THE ABSTRACTION LADDER

Summary of Barbara liskov's turing award lecture

Sai Sumanth Vanka

January 30, 2017

Abstract

In a reprise of her ACM Turing Award lecture, Barbara Liskov discusses the invention of abstract data types, the CLU programming language, clusters, polymorphism, exception handling, iterators, implementation inheritance, type hierarchies, the Liskov Substitution Principle, polymorphism, and future challenges such as new abstractions, parallelism, and the Internet.

1 Introduction

Power to abstract is fundamental to innovation. Abstraction is a technique for arranging complexity of computer systems. It works by establishing a level of complexity on which a person interacts with the system, suppressing the more complex details below the current level.
When i watched the lecture, i felt like the lecture is a small time travel machine to the seventies. Barbara talks about the history of how abstract datatypes and related concepts came to be. As someone who grew up with languages like Java, it is hard to imagine a world where all programming there is is literally done on assembler level.So, lots of the material presented should feel trivial to the modern mind, but Barbara manages to instill the spirit of the time into her listeners.

2 Data Abstraction Pre-History

- *Venus machine*
- *Venus Operating system*
- *Programming methodology*

In early 70's hardware was very expensive and software couldn't really account for cost. therefore the tentative to under buy the hardware and expect the program to make up the difference. Government was really concerned about this software contract problems as they want to built different defense systems, they are interested in. so, Barbara started working on it and developed Venus machine and Venus Operating system.

2.1 programming methodology:

There were two main concerns about software crisis.
1.How should programs be designed?
2.How should programs be structured?
For dealing this, Barbara read several research papers and started working on them. And the papers she discussed in the lecture are :

2.1.1 E. W. Dijkstra. Go To Statement Considered Harmful. Cacm, Mar. 1968

She supported that *GoTo* statement is not a good idea by describing Dijkstra's reasoning about quickness of program and debugging where well structured control structures are well working and go to leaves ambiguity. Then she also explained people opposition as they concern about performance and lack of expressive power in programming language.

2.1.2 N. Wirth. Program Development by Stepwise Refinement. Cacm, April 1971

This paper is about the *TopDown* methodology. She explained about wirth question "How do you design?" and also about TopDown methodology.

2.1.3 D. L. Parnas. Information Distribution Aspects of Design Methodology. IFIP Congress, 1971

She narrated the authors interest in structure of programs with the lines quoted "The connections between modules are the assumptions which the modules make about each other.”
Based these papers she wrote a paper and explained about $PARTITIONS$ in that.

3 Partitions (Idea)

3.0.1 B. Liskov. A Design Methodology for Reliable Software Systems. FJCC, Dec. 1972

The paper said, the entire system should be broken up into things like partitions. each partition has incited some hidden state and it provides a set of operations and the only way the partition interact is by making calls on one another’s operation.
she was interested in how can these ideas be applied to building programs. And she suddenly got an idea when she was working on the operating system, i.e."Connect partitions to data types". Later she sent this idea to ACM Sigplan-Sigops interface meeting and started working with Steven Zilles .They both worked on papers of S. Schuman and P. Jourrand, J. H. Morris and W. Wulf and M. Shaw. Where they found about the details of syntantic approach, protection of languages, impersonation and how global variable is harmful.
I liked the explanation of syntantic approach and protection of languages. syntantic approach is where they would give programmer the power to actually change the syntax of the language. The problem is it will become easy to write but it will become very diffivult to read the program by others. Protection is like encryption(seal) and decryption(unseal).

4 Abstract Data Types

B. Liskov and S. Zilles wrote a paper on abstract data types pointing out that encapsulation was key here. The paper proposed Abstract data types, Polymorphism, Static type checking (hoped), Exception handlingstatic . Abstract Data Types are:
1.A set of operations
2.A set of objects
Operations provide the only way that one can use the objects. The paper proposed: static type checking and sort of a sketch of a programming language. She took ideas in that paper and started turning them into real programming language which turned out to be called "CLU".

5 CLU

Why a Programming Language?She choose to work on programming language to ,
• Communicating to programmers
• Do ADTs work in practice?
• Getting a precise definition
• Achieving reasonable performance

The harder part was the language design. Language design is very akin to interface design. whenever you design interface you should be thinking about issues of expressive power, simplicity, performance, and ease-of-use. The other **Goals** they were concerned about building a language are
• Minimality
• Uniformity
• Safety
Uniformity – Data abstraction mechanism should not be be weaker than the built-in types.
Safety – Keeping mechanisms that could find errors both in compile time and run time.
She also kept some *restrictions* for language design i.e.
• No concurrency
• No go tos
• No inheritance

5.1 Some Assumptions/Decisions

• **Heap-based with garbage collection!**
This is good because when you are combining it with data abstraction, you don't have to worry about how big the objects are.
• **Separate compilation**
CLU is a compiled language.
• **Static type checking**
This is because one may get annoyed at the errors that he/she would have to find by debugging that a simple type checker could have removed in the first place.

Then she explained CLU mechanisms where she described about Clusters, Polymorphism, Exception handling and Iterators. she also explained that when objects came out from cluster, they sort of seal them up through CVT.so, they can convert back to rep type to Abstract type. Exceptions are very well dealt in CLU programming language.
Then Barbara went on into *distributed computing* and designed a language called **ARGUS** which is an object oriented language that allows you to write distributed applications. Later she shifted from programming languages to *system's area.* She worked om replication techniques, information flow control etc.,

6 Inheritance

When Barbara was invited to give a key note on oopsla when she looked into object oriented language then found that there was this inheritance mechanism.Inheritance was used for:
• Implementation
• Type hierarchy
she pointed that the implementation inheritance violates encapsulation. And gave example for type hierarchy as stacks vs. queues.

7 The Future

Modularity based on abstraction is the way things are done.
New abstraction mechanisms
Massively Parallel Computers
Internet Computer
I feel like future challenges are such as Storage and computation, Semantics, reliability, availability, security etc.,

8 Conclusion

Abstractions can prove useful when dealing with computer programs, because non-trivial properties of computer programs are essentially undecidable (see Rice's theorem). As a consequence, automatic methods for deriving information on the behavior of computer programs either have to drop termination (on some occasions, they may fail, crash or never yield out a result), soundness (they may provide false information), or precision (they may answer "I don't know" to some questions).
Languages that implement data abstraction include Ada and Modula-2. Object-oriented languages are commonly claimed to offer data abstraction; however, their inheritance concept tends to put information in the interface that more properly belongs in the implementation; thus, changes to such information ends up impacting client code, leading directly to the Fragile binary interface problem.

References

• turing award vedio lecture by Barbara Liskov.