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

Ludwig
Maximilians
Universität
München

LMU

A contribution to the Semantics of Xcerpt, a Web Query and Transformation Language (Extended Abstract)

François Bry, Sebastian Schaffert, Andreas Schroeder

Technical Report, Computer Science Institute, Munich, Germany
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-2004-3, January 2004

A contribution to the Semantics of Xcerpt, a Web Query and Transformation Language (Extended Abstract)

François Bry, Sebastian Schaffert and Andreas Schroeder

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

1 Introduction

Xcerpt [1] is a declarative and pattern-based query and transformation language for the Web with deductive capabilities. In contrast to Web query languages like XQuery and XSLT [2,3], Xcerpt relies on concepts and techniques from logic programming and automated theorem proving such as declarative "query patterns" and "rule chaining". Xcerpt can also be used for querying Web metadata, like OWL or RDF data [4,5], and reasoning on such metadata. In contrast to specific languages for OWL and RDF, however, Xcerpt is a general purpose query, transformation, and reasoning language, i.e. it can be used for reasoning not only with Web metadata but also with plain Web data.

Salient aspects of Xcerpt are its nonstandard "query patterns" for retrieving incompletely specified data and its unusual "grouping constructs" *some* and *all* that significantly depart from the standard approaches in logic programming or automated theorem proving. Xcerpt relies on a new, assymmetric unification, called *simulation unification* for evaluating *query patterns* that incompletely specify data. Furthermore, Xcerpt does not rely on meta reasoning for expressing and processing "grouping" constructs corresponding to Prolog's metalevel predicates *setof* and *bagof*.

This abstract gives a brief overview over challenges of applying logic programming techniques to Web querying. In particular it suggests two different approaches for treating the meta-level grouping constructs *all* and *some* in a proof calculus formalising the operational semantics of Xcerpt.

2 Requirements of a Web Query Language

2.1 Differences to Traditional Logic Programming

The observation that motivated the development of Xcerpt is that Web data formats like XML describe tree or graph structures just like terms in logic programming. However, the usage of these terms differs in several important aspects from the terms used in traditional logic programming, which are discussed below.

Information Representation. In logic programming, a database usually consists of a set of facts, each of which comprises an alternative entry in the database.

In the Web, the concept of a database is usually much broader. Besides considering a collection of terms (or documents) as a database, it is very common to represent a complete database within a single term, where the individual entries are *subterms* of the database.

Structure. Whereas logic programming (and relational databases, for that matter) assumes very homogenous sets of data, databases on the Web are in general more flexible and data items of a similar kind often have a slightly different structure. For example, an address book might contain one address entry which has two email addresses and no phone, and another which has no email address but a phone as well as a mobile number.

Schema. Terms in logic programming follow a rather rigid schema, in which both the term label and the arity are fixed (i.e. $f\{a\}$ and $f\{a,b\}$ are instances of different schemas and a query for $f\{X\}$ would match only the first).

Semistructured databases as found on the Web are much more flexible in this respect, mostly due to the heterogeneous and constantly evolving nature of the Web. In particular,

- documents are not required to have a schema at all
- if a schema exists, they do not need to fully comply to it
- schema languages like XML Schema or RelaxNG [6,7] allow more flexible structures, where subterms might be optional, alternatives, or repeated an arbitrary number of times

For example, $f\{a\}$ and $f\{a,b\}$ might both be instances of the same schema and should thus both match with the query $f\{X\}$.

2.2 Partial Patterns and Grouping Constructs

To summarise, a Web query language like Xcerpt needs to fulfill the following requirements:

- it needs to be able to work with partial information about the queried document, as schema information might be missing or incomplete
- it needs to be able to query several alternatives *within the same document*, which might even differ in their structure.
- it needs to be able to construct new documents in the same manner, i.e. where several alternatives are grouped in the same document

Xcerpt addresses the first two requirements by extending the notion of terms to *partial patterns*. Such partial patterns allow the programmer to specify only the minimum information that is necessary for querying (e.g. in an address book, it is sufficient to specify the name to retrieve an entry). Partial patterns also allow to query several alternatives in a single term, as these can be identified with the different alternative ways of matching a partial pattern with the term (e.g. a partial query for $f\{X\}$ against a database $f\{a,b\}$ matches either with $X = a$ or with $X = b$).

The last requirement is addressed by the grouping constructs *all* and *some* which are similar in meaning to the Prolog predicates *setof* or *bagof* in that they collect all possible alternative solutions. Since grouping constructs are very frequently used in Web querying, Xcerpt includes them into the language itself rather than as external predicates. As a consequence, the proof calculi should support such grouping constructs directly, whereas Prolog works around this problem with meta reasoning. An example of an Xcerpt rule containing both grouping constructs and partial query patterns is given in Figure 1.

```
CONSTRUCT
  books {
     all book {
       var TITLE, price-a { var PRICEA }, price-b { var PRICEB } }
  }
FROM
  and {
    in { resource { "http://bn.com" },
      bib {{
        book {{ var TITLE ⇝ title{{}}, price { var PRICEA } }}
      }} },
    in { resource { "http://amazon.com" },
      reviews {{
        entry {{ var TITLE ⇝ title{{}}, price { var PRICEB } }}
      }} }
  }
WHERE
  or {
    var PRICEA < 40,
    var PRICEB < 40
  }
END
```

Fig. 1. An Xcerpt rule that queries two book databases and returns a list of book titles with price comparisons. Partial query patterns are indicated by double braces. A more detailed explanation of Xcerpt's syntax can be found in [1].

3 Simulation Unification

Simulation unification [8] is a non-standard, assymetric unification method that respects partial term specifications. Simulation unification is based on a relation called *simulation* which is a partial ordering on the set of terms. Intuitively, a term t_1 is simulated in a term t_2 if the structure of t_1 can be found in t_2 (see Figure 2).

Simulation unification of a partial term t_1 and a term t_2 computes a set of alternative substitutions for the variables in t_1 and t_2 such that the ground instance of t_1 simulates into the ground instance of t_2. For instance, simulation unification of the partial term $f\{X\}$ and the term $f\{a, b\}$ yields the two alternative substitutions $\sigma_1 = \{X = a\}$ and $\sigma_2 = \{X = b\}$.

4 Approaches to Proof Calculi for Xcerpt

The suggested calculi are inspired by the SLD resolution used in logic programming. However, traditional approaches like the SLD resolution do not account

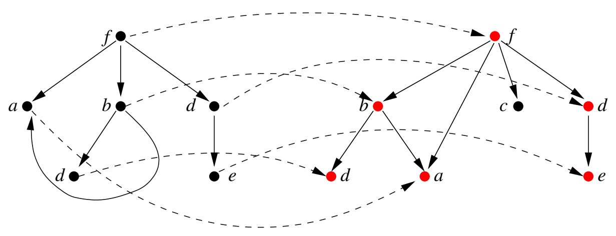

Fig. 2. A simulation between two graph representations of terms. Note that the subterm c is contained in the term on the right but not in the term on the left.

well for constructs like partial patterns or grouping constructs. Both kinds of constructs have implications on possible proof calculi.

High Branching Rate. In traditional logic programming, there are two elements of nondeterminism that lead to branching in the proof tree: selection of the predicate to unfold in the evaluation of a rule body, and the selection of the program rule used for further chaining. Xcerpt's usage of partial patterns adds a third element: When using partial patterns, there is in general no single way to match two terms. Instead, all possible alternative matchings have to be considered, which leads to a significantly higher branching rate.

Grouping Constructs all *and* some. Unlike Prolog's *setof* and *bagof* predicates, the grouping constructs *all* and *some* are an integral part of the language. It is hence desirable to support such higher order constructs in the proof calculus itself rather than treating them as external predicates.

This abstract gives a brief overview over possible approaches to proof calculi that are taking into account the above-mentioned issues. The remainder of this section introduces two approaches called "one at once" and "all at once", which differ in that "one at once" follows only a single proof path at a time (like SLD resolution), whereas "all at once" allows to follow a different proof path at each step, regardless of whether the previous path was finished or not.

In both approaches, the proof tree is represented as a formula of constraints, the *constraint store*. Such constraints are either *folded queries* (which may be unfolded by matching with the heads of rules) or *simulation constraints* (which specify that two terms have to be unified).

4.1 One at once

The "one at once" calculus is similar to the SLD resolution calculus with operational treatment of higher order predicates used in logic programming. Like SLD resolution, the calculus considers only a single path at a time. If a grouping construct occurs, the calculus interrupts the evaluation of the current path, visits each of the paths of the queries in scope of this grouping construct in turn and collects the respective solutions, and afterwards continues with the evaluation of the current path.

"One at once" has the advantage that it only needs to consider a single conjunctive path at a time. On the other hand, occurrences of grouping constructs externally "interrupt" the evaluation by recursive applications of the calculus to certain queries until all solutions are found.

4.2 All at once

The "all at once" calculus considers all paths in the proof tree at once. Thus, the considered constraint store contains conjunctions as well as disjunctions. Where "one at once" unfolds a query with only one of the alternatives at a time (and then relies on backtracking for finding different alternatives), "all at once" unfolds all possible alternatives simultaneously and adds them to the proof tree. If a grouping construct occurs, it adds a dependency constraint to a certain subtree of the proof tree. The evaluation may then continue at any node in the proof tree. If this subtree is completely solved, the grouping construct can be solved as well.

This approach has the advantage that higher order constructs are included more naturally into the calculus. Instead of relying on external control for solving higher order constructs, the dependency constraint can be treated by the rules of the calculus.

In addition, the possibility to continue at any node in the proof tree gives rise to interesting considerations about selection strategies. With a depth-first search, the calculus would resemble "one at once" or SLD resolution. Different search strategies might however be auspicious. A cost based A* search that tries to first select such nodes that contribute most to the result could provide performance benefits in practical applications, in particular in the context of the Web where IO costs for remote resources are often considerably higher than for local or even in-memory resources.

5 Related Work and Conclusion

This abstract gives a short overview over issues and problems of applying techniques used in logic programming to the Web query language Xcerpt. Two different approaches for treating Xcerpt's built-in higher level constructs *all* and *some* have been presented.

The language Xcerpt is work in progress. An comprehensive introduction into the language Xcerpt with many examples can be found in [1]. The simulation unification algorithm has first been presented at [8]. A declarative semantics in form of a model theory in the style of classical logic is currently being worked on and first results have been published in [9]. A prototype of Xcerpt exists and has been demonstrated at [10].

Xcerpt is not the only rule-based query language for Web data. Most noteably, the language UnQL [11] first introduced the concept of rule-based querying to the XML world, but it does not provide important features like rule chaining and is not based on logic programming.

The necessity of higher order predicates like *setof* and *bagof* in Prolog have been discussed in numerous articles (see e.g. [12]). Also, a formal semantics has been considered e.g. in [13]. However, such considerations in general do not include support for higher order constructs into the calculus itself but instead treat them as external predicates.

References

1. Bry, F., Schaffert, S.: A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In: Proc. Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web (RuleML' 02). (2002) (invited article).
2. W3C: XQuery: A Query Language for XML. (2001)
3. W3C: Extensible Stylesheet Language (XSL). (2000)
4. W3C: Web Ontology Language (OWL). (2003)
5. W3C: Resource Description Framework (RDF). (1999)
6. W3C: XML Schema Part 0: Primer; Part 1: Structures, Part 2: Datatypes. (2001)
7. Clark, J., Murata, M.: RELAX NG Specification, http://relaxng.org/spec-20011203.html. (2001) ISO/IEC 19757-2:2003.
8. Bry, F., Schaffert, S.: Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In: Proc. Int. Conf. on Logic Programming. LNCS 2401, Springer-Verlag (2002)
9. Bry, F., Schaffert, S.: An entailment relation for reasoning on the web. In: Proc. Int. Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML'03). LNCS 2876, Sanibel Island, Florida, USA, Springer-Verlag (2003)
10. Berger, S., Bry, F., Schaffert, S., Wieser, C.: Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data. In: Proc. Intl. Conference on Very Large Databases (VLDB03) – Demonstrations Track, Berlin, Germany (2003)
11. Buneman, P., Fernandez, M., Suciu, D.: UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. VLDB Journal **9** (2000)
12. Warren, D.H.D.: Higher-order extensions to prolog: Are they needed? In Hayes-Roth, M., Pao, eds.: Machine Intelligence. Volume 10. Ellis Horwood (1982)
13. Börger, E., Rosenzweig, D.: The mathematics of set predicates in prolog. In: Kurt Godel Colloquium. (1993) 1–13