1 Introduction

The aim of the project is to produce a type system for a subset of JavaScript which can be statically analysed, along with a gradual-typing framework extending this subset to include standard JavaScript programs which are too dynamic to evaluate statically. This subset should include the basic JavaScript primitives, as well as object literals and simple functions including most assignment, comparison and arithmetic operators. There is a large scope for extensions supporting more of the language. Evaluation will take the form of a type soundness proof for the system, and a quantitative analysis of the performance of the compiled JavaScript.

2 Starting Point

- confident javascript programmer
- builds on Part IB courses in Compiler Construction and Semantics
- Read some papers
- Will be using https://www.npmjs.org/package/acornAcorn to parse JavaScript and produce AST.

3 Resources

For this project I will mainly use my own machines - either a desktop (Ubuntu 14.04, 3.6GHz i7, 8GB RAM, 2x 1TB HDD) or laptop (Windows 8.1, 1.8GHz i7, 8GB RAM, 1TB HDD) as convenient. I will use Git for version control, with remotes set up on Github and on a VPS hosted by BHost for backup purposes. I require no additional special resources.

4 Work to be done

The project can be broken into the following sub-projects:

1. Definition of a formal type system along with the subset of JavaScript which is supported
   Problem - what format should the definition of supported JavaScript take, such that I can prove safety for it?

- Should have following properties:
  Progress: Given arbitrary gamma, e and T, gamma|-e:T=>e is a value or some e' exists which e can reduce to
  Preservation: Given arbitrary gamma, e, e' and T, (gamma|-e:T and e->*e') => gamma|-e':T

Decidability of checking: Given arbitrary gamma, e and T; gamma|-e:T is decidable
Type inference: Given arbitrary gamma and e; can find T such that gamma|-e:T or show no such T exists

1. Create a static type checker for the above type system (which will clearly need to include type inference)
2. Create a source-to-source compiler to implement gradual typing, allowing code which has had its type checked by the above to safely interface with unchecked code.

5 Success Criterion

6 Possible Extensions

7 Timetable