Quantitative Linguistic Computing with Perl

by

Fan Fengxiang

Deng Yaochen

2010

RAM-Verlag

Studies in quantitative linguistics

Editors
Fengxiang Fan (fanfengxiang@yahoo.com)
Emmerich Kelih (emmerich.kelih@uni-graz.at)
Reinhard Köhler (koehler@uni-trier.de)
Ján Mačutek (jmacutek@yahoo.com)
Eric S. Wheeler (wheeler@ericwheeler.ca)

ISBN: 978-3-942303-01-9

RAM-Verlag
Stüttinghauser Ringstr. 44
D-58515 Lüdenscheid
RAM-Verlag@t-online.de
http://ram-verlag.de

Preface

Empirical research in linguistics, in particular in quantitative linguistics, relies to a high degree on the acquisition of large amounts of appropriate data and, as a matter of course, on sometimes intricate computation. The last decades with the advent of faster and faster electronic machinery and at the same time growing storage capacities at falling prices contributed, together with advances in linguistic theory and analytic methods, to the availability of suitable linguistic material for all kinds of investigations on all levels of analysis.

Ideally, a researcher in quantitative linguistics has enough programming knowledge to acquire the data needed for his/her specific study. This is, however, not always the case. If professional programmers can be asked for help, most problems may be overcome; however, also this way is not always possible. And sometimes, it may be more awkward to explain a task than do perform it.

The selection of the appropriate programming language should not be conducted by taste or familiarity (as it is, unfortunately, very often even among programmers); instead, at least the following criteria should be taken into account:

1. Quality of the language. There is a number of quality-related properties of a programming language such as ability of preventing programming mistakes, readability of the code, changeability, testability, learnability. Unfortunately, these and other properties are not independent of each other; some of them compete (e.g. efficiency is always a competitor of most other properties) whereas others co-operate (e.g. readability advances most of the others). A programmer has to decide on priorities of the quality properties with respect to the individual task and application.
2. Nature of the problem. Every programming language has advantages and disadvantages also with respect to the task to be performed. One of the criteria often cited is the old distinction between low-level (close to the basic processor instructions and to the memory organisation of the computer) and high-level languages (with concepts close to the problems or algorithms). However, matters of efficiency etc. do not play a role any more (at least in the overwhelming majority of applications) since the compilers and their optimisers produce better code than most human programmers would be able to do. But there are still concepts and tasks that can be expressed in one language better than in another one, e.g. only few programmers would prefer a scripting language for the programming a data base.
3. Size and complexity of the problem. The larger a problem and the higher its complexity the more relevant become the quality properties of the language. If, e.g. several persons work on the same project, readability of the code is of fundamental importance but even if a single programmer

does all the coding of a complex problem he/she will encounter problems with his own code after some time if the programming language allows code in a less readable form. In any case, corrections, changes, and maintenance of a program depend crucially on some of the quality properties.

4. Security aspects. This is a simple matter: If the application you write is supposed to run in a environment that is accessible to potential attacks (e.g., the Internet or a computer network or if other users have access to the computer) and if the data your program works with should be protected, then a special focus should be put on security properties of the programming language. Scripting languages, e.g., are known to be frail, as a rule. You should make sure that the language you use has at least protection mechanisms you can switch on. Surprisingly, this aspect is very often neglected – even by institutions such as banks (Internet banking).
5. Reliability of the compiler/interpreter/libraries. Many popular languages are not *defined*. Instead, 'reference implementations' are offered. However, to be sure how a language element works you would have try it out in every possible form of use and combination with other elements – an unrealistic idea. Another aspect is even more significant in practice: Some languages, among them some of the currently most popular ones, are subject to substantial changes every now and then. The user of such a language is witness and victim of a ripening process (or mere experimentation): If your program will work with the next version of the language, is more or less a matter of chance. You should consider how much harm such a situation would do to your project if you decide to use a language that is not defined.
6. Frequency of application. It matters whether your program is an ad-hoc solution and will be used just once or a few times to evaluate some data and then will be discarded or whether it is meant to be useful for a longer time and may be changed and adapted for varying conditions. In the first case, not so much value is to be set on quality properties of the language; immediate availability of a practical solution may then come into foreground. In the latter case, however, readability, changeability and other properties play a bigger role.
7. Intended users of the software product. Similarly, if you alone will use a program, some disadvantages such as missing robustness or a bad user interface would not constitute a serious problem as you will exactly know which behaviour you have to expect and how to circumvent inelegance or even mistakes. If, on the other hand, an unknown number of unknown persons will use it you should base you product on reliable tools, among them the language you formulate your solution in.

The main problem, however, quantitative linguists will have to face – independent of how they are inclined to weight the criteria discussed above – is probably that they fail to have an overview about programming languages and their pro's and con's. Whenever a programming layman is asked for advice the probability is high that the answer will depend on personal taste and familiarity with a language and possibly on its current popularity. You should, at least, know what criteria to base your decision on; with an idea of your priorities at hand and after discussing them with a programming expert, you can increase the chance to obtain a good hint.

Perl belongs to the so-called scripting languages. To run a program written in Perl you need the Perl interpreter; it has to be installed on your computer before the program can be executed. The reason is that such a program is interpreted, line by line, each time it is started as opposed to compiled programs which can run without any interpreter. There are, again, pro's and con's of either solution. Scripting languages have, e.g. the advantage that a program can change its own logic and easily adapt its data structures while it runs, to an extent which is impossible with compiled programs – a comfortable but also potentially dangerous facility.

Linguists fancy, in particular, the powerful language elements of Perl, which enable a programmer to write powerful programs in very short time. This property is especially useful for string and text manipulation and analysis because many ready-made tools for string handling are 'innate' to the language. Advanced programmers will find it even more useful for Internet programming. A clear disadvantage is the not so readable program text which makes finding and correcting of mistakes sometimes awkward in long and complex programs. Therefore, careful formulations and exhaustive comments within the program code are strongly recommended. If these caveats are taken into account Perl can be used with much success with little effort – and make a lot of fun.

Reinhard Köhler

Table of Contents