

Lexikální analýza (Obsah)

- 1. Rekapitulace potřebných znalostí**
 - Regulární jazyky, regulární výrazy
 - Pravé lineární gramatiky
 - Konečné automaty (tabulka přechodů, stavový diagram, stavový strom)
 - Převod gramatika – konečný automat
 - Nedeterministický konečný automat, převod na deterministický
- 2. Levé lineární gramatiky**
- 3. Korespondence gramatik typu 3 a konečných automatů**
- 4. Vytváření derivačního stromu v případě lineárních gramatik**
- 5. Regulární atributované překladové gramatiky**
- 6. Princip lexikální analýzy**
- 7. Konstruktory lexikálního analyzátoru LEX, FLEX**

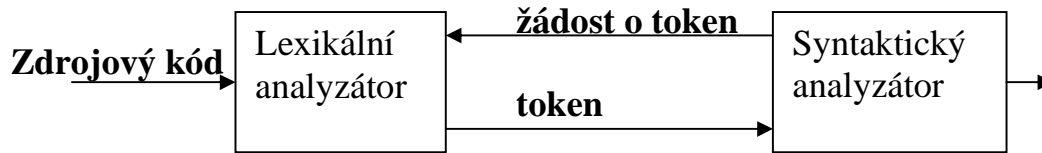
Úkoly lexikálního analyzátoru

- Čtení zdrojového textu,
- Nalezení a rozpoznání lexikálních symbolů ve volném formátu textu, včetně případného rozlišení klíčových slov a identifikátorů. Vyžaduje spolupráci s SA.
- Vynechání mezer a komentářů,
- Interpretace direktiv překladače,
- Uchování informace pro hlášení chyb,
- Zobrazení protokolu o překladu.

Proč je LA samostatnou částí

- Jednodušší návrh překladače
- Zlepšení efektivity překladu
- Lepší přenositelnost

**Lexikální analyzátor rozpoznává a zakóduje lexikální symboly jazyka
(lexémy anglicky tokens)**



Lexikální symboly jsou regulárním jazykem

- **Regulární jazyk**

Lze definovat gramatikou typu 3

$G = (N, T, P, S)$ kde P mají tvar

$X \rightarrow wY$ nebo $X \rightarrow w$ kde $w \in T^*$

(velkými písmeny označujeme neterminální symboly)

Př1. $S \rightarrow 1A$
 $A \rightarrow 0A \mid 1$

Př2. $S \rightarrow 1A \mid 1B$
 $A \rightarrow 0A \mid 0$
 $B \rightarrow 1B \mid 1$

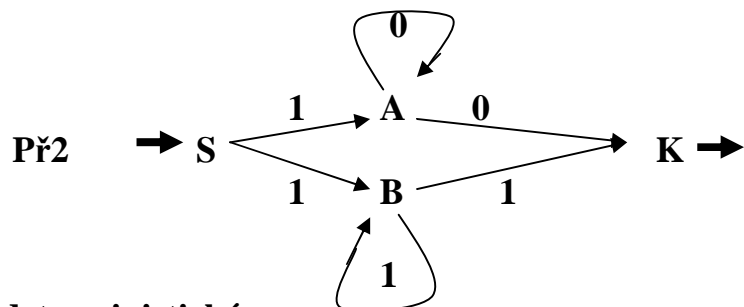
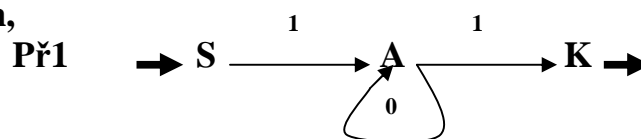
nebo konečným automatem

formální popis je pětice $KA = (Q, X, \delta, q_0, F)$

způsoby reprezentace přechodové funkce

-tabulka přechodů,

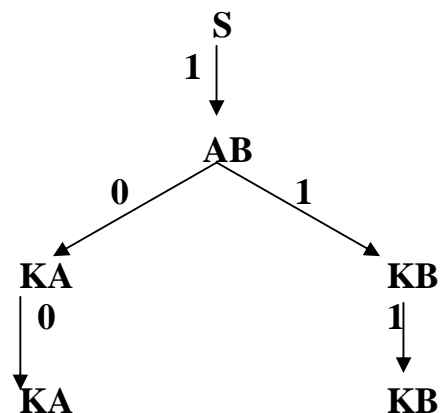
-stavový diagram,



Je nedeterministický

-stavový strom

Př2



Ted' je již deterministický

nebo regulárním výrazem

Př.2 $1 (0^* 0 \mid 1^* 1)$ pozn.: \mid také + také \vee
 dtto je $1 (0^+ \mid 1^+)$
 ? ale co $1 (0^n \mid 1^n)$ pro $n \geq 0$?

- Pumping lemma: Necht' L je regulární množina (regulární jazyk), pak existuje konstanta p taková, že je-li $w \in L$ a $|w| \geq p$, pak w lze zapsat ve tvaru xyz, kde $0 < |y| \leq p$ a $xy^iz \in L$ pro všechna $i \geq 0$

? je regulárním jazykem	$1 0^n 1$	pro $n \geq 0$
? je regulárním jazykem	$1^n 0^n$	„
? je regulárním jazykem	$1^n 2^n 3^n$	„
? je regulárním jazykem	$(1 2 3)^n$	„

Konfigurace automatu je dvojice (stav, ještě nezpracovaný vstup)

Počáteční konfigurace (S, věta), kde S je počáteční stav

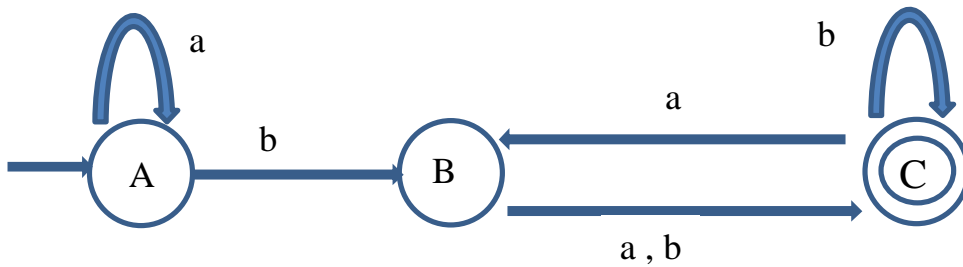
Koncová konfigurace (K, e), kde K je koncový stav a e je prázdný řetězec

\vdash je znak přechodu mezi konfiguracemi

Př. Analýza věty 1 0 0 1 jazyka 1 0ⁿ 1 (př.1)

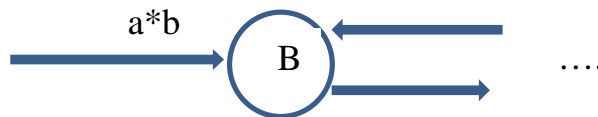
(S, 1001) \vdash (A, 001) \vdash (A, 01) \vdash (A, 1) \vdash (K, e)

Převod KA na regulární výraz

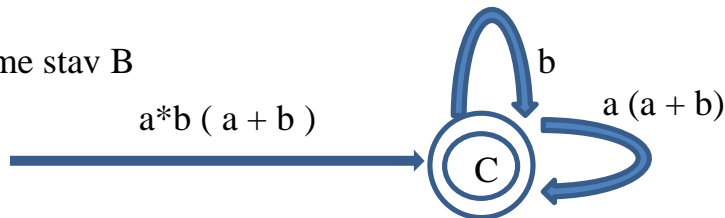


Eliminací stavů

-eliminujeme stav A



-eliminujeme stav B



-spojíme do konečného výsledku $a^*b (a + b) ((a (a + b))^* + b^*)^*$

Řešením regulárních rovnic

$A = A a$ protože existuje hrana z A do A přes a

$B = A b \mid C a$ protože existují hrany z A do B přes b a z C do B přes a

$C = B a \mid B b \mid C b$

A je počáteční stav, efektem rovnice pro A je hvězdičkování a^* . Dosadíme za A a pak za B do dalších rovnic.

$$A = a^*$$

$$B = a^* b \mid C a$$

$$C = a^* b a \mid C a a \mid a^* b b \mid C a b \mid C b$$

Řešíme hvězdičkováním rekurzivní rovnici pro C. Místo \mid použijeme $+$.

$$C = a^* b (a + b) \{ C [a (a + b) + b] \}$$

S výsledkem

$$C = a^* b (a + b) \{ [a (a + b)]^* + b^* \}^*$$

Levé a pravé lineární gramatiky

Pravá lineární:

$G = (N, T, P, S)$ kde P mají tvar $X \rightarrow w Y$ $w \in T^*$
 $X \rightarrow w$

Levá lineární:

$G = (N, T, P, S)$ kde P mají tvar $X \rightarrow Y w$ $w \in T^*$
 $X \rightarrow w$

Lze převést na tvar $X \rightarrow Y a$ a je term.symbol
 $X \rightarrow a$ příp. $X \rightarrow e$

Každou lineární gramatiku lze převést na regulární tvar

Konstrukce ekvivalentního KA pro levou regulární gramatiku:

- ❖ Neterminálnímu symbolu odpovídá stav
- ❖ Počáteční stav nepatří do N (je jím i stav A , pro nějž $A \rightarrow e \in P$)
- ❖ Každému pravidlu odpovídá větev takto:
 - 1) $Z Y$ do X označená a , je-li $X \rightarrow Y a \in P$
 - 2) Z počátečního stavu do X označená a , je-li $X \rightarrow a \in P$
 - 3) Koncovým stavem je počáteční symbol gramatiky

Př. Konstruuje KA pro $G = (N, T, P, Z)$, kde P : $Z \rightarrow U0 \mid V1$

S	U	$U \rightarrow Z1 \mid 1$
V	Z	$V \rightarrow Z0 \mid 0$

Doplňme

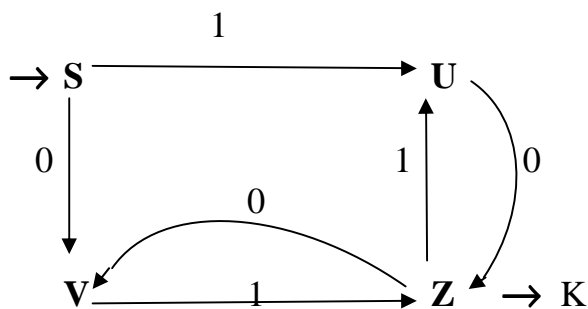
Jak by vypadala ekvival. pravá lin.g?
(má poč. symbol S)

Generování věty 101001 (symbol => znamená derivuje)

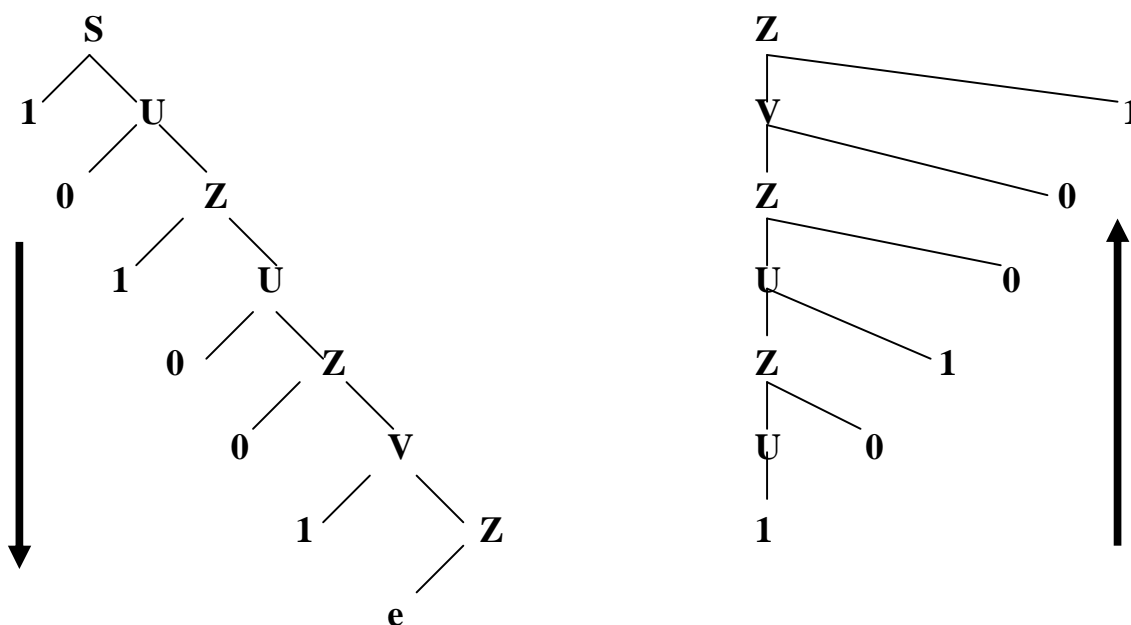
↓ $S \Rightarrow 1U \Rightarrow 10Z \Rightarrow 101U \Rightarrow 1010Z \Rightarrow 10100V \Rightarrow 101001Z \Rightarrow 101001$

↑ $Z \Rightarrow V1 \Rightarrow Z01 \Rightarrow U001 \Rightarrow Z1001 \Rightarrow U01001 \Rightarrow 101001$

Výsledek:



Graficky zachycuje derivaci derivační strom



Vstupující řetězec vždy čteme zleva doprava (všimněte si jak se liší konstrukce derivačního stromu, princip expanze neterminálu při ↓ versus redukce na neterminál při ↑, vstup se vždy zpracovává/čte zleva)

Př. Zapište gramatiku identifikátoru a) pravou, b) levou lineární gramatikou

Konstruujte ekvivalentní automat

Zkuste derivovat nějakou větu a vykreslit její derivační strom

Regulární atributované a překladové gramatiky

Atributovaná gramatika $AG = (G, \text{Atributy}, \text{Sémantická pravidla})$

Atributy jsou přiřazeny symbolům gramatiky a sémantická pravidla jednotlivým prepisovacím pravidlům. Při aplikaci prepisovacího pravidla se provedou příslušná sémantická pravidla a vypočtou hodnoty atributů. Atributy vyhodnocované průchodem derivačním stromem zdola nahoru nazýváme syntetizované, shora dolů nazýváme dědičné.

Překladová gramatika $PG = (N, T \cup D, P, S)$

Obsahuje disjunktní množiny T a D , vstupních a výstupních terminálních symbolů

Regulární pravá překladová gramatika má množinu pravidel tvaru

$X \rightarrow a w' Y$, $X \rightarrow a w'$ kde $a \in T$ a $w' \in D^*$,
a nebo $S \rightarrow e$, pokud se S nevyskytuje na pravé straně pravidel.

Př. $PG = (\{S, A, B, C\}, \{i, +, *\} \cup \{i', +', *'\}, P, S)$ s pravidly

$S \rightarrow i i' A$	$S \rightarrow i i'$
$A \rightarrow * C$	$A \rightarrow + B$
$B \rightarrow i i' +' A$	$B \rightarrow i i' +'$
$C \rightarrow i i' *' A$	$C \rightarrow i i' *'$

Derivujme vstupní řetězec $i * i + i$

$S \Rightarrow i i' A \Rightarrow i i' * C \Rightarrow i i' * i i' *' A \Rightarrow i i' * i i' *' + B$
 $\Rightarrow i i' * i i' *' + i i' +'$

Derivací vstupního řetězce vznikl řetěz výstupních symbolů $i' i' *' i' +'$

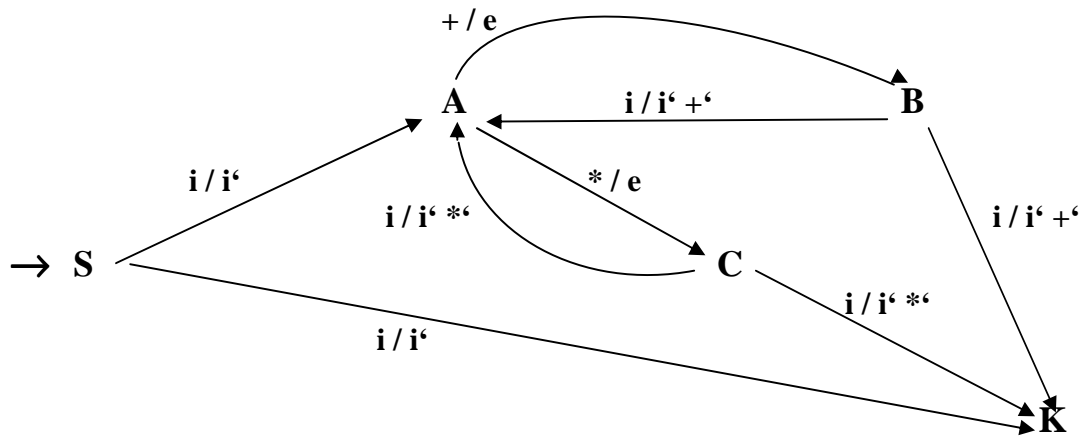
Vidíme jej v řetězci $i i' * i i' *' + i i' +'$ „brýlemi výstupního homomorfismu“ (těmi vidíme jen výstupní symboly)

Uvedená gramatika realizuje „nedokonalý“ překlad z infixového zápisu do postfixového. V čem je jeho nedokonalost?

Regulární překladové gramatice odpovídá konečný překladový automat KPA

	A		B	
S		C		doplňte graf
			K	

Výsledek:



Atributovaná překladová gr. APG = (PG, Atributy, Sémantická pravidla)

Př. Popište APG překlad znakového zápisu celých čísel do jeho hodnoty
 Gramatika celého čísla

G[C]: $C \rightarrow \check{c} C \mid \check{c}$ je nedeterministické, spravíme to

G[C]: $C \rightarrow \check{c} Z$ je deterministické
 $Z \rightarrow \check{c} Z \mid e$

Překladová gramatika

PG[C]: $T = \{ \check{c} \}$, $D = \{ \text{výstup} \}$
 $C \rightarrow \check{c} Z$
 $Z \rightarrow \check{c} Z \mid e \text{ výstup}$

APG[C]: bude navíc obsahovat atributy symbolů a sémantická pravidla

symbol	atributy	
	dědičné	syntetizované
\check{c}		kód
C	hodnota	
Z	hodnota	
<i>výstup</i>	hodnota	

syntax	sémantická pravidla
$C \rightarrow \check{c} Z$	$Z.\text{hodnota} = \text{ord}(\check{c}.\text{kód}) - \text{ord}('0')$
$Z^0 \rightarrow \check{c} Z^1$	$Z^1.\text{hodnota} = Z^0.\text{hodnota} * 10 + \text{ord}(\check{c}.\text{kód}) - \text{ord}('0')$
$Z \rightarrow e \text{ výstup}$	$\text{výstup}.\text{hodnota} = Z.\text{hodnota}$

Pozn.: Horním indexem odlišujeme stejně pojmenované symboly v pravidle

Př. Nakreslete ekvivalentní automat a interpretujte překlad věty 235

Princip lexikálního analyzátoru (Nalezení a rozpoznání lexikálního symbolu)

Třídy symbolů:

- Identifikátory
- Klíčová slova (rezervované identifikátory)
- Celá čísla
- Jednoznakové omezovače
- Dvouznakové omezovače

Gramatický popis tříd symbolů:

<identifikátor> → písmeno <id>

<id> → písmeno <id> | číslice <id> | e

<klíčové slovo> → begin | end | do | while

<celé číslo> → číslice | číslice <celé číslo>

<jednoznakový omezovač> → + | - | / | * | (|)

<dvouznakový omezovač> → // | ** | :=

poznámky tvaru: /* poznámka */

? co je počátečním symbolem?

<Symbol> → <identifikátor> |
 <celé číslo> |
 <jednoznakový omezovač> |
 <dvouznakový omezovač>

...

zakódování symbolů zvolme např.:

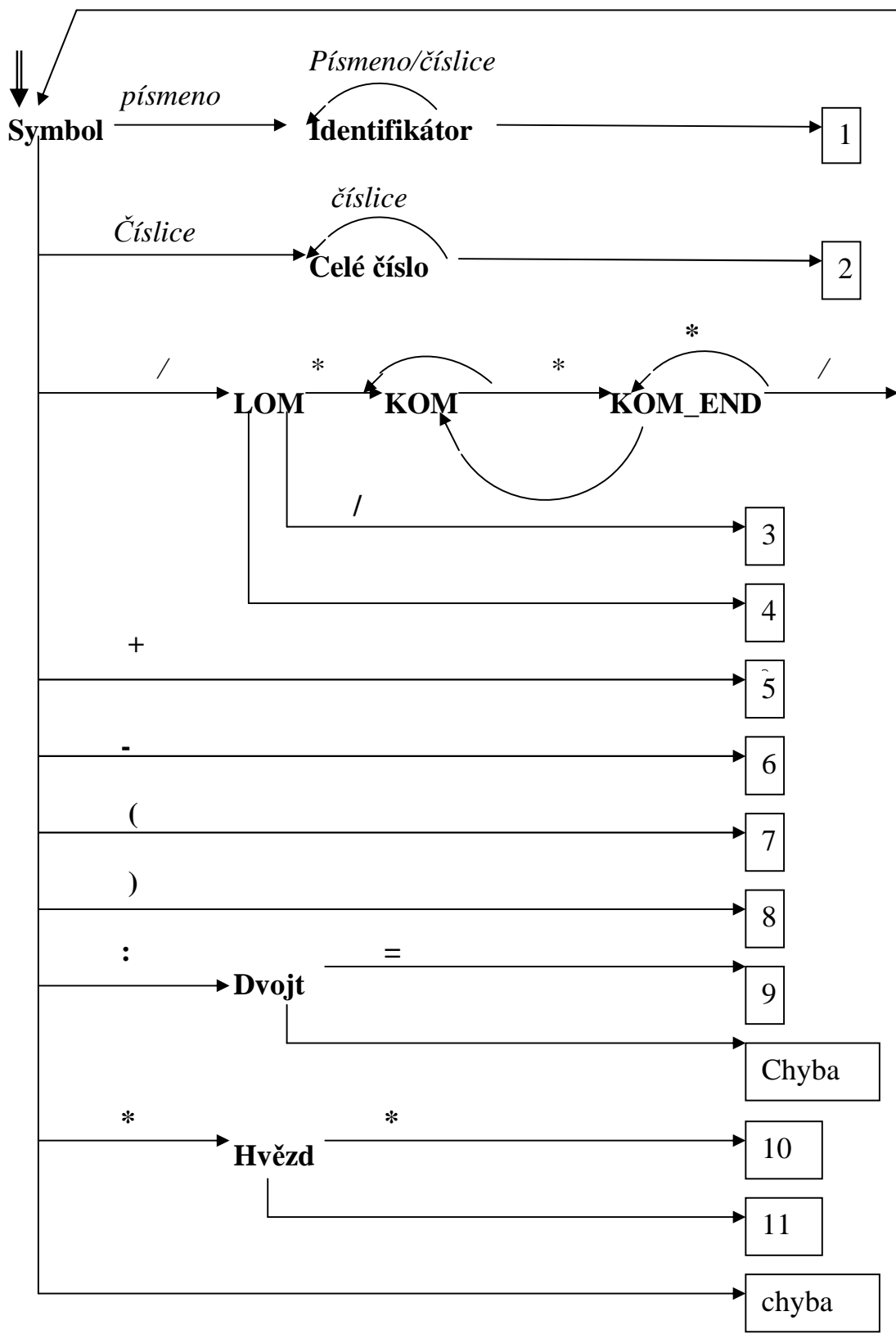
<u>symbol</u>	<u>kód</u>	<u>symbol</u>	<u>kód</u>	<u>symbol</u>	<u>kód</u>
identifikátor	1	celé číslo	2	//	3
/	4	+	5	-	6
(7)	8	:=	9
**	10	*	11	begin	12
end	13	do	14	while	15

? jak vypadá konečný automat?

- **Zpracování začíná prvním dosud nezpracovaným znakem ze vstupu,**
- **Zpracování končí, je-li automat v koncovém stavu a pro další vstupní znak již neexistuje žádný přechod**
- **Pro každou kategorii předpokládáme samostatný koncový stav,**
- **Neohodnocená větev se vybere, pokud vstupujícímu znaku neodpovídá žádná z ohodnocených větví**

Při zpracování sémantiky symbolů

- **Hodnoty atributů se vypočtou z lexému**
- **Klíčová slova / rezervované identifikátory rozlišíme za pomoci tabulky klíčových slov.**



Sémantické zpracování lexikálních elementů

Kód lexémů představuje informaci o druhu lexému, ne plně o jeho významu.

Rozdíl +, /,
 do, while
 1415, x1, alfa

LA musí předat i atributy lexému, tj.

- u čísel jejich hodnotu
- u identifikátorů textový tvar (či adresu / ukazatel do tabulky)

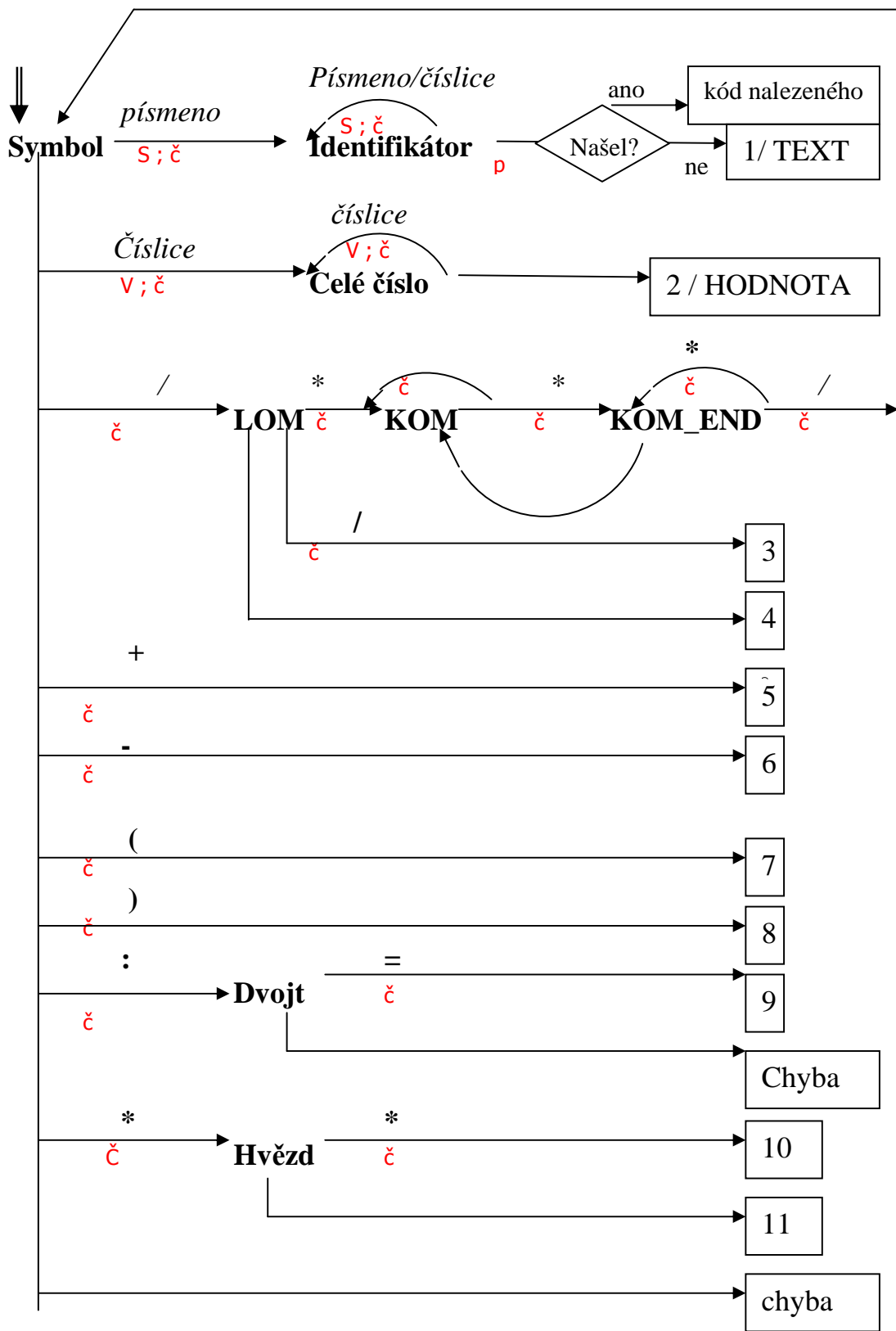
Předp. Identifikátor předáván jako dvojice 1, text
 Číslo „ „ „ „ 2. hodnota

Např. do /*dokud te to bavi*/ alfa := 10 * (x + y)
 převede na
 14, 1, alfa, 9, 2, 10, 11, 7, 1, x, 5, 1, y, 8
 nebo
 14, -, 1, alfa, 9, -, 2, 10, 11, -, 7, -, 1, x, 5, -, 1, y, 8, -

Automat LA bude rozšířen o funkce

- **č** ČTI čte jeden znak zdrojového textu (posouvá hlavičku KA)
- **s** SLOŽ zřetězuje znaky do proměnné TEXT
- **p** PROHLEDEJ hledá v tabulce rezervovaných slov a v případě nalezení vrací jeho kód
- **v** VYPOČTI po znacích vyčísluje hodnotu konstanty do proměnné HODNOTA

? jak je zařadit do diagramu ?



Nejednoznačnosti v lexikální analýze

Nastává v případě, kdy jeden symbol je prefixem jiného symbolu (== apod.)

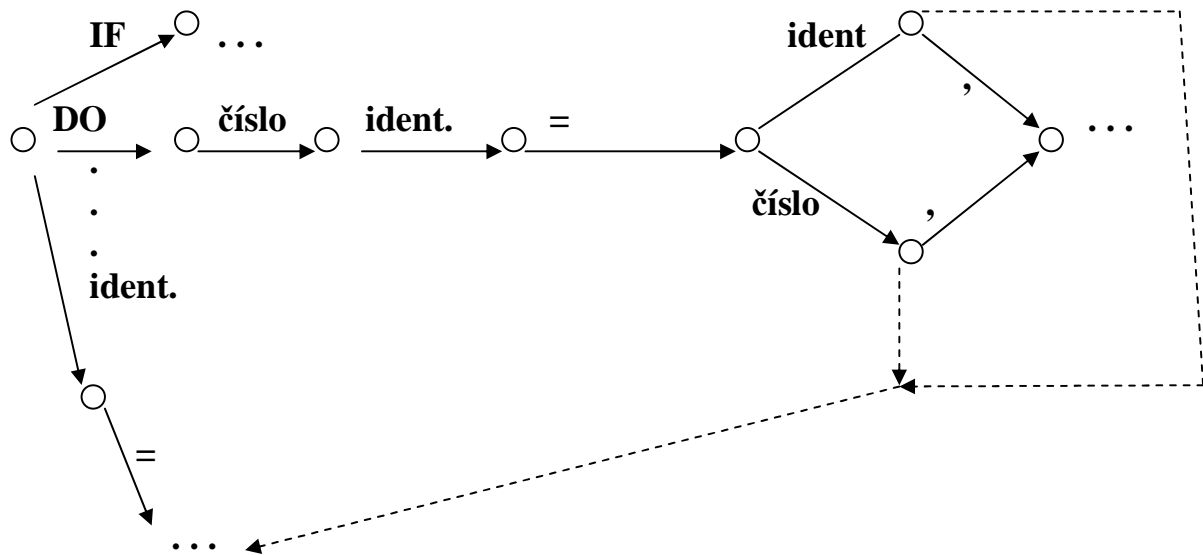
Pravidlo “hledej nejdelší symbol”

Nedokonalosti některých jazyků

Fortranské číslo versus relace 123 . EQ. Y
Nutno ukládat znaky do pomocného pole

Příkaz cyklu DO 10 I = 1 , 5

Vyžaduje nápovědu od syntaktického analyzátoru



LEX/FLEX (Unix/Linux)

Pro Windows lze stáhnout "complete package, except sources" ze stránky:

<http://gnuwin32.sourceforge.net/packages/bison.htm>

<http://gnuwin32.sourceforge.net/packages/flex.htm>

Generuje program v jazyce C do souboru lex.yy.c, který definuje funkci yylex(). Po přeložení generuje proveditelný kód. Manuál máte v souboru Flex.pdf

lex popis_lex_pravidel: popis → Lex → lex.yy.c

cc lex.yy.c -ll: lex.yy.c → C → a.out

a.out: vstupní_text → a.out → výstup

Obecný tvar vstupního souboru pro Lex:

```
{definice použité v regulárních výrazech a C deklarace}
%%
{pravidla v podobě regulárních výrazů a příslušných akcí}
%%
{doplňkové procedury}
```

-Definice - zahrnují deklarace proměnných,
konstant,
regulárních definic.

-Pravidla mají tvar -

p1	{akce1 v C notaci}
P2	{akce2 " " }
...	
pn	{akceN " " }

pi jsou regulární výrazy
{akcei} jsou programové fragmenty

-Doplňkové procedury jsou pomocné, mohou obsahovat C rutiny volané akcemi

Regulární výrazy v pravidlech mohou mít podobu:

```

je-li      c      jeden znak
           r      reg. výraz
           s      řetězec
           i      identifikátor
    
```

pak výrazu	odpovídá	např.
c	libov. neoperátorový znak c	a
\c	znak c literálně	*
"s"	řetězec s literálně	"**"
.	libov. znak mimo nový řádek	a.*b
^	začátek řádky	^abc
\$	konec řádky	abc\$
[s]	libov. znak z s	[abc]
[x-z]	znaky x, y, . . . z.	[0-9]
[^s]	" " " není-li z s	[^abc]
r*	nula nebo více r	a*
r+	jeden nebo více r	a+
r?	nula nebo jeden r	a?
r{m,n}	m až n výskytů r	a{1,5}
r1r2	r1 pak r2	ab
r1 r2	r1 nebo r2	a b
(r)	r	(a b)
r1/r2	r1 je-li n sledováno r2	abc/123
{i}	překlad i z definiční sekce	{PISMENO}

```

yylval      proměnná pro předání tokenu do Yacc (ten provádí synt. analýzu)
yytext      proměnná obsahující text odpovídajícího reg.výrazu
yylen      " " počet znaků "
yyless(n)   ubere n znaků z yytext[]
yymore()    přidá k obsahu yytext[] další koresp. část textu
REJECT      přejde na další pravidlo bez změny obsahu yytext[]
    
```

Příklad

```
%{
    /* definice manifestovych konstant
    LT, LE, EQ, NE, GT, GE, IF, THEN, ELSE,
    ID, NUMBER, RELOP */
}%

/* regularni definice */
delim  [\t\n]
ws     {delim}*
letter [A-Za-z]
digit  [0-9]
id     {letter}({letter}|{digit})*
number {digit}+(\.{digit}+)?(E[+-]?{digit}+)?

%%
{ws}      { /* zadna akce ani navrat */}
if        {return(IF);}
then      {return(THEN);}
else      {return(ELSE);}
{id}      {yylval=install_id(); return(ID);}
{number}  {yylval=install_num(); return(NUMBER);}
"<="     {yylval=LE; return(RELOP);}
"="       {yylval=EQ; return(RELOP);}
"<"      {yylval=NE; return(RELOP);}
">="     {yylval=GE; return(RELOP);}
"<"      {yylval=LT; return(RELOP);}
">"      {yylval=GT; return(RELOP);}
%%

install_id() {
    /* vlozi do tabulky symbolu lex.elem., jehoz prvý
    znak je urceny v yytext a delka je v yyleng.
    Vracenou hodnotou je ukazatel do tab.sym. NEROZEPSANA
    */
}
install_num() {
    /* podobne, pro instalaci cisla */
}
```

Prostředky pro regulární výrazy jiných programovacích jazyků jsou z Lex

viz <http://osteele.com/tools/rework/#>

Př.1) [a-zA-Z][0-9a-zA-Z]*

```
JavaScript "nejaky text".match(/[a-zA-Z][0-9a-zA-Z]*/g)
    re.exec("nejaky text")
```

```
PHP preg_match_all('/[a-zA-Z][0-9a-zA-Z]*/', "nejaky
text", $match)
```

```
Python re.findall(r'[a-zA-Z][0-9a-zA-Z]*', "nejaky
text")
```

```
Ruby "nejaky text".scan(/[a-zA-Z][0-9a-zA-Z]*/)
```

Použito v Python

```
>>> re.findall(r'[a-zA-Z][0-9a-zA-Z]*', 'ab1 nic 44')
['ab1', 'nic']
```

Př. 2) ([+-]?d*\d+([eE][+-]?d+)?)

```
JavaScript "-2.33e-2alfa11beta12e3?.12E3".replace(/(([-+
]?\d*\.\d+([eE][+-]?\d+)?)/g, "'expcislo' ")
```

```
PHP preg_replace('/(([-+]?\\d*\\.\\d+([eE][+-]
]?\d+)?)'/, "'expcislo' ", "-2.33e-
2alfa11beta12e3?.12E3")
```

```
Python re.sub(r'((-+)?\\d*\\.\\d+([eE][+-]?\d+)?)',
"'expcislo' ", "-2.33e-2alfa11beta12e3?.12E3")
```

```
Ruby "-2.33e-2alfa11beta12e3?.12E3".gsub(/((-+
]?\d*\.\d+([eE][+-]?\d+)?)/, "'expcislo' ")
```

Použito v Python

```
>>> re.sub(r'([-+]?d*\d+([eE][+-]?d+)?)', "'expcislo' ",
"-2.33e-2alfa11beta12e3?.12E3")
"'expcislo' alfa11beta12e3?'expcislo' "
>>> re.findall(r'([-+]?d*\d+([eE][+-]?d+)?)', "-2.33e-2alfa11beta12e3?.12E3")
[('-2.33e-2', 'e-2'), ('.12E3', 'E3')]
```