Journal of Computer and System Sciences 81 (2015) 1278–1297

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Regular expressions for data words

Leonid Libkin[a], Tony Tan[b], Domagoj Vrgoč[a,c,*]

[a] University of Edinburgh, United Kingdom
[b] Hasselt University and Transnational University of Limburg, Belgium
[c] PUC Chile and Center for Semantic Web Research, Chile

ARTICLE INFO

Article history:
Received 2 May 2014
Received in revised form 27 October 2014
Accepted 3 March 2015
Available online 24 March 2015

Keywords:
Data words
Register automata
Regular expressions

ABSTRACT

In this paper we define and study regular expressions for data words. We first define *regular expressions with memory* (REM), which extend standard regular expressions with limited memory and show that they capture the class of data words defined by register automata. We also study the complexity of the standard decision problems for REM, which turn out to be the same as for register automata. In order to lower the complexity of main reasoning tasks, we then look at two natural subclasses of REM that can define many properties of interest in the applications of data words: *regular expressions with binding* (REWB) and *regular expressions with equality* (REWE). We study their expressive power and analyse the complexity of their standard decision problems. We also establish the following strict hierarchy of expressive power: REM is strictly stronger than REWB, and in turn REWB is strictly stronger than REWE.

1. Introduction

Data words are words that, in addition to a letter from a finite alphabet, have a *data value* from an infinite domain associated with each position. For example, $\binom{a}{1}\binom{b}{2}\binom{b}{1}$ is a data word over the alphabet $\Sigma = \{a, b\}$ and $\mathbb{N}$ as the domain of values. It can be viewed as the ordinary word *abb* in which the first and the third positions are equipped with value 1, and the second position with value 2.

They were introduced by Kaminski and Francez in [14] who also proposed a natural extension of finite automata for them, called *register automata*. They have become an active subject of research lately due to their applications in XML, in particular in static analysis of logic and automata-based XML specifications, as well as in query evaluation tasks. For example, when reasoning about paths in XML trees, we may want to reason, not only about the labels of the trees, i.e. the XML tags, but also the values of attributes which can come from an infinite domain.

While the applications of logic and automata models for the structural part of XML (without data values) are well understood by now [17,21,25], taking into account the data values presents entirely new challenges [23,26]. Most efforts in this direction concentrate on finding "well behaved" logics and their associated automata [6,5,4,10], usually with the focus on finding logics with decidable satisfiability problem. In [23] Neven et al. studied the expressive power of various data word automata models in comparison with some fragments of FO and MSO. A well-known result of Bojanczyk et al. [4] shows that FO^2, the two-variable fragment of first-order logic extended by equality test for data values, is decidable over

* Corresponding author.
E-mail address: domagojvrgoc@gmail.com (D. Vrgoč).

http://dx.doi.org/10.1016/j.jcss.2015.03.005
0022-0000/© 2015 Elsevier Inc. All rights reserved.

data words. Recently, Ley et al. showed in [3,8] that the guarded fragment of MSO defines data word languages that are recognized by non-deterministic register automata.

Data words appear in the area of verification as well. In several applications, one would like to deal with concise and easy-to-understand representations of languages of data words. For example, in modelling infinite-state systems with finite control [9,12], a concise representation of system properties is much preferred to long and unintuitive specifications given by, say, automata.

The need for a good representation mechanism for data word languages is particularly apparent in graph databases [1] – a data model that is increasingly common in applications including social networks, biology, semantic Web, and RDF. Many properties of interest in such databases can be expressed by regular path queries [22], i.e. queries that ask for the existence of a path conforming to a given regular expression [7,2]. Typical queries are specified by the closure of atomic formulae of the form $x \xrightarrow{L} y$ under the "and" operation and the existential quantifier $\exists$. Intuitively, the atomic formula $x \xrightarrow{L} y$ asks for all pairs (x, y) of nodes such that there is a path from x to y whose label is in the regular language L [7].

Typically, such logical languages have been studied without taking data values into account. Recently, however, logical languages that extend regular conditions from words to data words appeared [20]; for such languages we need a concise way of representing regular languages, which is most commonly done by regular expressions (as automata tend to be too cumbersome to be used in a query language).

The most natural extension of the usual NFAs to data words is *register automata* (RA), first introduced by Kaminski and Francez in [14] and studied, for example, in [9,24]. These are in essence finite state automata equipped with a set of registers that allow them to store data values and make a decision about their next step based, not only on the current state and the letter in the current position, but also by comparing the current data value with the ones previously stored in registers.

They were originally introduced as a mechanism to reason about words over an infinite alphabet (that is, without the finite part), but they easily extend to describe data word languages. Note that a variety of other automata formalisms for data words exist, for example, pebble automata [23,28], data automata [4], and class automata [5]. In this paper we concentrate on languages specified by register automata, since they are the most natural generalization of finite state automata to languages over data words.

As mentioned earlier, if we think of a specification of a data word language, register automata are not the most natural way of providing them. In fact, even over the usual words, regular languages are easier to describe by regular expressions than by NFAs. For example, in XML and graph database applications, specifying paths via regular expressions is completely standard. In many XML specifications (e.g., XPath), data value comparisons are fairly limited: for instance, one checks if two paths end with the same value. On the other hand, in graph databases, one often needs to specify a path using both labels and data values that occur in it. For those purposes, we need a language for describing regular languages of data words, i.e., languages accepted by register automata. In [20] we started looking at such expressions, but in a context slightly different from data words. Our goal now is to present a clean account of regular expressions for data words that would:

1. capture the power of register automata over data words, just as the usual regular expressions capture the power of regular languages;
2. have good algorithmic properties, at least matching those of register automata; and
3. admit expressive subclasses with very good (efficient) algorithmic properties.

For this, we define three classes of regular expressions (in the order of decreasing expressive power): regular expressions with memory (REM), regular expressions with binding (REWB), and regular expressions with equality (REWE).

Intuitively, REM are standard regular expressions extended with a finite number of variables, which can be used to bind and compare data values. It turns out that REM have the same expressive power as register automata. Note that an attempt to find such regular expressions has been made by Kaminski and Tan in [15], but it fell short of even the first goal. In fact, the expressions of [15] are not very intuitive, and they fail to capture some very simple languages like, for example, the language $\{\binom{a}{d}\binom{a}{d'} \mid d \neq d'\}$. In our formalism this language will be described by a regular expression $(a{\downarrow}x) \cdot (a[x^{\neq}])$. This expression says: bind x to be the data value seen while reading a, move to the next position, and check that the symbol is a and that the data value differs from the one in x. The idea of binding is, of course, common in formal language theory, but here we do not bind a letter or a subword (as, for example, in regular expressions with backreferencing) but rather values from an infinite alphabet. This is also reminiscent of freeze quantifiers used in connection with the study of data word languages [9]. It is worthwhile noting that REM were also used in [20] and [16] to define a class of graph database queries called regular queries with memory (RQM).

However, one may argue that the binding rule in REM may not be intuitive. Consider the following expression: $a \downarrow x(a[x^{=}]a \downarrow x)^{*}a[x^{=}]$. Here the last comparison $x^{=}$ is not done with a value stored in the first binding, as one would expect, but with the value stored inside the scope of another binding (the one under the Kleene star). That is, the expression re-binds variable x inside the scope of another binding, and then crucially, when this happens, the original binding of x is lost. Such expressions really mimic the behavior of register automata, which makes them more procedural than declarative. (The above expression defines data words of the form $\binom{a}{d_1}\binom{a}{d_1}\cdots\binom{a}{d_n}\binom{a}{d_n}$.)

Losing the original binding of a variable when reusing it inside its scope goes completely against the usual practice of writing logical expressions, programs, etc., that have bound variables. Nevertheless, as we show in this paper, this feature was essential for capturing register automata. A natural question then arises about expressions using proper scoping rules,