Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie,
WEAIiIB, Informatyka III rok, 2013/2014

Dokumentacja do projektu
z przedmiotu
Języki Formalne i Kompilatory

Translator RedScript do JavaScript

Autorzy:

Agata Mysiak

Kamil Jamróz

Spis treści

1 Wstęp ... 3

1.1 Cel projektu ... 3

1.2 Język pseudokodu ... 3

1.3 Wykorzystywane technologie ... 3

2 Realizacja projektu ... 5

2.1 Implementacja ... 5

2.1.1 Literały ... 5

2.1.2 Komentarze ... 5

2.1.3 Znaki końca linii ... 5

2.1.4 Słowa kluczowe ... 6

2.1.5 Separatory ... 6

2.1.6 Operatory ... 6

2.1.7 Struktury* ... 6

2.2 Kolejne etapy konwersji ... 10

2.3 Lexer ... 11

2.4 Gramatyka ... 13

2.5 CodeBuilder ... 18

3 Użycie ... 18

3.1 Sposoby użycia parsera ... 18

3.1.1 Z linii poleceń ... 18

3.1.2 Z innego modułu ... 18

3.1.3 Przez stronę internetową ... 18

3.2 Plik wejściowy ... 19

3.3 Plik wyjściowy ... 19

4 Bibliografia ... 19

1 Wstęp

1.1 Cel projektu

Język JavaScript cechuje się uniwersalnością oraz prostotą użycia. Jego zdolność do wykonywania obliczeń po stronie użytkownika pozwala na odciążenie zarówno serwera. Cechy te sprawiają, iż język cieszy się od lat niesłabnącą popularnością. Mimo to, korzystając na co dzień z JS, możemy znaleźć wiele jego aspektów, które można usprawnić. Zdarza się nam również odczuwać brak pewnych rozwiązań.Aby, nie rezygnując z niewątpliwych zalet języka, usprawnić swoją pracę, dobrym pomysłem jest wykorzystanie parsera, który gotowy do użycia kod JavaScript uzyska z pseudokodu.

1.2 Język pseudokodu

Jako język, który będzie wejściem dla translatora, wybraliśmy RedScript. Jego składnia oparta jest na języku Ruby. Pozwala to na pisanie prostszego kodu, łatwego do późniejszego odczytania, dzięki czemu wprowadzanie w nim zmian również stwarza mniej problemów. Dostępne są także przydatne, chociaż nieobecne w JavaScript funkcjonalności, takie, jak dziedziczenie klas.

1.3 Wykorzystywane technologie

Pierwszym krokiem przy wyborze technologii budowy parsera było podjęcie decyzji, czy chcemy zbudować parser ręcznie czy wygenerować go automatycznie korzystając z powszechnie dostępnych narzędzi.

Zaletami ręcznej metody są niewątpliwie:

- Bardziej przejrzysty wynikowy kod parsera
- Możliwość implementacji bardziej szczegółowego debuggera
- Brak zależności od zewnętrznych programów - znaczenie ma to szczególnie w przypadku, gdy błąd zewnętrznego programu uniemożliwi dokończenie projektu w oczekiwanej formie

Z kolei zalety użycia zewnętrznych narzędzi to:

- Krótszy czas implementacji lub/oraz modyfikacji zasad działania parsera
- Zgodność działania parsera z powszechnie przyjętymi metodykami (LALR, LR, GLR, SLR etc.)
- W większości przypadków wydajniejsze działanie w porównaniu do ręcznie tworzonych parserów

Jest to nasz pierwszy projekt tego typu, więc bardziej przekonały nas zalety wykorzystania gotowych narzędzi. Uważamy, że w ten sposób lepiej poznamy podstawy pracy z parserami oraz szybciej uzyskamy oczekiwany efekt pracy.

Do stworzenia parsera zdecydowaliśmy się skorzystać z duetu:

- Flex - generator analizatorów leksykalnych, nowsza wersja Lexa
- Bison - generator parserów gramatyki, nowsza wersja Yacca

Do użycia wyżej wymienionych programów przekonała nas:

- Ogromna popularność obu generatorów w środowisku, łącznie z dużą ilością materiałów dydaktycznych, przykładów i/oraz rozwiązań typowych problemów implementacyjnych przy tworzeniu parsera własnego języka
- Łatwość przyswojenia podstaw i przystępna krzywa nauczania bardziej zaawansowanych struktur
- Intuicyjny sposób działania i budowy własnego kodu oparty o metodologię LALR, LR
- Powszechna dostępność - licencja GNU GPL

Z dostępnych implementacji zdecydowaliśmy się wykorzystać program Jison, który jest kopią Bisona pod JavaScript oraz posiada własny analizator leksykalny wzorowany na Flex.
Wykorzystanie tego konkretnego programu względem innych daje nam tę zaletę, że możemy korzystać z funkcji napisanych w JavaScript, co przyśpiesza i ułatwia realizację projektu, a także gwarantuje pełną kompatybilność funkcji pomocniczych z kodem wynikowym.

2 Realizacja projektu

2.1 Implementacja

Nasz projekt wspiera poniższe elementy języka JavaScript.

2.1.1 Literały

Jednostki leksykalne oznaczające pewną stałą wartość.

2.1.1.1 Integer

- reprezentacja dziesiętna
  - ciąg cyfr rozpoczynający się cyfrą inną niż 0 lub samo 0
- reprezentacja szestnastkowa
  - poprzedzony 0X (lub 0x) ciąg cyfr oraz liter z przedziału A-F (a-f)
- reprezentacja ósemkowa
  - rozpoczynający się cyfrą 0 ciąg cyfr z przedziału 0-7

2.1.1.2 FloatingPoint

- ciąg cyfr, opcjonalnie z kropką oddzielającą część całkowitą od ułamkowej lub notacją naukową

2.1.1.3 Boolean

- true
- false

2.1.1.4 String

- ciąg zawartych w podwójnym cudzysłowiezera lub więcej znaków różnych od znaku " chyba, że jest on poprzedzony znakiem /
- ciąg zawartych w pojedynczym cudzysłowie zera lub więcej znaków różnych od znaku ' chyba, że jest on poprzedzony znakiem /

2.1.1.5 Null

Brak wartości referencji reprezentowany przez literał:

null

2.1.2 Komentarze

Komentarz mogący rozciągać się na wiele linii kodu:

/* *komentarz* */

Komentarz umieszczany na końcu linii:

// *komentarz*

2.1.3 Znaki końca linii

LF

CR

CR LF

2.1.4 Słowa kluczowe

abstract	**boolean**	**break**	byte	**case**	**catch**
char	class	const	**continue**	**default**	**delete**
do	double	**else**	extends	**false**	final
finally	float	**for**	**function**	goto	**if**
implements	import	**in**	**instanceof**	int	interface
long	native	**new**	**null**	package	private
protected	public	**return**	short	static	super
switch	synchronized	**this**	**throw**	throws	transient
true	**try**	**var**	**void**	**while**	**with**

(wytłuszczonym drukiem wskazane obecnie używane w języku JavaScript; reszta z nich to słowa kluczowe zarezerwowane na wypadek przyszłego użycia)

2.1.5 Separatory

(	)	{	}	[	]	,	.

2.1.6 Operatory

=	>	<	!	~
?	:	==	<=	>=
!=	&&	||	++	--
+	-	*	/	&
|	^	%	<<	>>
>>>	+=	-=	*=	/=
&=	|=	^=	%=	<<=
>>=	>>>=			

2.1.7 Struktury*

Javascript	**RedScript**
Pętle	

`while foo < 200 {`
`  console.log( "This is a loop that never ends");`
`}`
	`while foo < 200`
	`  puts "This is a loop that never ends"`
	`end`

`while (!(i == 5)) {`
`  console.log(i);`
`  I += 1;`
`}`
	`until i ==5`
	`  puts i`
	`  I += 1`
	`end`

`for (vari=0; i< 5; i++) {`
`  console.log(i);`
`}`
	`for i in 0..5`
	`  puts i`
	`end`

`for (vari=0; i<= 5; i++) {`
`  console.log(i);`
`}`
	`# do 5 włącznie`
	`for i in 0…5`
	`puts i`
	`end`

Iteracja po tablicy	

`var basket = ['lemon', 'pear', 'apple'];`
 	
`for (var i1=0, len1=basket.length; i1 < len1; i1++) { var fruit = basket[i1];`
`  console.log(fruit);`
`}`
	`basket = ['lemon', 'pear', 'apple']`
	
	`for fruit of basket`
	`putsfruit`
	`end`

Iteracja po obiektach	

for (var key in obj) {
  alert(key);
}
	for key in obj
	  alert(key)
	end

for (var key in obj) { varval = obj[key];
  console.log('My key is: ' + key);
  console.log('My value is: ' + val);
}
	for key,val in obj
	  puts 'My key is: #{key}'
	puts 'My value is: #{val}'
	end

Funkcje	

var say = function(msg) {
  console.log("Message: " + msg);
};
	func say(msg)
	  puts "Message: #{msg}"
	end

var sayHello = function() {
  console.log(“Hello”);
};
	# bezparametrów
	funcsayHello
	  puts "Hello"
	end

//anonimowa funkcja
function() {
  // do stuff
}
	# anonimowa funkcja
	func
	  # do stuff
	end

function(a,b) {
  // do stuff
}
	func(a,b)
	  # do stuff
	end

if, else, elseif	

if (foo == 10) {
  bar("do stuff");
} else if (foo == 20) {
  bar("do stuff");
} else {
  bar("do stuff");
} end
	if foo == 10
	  bar("do stuff")
	else if foo == 20
	  bar("do stuff")
	else
	  bar("do stuff")
	end

if (err) { throw err; }	throw err if err
Switch	

switch (fruit) {
case "apple":
console.log("it's an apple");
break;
  case "banana":
console.log("banana"); break;
case "orange":
console.log("it's an orange");
break;
default:
console.log("???");
}
	switch fruit
	when "apple"
	  puts "it's an apple"
	  break
	when "banana" then puts("banana")
	when "orange"
	  puts "it's an orange"
	  break
	default
	puts "???"
	end

Try&catch	

try {
foo();
}
catch(err) {
alert('An error has occurred: '+err.message);
}
	try
	  foo()
	catch
	alert('An error has occurred: '+err.message)
	end

try {
foo();
}
catch(err) {
  alert('An error has occurred: '+err.message);
}
finally {
  alert('I wonder if foo worked?');
}
	try
	foo()
	catch
	  alert('An error has occurred: '+err.message)
	finally
	alert('I wonder if foo worked?')
	end

Definicja modułu	

define(function() {
 	
var foo = function(x) {
console.log(x);
}
 	
return foo;
  });
	define module
	
	func foo(x)
	  puts x
	end
	
	export foo

define(['jquery',
'./views/widget'],
function($, Widget) {
 	
var options = {
moonRoof: true,
seats: 5
}
 	
vargetCost = 16899;
var wheels = 4;
 	
return {
getCost : getCost,
hasMoonRoof : options.moonRoof,
getWheels : wheels
}
});
	define module
	require 'jquery' as $
	require './views/widget' as Widget
	
	options = {
	moonRoof: true,
	  seats: 5
	}
	
	getCost = 16899
	wheels = 4
	
	export
	getCost
	hasMoonRoof from options.moonRoof
	getWheels from wheels
	end

Deklaracja jako private	

var foo = 200;
 	
(function(){
var foo = 10;
})();
 	
alert(foo); // 200
	foo = 200
	
	private
	var foo = 10
	end
	
	alert(foo) # 200

Alias @ dla this	

function Foo(name, age) {
this.name = name
this.age = age
}
	functionFoo(name, age) {
	  @name = name
	  @age = age
	}

Tworzenie obiektów i metod	

var auto = {
wheels: 4,
engine: true,
 	
honk: function() {
console.log("hooooonk");
},
 	
sayHi: function(msg) {
   console.log(msg);
}
}
 	
// definicja metody na zewnątrz
obiektu
auto.add = function(x,y) {
   return x + this.wheels;
};
 	
// metoda dołączona do prototypu
Car.prototype.sub = function(x, y) {
return x - y;
};
	object auto
	  wheels: 4,
	  engine: true,
	
	def honk
	    puts "hoooonk"
	  end,
	
	defsayHi(msg)
	    puts msg
	  end
	end
	
	# definicja metody na zewnątrz
	obiektu
	def auto.add(x, y)
	return x + @wheels
	end
	
	# dołączenie metody do prototypu
	def Car >> sub(x,y)
	  return x - y
	end

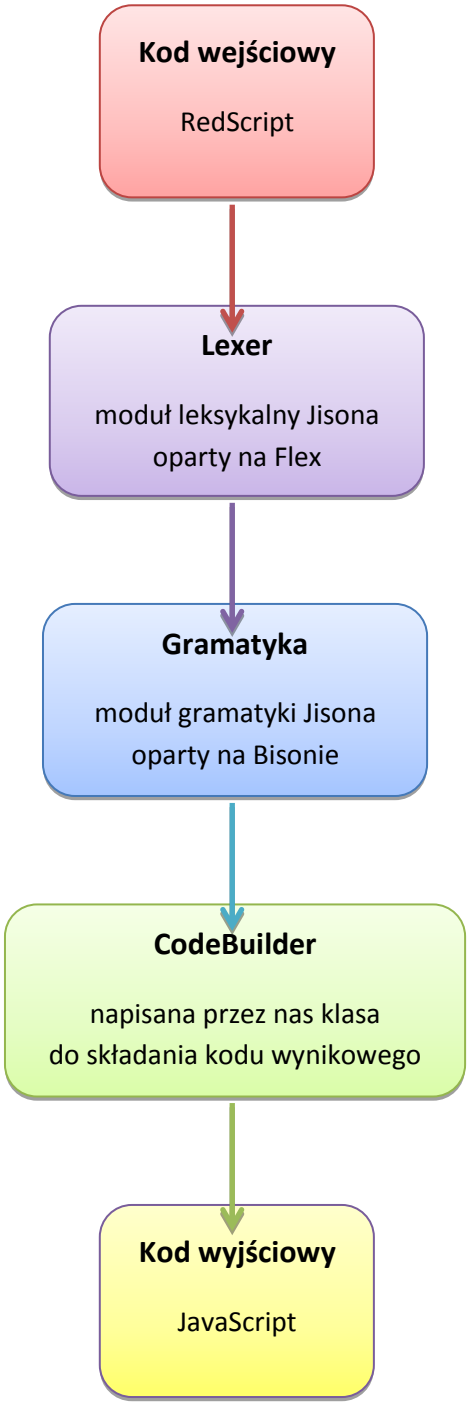
Klasyczne dziedziczenie	
var Animal = Class.extend({	class Animal

```
init: function(name) {
this.name = name;
},

sayHi: function() {
console.log("Hello");
}
});

var Duck = Animal.extend({
init: function(name) {
this._super(name);
alert(this.name + " is alive!")
},

sayHi: function() {
this._super();
console.log("Quack");
}
});

var duck = new Duck('Darick');
duck.sayHi();
```

```
definit(name)
    @name = name
  end,

defsayHi
    puts "Hello"
  end
end)

class Duck < Animal
definit
    super name
    alert("#{@name} is alive!")
  end,

defsayHi
    super
    puts "Quack"
  end
end)

duck = new Duck('Darick')
duck.sayHi()
```

(*) Szczegółowy opis składni znajduje się w dokumentacji języka RedScript.

2.2 Kolejne etapy konwersji



2.3 Lexer

```
"//"(.*)                    /* skip comments */
\s+                         /* skip whitespaces */
"NEW"                       { return 'NEW'; }
"FUNC"                      { return 'FUNCTION'; }
"END"                       { return 'END'; }
"ELSE IF"                   { return 'ELSIF'; }
"ELSEIF"                    { return 'ELSIF'; }
"IF"                        { return 'IF'; }
"ELSE"                      { return 'ELSE'; }
"DO"                        { return 'DO'; }
"WHILE"                     { return 'WHILE'; }
"FOR"                       { return 'FOR'; }
"UNTIL"                     { return 'UNTIL'; }
"IN"                        { return 'IN'; }
"OF"                        { return 'OF'; }
"CONTINUE"                  { return 'CONTINUE'; }
"BREAK"                     { return 'BREAK'; }
"RETURN"                    { return 'RETURN'; }
"WHEN"                      { return 'WHEN'; }
"SWITCH"                    { return 'SWITCH'; }
"DEFAULT"                   { return 'DEFAULT'; }
"THEN"                      { return 'THEN'; }
"THROW"                     { return 'THROW'; }
"TRY"                       { return 'TRY'; }
"CATCH"                     { return 'CATCH'; }
"FINALLY"                   { return 'FINALLY'; }
"DEBUGGER"                  { return 'DEBUGGER'; }
"PRIVATE"                   { return 'PRIVATE'; }
"CLASS"                     { return 'CLASS'; }
"DEF"                       { return 'DEFINE'; }
"EXTENDS"                   { return 'EXTENDS'; }
"OBJECT"                    { return 'OBJECT'; }
"CLONES"                    { return 'CLONES'; }
"CONST"                     { return 'CONST'; }
"VOID"                      { return 'VOID'; }
"DELETE"                    { return 'DELETE'; }
"THIS"                      { return 'THISTOKEN'; }
"TRUE"                      { return 'TRUETOKEN'; }
"FALSE"                     { return 'FALSETOKEN'; }
"NULL"                      { return 'NULLTOKEN'; }
"==="                       { return 'STREQ'; }
"!=="                       { return 'STRNEQ'; }
"=="                        { return 'EQ'; }
"!="                        { return 'NEQ'; }
"&&"                        { return 'AND'; }
"||"                        { return 'OR'; }
"+="                        { return 'PLUSEQUAL'; }
"-="                        { return 'MINUSEQUAL'; }
"<="                        { return 'LEQ'; }
">="                        { return 'GEQ'; }
"/="                        { return 'DIVEQUAL'; }
"<<="                       { return 'LSHIFTEQUAL'; }
">>="                       { return 'RSHIFTEQUAL'; }
">>>="                      { return 'URSHIFTEQUAL'; }
"&="                        { return 'ANDEQUAL'; }
"^="                        { return 'XOREQUAL'; }
"|="                        { return 'OREQUAL'; }
"%="                        { return 'MODEQUAL'; }
```

```
"++"                          { return 'INC'; }
"--"                          { return 'DEC'; }
"{"                           { return 'OPENBRACE'; }
"}"                           { return 'CLOSEBRACE'; }
"["                           { return '['; }
"]"                           { return ']'; }
"("                           { return '('; }
")"                           { return ')'; }
","                           { return ','; }
"."                           { return '.'; }
":"                           { return ':'; }
";"                           { return ';'; }
"!"                           { return '!'; }
"?"                           { return '?'; }
"&"                           { return '&'; }
"|"                           { return '|'; }
"^"                           { return '^'; }
"="                           { return '='; }
"~"                           { return '~'; }
"+"                           { return '+'; }
"-"                           { return '-'; }
"/"                           { return '/'; }
"*"                           { return '*'; }
"%"                           { return '%'; }
">"                           { return '>'; }
"<"                           { return '<'; }
[0-9]+(\.[0-9]+)?\b           { return 'NUMBER'; }
[a-zA-Z_$][a-zA-Z0-9_$]*      { return 'IDENTIFIER'; }
\"([^\"]*?)\"                 { return 'STRING'; }
\'([^\']*?)\'                 { return 'STRING'; }
<<EOF>>                       { return 'EOF'; }
.                             { YYDriver.LogError(); }
```

2.4 Gramatyka

```
Literal
    : NULLTOKEN
    | TRUETOKEN
    | FALSETOKEN
    | NUMBER
    | STRING
    ;

PrivateBlock
    : PRIVATE SourceElements END
    ;

VariableStatement
    : VariableDeclarationList
    ;

VariableDeclarationList
    : IDENTIFIER
    | IDENTIFIER Initializer
    ;

Initializer
    : '=' AssignmentExpr
    ;

AssignmentExpr
    : LeftHandSideExpr
    | LeftHandSideExpr AssignmentOperator AssignmentExpr
    ;

AssignmentOperator
    : '='
    | PLUSEQUAL
    | MINUSEQUAL
    | MULTEQUAL
    | DIVEQUAL
    | LSHIFTEQUAL
    | RSHIFTEQUAL
    | URSHIFTEQUAL
    | ANDEQUAL
    | XOREQUAL
    | OREQUAL
    | MODEQUAL
    ;

LeftHandSideExpr
    : NewExpr
    ;

NewExpr
    : MemberExpr
    | NEW NewExpr
    ;

MemberExpr
    : PrimaryExpr
    | MemberExpr '[' Expr ']'
    | MemberExpr '.' IDENTIFIER
    | NEW MemberExpr Arguments
    ;

PrimaryExpr
    : PrimaryExprNoBrace
    | OPENBRACE CLOSEBRACE
```

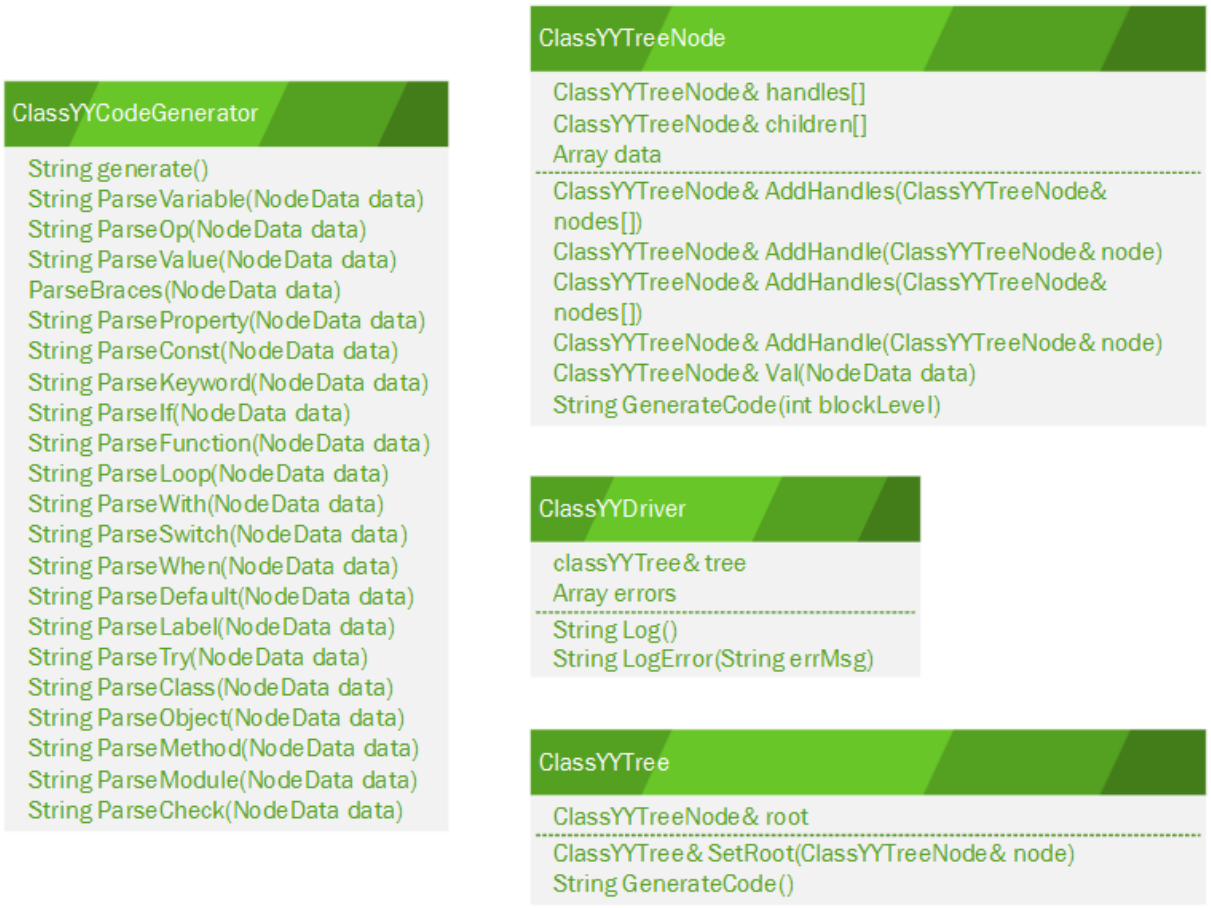
```
    | OPENBRACE PropertyList CLOSEBRACE
    | OPENBRACE PropertyList ',' CLOSEBRACE
    ;

PrimaryExprNoBrace
    : THISTOKEN
    | Literal
    | ArrayLiteral
    | IDENTIFIER
    | VOID Literal
    | '(' Expr ')'
    ;

ArrayLiteral
    : '[' ElisionOpt ']'
    | '[' ElementList ']'
    | '[' ElementList ',' ElisionOpt ']'
    ;

ElementList
    : ElisionOpt AssignmentExpr
    | ElementList ',' ElisionOpt AssignmentExpr
    ;

ElisionOpt
    :
    | Elision
    ;

Elision
    : ','
    | Elision ','
    ;

Expr
    : AssignmentExpr
    | Expr ',' AssignmentExpr
    ;

PropertyList
    : Property
    | PropertyList ',' Property
    ;



Property
    : IDENTIFIER ':' AssignmentExpr
    | STRING ':' AssignmentExpr
    | NUMBER ':' AssignmentExpr
    ;



FormalParameterList
    : IDENTIFIER
    | FormalParameterList ',' IDENTIFIER
    ;

FunctionBody
    :
    | SourceElements
    ;

Arguments
    : '(' ')'
```

```
    | '(' ArgumentList ')'
    ;

ArgumentList
    : AssignmentExpr
    | ArgumentList ',' AssignmentExpr
    ;

ConstStatement
    : CONST ConstDeclarationList
    ;

ConstDeclarationList
    : ConstDeclaration
    | ConstDeclarationList ',' ConstDeclaration
    ;

ConstDeclaration
    : IDENTIFIER
    | IDENTIFIER Initializer
    ;

DeleteStatement
    : DELETE IDENTIFIER
    ;

FunctionDeclaration
    : FUNCTION IDENTIFIER '(' ')' FunctionBody END
    | FUNCTION IDENTIFIER '(' FormalParameterList ')' FunctionBody END
    | FUNCTION IDENTIFIER FunctionBody END
    ;

IfStatement
    : IF Expr StatementAllowEmpty END %prec IF_WITHOUT_ELSE
    | IF Expr StatementAllowEmpty ELSE StatementAllowEmpty END
    ;

EmptyStatement
    :
    ;

StatementAllowEmpty
    : Statement
    | EmptyStatement
    ;


IterationStatement
    : WHILE Expr StatementAllowEmpty END
    | UNTIL Expr StatementAllowEmpty END
    | FOR IDENTIFIER OF IDENTIFIER StatementAllowEmpty END
    | FOR IDENTIFIER IN IDENTIFIER StatementAllowEmpty END
| FOR IDENTIFIER ',' IDENTIFIER IN IDENTIFIER StatementAllowEmpty END
    ;

ContinueStatement
    : CONTINUE
    | CONTINUE IDENT
    ;

BreakStatement
    : BREAK
    | BREAK IDENT
    ;

ReturnStatement
```

```
    : RETURN Expr END
    ;

WithStatement
    : WITH Expr StatementAllowEmpty END
    ;

SwitchStatement
    : SWITCH Expr CaseBlock END
    ;

CaseBlock
    : CaseClausesOpt
    | CaseClausesOpt DefaultClause
    ;

CaseClausesOpt
    :
    | CaseClauses
    ;

CaseClauses
    : CaseClause
    | CaseClauses CaseClause
    ;


CaseClause
    : WHEN Expr
    | WHEN Expr SourceElements
    | WHEN Expr THEN Statement
    ;

DefaultClause
    : DEFAULT SourceElements
    ;




LabelledBody
    :
    | SourceElements
    ;

LabelledStatement
    : IDENTIFIER ':' LabelledBody END
    ;

ThrowStatement
    : THROW Expr
    ;

TryStatement
    : TRY SourceElements FINALLY SourceElements END
    | TRY SourceElements CATCH Expr SourceElements END
    | TRY SourceElements CATCH Expr SourceElements FINALLY SourceElements
END
    ;

DebuggerStatement
    : DEBUGGER
    ;

ClassStatement
    : CLASS IDENTIFIER ObjectBody END
```

```
    | CLASS IDENTIFIER EXTENDS IDENTIFIER ObjectBody END
    ;

ObjectBody
    : ObjectBodyStatement
    | ObjectBody ObjectBodyStatement
    ;

ObjectBodyStatement
    : Property
    | MethodDeclaration
    ;

MethodDeclaration
    : DEFINE IDENTIFIER '(' ')' FunctionBody END
    | DEFINE IDENTIFIER '(' FormalParameterList ')' FunctionBody END
    | DEFINE IDENTIFIER FunctionBody END
    ;


ObjectStatement
    : OBJECT IDENTIFIER ObjectBody END
    | OBJECT IDENTIFIER CLONES IDENTIFIER ObjectBody END
    ;

Statement
    : PrivateBlock
    | VariableStatement
    | DeleteStatement
    | ConstStatement
    | FunctionDeclaration
    | IfStatement
    | IterationStatement
    | ContinueStatement
    | BreakStatement
    | ReturnStatement
    | WithStatement
    | SwitchStatement
    | LabelledStatement
    | TryStatement
    | DebuggerStatement
    | ClassStatement
    | ObjectStatement
    ;

SourceElements
    : Statement
    | SourceElements Statement
    ;

Program
    : EOF
    | SourceElements EOF
    ;
```

2.5 CodeBuilder



3 Użycie

3.1 Sposoby użycia parsera

3.1.1 Z linii poleceń

```
node redscript.js input.js > output.js
```

3.1.2 Z innego modułu

```
var parser = require("./redscript").parser;

function exec (input) {
    return parser.parse(input);
}

// input is String variable with RedScript code
// output is translated code to JavaScript
var output = exec(input);
```

3.1.3 Przez stronę internetową

```
<script src=”./redscript.js"></script>
<script>
parser.parse(input);
</script>
```

3.2 Plik wejściowy

Plikiem wejściowym jest plik zawierający kod zgodny ze specyfikacją RedScript. W tej postaci kod jest czytelny oraz umożliwia znacznie wygodniejsze korzystanie z możliwości, jakie daje JavaScript.

3.3 Plik wyjściowy

Po przekonwertowaniu tworzony jest plik zapisany całkowicie w JavaScript. Zachowuje on wszystkie funkcjonalności pliku wejściowego, a także może być używany w takim samym zakresie, jak każdy inny plik JS.

4 Bibliografia

JavaScript Language Specification : JavaScriptSpec.pdf

RedScript Language Specification : RedScriptSpec.pdf

Flex Manual : http://dinosaur.compilertools.net/flex/index.html

Bison Manual: http://dinosaur.compilertools.net/bison/index.html

Jison Documentation : http://zaach.github.io/jison/docs/