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 typu3 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**

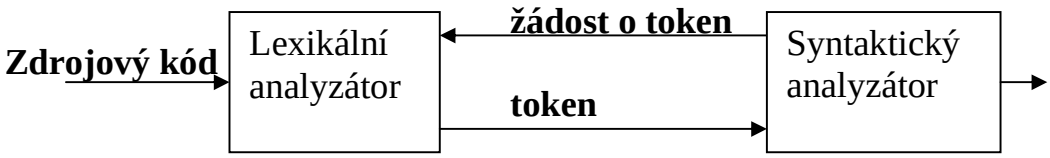
Úkoly lexikálního analyzátoru

- **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 → w Y nebo X → w kde w ϵ T^*

  (velkými písmeny označujeme neterminální symboly)

  Př1. **S → 1A**
  A → 0A | 1

  Př2. **S → 1A | 1B**
  A → 0A | 0
  B → 1B | 1

  nebo konečným automatem

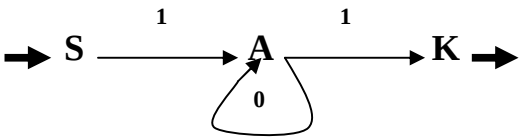
  formální popis je pětice KA = (Q, X, δ, q_0, F)

  způsoby reprezentace přechodové funkce

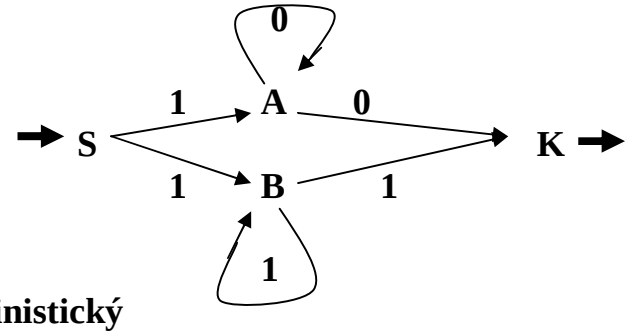
  -tabulka přechodů,

  -stavový diagram,

Př1



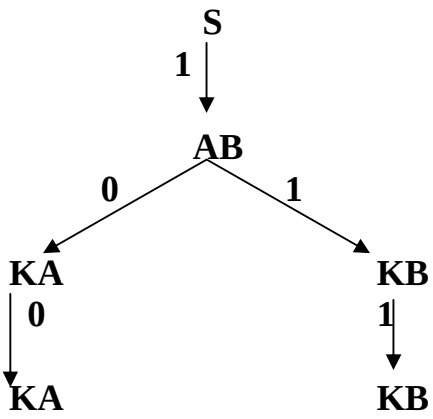
Př2



Je nedeterministický

-stavový strom

Př2



Teď je již deterministický

nebo regulárním výrazem

Př.2 $1\ (\ 0^*\ 0\ \mid 1^*\ 1\ )$ **pozn.: | také + také ∨**

dtto je $1\ (\ 0^+\ \mid 1^+\ )$

? ale co $1\ (\ 0^n\ \mid 1^n\ )$ **pro n>=0 ?**

- **Pumping lemma: Nechť L je regulární množina (regulární jazyk), pak existuje konstanta p taková, že je-li w ∈ L a |w| >= p, pak w lze zapsat ve tvaru xyz, kde 0 < |y| <= p a xy^iz ∈ L pro všechna i >= 0**

? je regulárním jazykem $1\ 0^n\ 1$ **pro n >= 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

├ je znak přechodu mezi konfiguracemi

Př. Analýza věty 1 0 0 1 jazyka $1\ 0^n\ 1$ **(př.1)**

(S, 1001)├ (A, 001) ├ (A, 01) ├ (A, 1) ├ (K, e)

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.** $X \rightarrow e$

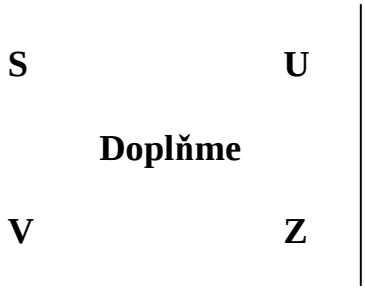
Každou lineární gramatiku lze převést na regulární tvar

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 $X \rightarrow Ya \in P$**
2) **Z počátečního stavu do X označená a. je-li $X \rightarrow a \in P$**
3) **Koncovým stavem je počáteční symbol gramatiky**

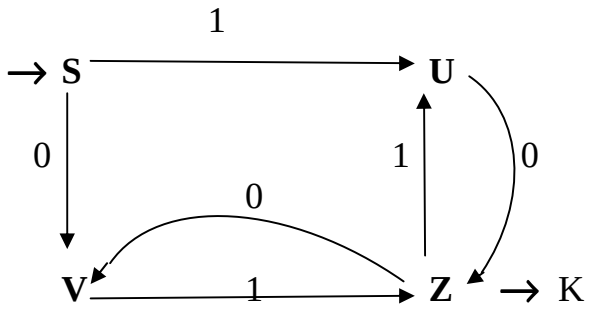
Př. Konstruujte KA pro G = (N, T, P, Z) , kde P: $Z \rightarrow U0 \mid V1$
$U \rightarrow Z1 \mid 1$
$V \rightarrow Z0 \mid 0$



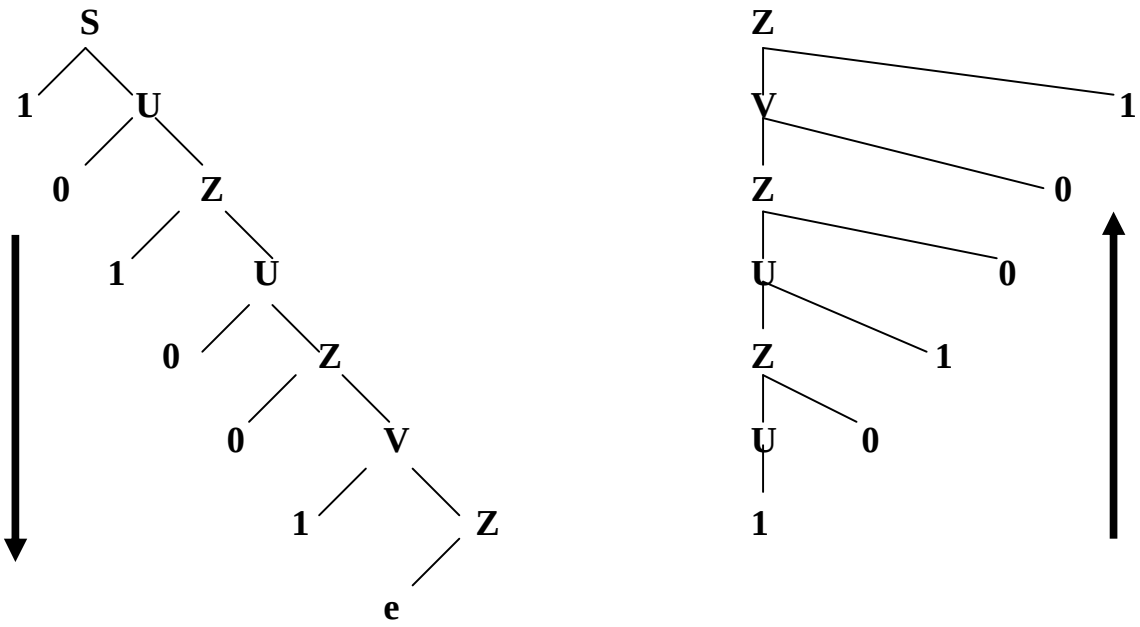
Jak by vypadala ekvival. pravá lin.g.?
(má poč. symbol S)

Generování věty 101001 (symbol => znamená derivuje)
↓ **S => 1U => 10Z => 101U => 1010Z => 10100V => 101001Z => 101001**
↑ **Z => V1 => Z01 => U001 => Z1001 => U01001 => 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 z leva)

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, Atributy, Sémantická pravidla)

Atributy jsou přiřazeny symbolům gramatiky a sémantická pravidla jednotlivým přepisovacím pravidlům. Při aplikaci přepisovacího 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 ∪ 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,+,*} ∪ {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 => i i' A => i i' * C => i i' * i i' *' A => i i' * i i' *' + B

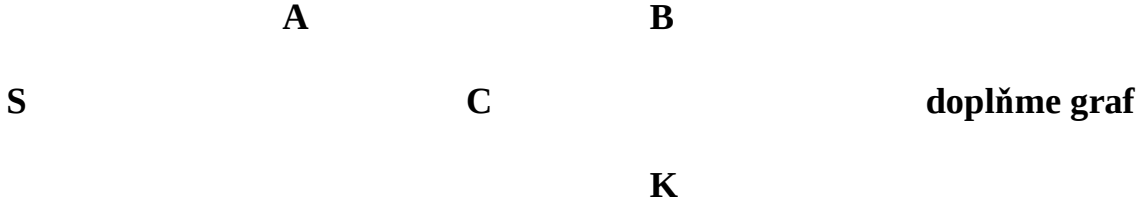
=> i i' * i i' *' + i i' +'

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)

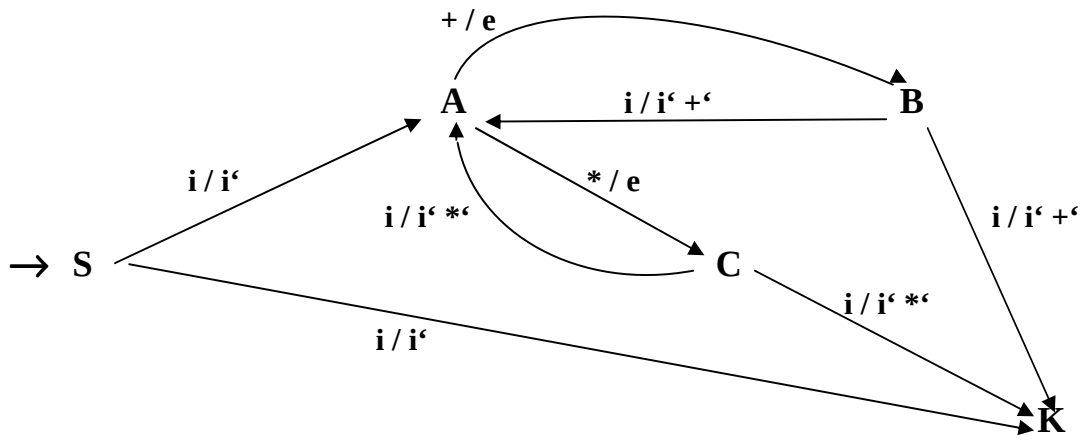
Uvedená gramatika realizuje „nedokonalý" překlad z infixového zápisu do postfixového. V čem je jeho nedokonalost?

Regulární překladové gramatice odpovídá konečný překladový automat KPA



doplňme graf

Výsledek:



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

Př. Popišme APG překlad znakového zápisu celých čísel do jeho hodnoty
Gramatika celého čísla
G[C]: C → č C | č je nedeterministické, spravíme to

G[C]: C → č Z je deterministické
Z → č Z | e

Překladová gramatika
PG[C]: T = { č }, D= { *výstup* }
C → č Z
Z → č Z | e *výstup*

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

symbol	atributy	
	dědičné	syntetizované
č		kód
C	hodnota	
Z	hodnota	
výstup	hodnota	

syntax	sémantická pravidla
C → č Z	Z.hodnota = ord(č.kód) – ord('0')
Z^0 → č Z^1	Z^1.hodnota = Z^0.hodnota * 10 + ord(č.kód) – ord('0')
Z → e *výstup*	*výstup*.hodnota = Z.hodnota

Pozn.: Horním indexem odlišujeme stejně pojmenované symboly v pravidle

Př. Nakreslete ekvivalentní automat a interpretujte překlad věty 235

<u>Princip lexikálního analyzátoru</u> (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.:

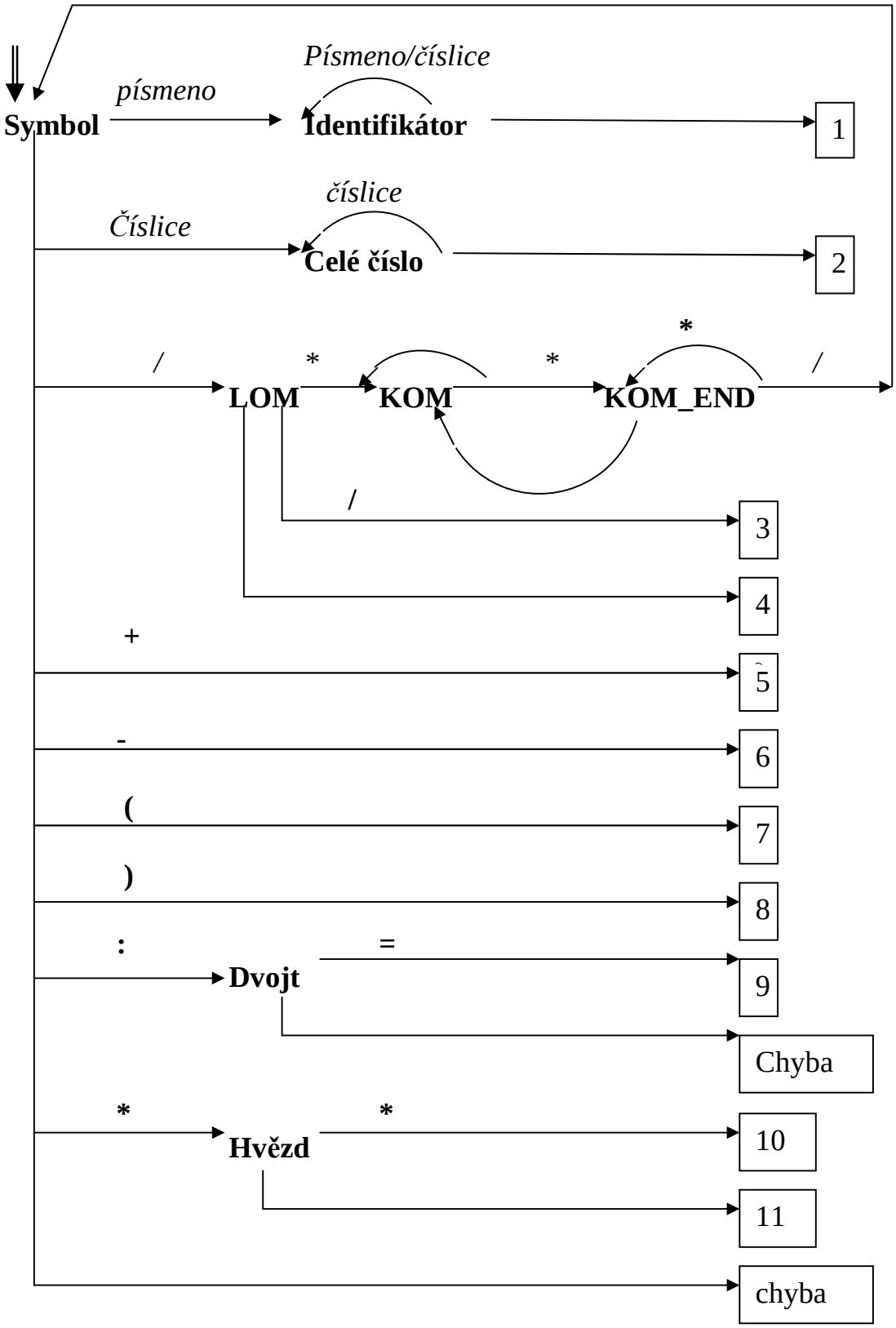
symbol	kód	symbol	kód	symbol	kód
identifikátor	1	celé číslo	2	//	3
/	4	+	5	-	6
(	7	)	8	:=	9
**	10	*	11	begin	12
end	13	do	14	while	15

? jak vypadá konečný automat?

- **Zpracování začíná prvým 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.**


Písmeno/číslice
písmeno
Symbol
Identifikátor
1
číslice
Číslice
Celé číslo
2
*
/
*
*
/
LOM
KOM
KOM_END
/
3
4
+
5
-
6
(
7
)
8
:
=
Dvojt
9
Chyba
*
*
Hvězd
10
11
chyba

Sémantické zpracování lexikálních elementů

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

Rozdíl +, /,
do, while
1415, x1, alfa

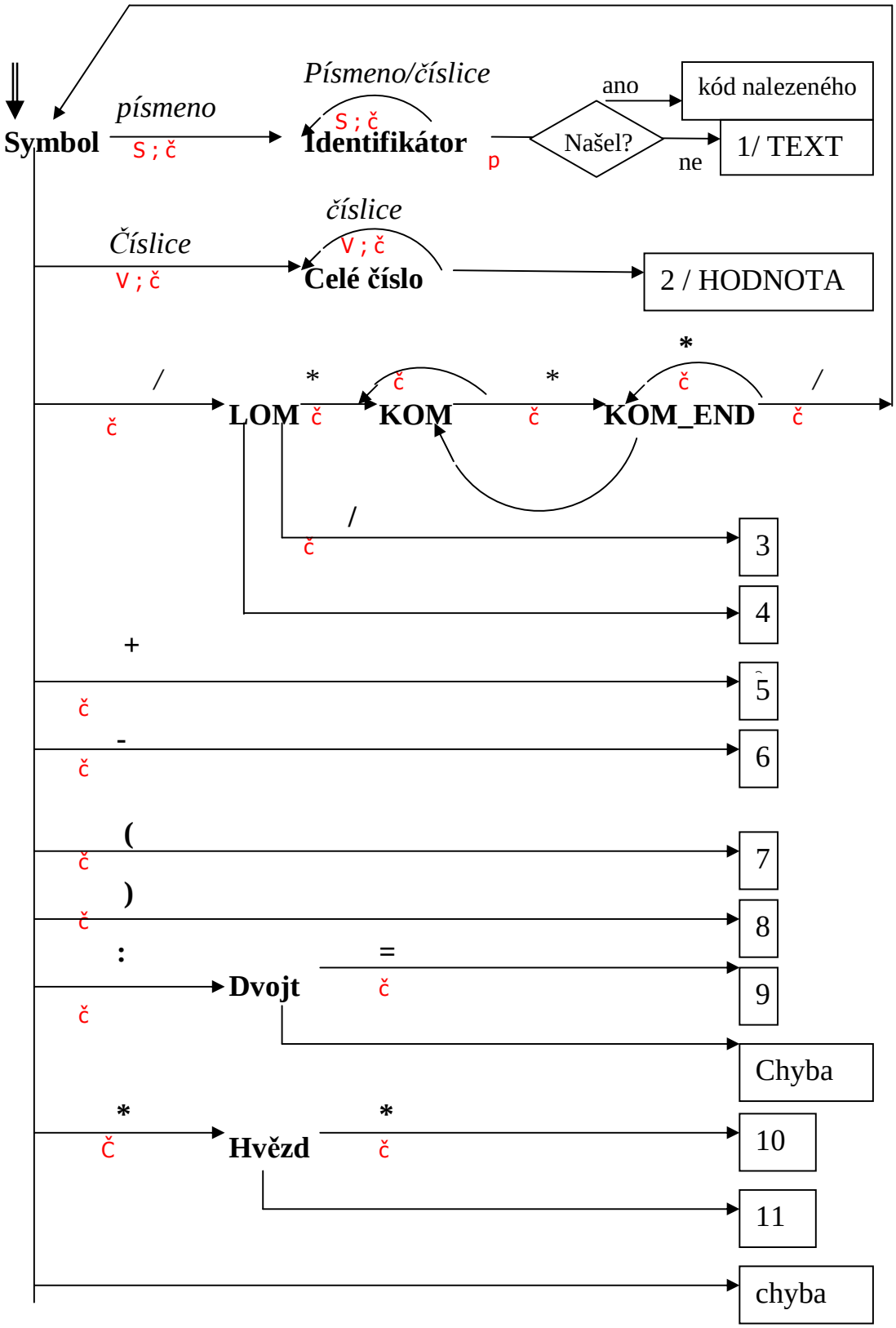
LA musí předat i atributy lexému, tj.
- u čísel jejich hodnotu
- u identifikátorů textový tvar (či adresu / ukazatel do tabulky)

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 ?


Symbol
písmeno
S ; č
Písmeno/číslice
S ; č
Identifikátor
p
Našel?
ano
kód nalezeného
ne
1/ TEXT
Číslice
V ; č
číslice
V ; č
Celé číslo
2 / HODNOTA
/
č
*
č
LOM
č
KOM
*
č
*
č
KOM_END
/
č
/
č
3
4
+
č
5
-
č
6
(
č
7
)
č
8
:
č
Dvojt
=
č
9
Chyba
*
Č
Hvězd
*
č
10
11
chyba

Nejednoznačnosti v lexikální analyze

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

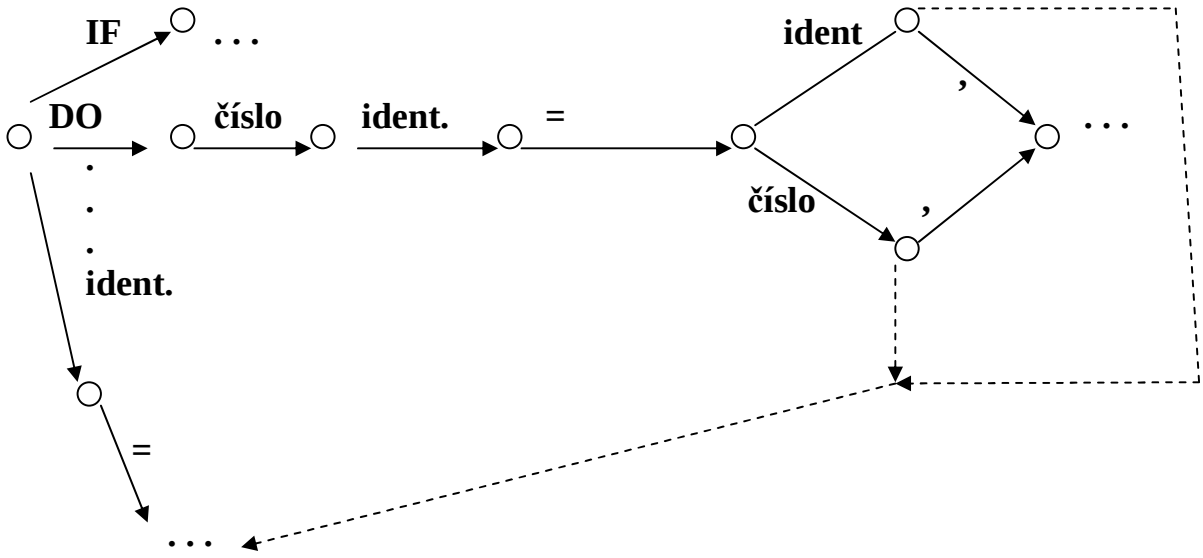
Pravidlo “hledej nejdelší symbol”

Nedokonalosti některých jazyků

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

Příkaz cyklu DO 10 I = 1 , 5

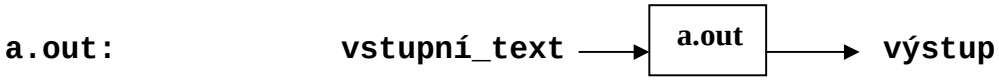
Vyžaduje nápovědu od syntaktického analyzátoru



LEX/FLEX (Unix/Linux)

Pro Windows lze stahnout "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




```
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**
yyleng	**" " 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 prvy
        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')]
```