Hackathon #7
Javascript AST manipulation

February 27, 2018

Motivation

- UI development is becoming increasingly verbose and tedious
- Compilers are designed to take high level concepts and output lower level code
- Why not use a compiler to eliminate the tedious parts?

Objective

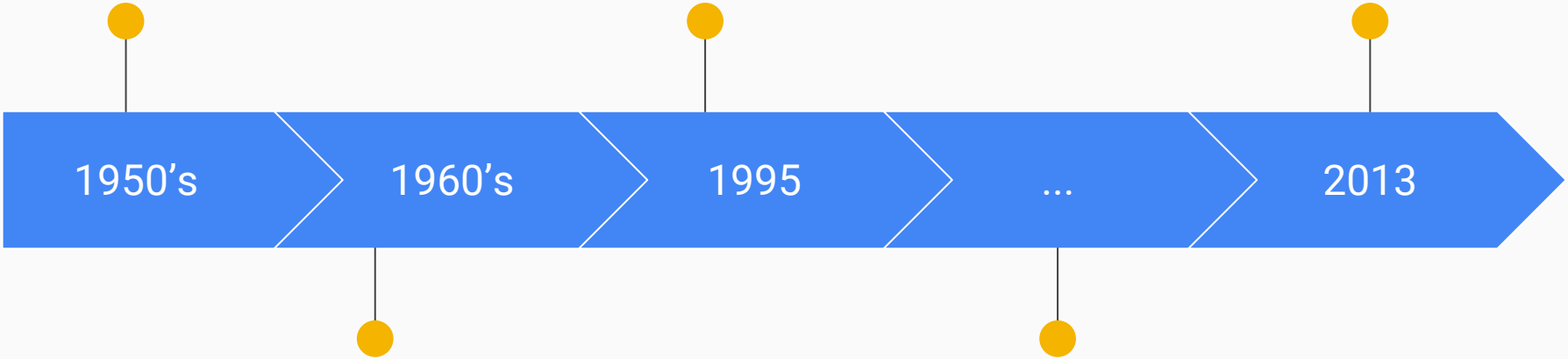
- Identify and define high level AST node abstractions used in UIs.
- Inject these new concepts into the AST.
- Transform the AST into something that can be compiled to normal JS.
- Create a UI to illustrate these concepts in action.

Example use case: Setting a variable

Assembly language invented

Javascript invented

React released



Higher level compilers

Lots of technical progress???

Then

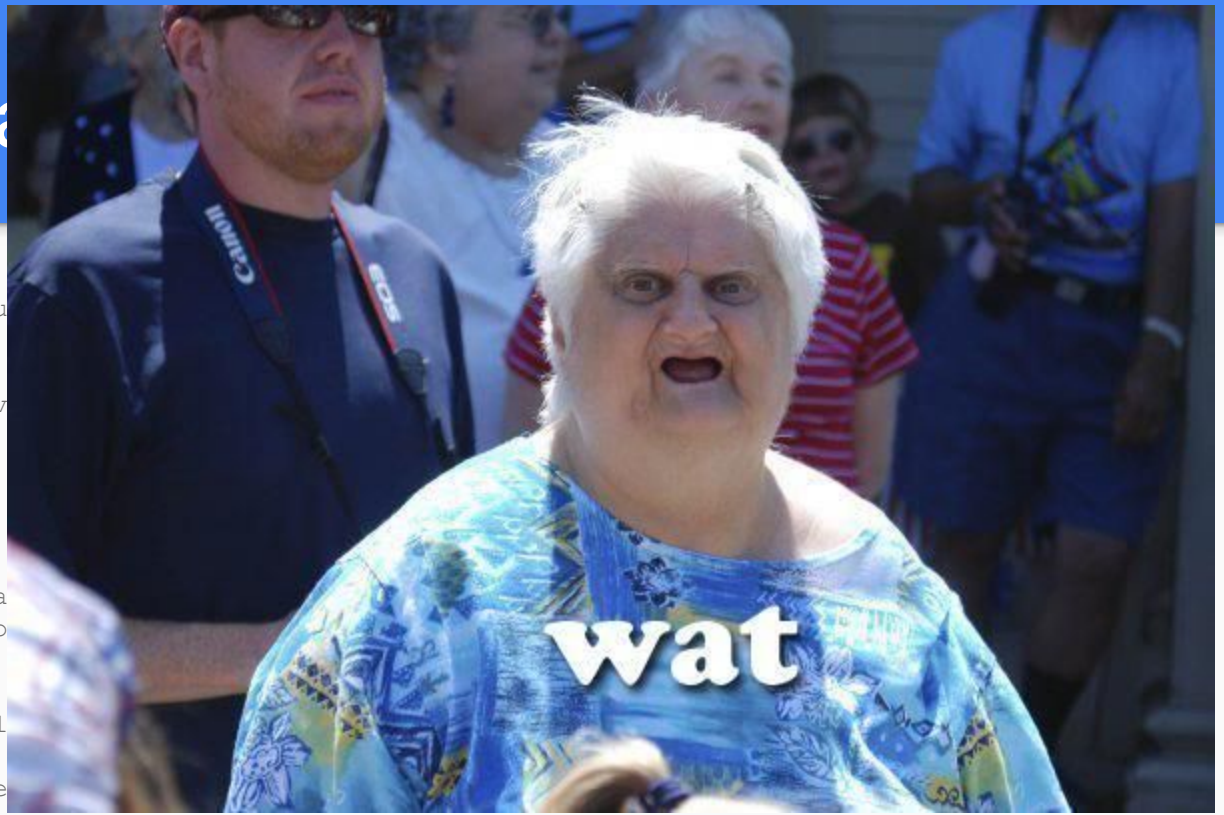
```
mov ax, 0x80
```

Now (abbreviated version)

```
const valueReducer = (state, action) => {
  ...
  return {
    ...state, value: action.payload
  }
}

const mapState = state => ({ value: state.value })

@connect(mapState)
class MyReactComponent extends React.Component {
  handleChange = value => dispatch => dispatch({ type: 'SET_VALUE', payload: value })
  render () {
    const { value } this.props
    ...
    <input type="text" value={value} onChange={this.handleChange} />
  }
}
```

Now (a

```
const valueRedu
  ...
  return {
    ...state, v
  }
}

const mapState

@connect(mapSta
class MyReactCo
  handleChange
  render () {
    const { val
    ...
    <input type
  }
}
```



How can we fix this?

Use a compiler to introduce higher level primitives and remove boilerplate.

What does a compiler do?

1. Lex/Parse: Takes a string of characters (source code) and converts it into an abstract syntax tree (AST).
2. Semantic Analysis: Perform type checking, optimizations, dead code elimination, etc as a series of stages that transform AST -> modified AST.
3. Compile AST into target code (Javascript)

Make “connect”ing easier

1. Parse the JS code to an AST
2. Traverse the AST looking for instances of “`connect-to`”
3. Modify the AST to add plumbing for the “`connect`”
4. Compile the AST back to Javascript

Demo

Ideas for future work

- Provide an easier way to define AST transformation rules
- Manipulate the AST directly instead of manipulating a stream of characters that eventually get translated into the AST
- UI tools that generate AST subtrees to increase productivity