OMetaJS - шаблонизация

конвертируем BEMHTML в BH

BEMHTML

```
block 'search' {

    tag: 'form'

    attrs: {
        action: this.ctx.action || '/',
        role: 'search'
    }

    content: {
        elem: 'table',
        content: {
            elem: 'row',
            content: this.ctx.content
        }
    }

    elem table, tag: 'table'

    elem row {
        tag: 'tr',
        bem: false
    }

    elem cell, tag: 'td'

}
```

BH

```
module.exports = function (bh) {
    bh.match('search', function (ctx, json) {
        ctx.tag('form', true);
        ctx.attrs(ctx.extend({
            'action': json.action || '/',
            'role': 'search'
        }, ctx.attrs()));
        ctx.content({
            'elem': 'table',
            'content': {
                'elem': 'row',
                'content': json.content
            }
        }, true);
    });
    bh.match('search__table', function (ctx, json) {
        ctx.tag('table', true);
    });
    bh.match('search__row', function (ctx, json) {
        ctx.tag('tr', true);
        ctx.bem(false, true);
    });
    bh.match('search__cell', function (ctx, json) {
        ctx.tag('td', true);
    });
};
```

BEMHTML

```
block 'search' {

    tag: 'form'

    attrs: {
        action: this.ctx.action || '/',
        role: 'search'
    }

    content: {
        elem: 'table',
        content: {
            elem: 'row',
            content: this.ctx.content
        }
    }

    elem table, tag: 'table'

    elem row {
        tag: 'tr',
        bem: false
    }

    elem cell, tag: 'td'

}
```

- выразительность
- полнота
- свой синтаксис
- XJST / OMetaJS

BH

- скорость
- JS-синтаксис
- ограничения, продиктованные практикой

```
module.exports = function (bh) {
    bh.match('search', function (ctx, json) {
        ctx.tag('form', true);
        ctx.attrs(ctx.extend({
            'action': json.action || '/',
            'role': 'search'
        }, ctx.attrs()));
        ctx.content({
            'elem': 'table',
            'content': {
                'elem': 'row',
                'content': json.content
            }
        }, true);
    });
    bh.match('search__table', function (ctx, json) {
        ctx.tag('table', true);
    });
    bh.match('search__row', function (ctx, json) {
        ctx.tag('tr', true);
        ctx.bem(false, true);
    });
    bh.match('search__cell', function (ctx, json) {
        ctx.tag('td', true);
    });
};
```

Разные семантики!

Компиляция

`tokenize < input` | `parse` | `serialise`

Lexical analysis

From Wikipedia, the free encyclopedia

In computer science, **lexical analysis** is the process of converting a sequence of characters into a sequence of tokens, i.e. meaningful character strings. A program or function that performs lexical analysis is called a **lexical analyzer**, **lexer**, **tokenizer**,[1] or **scanner**, though "scanner" is also used for the first stage of a lexer. A lexer is generally combined with a parser, which together analyze the syntax of computer languages, such as in compilers for program

Strictly speaking, a lexer is itself a kind of pa
(token structure), which is processed by the
regular language, whose alphabet consists
language, whose alphabet consists of the to
combined with the parser in scannerless par

LALR parser

From Wikipedia, the free encyclopedia

In computer science, an **LALR parser**[a] or **Look-Ahead LR parser** is a simplified version of a canonical LR parser, to parse (separate and analyze) a text according to a set of production rules specified by a formal grammar for a computer language. ("LR" means left-to-right, rightmost derivation.)

The LALR parser was invented
the practical difficulties at that ti
the LR(0) parser, while requiring
makes the LALR parser a memo
LR(1) languages that are not LA
languages,[2] including Java,[3] t

The original dissertation gave n
generation were published in 19
parsers.[5] LALR parsers can be
automatically generated code m

WIKIPEDIA
The Free Encyclopedia

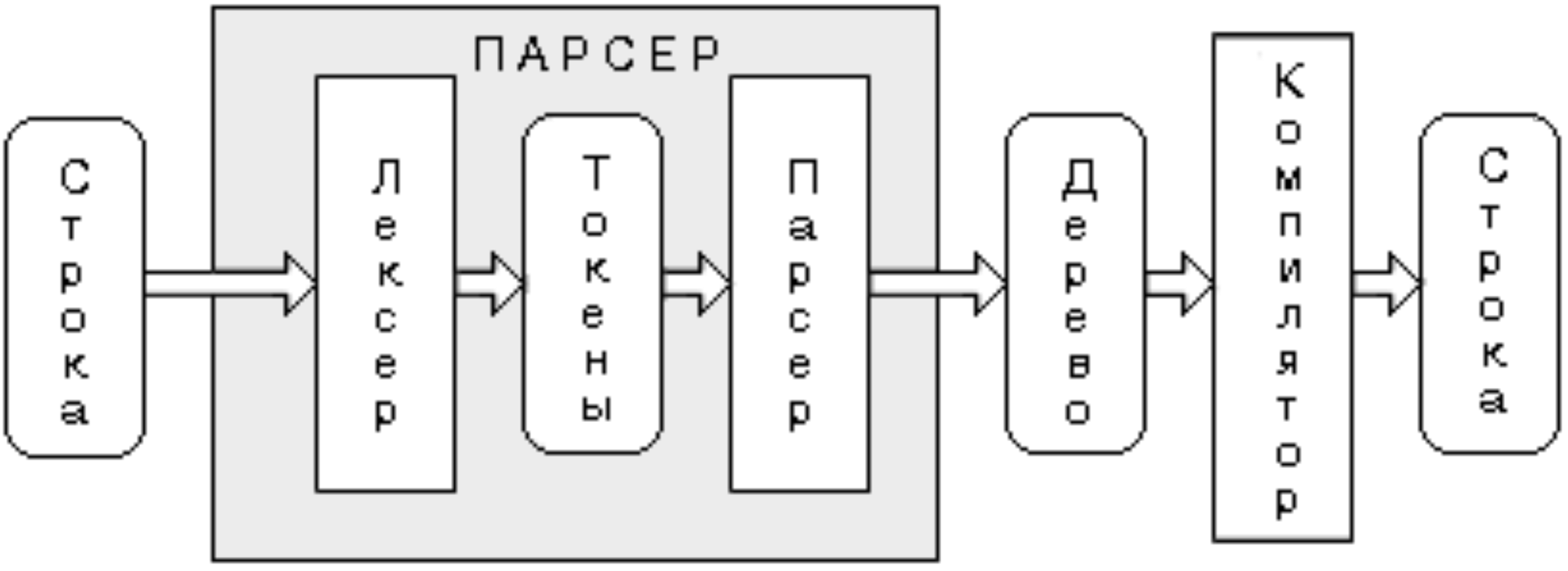
Major tasks in code generation [edit]

In addition to the basic conversion from an intermediate representation into a linear sequ
tries to optimize the generated code in some way.

Tasks which are typically part of a sophisticated compiler's "code generation" phase inclu

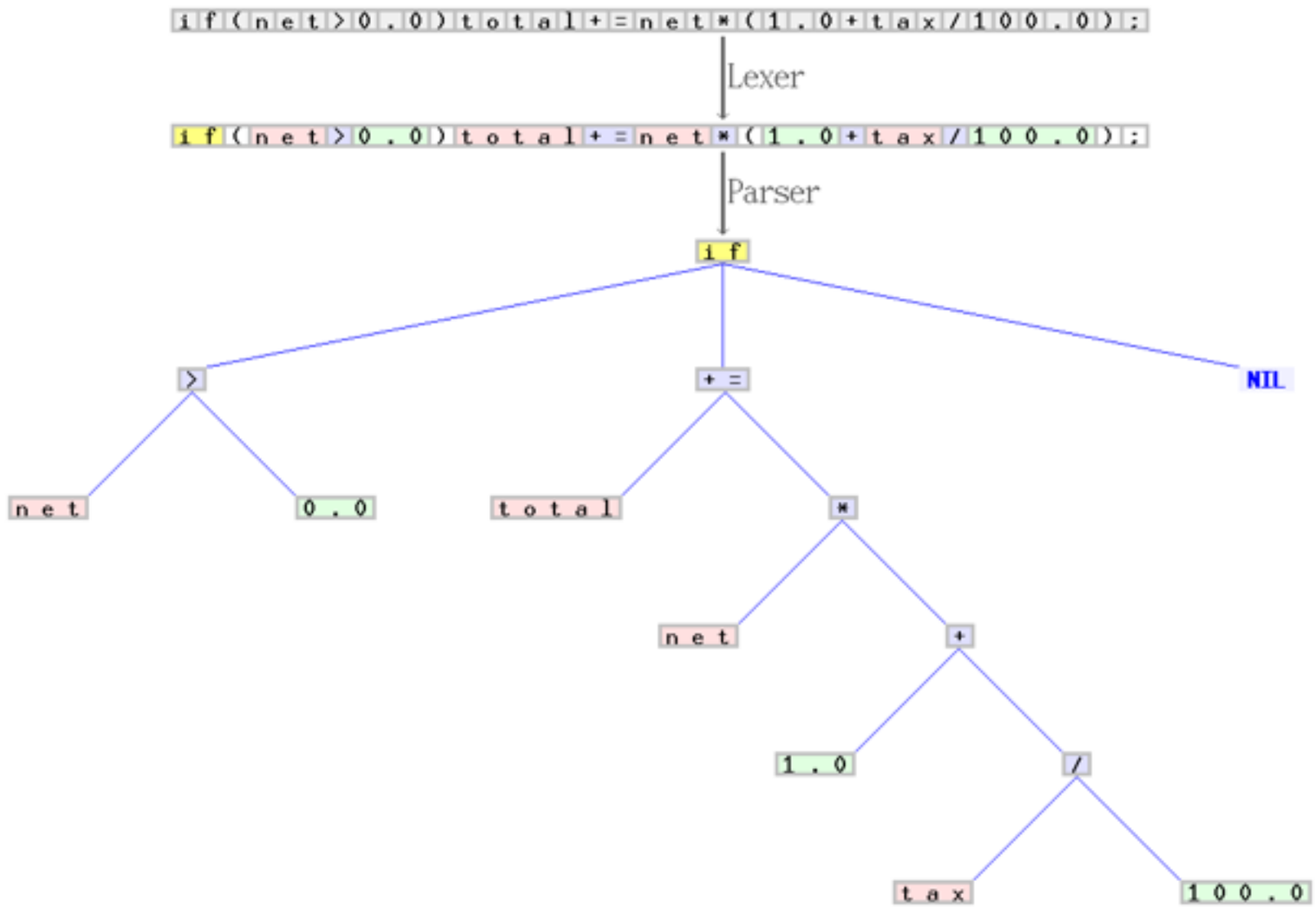
- Instruction selection: which instructions to use.
- Instruction scheduling: in which order to put those instructions. Scheduling is a speed
  machines.
- Register allocation: the allocation of variables to processor registers[1]
- Debug data generation if required so the code can be debugged.

Компиляция


Строка
ПАРСЕР
Лексер
Токены
Парсер
Дерево
Компилятор
Строка

Компиляция

`tokenize < input` | `parse`



Говорим грамматика

```
letter = "A" | "B" | "C" | "D" | "E" | "F" | "G"
       | "H" | "I" | "J" | "K" | "L" | "M" | "N"
       | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
       | "V" | "W" | "X" | "Y" | "Z" ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
symbol = "[" | "]" | "{" | "}" | "(" | ")" | "<" | ">"
       | "'" | '"' | "=" | "|" | "." | "," | ";" ;
character = letter | digit | symbol | "_" ;

identifier = letter , { letter | digit | "_" } ;
terminal = "'" , character , { character } , "'"
         | '"' , character , { character } , '"' ;

lhs = identifier ;
rhs = identifier
     | terminal
     | "[" , rhs , "]"
     | "{" , rhs , "}"
     | "(" , rhs , ")"
     | rhs , "|" , rhs
     | rhs , "," , rhs ;

rule = lhs , "=" , rhs , ";" ;
grammar = { rule } ;
```

Грамматика в OMetaJS

```
var ometajs = require('ometajs');

ometa PARSER {
    dig = /[0-9]/i,
    num = dig+:ds                -> ['num', parseInt(ds.join(''))],
    fac = fac:x '*' num:y -> ['mul', x, y]
        | fac:x '/' num:y -> ['div', x, y]
        | num,
    exp = exp:x '+' fac:y -> ['add', x, y]
        | exp:x '-' fac:y -> ['sub', x, y]
        | fac
}
```

OMeta

Знаю / RegExp / - к OMете готов

test-drive

Грамматика для JS

OMetaJS

- 428 строк декларативного кода
- Парсим
- Обходим дерево
- Сериализуем

Грамматика для JS

Esprima

- 3757 строк императивного код
- Парсим
- Обходим дерево
- Генерируем

OMeta

кратко

- OO язык для pattern matching
- Один инструмент для
  - лексического анализа
  - парсинга
  - траверса деревьев

Known for	Dynabook object-oriented programming Smalltalk graphical user interface windows
Notable awards	ACM Turing Award, Kyoto Prize, Charles Stark Draper Prize

OMeta

матчит

- Поток символов

- Структурные данные (массивы)

OMeta

умеет и позволяет

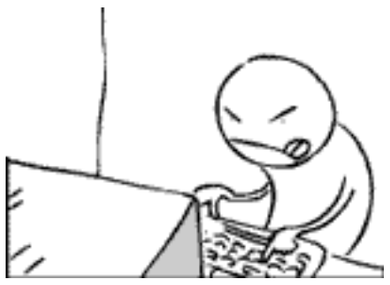
- Писать декларативный код на BNF-похожем синтаксисе
- Пользоваться всеми возможностями языка носителя
- Наследовать и расширять существующие грамматики
- Вызывать правила из внешних грамматик
- Вызывать правила с аргументами
- На эти аргументы матчиться
- Вызывать правила динамически, получая имя в рантайм (apply)
- Писать правила с "левой рекурсией"

Что могли бы использовать?

- Вариации на тему *Lex / Yacc (flex, bison)*
- Вариации на тему *PEGs (Packrat parsers)*
- Комбинаторы aka *Parser Combinators*
- Деривативы aka *Parsing with derivatives*

Наверняка, много чего еще или …

Выкатить свое кастомное барахло



WTF

- синтаксис
- инструментарий
- сообщения об ошибках
- правила с левой рекурсией

Что почитать?

- bemhtml-source-convert
  https://github.com/vkz/bemhtml-source-convert

- OMetaJS
  https://github.com/veged/ometa-js

- Ссылки на все связанное с OMeta на одной странице
  http://tinlizzie.org/ometa/

- OMeta playground
  http://www.tinlizzie.org/ometa-js/#Sample_Project

- [paper] OMeta an Object-Oriented Language for Pattern Matching, 2007 by Alessandro Warth, Ian Piumatra

- [paper] OMeta Experimenting with Programming Languages, 2009 by Alessandro Warth

- [book] Parsing Techniques by Grune and Jacobs

Спасибо

Вопросы?