Programming Languages as Ideal Languages

Patrick Hedfeld,
Dr. Bernd Ulmann

August 8, 2013

1 Introduction

Philosophy deals primarily with languages as such since it relies on language not only as a means of expressing thoughts but also as a tool for verification or falsification of ideas and concepts. As such it is natural to ask if there is something like an *ideal language* without the inherent idiosyncrasies of natural languages as Edsger Wybe Dijkstra illustrated them in the following quotation:[1]

> *"So-called 'natural language' is wonderful for the purposes it was created for, such as to be rude in, to tell jokes, to cheat or to make love in (and Theorists of Literary Criticism can even be content-free in it), but it is hopelessly inadequate when we have to deal unambiguously with situations of great intricacy, situations which unavoidably arise in such activities as legislation, arbitration, mathematics[2] or programming."*

Since languages shape the way we think by their respective possibilities of expressing thoughts and facts, there have been a number of attempts to develop artificial languages to be used in human interaction. One of the latest examples for these developments may be *Toki Pona* developed by Sonja Elen Kisa with the explicit goal of creating a language that could clarify the process of thought.[3]

Another well-known example of an artifical language together with a formal system to enable and simplify reasoning is Gottlob Frege's so-called *Begriffsschrift* which he published in 1879. The title of the book he wrote describing this language[4] states the ultimate goal he set out to achieve: "Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens" which can be roughly translated as being a formal language modeled on that of arithmetic [to aid] pure thought".

Going back further in history leads to the *Calculus Ratiocinator* of Gottfried Wilhelm Leibniz. His goal was even more ambitious than Frege's since he set out to develop a language which would allow to discover truth by applying rules of calculation to statements expressed in this language. Thereby he anticipated later developments which eventually led to programming languages like Prolog, short for *Programmation en Logique*, developed in the early 1970 by Alain Colmerauer.[5]

Obviously there was and still is a large desire to either remove those inadequacies and idiosyncrasies of natural languages that have accumulated over centuries

[1] Cf. [Dean et al. 1996].

[2] It should be noted that mathematical notation with its rather unusual embedment of natural language as glue between formal statements is an extremely mighty language in its own right (see [Iverson 1977] for example).

[3] Cf. [Dance 2007].

[4] Cf. [Frege 1879].

[5] See [Clocksin et al. 1987].

or, maybe less ambitious, to create new languages taylored to the specific purpose of aiding rational thought and reasoning. It is interesting that philosophers and linguists seem to have overlooked or maybe ignored that another field, that of computer science, already created a vast variety of artificial languages, so-called *programming languages* which are void of the drawbacks of natural languages and more than mere philosophical experiments. In fact, programming languages actively shape our world since they control machinery which is far too complex to be even described let alone controlled by natural languages.

The remaining sections of this paper give a short introduction to programming languages in general before focusing on how these languages may well be regarded as ideal languages and their ability of shaping the way we think and solve problems.

2 Language requirements

There is some principal criticism of ideal languages as pointed out by Ludwig Wittgenstein who criticized the very idea of an ideal language for its preciseness in definitions and expressions. He mentioned that the inherent impreciseness of natural languages should be considered an asset and not a weakness since many real-word problems are easier to express with a language that allows for some fuzzyness in its statements. As valid as this argument may seem it does not make ideal languages less ideal as the early works of Łukasiewicz and Tarski show which finally led to the development of fuzzy logic which shows that even these imprecisions and imperfections of natural languages are not inaccessible for formal systems and thus for ideal languages.

Any language requires well-defined terms and concepts, since everything else must be considered a *Privatsprache*[6], *private language*, a term coined by Ludwig Wittgenstein. Such a private language can only be understood by one individual, namely that who developed this language. Since such a private language is utterly useless, an ideal language has to fulfill a strict well-definedness requirement.[7,8]

Another aspect of the well-definedness requirement is completeness – a language needs sufficient tools to formulate statements about aspects of its problem domain.[9] Particularly nearly no aspect of interest in real life and especially in sciences can be described in a language void of precise means to denote quantities, relationships etc. Thus an artificial language like Toki Pona which only includes words to express the quantities zero,[10] one, two and many, can be safely regarded as being unsuitable for any but the simplest purposes of communication and rea-

[6] See [Wittgenstein 1945].

[7] See [Wittgenstein 2010][4.0311]:

> *"One name stands for one thing, and another for another thing, and they are connected together. And so the whole, like a living picture, presents the atomic fact."*

[8] Toki Pona is explicitly not well-defined (see [Kisa et al. 2005][p. 8]) which severely limits is usability as a *tool of thought* (as Iverson, see [Iverson 1980], would have put it):

> Do you see how several of the words in the vocabulary have multiple meanings? [...] Because of this vagueness, a speaker of Toki Pona is forced to focus on the very basic, unaltered aspect of things, rather than focusing on many minute details.

[9] The term *problem domain* is to be understood as a part of the real or imaginary world in this context.

[10] The value zero can only be expressed by *ala* which represents the concepts of "no" or "none" which stretches the idea of zero considerably.

soning.[11]

An interesting point has been made by Ludwig Wittgenstein who remarked that a (natural) language has a structure similar to a city which has grown over the centuries, a city with an unstructured and crowded center, a city with many suburbs and the like:[12]

> *"Unsere Sprache kann man ansehen als eine alte Stadt: Ein Gewinkel von Gässchen und Plätzen, alten und neuen Häusern, und Häusern mit Zubauten aus verschiedenen Zeiten; und dies umgeben von einer Menge neuer Vororte mit geraden und regelmäßigen Straßen und mit einförmigen Häusern."*

This argument has been often cited to discredit programming languages as not being "real" languages but these critics oversee that a language does not necessarily need hundreds of years to develop a rich and diverse history. A language like FORTRAN[13] which is considered being archaic by most modern programmers, has been actively developed for more than 50 years, resulting in a language[14] that incorporates many of the initial language features as well as modern concepts like object-orientation etc. Looking at matured programming languages from Wittgenstein's point of view shows that these often are not too different from natural languages with respect to their inner structure.

Another point of criticism concerning programming languages in comparison with natural languages is the alleged lack of literature written in the former. A closer look reveals that this argument does not hold either – there is an abundance in literature written in programming languages. Each and every program is a piece of literature worth of being read not only by a machine but also by a human reader as every professional programmer can confirm. John Bentley once asked

> *"When was the last time you spent a pleasant evening in a comfortable chair, reading a good program?"*[15]

In fact, one of the most noteworthy books on programming, Donald E. Knuth's "*The Art of Computer Programming*", relies on the formulation of ideas in a very basic programming language. In this particular case an assembler language for a hypothetical machine is defined in the first chapters on which all subsequent chapters rely, so this book (it is in fact a multi-volume book) can be regarded as being literature written in an ideal language, a programming language, intended explicitly for human readers. Most of the ideas expressed herein are of such great intricacy and delicacy that this book may also qualify as poetry which brings up the next topic: What about poetry? Is poetry possible in a programming language? Yes, it is. There is a variety of poetry contests in which poems must be written in a particular programming language. This is a far greater challenge than writing poetry in a natural language since the adamant syntax requirements of a programming language must be obeyed while still satisfying the literary requirements of a poem.[16]

---

[11] It should be noted that this restriction was built into Toki Pona explicitly to make it impractical to express large or small numbers. This decision is quite remarkable since only a few natural languages and populations are known to be devoid of a useful number system which can be regarded as a severe disadvantage. An early version of Toki Pona used the word *luka*, short for hand, to express the value five, which is still used by speakers although it has been removed from the langauge.

[12] See [Wittgenstein 1945][§ 18].

[13] FORTRAN is one of the oldest programming languages, developed by a team led by John Backus in the 1950s. The team's goal was to develop a language suitable for numeric computation and scientific computing.

[14] Namely Fortran 2008.

[15] See [Bentley 1986].

[16] See [Hopkins 2002], [Hillesley 2007] for an introduction to programming language poetry as well as some beautiful examples.

A further interesting point has been also made by the notorious Ludwig Wittgenstein who wrote *"Worte sind auch Taten"*[17] which can be translated as *"words are actions, too"* which is even stronger than his postulation that language is its use. This holds nowhere more true than for programming languages which ultimately control the actions of highly complex machines which in turn influence the real world by their decisions and transformations of information.

Last but not least a language which is to be considered being more than just a toy needs active speakers and writers. A recent study performed by the German organization *BITKOM*[18] revealed that 33% of the German population has at least basic experience in a programming language.[19,20] This clearly demonstrates the widespread use of programming languages in today's society.

Accordingly, this section may be concluded with the observation that a programming language fully qualifies as a language in its broadest sense, but without the unavoidable drawbacks of natural languages. The following section will demonstrate the influence languages have on they way we think by giving a simple example.

3 How programming languages influence thought

Early programming languages were solely intended to facilitate the description of algorithms in a way that enables a machine to execute a sequence of steps to solve a given problem. These early languages, ranging from so-called *machine languages* to early *high-level languages* were restricted to this particular direction of communication. Programs written in early languages like Alan Perlis' *IT*[21] are next to unreadable for human readers today, yet they facilitated the process of programming considerably compared with bare machine languages.

Early developments like the introduction of loops as part of a language changed this picture. While the underlying machine has no concept of a loop as such, i.e. the repetitive execution of a section of code under the control of some logical expression, programming languages now had this feature which allowed the ready implementation of basic mathematical operations like $\sum$ without the need to think about controlled branches and the like. Nevertheless these languages were still not powerful enough to express highly abstract and complex thoughts since there still was a siginificant impedance-mismatch between the problem- and solution-domain as well as a considerable gap between the programming language itself and the machine.

The first language that changed the way programmers think not only about notation of an algorithm but the way they think about problems in general, was *APL*,[22] developed by Ken E. Iverson et al. beginning in the late 1950s.[23] A simple example will show this qualitative difference made by APL.

If one is asked to compute a list of prime numbers, one straight-forward approach as implemented by many young programmers is that of repetitive division to test individual numbers for their primality. Such a solution could look like this short C-program:

```
#include <stdio.h>
```

[17] See [Wittgenstein 1945][§ 546].
[18] Short for *Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.*.
[19] See `http://www.bitkom.org/de/themen/54629_73532.aspx`, retrieved 03.08.2013.
[20] About the same amount of Germans speaks and understands French.
[21] Short for *Internal Translator*.
[22] Short for *A Programming Language*.
[23] See the wonderful paper [Iverson 1980].

```
int is_prime(int value)
{
  int divisor;

  if (!(value % 2)) return value == 2;

  for (divisor = 3; divisor * divisor < value; divisor += 2)
    if (!(value % divisor)) return 0;

  return 1;
}

int main()
{
  int i;

  for (i = 2; i < 101; i++)
    if (is_prime(i)) printf("%d ", i);

  printf("\n");
  return 0;
}
```

As simple as such an implementation is, one does not gain any additional insight into the properties of prime numbers by reading or writing it. The following APL-program also generates a list of prime numbers but is based on a completely different idea which is obvious in this language but not even *thinkable* in a language like C:

$$(\sim R \in R \circ . \times R)/R \leftarrow 1 \downarrow \iota R$$

At a first glance, a simple measure like lines-of-code shows that APL is a much more powerful language than C. But this comes at a price: The APL-solution looks rather incomprehensible for the uninitiated reader which is mostly due to the arcane character-set employed by APL. So let us dissect this program: Reading the line from right to left as is customary in APL,[24] the first operation is ιR, where R is a variable containing the last value to be contained in the list of prime numbers. Let us assume that R equals 100. ιR then generates a vector running from 1 to 100 and $1 \downarrow$ drops the first element of this vector. The resulting vector, running from 2 to 100 in unit increments is then stored in R by $\leftarrow$.

The left half of the expression is contained in parentheses: First an outer product[25] of the vector just generated is computed by $R \circ . \times R$. This results in a multiplication table without the 1-row and -column. And now comes the really brillant insight inspired by APL as a language: This multiplication table does not contain any prime numbers at all! So all that is left to do is to select all elements from the original vector stored in R which are not contained in this table. This is done by negating the result of the set-theoretic in-operation, written $\sim R \in$, which returns a vector containing true/false-values.[26] These are then used to select only the prime number elements from the vector stored in R by means of the select operation /.

[24] Right-to-left evaluation of arithmetic expressions is quite elegant since it often reduces the amount of parenthesis in contrast to the traditional left-to-right evaluation.

[25] An outer product generates a matrix out of two vectors by applying an operator like multiplication to every pair of elements from these two vectors.

[26] In fact, 1 and 0 may be used to denote these basic truth values.

In addition to that it is noteworthy that this approach is not only based on a rather unique insight in the interdependence between multiplication tables and prime numbers but also does not require any explicit loops. In contrast to this the naive C-implementation contains one loop to test an individual number for primality and a second loop in the main program. So here are two radically different approaches not only to programming but to the way one thinks![27,28]

This, in turn, shows that there are obviously different classes of programming languages based on the way they are normally used to express thoughts. Traditionally five classes of programming languages are differentiated: von Neumann *languages* which are also called *imperative languages* since programs are specified as a series of individual atomic steps, *object oriented languages* which are often based on von Neumann languages and most often extend these by an object model which controls the structuring of data and operations on data, *functional languages* which are based on the idea of nested functions in a mathematical sense, and, finally, *declarative languages* which specify the problem while its solution is left to the machine.[29]

These classes correspond to different programming language paradigms and it is difficult for someone being fluent in languages of one paradigm like that of von Neumann languages to become equally fluent in a language of a different class like a functional language. As Wilhelm von Humboldt put it, languages should be considered to be an expression of the *spirit* of a nation[30] which holds true for programming languages in particular. As a consequence, here are and always will be endless and fruitless discussions about which particular language or which basic paradigm is more mighty, more general etc.

4 Conclusion

If programming languages are not already ideal languages, it may be concluded that these languages are at least good candidates for this. The extremely short but amazingly rich history of programming languages shows many similarities to the much longer development of natural languages including the absorbtion of ideas from one paradigm into another one etc. One of the main advantages of programming languages in light of the study of the influence of language on thinking is their strictness and the inherent simplicity which often allows a thorough description of a language in a couple of pages or chapters in a book. Therefore it is comparatively easy to adapt a programming language to new challenges like testing new ideas and concepts.

As such, programming languages may very well qualify as ideal languages, thus Richard Rorty might have been finally wrong in his rejection of the quest for an ideal language.[31] Furthermore, programming languages and notation may also

[27] This has been conjectured to be true for natural languages for a long time and has been shown recently with quite some evidence, see [Boroditzky 2012].

[28] It should be noted that it is not sufficient to change the mere notation of a programming language to alter the way it influences thought. As an example, there are quite some developments regarding Arabic programming languages but all of those known to the authors, such as ARLOGO, AMMORIA etc. are all based on languages already well known such as LOGO, Scheme and C. So these languages may look like genuine Arabic programming languages but they are merely transliterations of languages already known.

[29] It should be noted that there exists a vast amount of programming languages which can be classified by such a taxonomy quite readily. In 2011 *HOPL*, short for *History Of Programming Languages*, listed 8512 known programming languages (see `http://hopl.murduch.edu.au`)! This is all the more remarkable since estimations concerning the number of natural languages are in the range of only about 7000 (see [Boroditzky 2012]).

[30] An idea every native French speaker or APL-programmer will readily accept.

[31] See [Rorty 1987].

be regarded as places of thought since they give an intellectual place where ideas can take shape and will be shaped by the place itself.

References

[Bentley 1986] Jon Bentley, "Programming pearls: literate programming", in *Communications of the ACM*, Volume 29, Issue 5, May 1986

[Boroditzky 2012] Lera Boroditzky, "Wie die Sprache das Denken formt", in *Spektrum der Wissenschaft*, 15.03.2012, see http://www.spektrum.de/alias/linguistik/wie-die-sprache-das-denken-formt/1145804, retrieved 07.08.2013

[Clocksin et al. 1987] William F. Clocksin, Christopher S. Mellish, *Programming in Prolog*, Springer-Verlag, 1987

[Dance 2007] Amber Dance, "In their own words – literally", in *Los Angeles Times*, August 24th, 2007

[Dean et al. 1996] C. N. Dean, M. G. Hinchey (Eds.), *Teaching and Learning Formal Methods*, Academic Press, 1996

[Frege 1879] Gottlob Frege, *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*, Verlag von Louis Nebert, 1879

[Hillesley 2007] Richard Hillesley, "Diving for Perls – the poetry of programming", in *Tux Deluxe*, April 2nd, 2007, http://tuxdeluxe.org/node/119, retrieved 25.07.2013

[Hopkins 2002] Sharon Hopkins, "Camels and Needles: Computer Poetry Meets the Perl Programming Language", in *The Perl Review*, issue 0.1, April 2002, http://www.theperlreview.com/articles/poetry.html, retrieved 25.07.2013

[Iverson 1977] [Iverson 1977] Ken E. Iverson, "Algebra as a Langauge", from *Algebra – An Algorithmic Treatment*, APL Press, 1977

[Iverson 1980] [Iverson 1980] Ken E. Iverson, "Notation as a Tool of Thought", in *Communications of the ACM*, August 1980

[Kisa et al. 2005] Sonja Elen Kisa-Richard, B. J. Knight, Robert Warnke, *Toki Pona, the language of the good – the simple way of life*, http://rowa.giso.de/languages/toki-pona/english/toki-pona-lessons.pdf, retrieved 29.07.2013

[Rorty 1987] Richard Rorty, *Der Spiegel der Natur*, Suhrkamp Verlag, 1987

[Wittgenstein 1945] Ludwig Wittgenstein, *Philosophische Untersuchungen*, http://www.geocities.jp/mickindex/wittgenstein/witt_pu_gm.html, retrieved 24.07.2013

[Wittgenstein 2010] Ludwig Wittgenstein, *Tractatus Logico-Philosophicus*, http://www.gutenberg.org/files/5740/5740-pdf.pdf, retrieved 24.07.2013