Intuitive Programming

Martin Vechev[1] and Eran Yahav[2]

1 **ETH Zurich**
`martin.vechev@inf.ethz.ch`
2 **Technion**
`yahave@cs.technion.ac.il`

Abstract

Despite significant progress on scripting languages, domain specific languages, and synthesis from a variety of higher level specifications (e.g., natural language descriptions), most programming tasks still require significant expertise. As more of our world becomes programmable, we need to make programming more accessible via what amounts to more relaxed notions of programs.

Intuitive programming eliminates the notion of a rigid programming language, and instead uses an imperative language where each step is described intuitively in a natural language (or a mixture of natural language and programming language symbols). An intuitive program does not have to follow a strict syntax, and is guaranteed to execute (in some sense) regardless of what the user writes. An intuitive program is similar to a reasoning system in that it executes based on some background knowledge and can infer facts that are added to this knowledge.

Intuitive programming is not the same as natural language programming (NLP) where for a given NLP query, the system returns to the user a set of programs that closely match the query. Fundamentally, NLP programming suffers from the problem that a large gap exists between the intent in the form of an NLP query and the returned program (usually in some low-level syntax) that now forces the developer to reconcile the two worlds.

An intuitive program can be viewed as a more relaxed scripting language, in which any program will be mapped to some meaning, and remain executable. Intuitive programming embraces ambiguity. It is quite possible that there will be more than one way to interpret an intuitive program. In such cases, the interpretation of an intuitive program may be biased towards the more commonly used interpretation (which is what happens in human interaction as well). In fact, the interpretation of an intuitive program may change over time, as more background knowledge is collected.

We will present several examples of intuitive programs, as well as some ideas on how such programs would be executed efficiently.

Keywords and phrases synthesis, statistical reasoning, machine learning

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

Simple Example

Consider the following simple example program:

```
pic = take a picture
dest = get contact named John
send pic to dest
```

Note the many levels of ambiguity in this program:

- Taking a picture is an activity with many possible parameters. For example: what application should be used to take the picture? what camera should be used (e.g., front or back of device)?

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–2

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- There may be multiple contacts named John. Should the picture be sent to *all* contacts named John? is that the first name or the last name of the contact? Should the picture be sent to *some* contact named John? If so - which one?
- Through which channel should the picture be sent?

The idea of intuitive programming is to execute the program based on the current available background knowledge. For example, the application used to take a picture could be the one that is most commonly used, the choice of camera can be the one most commonly used, or the most commonly used for taking pictures that are later shared with other contacts. The choice of contact to send the picture to could be based on the most recently accessed contact named John. The hallmark of intuitive programming is the continuous learning process in which background knowledge changes over time. In this simple example, the program may send the picture to a most recently contacted John Smith on one day, and to another recently contacted John Miller on another day.

Clearly, such relaxed notion of intuitive programming is not suitable for all domains. It is most appropriate in domains where: (i) there is some flexibility in the interpretation, and misinterpretation can be tolerated, and (ii) the user can easily determine whether the result is good or bad, such that corrective actions can be taken.

Related Work

The idea of programming with a natural language interface is very old. In fact, it is so old that it has already been scolded by Dijkstra in 1978 as a foolish idea [2]. Dijkstra's argument is that it is the narrowness of the man-machine interface (and in fact, any formal interface) that makes it effective. Making the interface wider, ends up creating additional work. In Dijkstra's words: *"We know in the meantime that the choice of an interface is not just a division of (a fixed amount of) labour, because the work involved in co-operating and communicating across the interface has to be added"*.

Using natural language for querying databases (NLIDB) dates back to the 1970s (e.g.,[3]) and has seen a lot of progress over the years [1]. A lot of the work in NLIDB focuses on generating SQL queries from expressive user queries in natural language (e.g., [5]).

There has also been some work on synthesis from natural language specifications. For example, Le et al. [4] described SmartSynth, a system for synthesis of smartphone automation scripts from natural language. These techniques work by synthesizing a program (once) from the higher level description. An intuitive program is re-interpreted considering the current background knowledge, every time that it is executed.

References

1 I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Natural language interfaces to databases – an introduction. *Natural Language Engineering*, 1:29–81, 3 1995.

2 Edsger W. Dijkstra. On the foolishness of "natural language programming". circulated privately, 1978.

3 Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2), June 1978.

4 Vu Le, Sumit Gulwani, and Zhendong Su. Smartsynth: Synthesizing smartphone automation scripts from natural language. In *MobiSys '13*, 2013.

5 Fei Li and Hosagrahar V Jagadish. Nalir: An interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 709–712. ACM, 2014.