Lenguajes y Sistemas Informáticos para la resolución de problemas complejos

Campus América 2019

Procesadores de Lenguajes

Casiano Rodríguez León

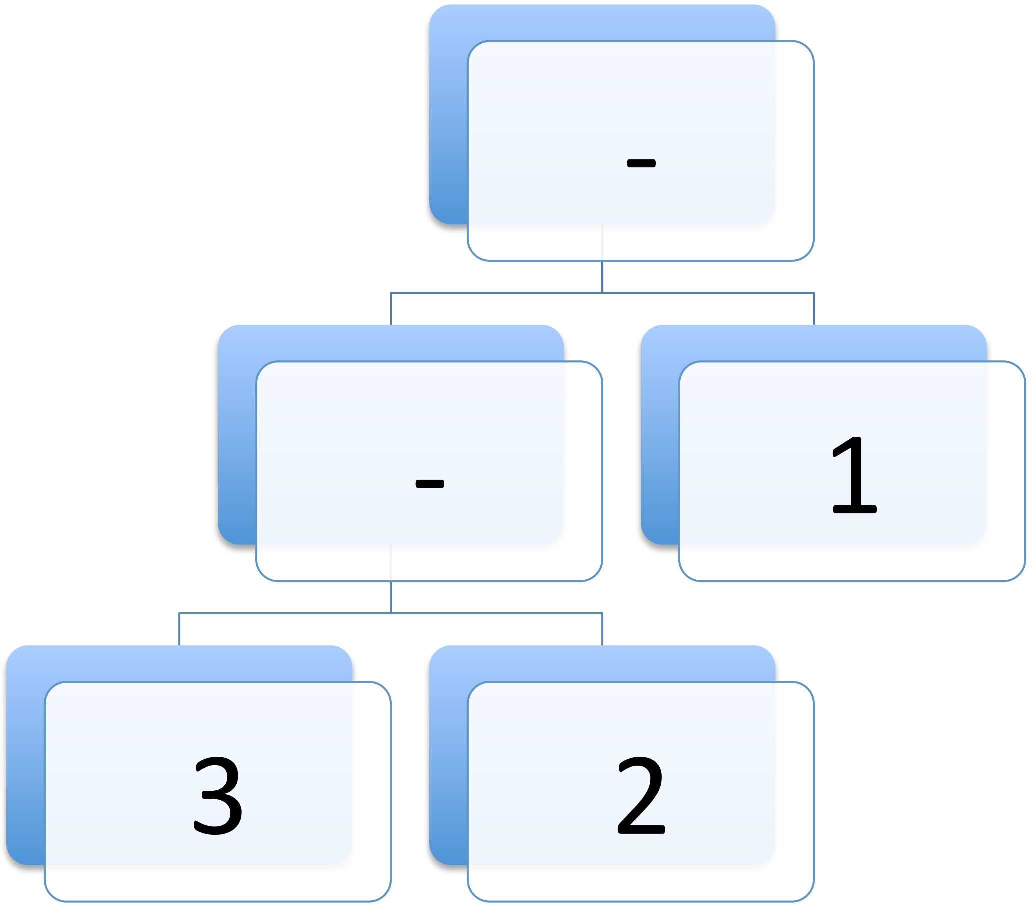
Índice

- Análisis Sintáctico y Árboles Sintácticos (AST)
- Semántica y Ambigüedad
- Esquemas de Traducción
- Análisis Bottom-Up (LR)
- Asociatividad y Prioridad
- Resolución Dinámica de Conflictos
- Recorrido del AST y Generación de Código
- Optimización de Código

3 - 2 - 1

Árbol Sintáctico Abstracto

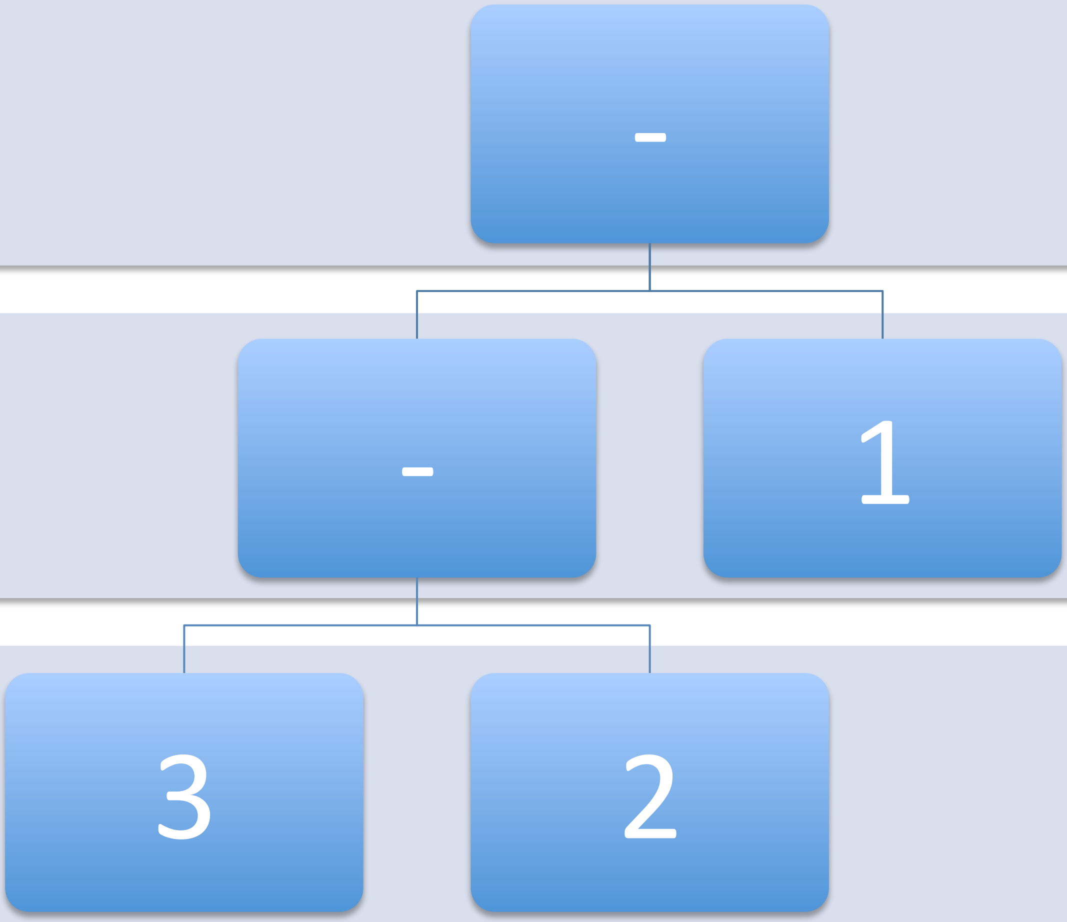
(3-2)-1



Semántica 3 - 2 - 1

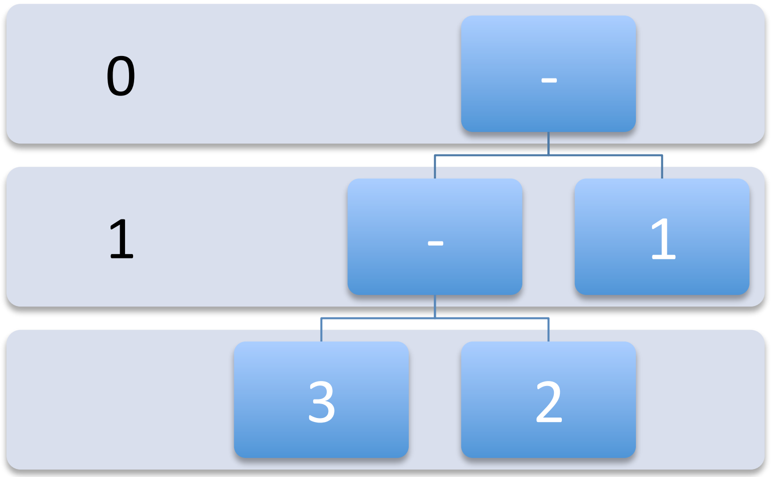
0 = 1 - 1

1 = 3 -2

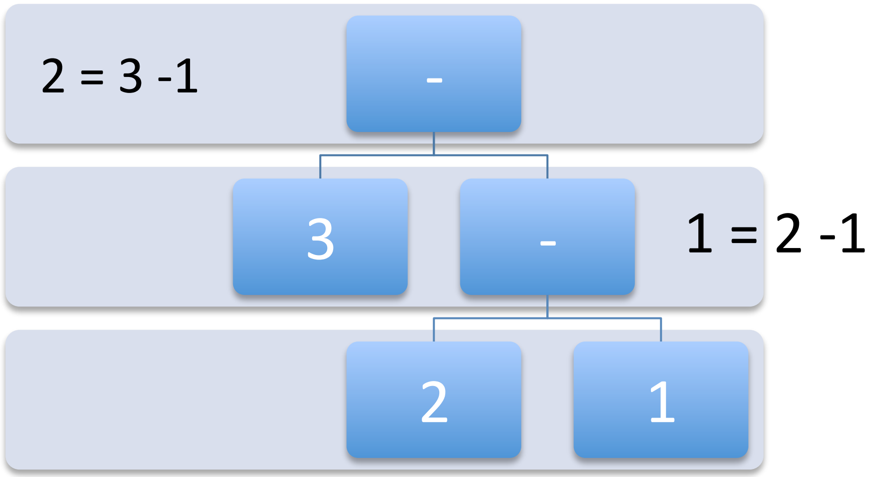


Semántica y Ambigüedad

(3-2)-1



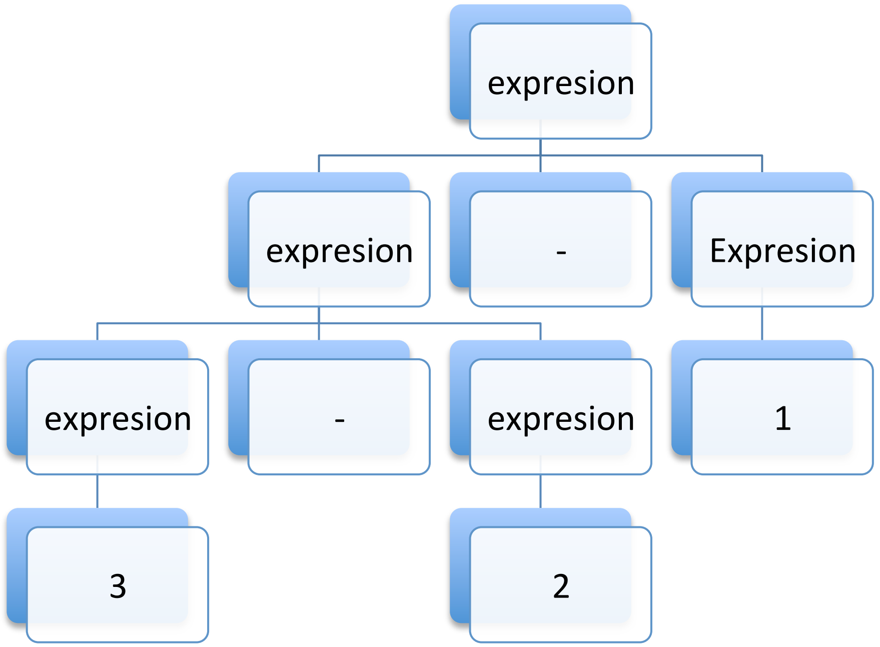
3-(2-1)



Gramática Independiente del Contexto

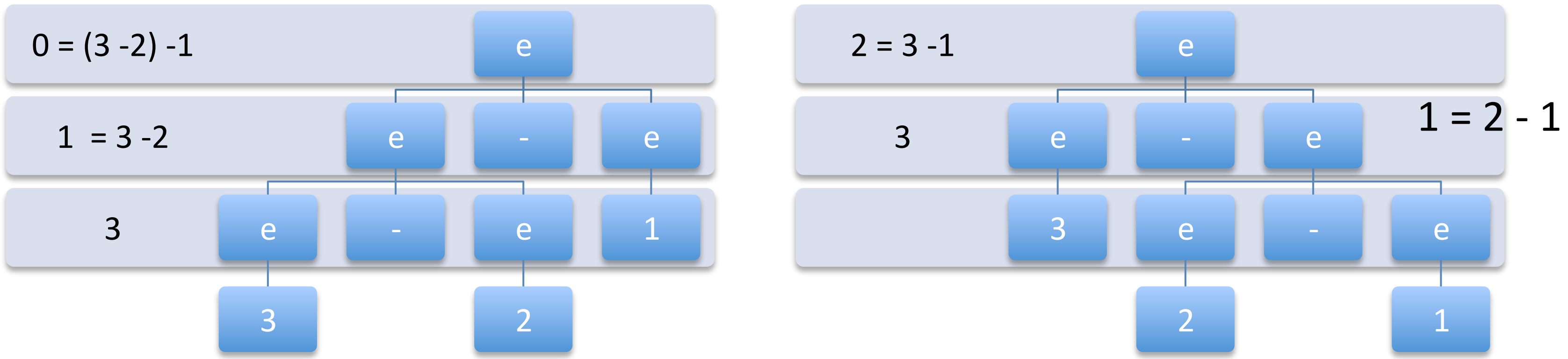
- expresion -> expresion ‘-’ expresion
- expresion -> NUMERO

(3-2)-1



Gramática Ambigua

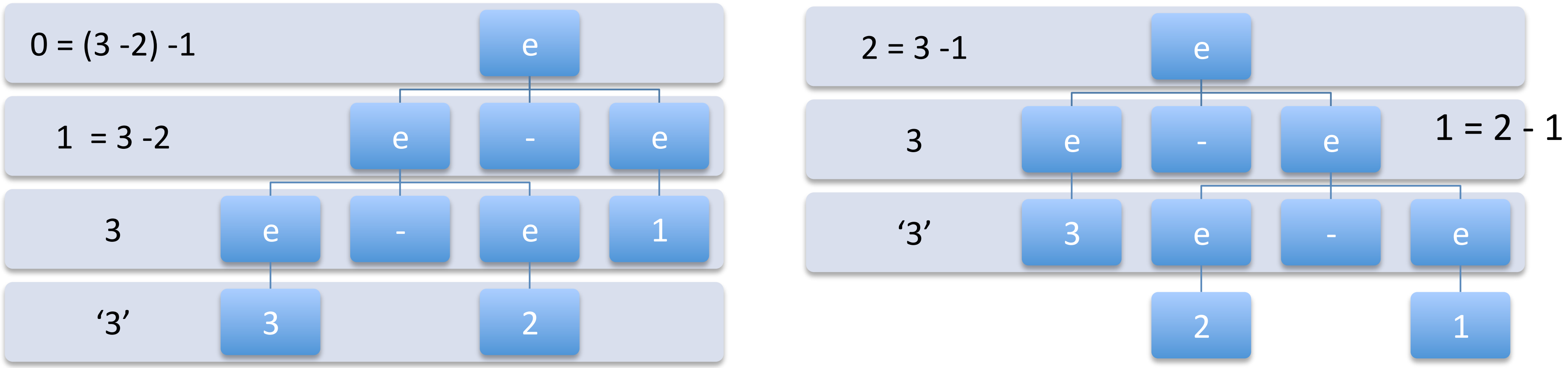
- expresion -> expresion '-' expresion
- expresion -> NUMERO



Esquema de Traducción (yacc)

```
e   -> e '-' e    { $$ =  $1 - $3;  }
e ->  NUM     { $$ = Number($1);   }
```

3 – 2 - 1



Parsing: Construcción del Árbol

e -> e '-' e { $$ = $1 - $3; }

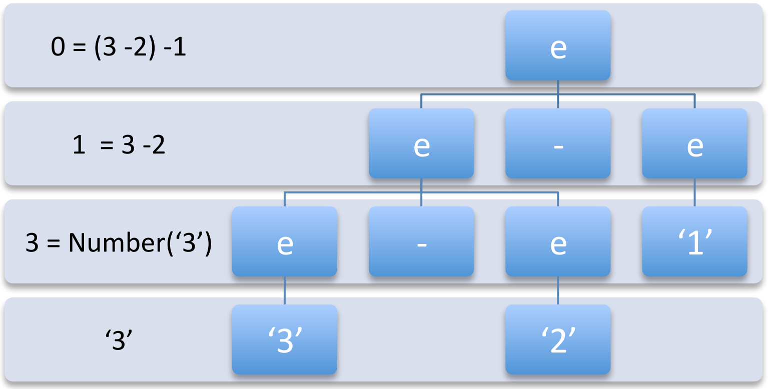
e -> NUM { $$ = Number($1); }

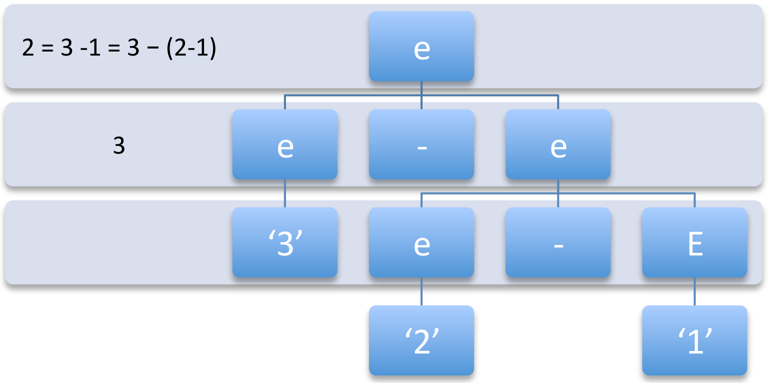
Análisis Sintáctico Ascendente:

.3 – 2 – 1 <= e. – 2 – 1 <= = e –. 2 – 1 <= = e – 2. – 1 <= e – e. – 1

¿Qué hacer?

1. <= e. – 1 <= e - . 1 <= e – 1. <= e – e. <= e.
2. <= e – e - . 1 <= e – e – 1. <= e – e – e. <= e – e. <= e.





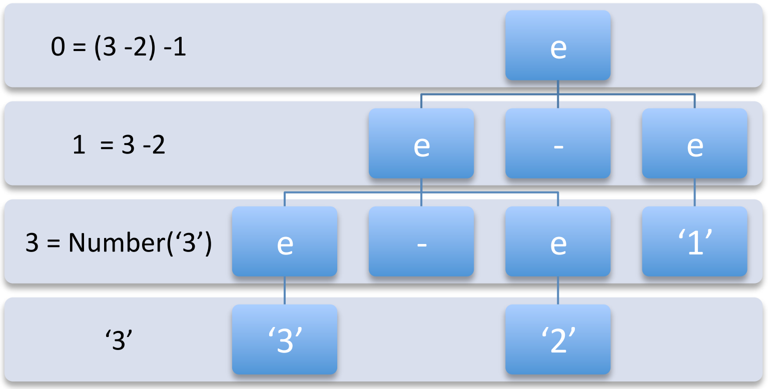
Conflicto Shift/Reduce

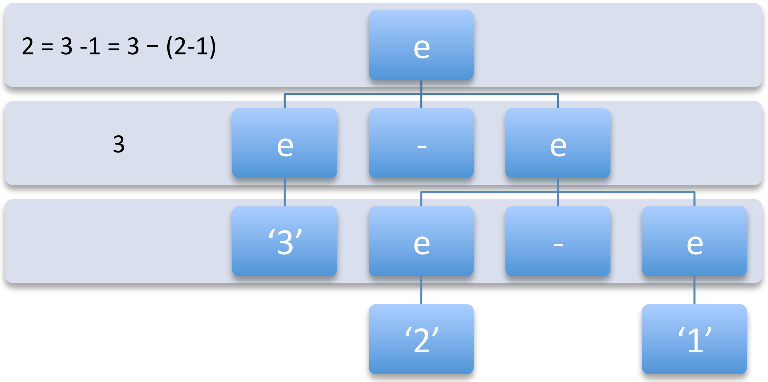
.3 – 2 – 1 <= e. – 2 – 1 <= = e –. 2 – 1 <= = e – 2. – 1 <= e – e. – 1

¿Qué hacer?

1. <= e. – 1 <= e - . 1 <= e – 1. <= e – e. <= e.
2. <= e – e - . 1 <= e – e – 1. <= e – e – e. <= e – e. <= e.

El conflicto puede verse como una lucha entre la regla e -> e '-' e y el terminal/token '-'





Un programa Yacc

```
%left  '-'
```

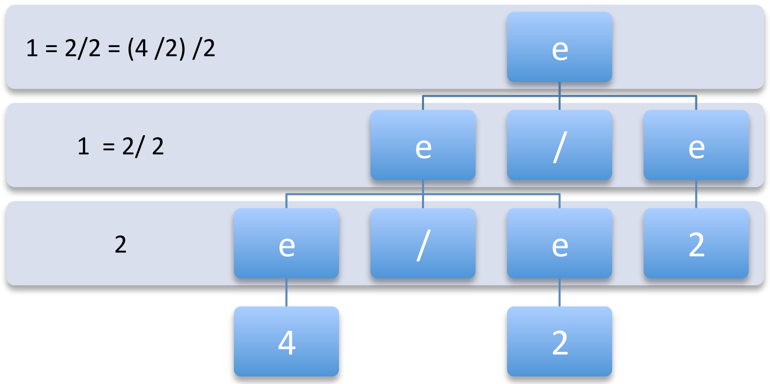
$\leftarrow$ **En la lucha entre la regla e -> e '-' e y el terminal/token '-' debe "ganar" la regla**

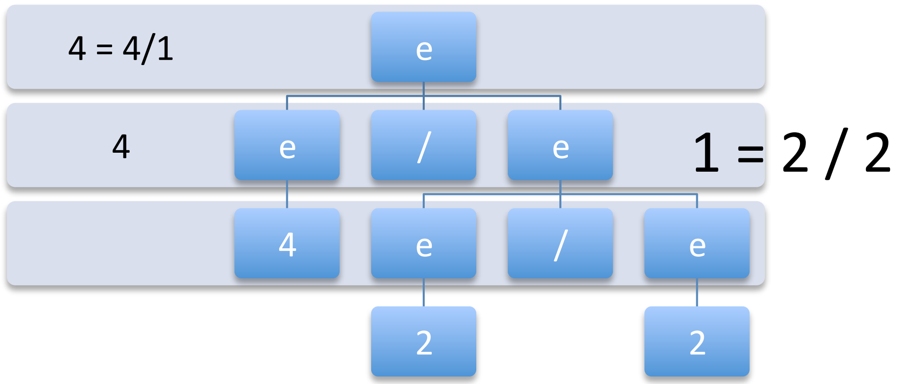
```
%%
s  :  e            { return $1; }
   ;

e  : e '-' e { $$ = $1 - $3; }
   | NUM  { $$ = Number($1);  }
   ;
```

Ambigüedad: Asociatividad
4/2/2

(4/2)/2

4/((2/2)

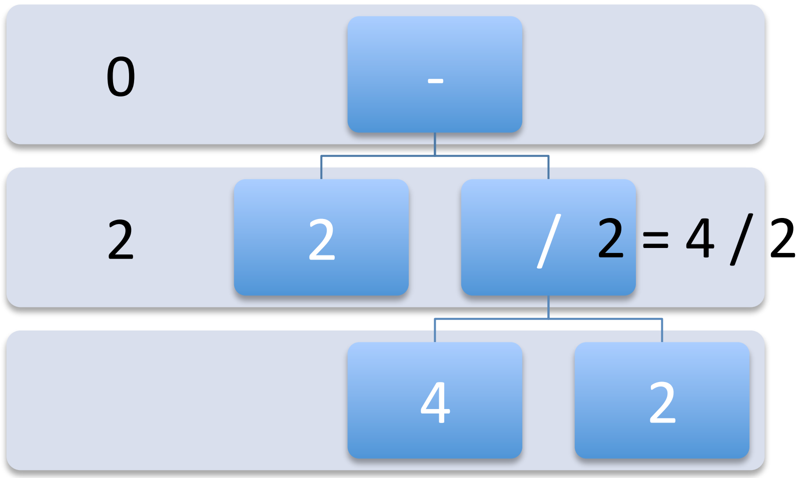




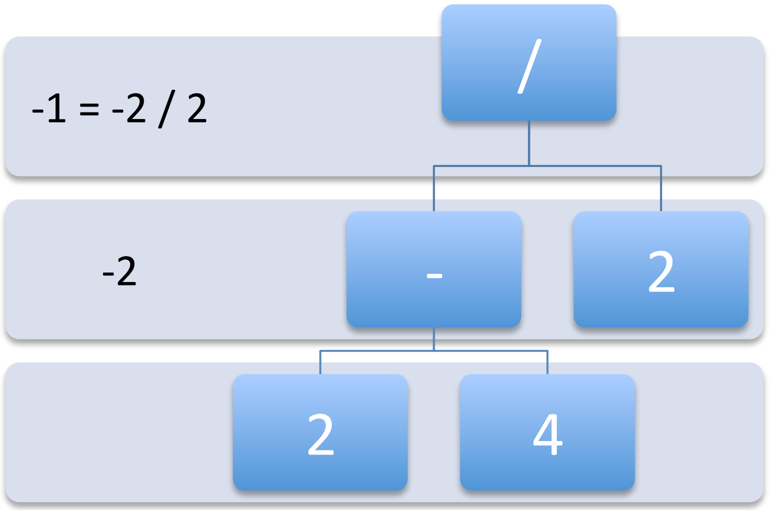
Ambigüedad: Prioridad

```
e  : e '-' e { $$ = $1 - $3; }
   | e '/' e { $$ = $1 / $3; }
   | NUM  { $$ = Number($1);  }
   ;
```

2-(4/2)



(2-4)/2



Ambigüedad: Prioridad 2-4/2

2.-4/2 <= e.-4/2<=e-.4/2<=e-4./2<=e-e./2

¿Qué hacer?

1. <= e./2 <= e/. 2 <= e/e.<= e.
2. <= e-e/.2 <= e–e/2.<= e–e/e.<= e–e. <= e.

El conflicto es entre la regla e -> e '-' e y el terminal '/'

Ambigüedad: Prioridad

Mas prioridad

```
%left '-'
%left '/'
%%
e  : e '– ' e { $$ = $1 - $3; }
   | e '/ ' e { $$ = $1 / $3; }
   | NUM  { $$ = Number($1);  }
   ;
```

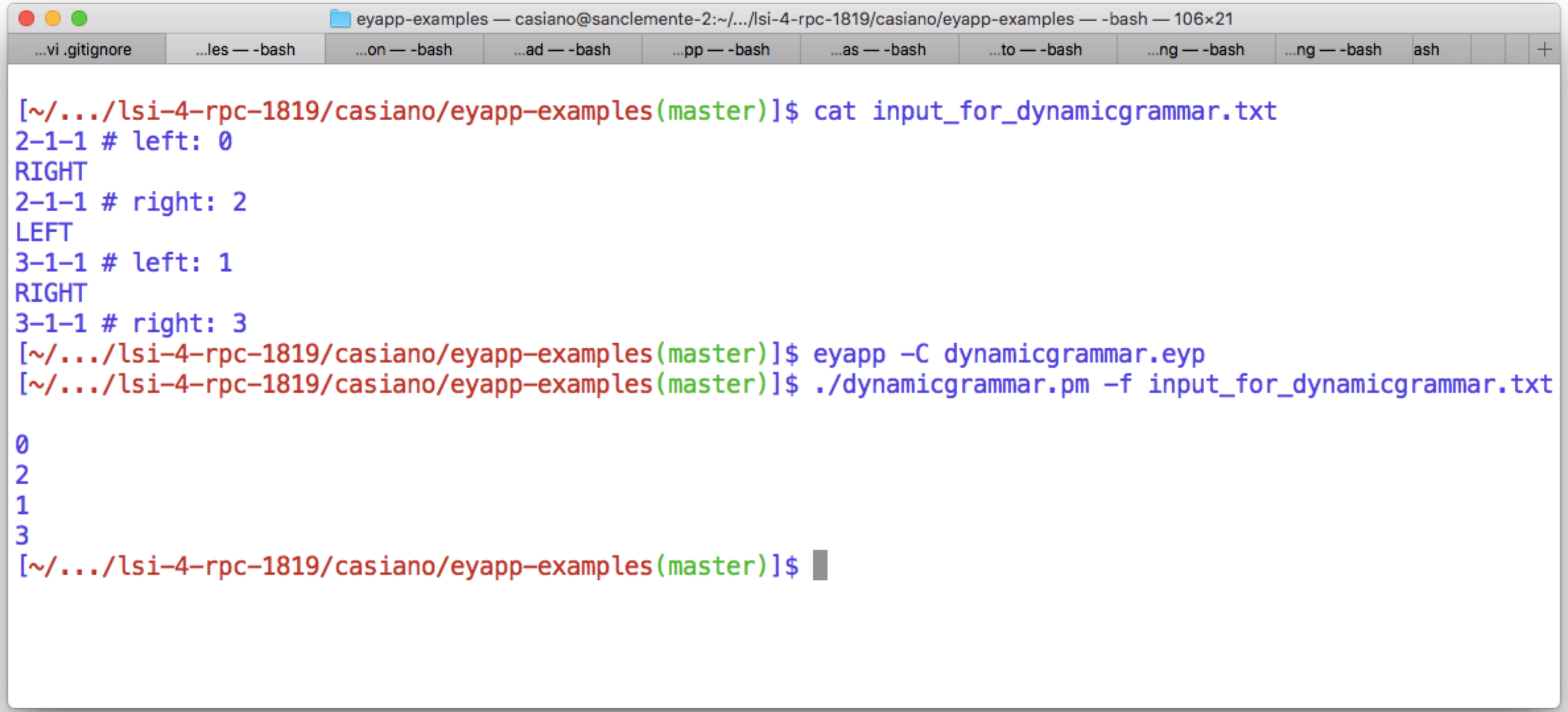
En la lucha entre reducir por la regla e -> e '-' e y desplazar el terminal '/' debe "ganar" el token

Dynamic Resolution of Shift-Reduce Conflicts

Write a language that accepts lists of two kind of commands: *arithmetic expressions* like *4-2-1* or one of two *commands*: *left* or *right*.

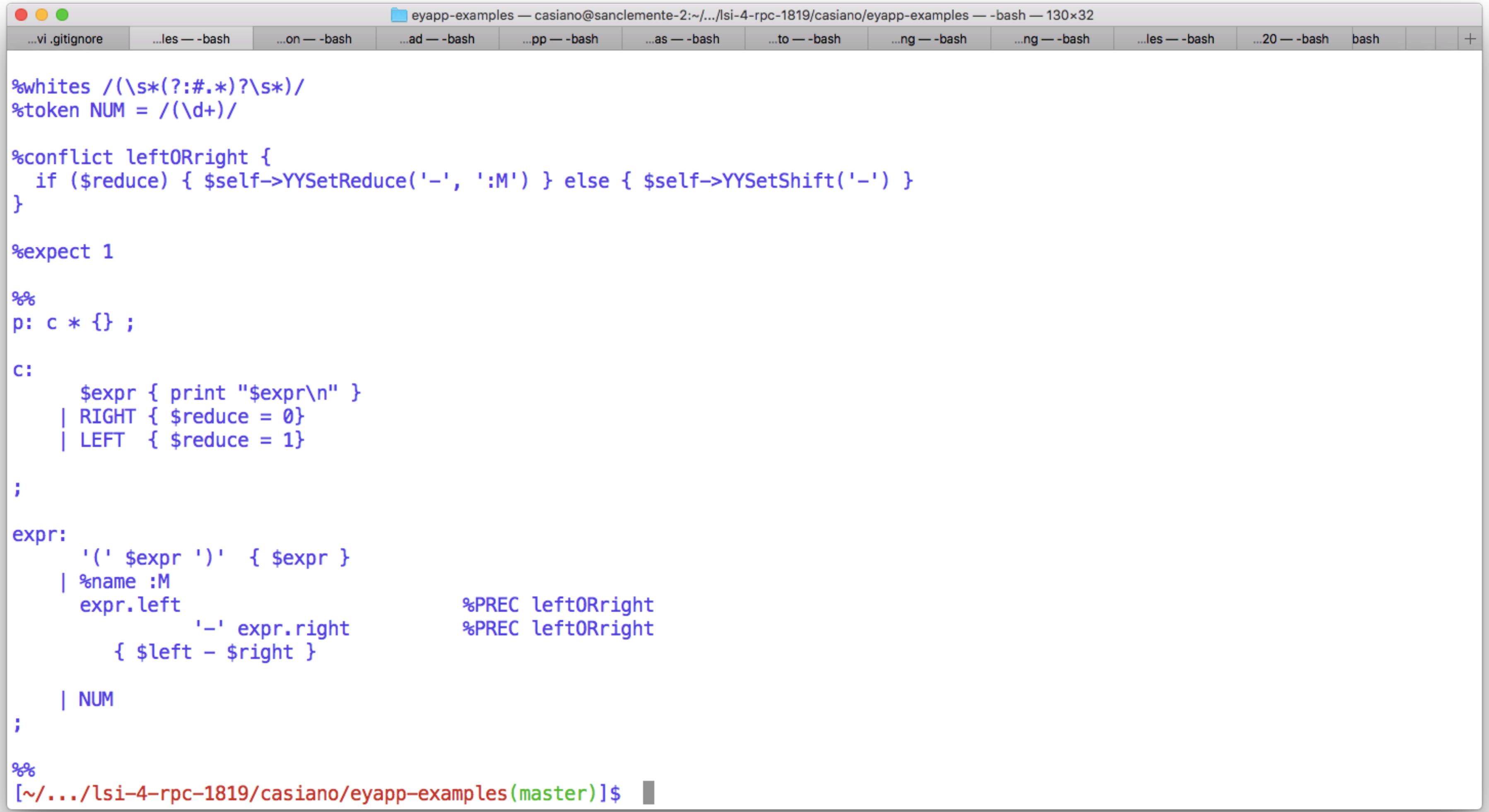
- When a *right* command is issued, the semantic of the *'-'* operator is changed to be right associative.
- When a *left* command is issued the semantic for *'-'* returns to left associative interpretation.

Dynamic Resolution of Shift-Reduce Conflicts



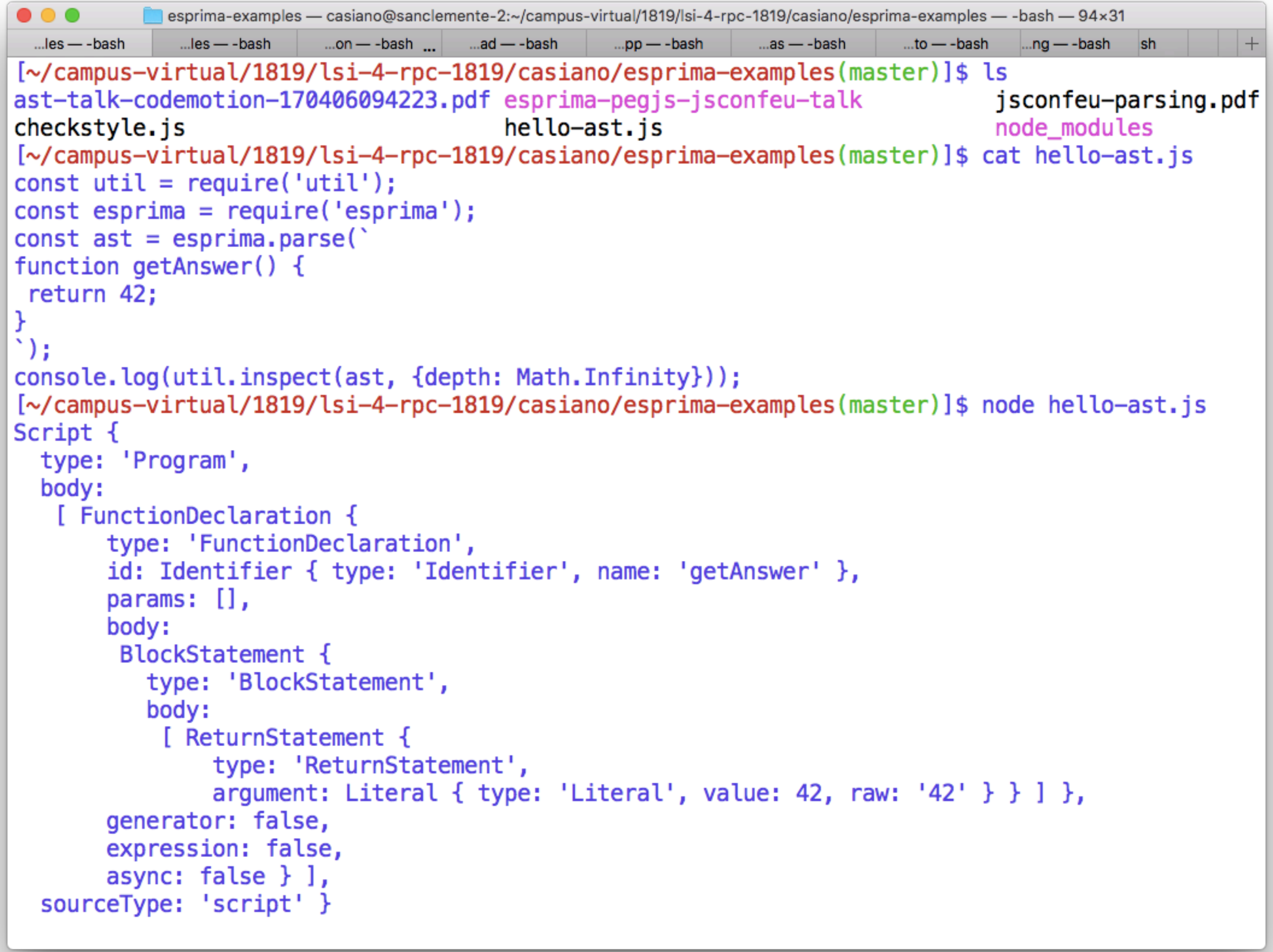
```
[~/.../lsi-4-rpc-1819/casiano/eyapp-examples(master)]$ cat input_for_dynamicgrammar.txt
2-1-1 # left: 0
RIGHT
2-1-1 # right: 2
LEFT
3-1-1 # left: 1
RIGHT
3-1-1 # right: 3
[~/.../lsi-4-rpc-1819/casiano/eyapp-examples(master)]$ eyapp -C dynamicgrammar.eyp
[~/.../lsi-4-rpc-1819/casiano/eyapp-examples(master)]$ ./dynamicgrammar.pm -f input_for_dynamicgrammar.txt

0
2
1
3
[~/.../lsi-4-rpc-1819/casiano/eyapp-examples(master)]$ 
```

Dynamic Resolution of Shift-Reduce Conflicts



```
%whites /(\s*(?:#.*)?\s*)/
%token NUM = /(\d+)/

%conflict leftORright {
  if ($reduce) { $self->YYSetReduce('-', ':M') } else { $self->YYSetShift('-') }
}

%expect 1

%%
p: c * {} ;

c:
      $expr { print "$expr\n" }
    | RIGHT { $reduce = 0}
    | LEFT  { $reduce = 1}

;

expr:
      '(' $expr ')'  { $expr }
    | %name :M
      expr.left                       %PREC leftORright
                '-' expr.right        %PREC leftORright
         { $left - $right }

    | NUM
;

%%
[~/.../lsi-4-rpc-1819/casiano/eyapp-examples(master)]$ 
```

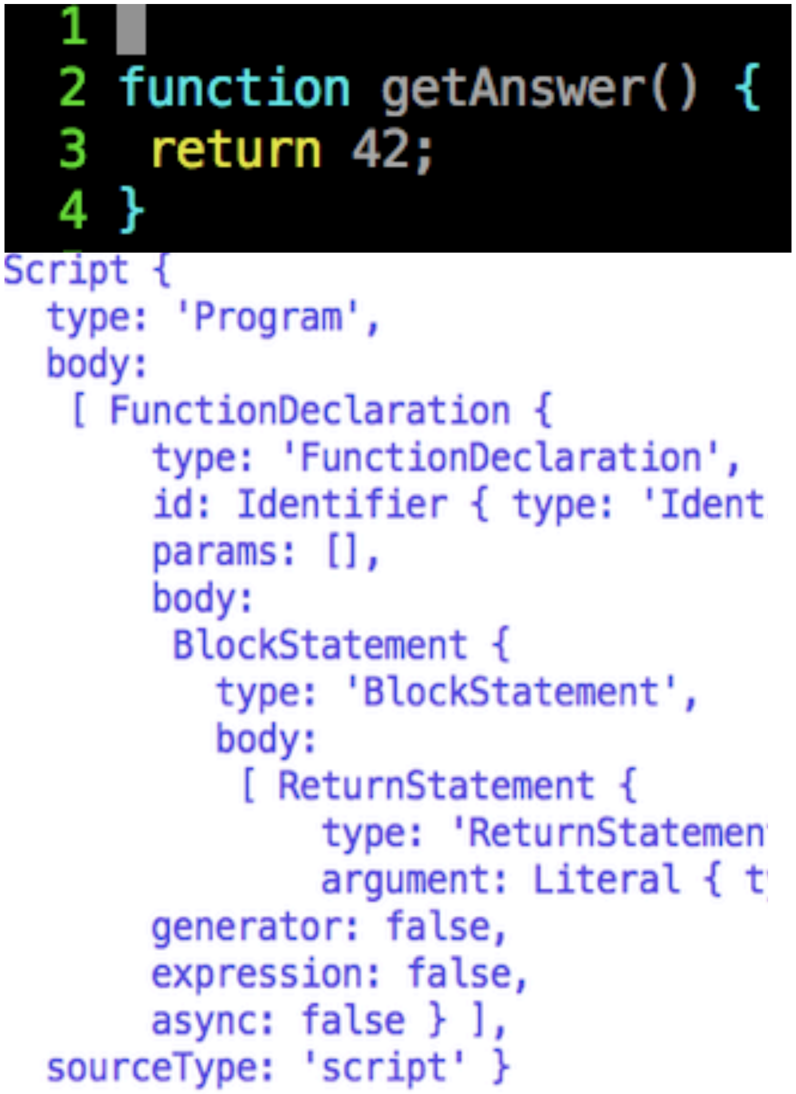
Parsing, Traversing and Code Generation



esprima-examples — casiano@sanclemente-2:~/campus-virtual/1819/lsi-4-rpc-1819/casiano/esprima-examples — -bash — 94×31

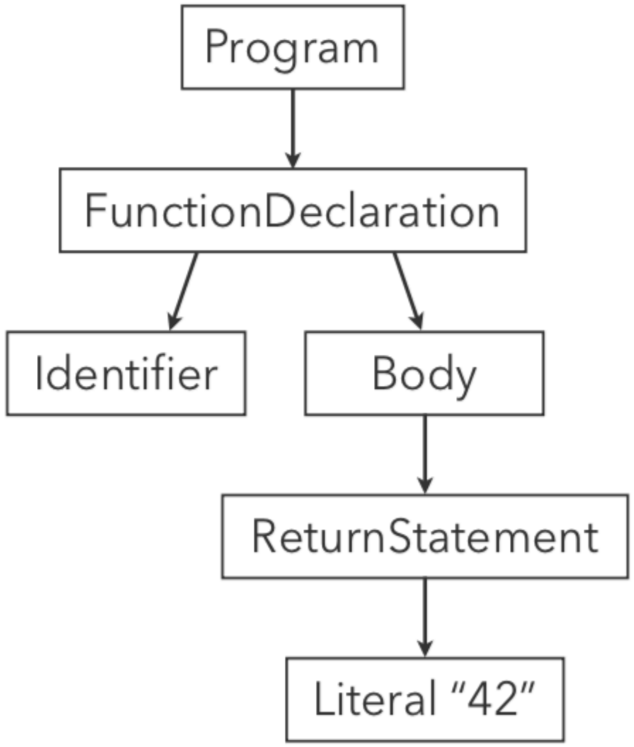
```
[~/campus-virtual/1819/lsi-4-rpc-1819/casiano/esprima-examples(master)]$ ls
ast-talk-codemotion-170406094223.pdf esprima-pegjs-jsconfeu-talk          jsconfeu-parsing.pdf
checkstyle.js                        hello-ast.js                         node_modules
[~/campus-virtual/1819/lsi-4-rpc-1819/casiano/esprima-examples(master)]$ cat hello-ast.js
const util = require('util');
const esprima = require('esprima');
const ast = esprima.parse(`
function getAnswer() {
 return 42;
}
`);
console.log(util.inspect(ast, {depth: Math.Infinity}));
[~/campus-virtual/1819/lsi-4-rpc-1819/casiano/esprima-examples(master)]$ node hello-ast.js
Script {
  type: 'Program',
  body:
   [ FunctionDeclaration {
       type: 'FunctionDeclaration',
       id: Identifier { type: 'Identifier', name: 'getAnswer' },
       params: [],
       body:
        BlockStatement {
          type: 'BlockStatement',
          body:
           [ ReturnStatement {
               type: 'ReturnStatement',
               argument: Literal { type: 'Literal', value: 42, raw: '42' } } ] },
       generator: false,
       expression: false,
       async: false } ],
  sourceType: 'script' }
```

Parsing, Traversing and Code Generation



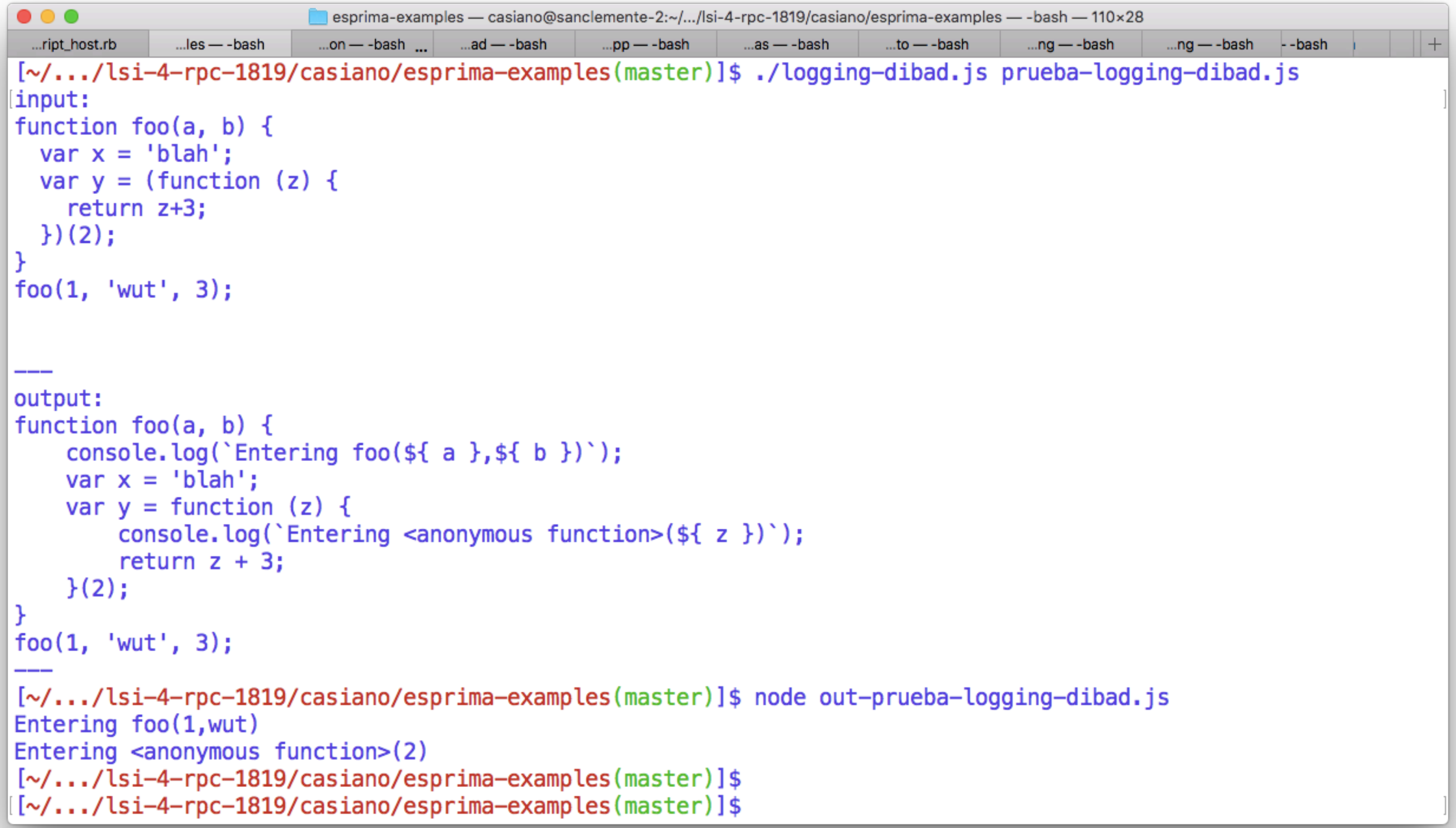
```

function getAnswer() {
 return 42;
}
```

```
Script {
  type: 'Program',
  body:
   [ FunctionDeclaration {
       type: 'FunctionDeclaration',
       id: Identifier { type: 'Ident
       params: [],
       body:
        BlockStatement {
          type: 'BlockStatement',
          body:
           [ ReturnStatement {
               type: 'ReturnStatemen
               argument: Literal { t
       generator: false,
       expression: false,
       async: false } ],
  sourceType: 'script' }
```



$$\begin{tikzcd}
 & \boxed{\text{Program}} \arrow[d] & \\
 & \boxed{\text{FunctionDeclaration}} \arrow[dl] \arrow[dr] & \\
\boxed{\text{Identifier}} & & \boxed{\text{Body}} \arrow[d] \\
 & & \boxed{\text{ReturnStatement}} \arrow[d] \\
 & & \boxed{\text{Literal "42"}}
\end{tikzcd}$$

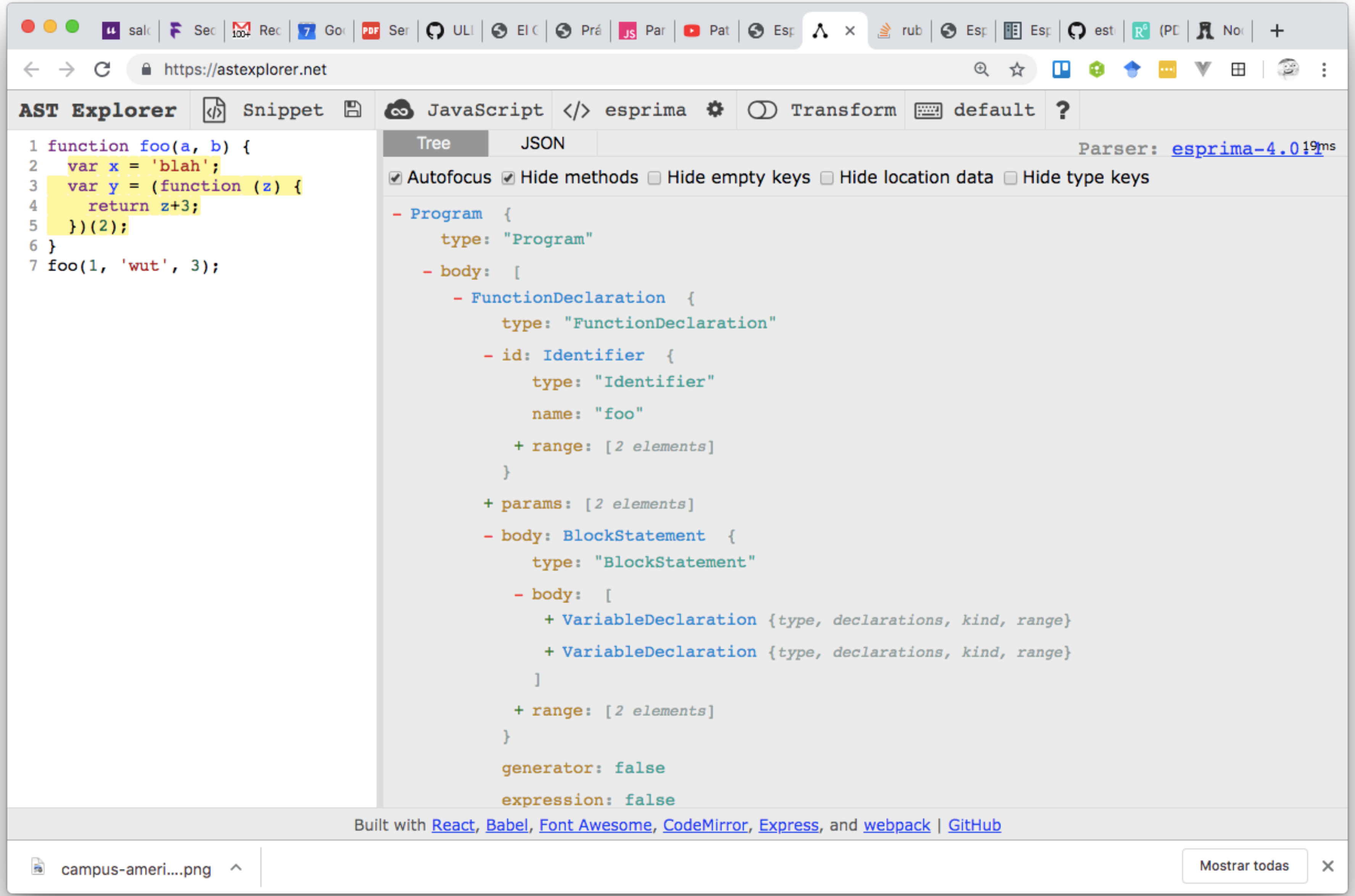
Parsing and Traversing Example: Logging function calls



esprima-examples — casiano@sanclemente-2:~/.../lsi-4-rpc-1819/casiano/esprima-examples — -bash — 110×28

```
[~/.../lsi-4-rpc-1819/casiano/esprima-examples(master)]$ ./logging-dibad.js prueba-logging-dibad.js
input:
function foo(a, b) {
  var x = 'blah';
  var y = (function (z) {
    return z+3;
  })(2);
}
foo(1, 'wut', 3);



---
output:
function foo(a, b) {
    console.log(`Entering foo(${ a },${ b })`);
    var x = 'blah';
    var y = function (z) {
        console.log(`Entering <anonymous function>(${ z })`);
        return z + 3;
    }(2);
}
foo(1, 'wut', 3);
---
[~/.../lsi-4-rpc-1819/casiano/esprima-examples(master)]$ node out-prueba-logging-dibad.js
Entering foo(1,wut)
Entering <anonymous function>(2)
[~/.../lsi-4-rpc-1819/casiano/esprima-examples(master)]$
[~/.../lsi-4-rpc-1819/casiano/esprima-examples(master)]$
```

https://astexplorer.net/

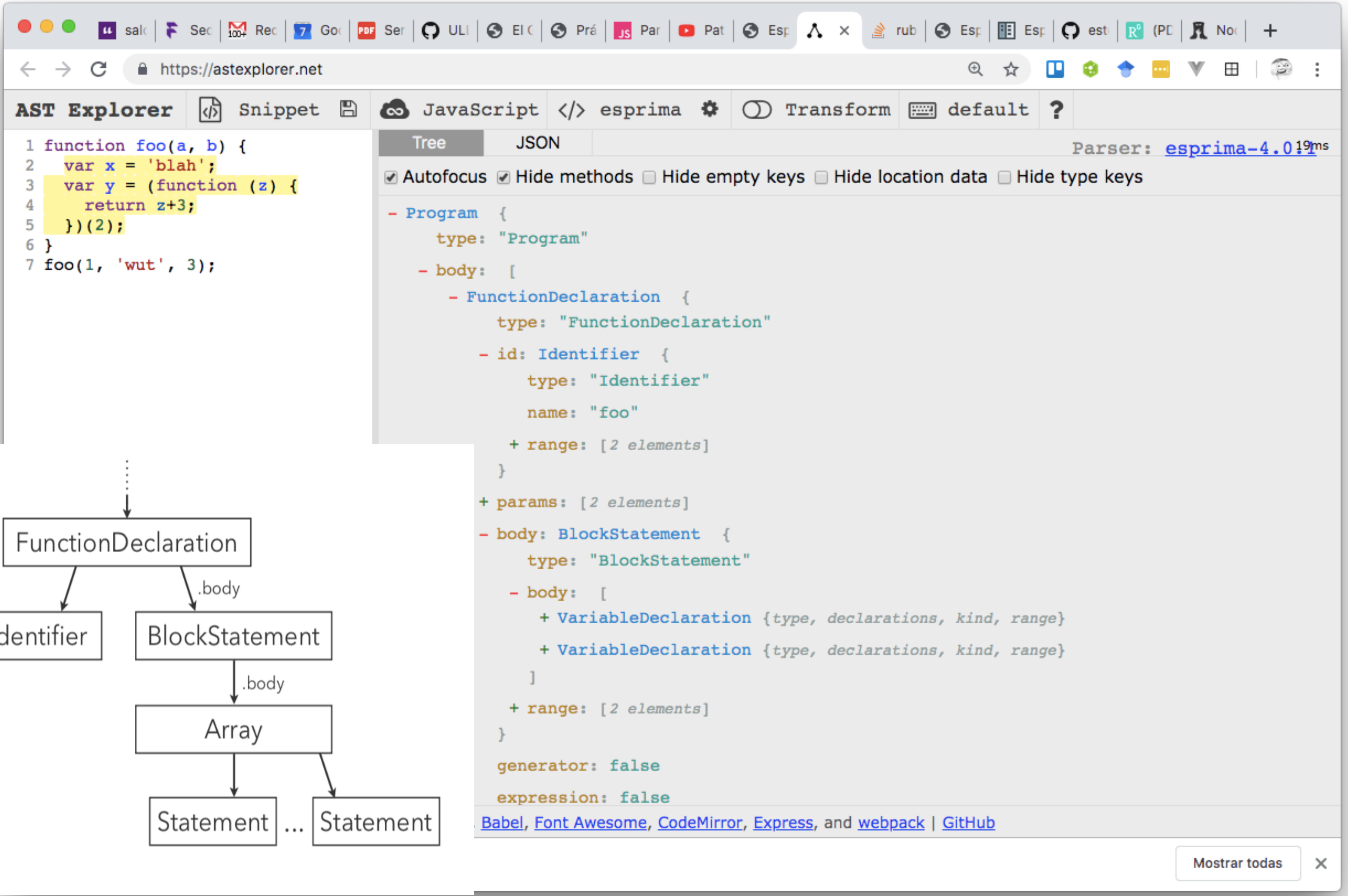


https://astexplorer.net
AST Explorer
Snippet
JavaScript
esprima
Transform
default
Tree
JSON
Parser: esprima-4.0.1
19ms
☑ Autofocus ☑ Hide methods ☐ Hide empty keys ☐ Hide location data ☐ Hide type keys
Built with React, Babel, Font Awesome, CodeMirror, Express, and webpack | GitHub
campus-ameri....png
Mostrar todas

```
function foo(a, b) {
  var x = 'blah';
  var y = (function (z) {
    return z+3;
  })(2);
}
foo(1, 'wut', 3);
```

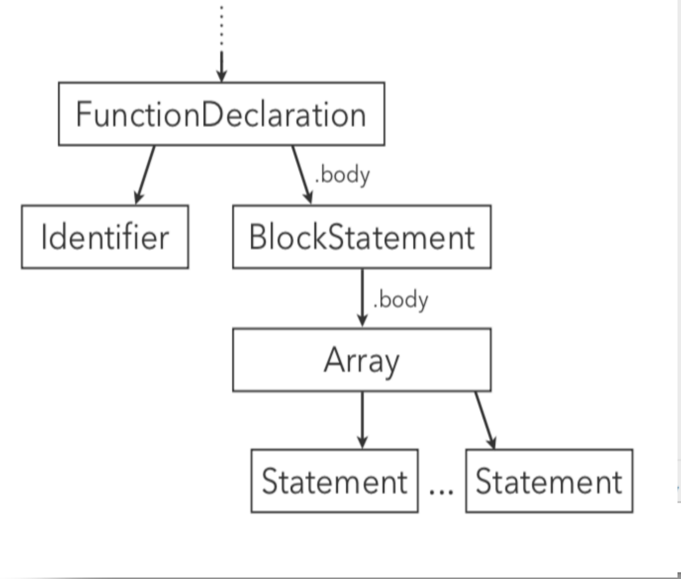
```
- Program {
    type: "Program"
  - body: [
    - FunctionDeclaration {
        type: "FunctionDeclaration"
      - id: Identifier {
          type: "Identifier"
          name: "foo"
        + range: [2 elements]
        }
      + params: [2 elements]
      - body: BlockStatement {
          type: "BlockStatement"
        - body: [
          + VariableDeclaration {type, declarations, kind, range}
          + VariableDeclaration {type, declarations, kind, range}
          ]
        + range: [2 elements]
        }
        generator: false
        expression: false
```

https://astexplorer.net/

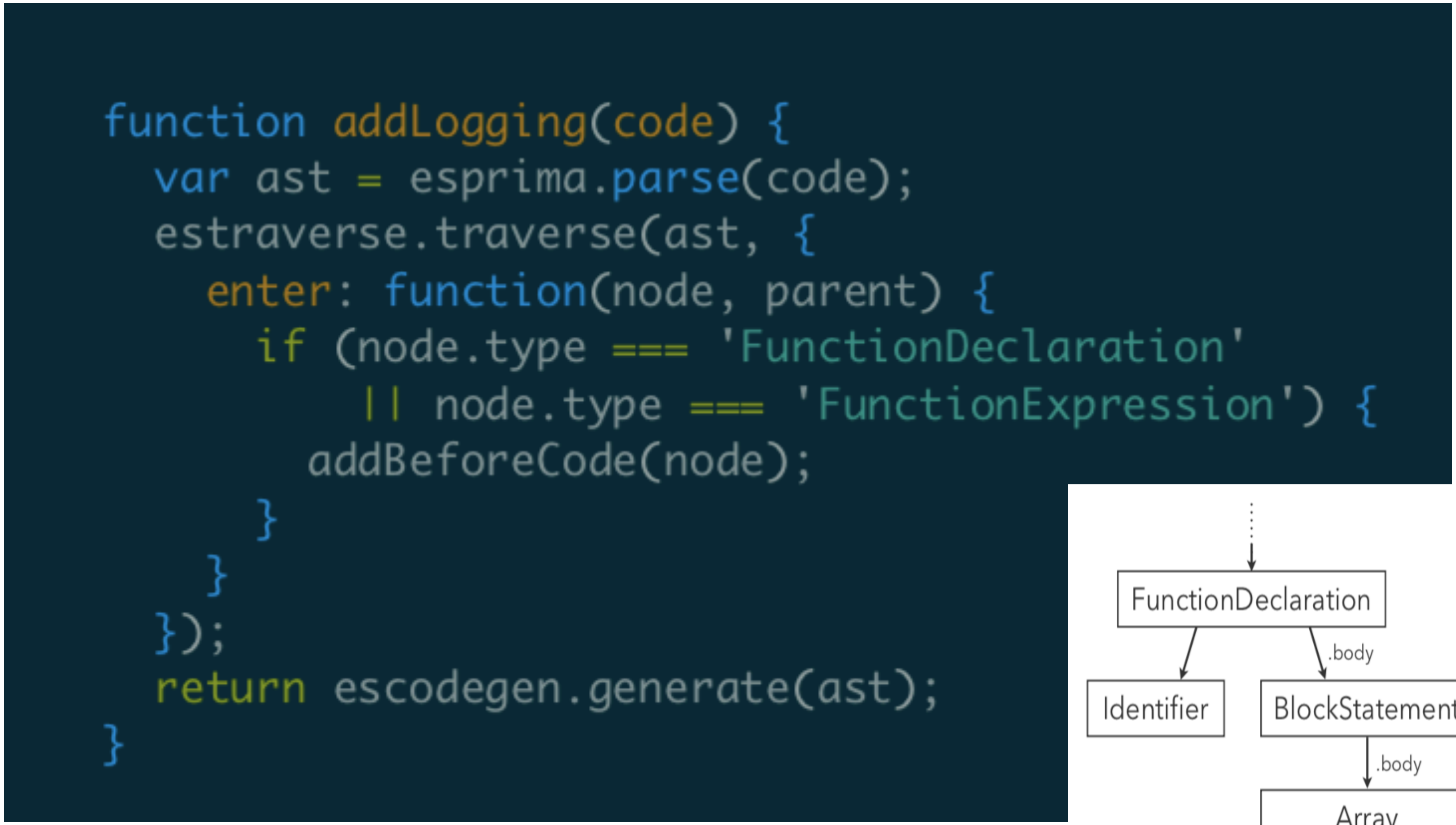


```
function foo(a, b) {
  var x = 'blah';
  var y = (function (z) {
    return z+3;
  })(2);
}
foo(1, 'wut', 3);
```

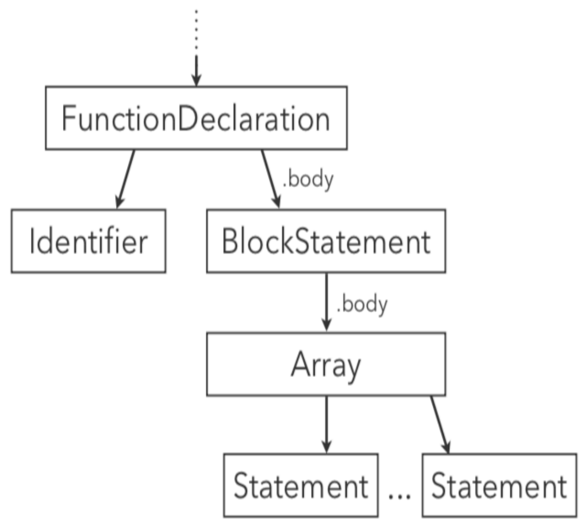
```
- Program {
    type: "Program"
  - body: [
    - FunctionDeclaration {
        type: "FunctionDeclaration"
      - id: Identifier {
          type: "Identifier"
          name: "foo"
        + range: [2 elements]
        }
      + params: [2 elements]
      - body: BlockStatement {
          type: "BlockStatement"
        - body: [
            + VariableDeclaration {type, declarations, kind, range}
            + VariableDeclaration {type, declarations, kind, range}
          ]
        + range: [2 elements]
        }
        generator: false
        expression: false
```



Parsing, Traversing and Generating Code

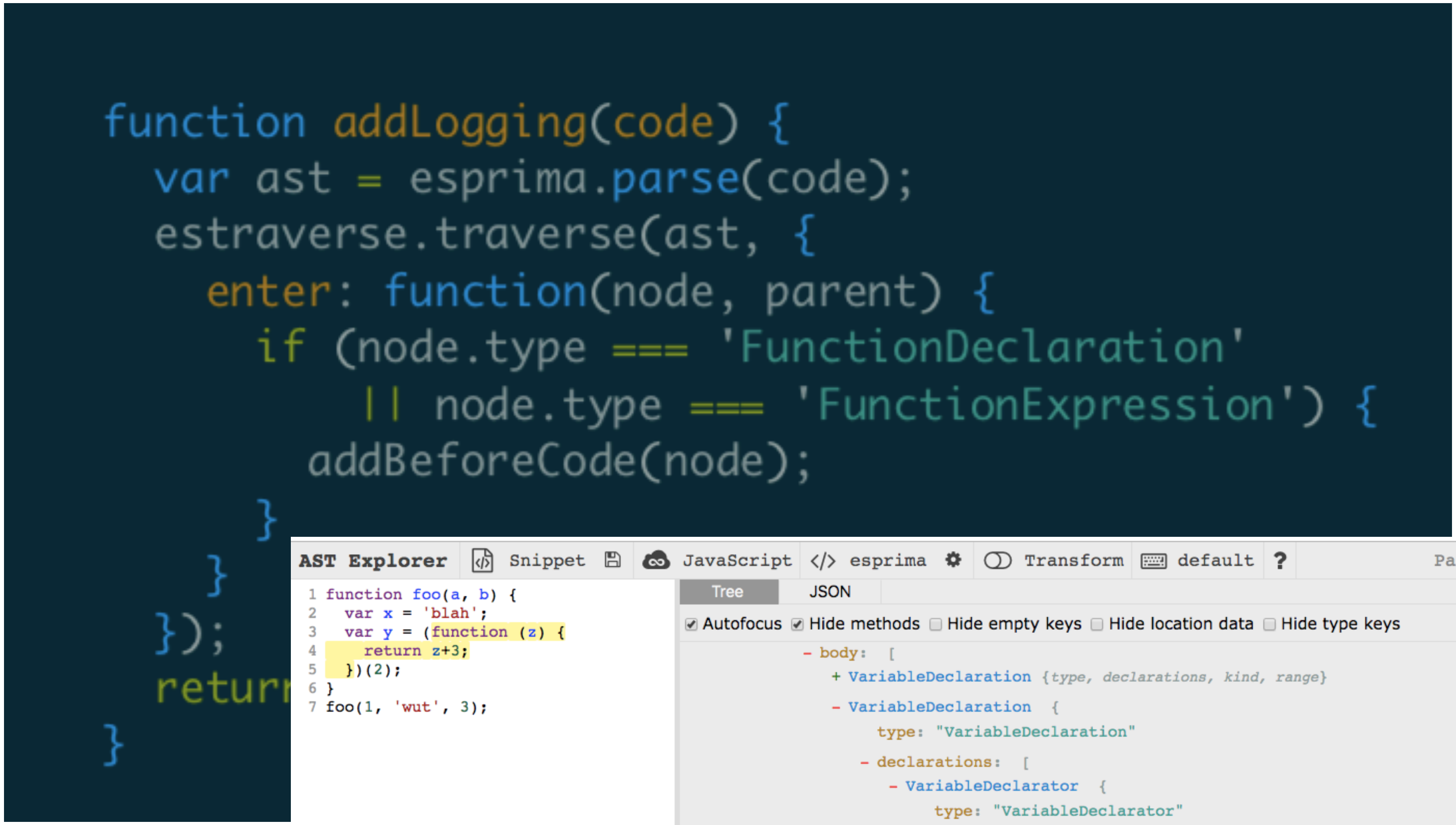


```
function addLogging(code) {
  var ast = esprima.parse(code);
  estraverse.traverse(ast, {
    enter: function(node, parent) {
      if (node.type === 'FunctionDeclaration'
          || node.type === 'FunctionExpression') {
        addBeforeCode(node);
      }
    }
  });
  return escodegen.generate(ast);
}
```

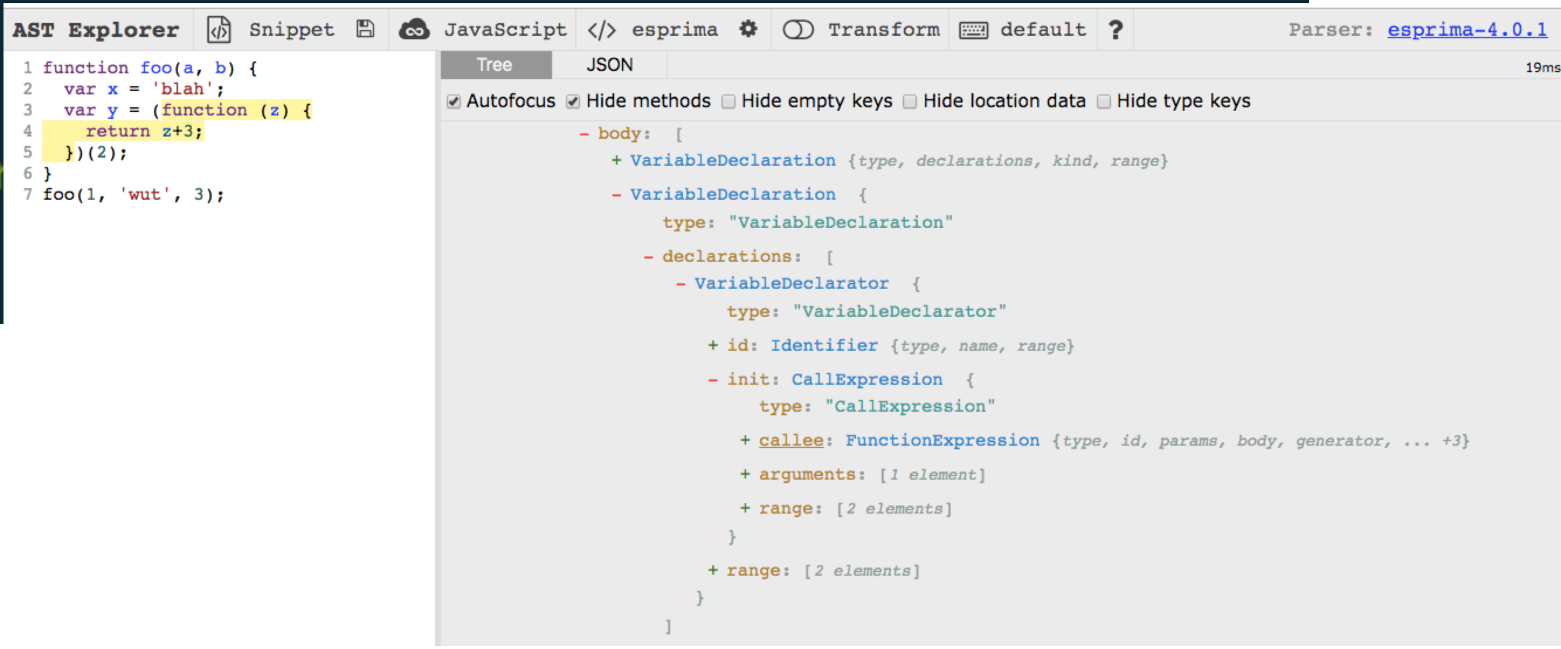


API de estraverse: https://github.com/estools/estraverse

Parsing, Traversing and Generating Code



```
function addLogging(code) {
  var ast = esprima.parse(code);
  estraverse.traverse(ast, {
    enter: function(node, parent) {
      if (node.type === 'FunctionDeclaration'
          || node.type === 'FunctionExpression') {
        addBeforeCode(node);
      }
    }
  });
  retur
}
```



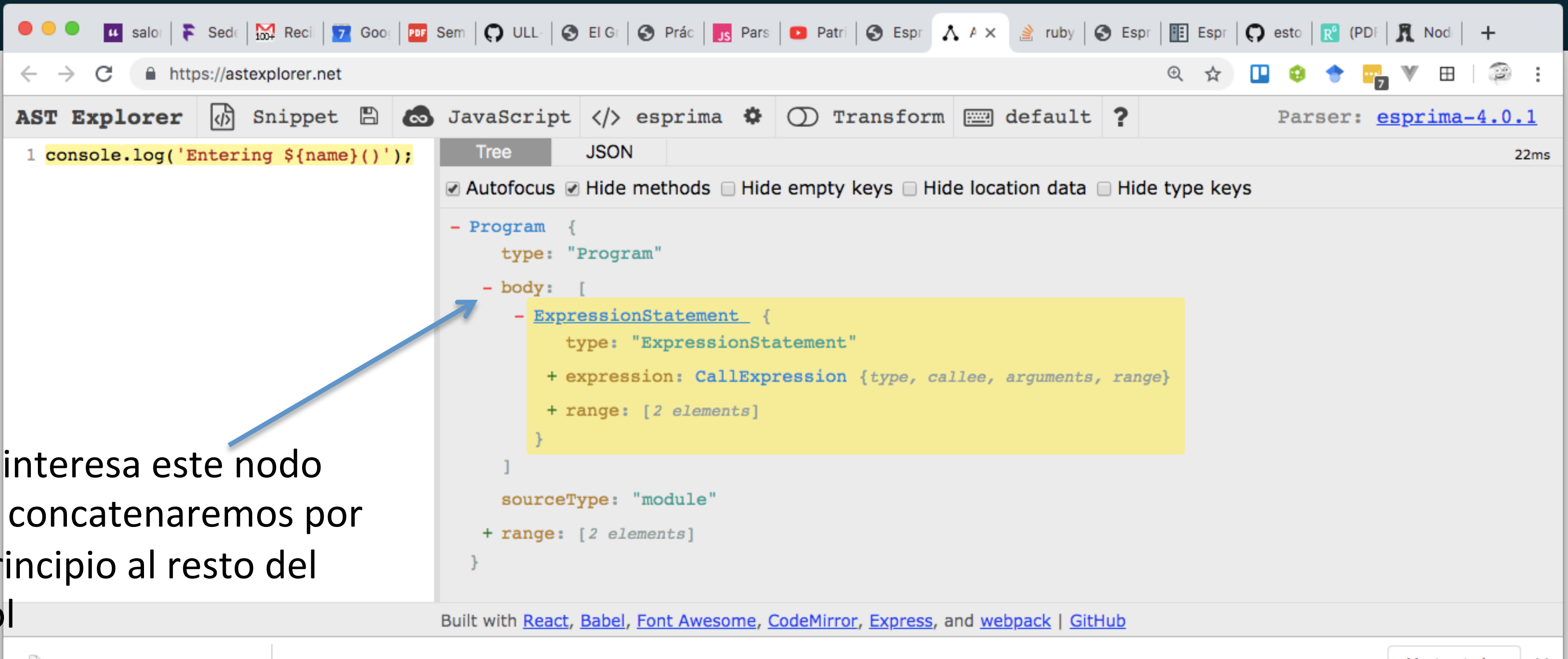
```
function foo(a, b) {
  var x = 'blah';
  var y = (function (z) {
    return z+3;
  })(2);
}
foo(1, 'wut', 3);
```

☑ Autofocus ☑ Hide methods ☐ Hide empty keys ☐ Hide location data ☐ Hide type keys

```
- body: [
    + VariableDeclaration {type, declarations, kind, range}
    - VariableDeclaration {
        type: "VariableDeclaration"
      - declarations: [
          - VariableDeclarator {
              type: "VariableDeclarator"
            + id: Identifier {type, name, range}
            - init: CallExpression {
                type: "CallExpression"
              + callee: FunctionExpression {type, id, params, body, generator, ... +3}
              + arguments: [1 element]
              + range: [2 elements]
              }
            + range: [2 elements]
            }
        ]
```

Traversing and Modifying the AST

```
function addBeforeCode(node) {
  var name = node.id ? node.id.name : '<anonymous function>';
  var beforeCode = "console.log('Entering " + name + "()');";
  var beforeNodes = esprima.parse(beforeCode).body;
  node.body.body = beforeNodes.concat(node.body.body);
}
```



```
console.log('Entering ${name}()');
```

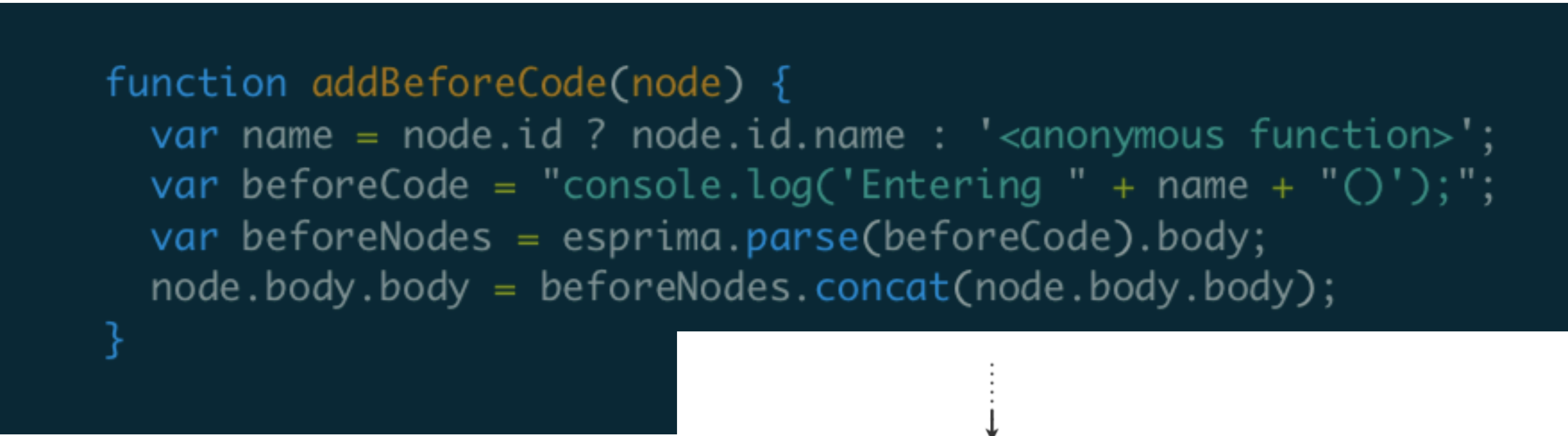
☑ Autofocus ☑ Hide methods ☐ Hide empty keys ☐ Hide location data ☐ Hide type keys

```
- Program {
    type: "Program"
  - body: [
     - ExpressionStatement {
         type: "ExpressionStatement"
       + expression: CallExpression {type, callee, arguments, range}
       + range: [2 elements]
       }
    ]
    sourceType: "module"
  + range: [2 elements]
}
```

Built with React, Babel, Font Awesome, CodeMirror, Express, and webpack | GitHub

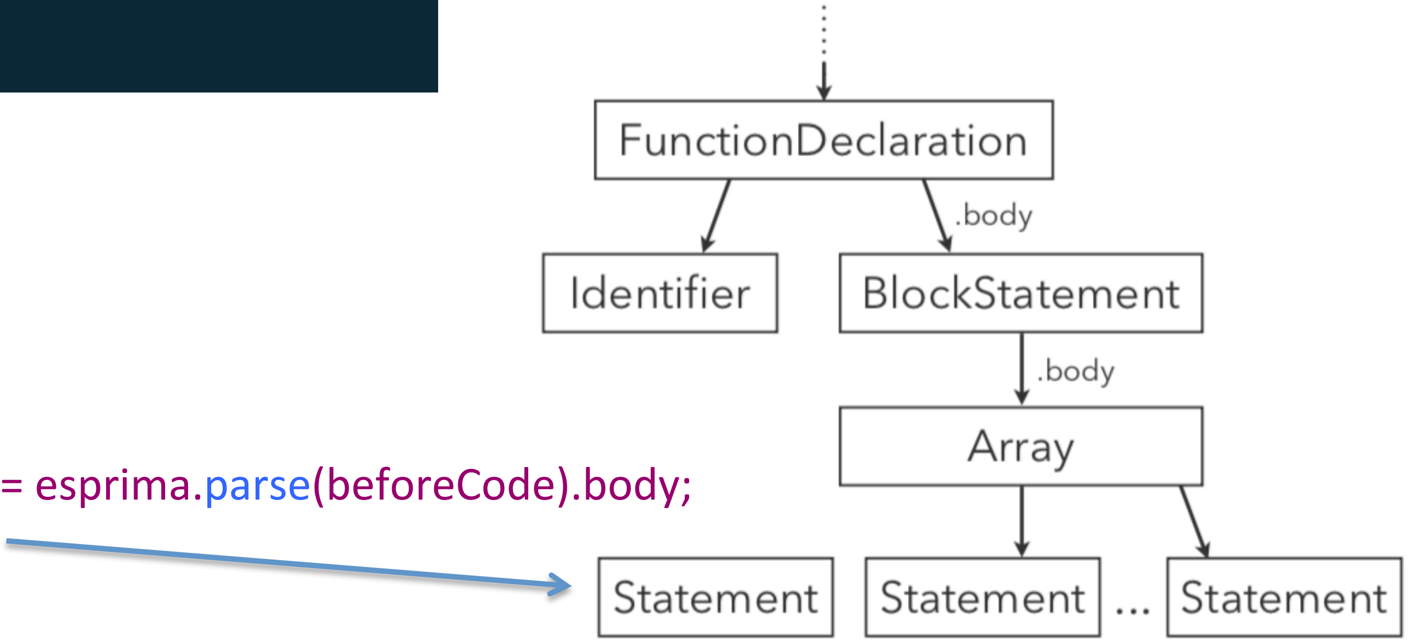
Nos interesa este nodo
Que concatenaremos por el principio al resto del árbol

Traversing and Modifying the AST

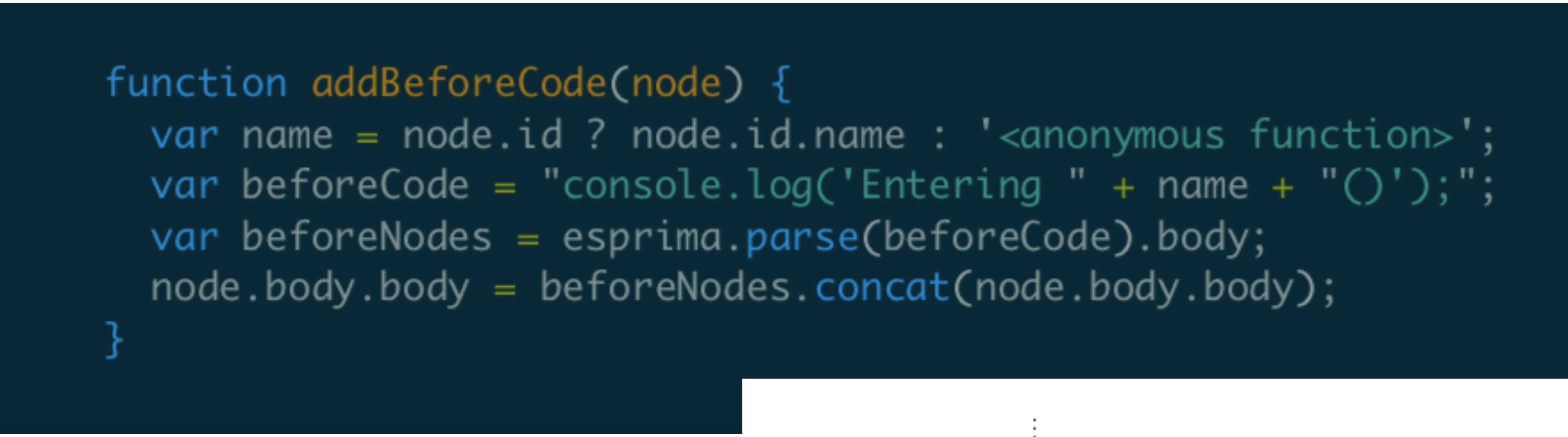


```
function addBeforeCode(node) {
  var name = node.id ? node.id.name : '<anonymous function>';
  var beforeCode = "console.log('Entering " + name + "()');";
  var beforeNodes = esprima.parse(beforeCode).body;
  node.body.body = beforeNodes.concat(node.body.body);
}
```

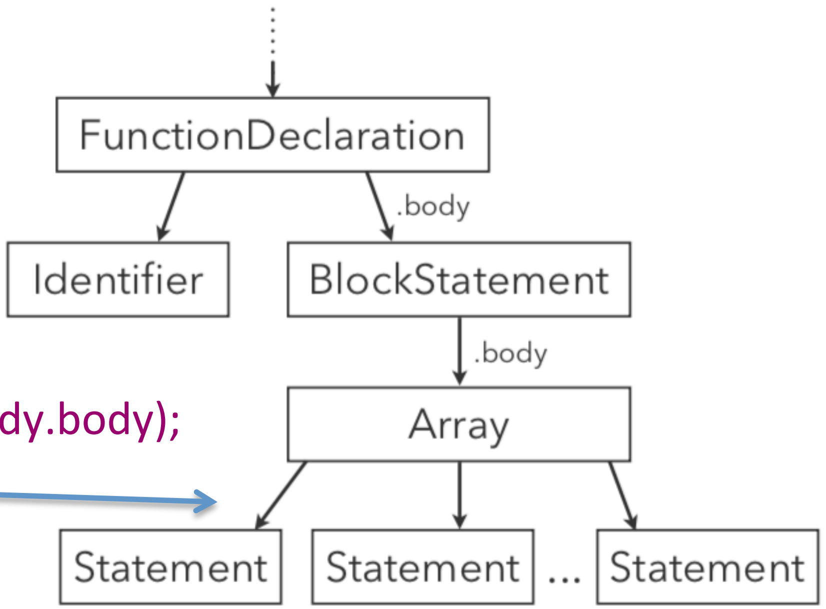
var beforeNodes = esprima.parse(beforeCode).body;



Parsing, Traversing and Modifying the AST



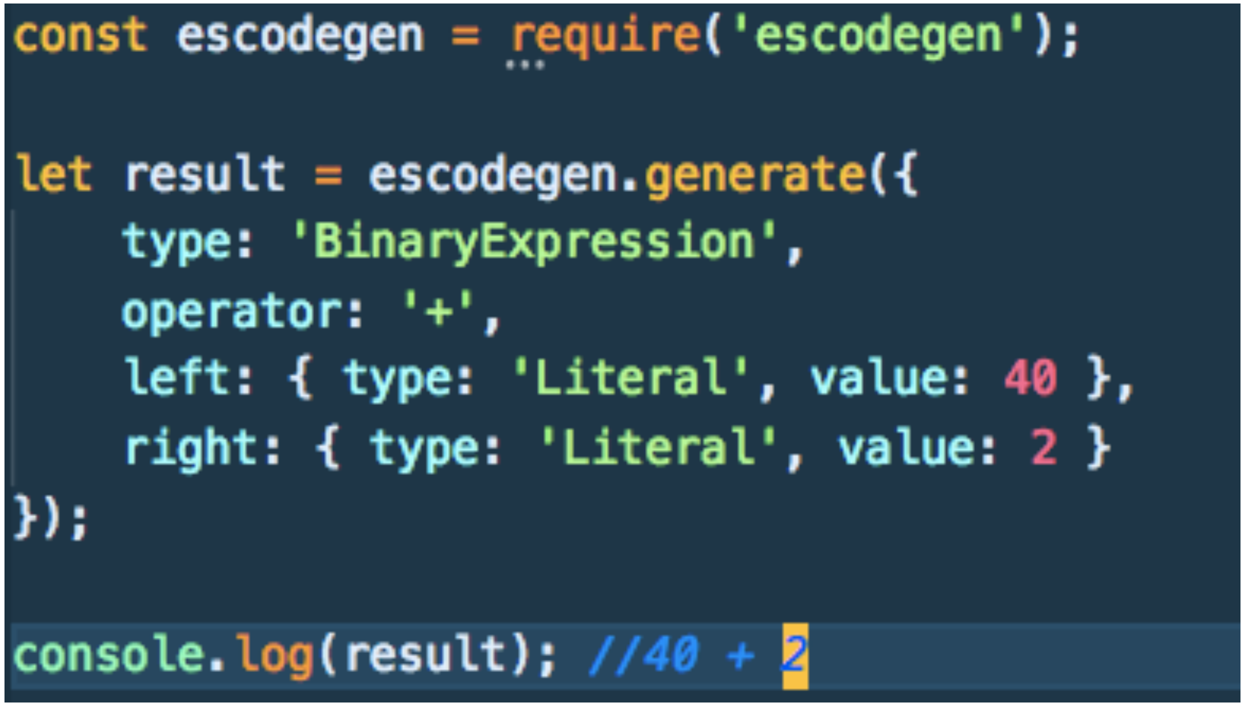
```
function addBeforeCode(node) {
  var name = node.id ? node.id.name : '<anonymous function>';
  var beforeCode = "console.log('Entering " + name + "()');";
  var beforeNodes = esprima.parse(beforeCode).body;
  node.body.body = beforeNodes.concat(node.body.body);
}
```



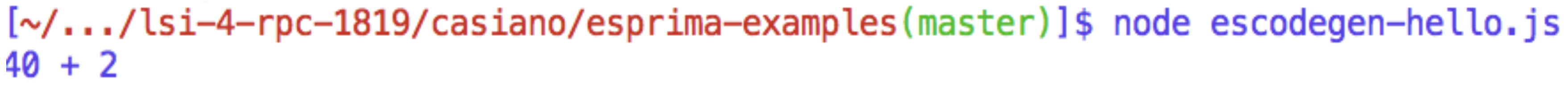
node.body.body = beforeNodes.concat(node.body.body);

El método concat() se usa para unir dos o más arrays

Generating Code from the AST



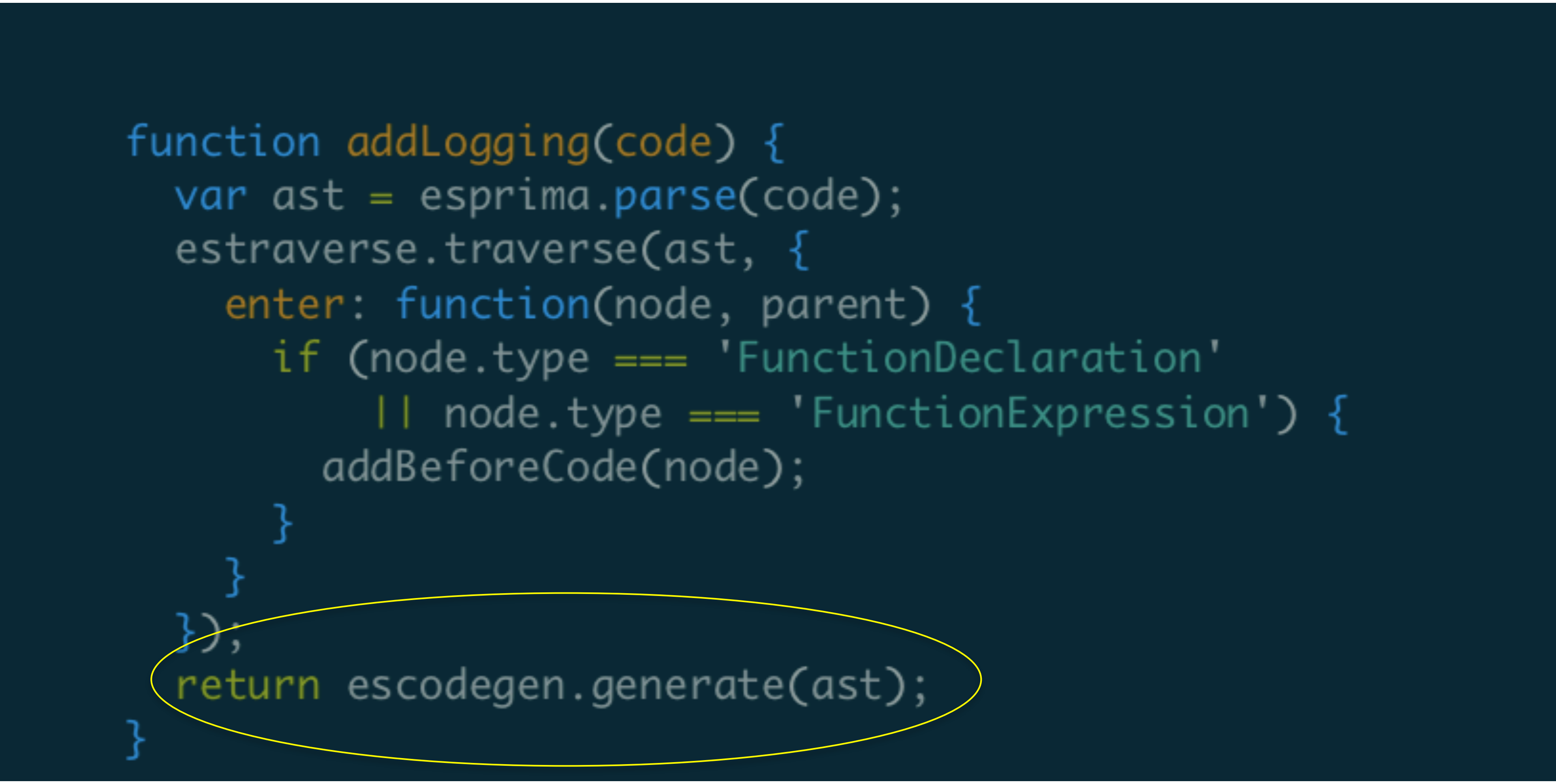
```
const escodegen = require('escodegen');

let result = escodegen.generate({
    type: 'BinaryExpression',
    operator: '+',
    left: { type: 'Literal', value: 40 },
    right: { type: 'Literal', value: 2 }
});

console.log(result); //40 + 2
```



```
[~/.../lsi-4-rpc-1819/casiano/esprima-examples(master)]$ node escodegen-hello.js
40 + 2
```

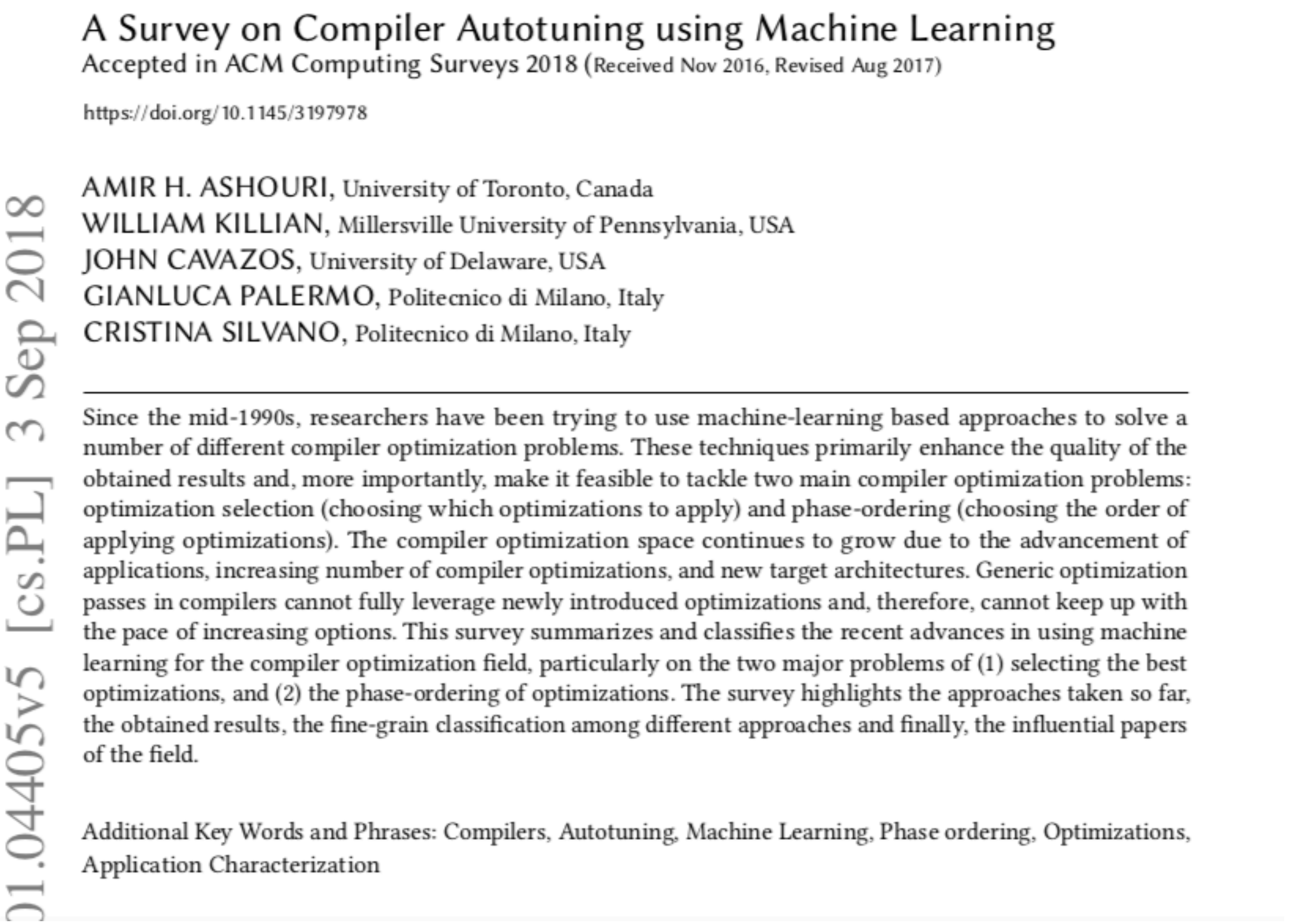
API de escodegen: https://github.com/estools/escodegen/wiki/API

Generating Code from the AST



```
function addLogging(code) {
  var ast = esprima.parse(code);
  estraverse.traverse(ast, {
    enter: function(node, parent) {
      if (node.type === 'FunctionDeclaration'
          || node.type === 'FunctionExpression') {
        addBeforeCode(node);
      }
    }
  });
  return escodegen.generate(ast);
}
```

Code Optimization



A Survey on Compiler Autotuning using Machine Learning

Accepted in ACM Computing Surveys 2018 (Received Nov 2016, Revised Aug 2017)

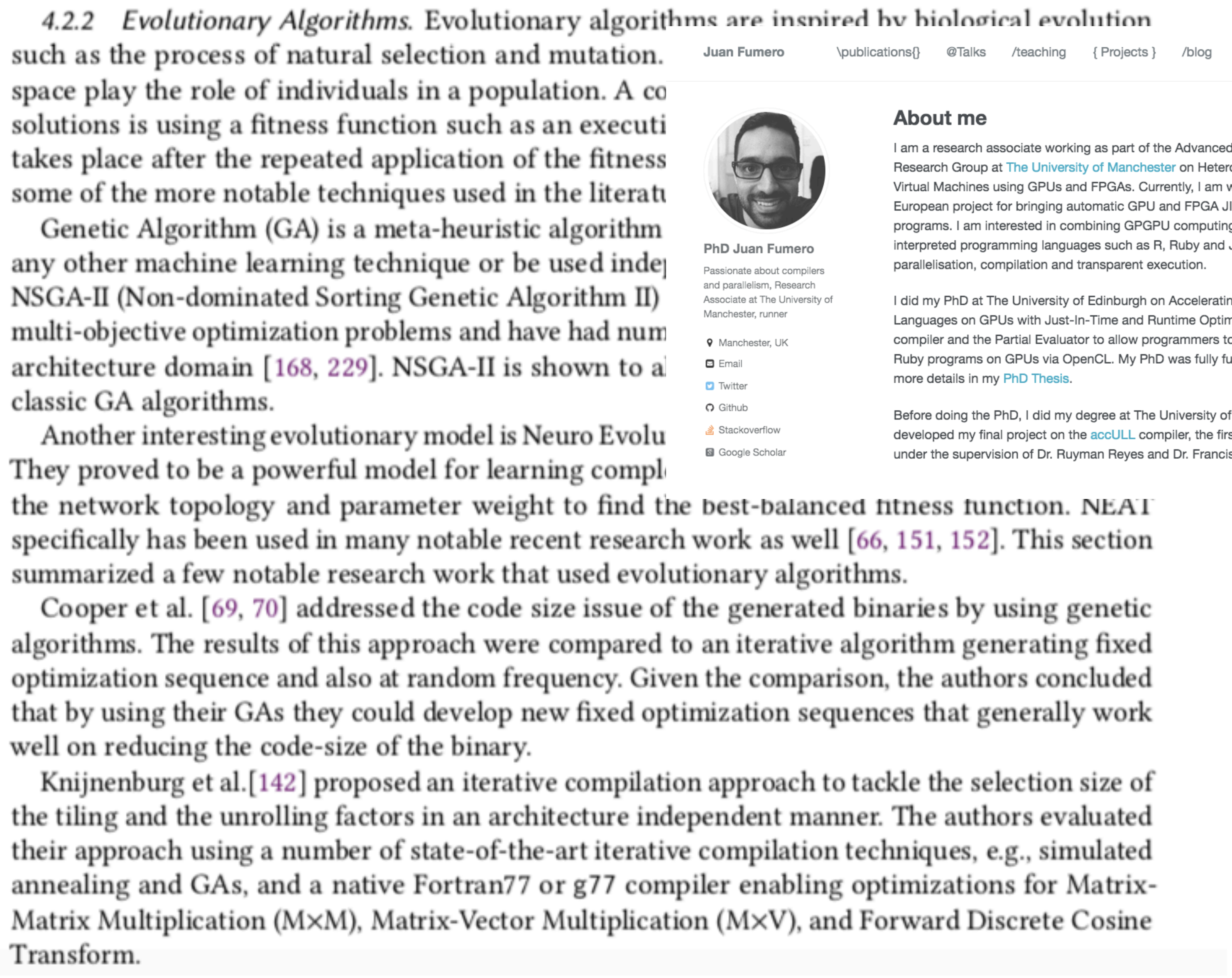
https://doi.org/10.1145/3197978

AMIR H. ASHOURI, University of Toronto, Canada
WILLIAM KILLIAN, Millersville University of Pennsylvania, USA
JOHN CAVAZOS, University of Delaware, USA
GIANLUCA PALERMO, Politecnico di Milano, Italy
CRISTINA SILVANO, Politecnico di Milano, Italy

Since the mid-1990s, researchers have been trying to use machine-learning based approaches to solve a number of different compiler optimization problems. These techniques primarily enhance the quality of the obtained results and, more importantly, make it feasible to tackle two main compiler optimization problems: optimization selection (choosing which optimizations to apply) and phase-ordering (choosing the order of applying optimizations). The compiler optimization space continues to grow due to the advancement of applications, increasing number of compiler optimizations, and new target architectures. Generic optimization passes in compilers cannot fully leverage newly introduced optimizations and, therefore, cannot keep up with the pace of increasing options. This survey summarizes and classifies the recent advances in using machine learning for the compiler optimization field, particularly on the two major problems of (1) selecting the best optimizations, and (2) the phase-ordering of optimizations. The survey highlights the approaches taken so far, the obtained results, the fine-grain classification among different approaches and finally, the influential papers of the field.

Additional Key Words and Phrases: Compilers, Autotuning, Machine Learning, Phase ordering, Optimizations, Application Characterization

Code Optimization



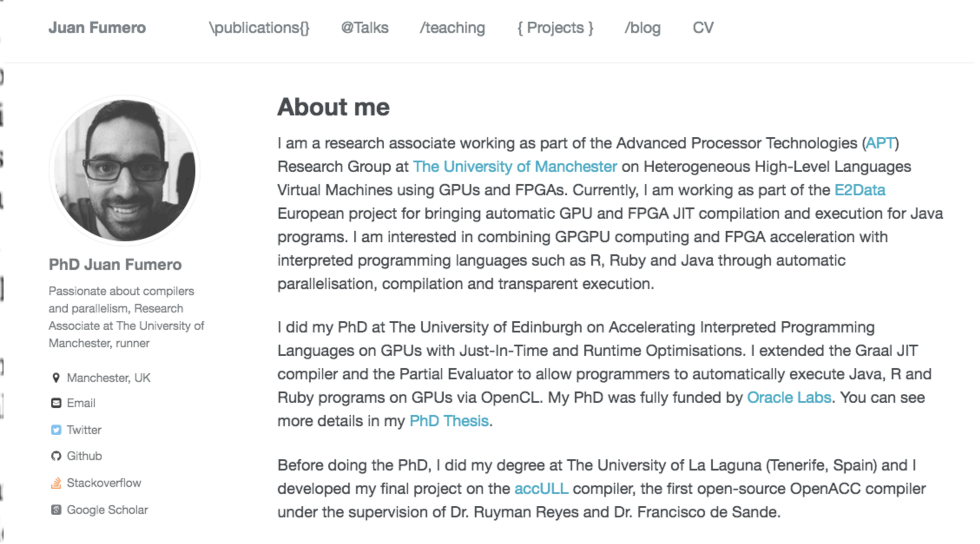
4.2.2 Evolutionary Algorithms. Evolutionary algorithms are inspired by biological evolution such as the process of natural selection and mutation.
space play the role of individuals in a population. A co
solutions is using a fitness function such as an executi
takes place after the repeated application of the fitness
some of the more notable techniques used in the literatu

Genetic Algorithm (GA) is a meta-heuristic algorithm
any other machine learning technique or be used inde
NSGA-II (Non-dominated Sorting Genetic Algorithm II)
multi-objective optimization problems and have had num
architecture domain [168, 229]. NSGA-II is shown to a
classic GA algorithms.

Another interesting evolutionary model is Neuro Evolu
They proved to be a powerful model for learning compl
the network topology and parameter weight to find the best-balanced fitness function. NEAT specifically has been used in many notable recent research work as well [66, 151, 152]. This section summarized a few notable research work that used evolutionary algorithms.

Cooper et al. [69, 70] addressed the code size issue of the generated binaries by using genetic algorithms. The results of this approach were compared to an iterative algorithm generating fixed optimization sequence and also at random frequency. Given the comparison, the authors concluded that by using their GAs they could develop new fixed optimization sequences that generally work well on reducing the code-size of the binary.

Knijnenburg et al.[142] proposed an iterative compilation approach to tackle the selection size of the tiling and the unrolling factors in an architecture independent manner. The authors evaluated their approach using a number of state-of-the-art iterative compilation techniques, e.g., simulated annealing and GAs, and a native Fortran77 or g77 compiler enabling optimizations for Matrix-Matrix Multiplication (M×M), Matrix-Vector Multiplication (M×V), and Forward Discrete Cosine Transform.



Juan Fumero \publications{} @Talks /teaching { Projects } /blog CV

PhD Juan Fumero

Passionate about compilers and parallelism, Research Associate at The University of Manchester, runner

Manchester, UK
Email
Twitter
Github
Stackoverflow
Google Scholar

About me

I am a research associate working as part of the Advanced Processor Technologies (APT) Research Group at The University of Manchester on Heterogeneous High-Level Languages Virtual Machines using GPUs and FPGAs. Currently, I am working as part of the E2Data European project for bringing automatic GPU and FPGA JIT compilation and execution for Java programs. I am interested in combining GPGPU computing and FPGA acceleration with interpreted programming languages such as R, Ruby and Java through automatic parallelisation, compilation and transparent execution.

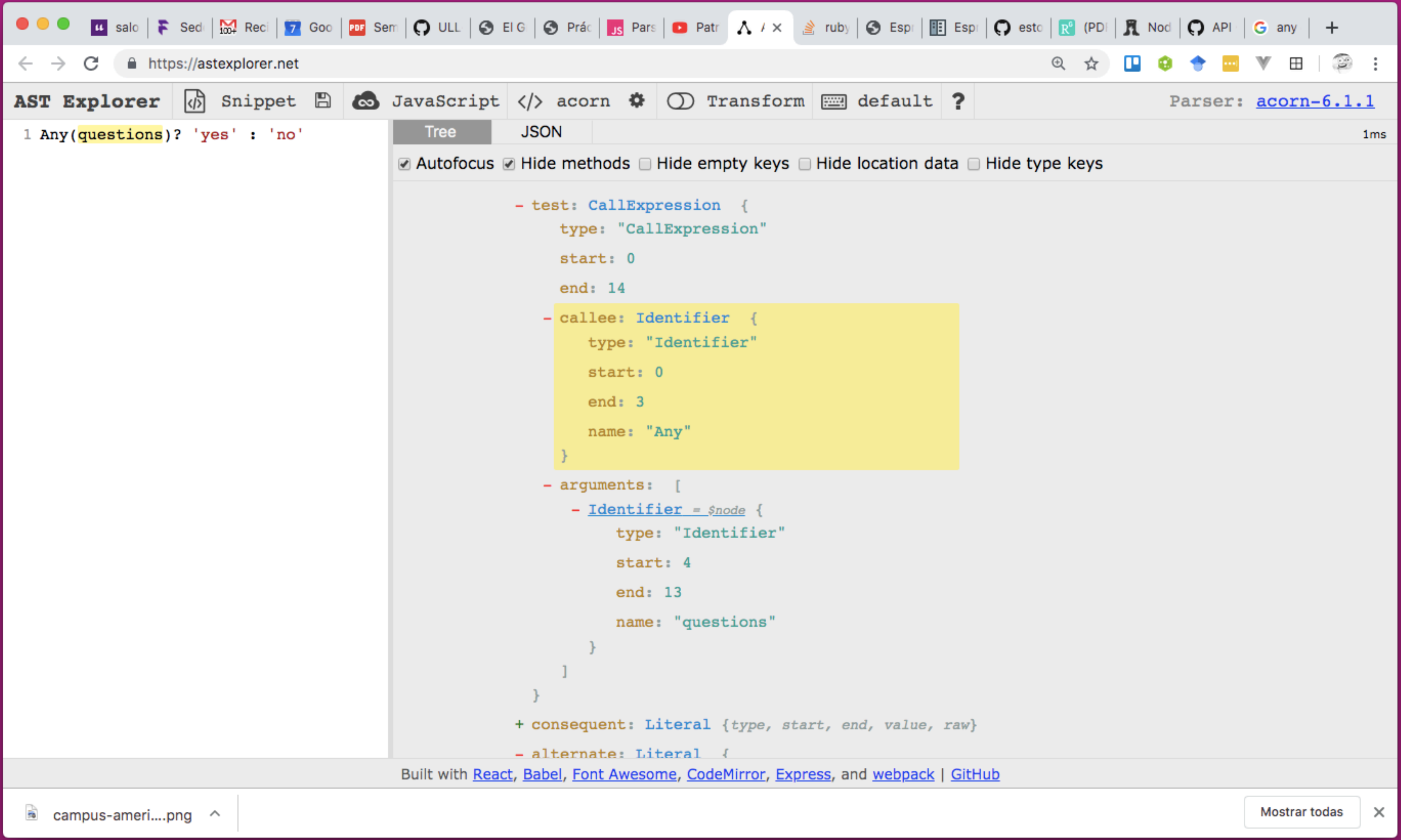
I did my PhD at The University of Edinburgh on Accelerating Interpreted Programming Languages on GPUs with Just-In-Time and Runtime Optimisations. I extended the Graal JIT compiler and the Partial Evaluator to allow programmers to automatically execute Java, R and Ruby programs on GPUs via OpenCL. My PhD was fully funded by Oracle Labs. You can see more details in my PhD Thesis.

Before doing the PhD, I did my degree at The University of La Laguna (Tenerife, Spain) and I developed my final project on the accULL compiler, the first open-source OpenACC compiler under the supervision of Dr. Ruyman Reyes and Dr. Francisco de Sande.

Recursos

- Repositorio GitHub con los recursos de la charla: https://github.com/ULL-LSI/campus-america-2019
- Apuntes de Procesadores de Lenguajes. Curso 2018/2019: https://ull-esit-pl-1819.github.io/introduccion/
- Rodriguez-Leon, Casiano & Garcia-Forte, L. (2011). Solving Difficult LR Parsing Conflicts by Postponing Them. Comput. Sci. Inf. Syst.. 8. 517-531. 10.2298/CSIS101116008R.
- Parse Eyapp en CPAN
- Parsing Strings and Trees with Parse::Eyapp (An Introduction to Compiler Construction). 2010
- Patrick Dubroy: Parsing, Compiling, and Static Metaprogramming
- https://astexplorer.net/

Any(Questions)?



```
Any(questions)? 'yes' : 'no'
```

☑ Autofocus ☑ Hide methods ☐ Hide empty keys ☐ Hide location data ☐ Hide type keys

```
- test: CallExpression  {
     type: "CallExpression"
     start: 0
     end: 14
   - callee: Identifier  {
        type: "Identifier"
        start: 0
        end: 3
        name: "Any"
     }
   - arguments:  [
      - Identifier = $node {
           type: "Identifier"
           start: 4
           end: 13
           name: "questions"
        }
     ]
  }
+ consequent: Literal {type, start, end, value, raw}
- alternate: Literal  {
```