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 (Revised Version of PMS-FB-2004-5)

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-2004-5, February 2004

A contribution to the Semantics of Xcerpt, a Web Query and Transformation Language

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 article 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 $f\{\{X\}\}$.

Note that this article uses for simplicity reasons a reduced syntax, in which terms are limited to the curly braces { }. Curly braces denote that the order of subterms is irrelevant. The full language Xcerpt [1] also allows a so called ordered term specification with square brackets [ ], which is a more precise representation for XML documents as they are always ordered.

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* (expressed by double curly braces as in $f\{\{a\}\}$) and by the *descendant* construct (expressed by the keyword *desc* as in $f\{\{desc\ a\}\}$). 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 descendant construct allows to match a pattern at arbitrary depth (e.g. a partial query for $f\{\{desc\ X\}\}$ against the database $f\{g\{a\}, h\{b\}\}$ matches with $X = g\{a\}$, with $X = h\{b\}$, 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 2.

Xcerpt has many constructs that are not covered here for space reasons. A more detailed introduction into Xcerpt can e.g. be found in [1].

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <authors>
      <author>
        <last>Stevens</last>
        <first>W.</first>
      </author>
    </authors>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1992">
    <title>
      Advanced Programming ...
    </title>
    <authors>
      <author>
        <last>Stevens</last>
        <first>W.</first>
      </author>
    </authors>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="2000">
    <title>Data on the Web</title>
    <authors>
      <author>
        <last>Abiteboul</last>
        <first>Serge</first>
      </author>
      <author>
        <last>Buneman</last>
        <first>Peter</first>
      </author>
      <author>
        <last>Suciu</last>
        <first>Dan</first>
      </author>
    </authors>
    <publisher>Morgan Kaufmann</publisher>
    <price>39.95</price>
  </book>
  ...
</bib>
```

```
<reviews>
  <entry>
    <title>Data on the Web</title>
    <price>34.95</price>
    <review>
      A very good discussion of semi-
      structured database systems and XML.
    </review>
  </entry>
  <entry>
    <title>
      Advanced Programming
    </title>
    <price>65.95</price>
    <review>
      A clear and detailed discussion of
      UNIX programming.
    </review>
  </entry>
  <entry>
    <title>TCP/IP Illustrated</title>
    <price>65.95</price>
    <review>
      One of the best books on TCP/IP.
    </review>
  </entry>
  ...
</reviews>
```

Fig. 1. Two bookstore databases with different structures but similar contents. Note that several alternative entries are contained within the same document and how `book` entries in the left database differ slightly in structure.

```
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. 2. An Xcerpt rule that queries two book databases (given in Figure 1) and returns a list of book titles with price comparisons (given in Figure 3). Partial query patterns are indicated by double braces. A more detailed presentation of Xcerpt can be found in [1].

```
<books>
  <book>
    <title>TCP/IP Illustrated</title>
    <price-a>65.95</price-a>
    <price-b>65.95</price-b>
  </book>
  <book>
    <title>Advanced Programming ...</title>
    <price-a>65.95</price-a>
    <price-b>65.95</price-b>
  </book>
  <book>
    <title>Data on the Web</title>
    <price-a>39.95</price-a>
    <price-b>34.95</price-b>
  </book>
</books>
```

Fig. 3. The XML document resulting from the evaluation of the Xcerpt rule in Figure 2. For each book, the element `price-a` contains the price of the first database of Figure 1, the element `price-b` the price from the second database.

3 Simulation Unification

Simulation unification [8] is a non-standard, asymmetric 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 4).

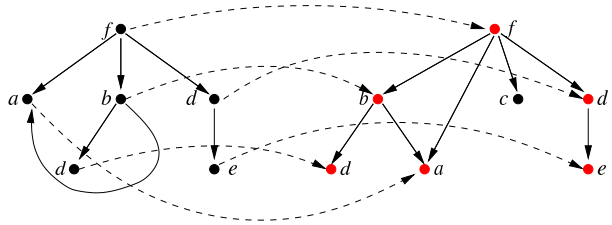

Fig. 4. A simulation between two graph representations of terms. Note that the subterm c is contained in the term on the right but not in the term on the left.

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 $\sigma_2 = \{X = b\}$.

The simulation unification algorithm is specified in terms of constraint reduction rules that operate on a constraint store initialised with $t_1 \preceq_S t_2$ (meaning that t_1 should simulation unify into t_2, i.e. after adequate variable bindings t_1 should simulate into t_2). All unification rules decompose a single constraint to a formula containing conjunctions and/or disjunctions of smaller constraints, until no further decomposition is possible (i.e. until either the left or the right side consists of a variable, or a constraint is reduced to one of the boolean values *true* or *false*). If no further rule is applicable, simulation unification creates a set of substitutions by computing the disjunctive normal form of the constraint store, and by replacing all constraints of the form $X \preceq_S t$ by $X = t$. Each disjunct in the disjunctive normal form is an alternative substitution.

It is assumed that the constraint store applies simplification rules as needed (e.g. remove conjunctions that contain a boolean value *false*). Furthermore, the following rule enforces consistency between different constraints for the same variable and ensures that after the evaluation there exists only a single upper bound for each variable.

$$\frac{X \preceq_S t_1 \wedge X \preceq_S t_2}{X \preceq_S t_1 \wedge t_1 \preceq_S t_2 \wedge t_2 \preceq_S t_1}$$

In case that the two bounds for the variable (t_1 and t_2) are inconsistent, i.e. cannot be unified, one of the constraints $t_1 \preceq_S t_2$ or $t_2 \preceq_S t_1$ is reduced to *false* in further evaluation steps.

3.1 Decomposition Rules

Root Elimination Root elimination rules compare the roots of the two terms and distribute the unification to the children.

Left Term without Children This set of rules consider all such cases where the left term does not contain child elements. These cases have to be treated separately from the general decomposition rules below as this would yield the wrong result. For instance, an empty *or* is equivalent to *False* but the result should always be *True* in case the left term is only a partial specification. In the following, let $m \geq 0$ and $k \geq 1$:

$$\frac{l\{\{\ \}\} \preceq_S l\{t_1^2, \ldots, t_m^2\}}{True} \qquad \frac{l\{\ \} \preceq_S l\{t_1^2, \ldots, t_k^2\}}{False} \qquad \frac{l\{\ \} \preceq_S l\{\ \}}{True}$$

As specified by these rules, a term without children, but with a partial specification (double braces) matches with any term which has the same label. If the term specification is not partial, it matches only with such terms that also do not have subterms.

Decomposition The general decomposition rule eliminates the two root nodes in parallel and distributes the unification to the various combinations of children that result from total/partial specification. If there exists no such combination, then the result is an empty *or*, which is equivalent to *False*.

In the following, let $n, m \geq 1$, and, given two terms $l\{t_1^1, \ldots, t_n^1\}$ and $l\{t_1^2, \ldots, t_m^2\}$, let $\Pi, \Pi_{surj} : \{1, \ldots, n\} \to \{1, \ldots, m\}$ be defined as follows:

- Π is the set of all total, injective functions from $\{1, \ldots, n\}$ to $\{1, \ldots, m\}$.
- Π_{surj} is the set Π restricted to all surjective functions

$$\frac{l\{\{t_1^1, \ldots, t_n^1\}\} \preceq_S l\{t_1^2, \ldots, t_m^2\}}{\bigvee_{\pi \in \Pi} \bigwedge_{1 \leq i \leq n} t_i^1 \preceq_S t_{\pi(i)}^2} \qquad \frac{l\{t_1^1, \ldots, t_n^1\} \preceq_S l\{t_1^2, \ldots, t_m^2\}}{\bigvee_{\pi \in \Pi_{surj}} \bigwedge_{1 \leq i \leq n} t_i^1 \preceq_S t_{\pi(i)}^2}$$

For instance, if the left term has a partial specification for the subterms, the simulation unification has to consider as alternatives all combinations of children from the left term with children from the right term, provided that each child on the left gets a matching partner on the right.

Label Mismatch In case of a label mismatch, the unification fails. In the following, let $l_1 \neq l_2$.

$$\frac{l_1\{\{t_1^1, \ldots, t_n^1\}\} \preceq_S l_2\{t_1^2, \ldots, t_m^2\}}{False} \qquad \frac{l_1\{t_1^1, \ldots, t_n^1\} \preceq_S l_2\{t_1^2, \ldots, t_m^2\}}{False}$$

Descendant Elimination The descendant construct is eliminated by adding a disjunction of constraints, which express that the current term t_2 is matched by t_1 or that at least one of the subterms of t_2 is matched by *desc* t_1, thus distributing the decomposition to the children of t_2. Let $m \geq 0$.

$$\frac{\text{desc } t^1 \preceq_S l_2\{t_1^2, \ldots, t_m^2\}}{t^1 \preceq_S l_2\{t_1^2, \ldots, t_m^2\} \vee \bigvee_{1 \leq i \leq m} \mathit{desc}\, t^1 \preceq_S t_i^2}$$

4 Two 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 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 article 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.

4.1 Common Properties

The evaluation of the two approaches yields a *set of substitutions* which is constructed in almost the same manner as for simulation unification above. In both approaches, the proof tree is represented as a formula of constraints, the *constraint store*. Such constraints are one of

- *folded queries* represent query parts that have not yet been evaluated (e.g. a query pattern or a conjunction of query patterns) and are expressed as $\langle Q \rangle$.
- *simulation constraints* specify that two terms t_1 and t_2 have to be unified and are expressed as $t_1 \preceq_S t_2$,
- *dependency constraints* specify that the evaluation of one constraint depends on the evaluation of another and are expressed as $(C_1 \mid C_2)$.

Furthermore, the following notations are used:

- $\mathcal{P}_{\text{grouping}}$ denotes the set of all rules in the program $\mathcal{P}$ that contain one of the grouping constructs *all* or *some*
- $\mathcal{P}_{\text{nongrouping}}$ denotes the set of all rules in the program $\mathcal{P}$ that do not contain one of the grouping constructs *all* or *some*
- $\mathcal{T}$ denotes the set of all database terms contained in the program, or referenced by resource specifications

The following constraint reduction rules are also common to both approaches:

Dependency Resolution. The dependency resolution is required for computations that involve the *all* and *some* constructs. A dependency constraint of the form $(t_1 \preceq_S t_2 \mid C_2)$ requires to evaluate the complete proof tree (in case of *all*) or parts of the proof tree (in case of *some*) of C_2 before C_1, and applies the resulting substitution Σ to t_2 (application to t_1 is not necessary, as t_1 and C_2 are variable disjunct).

$$\frac{(\ t_1 \ \preceq \ t_2 \ \mid \ D\ )}{\bigvee_{t_2' \in \Sigma(t_2)} \ t_1 \ \preceq \ t_2'} \quad \Sigma = \mathit{subst}(\mathit{solveall}(D))$$

The actual implementation of the *solveall* function depends on whether the "one at once" or "all at once" algorithm is used. In the "one at once" algorithm, solveall evaluates all paths in the proof tree. In the "all at once" approach, solveall evaluates the complete constraint store.

4.2 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" consists of three unfolding rules which are introduced below:

Query Unfolding against Database Term. Unfold a folded query term against a database term t by replacing the folded query term by a simulation constraint between the folded query term and the term t.

$$\frac{\langle t^q \rangle \quad t \in \mathcal{T}}{t^q \preceq_S t}$$

Query Unfolding against Rule. Unfold a folded query term t^q against the head t^c of a rule.

1. In case t^c contains none of the grouping constructs *all* and *some*, add a constraint for the simulation of t^q in t^c and add the query part of the rule as a folded query.
2. In case t^c contains at least one of the grouping constructs *all* and *some*, add a dependency constraint such that the unification of t^q in t^c is only evaluated in case the query part is evaluated successfully.

$$\frac{\langle t^q \rangle \quad (t^c \to Q) \in \mathcal{P}_{\text{nongrouping}}}{t^q \preceq_S t^c \wedge \langle Q \rangle} \qquad \frac{\langle t^q \rangle \quad (t^c \to Q) \in \mathcal{P}_{\text{grouping}}}{(t^q \preceq_S t^c \mid \langle Q \rangle)}$$

The dependency part in a dependency constraint (as in the result of the right rule) is solved in an auxiliary calculation. In case t^c contains an *all* construct, or nested *some* constructs, it is necessary to solve the complete query part. If t^c contains only a single *some* construct, it is sufficient to only search for solutions until a sufficient amount is found.

Disjunctive Split. Note that all of these rules need to select both a folded query to continue with and either a rule or a term, and backtrack in case the selected rule or term leads to failure. This selection with backtracking yields a so-called *proof tree*. Both the selection of constraints and of rules/terms is non-deterministic and different search strategies, like the depth-first search used in SLD resolution, are conceivable.

Some of the rules above may yield a disjunction as a result (most notably the dependency resolution and the unification part of the consistency verification). In such cases, the "one at once" approach needs to split the disjunction into different paths of the proof tree (and insert a choice point). The following rule represents this split. Assume that C is in disjunctive normal form:

$$\frac{C_1 \vee \ldots \vee C_n}{C_1 \mid \ldots \mid C_n}$$

"One at once" has the advantage that it only needs to consider a single conjunctive path at a time. On the other hand, only a depth first search is possible and occurrences of grouping constructs externally "interrupt" the evaluation by requiring an auxiliary application of the calculus to certain queries until all solutions are found.

4.3 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.

$$\frac{\langle t^q \rangle}{\begin{array}{l} \bigvee_{t \in \mathcal{T}} t^q \preceq_S t \quad \vee \\ \bigvee_{(t^c \leftarrow Q) \in \mathcal{P}_{\text{nongrouping}}} (t^q \preceq_S t^c \wedge \langle Q \rangle) \vee \\ \bigvee_{(t^c \leftarrow Q) \in \mathcal{P}_{\text{grouping}}} (t^q \preceq_S t^c \mid \langle Q \rangle) \end{array}}$$

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.

As the "all at once" approach works with both conjunctions and disjunctions of contraints, a further interesting aspect is to integrate the evaluation of the rule chaining with the evaluation of the simulation unification. Doing so might allow optimisations of the evaluation, e.g. by interleaving chaining and unification steps when feasible.

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. A project website is located at `http://www.xcerpt.org`. 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