Our project repository is located at: https://github.com/judebattista/algorithmW

While our initial plan was to develop a Visual Studio Code extension to provide linting support to JavaScript development by using the Hindley-Milner (HM) system for type inference, we quickly realized this plan flew directly past ambitious and careened towards hubris. JavaScript's amorphous syntax is a daunting task for any parser, and the resulting Abstract Syntax Tree (AST) for even a moderately complex program tends towards the sequoian. In order to preserve the shredded remnants of our collective sanity, we opted to work with a subset of JavaScript. A much smaller subset of JavaScript. Since the majority of HM implementations are done in functional programming languages, we decided to focus on comparable JavaScript artifacts. Our algorithm handles the classic Let, Lambda, Application, and Identification. Because we are our own people and not sheep, dammit, we used terms that better describe JavaScript's environment: Let, FunctionDefinition, FunctionCall, and Identifier.

While, as a team, we consumed a half dozen papers on the HM type inference system, we found that ultimately we needed to look at previous implementations of Algorithm W. In particular, Robert Smallshire's implementation in Python was helpful, being one of the few other implementations in a non-functional (dysfunctional?) programming language. Instead of writing our own JavaScript scanner and parser, we chose to start with Esprima, a well-known "parsing infrastructure for multipurpose analysis". Unfortunately for us, its multitude of purposes do not include customized output for easy assimilation into a type inference algorithm, which necessitated extensive transmission code between the two components, and in the end, the transmission turned out to be manual. Due to our decision to limit the scope of the JavaScript expressions, the extension is designed to work with simple functions using primitive types. In its current incarnation, the extensions allows the user to pick a JavaScript file to inspect, and then translates that file into an Abstract Syntax Tree, which is output to the command line. This AST should be in a suitable form for analysis by Algorithm W.

In order to use the extension through Visual Studio Code:

1. Clone the repository from https://github.com/judebattista/algorithmW
2. Navigate to the vscExtension folder inside the repository
3. Run ‘npm install’
4. Open the folder in VScode
5. Press F5, which should open up a new VSC window.
6. Go File -> Open File (or ctrl-O) to open your source code file.
7. Press Cmd+Shift+P to open the Command Palette
8. Enter "Check Types" or select the Check Types task from the drop down
9. See output on the debug log of the original screen

A sample javascript file, hello.js ready for parsing is included in the vscExtension folder to help you get started!

Since the VSC extension is largely a proof of concept at this time, should you crave more complex type inference, the brave(?) user may also create abstract syntax trees in the algorithmW.js script using the following nodes:

FunctionDeclaration (name, function body)
FunctionCall (function, argument list)
Let (name, definition, body)
Identifier (name)

The first argument to AlgorithmW is the AST you wish to parse. Please remember that since each node is a class, you must use the new operator to instantiate each node. The second argument is a dictionary containing known type data. If you wish to parse in a clean environment, simply use JavaScript’s empty dictionary: {}.

In order to parse and view an expression, expr, simply run

AlgorithmW(expr, {}).toString().

Happy Inferring!