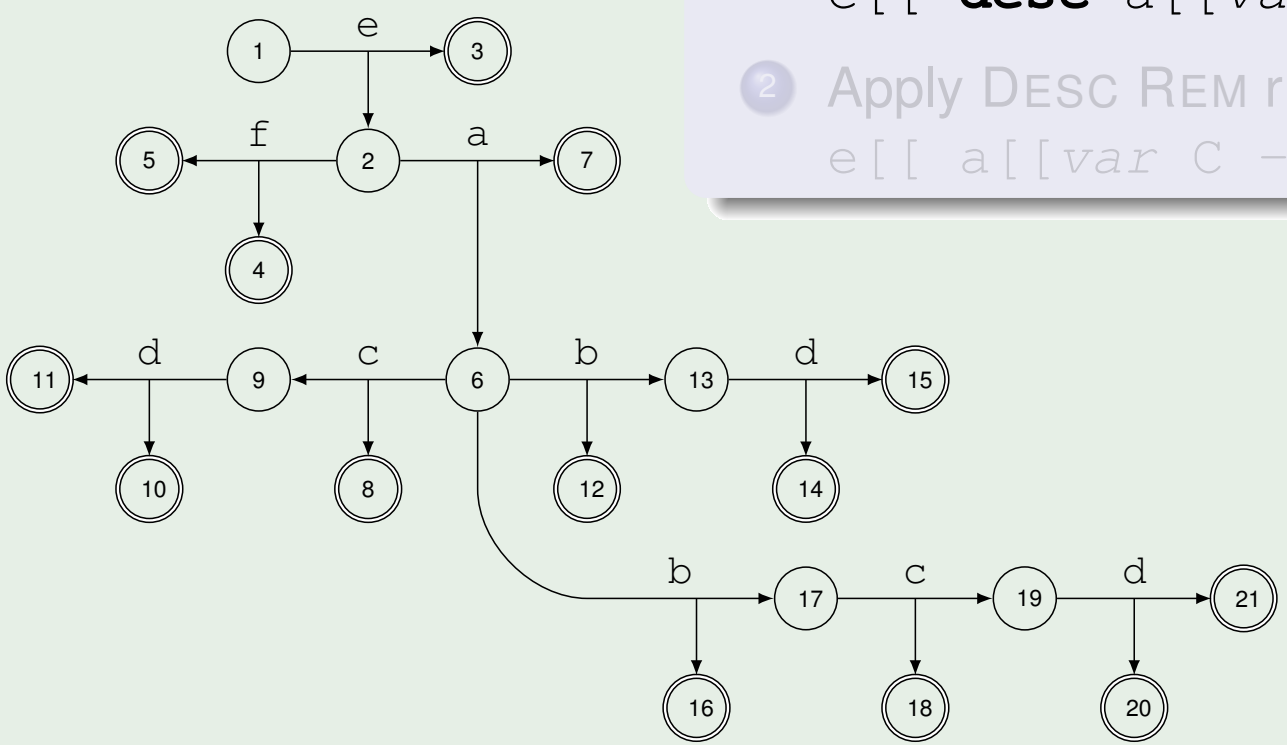
Incompleteness in depth

0. Original query:
   desc `a[[var C -> c]]`
1. Apply DESC EXP rule:
   `e[[ desc a[[var C -> c]] ]]`
2. Apply DESC REM rule:
   `e[[ a[[var C -> c ]] ]]`



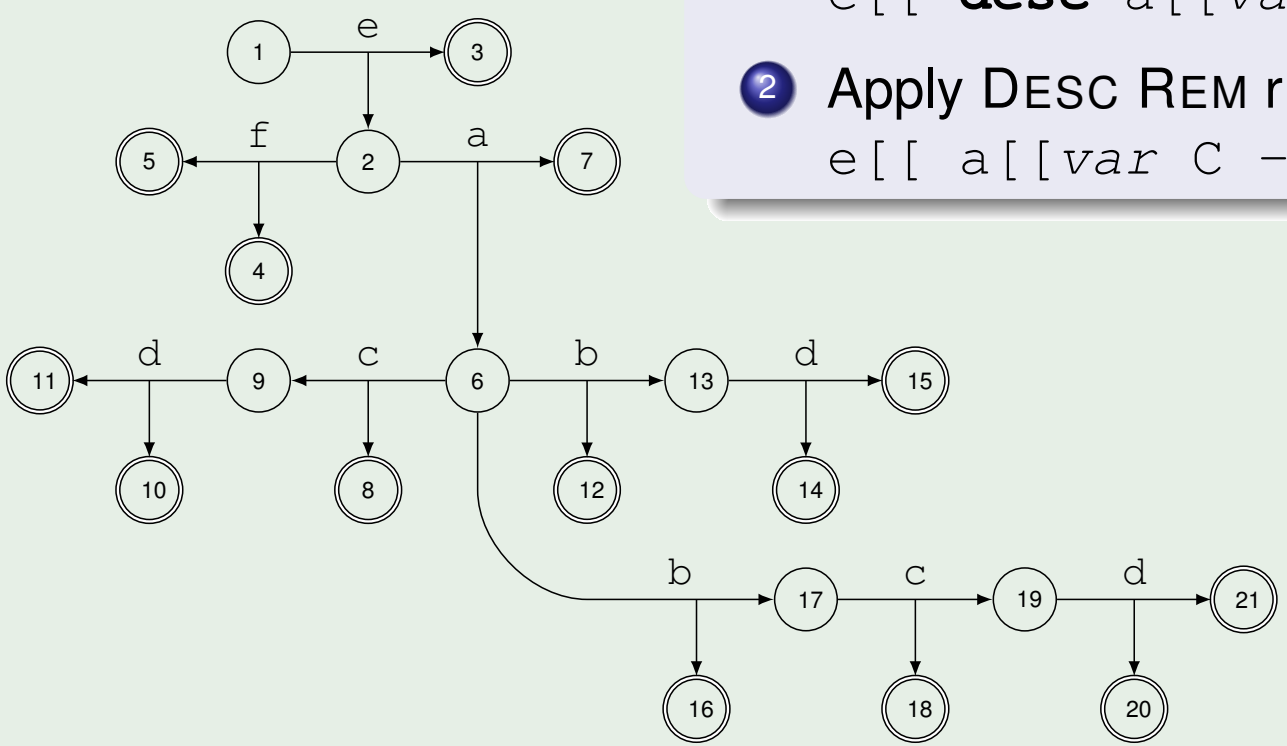
Incompleteness in depth

0. Original query:
   desc a[[*var* C -> c]]
1. Apply DESC EXP rule:
   e[[**desc** a[[*var* C -> c]]]]
2. Apply DESC REM rule:
   e[[a[[*var* C -> c]]]]



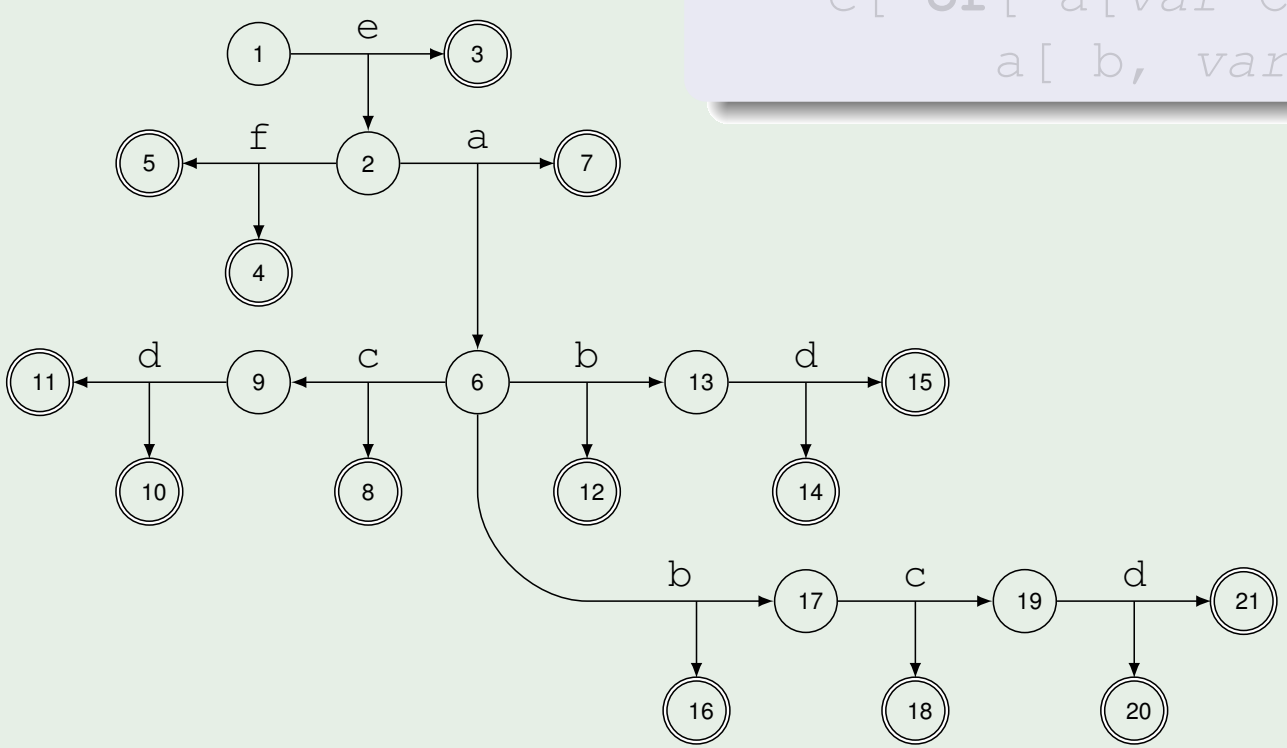
Incompleteness in depth

0. Original query:
   desc a[[*var* C -> c]]
1. Apply DESC EXP rule:
   e[[**desc** a[[*var* C -> c]]]]
2. Apply DESC REM rule:
   e[[a[[*var* C -> c]]]]



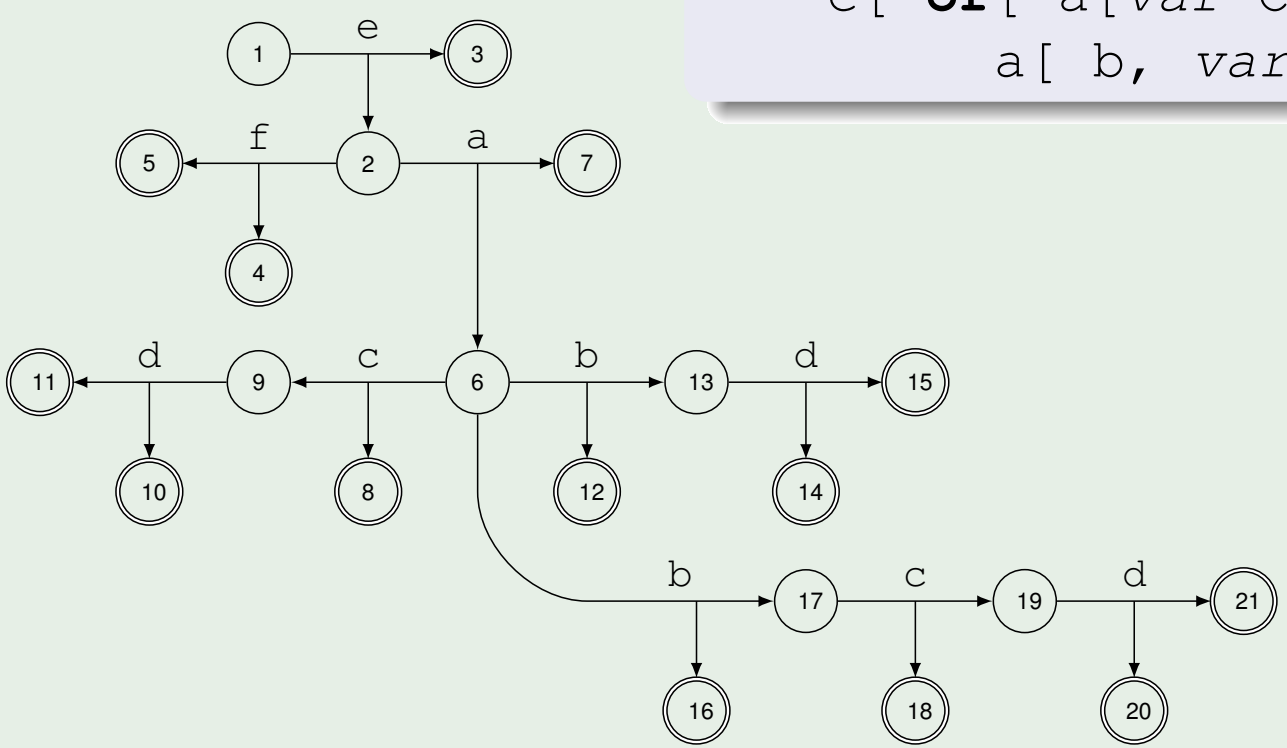
Incompleteness in breadth

3. Apply PARTIAL SPEC EXP rule:
   `e[ a[[var C -> c]] ]`
4. Second application of this rule:
   `e[ or[ a[var C -> c, d],`
   `a[ b, var C -> c, d] ] ]`



Incompleteness in breadth

3. Apply PARTIAL SPEC EXP rule:
   `e[ a[[var C -> c]] ]`
4. Second application of this rule:
   `e[ or[ a[var C -> c, d],`
   `a[ b, var C -> c, d] ] ]`



Overview of Incompleteness Reductiuon Rules

$$\frac{(\texttt{desc}\ \ t:\tau):(s,\texttt{l},c,e)}{\texttt{l[[(desc}\ \ t:\tau):(s',\texttt{l}',c',e')\texttt{]]}:(s,\texttt{l},c,e)}\ (\textsc{Desc1})$$

$$\frac{\texttt{var X}:\tau}{\texttt{var X ->}\ \bigstar:\tau}\ (\textsc{Var})$$

$$\frac{(\texttt{desc}\ \ t:\tau):(s,\texttt{l},c,e)}{t:(s,\texttt{l},c,e)}\ (\textsc{Desc2})$$

$$\frac{\bigstar:(s,\texttt{l},c,e)}{\texttt{l [[ ]]}:(s,\texttt{l},c,e)}\ (\textsc{Star})$$

$$\frac{\texttt{l[[}t_1,t_2,\ldots,t_q\texttt{]]}:(s,\texttt{l},c,e)}{\texttt{or}\left[\begin{array}{c}\ldots\\ \texttt{l[}map(\bigstar,z_1),t_1,map(\bigstar,z_2),t_2,\ldots,map(\bigstar,z_q),t_q,map(\bigstar,z_{q+1})\texttt{]}\\ \ldots\end{array}\right]}$$

$$z_1,\ldots,z_{q+1}\in hpts(c,s_{t_1})\times hpts(e_{t_1},s_{t_2})\times\ldots\times hpts(e_{t_{q-1}},s_{t_q})\times hptes(e_{t_q})\ (\textsc{Partial})$$

Outlook & Conclusion

We have defined equivalences for incompleteness removal

Forthcoming issues

- Next step: Experimental evaluation
- Then: Define optimization heuristics

Outlook & Conclusion

We have defined equivalences for incompleteness removal

Forthcoming issues

- Next step: Experimental evaluation
- Then: Define optimization heuristics

Novelties

- No normalization required
- Not focused on regular path expressions only

$\Rightarrow$ Heuristical query optimization using graph schemata

Outlook & Conclusion

We have defined equivalences for incompleteness removal

Forthcoming issues
- Next step: Experimental evaluation
- Then: Define optimization heuristics

Novelties
- No normalization required
- Not focused on regular path expressions only
- ⇒ Heuristical query optimization using graph schemata

- The presented equivalences need to be integrated into an necessarily heuristic optimization algorithm
- An implementation of these rules has to be provided with certain heuristics
  (Heuristics are required to define a cut-off point which, when reached, defines that a certain query expansion is no longer useful.)
- Combination and integration with other forms of query optimization and rewriting for web queries has to be considered
- Experimentally verify the improvement to query evaluation that can be obtained through applying our rules