Matthew Haigh (mlh2196)
Theodore Ahlfeld (twa2108)
Gideon Mendels (gm2597)
Konstantin Itskov (koi2104)

Proposal for TED

1. Introduction

In recent years, as big data analysis has become a more common practice among technology companies and university research teams, the demand for raw data to analyze has grown at an increasing rate. At first, this need for data was successfully fulfilled through a process of data extraction called web scraping/crawling. Web scraping is a technique used to develop a program, which connects to public-facing data sources, such as websites, web APIs, databases, etc., and is responsible for pulling unstructured data found on these sources and transferring it into a structured and formatted data, allowing for easy manipulation and analysis. To construct this type of application, a programmer must possess a strong understanding of the web, as well as an intermediate understanding of HTML, CSS, JaveScript, etc. As computers have become faster and more powerful, the amount of data scraping required continually increased, as did the demand for ever larger amounts of data. In response, programmers began utilizing multiprocessing and multithreading techniques to accommodate simultaneous extraction. However, these approaches require programmers to be far more skilled in systems language programming, such as C/C++ or Java, which made it more difficult to find engineers with the necessary skill set. In order to resolve this need for technical expertise, we are introducing a programming language, which abstracts the low level scraping systems and optimizations, allowing for a non-technical researcher to develop a set of instructions for data scraping. The immediate benefactors of this language will be university research teams working on new, cutting edge technologies, for which structured data does not yet exist, small

companies developing machine learning-based products, and professional individuals looking to work on innovative problems for which they will need to collect new pieces of information.

2. Language Description

TED (Token Expression Determiner), will enable the user to “crawl”, or parse through web pages and complete procedures according to patterns found. The user will indicate url address(s) and a series of patterns with their corresponding procedures. TED can recursively call the links found in the url and will output the resulting data (indicated via the print procedure) to a file or to the screen as indicated by the user. The .ted file will compile to Java which will be saved as .tedp. The user will run the program as indicated in section 3.

3. Language Invocation

The code for the proposed programming language will have the ability to either compile from a source file with the extension .ted or directly from the command line. In both cases the programming language will compile into target language and be stored in an output file with the .tedp extension. However in the case of command line invocation the compiled program will also be executed. Note that the invocation of the compilation process will require an input string that identifies the base web address the program must crawl. The input string can contain more than one base address in which case the compiled program will be executed on each of the strings sequentially. To do so each base address may be separated by a newline character and any white spaces will be ignored. Thus, the list of base strings may be written into a text file with one string per line and provided to the program’s input as input.

```
// Compilation from a file source
$ tedc filename.ted
$ echo "http://www.sample.com/" | ted filename.tedp
```

```
// Invocation from command line
$ echo "http://www.sample.com/" | ted pattern '{ action }'

// Input list of base addresses
$ cat example.txt
http://www.example1.com/
http://www.example2.com/

$ cat example.txt | ted filename.tedp
$ cat example.txt | ted pattern '{ action }'
```

4. Syntax

a. Comments

Comments may non compilable annotations inside of the source code and do not carry over to the target code. The may be multi-lined or single-lined or even nested within themselves.

```
/* Comment */
// Single line comment.
/* Multi line comments

       /* Nested comment */
       // Nested single line comment

are written like this. */
```

b. Operators

Three sets of operators exist within the language: arithmetic, relational, and logical.

```
// Assignment
= += -= *= /=

// Arithmetic
+ - * / % ++ --

// Relational
== != > < <= >=

// Conditional
and or not

// Logical/Bitwise
land lor lnot lxor
& | ~ ^
```

c. Primitives

There are a number of primitives existent within the programming language syntax. These consist of the int, float, str, list, and file. Note that a list is a collection of same type variables accessible in a sequential manner.

```
int x = 100;
float f = 100.9;
str name = "ted rocks";
list lst = ["ted", "is", "really", "COOL!"];
lst[0] = “Blue”;
FILE fp = “/file/path/to/file.ext”;
```

d. Data Structures

The programming language contains two non-primitive data types corresponding to the page being currently scraped which acts like a tree root. And the elements appearing inside of the tree like structure corresponding to the tags store inside html. Each of these trees will be connected through a child parent hierarchy. The tag element inside of the Element structure will contain the string representation of the html tag and its attributes corresponding to the particular tree node.

```
Page page {
    str base_url;
    list children;
}

Element elm {
    str tag;
    list children;
}
```

e. Flow Control

There are a number of flow control statements that are able to manipulate execution based on a variety of conditional expressions. These include if-then-else statements, for loops,

and while loops. Note that the for loop come in two varieties one with conditional execution and the other with iterable execution.

```
// If-then-else statement
if (condition) {
        // Statement ...
} else if (condition) {
        // Statement ...
} else {
        // Statement ...
}

// For loops
for (initialization, condition, modifier) {
        // Statement ...
}
for (type name in list) {
        // Statement ...
}

// While loop
while (condition) {
        // Statement ...
}
```

f. Function Declaration

A simple function definition which declares the input, output, and execution code of the function. A return statement must be provided within every function which returns a value corresponding to the same type as was defined in the function signature. A number of arguments may be provided through the function signature as well and must have their type declared.

```
type name(type arg1, type arg2, ...) {
        // Statement ...
       return 0;
}
```

g. Built-in Functions and Methods

- The **select** method accepts a string with a CSS selector and returns a list of Element corresponding to that selector. select is a built-in method for the Page and Element built in data structures.

```
Element select(str css_selector);
```

- The **attr** method parses the String content of Element.tag and returns the value of the HTML attribute requested.

```
str attr(str attribute);
```

- The **print** function prints the content of the input value to the standard output. This function's input value may be formatted in c/c++ style printf and the arguments provided after the input value. This function returns 0 if successful or an integer if an error occurred. You must specify a file to print to whether stdout, or custom file to print to.

```
int print(file, str format_string, type arg1, type arg2, ...);
```

- The **range** function produces a list of integers that allow for an easy way to iterate over them in a way the shorthand if-then-else statement would be easily able to iterate over simple integers.

```
list range(min, max, step);
```

- The **head** function produces the first element of the list.

```
type head(lst);
```

- The **tail** function produces a list without the head element of the original list.

```
List tail(lst);
```

- The **concat** function produces a new list that is the concatenation of two other lists.

```
List concat(lst1, lst2);
```

- The **append** function adds the data to the front of a list.

```
List append(data, lst);
```

- The **get** function produces a Page data structure which in turn will contain the links to all child html tags of that page in a tree form.

```
Page get(string_of_website);
```

- The **close** function closes the file descriptor

```
close(FILE);
```

h. Global Variables

At runtime the language will generate global variables similar to the php context variables concept. These will contain information such as the page currently being processed by the target program as well as matched elements.

```
$PAGE, $ELMS
```

5. Sample Program

```
//This program would would collect the title of the www.nytimes.com website.

// This is how global variables are essentially will be created.
//Page $PAGE = get("www.nytimes.com");
//list $ELMS = wlf.select("a .spf-link")

//print every title to stdout
echo "http://www.nytimes.com" | ted title {
        for(Element element in $ELMS){
                print(stdout, "%s",element.attr("text"))
        }
}
```