EXPLORING VISUAL PROGRAMMING CONCEPTS FOR PROBABILISTIC PROGRAMMING

Gabriel Cardoso Candal

Dissertation done under the supervision of *Prof. Hugo Sereno Ferreira*

1. Motivation

There is, among several domains with interesting and relevant problems to solve (computer vision, cryptography, biology, fraud detection, recommender systems, ...) [1], the recurring necessity to be able to make decisions in the face of uncertainty using machine learning (ML) methods. One of the ML approaches that can me used to tacke the problem is to build a probabilistic model through the use of a probabilistic programming language (PPL), which lets you write your model as a program and have off-the-shelf inference [2]. Despite PPLs' power and flexibility, it's difficult for their target audience (data scientists, statisticians, mathematicians, ...) to adapt to the textual interface these languages provide, which lack the graphical intuition provided by other tools they are accustomed to. We believe this negatively affects productivity and slows down the adoption of PPLs [3].

2. Goals

We aim to overcome the difficulties in learning a new language, either for inexperienced developers or seasoned ones, such as learning yet another syntax or getting accustomed to the language's idioms. It is known that typical languages are difficult to learn and use [4] and that there are advantages in providing a language with a visual interface [5].

The goal of this dissertation was to develop a Visual Programming (VP) language with probabilistic programming capabilities. The target audience is programmers and data scientists with background knowledge in statistics which aren't still comfortable with full blown PPLs, but wish to educate themselves on the topic so they can eventually leverage the power of this novel machine learning approach.

3. Work description

During this dissertation we have developed a Visual Programming Environment (VPE), a tool which allows the user to design its model graphically and then either execute it or translate it into code (we chose to generate Infer.NET code, a probabilistic programming library from Microsoft that is compatible with CLR languages such as C#, F# or IronPython).

By doing so we have provided a tool which can be used to make usability research on. For instances, it allows future studies where it could be empirically assessed if the hypothesis that people with knowledge in statistics can learn to use a PPL faster using VP rather than the traditional textual form. We have also shown that it is possible to represent a probabilistic program in a graphical manner by representing several probabilistic programs with a variety of features in our tool, one of them being represented in Fig. 1.

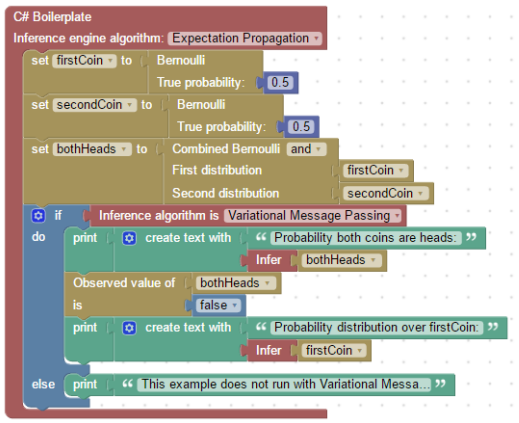

Fig. 1 – Example of a probabilistic model in our VPE.

During this process, we had to make choices regarding both the paradigms and the technologies in which we would base our VPE on, and we describe some of them below. In addition to detailing those choices, we are using the remainder of this section to elicit some VP concepts we find useful to have in a VP language for probabilistic programming.

3.1. Purely Visual vs Environment

The difference between a VPE and a purely visual language is that, while a purely visual language is a language *per se* (meaning that it there is a direct mapping between graphics and execution), VPEs offer a middle ground between regular textual languages and purely visual languages: they provide a graphical interface that can be used to generate code for a target language [6]. These two concepts are often refered in the literature as if they were mutually exclusive, but we included both of them: we have a VPE that can execute code, so that the user can get results sooner (does not require code to be copied to a separate environment) so that the visual feedback can be as immediate as possible, which is something essential in VP [7].

3.2. Dataflow vs Blockly

When choosing how to visually represent a model visually, we had to make a choice between these two alternatives. While visual dataflow programming is used in successful commercial applications such as RapidMiner and Blender, Blockly-like representations are most seen in educational tools such as MIT's Scratchy.

Due to the nature of our target audience, we thought that it would be suitable for our tool to offer a representation that could generate more human-like code, so that users could look use the tool as a safety net to generate code as if they were the ones writing it, until they are proficient enough that they won't need the VPE. Because of this, we decided to use a Blockly-like representation.

3.3. Visual Programming Concepts

There are a set of visual programming concepts that seem essential to include so that the user can be faster and make fewer mistakes when developing using any VP tool, such as:

- Comments
- Duplicate blocks
- Search blocks by name
- Use of icons and colors to convey meaning
- Get help (be able get details about a block's functionality)
- Serialization (save and load programs while also being able to share them)
- Zoom, block collapsing and compression (minize used space when details are not important)
- Minimap (have a bird's eye view over the program and be able to focus a specific area)

3.4. Type safety

Because we see a VPE based on a Blockly-like representation to be best suited for educational purposes, it is of the utmost importance to have type safety. This means that, besides not being able to represent a program with synactic errors, it is also impossible to make type mistakes (such as trying to use a boolean value to specify the mean of a gaussian distribution). This is done in a natural manner: like in a puzzle, blocks that violate type safety can't be connected. This is better than waiting for a compiler or run-time error, since the mistake is caught sooner and the visual feedback is as fast as possible.

3.5. Function Blocks

A visual programming concept that deserves a reference of its own are function blocks. This means empowering the user to create blocks made of blocks, which a concept analogous to functions or macros in programming. Providing such a feature helps transforming a VP language in one of an higher level, since users can iterate on the construction of these blocks and abstract complexity.

4. Conclusions

It is our belief that the graphical representation is clearer than its textual counterpart, mainly because parameters are named and due to the use of colors.

The main reason to use a VPE rather than the original textual representation is because of the safety net it provides and which prevents the user from making either syntax or type errors. Because of this, we believe this kind of visual programming environment can be useful to help data scientists who are inexperienced programmers take the first steps in using a PPL.

However, if we don't look at the VPE as an educational tool but rather as something that won't be transitional, we think that productivity-wise it falls short when compared to tools such as RapidMiner, Excel or even textual representations. The main rationale behind this is that, because we chose to adopt a Blockly representation, the resulting VPE is not a completely visual language since it violates concreteness and does not fully encapsulate textual programming concepts (such as variable declarations). If there was someone considering to develop a VPL to be used as a permanent substitute to the textual form of PPLs, we believe he should explore a dataflow representation, which is at an higher-level of abstraction than Blockly and has been used in successful applications both in industry and academia.

References

[1] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, second edition, 2010.

[2] Probabilistic Programming. *Handbooks in Operations Research and Management Science*, 10(C):267–351, 2003.

[3] Suresh Jagannathan. Probabilistic programming for advancing machine learning (ppaml). Available at http://www.darpa.mil/program/probabilistic-programming-for-advancing-machine-Learning, accessed in 3rd Jan 2016, 2013.

[4] G Lewis and G Olson. Can principles of cognition lower the barriers to programming? *Empirical Studies of Programmers: Second Workshop*, 2(15):248–263, 1987.

[5] K S R Anjaneyulu and John R Anderson. The Advantages of Data Flow Diagrams for Beginning Programming. 1992.

[6] Margaret Burnett. Visual programming. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 32(1-3):275–283, 1999.

[7] T.R.G. Green. Noddy's Guide to ... Visual Programming. *The British Computer Society • Human-Computer Interaction Group*, 1995.