Towards More Precise Typing Rules for Xcerpt

Włodzimierz Drabent

IPI PAN, ul. Ordona 21, PL – 01-237 Warszawa, Poland
IDA, Linköpings universitet, SE – 581 83 Linköping, Sweden
wdr@ida.liu.se

Abstract. In previous papers we presented 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. The type system was expressed by means of rules, which could be seen as a comprehensible abstraction of a typing algorithm. That system treats some query terms in a rather simplistic way. As a result the approximations produced for them are rather imprecise. In this paper we provide an improved type system, producing more precise results.

In previous papers [1, 2, 3, 8] we presented a type system for a substantial fragment of the Web query language Xcerpt [7, 6]. 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. A prototype implementation is presented in [8, 10].

The type system was expressed by means of rules, which could be seen as a comprehensible abstraction of a typing algorithm. That system treats some query terms in a rather simplistic way. As a result the approximations produced for them are rather imprecise. In the current paper we provide an improved type system, producing more precise results. The considered fragment of Xcerpt is the same as that dealt with in [1, 2]. It can be extended similarly as done in [8]. A preliminary version of a more precise type system appeared in [4], in a form of a rather complicated algorithm. Such a form of presentation was very difficult to understand and reason about.

The next section presents Xcerpt and the formalisms we use. The type system is introduced in Sect. 2 and proven correct in Sect. 3.

1 Preliminaries

To make the paper self-contained, we introduce here the underlying notions. We introduce data terms, which are our abstraction of XML documents, and a formalism of defining types (sets of data terms). We present the fragment of Xcerpt dealt with in this paper and define its formal semantics. This section is based on our former papers [9, 1, 2].
1.1 Modelling XML Data

We model XML data using a formalism of data terms similar to that defined in [7]. Data terms can be seen as mixed trees which are labelled trees where children of a node are either linearly ordered or unordered. The content of an element is a sequence of other elements or basic constants. Basic constants are basic values such as attribute values and all “free” data appearing in an XML document (PCDATA). The set of basic constants will be denoted by $B$. Tag names and attribute names of XML correspond to labels of data terms. The set of labels is denoted by $L$.

Definition 1. A data term is an expression defined inductively as follows:

- Any basic constant is a data term,
- If $l$ is a label and $t_1, \ldots, t_n$ are $n \geq 0$ data terms, then $l[t_1, \ldots, t_n]$ and $l\{t_1, \ldots, t_n\}$ are data terms.

The linear ordering of children of the node with label $l$ is denoted by enclosing them by brackets $[ ]$, while unordered children are enclosed by braces $\{ \}$. A subterm of a data term $t$ is defined inductively: $t$ is a subterm of $t$, and any subterm of $t_i$ ($1 \leq i \leq n$) is a subterm of $l'[t_1, \ldots, t_n]$ and of $l'\{t_1, \ldots, t_n\}$. Data terms $t_1, \ldots, t_n$ will be sometimes called the arguments of $l'$, or the direct subterms of $l'[t_1, \ldots, t_n]$ (and of $l'\{t_1, \ldots, t_n\}$). The root of a data term $t$, denoted $\text{root}(t)$, is defined as follows. If $t$ is of the form $l[t_1, \ldots, t_n]$ or $l\{t_1, \ldots, t_n\}$ then $\text{root}(t) = l$; for $t$ being a basic constant we assume that $\text{root}(t) = \$$. 

1.2 Type Definitions

Here we introduce a formalism for specifying a class of decidable sets of data terms representing XML documents. First we specify a set of type names $T = C \cup S \cup V$ which consist of type constants from the alphabet $C$, enumeration type names from the alphabet $S$, and type variables from the alphabet $V$. (In our former papers, enumeration type names were called special type names).

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 $T$. For $T$ being a type constant or an enumeration type name, the elements of $[T]$ are basic constants.

Type constants correspond to base types of XML schema languages. The set of type constants is fixed and finite; for each type constant $T \in C$ the set $[T] \subseteq B$ is fixed.

We denote the empty string by $\epsilon$. A regular expression over an alphabet $\Sigma$ is $\epsilon, \phi, \text{any } a \in \Sigma$ and any $r_1r_2, r_1|r_2$ and $r_1^*$, where $r_1, r_2$ are regular expressions. A language $L(r)$ of strings over $\Sigma$ is assigned to each regular expression $r$ in a standard way: $L(\phi) = \emptyset$, $L(\epsilon) = \{ \epsilon \}$, $L(a) = \{ a \}$, $L(r_1r_2) = L(r_1)L(r_2)$, $L(r_1|r_2) = L(r_1) \cup L(r_2)$, and $L(r_1^*) = L(r_1)^*$. 

Towards More Precise Typing Rules for Xcerpt