INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians
Universität
München
LMU

Pattern Queries for XML and Semistructured Data

François Bry and Sebastian Schaffert

Technical Report, Computer Science Institute, Munich, Germany
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-2002-5, March 2002

Pattern Queries for XML and Semistructured Data

François Bry and Sebastian Schaffert

Institute for Computer Science, University of Munich
http://www.pms.informatik.uni-muenchen.de/

1 Introduction

Essential to semistructured data is the selection of data from incompletely specified data items. For such a data selection, a path language such as XPath [1] is convenient because it provides with regular expressions such as $*$, $+$, ?, and "wildcards" that give rise to a flexible node retrieval.

Query and transformation languages developed since the mid 90es for XML [1] and semistructured data – e.g. XQuery [1], the precursors of XQuery [], and XSLT [1] – rely upon such a path-oriented selection. They use patterns (also called templates) for expressing how the selected data, expressed by paths, are re-arranged (or re-constructed) into new data items. Thus, such languages intertwine construct parts, i.e. the construction patterns, and query parts, i.e. path selectors. This intertwining has some advantages: For simple query-construct requests, the approach is rather natural and results in an easily understandable code. However, intertwining construct and query parts also has drawbacks: 1. Query-construct requests involving a complex data retrieval might be confusing, 2. unnecessarily complex path selections, e.g. XPath expressions involving both forward and reverse axes, are possible [2], 3. in case of several path selections, the overall structure of the retrieved data items might be difficult to grasp.

This paper addresses using patterns instead of paths for querying XML and semistructured data. A metaphor for this approach is to see queries as forms, answers as form fillings yielding database items. With this approach, patterns are used not only in construct expressions, but also for data selection.

In the following, a basic query language is introduced. An answer to a query in this language is formalized as a simulation [3] of a ground instance of the query in a database item. This formalization yields a compositional semantics.

2 A Basic Query Language

The following principles have prevailed to the definition of the query language:
Pattern-based or positional instead of navigational. A query should correspond to a form, an answer to a filling yielding a database item. The relative positions of variables in a query should be easily recognizable.
Referential transparency. The meaning of an expression, especially of a variable, should be the same wherever it appears. Therefore, destructive assignments are prohibited and variables must be functional or logic programming variables.
Compositional semantics. A (structurally) recursive definition of the semantics of a query in terms of the semantics of its parts, i.e. a Tarski-style model theory, is sought for.

Multiple variable bindings. Like with SQL and other query languages, queries might have several answers, each answer binding the query variables differently. *Symmetry.* Queries should allow similar forms of incomplete specifications in breadth, i.e. concerning siblings, and in depth, i.e. concerning children.

Note that the requirements of [4] are fulfilled by or compatible with the basic query language defined below.

Below, the following pairwise disjoint sets of symbols are referred to: A set $\mathcal{I}$ of identifiers, a set $\mathcal{L}$ of labels (or tags or strings), a set $\mathcal{V}_l$ of label variables, a set $\mathcal{V}_t$ of term (or data item) variables. Identifiers are denoted by id, labels (variables, resp.) by lower (upper, resp.) case letters with or without indices. The following meta-variables (with or without indices and/or superscripts) are used: *id* denotes an identifier, l denotes a label, L a label variable, X a term variable, t a term (as defined below), v a label or a term, and V a label or term variable.

A database is a set (or multiset) of database terms. The children of a document node may be either ordered (as in standard XML), or unordered. In the following, a term whose root is labelled l and has *ordered* (*unordered*, resp.) children $t_1, \ldots, t_n$ is denoted $l[t_1, \ldots, t_n]$ ($l\{t_1, \ldots, t_n\}$, resp.).

Definition 1 (Database Terms). Database terms *are expressions inductively defined as follows and satisfying Conditions 1 and 2 given below:*

1. *If l is a label, then l is a (atomic) database term.*
2. *If id is an identifier and t is a database term neither of the form $id_0\colon t_0$ nor of the form $\uparrow id_0$, then id$\colon t$ is a database term.*
3. *If id is an identifier, then $\uparrow$id is a database term.*
4. *If l is a label and $t_1, \ldots, t_n$ are $n \geq 1$ database terms, then $l[t_1, \ldots, t_n]$ and $l\{t_1, \ldots, t_n\}$ are database terms.*

Condition 1: *For a given identifier id an* identifier definition *id$\colon t_0$ occurs at most once in a term.*
Condition 2: *For every* identifier reference *$\uparrow$id occurring in a term t an identifier definition id$\colon t_0$ occurs in t.*

A query term is a pattern that specifies a selection of database terms very much like logical atoms and SQL selections. The evaluation of query terms (cf. below Definition 9) differs from the evaluation of logical atoms and SQL selections as follows: 1. Answers might have additional subterms to those mentioned in the query term. 2. Answers might have another subterm ordering than the query. 3. A query term might specify subterms at an unspecified depth.

In query terms, the double square and curly brackets, [[ ]] and {{ }}, denote exact subterm patterns, i.e. double (square or curly) brackets are used in a query term to be answered by database terms with no more subterms than those given in the query term. [[ ]] is used if the subterm order in the answers is to be that of the query term, {{ }} is used otherwise. Thus, possible answers to the query term $t_1 = a[[b, c\{d, e\}, f]]$ are the database terms $a[b, c\{d, e, g\}, f]$ and $a[b, c\{d, e, g\}, f\{g, h\}]$ and $a[b, c\{d, e\{g, h\}, g\}, f\{g, h\}]$ and $a[b, c[d, e], f]$. In contrast, $a\{b, c\{d, e\}, f, g\}$ and $a[b, c\{d, e\}, f, g]$ and $a\{b, c\{d, e\}, f\}$ are no answers to t_1.

In a query term, a term variable X can be constrained to some query terms using the construct $\rightsquigarrow$, read "as". Thus, the query term $t_2 = a[X_1 \rightsquigarrow b[c,d], X_2, e]$ constrains the term variable X_1 to such database terms that are possible answers to the query term $b[c,d]$. Note that the term variable X_2 is unconstrained in t_2. Possible answers to t_2 are $a[b[c,d], f, e]$ which binds X_1 to $b[c,d]$ and X_2 to f, $a[b[c,d], f[g,h], e]$ which binds X_1 to $b[c,d]$ and X_2 to $f[g,h]$, $a[b[c,d,e], f, e]$ which binds X_1 to $b[c,d,e]$ and X_2 to f, and $a[b[c,e,d], f, e]$ which binds X_1 to $b[c,e,d]$ and X_2 to f. In query terms, the construct *desc*, read "descendant", specifies a subterm at an unspecified depth. Thus, possible answers to the query term $t_3 = a[X \rightsquigarrow \mathit{desc}\ f[c,d], b]$ are $a[f[c,d], b]$ and $a[g[f[c,d]], b]$ and $a[g[f[c,d], h], b]$ and $a[g[g[f[c,d]]], b]$ and $a[g[g[f[c,d], h], i], b]$.

Definition 2 (Query Terms). Query terms *are expressions inductively defined as follows and satisfying Conditions 1 and 2 of Definition 1:*

1. *If l is a label and L is a label variable, then l, L, $l\{\}$, and $L\{\}$ are (atomic) query terms.*
2. *A term variable is a query term.*
3. *If id is an identifier and t is a query term neither of the form $id_0\colon t_0$ nor of the form $\uparrow id_0$, then $id\colon t$ is a query term.*
4. *If id is an identifier, then $\uparrow id$ is a query term.*
5. *If X is a variable and t a query term, then $X \rightsquigarrow t$ is a query term.*
6. *If X is a variable and t is a query term, then $X \rightsquigarrow \mathit{desc}\ t$ is a query term.*
7. *If l is a label, L a label variable and $t_1, \ldots, t_n$ are $n \geq 1$ query terms, then $l[t_1, \ldots, t_n]$, $L[t_1, \ldots, t_n]$, $l\{t_1, \ldots, t_n\}$, $L\{t_1, \ldots, t_n\}$, $l[[t_1, \ldots, t_n]]$, $L[[t_1, \ldots, t_n]]$, $l\{\{t_1, \ldots, t_n\}\}$, and $L\{\{t_1, \ldots, t_n\}\}$ are query terms.*

A query term in which no variables occur is ground. *Query terms that are not of the form $\uparrow id$, are* strict. Leftmost labels *of strict ground query terms are defined as follows: For l, $l\{\}$, $l[t_1, \ldots, t_n]$, and $l\{t_1, \ldots, t_n\}$ it is l; for $id\colon t$ it is that of t; and for $\mathit{desc}\ t$ it is $\mathit{desc}\ l$ if l is the leftmost label of t.*

Database terms are (simple kinds of) query terms. However, the set of answers to a database term t (considered as a query term) in a database D in general contains not only t (cf. below Definition 9). E.g. the database terms f and $f\{a\}$ and $f\{b\}$ are possible answers to the query f. However, f is the only possible answer to the query term $f\{\}$. In a query term, multiple occurrences of a same term variable are not precluded. E.g. a possible answer to the query term $a\{X \rightsquigarrow b\{c\}, X \rightsquigarrow b\{d\}\}$ is $a\{b\{c,d\}\}$. The query term $a[X \rightsquigarrow b\{c\}, X \rightsquigarrow f\{d\}]$, however, has no answers, for labels b and f are distinct. *Child subterms* and *subterms* of query terms are defined such that if $t = f[a, g\{Y \rightsquigarrow \mathit{desc}\ b\{X\}, h\{a, X \rightsquigarrow k\{c\}\}]$, then a and $g\{Y \rightsquigarrow \mathit{desc}\ b\{X\}, h\{a, X \rightsquigarrow k\{c\}\}$ are the only child subterms of t and e.g. a and X and $Y \rightsquigarrow \mathit{desc}\ b\{X\}$ and $h\{a, X \rightsquigarrow k\{c\}\}$ and $X \rightsquigarrow k\{c\}$ and t itself are subterms of t. Note that f is not a subterm of t.

Definition 3 (Variable Well-Formed Query Terms). *A term variable X* depends *on a term variable Y in a query term t if $X \rightsquigarrow t_1$ is a subterm of t and Y is a subterm of t_1. A query term t is* variable well-formed *if t contains*

no term variables $X_0, \ldots, X_n$ $(n \geq 1)$ *such that 1.* $X_0 = X_n$ *and 2. for all* $i = 1, \ldots, n$, X_i *depends on* X_{i-1} *in* t.

E.g. $f\{X \rightsquigarrow g\{X\}\}$ and $f\{X \rightsquigarrow g\{Y\}, Y \rightsquigarrow h\{X\}\}$ are not variable well-formed. Variable well-formedness precludes queries specifying infinite answers. In the following, query terms are assumed to be variable well-formed.

3 Query Semantics

The semantics is based on graph simulation. The graphs considered are directed, ordered and rooted and their nodes are labelled.

Definition 4 (Simulation). *Let* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *be two graphs. Let* $\sim$ *be an equivalence relation on* $V_1 \cup V_2$. *A relation* $\mathcal{S} \subseteq V_1 \times V_2$ *is a* simulation *with respect to* $\sim$ *of* G_1 *in* G_2 *if:*

1. *If* $(v_1, v_2) \in \mathcal{S}$, *then* $v_1 \sim v_2$.
2. *If* $(v_1, v_2) \in \mathcal{S}$ *and* $(v_1, v_1') \in E_1$, *then there exists* $v_2' \in V_2$ *such that* $(v_1', v_2') \in \mathcal{S}$ *and* $(v_2, v_2') \in E_2$.

Let $\mathcal{S}$ *be simulation* $\mathcal{S}$ *of* $G_1 = (V_1, E_1)$ *in* $G_2 = (V_2, E_2)$. $\mathcal{S}$ *is* total, *if for each* $v_1 \in V_1$ *there exists at least one* $v_2 \in V_2$ *such that* $(v_1, v_2) \in \mathcal{S}$. *If* G_1 *has a root* r_1, G_2 *has a root* r_2 *and* $(r_1, r_2) \in \mathcal{S}$, *then* $\mathcal{S}$ *is a* rooted simulation. $\mathcal{S}$ *is* minimal, *if there are no simulations* $\mathcal{S}' \subseteq \mathcal{S}$ *of* G_1 *in* G_2 *such that* $\mathcal{S}' \neq \mathcal{S}$.

Note that every rooted simulation is total.

Definition 5 (Graph Induced by a Ground Query Term). *Let* t *be a ground query term. The* graph $G_t = (N_t, V_t)$ induced by t *is defined by:*

1. N_t *is the set of strict subterms (cf. Definition 2) of* t *and each* $t' \in N_t$ *is labelled with the leftmost label (cf. Definition 2) of* t'.
2. V_t *is the set of pairs* (t_1, t_2) *such that either* t_2 *is a child subterm of* t_1, *or* $\uparrow id$ *is a child subterm of* t_1 *and the identifier definition* $id\colon t_2$ *occurs in* t.
3. *The children of a node are ordered in* G_t *like in* t.

Note that t is the root of G_t. Figure 1 in Appendix illustrates Definition 5. Below, a database term is often identified with the graph it induces.

Definition 6 (Ground Query Term Simulation). $\preceq$ *is the relation on ground query terms defined by* $t^1 \preceq t^2$ *if there exists a (minimal) rooted simulation with respect to label identity* $\mathcal{S}$ *of* t^1 *in* t^2 *such that:*

1. *if* $v_1 = l\{\{\}\}$ *occurs in* t^1 *and* $(v_1, v_2) \in \mathcal{S}$, *then* v_2 *has no children in* t^2.
2. *if* $v_1 = l[t_1^1, \ldots, t_n^1]$ $(n \geq 1)$ *occurs in* t^1, $(v_1, v_2) \in \mathcal{S}$ *and if* $(t_i^1, t_j^2) \in \mathcal{S}$ $(1 \leq j \leq m \leq n)$, *then* $t_1^2, \ldots, t_m^2$ *occur in this order as children of* v_2 *in the graph induced by* t^2.
3. *if* $v_1 = l[[t_1^1, \ldots, t_n^1]]$ $(n \geq 1)$ *occurs in* t^1, $(v_1, v_2) \in \mathcal{S}$ *and if* $(t_i^1, t_j^2) \in \mathcal{S}$ $(1 \leq j \leq m \leq n)$, *then the* t_j^2 *are pairwise distinct (i.e.* $m = n$*) and occur in this order as children of* v_2 *in the graph induced by* t^2.

4. *if* $v_1 = \{\{t_1^1, \ldots, t_n^1\}\}$ *occurs in* t^1, $(v_1, v_2) \in \mathcal{S}$ *and* $(t_i^1, t_j^2) \in \mathcal{S}$ $(1 \leq j \leq m \leq n)$, *then* v_2 *has no other children than the* t_i^2 *in* t^2.

Figure 2 in Appendix illustrates Definition 6. By Definition 4, $\preceq$ is reflexive and transitive, i.e. it is a preorder on the set of database terms. $\preceq$ is not a partial order, for although $t_1 = f\{a\} \preceq t_2 = f\{a, a\}$ and $t_2 = f\{a, a\} \preceq t_1 = f\{a\}$ (both a of t_2 can be simulated by the same a of t_1), $t_1 = f\{a\} \neq t_2 = f\{a, a\}$.

Rooted simulation with respect to label equality is a first notion towards a formalisation of answers to query terms: If there exists a rooted simulation of a database term t_1, considered as a query term, in a database term t_2, then t_2 is an answer to t_1. An answer in a database D to a query term t^q is characterised by bindings for the variables in t^q such that the database term t resulting from applying these bindings to t^q is an element of D. Consider e.g. the query $t^q = f\{X \rightsquigarrow g\{b\}, X \rightsquigarrow g\{c\}\}$ against the database $D = \{f\{g\{a, b, c\}, g\{a, b, c\}, h\},\ f\{g\{b\}, g\{c\}\}\}$. The $\rightsquigarrow$ constructs in t^q yield the constraint $g\{b\} \preceq X \wedge g\{c\} \preceq X$. Matching t^q with the first database term in D yields the constraint $X \preceq g\{a, b, c\}$. Matching t^q with the second database term in D yields the constraint $X \preceq g\{b\} \wedge X \preceq g\{c\}$. $g\{b\} \preceq X \wedge g\{c\} \preceq X$ is not compatible with $X \preceq g\{b\} \wedge X \preceq g\{c\}$. Thus, the only possible value for X is $g\{a, b, c\}$, i.e. the only possible answer to t^q in D is $f\{g\{a, b, c\}, g\{a, b, c\}, h\}$.

Definition 7 (Ground Instances of Query Terms). *A* grounding substitution *is a function which assigns a label to each label variable and a database term to each term variable of a finite set of (label or term) variables. Let* t^q *be a query term,* $V_1, \ldots, V_n$ *be the (label or term) variables occurring in* t^q *and* σ *be a grounding substitution assigning* v_i *to* V_i. *The* ground instance $t^q\sigma$ *of* t^q *with respect to* σ *is the ground query term that can be constructed from* t^q *as follows:*

1. *Replace each subterm* $X \rightsquigarrow t$ *by* X.
2. *Replace each occurrence of* V_i *by* v_i $(1 \leq i \leq n)$.

Requiring in Definition 2 *desc* to occur to the right of $\rightsquigarrow$ makes it possible to characterise a ground instance of a query term by a grounding substitution. This is helpful for formalising answers but not necessary for language implementions. Not all ground instances of a query term are acceptable answers, for some instances might violate the conditions expressed by the $\rightsquigarrow$ and *desc* constructs.

Definition 8 (Allowed Instances). *The* constraint induced by a query term t^q *and a substitution* σ *is the conjunction of all inequalities* $t\sigma \preceq X\sigma$ *such that* $X \rightsquigarrow t$ *is a subterm of* t^q *not of the form desc* t_0, *and of all expressions* $X\sigma \lhd t\sigma$ *(read "*$t\sigma$ *subterm of* $X\sigma$*") such that* $X \rightsquigarrow$ *desc* t *is a subterm of* t^q, *if* t^q *has such subterms. If* t^q *has no such subterms, the constraint induced* t^q *and* σ *is the formula true. Let* σ *be a grounding substitution and* $t^q\sigma$ *a ground instance of* t^q. $t^q\sigma$ *is* allowed *if:*

1. *Each inequality* $t_1 \preceq t_2$ *in the constraint induced by* t^q *and* σ *is satisfied.*
2. *For each* $t_1 \lhd t_2$ *in the constraint induced by* t^q *and* σ, t_2 *is a subterm of* t_1.

Definition 9 (Answers). *Let t^q be a query term and D a database. An answer to t^q in D is a database term $t^{db} \in D$ such that there exists an allowed ground instance t of t^q satisfying $t \preceq t^{db}$.*

3.1 Concluding Remarks

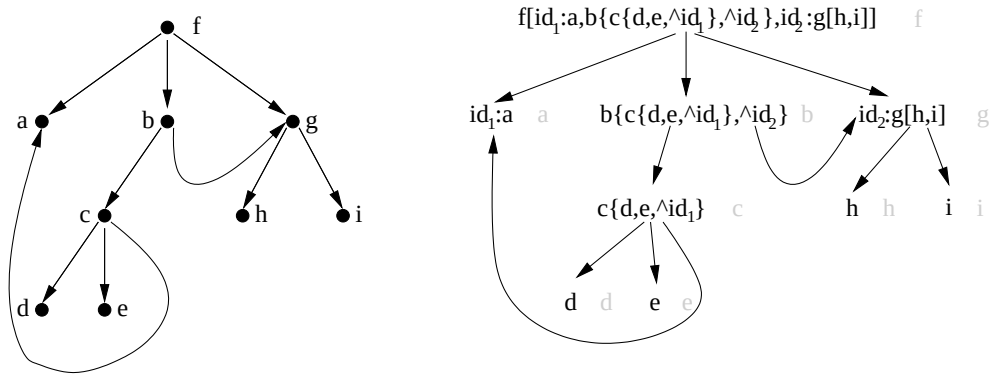
In previous works, simulation has been used for verifying the conformity of semistructured data to a schema cf. e.g. [5,6]. Here, it is used for query answering. The authors are not aware of former uses of simulation for query answering. [7] describes a language called fxt that has variables for terms, corresponding to trees, and forests. In fxt, node selection is done with regular expressions. In contrast to *xcerpt* and the basic language described above uses term variables for this purpose. In [8], a query and transformation language is described that is related to logic and Prolog. This language has (in the terminology used above) only label variables. In contrast, the basic query language introduced above also has term variables. In [9], the notion of matching subjacent to Definition 9 is shown to be decidable.

A language called *xcerpt* is under development which builds upon the basic language introduced above. *xcerpt* has construct terms in which variables but no *desc* and $\rightsquigarrow$ may occur. A construct term with variables $V_1, \ldots, V_n$ is associated with a conjunction or disjunction of (possibly negated) query terms in which all of $V_1, \ldots, V_n$ occur. *xcerpt* has several additional features that for space reasons cannot be mentioned here. In [9] some of these features and part of an operational semantics are introduced. A prototype has been realized that implement a set-oriented backward reasoning operational semantics. First experiments suggest that the *xcerpt* approach to querying is convenient in practice.

References

1. World Wide Web Consortium (W3C) http://www.w3.org/. (2002)
2. Olteanu, D., Meuss, H., Furche, T., Bry, F.: Xpath: Looking forward. In: Proceedings of Workshop on XML Data Management (XMLDM), http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-4, Springer LNCS (2002)
3. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing Simulations on Finite and Infinite Graphs (1996)
4. Maier, D.: Database Desiderata for an XML Query Language. In: Proceedings of QL'98 - The Query Languages Workshop. (1998) http://www.w3.org/TandS/QL/QL98/.
5. Fernandez, M., Suciu, D.: Optimizing Regular Path Expressions Using Graph Schemas. In: Proceedings of the Int. Conf. on Data Engineering. (1988) 14–23
6. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web. From Relations to Semistructured Data and XML. Morgan Kaufmann (2000)
7. Berlea, A., Seidl, H.: fxt – A Transformation Language for XML Documents. J. of Computing and Information Technology (CIT), Special Issue on Domain-Specific Languages (2001)
8. Grahne, G., Lakshmanan, L.V.S.: On the Difference between Navigating Semi-structured Data and Querying It. In: Proceedings of Workshop on Database Programming Languages. (1999) 271–296
9. Bry, F., Schaffert, S.: Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. Technical Report PMS-FB-2002-2, Inst. for Computer Sciences, University of Munich, http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-2 (2002)

Appendix



(a) Abstract node representation

(b) Full node representation
(the labels are printed in gray)

Fig. 1. Graph induced by $t = f[\mathrm{id}_1 : a, b\{c\{d, e, \uparrow \mathrm{id}_1\}, \uparrow \mathrm{id}_2\}, \mathrm{id}_2 : g[h, i]]$.

Note that the graph induced by a ground query term does not fully convey the term structure: Missing are graphical representations of the various nestings [ ], { }, [[ ]] and {{ }}.

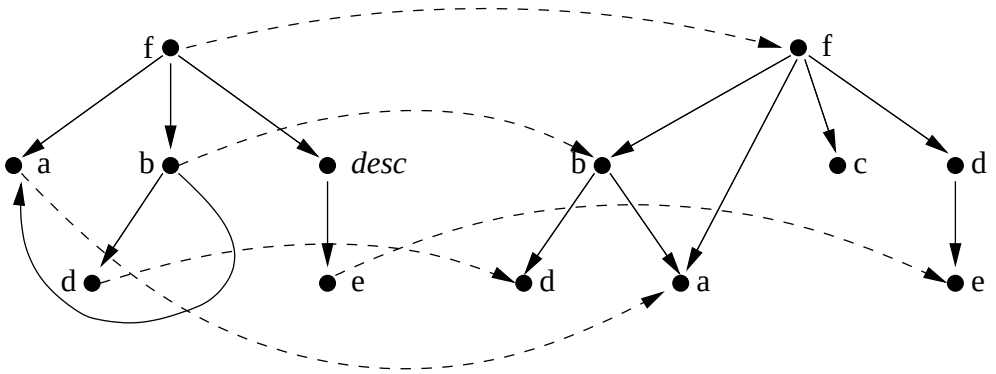


Fig. 2. A minimal simulation of the (graph induced by the) ground query term $t^q = f\{\mathrm{id}_1 : a, b[d\{\{\}\}, \uparrow \mathrm{id}_1], desce\}$ in the (graph induced by the) database term $t^{db} = f[b[d, \mathrm{id}_2 : a], \uparrow \mathrm{id}_2, c, d\{e\}]$.