



24



24



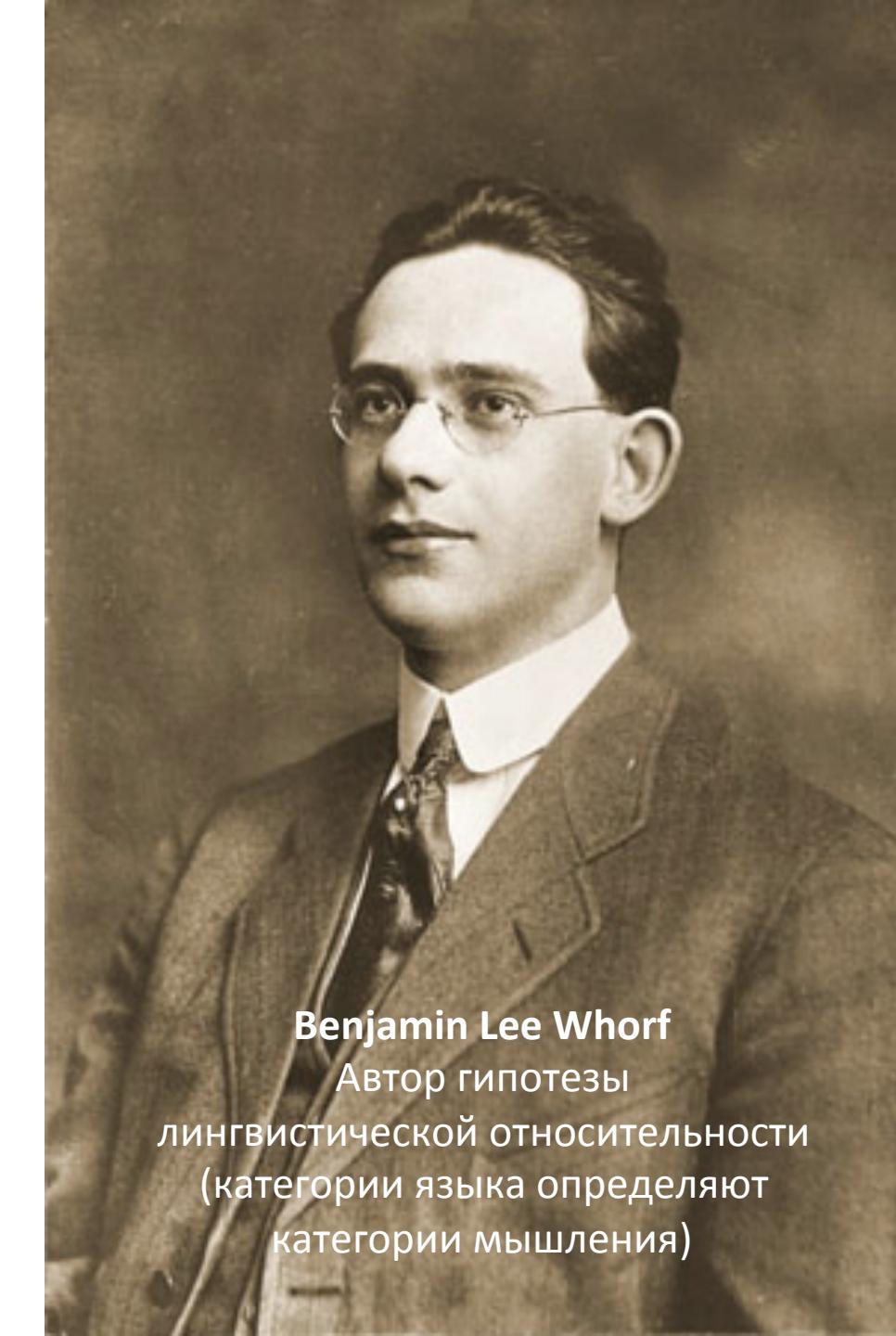
24

Как написать компилятор за 15 минут?

Андрей Гершун

AlaSQL

Зачем еще языки кроме JavaScript?



Benjamin Lee Whorf

Автор гипотезы
лингвистической относительности
(категории языка определяют
категории мышления)



Содержание

- Знакомьтесь – MATRIX!
- Инструменты
- Орфография
- Грамматика
- Семантика
- Пишем интерпретатор
- Венец творения
– компилятор!

Как происходит процесс понимания и выполнения?

- Лексер
- Парсер
- Run-time library
- Интерпретатор

или

- Компилятор



Знакомьтесь, MATRIX!

MATRIX

- Язык работы с матрицами
- Вдохновлен MATLAB, OCTANE, R и Q

$$c = a * b'$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_{11} \cdot b_1 + a_{12} \cdot b_2 + a_{13} \cdot b_3 \\ a_{21} \cdot b_1 + a_{22} \cdot b_2 + a_{23} \cdot b_3 \\ a_{31} \cdot b_1 + a_{32} \cdot b_2 + a_{33} \cdot b_3 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$$

Пример программы на MATRIX

A = 1 2 3

B = 4|5|6

PRINT A*B

- Если все правильно,
то мы должны увидеть: 32

Примитивы

1 // число

1 2 3 4 // вектор-строка

1 | 2 | 3 // вектор-столбец

1 2 | 3 4 // матрица 2x2

Операции MATRIX

1 2 + 3 4 // => 4 6 - сложение

2 * 5 6 // => 10 12 - умножение

1 2' // => 1|2 -транспонирование

(1 2 + 3 4)*2 // => 8 12 - скобки

size(1 2|3 4) // => 2 2 - функция размера



Zach Carter
Автор Json



Воспользуемся
генератором
парсеров:

- `Json`

Альтернативы:

- `PEG.js`
- `ALTNR/4`

Структура файла с грамматикой (matrix0.json)

```
%{ /* А. Вспомогательный код */ %}
```

```
%lex
```

```
%options case-insensitive
```

```
%%
```

```
/* Б. Лексемы */
```

```
/lex
```

Структура файла (продолжение)

%ebnf

%start main

/* В. Приоритеты правил */

%%

/* Г. Грамматические правила */

Слова языка

- Комментарии //
- Пробелы
- Число
- Названия переменных и функций
- Знаки: + * ' = ()
- Ключевое слово: PRINT
- Конец файла: <<EOF>>

Как работает лексер?

a = 1 2 3 '



Описываем лексемы

регулярка

```
return 'лексема'
```

Бритва Оккама: Отсекаем все ненужное: комментарии и пробелы

// Комментарии

\//.*\$ return

// Пробелы, переносы строк

\s return



William of Ockam

Ключевые слова

'PRINT'

return 'PRINT'

'GO TO'

return 'GOTO'

Синонимы

'GOTO'

return 'GOTO'

Числа и литералы (после ключевых слов!)

```
/* Числа */  
[0-9]+(\.[0-9]*)<?           return 'NUMBER'
```

```
/* Литералы: названия переменных и  
функций */  
[A-Za-z_][A-Za-z_0-9]*<?      return 'LITERAL'
```

Немного магии (служебные лексемы)

```
/* Конец файла */  
<<EOF>>      return 'EOF'  
  
/* В конце отлавливаем  
нераспознанные символы */  
.            return 'INVALID'
```

Кофе-брейк 1:

Что выдаёт на выходе лексер?

- Проверим лексику:
 > `jison matrix1.json`
- Отладим с помощью `jison-debugger`

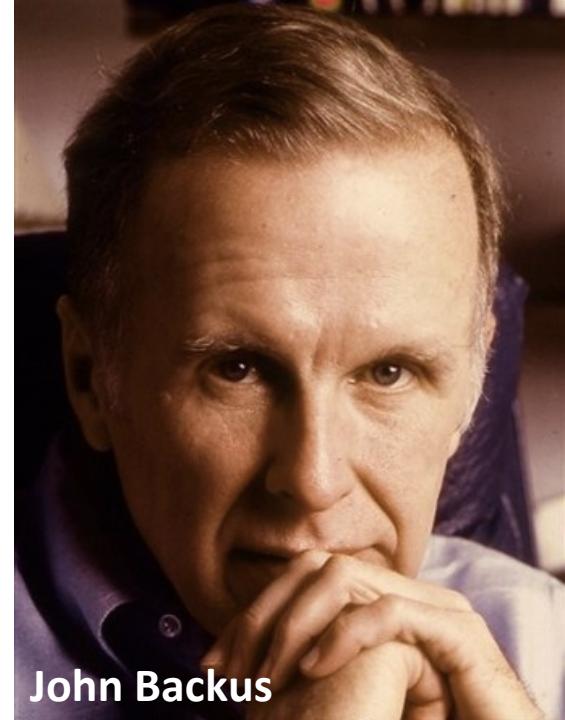
a = 1 2 3 '

| | | | | | | | |
|---------|----|--------|--------|--------|-------|-----|-------|
| a | = | 1 | 2 | 3 | ' | EOF | EOF |
| LITERAL | EQ | NUMBER | NUMBER | NUMBER | STRIH | EOF | \$end |

Теперь грамматика!

правило

```
: набор лексем 1
| набор лексем 2
| набор лексем 3
;
```



Описываем операторы

```
main
    : Statement* EOF ;
```

Statement

: Print | Set ;

Выражения

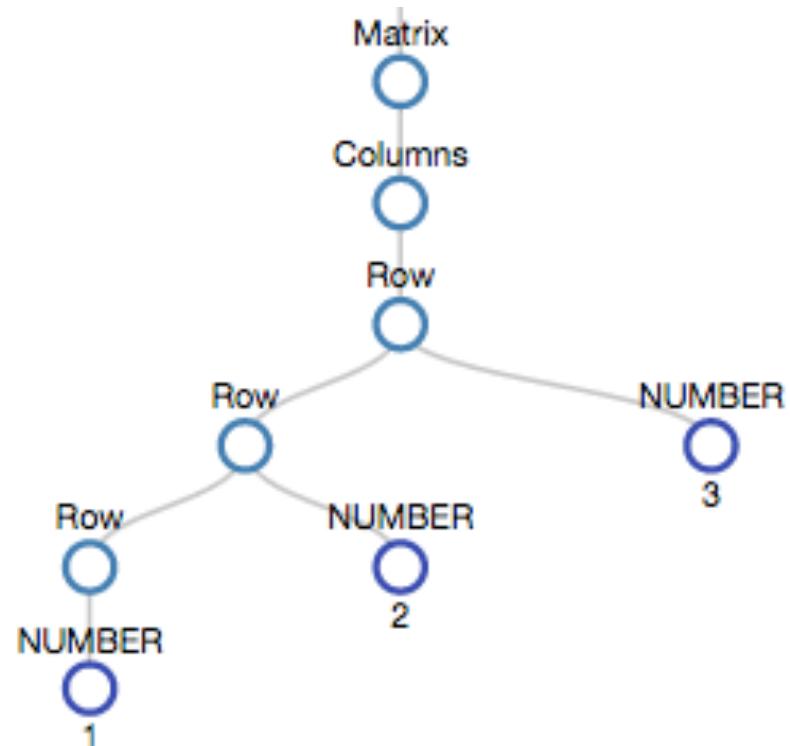
Expression

```
: Matrix
| LITERAL
| LPAR Expression RPAR
| Expression PLUS Expression
| Expression STAR Expression
| Expression SHTRIH
;
```

Матрицы

```
Matrix
  : Columns
  ;
Row
  : NUMBER
  | Row NUMBER
  ;
Columns
  : Row
  | Columns PALKA Row
  ;
```

- Пример 1 2 3

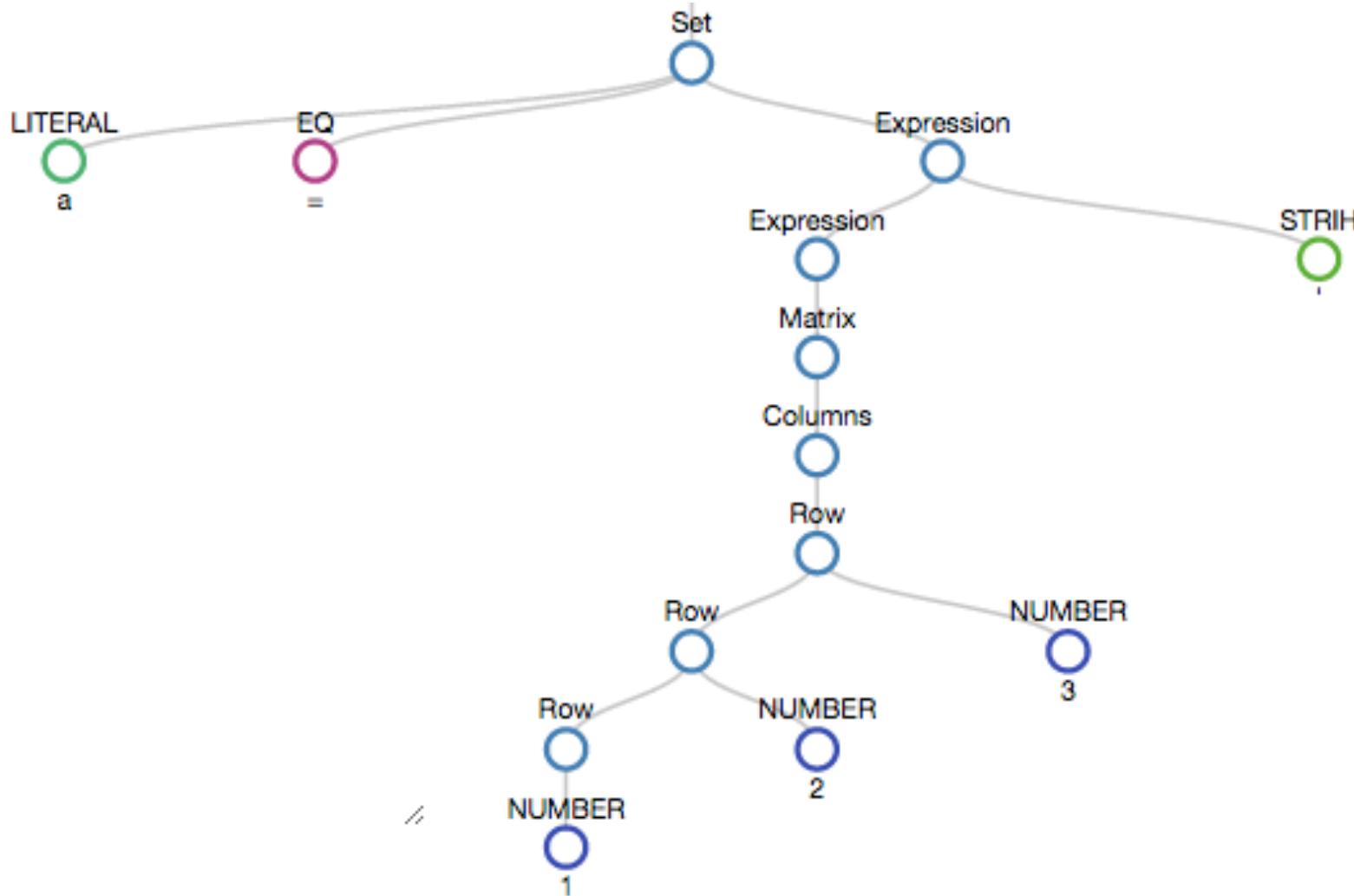


Приоритеты операций

| | |
|--------------|---------------|
| %left PLUS | // 1 + 2 + 3 |
| %left STAR | // 1 + 2 * 3 |
| %right STRIH | // 1 2 * 3 4' |

Кофе-брейк 2: дерево разбора

a = 1 2 3'



Пора уже что-то делать!

Компилируем грамматику

> `jison matrix2.json`

Выполняем проверку грамматики

> `node matrix2.js program.mat`

Если все правильно, то ничего не будет!

Пишем
интерпретатор!



С3Р0
интерпретатор 29

Подготовимся интерпретировать (run-time library MATRIX)

- Операторы
 - MATRIX.print()
- Операции
 - MATRIX.add()
 - MATRIX.multiply()
 - MATRIX.transpose()

Добавляем семантику: подставляем лексемы

Правило

: Expression PLUS Expression

{

// \$\$ - возвращаемая переменная

// \$1, \$2... - лексемы

\$\$ = \$1 + \$3;

}

;

Начнем с печати!

Print

```
: PRINT Expression  
{ MATRIX.print($2); }
```

;

Присваивание переменной

Set

```
: LITERAL EQ Expression
{ MATRIX.mem[$1] = $3; }
```

;

// Когда будем забирать значение,
// просто возьмем:

MATRIX.mem.a

Матрица

Внутреннее представление – массив массивов, например, матрица:

| | | | | | | |
|---|---|---|--|---|---|---|
| 1 | 2 | 3 | | 4 | 5 | 6 |
|---|---|---|--|---|---|---|

Представляется как:

`[[1,2,3],[4,5,6]]`

Строки и столбцы матрицы

```
/* 1 2 3 */
```

```
Row
```

```
: NUMBER
  { $$ = [parseFloat($1)]; }
| Row NUMBER
  { $$ = $1; $$ .push(parseFloat($2)); }
;
```

```
/* 1 2 | 3 4 */
```

```
Columns
```

```
: Row
  { $$ = [$1]; }
| Columns PALKA Row
  { $$ = $1; $$ .push($3); }
;
```

И все - интерпретатор готов!



Кофе-брейк 3: Настоящий запуск!

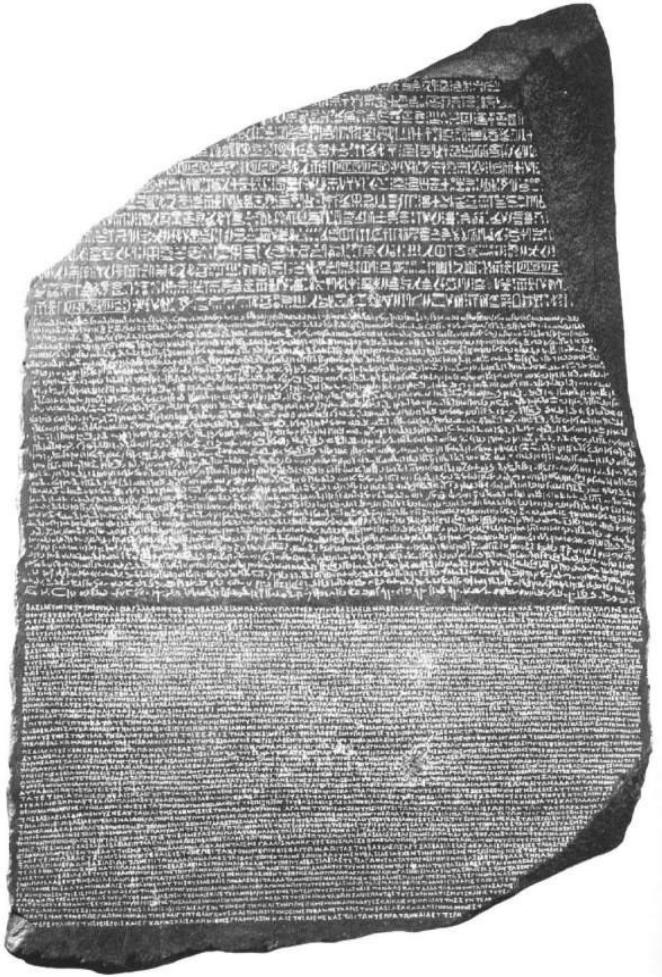
```
> json matrix3.json  
> node matrix3 program.mat
```

Voila!

... наша матрица!



**Жан-Франсуа Шампольон
(транслятор)**



Розеттский камень

Интерпретатор

Возвращает значение:

Expression PLUS Expression
{ \$\$ = MATRIX.add(\$1,\$3); }

А компилятор...

Возвращает строку кода на JavaScript:

Expression PLUS Expression

```
{ $$ = 'MATRIX.add('+$1+', '+$3+'); }
```

Главная функция интерпретатора

Ничего не возвращает

main

: Statement* EOF;

Главная функция компилятора

Возвращает откомпилированную функцию на JavaScript:

```
main
  : Statement* EOF
    { return new Function(
        'var MATRIX = this;'
        +$1.join(';'))
      .bind(MATRIX); }
;
;
```

Праздничный ужин!

Компилируем грамматику

> jison matrix4.json

```
// Запуск из JavaScript
var parser = require('./matrix4.js');
var f = parser.parse('print 1 2+100');

f();
//вернет 101 102
```

В браузере

```
<script src="matrix4.js"></script>
<script>
  var f = parse.parse('print 1 2+100');
  f();
</script>
```

Итак, для интерпретатора/ компилятора нужно:

- Определить лексемы
- Определить правила
- Разработать runtime библиотеку
- Определить семантику (как будут интерпретироваться или компилироваться правила)

Хотите написать свой ES6?

- Парсеры JavaScript
 - Esprima
 - Acorn
 - UglifyJS
 - SpiderMonkey
- Линтеры – используют парсеры
- CoffeeScript – использует Jison



Андрей Гершун agershun@gmail.com
<http://github.com/agershun/matrix>