Science of Computer Programming 98 (2015) 28–51

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Grammar Zoo: A corpus of experimental grammarware

Vadim Zaytsev[a,b,*]

[a] Software Analysis & Transformation Team (SWAT), Centrum Wiskunde & Informatica (CWI), The Netherlands
[b] Universiteit van Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 2 November 2012
Received in revised form 26 May 2014
Accepted 20 July 2014
Available online 10 September 2014

Keywords:
Grammarware engineering
Grammar recovery
Experimental infrastructure
Curated corpus

ABSTRACT

In this paper we describe composition of a corpus of grammars in a broad sense in order to enable reuse of knowledge accumulated in the field of grammarware engineering. The Grammar Zoo displays the results of grammar hunting for big grammars of mainstream languages, as well as collecting grammars of smaller DSLs and extracting grammatical knowledge from other places. It is already operational and publicly supplies its users with grammars that have been recovered from different sources of grammar knowledge, varying from official language standards to community-created wiki pages.

We summarise recent achievements in the discipline of grammarware engineering, that made the creation of such a corpus possible. We also describe in detail the technology that is used to build and extend such a corpus. The current contents of the Grammar Zoo are listed, as well as some possible future uses for them.

1. Introduction

This paper contains a description of a method to compose a corpus of grammars in a broad sense. Having such a corpus could be profitable for mining new properties and patterns from the existing body of grammatical knowledge, for comparing grammar-based techniques and developing new ones. Formal grammars are inherently complex software artefacts, and until recently it was technically unfeasible to create such a large scale corpus, so in existing literature most case studies involve one, two or no more than a handful of grammars, and many statements about software language design remain statistically unchecked and empirically unvalidated or even unprovable.

The main contributions of this paper are:

- The Grammar Zoo as the big and still growing corpus of hundreds of grammars in a broad sense.
- An open source toolkit for supporting the creation and expansion of the Grammar Zoo.
- A Grammar Zoo entry metadata model for enabling efficient sampling and reuse.

The paper is organised as follows: Section 2 explains the problem in detail, motivates the need for its solution, sets goals, defines context and envisions possible problems. In Section 3 we revisit those grammarware engineering challenges that have already been addressed in prior work, and have made the current development possible.[1] In Section 4, the metadata

* Correspondence to: Software Analysis & Transformation Team (SWAT), Centrum Wiskunde & Informatica (CWI), The Netherlands.
E-mail address: vadim@grammarware.net.

[1] Parts of the section (e.g., Section 3.1) have previously appeared in a workshop publication [1].

http://dx.doi.org/10.1016/j.scico.2014.07.010
0167-6423/© 2014 Elsevier B.V. All rights reserved.

model for the corpus is presented and the tools available for grammar extraction, recovery and evolution are highlighted.[2] Section 5 lists the current contents of the Grammar Zoo and sketches directions for future work. Section 6 concludes the paper.

2. A repository of grammars

In [3], Klint et al. have defined the field of "grammarware" and identified its set of problems, promises, principles and challenges. The foundation of their work was formed by the vast existing body of knowledge about formal grammars, compiler construction, metaprogramming, source code analysis, term rewriting, parsing techniques, generative programming, attribute grammars, graph transformation and other adjacent fields. In the years after that, there have appeared many publications contributing directly to this domain, and the "engineering discipline for grammarware" from the ideal long term goal has turned into a technically achievable and partially even achieved objective. However, comparison of different grammar-based methods is still hindered by the relative lack of stability in grammar metrics and their sensitivity to many factors ranging from grammar development style (e.g., horizontal or vertical style of writing production rules has a substantial impact on the number of rules) to the choice of grammar-based technology (some syntactic notations are more expressive than others; some technologies implicitly expect grammars to be written with a specific kind of recursion, etc.).

In contemporary software engineering, especially in empirical studies thereof, a similar problem has been addressed by introducing a curated collection of code artefacts [4]. In model-driven engineering, many metamodels — artefacts commonly compared to grammars in the literature — have been collected in one place to form a corpus available in many formats [5]. Such reference corpora can be used as an input of various newly proposed analysis and transformation techniques, allowing their output to be measured and reported in a systematic manner.

In [6], van Wijngaarden states that a powerful and elegant language should not contain many concepts and should be explainable in few words. In [7], Wirth concludes that language simplicity should be achieved through modularity and not through generalisation. In [8], Hoare claims that in programming language design, simplicity is different from and more important than modularity. In [9], Mernik et al. argue that language modularity is positively influenced by the presence of a textual notation. In [10], Völter et al. tie the multitude and diversity of general purpose programming languages to their domain-specific optimisations. In [11], Chomsky elaborates that semantic and statistical considerations should be of no consequence to the grammatical structure of the language. In [12], Erwig and Walkingshaw maintain that language design should be semantics driven. In [13], Tratt establishes that the evolution of a domain specific language mostly involves adding functionality found in general purpose programming languages. In [14], Herrmannsdörfer et al. observe that modelling languages evolution is bound to requirements creep and technological progress. In [15], Hutchinson et al. claim that effectiveness of a domain specific language and narrowness of its domain are in inverse proportion. Many more claims like these can be found in academic and engineering publications about grammarware and related topics — most are backed by expert opinions and case studies of manageable size. However, we still lack the luxury of (re)formulating them as research hypotheses and subsequently validating against (a chosen part of) the corpus of grammars and languages. We construct the Grammar Zoo in order to enable such activities in the future.

It has been previously noted by Do et al. that obtaining the right kind of infrastructure for setting up experiments is nontrivial and labour intensive, and its usefulness has huge impact on future experiments [16]. According to Do et al., the users of such infrastructure mostly face the following challenges:

Supporting replicability across experiments. Homogeneity or well-documented heterogeneity of the collected artefacts and completeness of metadata are the key factors for the creators of the infrastructure, to help addressing this challenge [16–18].

Supporting aggregation of findings. Systematic capture of the experimental context is required to complement high replicability, in order to guarantee correct aggregation of findings from different experiments [16,19].

Reducing the cost of controlled experiments. In order to facilitate painless artefact reuse, artefact organisation needs to be standardised, they need to be complete in some sense (preferably by conforming to a well-defined completeness level) and require as little manual handling as possible [16].

Obtaining sample representativeness. The main problems foreseen by [16] in allowing the users to acquire representative samples, are small sample sizes and sampling bias. These are to be addressed by including many artefacts obtained from different heterogeneous sources.

Isolating the effects of individual factors. Isolating software language design concerns and decoupling conceptual modules within one software language have always been challenging problems, and still pose great difficulty. Since this is an open research question, we will not be able to prevent all problems that it leads to.

[2] Parts of the section (e.g., Section 4.3) have previously appeared in a workshop publication [2].