# TYPE INFERENCE IN JAVASCRIPT

CS252 Project Report

Presented to

Dr. Thomas Austin

Department of Computer Science

San José State University

By

Yogesh Kiran DIxit

May 2016

## ABSTRACT

JavaScript is a dynamically typed language. JavaScript programmers are not required to specify the specific types to the variables and expressions while writing a JavaScript program. As a result of these two things, compilers or JavaScript engines that are executing JavaScript code do not have any static type information to refer to. This makes it impossible for JavaScript engines to find and throw type errors at the programmers when there are any in the program. As a result, there are high chances of getting run-time failures or unexpected output. I seek to solve this problem by inferring types of the expressions in the given JavaScript code. Utility program will detect any possible type errors found in the given code and notify the programmer.

JavaScript type inference utility program comes in two flavors. One is Node JS flavor and another comes in a Mozilla Firefox add-on. Both the flavors use reflect js library for parsing the given JavaScript code and constructing AST out of it. This AST is further used to actually infer the types.

**TABLE OF CONTENTS**

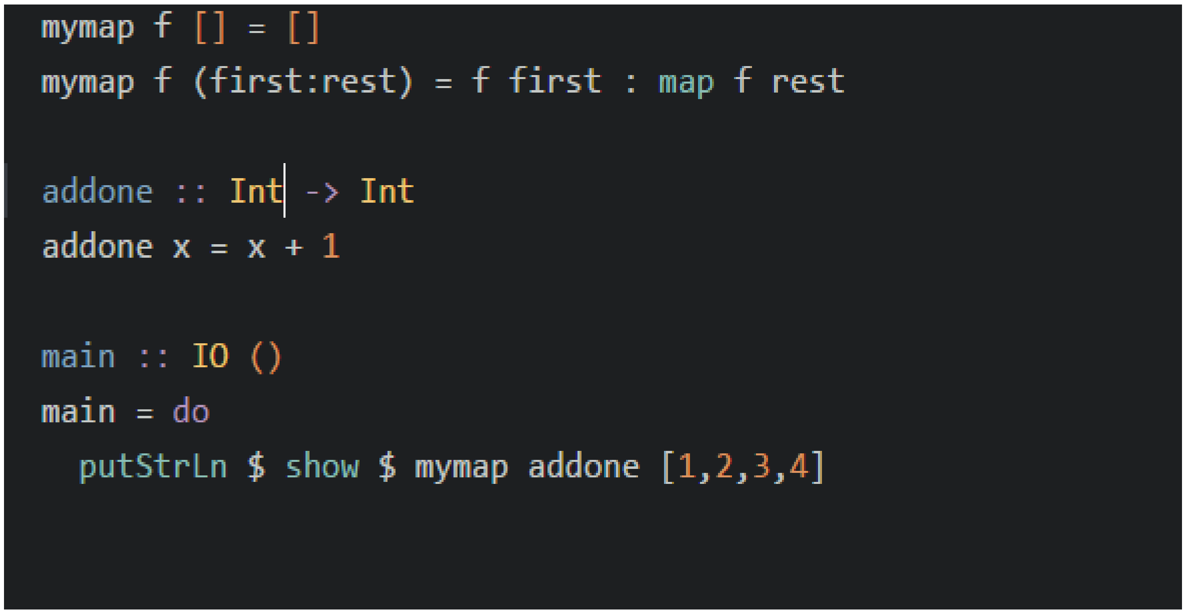
## 1. INTRODUCTION

Type Inferencing is the ability to automatically conclude the data type of an expression in a programming language. It basically checks the given program for its type correctness without making programmer declare the types explicitly. It means that you don't have to care about declaring the types everywhere in your program and yet it is type safe.

Some programming languages that come with the type inferencing feature are Haskell, OCaml, and Scala.

Given in figure 1 is a Haskell program, we will try and see how types inference works.

- mymap is a function with two arguments (therefore of the form a -> b-> c)
- Patterns [ ] and (first:rest) always match lists, so second argument is list [d] (list with members of type d)
- First argument f is applied to the “first” which must be of type d.
- Thus f is a function of type d -> e
- Finally mymap produces a list of whatever function f produces, thus [e]



```haskell
mymap f [] = []
mymap f (first:rest) = f first : map f rest

addone :: Int -> Int
addone x = x + 1

main :: IO ()
main = do
  putStrLn $ show $ mymap addone [1,2,3,4]
```

*Figure 1- Haskell Type Inference Example*

- Putting all the parts together we obtain signature of mymapwould be mymap:: (a -> b) -> [d] -> [b] .

## 2. JAVASCRIPT WITHOUT TYPE INFERENCE

JavaScript is a dynamically typed programming language [2]. This feature of the JavaScript language states that types are associated with each values int he program instead of each expression [3]. And variable assigned to one type value can be reassigned to a value of different type later in the program.

Example: var myInt = 3 + 2;

Here in the above example of a JavaScript, there is an expression statement in which binary operation (3 + 2) is assigned to myInt. Here, as said earlier "numeric" type is associated with the values 3 and 2 but no type is associated with myInt.

This behavior of JavaScript may result in large runtime overhead, increase the complexity of the compiler and runtime errors.

Consider the example JavaScript program in Figure 2.

```
1 var num = 12;
2 var str = 'hello';
3 var y = num - str;
4 console.log("y : "+ y);
5 console.log("Type of y : "+ typeof y);
```

*Figure 2: JavaScript Program*

In this example, value 12 is of type numeric and value 'hello' is of type String. But, no types are specified for variables *num, str, and y*. Here we are specifically interested in the operation " *var y = num – str* " that is performed on line number 3. Here as the types of var num and var str are not known at the static time, the operation will result in a value NaN at runtime.

Instead, if we had inferred types of "*var num*" and "*var str*" before program execution could have detected a type error as subtract operation can be performed only on values with numeric types.

This creates a need for a system or some utility program which will infer types for a JavaScript program and notify about type errors if present.

## 3. IMPLEMENTING TYPE INFERENCE SYSTEM

I have implemented JavaScript type inference system, which infers types for the expressions from the given JavaScript code. My implementation of type inference system first parses the given JavaScript code to construct abstract syntax tree (AST). I am using reflect a JavaScript library's parse API to construct the AST. Reflect JS internally uses Parser API's provided by Mozilla. Once the abstract syntax tree of the given program is constructed, the utility program walks through this AST to infer the types. Type Inference is done only for newly declared variables, statement variables, arrays and method return types [2].

```
var y;
var myInt = [10, 11];
var myString = 'hello';
function test()
{
  var x = 12;
  return x;
}
var my = 3 + test
```

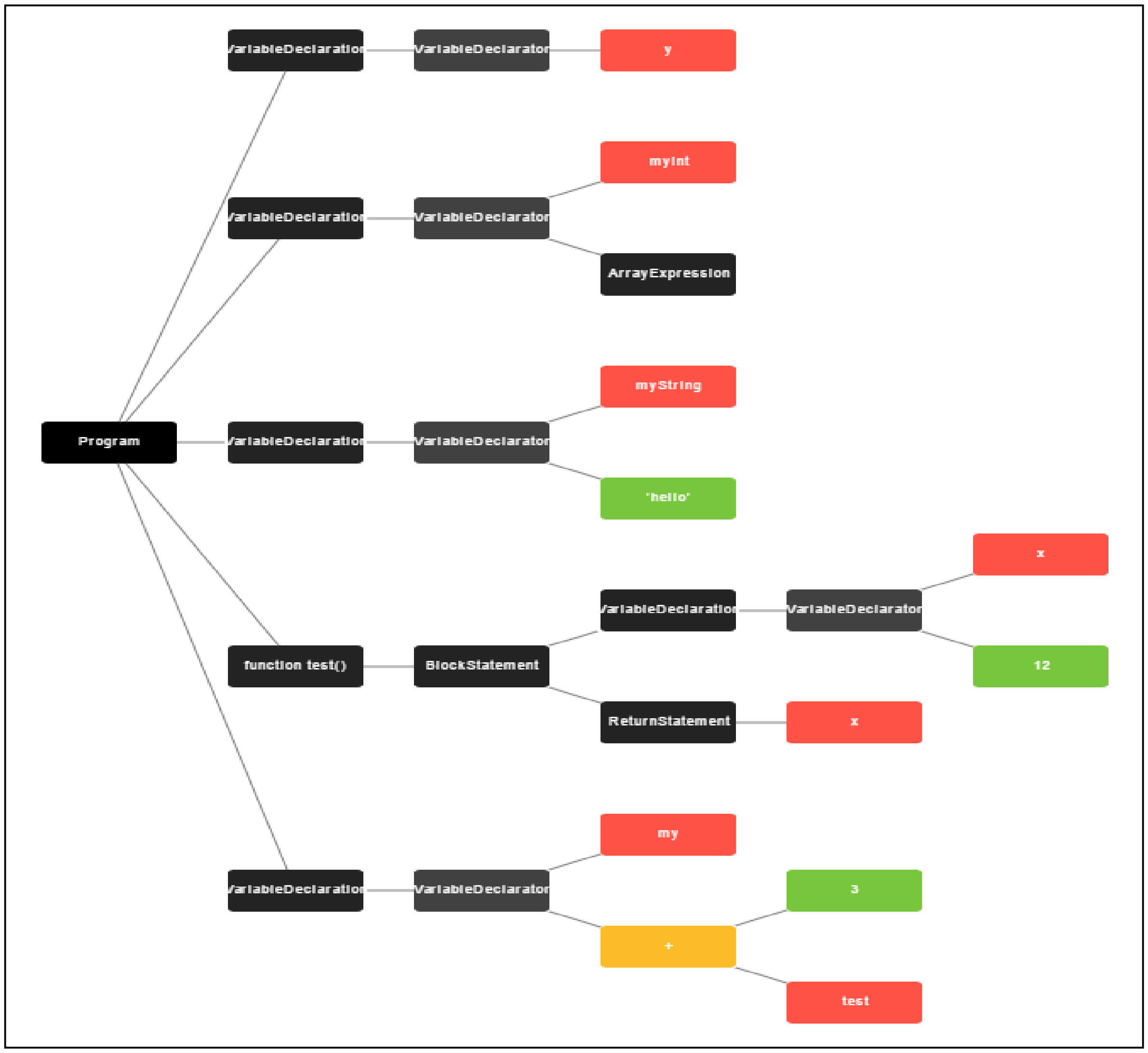


*Figure 3JavaScript Program and constructed AST*

- <u>Variables</u>:

When the utility program finds a var declaration, the type for the new variable will be inferred on the basis of the type of the expression on the right.

var myString = "hello";

(Inferred type for myString based on its value "hello" would be String)

Type of the variable would be inferred only for its first occurrence (declaration). Whenever the variable is reassigned to a new value later utility program will check if the reassignment is valid based on its previously inferred type.

myString = 3;

(This will give a type mismatch error as inferred type for myString was String and now we are trying to assign a numeric value to it)

- Arrays:

For arrays, at the time of declaration if all the values in the array belong to some specific type then the inferred type for the array would be same as that of elements it contains.

If the array contains different type values then I assign "DefaultArrayType" as type value.

var myNumArray = [2, 3, 4] // myNumArray is of type [numeric]

var myStringArray = ["hello", "my", "name"] // myStringArray is of type [String]

var myMixArray = [2, "CS"] // myMixArray is of type DefaultArrayType

- Statement Variables:

For statement variables, right part of the expression would be evaluated and its type would be inferred and then it would be assigned to the variable on the left side.

```
var x = 3; // Inferred type for x is numeric

var y = 4; // Inferred type for y is numeric

var xy = x + y; // Inferred type for xy would be numeric as (x+y) is a valid operation and produce numeric value.
```

- <u>Method Return Type</u>:

To infer the return type of the functions, type of the return value would be considered. If a function returns a numeric value then the return type of that function would be numeric.

```
function test()

{

   var x = 12; // inferred type for x is numeric

   return x;

}

// based on the type of the return value of a method, inferred type of function test would be numeric.

var myTest = test + 3 // as return type of function test is numeric, this becomes a valid operation and inferred type for myTest would be numeric.
```

## 4. NODE JS FLAVOR AND FIREFOX ADD-ON

I implemented the type inference system that is explained in the above section using Node JS. Main challenge while implementing the system in Node JS was to walk through the AST and categorize the type of the each statement into ExpressionStatement, DeclarationStatement or FunctionDeclaration.

When the type of the statement is either DeclarationStatement or FunctionDeclaration then the type of the expression involved in that statement would be inferred.

If the statement is of type Expression then the system will simply check whether types of left and right part of the expression match. And if they fail to match then the utility program will throw the appropriate error message.

In the Firefox add-on flavor of the system, almost all the code from Node JS flavor is reused, only difference is the reflect js library that is used. In the case of Firefox add-on as it runs on client browser an independent version of the library was included in the add-on code. Main challenge involved in creating add-on was to learn how to create a Firefox add-on and merge the existing Node JS functionality with the add-on code.

## 5. SAMPLE INPUT AND OUPUT

```
var y;
var myInt = [10, 11];
var myString = 'hello';
function test()
{
  var x = 12;
  return x;
}
var my = 3 + test;
```

```
Inferred type for myInt is
[Numeric]
Inferred type for myString is
String
Inferred type for x is Numeric
Inferred type for test is Numeric
Inferred type for my is Numeric
No errors found in the given JS
program.... you are type safe now
```

*Figure 4: Input and output (type safe program)*

```
var num = 12;

var str = 'hello';

var y = num - str;
```

```
Inferred type for num is Numeric
Inferred type for str is String
Type error...
 ***** Type error at line 3 and
column 6 *****
```

*Figure 5: input and output (type error)*

```
var x = 3;

var y = "hi there";

y = 12 + x;
```

```
Inferred type for x is Numeric
Inferred type for y is String
Cannot assign Numeric to String
 ***** Type error at line 3 and
column 2 *****
```

*Figure 6: input and output (type incorrect reassignment)*

## 6. CONCLUSION & FUTURE WORK

I have described working and implementation of the type inference system for JavaScript in above sections. The output of the JavaScript code written and tested with the type inference system would be more efficient and reliable with respect to type safety as compared to normal JavaScript code. Also, it will remove the burden from the programmer’s shoulder of thinking about the type safety of the code.

As a part of future implementation work, I will be looking forward to extending the type inference system to cover more complex constructs of JavaScript code.

## REFERENCES

[1] C. Anderson, P. Giannini and S. Drossopoulou. "Towards type inference for JavaScript," in *ECOOP 2005-Object-Oriented Programming*Anonymous 2005, Available: http://pubs.doc.ic.ac.uk/typeinferenceforjavascript-ecoop/typeinferenceforjavascript-ecoop.pdf.

[2] B. Hackett and S. Guo. Fast and precise hybrid type inference for JavaScript. Presented at ACM SIGPLAN Notices. 2012.

[3] D. Vardoulakis and O. Shivers. "CFA2: A context-free approach to control-flow analysis," in *Programming Languages and Systems*Anonymous 2010.