Jack Wang
yw4014

I have the two following proposal ideas. Please let me know whether both of them are good.

Project Idea 1: Web Scraper

This project aims to build a functional web scraper for data analysis purposes. It will selectively visit and crawl web pages, extract hyperlinks and other meta information from them, and build model representations of the pages for analysis and summarization.

The project has three identifiable components that can be further broken down:

1. The downloader
This part acts kind of like the lower-level internals of a web browser. It handles the actual requests and data flow, allows imposing rate limits, controls parallelism on the connection level, and is responsible for delivering the page data for analysis and filtering out unwanted binary content such as images early.

The challenges here include handling different forms of chunking, compression, and input encoding. For simplification, only certain methods can be implemented first, allowing scraping within one server using a consistent set of rules.

An important part of this component is also the data delivery, which in some cases can be streamed into the processing components before the entire page is loaded.

2. The processing pipeline
This part is responsible for parsing the hyperlinks out of the response, deduplicating them, controlling the crawling depth, and feeding new paths back into the downloader.

It needs to deal with potentially malformed input, make sure the parsed content is workable and efficiently process the input, producing both the output model and more potential links to crawl.

3. The data model
Since a web page can be represented as a graph, where links are nodes and the existence of a link on one page leading to another as edges, it’s possible to model it as such. The crawler could take this approach, or model the page in a simplified way, such as a tree or a simple unique list.

The data model can be used for querying after the page has been crawled, or if the earlier components allow it, even during the crawling itself in a “streaming” mode.

Project Idea 2: Implementation of Forth-like programming language in Haskell

The Forth programming language appeared and developed in the 1970s. Forth is a stack-based programming language, in which all expressions are written in prefix notation. It is easy to parse source code written in such a language. However, it takes some effort to understand what it is computed: all operations manipulate stack, without the use of local variables.

For example, expressions like (2 + 3)*3 - 5 would be written as follows:

```
2 3 + 3 5 -
```

Numeric literals are simply put into the stack once read. When the interpreter reads an operation, e.g. +, it applies it to whatever it finds in the stack.

This approach applies not only to arithmetic, but to control flow expressions as well.

```
cond IF then-clause ELSE else-clause THEN
```

In this cond can be whatever expression that puts a boolean onto the stack. When the interpreter reads the IF keyword, it reads all expressions till ELSE and executes them only if the value on the stack was truthful. Then

```
2 3 < IF CR .” value was truthful” ELSE CR .” value was false” THEN
```

This expression should print “value was truthful”, since 2 < 3.

This radical idea of using a stack to pass arguments removes any need to work with variables all arguments are passed via stack. However, it is necessary to provide a lot of subroutines that would manipulate the stack: duplicate values, swap them around, etc.

Some of the standard subroutines for stack manipulation and their stack effects:

```
DUP  ( w -- w w )
OVER ( w1 w2 -- w1 w2 w1 )
```

A list of such subroutines can be found here: https://gforth.org/manual/Data-stack.html

We propose to implement a subset of this language in Haskell.