Keystone Final Submission Report on

Bridge: Development of an Intermediary Programming Language

Vidhu Nath

Advisor: Andrei Lapets, Ph.D., The Rafik B. Hariri Institute for Computing and Computational Science

**I. Introduction**

This report contains the details of the Bridge programming language, from details of the development and the background to the general uses it serves and the functionality it contains. Many details are summarized for the sake of brevity, but the general concepts are clearly enunciated to provide clarity in understanding the various aspects of this project.

**II. Background Knowledge**

All ***programming languages*** have the basis in their operation: they parse ***code*** that is typed by a user and change the bits of the machine the code runs on in order to execute the user ***program***. Much like many spoken or written languages, programming languages have their own ***grammar*** which determines the ***syntax*** of the language. The grammar sets the rules for what is allowed by the program – i.e. what functionality is contained within the language – via the ***operational semantics*** of the language and the syntax provides the user the correct way to type the language. These are two of most important foundations for developing programming languages.

At the highest level of a program is the code that is written to provide the instructions for the program. This code, which is called the ***concrete syntax*** is then deconstructed into pieces called ***tokens*** by a component called the ***tokenizer***. These tokens are essentially each typed code snippet, and they are stored in a list that is passed to the ***parser***. The parser takes the list of tokens, compares them to the syntax, and sorts them into the correct execution sequence, often called ***abstract syntax trees***. Once ordered, the sequence is sent to the final component, the ***interpreter***, which compares the syntax sequences to the grammar to execute the code properly. Interpreters are usually written in a base language, so once the interpreter begins execution, the base language parses the interpreters instructions in much the same process as described above; the result, however, is a set of instructions written in ***assembly language***, which is the lowest level language on a machine. This assembly language is the interface between programming languages and the ***bits*** on the machine; as per the instructions, the assembly flips the appropriate bits and the hardware executes the code. A fourth component, called the ***compiler***, serves as the means for determining which bits are flipped by the instructions of the assembly language.

The process that was used to develop Bridge follows the steps described above. Thanks to the efforts of Eric Dunton, Ellie Vial, and Andrei Lapets, several tools – called UxADT and Imparse – were used to provide a cross-language compiler which takes care of the tokenizing and parsing steps. Thus it was incumbent upon this student to implement the interpreter, the details of which are below in Section IV.

**III. Purpose**

Bridge was not developed to produce software. There are a plethora of programming languages in existence, and many more being constantly developed. There is a disproportionate number of positions, however, which require the technical knowledge and expertise of programming languages when compared to the number of qualified people. Due to misconceptions about what programming languages are, how they operate, and what software development is, leading to stigmatization of developers and programmers in a sense, many people lack the foundational skills and logical mindset to approach programming.

As a solution to this lack of education regarding programming, Bridge was conceived and developed. As a purely educational tool, Bridge has constrained functionality with a simple syntax, so that users are able to easily grasp the language. Many other languages boast vast amounts of functionality, which is useful for programmers, but is often overwhelming for those people who do not have a technical background, such as those in the humanities. By limiting its functionality to only core processes, Bridge is able encapsulate the different concepts of programming – such as variable assignment and initialization, iterative processing, conditional flow, and operator precedence – and provide a simple interface with many examples to ease users into understanding programming.

**IV. Functionality**

Bridge is an interpreted language, so the interpreter is the component which executes the user code. The base language for Bridge is JavaScript, which is a language used for functionality in websites. Therefore, Bridge is contained within its web interface – there is no need to download any external files or programs to run Bridge.

There are four main programs that can be run via Bridge: printing values to the screen, assignment values to variables, setting a condition, and iterating a set amount of times. There is also the minor program of ending the running program. These are designated by **Print**, **Assign**, **If**, **For**, and **End** keywords. Each of these keywords are used to specify how the interpreter will execute the code. Printing values means that the value is simply shown to screen; it is a useful way in determining the current state of a variable. **Assign** statements are how variables take on values to use later; variables can be reassigned, but that is limited only to outside of the other programs. Statements beginning with **If** work similar to conditions in spoken/written languages; if the condition is true, then the inside program will run. Otherwise, the inside program will not run. The outside program would run regardless. **For** statements imply looping, and whatever the inside program is, it will execute the amount of times set by the **For** loop.

The other components of Bridge, primarily the **Formula** and **Term** choices, provide the arithmetic and logical base for the programs above to run. **Formula** provides the logical operations needed for the **If** statements’ conditional executions; the ***primitive***, or most basic, values for **Formula** are **true** and **false**; every other grammar choice in **Formula** results in either **true** or **false**, such as operators like Greater Than, or Less Than Equals, or Equals. **Term** contains the arithmetic operations such as addition, multiplication, and exponentiation. Within **Term**, number and variables are defined so that they can be used in other parts of the program. Both **Formula** and **Term** have subclasses, called **Component** and **Factor**, respectively, which are used to enforce the precedence of operations (essentially, the order of operations). Both **Formula** and **Term** are themselves contained in a superclass called **Expression** to manage the situations when either choice is acceptable, such as assignment values to variables.

In the Bridge website – which is currently offline, so it can only run by manually opening the website inside the browser through the Open option in the File menu – there are several main features. The first is the inclusion of the ***Backus-Naur Form*** of the language, which is a representation for the rules of the language. Here, users will be able to see the actual language rules, and can determine what choice is allowed for each program. The second feature is an area containing a host of examples which can be used to understand the language or for users to attempt their own programs. There is also a text area which contains the abstract syntax tree for the code that was executed, allowing users to see the structure of the language and how their code was broken up.

**V. Future Plans and Conclusion**

The Bridge language is officially complete and contains all of the functionality it was originally slated to have; however, like most software projects, there will be many updates in the future to expand or focus on functionality, but not to the point where the original use is lost. Most updates will be concerning the ***UI***, which is the user interface, of the website, so that future users will have a smoother website to look at and interact with. Tutorials will also be posted so that users who are unsure as to the nature of the examples will be able to gain a better understanding of the language.

Bridge as a language should be used to comprehend programming, not to implement software. It contains the basic functionality of most languages without delving into the technical details that many other languages employ. Thus, it is useful as an educational tool in understanding the concepts of programming and gaining the technical mindset necessary for programming.

**VI. Acknowledgements**

Thank you to Professor Andrei Lapets for guiding me through this project as my advisor and as the developer of UxADT. Thank you to Eric Dunton and Ellie Vial for the immense contributions to the program suites UxADT and Imparse, which were instrumental in the implementation of Bridge. Thank you to the Kilachand Honors College and to Boston University for allowing me to pursue this project and for providing any of the resources that were used in the making of this project.

For further information, please reference the Bridge repository hosted on Github at:

<https://github.com/vidnath/Bridge_Project>