Projektová dokumentace

Implementace překladače imperativního jazyka IFJ19

Tým 096, vyrianta II

Michal Šedý (xsedym02): 34%

Ondřej Pavlacký (xpavla15): 33%

Radek Sedlář (xsedla1j): 33%

Vávra (xvavra21): 0%

Obsah

1. Úvod

Za úkol bylo vytvořit interpret jazyka IFJ19. IFJ19 je vytvořen z jazyka Python a náš interpret má fungovat jako konzolová aplikace, která postupně přijímá příkazy a vytváří mezikód, popřípadě vytváří chybové hlášení.

2. Návrh a implementace

Projekt je sestaven z několika modulů, které jsou níže popsány.

2.1 Lexikální analýza

Lexikální analýza je implementovaná jako scanner, jejíž hlavní funkcí je „`bool token_get_next(token_t* token)`“, která naplní token jenž dostává jako parametr při jejím volání. Pracuje podobně jako přiložený automat. V této funkci je pro každý typ tokenu kterého může nabývat jedna if-else konstrukce. Podle pravidel do jedné z nich spadne a vygeneruje jeden z příslušných tokenů v této konstrukci. Některé if-else jsou sdílené podobnými tokeny, např.: INT+FLOAT, ASSIGN+OPERATOR_ROVNOST. Pokud je nalezena chyba, tak se vrátí false. Pokud vše proběhne jak má vrátí true.

Escape sequence se zpracovávají až na konci tokenu a to jen u tokenu STRING a COMMENT.

Značení řetězců a dokumentačních řetězců bylo rozšířeno na úroveň Pyhon 3. Řetězce jsou ohraničeny buď uvozovkami, nebo apostrofy. Dokumentační řetězec je možno ohraničit trojicí uvozovek, i apostrofů.

2.2 Syntaktická analýza

Jedná se o klíčovou část programu. Na základě LL gramatiky a rekurzivního sestupu provádí funkce reprezentující různá pravidla. Rozhoduje o dalším postupu podle typu tokenu který získává voláním funkce `token_get_next` a příslušného pravidla v LL tabulce. Poté vytváří tříadresný kód, který jen uložen do instrukční pásky.

2.3 Sémantická analýza

Realizuje kontroly specifikované zadání. Využívá tabulku s rozptýlenými symboly. V případě vytváření funkce provedeme kontrolu v tabulce funkce zda je proměnná definována lokálně a pak kontroluje globální. V případě že není nalezena je navrácen kód sémantické chyby.

2.4 Generátor cílového kódu

Přímá instrukční pásku a provede překlad do mezijazyka IFJcode19 z interního tříadresného kódu.

2.5 Překlad projektu

Probíká pomocí přiloženého Makefile.

3. Speciální algoritmy a datové struktury

string_t

Je to struktura, která obsahuje [int len, int free, char* data]. Len odpovídá délce řetězce který jen uložen v paměti. Free odpovídá nadbytečnému místu alokovanému v paměti. Data je ukazatel na řetězec charakterů v paměti. Tuto strukturu jsme vytvořili k zjednodušení práce s řetězci. Spolu s touto strukturou jsou vytvořené funkce pro práci s ní.

token_t

Jedná se o strukturu která slouží pro odesílání dat mezi moduly programu. Obsahuje [enum type, token_data_t *data]. Type je enum, který nabývá hodnot jako jsou klíčová slova, EOL, EOF, DEDENT, INDENT… Data je odkaz do paměti na data která jsou typu podle výše uvedeného enumu.

token_data_t

Union která nabývá hodnot: int, double, char, string_t*. Toto jsou přímo data uložená v tokenu.

Instr_tape_t
Struktura reprezentující instrukční pásku. Jejíž prvek instr_t představující jednu instrukci, obsahuje kód instrukce, tři adresy operandu a ukazatel na další instrukci.

stack_t a prec_stack_t
Standartní pomocné stacky. Stack_t uchovává stupně odsazení. Prec_stack_t obsahuje ukazatele do tabulky a je využit při precedenční analýze matematických výrazů.

htab_t
Skládá se z počítadla parametrů, jejich zřetězeného seznamu a prvků samotných.

Rozptylová funkce
Využívá magickou hodnotu pro efentivní výpočet adresy prvku v tabulce.

4. Práce v týmu

4.1 Komunikace
Jako komunikační kanály jsme zvolily Facebook, Microsoft Teams. Komunikace probíhala vždy dobře, ale stejně se nic nevyrovnalo komunikaci z očí do očí. Většina problémů se vyřešila pouze takto.

4.2 Verzování
K verzování jsme použili Microsoft Teams. Rozděleno na několik podadresářů. Každý měl specifický účel.

4.3 Rozdělení
Rozdělení práce bylo stanoveno na první schůzi. Každý dostal cca 25% práce. Pan Vávra nás ubezpečoval že vše zvládne a na projektu pracuje. V jednu chvíli přestal komunikovat a zablokoval si nás na sociálních sítích. Michal Šedý a Ondřej Pavlacký si rozebrali jeho práci vše zvládli.

Ondřej Pavlacká – hashovací tabulka, instrukční páska, generátor kódu, výpis chyb, precedenční analýza, zásobník odsazení
Radek Sedlář – testy, dokumentace, dynamické řetězce, lexikální analýza
Michal Šedá – generátor kódu, syntaktická analýza, příjímání parametrů, vedení týmu, konzultace

5. Závěr
I přes mnohé organizační obtíže, které se vyskytly během implementace překladače, se podařilo vytvořit funkčně správný program. Bohužel, byl náš tým znevýhodněn odhodem jednoho člena, ale ani to nemohlo zabrání naši další spolupráci již pouze ve třech členech.

6. Přiložené grafy, tabulky, diagramy

Gramatická pravidla

##1## <dedent_eof> --> DEDENT

##2## <dedent_eof> --> e

##3## <prog> --> <free-line><prog-N>

##4## <free-line> --> EOL <free-line> EOL

##5## <free-line> --> e

##05## <prog-N> --> <stat><eol_eof><free-line><stat-list>

##06## <prog-N> --> e

##07## <stat-list> --> <free-line><stat><eol_eof><free-line><stat-list-N>

##08## <stat-list-N> --> <free-line><stat><eol_eof><free-line><stat-list-N>

##09## <stat-list-N> --> e

##10## <stat-list> --> <free-line><stat_if_while><eol_eof><free-line><stat-list-N>

##11## <stat_list_n_if_while> --> <free-line><stat_if_while><free-line><stat_list_n_if_while>

##12## <stat-stat_list_n_if_while> --> e

##13## <stat_if_while> --> if<expr>:EOL INDENT<stat_list_if_while>DEDENT else: EOL INDENT <stat_list_if_while><dedent_eof>

##14## <stat_if_while> --> while<expr>: EOL INDENT <stat_list_if_while><dedent_eof>

##15## <stat_if_while> --> <stat-const>

##16## <stat> --> if<expr>:EOL INDENT<stat_list_if_while>DEDENT else: EOL INDENT <stat_list_if_while><dedent_eof>

##17## <stat> --> while<expr>: EOL INDENT <stat_list_if_while><dedent_eof>

##18## <stat> --> def id(<params>): EOL INDENT <stat-list-F><dedent_eof>

##19## <stat> --> <stat-const>

##20## <stat_const> --> PASS

##21## <stat_const> --> (<expr>)

##22## <stat_const>--> id <assg-op-fce>

##23## <stat_const> --> STR-DOKUMENT

##24## <stat_const> --> INPUTS()

##25## <stat_const> --> INPUTI()

##26## <stat_const> --> INPUTF()

##27## <stat_const> --> PRINT(<params-print>)

##28## <stat_const> --> LEN(STR)

##29## <stat_const> --> SUBSTR(STR, NUM, NUM)

##30## <stat_const> --> ORD(STR, NUM)

##31## <stat_const> --> CHR(NUM)

##32## <stat_const> --> STRING<expr2>

##33## <stat_const> --> NONE<expr2>

##34## <stat_const> --> INT<expr2>

##35## <stat_const> --> FLOAT<expr2>

##36## <assg-op-fce> --> = <assign>

##37## <assg-op-fce> --> (<params-call>)

##38## <assg-op-fce> --> <operator> <expr>

##39## <expr> --> MATH

##40## <expr2> --> MATH2

##41## <eol_eof> --> EOL

##42## <eol_eof> --> e

##43## <assing> --> = identifier <expr_fce>

##44## <assing> --> <expr>

##45## <assing> --> INPUTI()

##46## <assing> --> INPUTF()

##47## <assing> --> INPUTS()

##48## <assing> --> LEN(STR)

##49## <assing> --> SUBSTR(STR, <expr>, <expr>)

##50## <assing> --> ORD(STR, <expr>)

##51## <assing> --> CHR(<expr>)

##52## <assign> --> PRINT(<params-print>)

##53## <expr_fce> -> (<params_call)

##54## <expr_fce> -> <operator><expr2>

##55## <params_call> --> id<params_n_call>

##56## <params_call> --> e

##57## <params_n_call> --> ,id<params_n_call>

##58## <params_n_call>--> ,DATA<params_n_call> data jsou zace int string atd.

##58## <params_n_call> --> e

##59## <operator> --> všechny operatory

##60## <params> --> id <param-N>

##61## <params> --> e

##45## <param-N> --> ,id <param-N>

##61## <param-N> --> e

##62## <params_print> --> <expr><params_print_n>

##63## <params_print> --> e

##64## <param-print-N> --> ,<expr><param_print_n>

##65## <param-print-N> --> e

##66## <stat-list> --> <free-line><stat-F><free-line><stat-list-F-N>

##67## <stat-list-N> --> <free-line><stat-F>EOL<free-line><stat-list-F-N>

##68## <stat-F> --> if<expr>:EOL INDENT<stat-list-F>DEDENT else: EOL INDENT <stat-list-F>DEDENT

##69## <stat-F> --> while<expr>: EOL INDENT <stat-list-F> DEDENT

##70## <stat-F> --> return <expr>

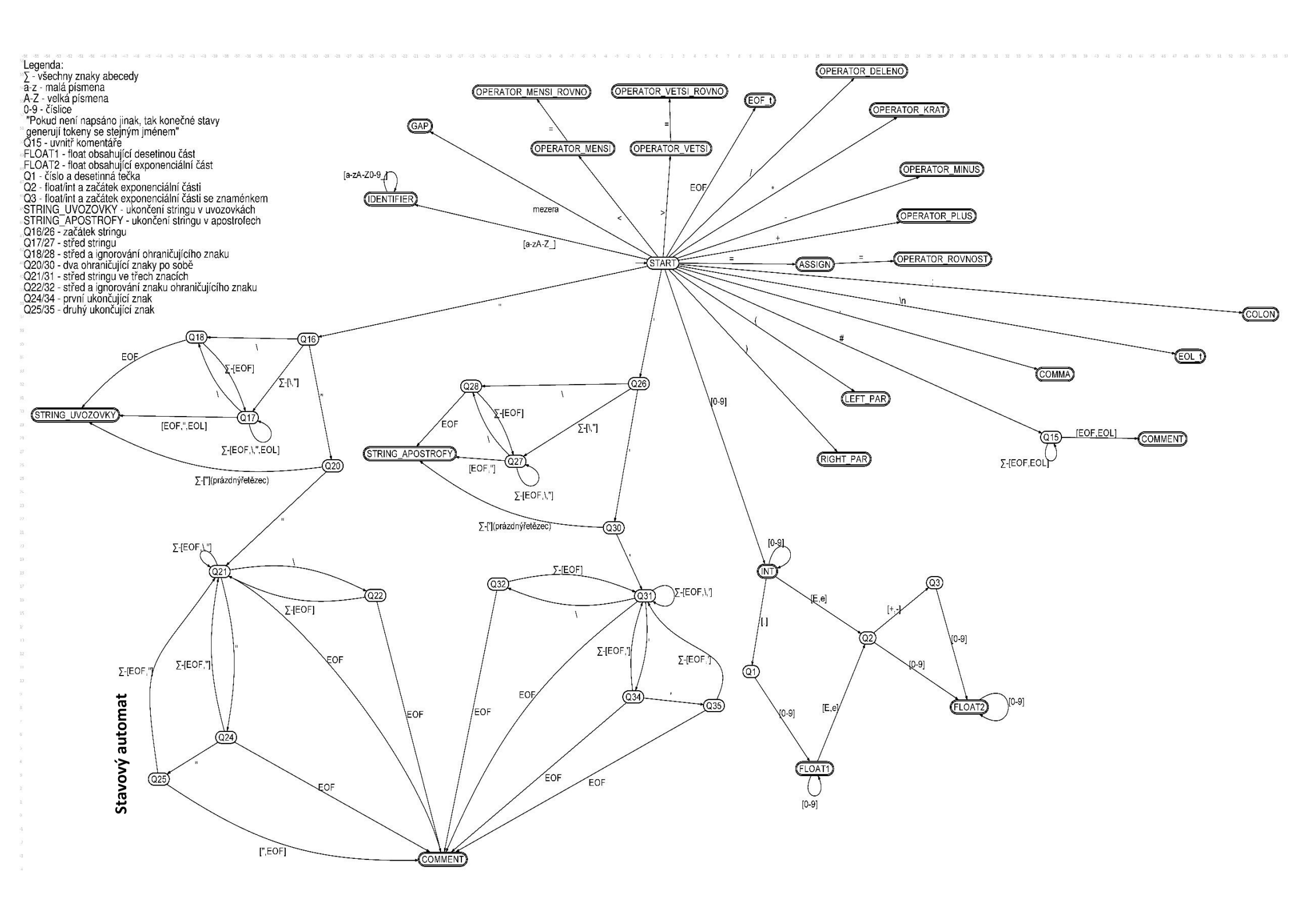
##71## <stat-F> --> <stat-const>

LL(1) tabulka

	DEDENT	EOL	if	else	:	INDENT	def	id	(	)	while	PASS	STR-DOKUMENT	INPUTS	INPUTI	INPUTF	LEN	STR	SUBSTR	ORD	,	CHR	PRINT	(,+,-,*,//,>,<,>=,<=,==,!=)	DATA	return	None	INT	FLOAT	MATH	MATH2	identifier	$
<dedent_eof>	1																																
<prog>		3	3				3	3	3		3	3	3	3	3	3	3	3	3	3		3	3				3	3	3				
<free-line>		4																															
<prog-N>			5				5	5	5		5	5	5	5	5	5	5	5	5	5		5	5				5	5	5				
<stat-F>			68					71	71		60	71	71	71	71	71	71	71	71	71		71	71	71		70				71			
<stat-list>			8				8	8	8		8	8	8	8	8	8	8	8	8	8		8	8				8	8	8				
<stat-list-N>			7				7	7	7		7		7	7	7	7	7	7	7	7		7	7				7	7	7				
<stat-stat_list_n_if_while>		10	10					10	10		10	10	10	10	10	10	10	10	10	10	10	10	10				10	10	10				11;10
<stat_if_while>			13					15	15		14	15	15	15	15	15	15	15	15	15		15	15				15	15	15				
<stat>			16				18	19	19		17	19	19	19	19	19	19	19	19	19		19	19				19	19	19				19;20;21
<stat_const>								22	21			20	23	24	25	26	28	32	29	30		31	27				33	34	35				
<ass-op-fce>									39					36	36	36	36		36	36		36	36	38						36			
<assing>														47	45	46	48		49	50		51	52							44			
<expr_fce>									53															54									
<params_call>								55																									
<params_n_call>																					57												
<operator>																								59									
<params-N>																					45												
<params>								60																									
<params_print>																														62			
<expr2>																															40		
<eol_eof>		41																															
<expr>																														39			
<params_print-N>																					64												

Precedenční tabulka

	*	/	//	+	-	rel	i	(	)	$
*	>	>	>	>	>	>	<	<	>	>
/	>	>	>	>	>	>	<	<	>	>
//	>	>	>	>	>	>	<	<	>	>
+	<	<	<	>	>	>	<	<	>	>
-	<	<	<	>	>	>	<	<	>	>
rel	<	<	<	<	<	X	<	<	>	>
i	>	>	>	>	>	>	X	X	>	>
(	<	<	<	<	<	<	<	<	=	X
)	>	>	>	>	>	>	X	X	>	>
$	<	<	<	<	<	<	<	<	X	X


Legenda:
∑ - všechny znaky abecedy
a-z - malá písmena
A-Z - velká písmena
0-9 - číslice
"Pokud není napsáno jinak, tak konečné stavy generují tokeny se stejným jménem"
Q15 - uvnitř komentáře
FLOAT1 - float obsahující desetinou část
FLOAT2 - float obsahující exponenciální část
Q1 - číslo a desetinná tečka
Q2 - float/int a začátek exponenciální části
Q3 - float/int a začátek exponenciální části se znaménkem
STRING_UVOZOVKY - ukončení stringu v uvozovkách
STRING_APOSTROFY - ukončení stringu v apostrofech
Q16/26 - začátek stringu
Q17/27 - střed stringu
Q18/28 - střed a ignorování ohraničujícího znaku
Q20/30 - dva ohraničující znaky po sobě
Q21/31 - střed stringu ve třech znacích
Q22/32 - střed a ignorování znaku ohraničujícího znaku
Q24/34 - první ukončující znak
Q25/35 - druhý ukončující znak
Stavový automat
START
GAP
IDENTIFIER
[a-zA-Z0-9_]
[a-zA-Z_]
mezera
OPERATOR_MENSI_ROVNO
OPERATOR_MENSI
OPERATOR_VETSI_ROVNO
OPERATOR_VETSI
EOF_t
OPERATOR_DELENO
OPERATOR_KRAT
OPERATOR_MINUS
OPERATOR_PLUS
ASSIGN
OPERATOR_ROVNOST
COLON
EOL_t
COMMA
LEFT_PAR
RIGHT_PAR
Q15
[EOF,EOL]
COMMENT
Σ-[EOF,EOL]
Q16
Q17
Q18
STRING_UVOZOVKY
[EOF,",EOL]
Σ-[EOF,\,",EOL]
Σ-[\,"]
Σ-[EOF]
Σ-["](prázdnýřetězec)
Q20
Q21
Q22
Q24
Q25
Σ-[EOF,\"]
Σ-[EOF,"]
Σ-[EOF,']
EOF
[",EOF]
Q26
Q27
Q28
STRING_APOSTROFY
[EOF,']
Σ-[EOF,\']
Σ-[']
Σ-[']
Q30
Q31
Q32
Q34
Q35
INT
FLOAT1
FLOAT2
Q1
Q2
Q3
[0-9]
[E,e]
[+,-]
[.]