Simulation Subsumption or Déjà vu on the Web

François Bry, Tim Furche, and Benedikt Linse

Institute for Informatics, University of Munich,
Oettingenstraße 67, 80538 München, Germany
http://pms.ifi.lmu.de/

Abstract. Simulation unification is a special kind of unification adapted to retrieving semi-structured data on the Web. This article introduces *simulation subsumption,* or containment, that is, query subsumption under simulation unification. Simulation subsumption is crucial in general for query optimization, in particular for optimizing pattern-based search engines, and for the termination of recursive rule-based web languages such as the XML and RDF query language Xcerpt. This paper first motivates and formalizes simulation subsumption. Then, it establishes decidability of simulation subsumption for advanced query patterns featuring descendant constructs, regular expressions, negative subterms (or subterm exclusions), and multiple variable occurrences. Finally, we show that subsumption between two query terms can be decided in $\mathcal{O}(n!^n)$ where n is the sum of the sizes of both query terms.

1 Introduction

Xcerpt query terms [1] are an answer to accessing Web data in a rule-based query language. Like most approaches to Web data (or semi-structured data, in general), they are distinguished from relational query languages such as SQL by a set of query constructs specifically attuned to the less rigid, often diverse, or even entirely schema-less nature of Web data. Xcerpt terms are similar to normalized forward XPath (see [2]) but extended with variables, deep-equal, a notion of injective match, regular expressions, and full negation. Thus, they achieve much of the expressiveness of XQuery without sacrificing the simplicity and pattern-structure of XPath.

When used in the context of Xcerpt, query terms serve a similar role to terms of first-order logic in logic languages. Therefore, the notion of unification has been adapted for Web data in [3], there called "simulation unification". This form of unification is capable of handling all the extensions of query terms over first-order terms that are needed to support Web data: selecting terms at arbitrary depth (`desc`), distinguishing partial from total terms, regular expressions instead of plain labels, negated subterms (`without`), etc.

To illustrate the notion of query term, consider the following query term. It selects the content of title elements at arbitrary depth (`desc`) under a book element in a bibliography database. In addition, we ask for the author of such a book, but only if both first-name and last-name of that author are recorded in

D. Calvanese and G. Lausen (Eds.): RR 2008, LNCS 5341, pp. 28–42, 2008.

that order. Finally, if there is also information about the publisher of that book, we retrieve that information as well:

```
bib{{
  book{{
    desc title{ var Title },
    var Author as author[[ first-name{{ }}, last-name{{ }} ]],
    optional var publisher as publisher{{ }}
  }}
}}
```

Subsumption or containment of two queries (or terms) is an established technique for optimizing query evaluation: a query q_1 is said to be *subsumed* by or *contained* in a query q_2 if every possible answer to q_1 against every possible data is also an answer to q_2. Thus, given all answers to q_2, we can evaluate q_1 only against those answers rather than against the whole database.

For first-order terms, subsumption is efficient and employed for guaranteeing termination in tabling (or memoization) approaches to backward chaining of logic [4,5]. However, when we move from first-order terms to Web queries subsumption (or containment) becomes quickly less efficient or even intractable. Xcerpt query terms have, as pointed out above, some similarity with XPath queries. Containment for various fragments of XPath is surveyed in [6], both in absence and in presence of a DTD. Here, we focus on the first setting, where no additional information about the schema of the data is available. However, Xcerpt query terms are a strict super-set of (navigational) XPath as investigated in [6]. In particular, the Xcerpt query terms may contain (multiple occurrences of the same) variables. This brings them closer to *conjunctive queries* (with negation and deep-equal), as considered in [7] on general relations, and in [8] for tree data. Basic Xcerpt query terms can be reduced to (unions of) conjunctive queries with negation. However, the injectivity of Xcerpt query terms (no two siblings may match with the same data node) and the presence of deep-equal (two nodes are deep-equal iff they have the same structure) have no direct counterpart in conjunctive query containment. Though [9] shows how inequalities in general affect conjunctive query containment, the effect of injectivity (or all-distinct constraints) on query containment has not been studied previously. The same applies to deep-equal, though the results in [10] indicate that in *absence* of composition deep-equal has no effect on evaluation and thus likely on containment complexity.

For Xcerpt query terms, subsumption is, naturally, of interest for the design of a terminating, efficient Xcerpt engine. Beyond that, however, it is particularly relevant in a Web setting. Whenever we know that one query subsumes another, we do not need to access whatever data the two queries access twice, but rather can evaluate both queries with a single access to the basic data by evaluating the second query on the answers of the first one. This can be a key optimization also in the context of search engines, where answers to frequent queries can be memorized so as to avoid their repeated computation. Even though today's search engines are rather blind of the tree or graph structure of HTML, XML and

RDF data, there is no doubt that some more or less limited form of structured queries will become more and more frequent in the future (see Google scholar's "search by author, date, etc."). Query subsumption, or containment, is key to a selection of queries, the answers to which are to be stored so as to allow as many queries as possible to be evaluated against that small set of data rather than against the entire search engine data. Thus, the notion of simulation subsumption proposed in this paper can be seen as a building block of future, structure-aware search engines.

Therefore, we study in this paper subsumption of Xcerpt query terms. To that end, the main contributions are:

- we introduce and formalize a notion of subsumption for Xcerpt query terms, called *simulation subsumption*, in Section 3. To the best of our knowledge, this is the first notion of subsumption for queries with injectivity of sibling nodes and deep-equal.
- we show, also in Section 3, that simulation on query terms is equivalent to simulation subsumption. This also shows that simulation unification as introduced in [3] indeed captures the intuition that a query term that simulates into another query term subsumes that term. Furthermore, all results for simulation subsumption apply equally to simulation unification.
- we define, in Section 4, a *rewriting system* that allows us to reduce the test for subsumption of q in q' to finding a sequence of syntactic transformations that can be applied to q to transform it into q'.
- we show, in Section 5, that this rewriting system gives rise to an algorithm for testing subsumption that is sound and complete and can determine whether q subsumes q' in time $\mathcal{O}(n!^n)$. In particular, this shows that simulation subsumption is decidable.

2 Xcerpt Basics: Query Terms and Simulation

This section lays the foundations for simulation subsumption by introducing the notions of semi-structured trees (Definition 1), query terms (Definition 2) and simulation (Definition 5). Semi-structured trees are an abstraction for all kinds of Web data such as XML-documents, RDF graphs or HTML-documents.

Definition 1 (Semi-structured Trees). *Trees (also called data terms in the following) are are inductively defined as follows:*

- *a label is an atomic tree*
- *if l is a label and $t_1, \ldots, t_n$ are trees with $n \geq 1$, then $l\{t_1, \ldots, t_n\}$ is a tree.*

Query terms are an abstraction for queries that can be used to extract data from semi-structured trees. In contrast to XPath queries, they may contain (multiple occurrences of the same) variables and demand an *injective mapping* of the child terms of each term. For example, the XPath query `/a/b[c]/c` demands that the document root has label `a`, and has a child term with label `b` that has itself

a child term with label c. The subterm c that is given within the predicate of b can be mapped to the same node in the data as the child named c of b. Therefore this XPath query would be equivalent to the query term $a\{\{b\{\{c\}\}\}\}$, but not to $a\{\{b\{\{c,c\}\}\}\}$. Simulation could be, however, easily modified to drop the injectivity requirement.

Recall the example query term from Section 1:

```
bib{{
  book{{
    desc title{ var Title },
    var Author as author[[ first-name{{ }}, last-name{{ }} ]],
    optional var publisher as publisher{{ }}
  }}
}}
```

This query term illustrates most of the features of Xcerpt query terms relevant for this paper. As stated above, it selects titles, authors, and optionally publishers of the same book. Titles may occur at any depth under the book element (indicated by desc), but authors and publishers must be children. For authors we further ask that they also record first and last name and that these are recorded in that order (i.e., not last-name before first-name).

Single (double, resp.) braces or brackets in an Xcerpt query term mean that the term's content model is completely (incompletely, resp.) specified (i.e. there must only be a single subterm within the title element of the example from Section 1, but the author element may contain other children besides first-name and last-name). (Curly) braces mean that the order of occurrence of the subterm in the data is irrelevant, (square) brackets enforce the same order in the data as in the query term (i.e. first-name must appear before last-name in the data, otherwise the query term from Section 1 does not match).

Even though there are (many) XML serializations for RDF data, most prominently RDF/XML, none convey the inherent graph structure of RDF data. Each RDF serialization either approximates an RDF graph by a tree, or decomposes it into triples. Xcerpt natively supports RDF with constructs conveying RDF specifics such as containers, collections, the type system and reification. For the sake of focus and simplicity, these RDF constructs are not addressed in the present paper. A complete presentation of Xcerpt's construct for RDF is given in [11].

Definition 2 ((Xcerpt) Query Terms). *Query terms over a set of labels $\mathcal{N}$, a set of variables $\mathcal{V}$, and a set of regular expressions $\mathcal{R}$ are inductively defined as follows:*

- for each label $l \in \mathcal{N}$, $l\{\{\ \}\}$ and $l\{\ \}$ are atomic query terms. l is a short hand notation for $l\{\{\ \}\}$. The formal treatment of square brackets in query terms is omitted in this contribution for the sake of brevity.
- for each variable $X \in \mathcal{V}$, var X is a query term
- for each regular expression $r \in \mathcal{R}$, $/r/$ is a query term. With $\mathcal{L}(r)$ we denote the set of labels matched by r, i.e. the language defined by the regular expression.

- for each variable $X \in \mathcal{V}$ and query term t, X `as` t is a query term. t is called a *variable restriction* for X.
- for each query term t, `desc` t is a query term and called *depth-incomplete* or *incomplete in depth*.
- for each query term t, `without` t is a query term and called a *negated subterm*.
- for each label l and query terms $t_1, \ldots, t_n$ are query terms with $n \geq 1$,

$$q_1 = l\texttt{\{\{ } t_1\texttt{, \ldots, } t_n\texttt{\}\}}$$
$$q_2 = l\texttt{\{ } t_1\texttt{, \ldots, } t_n\texttt{\}}$$

  are query terms. q_1 is said to be *incompletely specified in breadth*, or simply *breadth-incomplete*, whereas q_2 is *completely specified in breadth*, or simply *breadth-complete*.

In the following, we let $\mathcal{D}$ and $\mathcal{Q}$ denote the set of all semi-structured trees and query terms, respectively.

A query term and a semi-structured tree are in the simulation relation, if the query term "matches" the data. Matching trees with data is very similar to matching Xpath queries with XML documents – apart from the variables and the injectivity requirement in query terms. The formal definition of simulation of a query term with a semi-structured tree is somewhat involved. To shorten the presentation, we first introduce some notation:

Definition 3 (Injective and Bijective Mappings). *Let* $I := \{t_1^1, \ldots, t_k^1\}$, $J := \{t_1^2, \ldots, t_n^2\}$ *be sets of query terms and* $\pi : I \Rightarrow J$ *be a mapping.*

- π *is* injective, *if all* $t_i^1, t_j^1 \in I$ *satisfy* $t_i^1 \neq t_j^1 \Rightarrow \pi(t_i^1) \neq \pi(t_j^1)$.
- π *is* bijective, *if it is injective and for all* $t_j^2 \in J$ *there is some* $t_i^1 \in I$ *such that* $\pi(t_i^1) = t_j^2$.

We use the following abbreviations to reference parts of a query term q:

$l(q)$**:** the label of q,
$ChildT(q)$**:** the set of child subterms of q, i.e. those directly nested inside of q.
$ChildT^+(q)$**:** the set of positive direct subterms (i.e. those direct subterms which are not of the form *without*...),
$ChildT^-(q)$**:** the set of negated direct subterms (i.e. the direct subterms of the form *without*...),
$Desc(q)$**:** the set of direct descendant subterms of q (i.e. those of the from *desc*...),
$SubT(q)$**:** the direct or indirect subterms of q, i.e. all direct subterms as well as their subterms.
$ss(q)$**:** the subterm specification of q. It can either be *complete* (single curly braces) or *incomplete* (double curly braces).
$vars(q)$**:** the set of variables occurring somewhere in q.
$pos(q)$**:** q', if q is of the form `without` q' for some query term, q otherwise.

Definition 4 (Ground Query Term Simulation). *Let q be a query term and d be a semi-structured tree, A relation $\mathcal{S} \subseteq (SubT(q) \cup \{q\}) \times (SubT(d) \cup \{d\})$ is a simulation of q into d if the following holds:*

- $q \; \mathcal{S} \; d$
- *if* $q := l_1\{\{q_1, \ldots, q_n\}\} \; \mathcal{S} \; l_2\{d_1, \ldots, d_m\} =: d$ *then* l_1 *must subsume* l_2*, and there must be an injective mapping* $\pi : ChildT^+(q) \rightarrow ChildT^+(d)$ *such that* $q_i \; \mathcal{S} \; \pi(q_i)$ *for all* $i \in ChildT^+(q)$*. Moreover, there* must not *be a* $q_j \in ChildT^-(q)$ *and* $d_l \in ChildT^+(d) \setminus range(\pi)$ *such that* $pos(q_j) \; \mathcal{S} \; d_l$*.*
- *if* $q := l1\{q_1, \ldots, q_n\} \; \mathcal{S} \; l2\{d_1, \ldots, d_m\} =: d$ *then* l_1 *must subsume* l_2*, and there must be a bijective mapping* $\pi : ChildT^+(q) \rightarrow ChildT^+(d)$ *such that* $q_i \; \mathcal{S} \; \pi(q_i)$ *for all* $i \in ChildT^+(q)$*. Note that the set* $ChildT^-(q)$ *of negated direct subterms of* q *should be empty – the presence of negated subterms in breadth-complete query terms is irrelevant.*
- *if* $q = desc \; q' \; \mathcal{S} \; d$ *then* $q' \; \mathcal{S} \; d$ *or* $q' \; \mathcal{S} \; d'$ *for some subterm* d' *of* d*.*

If there is a relation $\mathcal{S}$ that satisfies the above conditions, q simulates into *d (short: $q \preceq d$; to state the contrary we write $q \not\preceq d$).*

Since every semi-structured tree is also a query term, the above definition of simulation between a query term and a tree can be extended to a relation between pairs of query terms. For the sake of brevity this full definition of *extended ground query term simulation* is given in the appendix of the online version [12].

The existence of a ground query term simulation states that a given semi-structured tree satisfies the conditions encapsulated in the query term. Many times, however, query authors are not only interested in checking the structure and content of a document, but also in extracting data from the document, and therefore query terms may contain logical variables. To formally specify the data that is extracted by matching a query term with a semi-structured tree, non-ground query term is introduced (Definition 5). Substitutions are defined as usual, and the application of a substitution to a query term is the consistent replacement of the variables by their images in the substitution.

Definition 5 (Non-Ground Query Term Simulation). *A query term q with variables simulates into a semi-structured tree d iff there is a substitution $\sigma : Vars(q) \rightarrow \mathcal{D}$ such that $q\sigma$ simulates into d.*

3 Simulation Subsumption

In this section, we first introduce simulation subsumption (Definition 6), then for several query terms we discuss whether one subsumes the other to give an intuition for the compositionality of the subsumption relationship. Subsequently, the transitivity of the subsumption relationship is proven (Lemma 1), some conclusions about the membership in the subsumption relationship of subterms, given the membership in the subsumption relationship of their parent terms are

stated. These conclusions formalize the compositionality of simulation subsumption and are a necessary condition for the completeness of the rewriting system introduced in Section 4.

In tabled evaluation of logic programs, solutions to subgoals are saved in a solution table, such that for equivalent or subsumed subgoals, these sets do not have to be recomputed. As mentioned before, this avoidance of re-computation does not only save time, but can, in certain cases be crucial for the termination of a backward chaining evaluation of a program. In order to classify subgoal as solution or look-up goals, boolean subsumption as specified by Definition 6 must be decided. Although Xcerpt query terms may contain variables, n-ary subsumption as defined in [6] would be too strict for our purposes. To see this, consider the Xcerpt query terms $q_1 := a\{\{var\ X\}\}$ and $q_2 := a\{\{c\}\}$. Although all trees that are relevant for q_2 can be found in the solutions for q_1, q_1 and q_2 cannot be compared by n-ary containment, because they differ in the number of their query variables.

Definition 6 (Simulation Subsumption). *A query term q_1* subsumes *another query term q_2 if all data or query terms that q_2 simulates with are also simulated by q_1.*

Example 1 (Examples for the subsumption relationship). Let $q_1 := a\{\{\}\}$, $q_2 := a\{\{desc\ b, desc\ c, d\}\}$, $q_3 := a\{\{desc\ b, c, d\}\}$, $q_4 := a\{\{without\ e\}\}$, and $q_5 := a\{\{without\ e\{\{without\ f\}\}\}\}$. Then the following subsumption relationships hold:

- q_2 subsumes q_3 because it requires less than q_3: While q_3 requires that the data has outermost label a, subterms c and d as well as a descendant subterm b, q_2 requires not that there is a direct subterm c, but only a descendant subterm. Since every descendant subterm is also a direct subterm, all trees simulating with q_3 also simulate with q_2.
  But the subsumption relationship can also be decided in terms of simulation: q_2 subsumes q_3, because there is a mapping π from the direct subterms $Child(q_2)$ of q_2 to the direct subterms $Child(q_3)$ of q_3, such that q_i subsumes $\pi(q_i)$ for all q_i in $Child(q_2)$.
- q_3 does not subsume q_2, since there are trees that simulate with q_2, but not with q_3. One such tree is $d := a\{b, e\{c\}, d\}$.
  Again, the subsumption relationship between q_3 and q_2 (in this order) can be decided by simulation. There is no mapping π from the direct subterms of q_3 to the direct subterms of q_2, such that a simulates into $\pi(a)$.
- q_1 subsumes q_4 since it requires less than q_4. All trees that simulate with q_4 also simulate with q_1.
- q_4 does not subsume q_1, since the tree $a\{\{e\}\}$ simulates with q_1, but does not simulate with q_4.
- q_5 subsumes q_4, but not the other way around.

Proposition 1. *The subsumption relationship between query terms is transitive, i.e. for arbitrary query terms q_1, q_2 and q_3 it holds that if q_1 subsumes q_2 and q_2 subsumes q_3, then q_1 subsumes q_3.*

Proposition 1 immediately follows from the transitivity of the subset relationship. Query term simulation and subsumption are defined in a way such that, given the simulation subsumption between two query terms, one can draw conclusions about subsumption relationships that must be fulfilled between pairs of subterms of the query terms. Lemma 1 formalizes these sets of conclusions.

Lemma 1 (Subterm Subsumption). *Let q_1 and q_2 be query terms such that q_1 subsumes q_2. Then there is an injective mapping π from $Child^+(q_1)$ to $Child^+(q_2)$ such that q_1^i subsumes $\pi(q_1^i)$ for all $q_1^i \in Child^+(q_1)$.*

Furthermore, if q_1 and q_2 are breadth-incomplete, then there is a (not necessarily injective) mapping σ from $Child^-(q_1)$ to $Child^-(q_2)$ such that $pos(\sigma(q_1^j))$ subsumes $pos(q_1^j)$ for all $q_1^j \in Child^-(q_1)$.

If q_1 is breadth-incomplete and q_2 is breadth-complete then there is no q_1^j in $Child^-(q_1)$ and $q_2^k \in Child^+(q_2) \setminus range(\pi)$ such that $pos(q_1^j) \preceq q_2^k$.

Lemma 1 immediately follows from the equivalence of the subsumption relationship and the extended query term simulation (see Lemma 4 in the appendix of the online version [12]).

4 Simulation Subsumption by Rewriting

In this section we lay the foundations for a proof for the decidability of subsumption between query terms according to Definition 6 by introducing a rewriting system from one query term to another, which is later shown to be sound and complete. Furthermore this rewriting system lays the foundation for the complexity analysis in Section 5.3.

The transformation of a query term q_1 into a subsumed query term q_2 is exemplified in Figure 1.

Definition 7 (Subsumption Monotone Query Term Transformations). *Let q be a query term. The following is a list of so-called* subsumption monotone *query term transformations.*

- *if q has incomplete subterm specification, it may be transformed to the analogous query term with complete subterm specification.*

$$\frac{a\{\{q_1, \ldots, q_n\}\}}{a\{q_1, \ldots, q_n\}}, \tag{1}$$

- *if q is of the form desc q' then the descendant construct may be eliminated or it may be split into two descendant constructs separated by the regular expression `/.*/`, the inner descendant construct being wrapped in double curly braces.*

$$\frac{desc\ q}{q}, \qquad \frac{desc\ q}{desc\ /.*/\{\{desc\ q\}\}} \tag{2}$$

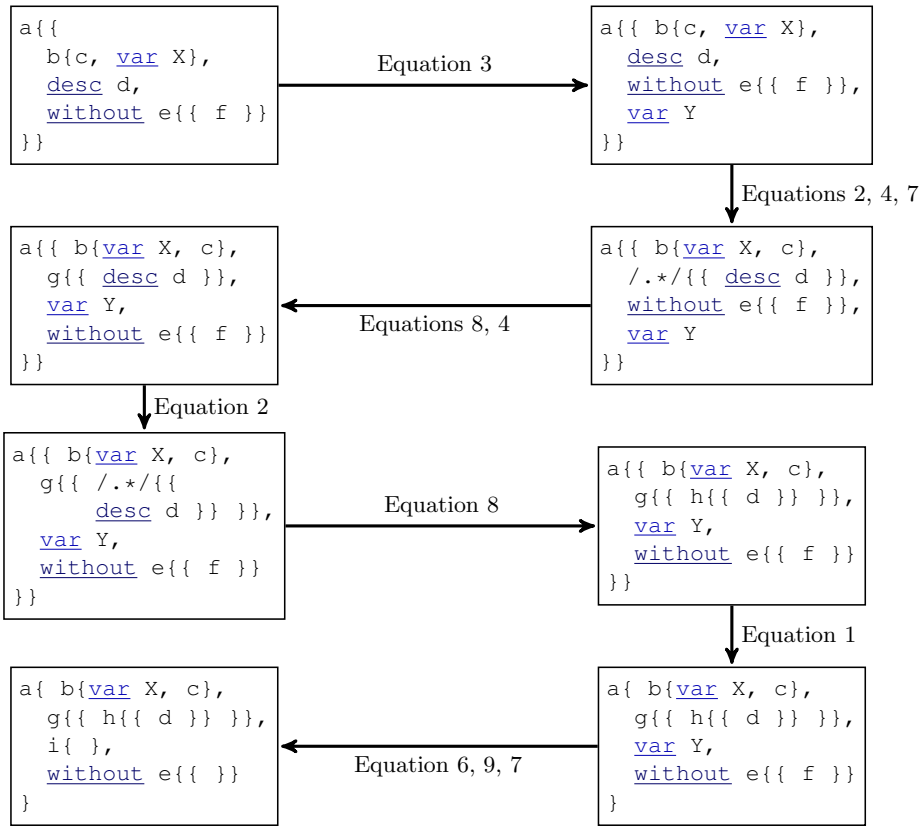

Fig. 1.

– *if q has incomplete-unordered subterm specification, then a fresh variable X may be appended to the end of the subterm list. A* fresh *variable is a variable that does not occur in q_1 or q_2 and is not otherwise introduced by the rewriting system.*

$$X \text{ fresh} \Rightarrow \frac{a\{\{q_1, \ldots, q_n\}\},}{a\{\{q_1, \ldots, q_n, var\ X\}\}} \tag{3}$$

– *if q has unordered subterm specification, then the subterms of q may be arbitrarily permuted.*

$$\pi \in Perms(\{1, \ldots, n\}) \Rightarrow \frac{a\{\{q_1,\ \ldots,\ q_n\}\}}{a\{\{q_{\pi(1)},\ \ldots,\ q_{\pi(n)}\}\}} \tag{4}$$

$$\pi \in Perms(\{1, \ldots, n\}) \Rightarrow \frac{a\{q_1,\ \ldots,\ q_n\}}{a\{q_{\pi(1)},\ \ldots,\ q_{\pi(n)}\}} \tag{5}$$

– *if q contains a variable* `var X`*, which occurs in q at least once in a positive context (i.e. not within the scope of a without) then all occurrences of* `var X` *may be substituted by another Xcerpt query term.*

$$X \in PV(q), t \in QTerms \Rightarrow \frac{q}{q\{X \mapsto t\}} \tag{6}$$

This rule may only be applied, if q contains all *occurrences of X in q_1. Furthermore, no further rewriting rules may be applied to the replacement term t.*

Notice that if a variable appears within q only in a negative context (i.e. within the scope of a `without`*), the variable cannot be substituted by an arbitrary term to yield a transformed term that is subsumed by q. The query terms* `a{{ without var X }}` *and* `a{{ without b{ } }}` *together with the tree* `a{ c }` *illustrate this characteristic of the subsumption relationship. For further discussion of substitution of variables in a negative context see Example 2.*

– *if q has a subterm q_i, then q_i may be transformed by any of the transformations in this list except for Equation 6 to the term $t(q_i)$, and this transformed version may be substituted at the place of q_i in q, as formalized by the following rule:* [1,2]

$$\frac{q_i}{t(q_i)} \Rightarrow \frac{a\{\{q_1, \ldots, q_n\}\}}{a\{\{q_1, \ldots, q_{i-1}, t(q_i), q_{i+1}, \ldots q_n\}\}} \tag{7}$$

– *if the label of q is a regular expression e, this regular expression may be replaced by any label that matches with e, or any other regular expression e' which is subsumed by e (see Definition 8 in the appendix of the online version [12]).*[1]

$$e \in RE, e \text{ subsumes } e' \Rightarrow \frac{e\{\{q_1, \ldots, q_n\}\}}{e'\{\{q_1, \ldots, q_n\}\}} \tag{8}$$

– *if q contains a negated subterm $q_i = $ without r and r' is a query term such that $t(r') = r$ (i.e. r' subsumes r) for some transformation step t, then q_i can be replaced by $q_i' :=$ without r'.*

$$(q_i = \text{without } r) \wedge \frac{r'}{r} \wedge (q_i' = \text{without } r') \Rightarrow \frac{a\{\{q_1, \ldots, q_i, \ldots, q_n\}\}}{a\{\{q_1, \ldots, q_i', \ldots q_n\}\}} \tag{9}$$

[1] The respective rules for complete-unordered subterm specification, incomplete-ordered subterm specification and complete-ordered subterm specification are omitted for the sake of brevity.

[2] The exclusion of Equation 6 ensures that variable substitutions are only applied to entire query terms and not to subterms. Otherwise the same variable might be substituted by different terms in different subterms.

5 Properties of the Rewriting System

In this section we show that the rewriting system introduced in the previous section is sound (Section 5.1) and complete (Section 5.2). Furthermore we study the structure of the search tree induced by the rewriting rules, show that it can be pruned without losing the completeness of the rewriting system and conclude that simulation subsumption is decidable. Finally we derive complexity results from the size of the search tree in Section 5.3.

5.1 Subsumption Monotonicity and Soundness

Lemma 2 (Monotonicity of the Transformations in Definition 7). *All of the transformations given in Definition 7 are subsumption monotone, i.e. for any query term q and a transformation from Definition 7 which is applicable to q, q subsumes $t(q)$.*

The proof of Lemma 2 is straight-forward since each of the transformation steps can be shown independently of the others. For all of the transformations, inverse transformation steps t^{-1} can be defined, and obviously for any query term q it holds that $t^{-1}(q)$ subsumes q.

Lemma 3 (Transitivity of the Subsumption Relationship, Monotonicity of a Sequence of Subsumption Monotone Query Term Transformations). *For a sequence of subsumption monotone query term transformations $t_1, \ldots, t_n$, and an arbitrary query term q, q subsumes $t_1 \circ \ldots \circ t_n(q_1)$.*

The transitivity of the subsumption relationship is immediate from its definition (Definition 6) which is based on the subset relationship, which is itself transitive.

As mentioned above, the substitution of a variable X in a negative context of a query term q by a query term t, which is not a variable, results in a query term $q' := q[X \mapsto t]$ which is in fact more general than q. In other words $q[X \mapsto t]$ subsumes q for any query term q if X only appears within a negative context in q. On the other hand, if X only appears in a positive context within q, then q' is less general – i.e. q subsumes q'. But what about the case of X appearing both in a positive and a negative context within q? Consider the following example:

Example 2. Let $q :=$ `a{{ var X, without b{{ var X }} }}`. One may be tempted to think that substituting X by `c[]` to give q' makes the first subterm of q less general, but the second subterm of q more general. In fact, a subterm `b[ c ]` within a tree would cause the subterm `without b{{ var X }}` of q to fail, but the respective subterm of q' to succeed, suggesting that there is a tree that simulation unifies with q', but not with q, meaning that q does not subsume q'. However, there is no such tree, which is due to the fact that the second occurrence of `X` within q is only a consuming occurrence. When this part of the query term is evaluated, the variable `X` is already bound.

The normalized form for Xcerpt query terms is introduced, because for an unnormalized query term q_1 that subsumes a query term q_2 one cannot guarantee that there is a sequence of subsumption monotone query term transformations $t_1, \ldots, t_n$ such that $t_n \circ \ldots \circ t_1(q_1) = q_2$. To see this, consider example 3.

Example 3 (Impossibility of transforming an unnormalized query term). Let $q_1 := a\{\{var\ X\ as\ b\{\{c\}\}, var\ X\ as\ b\{\{d\}\}\}\}$ and $q_2 := a\{\{b\{\{c,d\}\}, b\{\{c,d\}\}\}\}$. q_2 subsumes q_1, in fact both terms are even simulation equivalent. But there is no sequence of subsumption monotone query term transformations from q_2 to q_1, since one would have to omit one subterm from both the first subterm of q_2 and from the second one. But such a transformation would not be subsumption monotone.

To overcome this issue, query terms are assumed to be in normalized form (Definition 9 in the appendix of the online version [12]). In fact, almost all Xcerpt query terms can be transformed into normalized form.

5.2 Completeness

Theorem 1 (Subsumption by Transformation). *Let q_1 and q_2 be two query terms in normalized form such that q_1 subsumes q_2. Then q_1 can be transformed into q_2 by a sequence of subsumption monotone query term transformations listed in Definition 7.*

Proof. We distinguish two cases:

- q_1 and q_2 are subsumption equivalent (i.e. they subsume each other)
- q_1 strictly subsumes q_2

The first case is the easier one. If q_1 and q_2 are subsumption equivalent, then there is no tree t, such that t simulates with one, but not the other. Hence q_1 and q_2 are merely syntactical variants of each other. Then q_1 can be transformed into q_2 by consistent renaming of variables (Equation 7), and by reordering sibling terms within subterms of q (Equation 4). Note that this would not be true for unnormalized query terms as Example 3 shows.

The second is shown by structural induction on q_1.

For both the induction base and the induction step, we assume that q_1 subsumes q_2, but not the other way around. Then there is a tree d, such that q_1 simulates into d, but q_2 does not. In both the induction base and the induction step, we give a distinction of cases, enumerating all possible reasons for q_1 simulating into d but q_2 not. For each of these cases, a sequence of subsumption monotone transformations $t_1, \ldots t_n$ from Definition 7 is given, such that $q_1' := t_n \circ t_{n-1} \circ \ldots \circ t_1(q_1)$ does *not* simulate into d. By Lemmas 2 and 3, q_1' still subsumes q_2. Hence by considering d and by applying the transformations, q_1 is brought "closer" to q_2. If q_1' is still more general than q_2, then one more dataterm d' can be found that simulates with q_1', but not with q_2, and another

sequence of transformations to be applied can be deduced from this theorem. This process can be repeated until q_1 has been transformed into a simulation equivalent version of q_2. For the proof see the appendix of the online version [12].

5.3 Decidability and Complexity

In the previous section, we establish that, for each pair of query terms q_1, q_2 such that q_1 subsumes q_2, there is a (possibly infinite) sequence of transformations $t_1, \ldots, t_k$ by one of the rules in Section 4 such that $t_k \circ \ldots \circ t_1(q) = q_2$.

However, if we reconsider the proof of Theorem 1, it is quite obvious that the sequence of transformations can in fact not be infinite: Intuitively, we transform at each step in the proof q_1 further towards q_2, guided by a data term that simulates in q_1 but not in q_2. In fact, the length of a transformation sequence is bounded by the sum of the sizes of the two query terms. As size of a query term we consider the total number of its subterms.

Proposition 2 (Length of Transformation Sequences). *Let q_1 and q_2 be two Xcerpt query terms such that q_1 subsumes q_2 and n the sum of the sizes of q_1 and q_2. Then, there is a sequence of transformations $t_1, \ldots, t_k$ such that $t_k \circ \ldots \circ t_1(q_1) = q_2$ and $k \in \mathcal{O}(n)$.*

Proof. We show that the sequences of transformations created by the proof of Theorem 1 can be bounded by $\mathcal{O}(n + m)$ if computed in a specific way: We maintain a mapping μ from subterms of q_1 to subterms of q_2 indicating how the query terms are mapped. μ is initialized with (q_1, q_2). In the following, we call a data term d *discriminating* between q_1 and q_2 if q_1 simulates in d but not q_2.

(1) For each pair (q, q') in μ, we first choose a discriminating data term that matches case 1 in the proof of Theorem 1. If there is such a data term, we apply Equation (8), label replacement, once to q obtaining $t(q)$ and update the pair in μ by $(t(q), q')$. This step is performed at most once for each pair as $(t(q), q')$ have the same label and thus there is no more discriminating data term that matches case 1.

(2) Otherwise, we next choose a discriminating data term that matches case 2.a.i or 2.b.i. In both cases, we apply Equation (3), variable insertion, to insert a new variable and update the pair in μ. This step is performed at most $|q_2| - |q_1| \leq n$ times for each pair.

(3) Otherwise, we next choose a discriminating data term that matches case 2.a.ii and apply Equation (1), complete term specification and update the pair in μ. This step is performed at most once for each pair.

(4) Finally, the only type of discriminating data term that remains is one with the same number of positive child terms as q_2. We use an oracle to guess the right mapping σ from child terms of q_1 to child terms of q_2. Then we remove the pair from μ and add $(c, \sigma(c))$ to μ for each child term of q_1. This step is performed at most once for each pair in μ.

Since query subterms have a single parent, we add each subterm only once to μ in a pair. Except for case 2, we perform only a constant number of transformations

to each pair. Case 2 allows up to n transformations for a single pair, but the total number of transformations (over all pairs) due to case 2 is bound by the size of q_2. Thus in total we perform at most $4 \cdot n$ transformations where n is the sum of the number of the sizes of q_1 and q_2.

Though we have established that the length of a transformation sequence is bound by $\mathcal{O}(n)$, we also have to consider how to *find* such a transformation sequence. The proof of Proposition 2, already spells out an algorithm for finding such transformation sequences. However, it uses an oracle to guess the right mapping between child terms of two terms that are to be transformed. A naive deterministic algorithm needs to consider all possible such mappings whose number is bound by $\mathcal{O}(n!)$. It is worth noting, however, that in most cases the actual number of such mappings is much smaller as most query terms have fairly low breadth and the possible mappings between their child terms are severely reduced just by considering only mappings where the labels of child terms simulate. However, in the worst case the $\mathcal{O}(n!)$ complexity for finding the right mapping may be reached and thus we obtain:

Theorem 2 (Complexity of Subsumption by Rewriting). *Let q_1 and q_2 be two Xcerpt query terms. Then we can test whether q_1 subsumes q_2 in $\mathcal{O}(n!^n)$ time.*

Proof. By proposition 2 we can find a $\mathcal{O}(n)$ length transformation sequence in $\mathcal{O}(n!^n)$ time and by Theorem 1 q_1 subsumes q_2 if and only if there is such a sequence.

6 Conclusion

Starting out from the problem of improving termination of logic programming based on rich kinds of simulation such as simulation unification, the problem of deciding simulation subsumption between query terms is investigated in this paper. A rewriting system consisting of subsumption monotone query term transformations is introduced and shown to be sound and complete. By convenient pruning of the search tree defined by this rewriting system, the decidability of simulation subsumption is proven, and an upper bound for its complexity is identified.

Future work includes (a) a proof of concept implementation of the rewriting system, (b) the development of heuristics and their incorporation into the prototype to ensure fast termination of the algorithm in the cases when it is possible, (c) the study of the complexity of the problem in absence of subterm negation, descendant constructs, deep-equal, and/or injectivity, (d) the implementation of a backward chaining algorithm with tabling, which uses subsumption checking to avoid redundant computations and infinite branches in the resolution tree, and (e) the adaptation of the rewriting system to XPath in order to decide subsumption and to derive complexity results for the subsumption problem between XPath queries.

References

1. Schaffert, S., Bry, F.: Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In: Proc. Extreme Markup Languages (Int'l. Conf. on Markup Theory & Practice) (2004)
2. Olteanu, D., Meuss, H., Furche, T., Bry, F.: XPath: Looking forward. In: Chaudhri, A.B., Unland, R., Djeraba, C., Lindner, W. (eds.) EDBT 2002. LNCS, vol. 2490. Springer, Heidelberg (2002)
3. Schaffert, S.: Xcerpt: A Rule-Based Query and Transformation Language for the Web. PhD thesis, University of Munich (2004)
4. Tamaki, H., Sato, T.: OLD resolution with tabulation. In: Proc. Int'l. Conf. on Logic Programming (ICLP), London, UK, pp. 84–98. Springer, Heidelberg (1986)
5. Chen, W., Warren, D.S.: Tabled evaluation with delaying for general logic programs. Journal of the ACM 43(1), 20–74 (1996)
6. Schwentick, T.: XPath query containment. SIGMOD Record 33(1), 101–109 (2004)
7. Wei, F., Lausen, G.: Containment of conjunctive queries with safe negation. In: Calvanese, D., Lenzerini, M., Motwani, R. (eds.) ICDT 2003. LNCS, vol. 2572, pp. 343–357. Springer, Heidelberg (2002)
8. Björklund, H., Martens, W., Schwentick, T.: Conjunctive Query Containment over Trees. In: Arenas, M., Schwartzbach, M.I. (eds.) DBPL 2007. LNCS, vol. 4797, pp. 66–80. Springer, Heidelberg (2007)
9. Klug, A.: On Conjunctive Queries containing Inequalities. Journal of the ACM 35(1), 146–160 (1988)
10. Koch, C.: On the Complexity of Nonrecursive XQuery and Functional Query Languages on Complex Values. Transactions on Database Systems 31(4) (2006)
11. Pohl, A.: RDF Querying in Xcerpt: Language Constructs and Implementation. Deliverable I4-Dx2, REWERSE (2008)
12. Bry, F., Furche, T., Linse, B.: Simulation subsumption or déjà vu on the web (extended version), `http://www.pms.ifi.lmu.de/mitarbeiter/linse/bry-simulation.pdf`