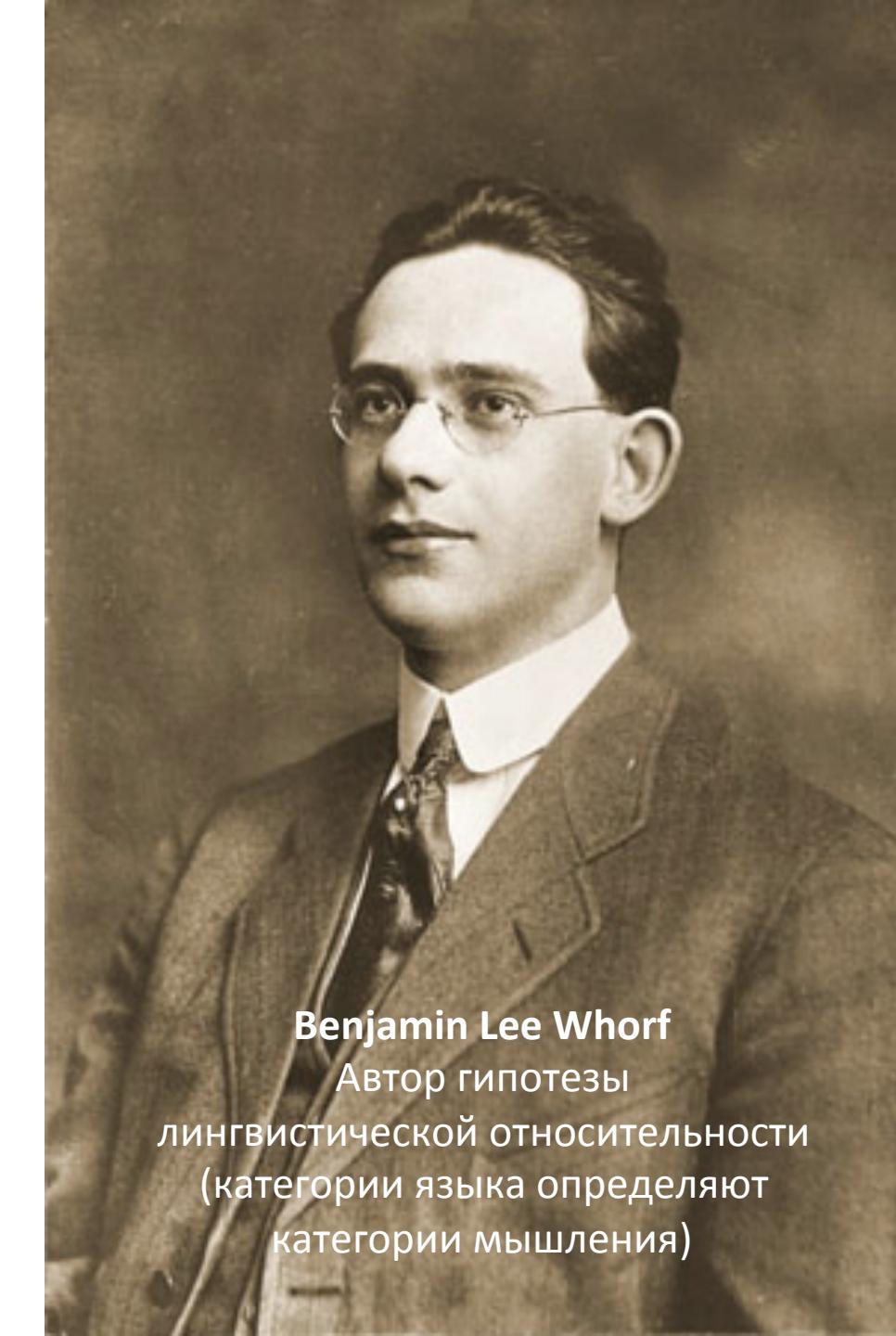


Jison: Как написать компилятор за 15 минут

Андрей Гершун
AlaSQL

7 ноября 2015 года, KharkivJS

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!

Язык 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



Воспользуемся
генератором
парсеров:

- `Json`

Альтернативы:

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

Отладчик грамматик

- Nolan Lawson (PouchDB)

jison-debugger

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 (слова языка)

регулярка

return 'лексема'

1. Отсекаем все ненужное: комментарии и пробелы

// Комментарии

\//.*\$ return

// Пробелы, переносы строк

\s return



William of Ockam

2. Ключевые слова

'PRINT'

return 'PRINT'

'GO TO'

return 'GOTO'

Синонимы

'GOTO'

return 'GOTO'

3. 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'
```

4. Немного магии (служебные лексемы)

```
/* Конец файла */
```

```
<<EOF>>           return 'EOF'
```

```
/* В конце отлавливаем  
нераспознанные символы */
```

```
.                   return 'INVALID'
```

Круг 1:

Что выдает на выходе лексер?

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

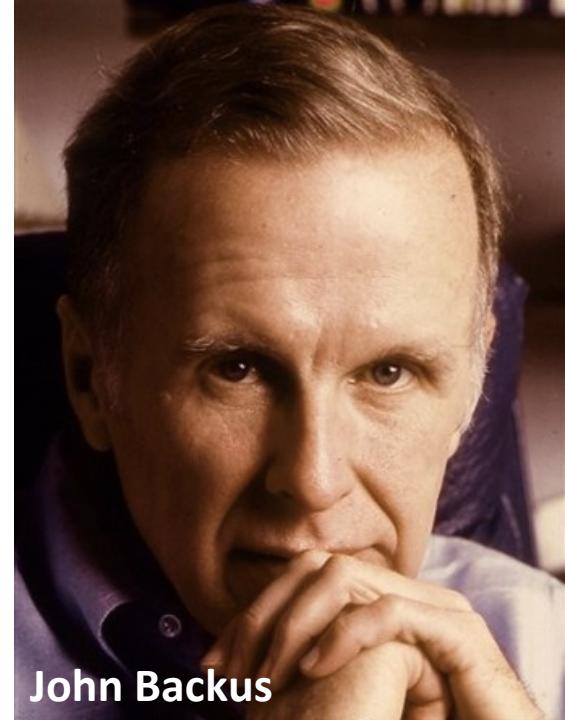
a = 1 2 3 '

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 ;

Set a = 1 2 3
: LITERAL EQ Expression ;

Print PRINT A*b'
: PRINT Expression ;

Выражения

Expression

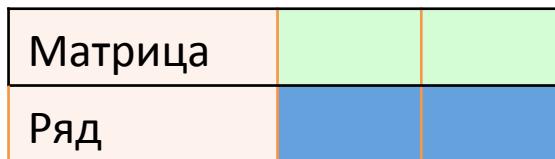
```
: Matrix                                // 1 2 3
| LITERAL                               // a
| LPAR Expression RPAR                // (1+2)
| Expression PLUS Expression          // 1+2
| Expression STAR Expression          // 1*2
| Expression SHTRIH                  // a'
;
;
```

Матрицы

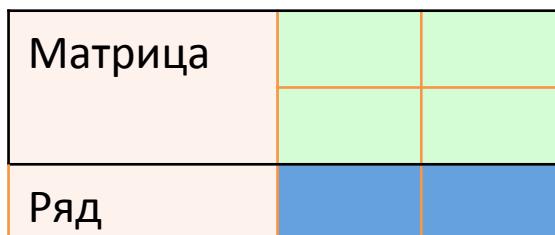
(рекурсивное определение)



- это матрица



- это тоже матрица

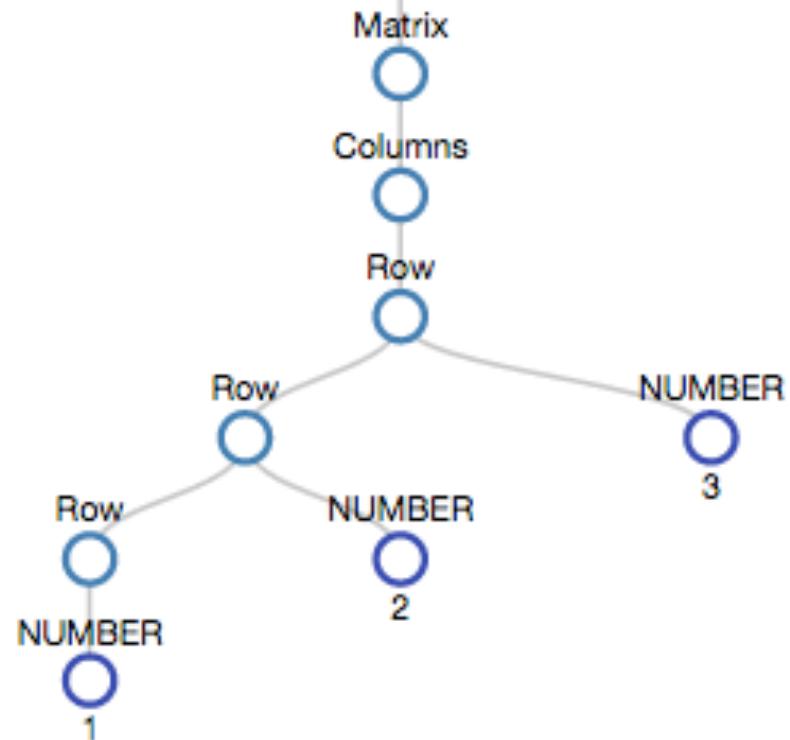


- и это матрица

Матрицы (рекурсивное определение)

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

Пример 1 2 3

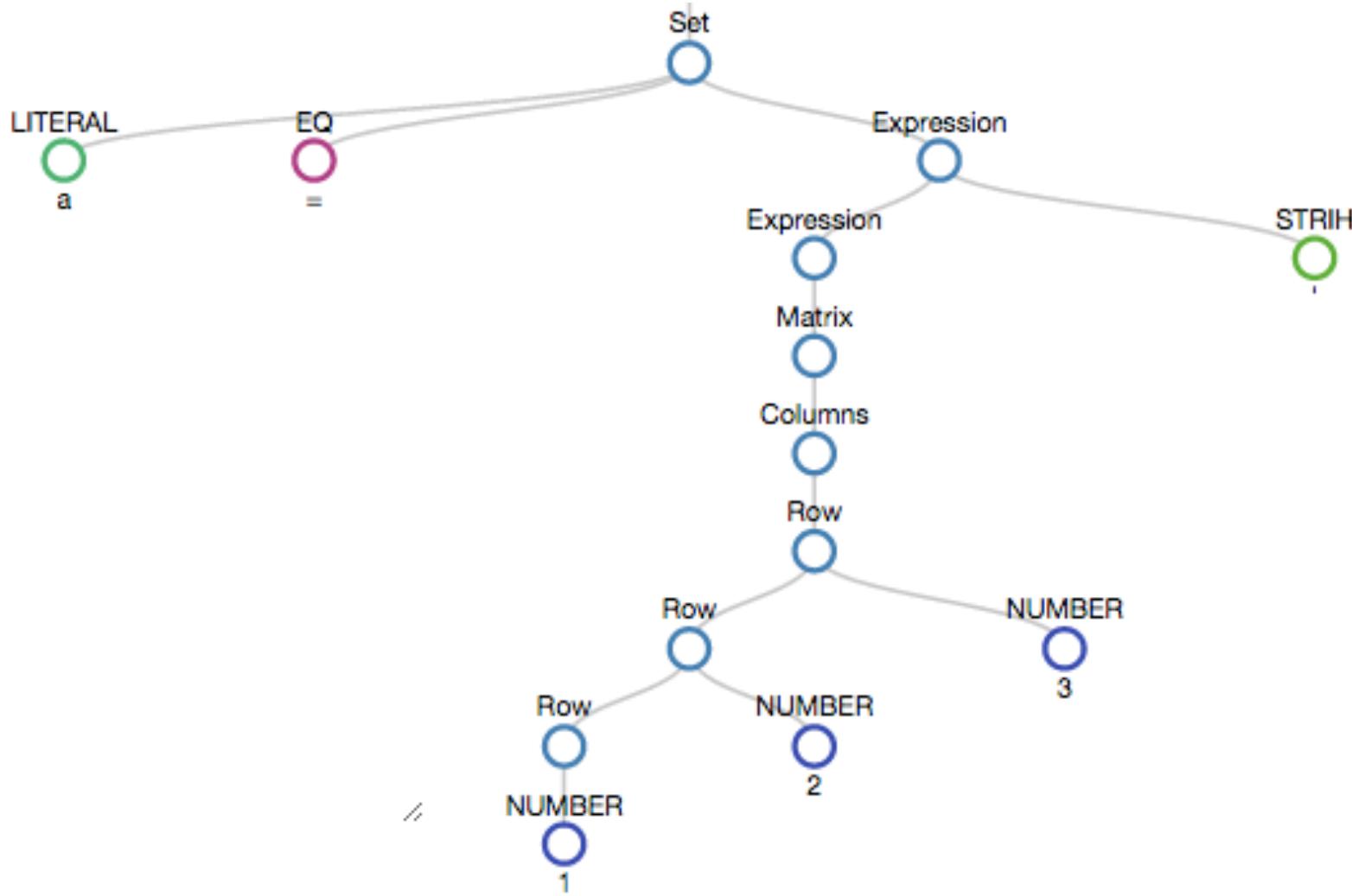


Приоритеты операций

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

Круг 2: дерево разбора

a = 1 2 3'



Пора уже что-то делать!

Компилируем грамматику

> `jison matrix2.json`

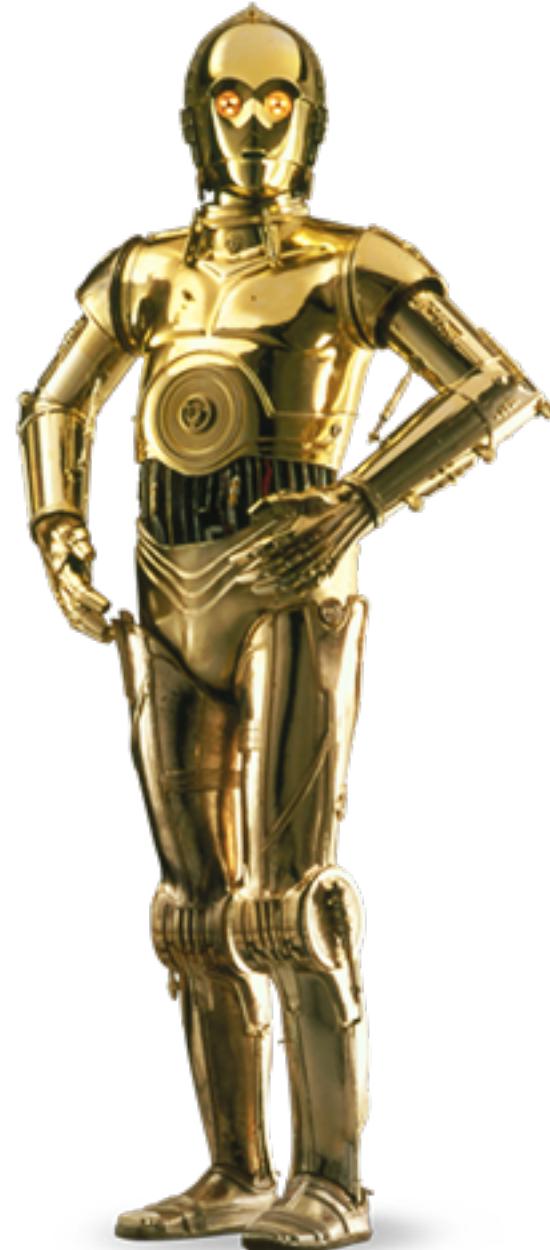
Выполняем проверку грамматики

> `node matrix2.js program.mat`

Если все правильно, то опять ничего не будет!

Лексер готов.

Пишем
интерпретатор!



СЗРО
интерпретатор 31

Подготовимся: Run-Time Library MATRIX

// Будущая библиотека

```
var MATRIX = {};
```

// Оператор PRINT a

```
MATRIX.print = function(a){  
    console.log(a)  
}
```

Подготовимся: Run-Time Library MATRIX

- Операторы
 - `MATRIX.print()`
- Операции
 - `MATRIX.add()` // $a+b$
 - `MATRIX.multiply()` // a^2
 - `MATRIX.transpose()` // a'

Вставки на JavaScript

: Правило из лексем

{

Вставка на JavaScript

}

;

Добавляем семантику: подставляем лексемы

```
// Правило сложения a+b
: Expression PLUS Expression
{
    // $$ - возвращаемый результат
    // $1,$2,$3... - лексемы
    $$ = $1 + $3;
}
```

Начнем с печати!

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

Запасемся местом
под переменные

```
MATRIX.mem = {};
```

Присваивание переменной

```
// Оператор присваивания b = a*2
: LITERAL EQ Expression
{ MATRIX.mem[$1] = $3; }

;
// Когда будем забирать значение,
// просто возьмем:
: Literal
{
    $$ = MATRIX.mem[$1];
}
;
```

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 (опять рекурсивное определение!)

```
/* 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!



Круг 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
```

Запуск в браузере

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

Итак, для интерпретатора/ компилятора нужно:

1. Определить лексемы
2. Определить правила
3. Разработать runtime библиотеку
4. Определить семантику (как будут
интерпретироваться или компилироваться
правила)

Хотите написать свой ES6?

- Парсеры JavaScript
 - Esprima
 - Acorn
 - UglifyJS
 - SpiderMonkey
- Babel, линтеры и другие – используют парсеры
- Кстати:
 - CoffeeScript – использует Jison



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