G53IDS Project Proposal

A JavaScript Optimiser written in Haskell

Nick Brunt

Project Proposal

Aim

To develop a program which optimises a JavaScript file by reducing the number of characters used to express its intentions. The newly compressed JavaScript code must maintain the semantics of the original version.

Background

JavaScript compression (or minification as it is sometimes called) is an important field in web development. When a webpage is loaded in a browser, all the dependencies of the page are loaded too and these often include JavaScript files. The smaller these files are, the faster the page loads, so developing an optimiser which can reduce the size of a script file whilst maintaining its functionality could make a noticeable difference.

With an optimisation program at his/her disposal, a web developer is free to write expressive, well laid out code, without having to worry about the file size.

Motivation

All extensive JavaScript libraries provide a compressed version of themselves. For example, jQuery is a library that provides many features and is included in millions of websites across the Internet. Their latest version (1.6.4) in its uncompressed "developer" format is 242KB whereas the compressed version is 90KB, only 37% of the original size.

For a single user, this may not make a huge difference, but when you consider how many people surf the web every day, the savings scale up rather quickly. Consider the following:

If jQuery is included in 1 million websites and each of those websites is viewed on average 100 times per day, that's 100 million requests for the jQuery library. Having compressed the library, 15.2 TB of data does not need to be transferred. This saves both network and server costs.

Now what if the compressor had managed to shave off just one more byte of data? A further 100 MB could have been saved. People are spending increasingly more time online and websites are using increasingly more dynamic content. When every character counts, JavaScript optimisers are very necessary.

Objectives

The primary objective of this project is to build a Haskell program which takes a JavaScript file as input and outputs a new JavaScript file which will behave in the same way when executed as the original. By "behave in the same way" we mean that any input should produce the same output as it would have with the original code. It is possible that the space and time complexity of the code will change, but as long as the results of execution remain identical, this is acceptable.

The new file must either be of a smaller size than the original, or exactly the same size, it must not be larger. It may be impossible to compress a certain file if it is already in its most optimal form.

We will attempt to obtain a smaller file size by using the following types of optimisation:

- Code compression:
    - Removal of characters that are unnecessary for execution such as comments, excess whitespace, and semi-colons before closing braces
    - Shortening of variable names
    - Converting certain statements to the corresponding shorthand equivalents – for example
        - Tertiary conditionals:

```
if (a) b(); else c();
```

would become

```
a ? b() : c();
```

        - Array declarations:

```
a = new Array
```

would become

```
a = []
```

- Partial evaluation:

Example 1

```
function errorMsg(msg) {
  alert("Error: " + msg);
}
errorMsg("Invalid input");
```

would become

```
alert("Error: Invalid input");
```

Example 2

```
function powerOf(x, y) {
  if (y <= 0) return 0;
    else if (y == 1) return x;
    else {
      return x * powerOf(x,y-1);
  }
}
powerOf(x,3);
```

would become

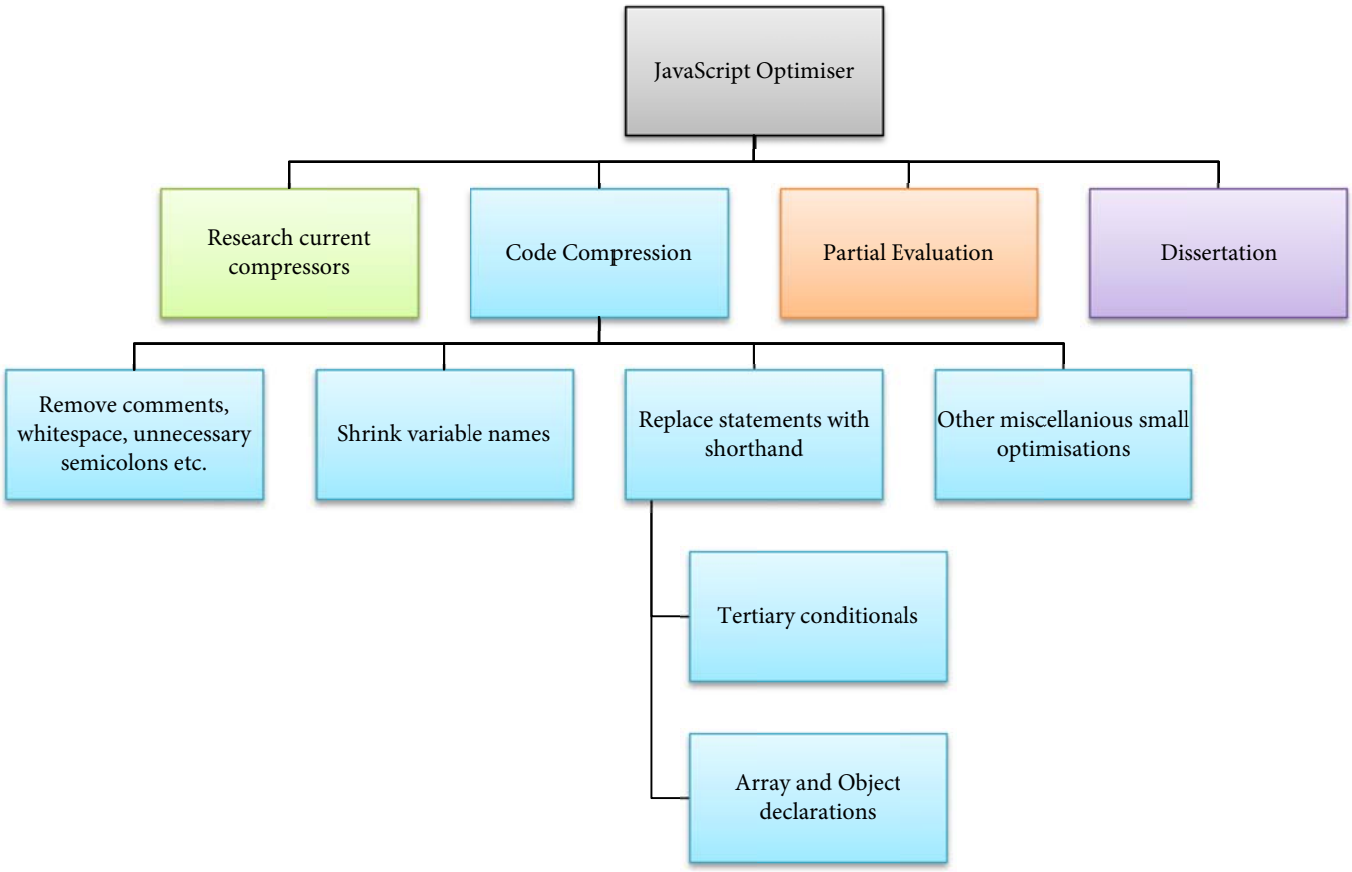
```
x*x*x;
```

Note that while techniques such as partial evaluation can improve the speed of execution, the main focus of this project is to reduce the code *size* and as such only partial evaluations which result in fewer characters will be included in the final output.

Plan

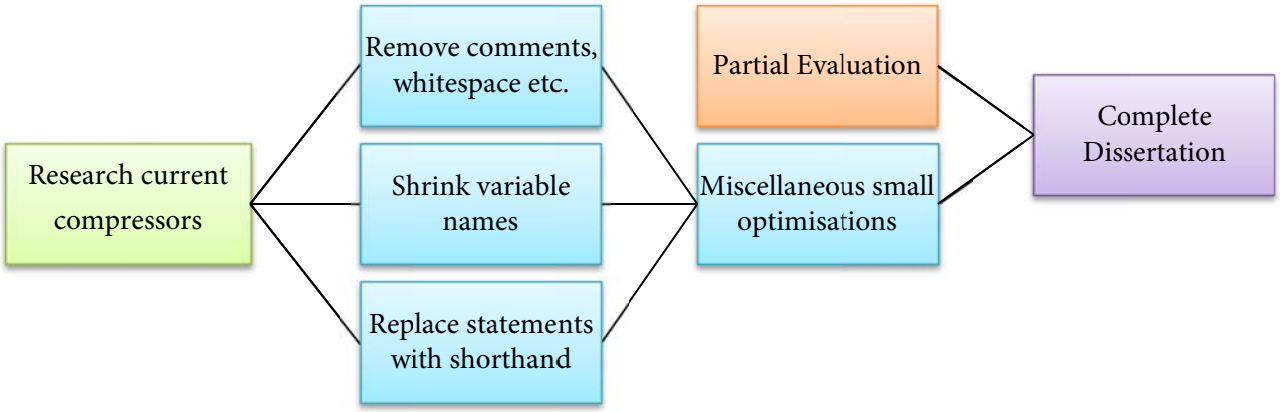
Project Breakdown

The project can be broken down into four major modules which will need to be completed separately: research, code compression, partial evaluation, and the written dissertation. The following chart outlines the most distinct modules in the project. I will use these modules as a basis for planning my time.

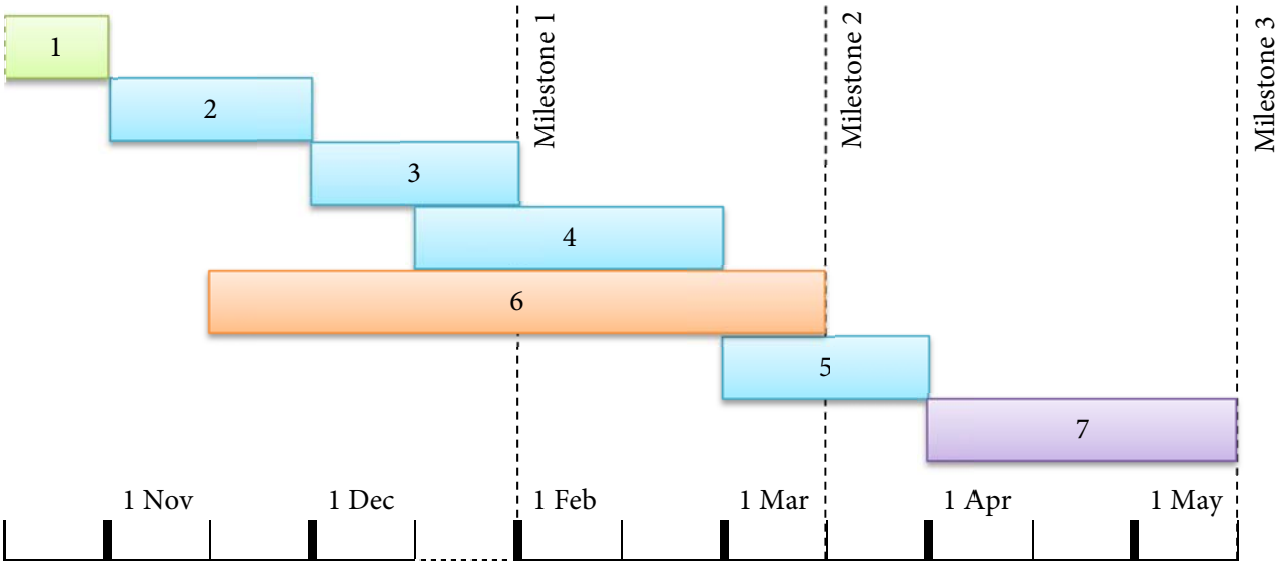


Activity Sequencing

The completion of some modules is dependent upon others having been concluded, so it is important to understand the relationships between each section of the project. The following chart shows the sequence in which each module must be developed.



Scheduling



Key

1. Research current compressors
2. Remove comments and whitespace etc.
3. Shrink variable names
4. Replace statements with shorthand
5. Miscellaneous small optimisations
6. Partial evaluation
7. Complete dissertation