Type Inference and Rule Dependencies in Xcerpt

Włodzimierz Drabent[1,2], Artur Wilk[1]

May 2007

[1]Dept. of Computer and Information Science,
Linköping University, S 581 83 Linköping, Sweden
[2]Institute of Computer Science, Polish Academy of Sciences,
ul. Ordona 21, Pl – 01-237 Warszawa, Poland
{wdr, artwi}@ida.liu.se

Abstract. We present a type system for a substantial fragment of the Web query language Xcerpt. It is a descriptive type system: the typing of a program is an approximation of its semantics. This paper augments the previous work on typing single Xcerpt rules, by a type inference method for possibly recursive Xcerpt programs (consisting of many rules).
The method may be seen as abstract interpretation. We provide a way to make fixed point computations finite, and to estimate in advance the number of iterations needed to obtain a fixed point. The latter is necessary, as a test for a fixed point is too expensive.
We also show how the obtained type information can be used for discovering dependencies between rules. We expect that besides typical usage of a type system, such as discovering type errors, the presented methods can be used for improving efficiency of program evaluation.

1 Introduction

This paper presents a type system for a substantial fragment of the Web query language Xcerpt [13,12]. It is a descriptive type system: the typing of a program is an approximation of its semantics. In particular, types are sets (of data objects). The type system makes possible type derivation (computing an approximation of the set of the results of a program applied to data from a given set) and type checking (finding whether the results are included in a specified set of allowed results). The intended application is to help the programmer in finding errors in programs, and to provide a static analysis tool for efficient Xcerpt implementation.

Our previous work [2,3,8,14] was focused on typing a single Xcerpt rule. In particular, recursive programs were not dealt with. The type system was expressed by an abstract form of a type inference algorithm. A correctness proof [3] and a prototype implementation [14,16] were provided. This paper presents a type inference method for multiple rule Xcerpt programs, including recursive ones. It employs the methods of typing single rules from the previous work.

Our approach can be seen as an instance of the abstract interpretation paradigm [7]. We deal with two immediate consequence operators; one describes the concrete semantics of Xcerpt, the other describes semantic approximations, expressed by means of types. A fixed point of the latter operator is an approximation of the semantics of the program. Generally, a fixed point cannot be obtained by finite iteration of the operator. We present a way of computing an approximation of the

semantics by a finite computation. For this we introduce an appropriate operator on types, for which a fixed point is obtained in a finite number of iterations. As it is too expensive to check whether a fixed point has been reached, we provide a way to find out the required number of iterations in advance.

We also show how type inference can be used to discover dependencies between rules. In many cases the proposed approach provides a dependency graph which is more precise than those of previous approaches [12,14]. Thus we expect it to be usable, especially in combination with the previous approaches, in constructing more efficient Xcerpt implementations. We expect that the knowledge on rule dependencies can be employed in scheduling evaluation of rules, and for optimizations based on knowing the form of data passed between rules [9].

The paper is organized as follows. Section 2 introduces the semantics of Xcerpt and a formalism for type specification. Section 3 presents a method for type inference for rules in Xcerpt programs. Section 4 describes a way of approximating rule dependencies in programs. Finally, Section 5 contains a short summary. We also provide an appendix with proofs of the theorems of this paper.

2 Preliminaries

2.1 Xcerpt – Introduction

This section is a short and informal introduction to Xcerpt, a rule based query and transformation language for XML [13,12]. Xcerpt is inspired by logic programming and in contrast to many other XML query languages it uses patterns instead of path expressions. In the next section will present a formal semantics of Xcerpt. For any issues not explained here, see [12] or [2,14].

Informally an Xcerpt program is a set of **query rules** of the form $c \leftarrow Q$ consisting of a body and a head. The body of a rule is a query intended to match data terms. If the query contains variables such matching results in answer substitutions for variables. The head uses the substitutions to construct new data terms. The queried data is either specified in the body or is produced by rules of the program. There are two kinds of query rules: goal rules produce the final output of the program, while construct rules produce intermediate data, which can be further queried by other rules (or the same rule).

XML data is represented in Xcerpt as **data terms**. Data terms are built from basic constants and labels using two kinds of parentheses: brackets [ ] and braces { }. Basic constants represent basic values such as attribute values and character data (#PCDATA of XML). A label represents an XML element name. The parentheses following a label include a sequence of data terms (its direct subterms). Brackets are used to indicate that the direct subterms are ordered (in the order of their occurrence in the sequence), while braces indicate that the direct subterms are unordered. The latter alternative is used to encode attributes of an XML element by a data term of the form $attr\{l_1[v_1], \ldots, l_n[v_n]\}$ where each l_i is a name of an attribute and v_i is its respective value.

Example 1. This is an XML element and the corresponding data term.

```
<CD price="9.90">
  <title>Empire Burlesque</title>
  <artist>Bob Dylan</artist>
</CD>
```

CD[*attr*{ *price*["9.90"] },
title["Empire Burlesque"],
artist["Bob Dylan"]
] □

There are two other kinds of terms in Xcerpt: query terms and construct terms.

Query terms are (possibly incomplete) patterns which are used in a rule body (query) to match data terms. In particular, every data term is a query term. Generally query terms may include variables so that a successful matching binds variables of a query term to data terms[1]. Such bindings are called answer substitutions. A result of a query term matching a data term is a set of answer substitutions. For example a query term $a[\,\mathtt{b},\ X\,]$ matches a data term $a[\,\mathtt{b},\mathtt{c}\,]$ resulting in the answer substitution set $\{\{X/\mathtt{c}\}\}$. Query terms can be ordered or unordered patterns, denoted, respectively, by brackets and braces. For example, a query term $a[\,\mathtt{c},\mathtt{b}\,]$ is an ordered pattern and it does not match a data term $a[\,\mathtt{b},\ \mathtt{c}\,]$ but a query term $a\{\,\mathtt{c},\ \mathtt{b}\,\}$, which is an unordered pattern, matches $a[\,\mathtt{b},\ \mathtt{c}\,]$. Query terms with double brackets or braces are incomplete patterns. For example a query term $a[[\,\mathtt{b},\mathtt{d}\,]]$ is an incomplete pattern which matches a data term $a[\,\mathtt{b},\ \mathtt{c},\ \mathtt{d}\,]$ (because $\mathtt{b},\mathtt{d}$ is a subsequence of $\mathtt{b},\mathtt{c},\mathtt{d}$). As the query term uses brackets the matching subterms of the data term must occur in the same order as in the pattern. Thus the query term $a[[\,\mathtt{b},\mathtt{d}\,]]$ does not match a data term $a[\,\mathtt{d},\ \mathtt{b},\ \mathtt{c}\,]$. In contrast a query term $a\{\{\mathtt{b},\ \mathtt{d}\}\}$ does. To specify subterms at arbitrary depth a keyword `desc` is used e.g. a query term `desc d` matches a data term $a[\,b[\,\mathtt{d}\,],\mathtt{c}\,]$.

A query term q in a rule body may be associated with a resource r storing XML data or data terms. This is done by a construct of the form $\mathtt{in}[\,r,\ q\,]$, called targeted query term. Its meaning is that q is to be matched against data in r. Query terms in the body of a rule which have no associated resource are matched against data generated by rules of the Xcerpt program.

Queries are constructed from (targeted) query terms using logical connectives such as `or`, `and`, and `not`. A rule body is a query.

A query term q occurring in a query Q is a top query term if it is standalone in Q i.e. it is not a part of a (targeted) query term. For example, $a[\,b[\,X\,]\,]$ is the only top query term of the query $\mathtt{and}[\,\mathtt{in}[\,"www.ex.com/ex1.xml",\ c[\,X\,]\,],\ a[\,b[\,X\,]\,]\,]$.

Construct terms are used in rule heads to construct new data terms. They are similar to data terms, but may contain variables. Data terms are constructed out of construct terms by applying answer substitutions obtained from a rule body. Construct terms may also use **grouping constructs** `all` and `some` which are used to collect all or, respectively, some instances that result from different answer substitutions.

[1] In our examples variables are denoted by letters X, Y, Z and we skip the keyword `var` used in Xcerpt to denote variables. We also skip quotation marks for basic constants. For example, a basic constant "a" will be denoted as `a`.

Example 2. This is an extension of the "Clique of Friends" example from [12]. Consider an XML document *addrBooks.xml* represented by a data term:

```
addr-books[
  addr-book[ owner[ Donald Duck ],
             entry[ name[ Daisy Duck ], relation[ friend ], phoneNo[ +112345 ],
                    address[ street[ Hayes 51 ], zip-code[ 21213 ],
                             city[ Los Angeles ], country[ USA ] ] ],
             ...,
             entry[...] ],
  ...,
  addr-book[...] ]
```

The document is a collection of address books where each address book has its owner and a set of entries with information about people the owner knows. The information contains an annotation about the relation between the owner and the particular person such as: friend, colleague, family. The following Xcerpt program extracts a relation *friend of a friend* (*foaf*) which is the transitive closure of a relation *friend of* (*fo*). The relation *fo* is computed by the rule p_1 and its transitive closure is computed by the recursive rule p_2. The third rule g, which is a goal, returns a data term with a sequence of pairs representing the relation *foaf*.

$p_1 =$ *fo*[*X*, *Y*] ← `in`["*file*:*addrBooks.xml*",
addr-books{{ *addr-book*{{ *owner*[*X*], *entry*{{ *name*[*Y*], *relation*[friend] }} }} }}]

$p_2 =$ *foaf*[*X*, *Y*] ← `or`[*fo*[*X*, *Y*], `and`[*fo*[*X*, *Z*], *foaf*[*Z*, *Y*]]]

g = *clique-of-friends*[`all` *foaf*{ *X*, *Y* }] ← *foaf*[*X*, *Y*] □

2.2 Formal Semantics of Xcerpt

Now we present a formal semantics with respect to which our typing approach will be proved sound. We define the semantics of Xcerpt programs given a semantics of single query rules. We employ the semantics of single rules from [2,14], thus we neglect the constructs of Xcerpt not dealt with by the semantics. A main restriction is that our data terms represent trees while in full Xcerpt they are used to represent graphs.

Definition 1 (Xcerpt program). *An Xcerpt program $\mathcal{P}$ is a pair (P, G) where P and G are sets of query rules such that $G \subseteq P$ and $|G| > 0$. The query rules from G are called goals.*

Let p be a query rule and let for each URI r_i of an external resource in p, $d(r_i)$ be a data term associated with r_i. The query rule p queries each $d(r_i)$ and a set of data terms Z produced by query rules of a program. The set of results of the query rule p and the set of data terms Z is denoted as $res(p, Z)$ and defined in [14] by Definition 10. Now we are ready to describe the effect of applying a set of rules to a set of data terms, and then the semantics of a program.

Definition 2 (Immediate consequence operator for rule results). *Let P be a set of Xcerpt query rules. R_P is a function on sets of data terms such that $R_P(Z) = Z \cup \bigcup_{p \in P} res(p, Z)$.*

Definition 3 (Rule result, no grouping constructs). *Let $\mathcal{P} = (P, G)$ be an Xcerpt program without grouping constructs and $P' = P \backslash G$. Given fixed data terms $d(r_i)$ associated with external resources occurring in the rules from P, a data term d is a* **result of a rule** *p in P if $d \in res(p, R^i_{P'}(\emptyset))$ for some $i \geq 0$.*

A **result of a program** *P is a data term which is a result of a goal of P.*

Example 3. Let $P' = \{ p \}$, where $p = c[X] \leftarrow \texttt{or}[X, \texttt{in}[r, b[X]]$ and $d(r) = b[\mathsf{a}]$. $R^i_{P'}(\emptyset) = \{ c[\mathsf{a}], c[c[\mathsf{a}]], \ldots, c^i[\mathsf{a}] \}$, for $i > 0$, and $res(p, R^i_{P'}(\emptyset)) = \{ c[\mathsf{a}], \ldots, c^{i+1}[\mathsf{a}] \}$. □

To define semantics for programs with grouping constructs we employ a notion of static rule dependency to split programs into strata. This notion is equivalent to the rule dependency used in [12]. We also introduce a weaker kind of dependency, as the static dependency does not reflect some issues related to types.

Definition 4 (Static rule dependency). *Let $\mathcal{P} = (P, G)$ be an Xcerpt program. A rule $c \leftarrow Q \in P$* directly statically depends *on a rule $c' \leftarrow Q' \in P \backslash G$, if a top query term from Q matches some instance of the construct term c'. The fact that a rule p directly statically depends on a rule p' is denoted as $p \succ_s p'$.*

A rule $p \in P$ statically depends *on a rule $p' \in P \backslash G$ if $p \succ_s^+ p'$ (where $\succ_s^+$ is the transitive closure of $\succ_s$ i.e. $p \succ_s^+ p'$ if $p \succ_s p_1 \succ_s \ldots \succ_s p_k \succ_s p'$ for some rules $p_1, \ldots, p_k$ in $P \backslash G$ where $k \geq 0$).*

Definition 5 (Weak static rule dependency). *Let $\mathcal{P} = (P, G)$ be an Xcerpt program. A rule $c \leftarrow Q \in P$* directly weakly statically depends *(shortly, directly w-depends) on a rule $c' \leftarrow Q' \in P \backslash G$, if a top query term from Q matches some instance of the construct term c'', where c'' is c' with every occurrence of a variable replaced by a distinct variable. The fact that a rule p directly w-depends on a rule p' is denoted as $p \succ_w p'$.*

A rule $p \in P$ weakly statically depends *(shortly, w-depends) on a rule $p' \in P \backslash G$ if $p \succ_w^+ p'$. The program $\mathcal{P}$ is weakly statically recursive (shortly, w-recursive) if $p \succ_w^+ p$ for some $p \in P$. We also say that $P \backslash G$ is w-recursive.*

Static dependency between rules implies weak static dependency.

Example 4. Consider the following query rules of an Xcerpt program:

$$p_1 = a[Y] \leftarrow b[\mathsf{d}, \mathsf{e}, Y], \quad p_2 = b[X, X, Y] \leftarrow c[X, Y].$$

It holds: $p_1 \succ_w p_2$ but $p_1 \not\succ_s p_2$. □

A simple algorithm for finding w-dependencies can be obtained by a slight modification of the typing rules for query terms presented in [2,14]. We skip the details.

Now we generalize the semantics of Definition 3 to programs with grouping constructs. The semantics is used in the proofs but not referred to explicitly in the paper. Thus the rest of this section, except for Definition 8, may be skipped at the first reading.

If a query rule p in a program contains a grouping construct, it can be executed only after all data terms queried by p have been obtained. This is ensured in the following way. The query rules of the program are divided into sets called strata.

A query rule p' with a grouping construct can statically depend only on rules from a lower stratum. Hence no rule of the same stratum as p' can produce data that can be queried by p'. Moreover, for an arbitrary rule p, no rule of a higher stratum than p can produce data that can be queried by p. The rules from a given stratum are not executed until the execution of the rules from lower strata is completed.

Definition 6 (Stratification). *Let $\mathcal{P} = (P, G)$ be an Xcerpt program and $P_1, \ldots, P_n$ ($n \geq 0$) be disjoint sets of query rules such that $P \backslash G = P_1 \cup \ldots \cup P_n$. The sequence $P_1, \ldots, P_n, G$ is a stratification of $\mathcal{P}$ if for any pair of rules $p, p' \in P \backslash G$, if $p \succ_s^+ p'$ then $p \in P_i$ and $p' \in P_j$, where $1 \leq j \leq i \leq n$ and if p has a grouping construct in its head then $j < i$.*

Any program (P, G) without grouping constructs is stratifiable and its stratification is $P \backslash G, G$. As in [12] we assume that we deal with stratifiable programs.

Example 5. Consider a program $\mathcal{P} = (\{p_1, p_2, p_3, g\}, \{g\})$, where:

$$g = h[\,X\,] \leftarrow a[\,X\,], \qquad p_1 = a[\,\texttt{all}\; X\,] \leftarrow b[\,X\,],$$
$$p_2 = b[\,Y\,] \leftarrow c[\,f[\,Y\,]\,], \quad p_3 = c[\,Y\,] \leftarrow \texttt{in}[\,r_1, Y\,].$$

A sequence $\{p_2, p_3\}, \{p_1\}, \{g\}$ is a stratification of $\mathcal{P}$. □

Definition 7 (Rule result). *Let $\mathcal{P} = (P, G)$ be an Xcerpt program, $P_1, \ldots, P_n, G$ be a stratification of $\mathcal{P}$. Let $Z_0 = \emptyset$ and, for $j = 1, \ldots, n$, let $Z_j = R_{P_j}^{l_j}(Z_{j-1})$ for such $l_j > 0$ that $R_{P_j}^{l_j}(Z_{j-1}) = R_{P_j}^{l_j+1}(Z_{j-1})$. Given fixed data terms $d(r_i)$ associated with external resources occurring in the rules from P, a data term d is a* **result of a rule** *p in P if $d \in res(p, Z_n)$. A* **result of a program** *$\mathcal{P}$ is a result of a goal rule $p \in G$ in P.*

According to this definition, if the program loops (i.e. $R_{P_j}^{i+1}(Z_{j-1}) \neq R_{P_j}^{i}(Z_{j-1})$ for some j and every $i = 1, 2, \ldots$) then $Z_j, \ldots, Z_n$ do not exist and no result exist, for any p in P. For simplicity reasons we do not provide a more sophisticated definition describing results of a looping program (i.e. those obtained before the program enters the infinite loop). A slight generalization of Def. 6 makes it possible to describe the semantics of Xcerpt negation, omitted here due to lack of space.

Having defined the set of rule results we can introduce dynamic rule dependency which, in a sense, describes the data flow of an Xcerpt program.

Definition 8 (Dynamic rule dependency). *Let $\mathcal{P} = (P, G)$ be an Xcerpt program and $c \leftarrow Q \in P$ and $p \in P \backslash G$ be query rules. The rule $c \leftarrow Q$* directly dynamically depends *on p (which is denoted as $p \succ p'$), if a top query term from Q matches a result of p in P.*

A rule $p \in P$ dynamically depends *on a rule $p' \in P \backslash G$ if $p \succ^+ p'$.*

It follows that $p \succ p'$ implies both $p \succ_s p'$ and $p \succ_w p'$.

Example 6. Consider the rules p_2, p_3 from Example 5: $p_2 \succ p_3$ iff the data term $d(r_1)$, specified by the URI r_1, is of the form $f[\,t\,]$ (for some data term t), while $p_2 \succ_s p_3$ independently from $d(r_1)$.

2.3 Type Definitions

Here we present a formalism for specifying a class of decidable sets of data terms representing XML documents. The formalism, called *type definitions* [15,5] is similar to unranked tree automata [4,11] and other related formalisms employed for XML processing languages such as XDuce [10], CDuce [1] or XCentric [6]. A novelty of our approach is that we deal with data terms representing mixed trees where the order of children of a node may be irrelevant. The types defined by type definitions roughly correspond to the sets of documents defined by various XML schema languages.

First we specify a set of **type names** $\mathcal{T} = \mathcal{C} \cup \mathcal{S} \cup \mathcal{V} \cup \{Top\}$ which consists of **type constants** from the alphabet $\mathcal{C}$, **enumeration type names** from the alphabet $\mathcal{S}$, **type variables** from the alphabet $\mathcal{V}$, and a type name *Top*.

A type definition associates type names with sets of data terms. The set $[\![T]\!]$ associated with a type name T is called the **type** denoted by T (or simply type T). For T being a type constant or an enumeration type name, the elements of $[\![T]\!]$ are basic constants. The type $[\![Top]\!]$ is the set of all data terms.

Type constants correspond to basic types of XML schema languages such as *String* or *Integer*. The set of type constants is fixed and finite; for each type constant $T \in \mathcal{C}$ the set of basic constants $[\![T]\!]$ is fixed.

A *regular expression* over an alphabet Σ is ε, ϕ, any $a \in \Sigma$ and any $r_1 r_2$, $r_1|r_2$ and r_1^*, where r_1, r_2 are regular expressions. A language $L(r)$ of strings over Σ is assigned to each regular expression r in a standard way: $L(\phi) = \emptyset$, $L(\varepsilon) = \{\epsilon\}$, where ϵ is the empty string, $L(a) = \{a\}$, $L(r_1 r_2) = L(r_1)L(r_2)$, $L(r_1|r_2) = L(r_1) \cup L(r_2)$, and $L(r_1^*) = L(r_1)^*$.

Definition 9. *A* **regular type expression** *is a regular expression over the alphabet of type names $\mathcal{T}$. We abbreviate a regular expression $r^n|r^{n+1}|\cdots|r^m$, where $n \leq m$, as $r^{(n:m)}$, $r^n r^*$ as $r^{(n:\infty)}$, rr^* as r^+, and $r^{(0:1)}$ as $r^?$. A regular type expression of the form*

$$T_1^{(n_{1,1}\,:\,n_{1,2})} \cdots T_k^{(n_{k,1}\,:\,n_{k,2})}$$

where $k \geq 0$, $0 \leq n_{i,1} \leq n_{i,2} \leq \infty$ for $i = 1, \ldots, k$, and $T_1, \ldots, T_k$ are distinct type names, will be called a **multiplicity list**.

Multiplicity lists will be used to specify multisets of type names.

Definition 10. *A* **type definition** *is a set D of rules of the form*

$$T \to l[r], \quad T \to l\{s\}, \quad \text{or} \quad T' \to c_1 \mid \ldots \mid c_n,$$

where T is a type variable, T' an enumeration type name, l a label, r a regular type expression, s a multiplicity list, and $c_1, \ldots, c_n$ are basic constants. A rule $U \to G \in D$ will be called a **rule for** *U in D. We require that for any type name $U \in \mathcal{V} \cup \mathcal{S}$ occurring in D there is exactly one rule for U in D.*

Type definitions are a kind of grammars, they define sets by means of derivations, where a type variable T is replaced by the right hand side of the rule for T and a regular expression r is replaced by a string from $L(r)$; if T is a type constant or an enumeration type name then it is replaced by a basic constant from respectively $[\![T]\!]$, or from the rule for T. For a formal definition see [2,3].

Example 7. Consider a type definition D which will be used in the next example:

$AddrBs \rightarrow addr\text{-}books[AddrB^*]$
$AddrB \rightarrow addr\text{-}book[Owner\ Entry^*]$
$Entry \rightarrow entry[Name\ Rel\ PhNo^*\ Address^?\ ]$
$RelCat \rightarrow \mathsf{friend} \mid \mathsf{family} \mid \mathsf{colleague} \mid \mathsf{acquaintance}$
$Address \rightarrow address[Street\ ZipC^?\ City\ Country^?]$
$ZipC \rightarrow zip\text{-}code[Text]$
$Country \rightarrow country[Text]$
$Owner \rightarrow owner[Text]$
$Name \rightarrow name[Text]$
$Rel \rightarrow relation[RelCat]$
$PhNo \rightarrow phoneNo[Text]$
$Street \rightarrow street[Text]$
$City \rightarrow city[Text]$

D contains a rule for each of type variables: *AddrBs, AddrB, Owner*, etc., and a rule for enumeration type name *RelCat*. *Text* is a type constant representing the set of strings of characters (similarly as `#PCDATA` in DTD).

The set $[\![AddrBs]\!]$ of data terms derivable from the type name *AddrBs* contains the data term *addr-books*[...] (when completed reasonably) from Example 2. Its subterm *address*[*street*[Hayes 51], *zip-code*[21213], *city*[Los Angeles], *country*[USA]] can be derived from the type name *Address*. □

In the following sections we will describe derivations of new types given a type definition D. We assume that whenever a new type is derived the type definition D is extended by rules defining the new type. The notation $[\![U]\!]$ will be used also for a set of type names $U = \{T_1, \ldots, T_n\}$. We define it as $[\![U]\!] = [\![T_1]\!] \cup \ldots \cup [\![T_n]\!]$.

3 Type Inference

The section presents a type inference method for multiple rule Xcerpt programs. We refer to our previous work [2,3,14] where we described type inference for a single Xcerpt rule. Let p be a query rule which may query intermediate results of the program (i.e. results of the rules of the program) as well as external resources (e.g. XML data). We assume that specifications of types of the external resources (such as DTD's which can be translated into type definitions) are available and they are given by a mapping *type*. The mapping associates each resource r occurring in p with a type $T = type(r)$ defined by a type definition D. The type contains the data term $d(r)$ referred to by r (i.e. $d(r) \in [\![T]\!]$). We also assume that U is a type of intermediate results of the program (i.e. the set $[\![U]\!]$ contains the intermediate results to be queried by p). The algorithm presented in [2,3,14] derives, for a given p and U, a set of type names which we will denote by $resType(p, U)$. The set approximates the set of results of p applied to $[\![U]\!]$. Formally (for a proof see [3]):

Theorem 1 (Soundness of type inference for a rule). *Let D be a type definition and p be a query rule, where for each targeted query term* `in`$[r, q]$ *in p there is a type name $T = type(r)$ defined in D such that $d(r) \in [\![T]\!]$. Let U be a set of type names and Z a set of data terms. If $Z \subseteq [\![U]\!]$ then $res(p, Z) \subseteq [\![resType(p, U)]\!]$.*

The algorithm is described [2,3,14] at an abstract level by means of typing rules for various Xcerpt constructs. Due to lack of space we do not discuss its details.

Here we propose a way of deriving a result type of a query rule in a context of a program (where the set of intermediate result types U is unknown). In other words

we propose a way of typing programs instead of single rules. The following algorithm is based on the fixed point semantics of Xcerpt expressed by the Definitions 3 and 7. It iteratively computes types of intermediate results of query rules given a type of intermediate results obtained in the previous step (in the first iteration the type of intermediate results is empty). This process is repeated until a fixed point is reached i.e. the type of intermediate results does not change in the consecutive steps.

Definition 11 (Immediate consequence operator for rule result types). *Let P be a set of Xcerpt query rules. T_P is a function defined on sets of type names such that $T_P(U) = \bigcup_{p \in P} resType(p, U)$.*

In what follows we restrict our considerations to those programs P for which T_P is monotonic (see the Appendix). Then $[\![T_P^i(\emptyset)]\!] \subseteq [\![T_P^j(\emptyset)]\!]$ for $i \leq j$. The following theorem shows how to infer result types of query rules of a program.

Theorem 2. *Let $\mathcal{P} = (P', G)$ be an Xcerpt program and $P = P' \backslash G$. If d is a result of a rule p in P' then there exists $i > 0$ such that*

$$d \in [\![resType(p, T_P^i(\emptyset))]\!] \subseteq [\![T_{P'}(T_P^i(\emptyset))]\!].$$

If $[\![T_P^{j+1}(\emptyset)]\!] = [\![T_P^j(\emptyset)]\!]$ for some $j > 0$ then the above holds for $i = j$.

For a proof see the Appendix.

Example 8. Consider the Xcerpt program $\mathcal{P} = (\{p_1, p_2, g\}, \{g\})$ from Example 2 and the type definition from Example 7. Assume that the XML document queried by the rule p_1 is of the type *AddrBs* from the type definition i.e. *type("file:addrBooks.xml") = AddrBs*. We want to derive the result types of the rules of the program.

We employ T_P where $P = \{p_1, p_2\}$. $T_P(\emptyset) = resType(p_1, \emptyset) \cup resType(p_2, \emptyset)$. The type inference algorithm returns $resType(p_1, \emptyset) = \{Fo\}$, where the type *Fo* is defined as *Fo* $\rightarrow$ *fo*[*Text Text*], and $resType(p_2, \emptyset) = \emptyset$ (as the rule p_2 does not query any external data). Thus $T_P(\emptyset) = \{Fo\}$.

$T_P^2(\emptyset) = T_P(T_P(\emptyset)) = T_P(\{Fo\}) = resType(p_1, \{Fo\}) \cup resType(p_2, \{Fo\}) = \{Fo\} \cup \{Foaf\} = \{Fo, Foaf\}$, where the rule for *Foaf* is *Foaf* $\rightarrow$ *foaf*[*Text Text*]. Similarly $T_P^3(\emptyset) = \{Fo, Foaf\}$. Hence, $U^\infty = \{Fo, Foaf\}$ is a fixed point of T_P.

Now, we can obtain the final result types of the rules of $\mathcal{P}$: $resType(p_1, U^\infty) = \{Fo\}$, $resType(p_2, U^\infty) = \{Foaf\}$ and $resType(g, U^\infty) = \{Cof\}$, where the type *Cof* is defined as *Cof* $\rightarrow$ *clique-of-friends*{*Foaf*$^+$}. □

3.1 Termination

There are two difficulties related to computing a fixed point of T_P. First, we have to check whether the current iteration of T_P produces a fixed point. Then, the iterations T_P may not terminate (all the sets $[\![T_P^i(\emptyset)]\!]$ may be distinct).

As $[\![T_P^i(\emptyset)]\!] \subseteq [\![T_P^{i+1}(\emptyset)]\!]$, for checking $[\![T_P^i(\emptyset)]\!] = [\![T_P^{i+1}(\emptyset)]\!]$ it is sufficient to check if $[\![T_P^i(\emptyset)]\!] \supseteq [\![T_P^{i+1}(\emptyset)]\!]$. This cannot be done efficiently (the task is EXPTIME-hard), and an algorithm is complicated. Efficient and simple algorithms

exist for type definitions satisfying certain restrictions; see [5] for a discussion. However the restrictions are often not satisfied by the type definitions created by evaluating T_P.

In this section we show that for non w-recursive programs the computing of a fixed point terminates and determining when the fixed point is obtained is easy.

Proposition 1. *Let P be a set of rules and $n > 0$. If $[\![T_P^{n-1}(\emptyset)]\!] \neq [\![T_P^n(\emptyset)]\!]$ then there exist $p_1, \ldots, p_n \in P$ such that $p_n \succ_w \cdots \succ_w p_1$.*

Proof. See the Appendix.

From the Proposition it follows that if P is not w-recursive then the fixed point of T_P is reached in at most $|P|$ steps: $[\![T_P^i(\emptyset)]\!] = [\![T_P^{i+1}(\emptyset)]\!]$ for any $i \geq |P|$. Thus $T_P^{|P|}(\emptyset)$ is a fixed point of T_P. Moreover, if the longest chain $p_k \succ_w \cdots \succ_w p_1$ of rules in P contains k rules then the fixed point is reached in k steps.

3.2 Dealing with Recursion

Weak static recursion in a program $\mathcal{P}$ can prevent reaching a fixed point of T_P, thus it may make impossible finding result types of query rules of $\mathcal{P}$. Now we show how this problem can be overcome.

One way of assuring that a fixed point will be reached in a w-recursive program $\mathcal{P}$ is breaking the cycles in the graph of relation $\succ_w$ of $\mathcal{P}$. This may be achieved by selecting a rule p belonging to the cycle, finding an approximation (a superset) $[\![W_p]\!]$ of the set of results of p in some independent way (described later on), and removing p from the program. Instead, W_p is added to the type computed at each iteration. Thus instead of computing $T_P^i(\emptyset)$, we compute $\widehat{T}_P^i(\emptyset)$, where $\widehat{T}_P(U) = T_{P\setminus\{p\}}(U) \cup W_p$.

This approach can be applied to break all cycles detected in the graph. Let $\mathcal{P} = (P', G)$ and $P = P' \setminus G$. Assume that $P_0 = \{p_1, \ldots, p_m\}$ are rules removed from P to break all cycles. Assume also that a set of type names $W = W_{p_1} \cup \ldots \cup W_{p_m}$ is an approximation of their results, i.e. that if d is a result of p_i in $\mathcal{P}$ then $d \in [\![W_{p_i}]\!]$. Instead of T_P, we employ $\widehat{T}_P$, defined by $\widehat{T}_P(U) = T_{P\setminus P_0}(U) \cup W$. If all cycles are broken in the the program, i.e. there is no w-recursion in $P \setminus P_0$, then the fixed point U^∞ of $\widehat{T}_P$ will be found after at most $|P| - m$ iterations: $U^\infty = \bigcup_{i=1}^{\infty} \widehat{T}_P^i(\emptyset) = \bigcup_{i=1}^{|P|-m} \widehat{T}_P^i(\emptyset)$. (This follows from Proposition 1, which also holds for $\widehat{T}_P$ with basically the same proof.)

To make the approach work, we must know how to find a correct approximation W_p of the set of results of a rule p in $\mathcal{P}$. A rough approximation can be obtained based on the head h of p alone. If no variable occurs twice in h then the approximation is the type of all instances of h. Otherwise we take the set of all instances of h', where h' is h with each variable occurrence replaced by a distinct variable. For instance, such approximation for a rule $c[\,a[\,X\,],\,X\,] \leftarrow Q$ is the type T defined by a type definition $D = \{\,T \rightarrow c[\,A\ \mathit{Top}\,],\ A \rightarrow a[\,\mathit{Top}\,]\,\}$.

A more precise approximation may be provided by the user. In this case it should be checked that the approximation is indeed correct. This can be achieved by checking whether $[\![\mathit{resType}(p, U^\infty)]\!] \subseteq [\![W_p]\!]$ for each employed approximation

W_p of the results of a rule p. (The problems with inefficiency of inclusion checking, discussed at the beginning of Section 3.1, can be avoided by imposing certain restrictions [5] on the type definition provided by the user.)

We presented a method of approximating the result sets of w-recursive programs. The following theorem establishes its correctness.

Theorem 3. *Let $\mathcal{P} = (P', G)$ be an Xcerpt program, $P = P' \setminus G$, and $P_0 \subseteq P$ such that $P \setminus P_0$ is not w-recursive. Assume that T_P is monotonic. Let W be a set of type names, and let $\widehat{T}_P(U) = T_{P\setminus P_0}(U) \cup W$ for any set U of type names. Let $U^\infty = \widehat{T}_P^k(\emptyset)$ be a fixed point of $\widehat{T}_P$ (i.e. $[\![\widehat{T}_P(U^\infty)]\!] = [\![U^\infty]\!]$). If $[\![resType(p, U^\infty)]\!] \subseteq [\![W]\!]$ for each $p \in P_0$ then*

$$\begin{array}{ll} d \in [\![resType(p, U^\infty)]\!] \subseteq [\![U^\infty]\!] & \text{for any result } d \text{ of a rule } p \in P, \\ d \in [\![resType(p, U^\infty)]\!] \subseteq [\![T_{P'}(U^\infty)]\!] & \text{for any result } d \text{ of a rule } p \in P', \\ [\![T_P(U^\infty)]\!] \subseteq [\![U^\infty]\!] \quad \text{and} \quad [\![T_P^j(\emptyset)]\!] \subseteq [\![U^\infty]\!] & \text{for any } j > 0. \end{array}$$

Moreover, U^∞ in the last three lines may be replaced by $T_P^j(U^\infty)$, for any $j > 0$.

Thus $\cdots \subseteq [\![T_P^i(\emptyset)]\!] \subseteq [\![T_P^{i+1}(\emptyset)]\!] \subseteq \cdots \subseteq [\![T_P^{j+1}(U^\infty)]\!] \subseteq [\![T_P^j(U^\infty)]\!] \subseteq \cdots$.

For a proof see the Appendix. The theorem shows a way of more precise approximating the set of program results. After obtaining a fixed point U^∞ of $\widehat{T}_P$, we iteratively apply T_P a few times. An intuitive explanation is that in U^∞ the approximation of the results of a rule $p \in P_0$ is the same as that given by W. Analyzing the results of p applied to the data from $[\![U^\infty]\!]$ may exclude some data terms from $[\![W]\!]$. Thus it may improve the approximation of results of p, which in turn may improve the approximation of results of the rules which w-depend on p.

Example 9. Consider an Xcerpt program $\mathcal{P} = (\{p_1, p_2, g\}, \{g\})$, where

$$g = r[\,\texttt{all}\ X\,] \leftarrow c[\,X\,], \qquad \begin{array}{l} p_1 = c[\,b[\,X\,]\,] \leftarrow \texttt{and}[\,c[\,X\,],\ \texttt{in}[\,res,\ \texttt{desc}\ X\,]\,], \\ p_2 = c[\,X\,] \leftarrow \texttt{in}[\,res,\ b[[\,a[\,X\,]\,]]\,] \end{array}$$

and a type definition $D = \{\, A \to a[\,Text\,],\ T \to b[\,(A \mid T \mid Text)^*\,]$. Assume that the type of the resource *res* is T.

We want to approximate the set of results of $\mathcal{P}$. We show that a fixed point cannot be obtained by computing $T_P^i(\emptyset)$ where $P = \{p_1, p_2\}$. Then we apply Theorem 3. As $[\![W]\!]$ we first use the set of all instances of the head of the w-recursive rule p_1. Then we show how a better approximation can be obtained by employing a more precise initial specification W.

We first find that, independently from U, $resType(p_2, U) = \{C_1\}$, where the rule for C_1 is $C_1 \to c[\,Text\,]$. This is because the query $\texttt{in}[\,res,\ b[[\,a[\,X\,]\,]]\,]$ binds X to a value from $[\![Text]\!]$. Thus $T_P(U) = resType(p_1, U) \cup resType(p_2, U) = resType(p_1, U) \cup \{C_1\}$.

Hence $T_P(\emptyset) = resType(p_1, \emptyset) \cup \{C_1\} = \emptyset \cup \{C_1\} = \{C_1\}$. Now $resType(p_1, \{C_1\}) = \{C_2\}$, where type C_2 is defined by the rules $C_2 \to c[\,B_1\,]$, $B_1 \to b[\,Text\,]$ (as the query $c[X]$ binds X to a value from $[\![Text]\!]$ and $\texttt{in}[\,res,\ \texttt{desc}\ X\,]$ binds X to a value from $[\![\{\,Text, A, T\}]\!]$). Hence $T_P(T_P(\emptyset)) = T_P(\{C_1\}) = \{C_2\} \cup \{C_1\}$.

Generally we obtain $T_P^i(\emptyset) = \{C_1, \ldots, C_i\}$ $(i > 1)$, with rules $C_j \to c[B_{j-1}]$, $B_j \to b[B_{j-1}]$ (for $j > 1$). All the sets $[\![T_P^i(\emptyset)]\!]$ are distinct and the fixed point will never be reached.

We can however approximate the results of $\mathcal{P}$ by applying Theorem 3. The program with p_1 removed is not w-recursive. The set of results of p_1 can be approximated by the set $[\![C_a]\!]$ of all the instances of the head of p_1; the type C_a is defined by rules $C_a \rightarrow c[\, B_a \,]$, $B_a \rightarrow b[\, \mathit{Top} \,]$. We look for a fixed point of $\widehat{T}_P$, where $\widehat{T}_P(U) = T_{\{p_2\}}(U) \cup \{C_a\} = \mathit{resType}(p_2, U) \cup \{C_a\}$. By the discussion following Proposition 1, the fixed point is $U^\infty = \widehat{T}_P^1(\emptyset) = \{C_1, C_a\}$. As an approximation of the set of results of $\mathcal{P}$ we obtain $\mathit{resType}(g, \{C_1, C_a\}) = \{R\}$ where type R is defined as $R \rightarrow r[\, (\mathit{Text} \,|\, B_a)^+ \,]$.

To obtain a better approximation we can apply T_P to the set U^∞. $U_1^\infty = T_P(U^\infty) = \mathit{resType}(p_1, U^\infty) \cup \{C_1\} = \{C_1'\} \cup \{C_1\}$, where type C_1' is defined by the rules $C_1' \rightarrow c[\, B_1' \,]$, $B_1' \rightarrow b[\, \mathit{Text} \,|\, B' \,]$, $B' \rightarrow b[\, \mathit{Text} \,|\, A \,|\, T \,]$. This allows to obtain a more precise type of the goal rule which is $\mathit{resType}(g, U_1^\infty) = \{R_1\}$, where type R_1 is defined as $R_1 \rightarrow r[\, (\mathit{Text} \,|\, B_1')^+ \,]$.

By applying T_P to U^∞ multiple times we can further improve the precision of the approximation. $U_i^\infty = T_P^i(U^\infty) = \{C_1, C_i'\}$, where type C_i' is defined by the rules $C_i' \rightarrow c[\, B_i' \,]$, $B_i' \rightarrow b[\, \mathit{Text} \,|\, B_{i-1}' \,]$ for $i > 1$. This produces a type R_i of results of $\mathcal{P}$ defined as $R_i \rightarrow r[\, (\mathit{Text} \,|\, B_i')^+ \,]$.

The above approximations are obtained based on the automatic rough approximation C_a of the set of results of the rule p_1. However, the user can provide a more precise result type of the rule p_1 than C_a e.g. a type C_u defined by the rules $C_u \rightarrow c[\, B_u \,]$, $B_u \rightarrow b[\, \mathit{Text} \,|\, B_u \,|\, C_u \,]$. Based on this a fixed point of the operator $\widehat{T}_P'(U) = T_{\{p_2\}}(U) \cup \{C_u\}$ can be computed, which is $U_u^\infty = \{C_1, C_u\}$. To make sure that the approximation C_u provided by the user is correct we test whether $[\![\mathit{resType}(p_1, U_u^\infty)]\!] \subseteq [\![C_u]\!]$. $\mathit{resType}(p_1, U_u^\infty) = \{C\}$, where type C is defined by the rules $C \rightarrow c[\, B \,]$, $B \rightarrow b[\, \mathit{Text} \,|\, B \,]$. As $[\![C]\!] \subseteq [\![C_u]\!]$ the test is successful.

To improve the approximation U_u^∞ of the set of results of p_1, p_2 we can apply the operator T_P to U_u^∞. $U_{u_1}^\infty = T_P(U_u^\infty) = \{C_1, C\}$. Further applications of T_P to U_u^∞ provide the same results i.e. $T_P^i(U_u^\infty) = U_{u_1}^\infty$, for $i > 0$. Based on $U_{u_1}^\infty$ we obtain a precise type R_u of the goal rule which is defined as $R_u \rightarrow r[\, (\mathit{Text} \,|\, B)^+ \,]$. □

4 Type-based Rule Dependency

In this section we show how to employ the presented type inference methods to approximate dynamic dependency of rules in a program. The goal is to obtain better approximations than those given by static dependency. Let $\mathcal{P} = (P', G)$ be an Xcerpt program and $P = P' \setminus G$. By a **typing** of P we mean an approximation of the set of rule results of P by a set of type names. Formally, U is a typing for P if $[\![U]\!]$ contains each result of each rule from P. We presented two ways of obtaining such a typing: finding a fixed point of T_P or applying Theorem 3. Let U^P be a typing for P and $U_i^P = \mathit{resType}(p_i, U^P)$ for each $p_i \in P$.

The function *resType* is not useful for finding dynamic rule dependencies. This is because the fact $\mathit{resType}(p_i, U_j^P) \neq \emptyset$ does not imply $p_i \succ p_j$. The rule p_i may query external data so *resType* may return a non empty type independently of U_j^P (including cases when no data in U_j^P is matched by any top query term in p_i). The

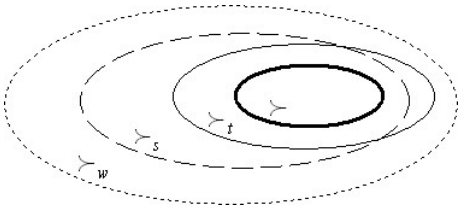

Fig. 1. Relation between dependencies for rules in a program.

inverse implication, i.e. $p_i \succ p_j$ implies $resType(p_i, U_j^P) \neq \emptyset$, is neither true[2]. Thus we need some better way of approximating dynamic rule dependencies.

A part of the process of obtaining $resType(p, U)$, described in [2,14,8], is typing of query terms. Given a query term q and a type $T \in U$ of data to which q is applied, mappings of variables occurring in q to types are constructed. The mappings specify sets of substitutions (of data terms for variables). If q matches some data term d from $[\![T]\!]$ then a variable-type mapping Γ is produced; Γ describes a non empty set of substitutions. (The set contains the result, or some of the results, of matching q with d.)

The algorithm for *resType* can be easily augmented to compute a Boolean function *matchesType* whose arguments are a query rule p and a set of type names U; $matchesType(p, U)$ is true iff a Γ describing a non empty set of substitutions is obtained for a top query term q from p and a type $T \in U$. Thus the following holds.

Proposition 2. *Let $\mathcal{P} = (P', G)$ be an Xcerpt program and $p, p' \in P'$. If $p \succ p'$ then $matchesType(p, U)$ for any set U of type names such that the results of p' are contained in $[\![U]\!]$.*

Now we can define a new kind of rule dependency.

Definition 12 (Type-based dependency). *Let $\mathcal{P} = (P', G)$ be an Xcerpt program, $P = P' \setminus G$, $p_i \in P$, U^P be a typing of P, and $U_i^P = resType(p_i, U^P)$. A rule $p \in P'$* type-based directly depends *on p_i (denoted by $p \succ_t p_i$) if $matchesType(p, U_i^P)$.*

Proposition 3. *If $p \succ p'$ then $p \succ_t p'$. If $p \succ_t p'$ then $p \succ_w p'$.*

On the other hand, neither $p \succ_s p'$ implies $p \succ_t p'$ nor $p \succ_t p'$ implies $p \succ_s p'$. Both the type-based rule dependency and the static rule dependency approximate dynamic rule dependency in a program. Combining them provides better approximation than any of them separately.

Example 10. Consider a type definition $D = \{ T \to l[\, T_1\, T_2\, T_1^* \,],\ T_1 \to \mathsf{e} \,|\, \mathsf{f},\ T_2 \to \mathsf{e} \}$ and an Xcerpt program $\mathcal{P} = (\{g, p_1, p_2, p_3\}, \{g\})$ which queries an external resource *res* of type T. Next to the rules of the program there are specifications of the corresponding result types inferred for them:

[2] Consider query rules: $p_1 = c[\, X \,] \leftarrow \texttt{and}[\, a[\, X \,],\ b[\, X \,] \,],\quad p_2 = a[\, X \,] \leftarrow d[\, X \,]$. $resType(p_1, U_2^P) = \emptyset$ although $p_1 \succ p_2$ (unless p_2 produces no results).

$$
\begin{array}{ll}
g = a[\,X\,] \leftarrow b[\,\mathtt{e},\,\mathtt{f},\,X\,] & A \to a[\,T_2\,] \\
p_1 = b[\,X,\,Y,\,Z\,] \leftarrow \mathtt{in}[\,res,\,l[\,X,\,Y,\,Z\,]\,] & A_1 \to b[\,T_1\,T_2\,T_1\,] \\
p_2 = b[\,X,\,X,\,Y\,] \leftarrow \mathtt{in}[\,res,\,l[\,X,\,Y\,]\,] & A_2 \to b[\,T_1\,T_1\,T_2\,] \\
p_3 = b[\,X,\,Z,\,Y\,] \leftarrow \mathtt{in}[\,res,\,l[\,X,\,Y,\,Z\,]\,] & A_3 \to b[\,T_1\,T_1\,T_2\,]
\end{array}
$$

The rule g can dynamically depend only on the rule p_3 and only this dependency should be considered by optimal evaluation of the program. The rule g type-based depends only on the rules p_2, p_3 and statically depends on the rules p_1, p_3. (Hence $g \succ_w p_1$, $g \succ_w p_2$ and $g \succ_w p_3$.) Thus combination of type-based dependency and static dependency better approximates dynamic dependency than both dependencies separately. □

5 Summary

The paper presents a method of type inference for Xcerpt. The inferred types approximate the sets of results of programs (or of particular rules in the programs). The previous work [2,14] dealt mainly with single rules of Xcerpt, here we add a treatment of possibly recursive multiple rule programs. The method deals with a substantial fragment of Xcerpt. We believe it can be easily extended, by providing a way of deriving at least very rough type approximations for Xcerpt constructs not dealt with. This work is in progress.

Our approach can be seen as abstract interpretation [7]. The abstract domain is the set of possible types, the height of the domain is infinite. A special feature is that in computing a fixed point the number of iterations is known in advance. This makes it possible to avoid an expensive test for a fixed point (i.e. type inclusion).

Our method allows to perform type checking for programs which is done by testing whether the inferred type is included in the specified one. The test is not expensive if the specified type satisfies the conditions of [5].

We also discuss dependencies between rules in Xcerpt programs. From a point of view of efficient evaluation of programs, dynamic dependency between rules is essential [9]. Static dependency used previously [12] is only its rough approximation. We show how a more precise approximation can be obtained by employing type analysis.

References

1. V. Benzaken, G. Castagna, and A. Frisch. CDuce: An XML-centric general-purpose language. In *ICFP 2003*. ACM Press.
2. S. Berger, E. Coquery, W. Drabent, and A. Wilk. Descriptive typing rules for Xcerpt. In *PPSW 2005*, number 3703 in LNCS. Springer Verlag.
3. S. Berger, E. Coquery, W. Drabent, and A. Wilk. Descriptive typing rules for Xcerpt and their soundness. Technical Report REWERSE-TR-2005-01, REWERSE, 2005. http://rewerse.net/publications/#REWERSE-TR-2005-01. Errata: http://www.ida.liu.se/~wlodr/errata.LNCS3703.pdf.
4. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, April 2001.

5. F. Bry, W. Drabent, and J. Maluszynski. On subtyping of tree-structured data: A polynomial approach. In *PPSWR 2004*, number 3208 in LNCS. Springer Verlag.
6. J. Coelho and M. Florido. "XCentric: A Logic Programming Language for XML Processing". In *PLAN-X 07*.
7. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM POPL 1977*.
8. W. Drabent. Towards More Precise Typing Rules for Xcerpt. In *PPSWR 2006*, number 4187 in LNCS. Springer Verlag.
9. Tim Furche, 2006. Personal communication.
10. H. Hosoya and B. C. Pierce. "XDuce: A Typed XML Processing Language". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, 2000.
11. W. Martens, F. Neven, and T. Schwentick. Which XML Schemas Admit 1-Pass Preorder Typing? In *ICDT*, number 3363 in LNCS, 2005.
12. S. Schaffert. *Xcerpt: A Rule-Based Query and Transformation Language for the Web*. PhD thesis, University of Munich, Germany, 2004. http://www.wastl.net/download/dissertation/dissertation_schaffert.pdf.
13. S. Schaffert and F. Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Extreme Markup Languages 2004*.
14. A. Wilk. Descriptive Types for XML Query Language Xcerpt, 2006. Licentiate Thesis. Linköping University. http://www.diva-portal.org/liu/abstract.xsql?dbid=7542.
15. A. Wilk and W. Drabent. On types for XML query language Xcerpt. In *Principles and Practice of Semantic Web Reasoning 2003*, LNCS. Springer Verlag.
16. A. Wilk and W. Drabent. A prototype of a descriptive type system for Xcerpt. In *Principles and Practice of Semantic Web Reasoning 2006*, LNCS. Springer-Verlag.

APPENDIX
for
Type Inference and Rule Dependencies in Xcerpt
by W. Drabent and A. Wilk
May 2007

The appendix provides proofs of Theorems 2, 3 and Proposition 1. The proofs are based on the following property proved in [1]:

Theorem 1. *Let D be a type definition and p be a query rule, where for each targeted query term $\mathtt{in}[r, q]$ in Q there is a type name $T = type(r)$ defined in D such that $d(r) \in [\![T]\!]_D$. Let U be a set of type names and Z a set of data terms. If $Z \subseteq [\![U]\!]$ then $res(p, Z) \subseteq [\![resType(p, U)]\!]$.*

We will also use the properties of *resType* expressed by the following two lemmas. In this report we are interested only in those Xcerpt rules and Type Definitions for which Lemma 1 holds.

Lemma 1. *If $[\![U]\!] \subseteq [\![U']\!]$ then $[\![resType(p, U)]\!] \subseteq [\![resType(p, U')]\!]$.*

Lemma 2. *Let p, p' be a pair of rules, U, U'' be sets of type names and $U' = resType(p', U'')$. If $p \not\succ_w p'$ then $[\![resType(p, U \cup U')]\!] = [\![resType(p, U)]\!]$.*

Corollary 1 (Monotonicity of T_P). *Let P be a set of rules and U, U' be sets of type names such that $[\![U]\!] \subseteq [\![U']\!]$. Then $[\![T_P(U)]\!] \subseteq [\![T_P(U')]\!]$ and $[\![T_P^i(\emptyset)]\!] \subseteq [\![T_P^{i+1}(\emptyset)]\!]$ for $i = 0, 1, \ldots$.*

Proof. Follows from Lemma 1. □

Lemma 3. *Let P, P' be sets of rules such that $P' \subseteq P$. Let $U = T_P^i(\emptyset)$ for some $i > 0$ and Z be a set of data terms. If $Z \subseteq [\![U]\!]$ then $R_{P'}^j(Z) \subseteq [\![T_P^j(U)]\!]$ for each $j = 0, 1, \ldots$.*

Proof. By Theorem 1, for each rule $p \in P'$, $res(p, Z) \subseteq [\![resType(p, U)]\!] \subseteq [\![\bigcup_{p \in P} resType(p, U)]\!] = [\![T_P(U)]\!]$. This implies that $\bigcup_{p \in P'} res(p, Z) \subseteq [\![T_P(U)]\!]$. Also $Z \subseteq [\![U]\!] \subseteq [\![T_P(U)]\!]$, by Corollary 1. Thus $R_{P'}(Z) = Z \cup \bigcup_{p \in P'} res(p, Z) \subseteq [\![T_P(U)]\!]$. Hence $R_{P'}^j(Z) \subseteq [\![T_P^j(U)]\!]$, by induction on j. □

Lemma 4. *Let $\mathcal{P} = (P', G)$ be an Xcerpt program, $P_1, \ldots, P_n, G$ be a stratification of $\mathcal{P}$ and $P = P' \backslash G$. Let $Z_0 = \emptyset$ and, for $j = 1, \ldots, n$, let $Z_j = R_{P_j}^{l_j}(Z_{j-1})$ for such $l_j > 0$ that $R_{P_j}^{l_j}(Z_{j-1}) = R_{P_j}^{l_j+1}(Z_{j-1})$. Then, for $j = 1, \ldots, n$,*

$$Z_j \subseteq [\![T_P^{k_j}(\emptyset)]\!] \qquad \text{where} \qquad k_j = \sum_{m=1}^{j} l_m.$$

Proof. By induction on j. For $j = 0$ the conclusion trivially holds. Assume $Z_{j-1} \subseteq [\![T_P^{k_{j-1}}(\emptyset)]\!]$. By Lemma 3, $R_{P_j}^{l_j}(Z_{j-1}) \subseteq [\![T_P^{l_j}(T_P^{k_{j-1}}(\emptyset))]\!] = [\![T_P^{l_j+k_{j-1}}(\emptyset))]\!] = [\![T_P^{k_j}(\emptyset))]\!]$. So $Z_j \subseteq [\![T_P^{k_j}(\emptyset)]\!]$. □

Theorem 2. *Let $\mathcal{P} = (P', G)$ be an Xcerpt program and $P = P' \backslash G$. If d is a result of a rule p in P' then there exists $i > 0$ such that*

$$d \in [\![resType(p, T_P^i(\emptyset))]\!] \subseteq [\![T_{P'}(T_P^i(\emptyset))]\!].$$

If $[\![T_P^{j+1}(\emptyset)]\!] = [\![T_P^j(\emptyset)]\!]$ for some $j > 0$ then the above holds for $i = j$.

Proof. Let $P_1, \ldots, P_n, G$ be a stratification of $\mathcal{P}$. Let $Z_0 = \emptyset$ and for $j = 1, \ldots, n$, $Z_j = R_{P_j}^{l_j}(Z_{j-1})$ for such $l_j > 0$ that $R_{P_j}^{l_j}(Z_{j-1}) = R_{P_j}^{l_j+1}(Z_{j-1})$.

By Lemma 4, $Z_n \subseteq [\![T_P^{k_n}(\emptyset)]\!]$, where $k_n = \sum_{m=1}^{n} l_m$. Let $i = k_n$. By Definition 7 and Theorem 1, $d \in res(p, Z_n) \subseteq [\![resType(p, T_P^{k_n}(\emptyset))]\!]$.

By Definition 11, $[\![resType(p, U)]\!] \subseteq [\![T_{P'}(U)]\!]$ for any U. Thus the first conclusion of the Theorem holds.

If $[\![T_P^{j+1}(\emptyset)]\!] = [\![T_P^j(\emptyset)]\!]$ then $[\![T_P^i(\emptyset)]\!] \subseteq [\![T_P^j(\emptyset)]\!]$ for any $i \geq 0$. Thus $d \in [\![resType(p, T_P^i(\emptyset))]\!] \subseteq [\![resType(p, T_P^j(\emptyset))]\!]$ and the conclusion holds with i replaced by j. □

Proposition 1. *Let P be a set of rules and $n > 0$. If $[\![T_P^{n-1}(\emptyset)]\!] \neq [\![T_P^n(\emptyset)]\!]$ then there exist $p_1, \ldots, p_n \in P$ such that $p_n \succ_w \cdots \succ_w p_1$.*

Proof. For $n = 1$ the proposition holds trivially, as $\emptyset \neq [\![T_P(\emptyset)]\!]$ implies that P is nonempty. So assume that $n > 1$. Let $U^i = T_P^i(\emptyset)$. We have $T_P(U) = \bigcup_{p \in P} resType(p, U)$ by the definition of T_P, and

$$resType(p, U^i) = resType\Big(p, \bigcup_{\substack{p' \in P \\ p \succ_w p'}} resType(p', U^{i-1})\Big) \tag{1}$$

for $i > 0$, by Lemma 2. From $[\![U^{n-1}]\!] \neq [\![U^n]\!]$ it follows that $T_P(U^{n-2}) \neq T_P(U^{n-1})$ and then $resType(p_n, U^{n-2}) \neq resType(p_n, U^{n-1})$ for some $p_n \in P$.

By (1), if $resType(p, U^{i-1}) \neq resType(p, U^i)$ then there exists a rule $p' \in P$ such that $p \succ_w p'$ and if $i > 1$ then $resType(p', U^{i-2}) \neq resType(p', U^{i-1})$. From this by induction we obtain that if $resType(p_n, U^{n-2}) \neq resType(p_n, U^{n-1})$ then there exist $p_1, \ldots, p_n \in P$ such that $p_n \succ_w \cdots \succ_w p_1$. □

Theorem 3. *Let $\mathcal{P} = (P', G)$ be an Xcerpt program, $P = P' \setminus G$, and $P_0 \subseteq P$ such that $P \setminus P_0$ is not w-recursive. Let W be a set of type names, and let $\widehat{T}_P(U) = T_{P \setminus P_0}(U) \cup W$ for any set U of type names. Let $U^\infty = \widehat{T}_P^k(\emptyset)$ be a fixed point of $\widehat{T}_P$ (i.e. $[\![\widehat{T}_P(U^\infty)]\!] = [\![U^\infty]\!]$). If $[\![resType(p, U^\infty)]\!] \subseteq [\![W]\!]$ for each $p \in P_0$ then*

$$\begin{array}{ll} d \in [\![resType(p, U^\infty)]\!] \subseteq [\![U^\infty]\!] & \text{for any result } d \text{ of a rule } p \in P, \\ d \in [\![resType(p, U^\infty)]\!] \subseteq [\![T_{P'}(U^\infty)]\!] & \text{for any result } d \text{ of a rule } p \in P', \\ [\![T_P(U^\infty)]\!] \subseteq [\![U^\infty]\!] \quad \text{and} \quad [\![T_P^j(\emptyset)]\!] \subseteq [\![U^\infty]\!] & \text{for any } j > 0. \end{array}$$

Moreover, U^∞ in the last three lines may be replaced by $T_P^j(U^\infty)$, for any $j > 0$.

Proof. Notice that $[\![T_{P\setminus P_0}(U^\infty)]\!] \subseteq [\![U^\infty]\!]$, as $[\![T_{P\setminus P_0}(U^\infty)]\!] \cup [\![W]\!] \subseteq [\![U^\infty]\!]$. Notice also that $[\![resType(p, U^\infty)]\!] \subseteq [\![U^\infty]\!]$ for $p \in P_0$. Hence $[\![T_P(U^\infty)]\!] \subseteq [\![U^\infty]\!]$, as $T_P(U) = T_{P\setminus P_0}(U) \cup \bigcup_{p\in P_0} [\![resType(p, U)]\!]$. From monotonicity of T_P we obtain by induction that $[\![U^\infty]\!] \supseteq [\![T_P(U^\infty)]\!] \supseteq [\![T_P^2(U^\infty)]\!] \supseteq \cdots$, and that $[\![T_P^i(U^\infty)]\!] \supseteq [\![T_P^i(\emptyset)]\!]$ for $i \geq 0$. Hence $\cdots \subseteq [\![T_P^i(\emptyset)]\!] \subseteq [\![T_P^{i+1}(\emptyset)]\!] \subseteq \cdots \subseteq [\![T_P^{j+1}(U^\infty)]\!] \subseteq [\![T_P^j(U^\infty)]\!] \subseteq \cdots \subseteq [\![U^\infty]\!]$ for each $i, j \geq 0$. Thus, by Theorem 2 any result of a rule p of P' is in $[\![resType(p, T_P^j(U^\infty))]\!]$, for each $j \geq 0$. The latter is a subset of $[\![T_{P'}(U^\infty)]\!]$ and a subset of $[\![T_P^j(U^\infty)]\!]$ if $p \in P$. □

References

1. S. Berger, E. Coquery, W. Drabent, and A. Wilk. Descriptive typing rules for Xcerpt and their soundness. Technical Report REWERSE-TR-2005-01, REWERSE, 2005. http://rewerse.net/publications/#REWERSE-TR-2005-01. Errata: http://www.ida.liu.se/~wlodr/errata.LNCS3703.pdf.