Journal of Logical and Algebraic Methods in Programming 85 (2016) 681–706

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

A descriptive type foundation for RDF Schema

Gabriel Ciobanu [a], Ross Horne [a,b,c,*], Vladimiro Sassone [d]

[a] Romanian Academy, Institute of Computer Science, Blvd. Carol I, no. 8, Iași, Romania
[b] Nanyang Technological University, School of Computer Engineering, Singapore
[c] Kazakh–British Technical University, Faculty of Information Technology, Almaty, Kazakhstan
[d] University of Southampton, Electronics and Computer Science, Southampton, UK

ARTICLE INFO

Article history:
Received 16 April 2015
Received in revised form 26 February 2016
Accepted 26 February 2016
Available online 2 March 2016

Keywords:
Linked Data
RDF
Schema
Type systems
Operational semantics

ABSTRACT

This paper provides a type theoretic foundation for descriptive types that appear in Linked Data. Linked Data is data published on the Web according to principles and standards supported by the W3C. Such Linked Data is inherently messy: this is due to the fact that instead of being assigned a strict a priori schema, the schema is inferred a posteriori. Moreover, such a posteriori schema consists of opaque names that guide programmers, without prescribing structure. We employ what we call a descriptive type system for Linked Data. This descriptive type system differs from a traditional type system in that it provides hints or warnings rather than errors and evolves to describe the data while Linked Data is discovered at runtime. We explain how our descriptive type system allows RDF Schema inference mechanisms to be tightly coupled with domain specific scripting languages for Linked Data, enabling an interactive feedback to Web developers.

1. Introduction

This paper is the second of two journal papers that address RDF Schema [9] from a type theoretic perspective. RDF Schema is a data-modelling vocabulary for the Resource Description Framework (RDF) [16], where RDF is a W3C recommended data format for publishing data on the Web.

This work spans two journal papers, since certain aspects of RDF Schema are best treated using a conventional approach to typing, whereas other aspects are quite unconventional even to the seasoned type theorist. The previous paper in the series [14] treated the conventional typing aspects of RDF Schema that concern familiar simple datatypes [43] such as integers and strings. As we know, if a type system guarantees that a variable is a string, but the same variable appears in an expression for integers, then a type error arises. We call such conventional type systems *prescriptive* type systems, since the type system prescribes that the variable concerned *must* be a particular type, hence can only be used in the manner prescribed. A prescriptive type system is appropriate for aspects of RDF Schema concerning simple data types. However prescriptive typing is less appropriate for other aspect of RDF Schema.

In this second paper in the series, that may be read independently of the first, we address less conventional aspects of RDF Schema types. The aspects we model concern opaque names, where there is no difference in the underlying structure of names that inhabit distinct types. In this paper, all resources are named by a URI — a Web address such as *res:Vitali_Klitschko* or *res:Udar*. Since all URIs are URIs, no runtime type error would arise if one URI is accidentally used in place of another

* Corresponding author at: School of Computer Engineering, Nanyang Technological University, Singapore.
E-mail addresses: gabriel@info.uaic.ro (G. Ciobanu), rhorne@ntu.edu.sg (R. Horne), vs@ecs.soton.ac.uk (V. Sassone).

http://dx.doi.org/10.1016/j.jlamp.2016.02.006
2352-2208/

URI. However, these URIs are intended to represent resources that are understandable to human beings. If resources are used in the wrong place in data, then the data may not make any sense.

The descriptive type system we introduce enables simple routine data-modelling slips to be detected. RDF is based on triples of URIs that represent how two URIs are related to each other. Here the property that relates the two URIs is also a URI, e.g. *free:government/politician/party*. The programmer may simply have accidentally switched the expected order of the two URIs related to imply something nonsensical, such as: "*res:Udar* is a member of the political party *res:Vitali_Klitschko*". Since our descriptive type system would describe that the above property relates politicians to political parties, then our type system issues a warning suggesting that either *res:Udar* is politician, or there is some problem with the data. In another scenario, the wrong person may be used. For example, the statement "*res:Wladimir_Klitschko* is a member of the political party *res:Udar*" would result in a warning, since Wladimir Klitschko, Vitali's brother, is not a politician. From meaningful warnings a human is likely to spot the problem with the data. Note that types are themselves named using URIs that do not impose any structure on the data itself. For example, the type politician can be represented by the URI *free:government.politician*.

The descriptive type approach illustrated above differers from the standard approach to RDF Schema inference [25]. In both of the above examples, standard RDF Schema inference would wrongly infer that *res:Udar* and *res:Wladimir_Klitschko* are politicians. In the descriptive typing approach a warning that presents a menu of options is generated. From the menu, the human reading the warning can select the best option, where the options include the standard RDF Schema inference along with several other possible courses of action. Furthermore, since, unlike errors, warnings may be ignored, the choice of inference may be suspended while the program continues. At a later point more illuminating data may be obtained that helps resolve the warnings; or, perhaps, the warning can be ignored indefinitely citing imperfect schema information.

This line of work also considers how descriptive types can be of assistance to programming languages that consume Linked Data [19,24,28]. Linked Data is data published on the Web according to certain principles and standards. The main principle laid down by Berners-Lee in a note [6] is to use HTTP URIs to identify resources in data. By using HTTP URIs, anyone can use the HTTP protocol to look up (dereference) resources that appear in data in order to obtain more data. All URIs that appear in this paper are real dereferenceable URIs that you can dereference by following the links in the electronic version of this article.

The descriptive type system introduced in this work can be used for typing programs, as well as data. For example, the descriptive type system can raise warnings when a query over RDF data involves properties that make no sense according the their schema, for example the subject and object of a statement are accidentally reversed. When a program is well typed, the program can be used in confidence that there will be no warnings and hence unwanted RDF Schema inferences will never be applied.

If you ask the Linked Data scientist whether there is any link between types in RDF and type systems, they will explain that there is almost no connection. Traditionally, type systems are used for static analysis to prescribe a space of constraints on data and programs. In contrast, types in RDF change to describe the data instead of prescribing constraints on the data. In this work, we provide a better answer to the question of the type-theoretic nature of types in Linked Data, by distinguishing between *prescriptive type systems* and *descriptive type systems*. The idea of descriptive types arose in joint work with Giuseppe Castagna and Giorgio Ghelli, instantiated here for our Linked Data scripting language [14]. Descriptive type systems, not formally related to this work, appear in work on logic programs, tree data structures and dynamically typed objects [22,33,15,5,17,26].

This work is an extended version of the invited conference version presented at PSI 2014 [13]. This version of the paper closes further the gap between the descriptive type system in the conference version and W3C standards. The new contributions compared to the conference version are:

- An extended introduction that presents the W3C standard RDF Schema inference mechanism called *simple entailment* and, by intuitive examples, compares the standards to the approach enabled by the descriptive type system in this work.
- An extended syntax and type system covering a larger subset of the RDF Schema standard, with features corresponding to not only *rdf:type* triples but also triples with *rdfs:subClassOf*, *rdfs:domain* and *rdfs:range* as the property.
- A proposition formally relating W3C standard simple entailment to inference for descriptive types; accompanied by common-sense recommendations about good practice for designing ontologies to work well with both descriptive type systems and the W3C standards.

This version also expands considerably the discussion and deals more carefully with algorithmic issues regarding generating and solving subtype constraints. Hence this version fully supersedes the invited conference version.

In Section 2, we provide a self-contained section that explains this work in the context of existing work on type systems for semi-structured data and RDF Schema. We present a motivating example of a scenario where descriptive typing can be applied to Linked Data to present meaningful warnings to a programmer that would like to interact with Linked Data. This section can be read separately without going into details of the type system or the scripting language.

In Section 3, we develop technical prerequisites for our descriptive type system. In particular, we require a notion of type and a consistent notion of subtyping. We develop these notions and present supporting results.

In Section 4, we continue the technical development of the type system. We introduce a simple scripting language for dereferencing resources over the Web and querying Linked Data in a local store. We devise an algorithmic type system