**CPSC 311 – Term Project**

*Proposal: Logical Expression -> Logical Circuit (LETLC language)*

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Student # | CS ID | E-Mail |
| Zhe Li | 88792486 | d3q1b | lizhe918@alumni.ubc.ca |
| Maxwell Yang | 96245428 | r0g2b | shukany@alumni.ubc.ca |
| Eric Yan | 22548424 | g5h2b | ericy676@students.cs.ubc.ca |
|  |  |  |  |
|  |  |  |  |

The Logical Expression -> Logical Circuit Language (or simply LELCL, pronounced “Leco”) is a domain specific language designed to autonomously generate logical circuit diagrams from user-provided logical expressions. To elaborate, the user can enter a logical expression in a concrete syntax similar to the Racket language into a Racket file. The LELCL will then parse the given expression into an abstract syntax and proceed to interpret it into a corresponding circuit diagram. Due to familiarity, we opted to base our language on Racket. We also anticipate heavy use of Racket’s pict library, to render our circuit diagrams. The primary motivation behind the development of this language is to provide a clearer visual representation of unfamiliar Boolean algebra and circuit design concepts for students. Both of which are heavily emphasized in lower level undergraduate Computer Science courses. The language may also be used by course staff to help the development of new exam/homework questions, as well as solutions to some logical statement/circuit design problems.

Our core goal is for the language to be able to convert simple logical expressions into their corresponding circuit diagram. (**INSERT PHOTO from Logism**) Notice that each Boolean variable (A, B, C etc.) only occurs once due to complexities with reorganizing the wires and placing the corresponding gates.

A ^ B

~(A ^ B) ^ C

~(A v B) v C

(A ^ B) v ~ (C ^ D)

For our full goal, we want to handle complex logical expressions with the same variable appearing multiple times at distinct locations (e.g. (A ^ B) v (A ^ C)) since this would inevitably cause the overlap of wires. We also want to implement methods in the interpreter that can simplify the logical expression to produce simpler expressions and hence simpler circuits. Another idea that we want to achieve is to have different coloring for different wires to show if their current state (either true or false). Finally, if time permits, we may want to implement a simple GUI and have interactable circuit diagrams.

To realize the language, we will split the project into multiple milestones. The first step in our strategy is to perform sufficient background research on relevant topics. The central research topic would be methods to convert logic formulas into circuit diagrams. Consequently, this would involve researching about formula parsing, simplification of logic expressions, diagram generation, image creation for functional programming (or recursive algorithms). All group members will contribute to the research of these core topics above. Furthermore, as the Racket pict library is core to our language interpreter, we will likely dedicate 1-2 group members to specially focus on the documentations, tutorials and demos involving the use of the library. Some other topics that we will might also conduct research on include syntax choices, impact in educational settings and possible GUI designs etc. Once we have completed most of the research (1 week before background research report submission), we will compile our results into a single background research document highlighting our research on the topic.

The next milestone of our project is the proof of concept and plan. At this stage, we will have clarified our “minimal” core goals and provided some of our more “ambitious” full goals. To demonstrate our capability for accomplishing the core goals, we will elaborate on how to achieve some key components of the project by providing written explanation, diagrams and code snippets. Some of the interesting parts that currently comes to us includes how the parser will be written, how each logical gate is defined and stored, how we might recursively draw the circuit diagram. We will also provide possible solutions to achieve the more “ambitious” full goals. They will be in a similar format as the minimal goals. Considering the importance of achieving the core goals first, all members in the group will initially collaborate on providing methods to achieve them. We will also attempt to build a language that supports most of the features that we specified in the main goals. After completing this phase, we will then move on to the full goals.

(Poster part)

The last and final part of our project is the actual implementation of our language, LETLCL. Instead of inputting a .txt file into the program which would require some extra time researching on topics such as I/O streams and even interfaces, we will choose a easier approach, which is to use the classical “match” method provided in Racket to parse the input, considering how similar a logical expression looks compared to a Racket expression. Most of our time, however, will be spent on the interpreter, which is supposed to output images of the corresponding logical circuit. Based on the research we would have done on the graphics library of Racket, we would be able to know about shape drawing for parts like logical gates. The hard part will be the wiring and position of each gates and inputs, but since we would have done the proof of concept and plan by this point, the actual implementation should heavily reply on the results we came up with, so it should not be a major issue. After using everything we would have at that point, the program should be executing as expected.

**Exact project to be determined**

1. **The Project Topic and Type**
2. The Project Topic
3. How the Project Belongs to the Proposed Type
4. What the Project Should be after Finished
5. **Plan for Subsequent Milestones.**

**References**