

Nešvera

nesvera@fel.cvut.cz

cs/~nesvera

konzultace č. 10⁰⁰ - 11³⁰

<http://přibud.wz.cz>

- test. 12. říjen

- dom. práce

do 2-3 týdnů vymyšlené řešení

syntaktická analýza netriviálního jazyka
výsledek je na nás

- vstup zdrojůk → analyzujeme → výsledky

- překladač z 1 jazyka do jiného

- formátování podle pravidel, složitější obarvování
převod do HTML, přidat linky na fce --

do LATEXu

- práce s XML, SQL, grafické jazyky (vektorové formáty),
skriptovací jazyky

- symbolické manipulace .. derivování rozdání fce

- transformace gramatiky

- L-systémy - symboly interpretované - vykreslování
grafiky

jazyk L

abeceda A - konečná neprázdná množ. symbolů

řetězec - asi konečná posloupnost symbolů z abecedy

A^*

ϵ - prázdný řetězec

$L \subseteq A^*$ $\left\{ \begin{array}{l} \text{syntaxe} \\ \text{semantika} \end{array} \right.$ jazyk je podmnož. všech řetězců nad abecedou

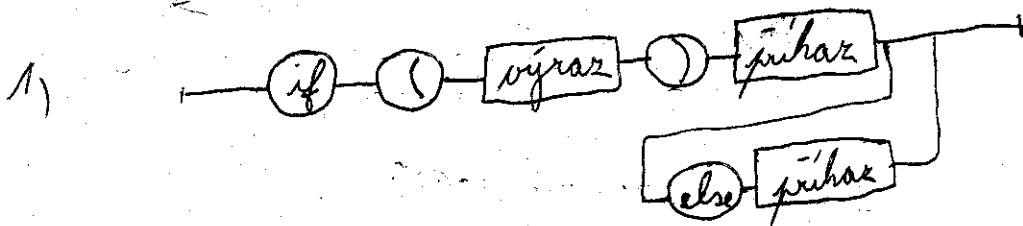
definováno pravidly

- syntaktický diagram

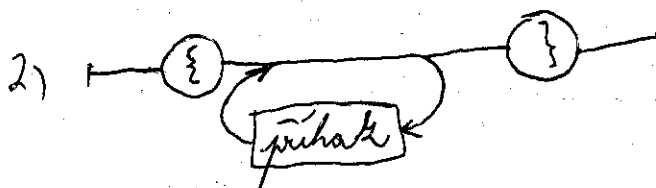
$$A^* = \bigcup_{i=0}^{\infty} A^i$$

výsledek křtěření

Podm. příkaz

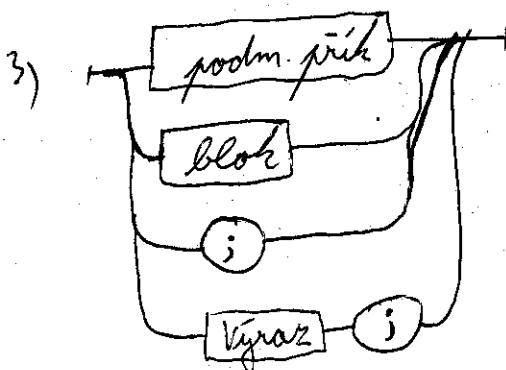


Blok



iterace

Příkaz



rekurziv

- gramatika

lineární řád

terminál - část jazyka, rozkládá se symbolickou konstrukcí

1) Podm. příkaz $\rightarrow$ if (výraz) Příkaz Zbytek
Zbytek $\rightarrow \epsilon$
Zbytek $\rightarrow$ else Příkaz

2) Blok $\rightarrow$ { Vnitřek }
Vnitřek $\rightarrow \epsilon$
Vnitřek $\rightarrow$ Příkaz Vnitřek

- E-BNF

Podm. příkaz = if (výraz) Příkaz [else Příkaz]

Blok = { {Příkaz} }

Příkaz = Podm. příkaz | Blok ; | Výraz ;

řetězení výrazů priorita než !
ne změnit záponkami ()

koncový automat
zásobníkový automat

Pr. Kalkulačka
sčítat, odčítat celá čísla

$$10 + 20 - 4 =$$

$$17 =$$

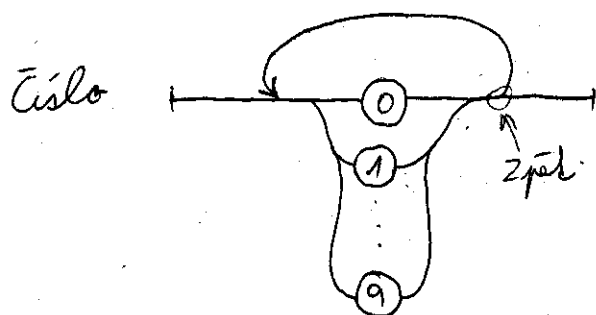
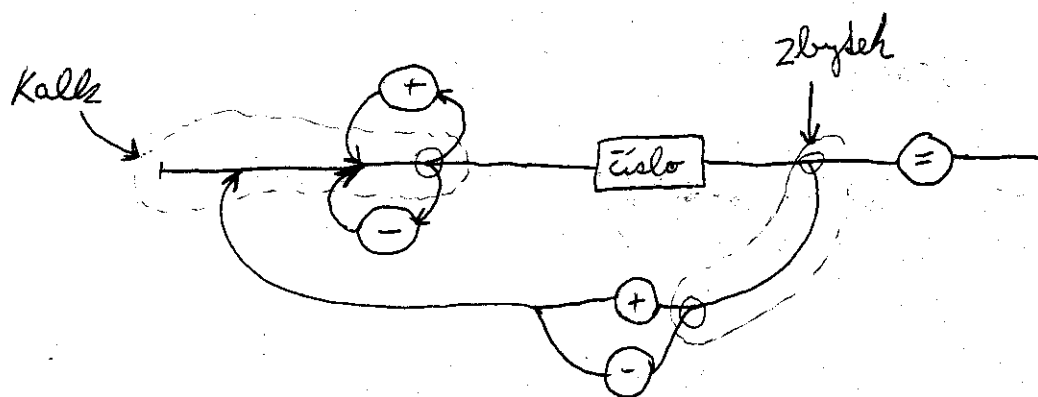
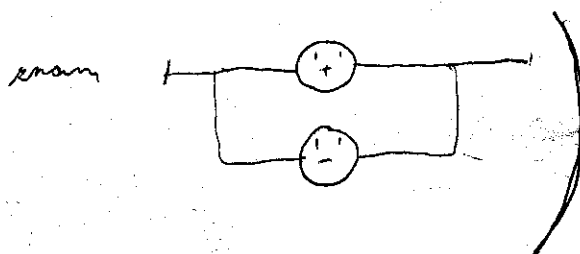
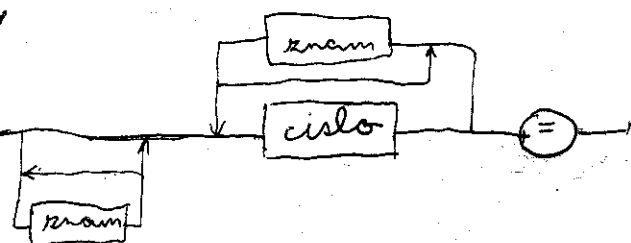
$$-160 =$$

$$+ - 5 =$$

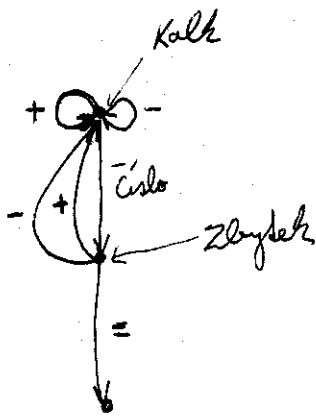
$$1 + - 6 - + 10 =$$

$$007 =$$

- syst. diagram
- gramatika
- E-BNF
- det. KA



Není zde žádná rekurze - není potřeba zásobníků
→ určité je to regulární jazyk



Kalk $\rightarrow$ - Kalk

Kalk $\rightarrow$ + Kalk

Kalk $\rightarrow$ Číslo Zbytek

Zbytek $\rightarrow$ - Kalk

Zbytek $\rightarrow$ + Kalk

Zbytek $\rightarrow$ =

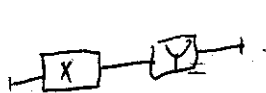
Číslo $\rightarrow$ 0 Zpět

9

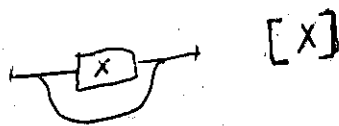
Zpět $\rightarrow$ číslo

Zpět $\rightarrow$ ε

E-BNF



XY



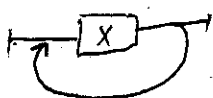
[X]



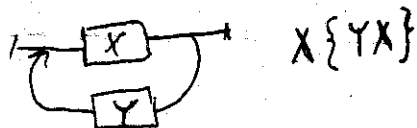
X|Y



{X}



X{X}



X{YX}

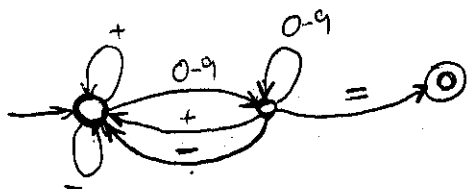
normálně výpisi pravidla křetězení, změny ()

$Kalk = \{+|- \} \text{ číslo } \{+|- \} \{+|- \} \text{ číslo } =$

$\text{Číslo} = \text{číslice} \{ \text{číslice} \}$

$\text{číslice} = 0|1| \dots |9$

KA



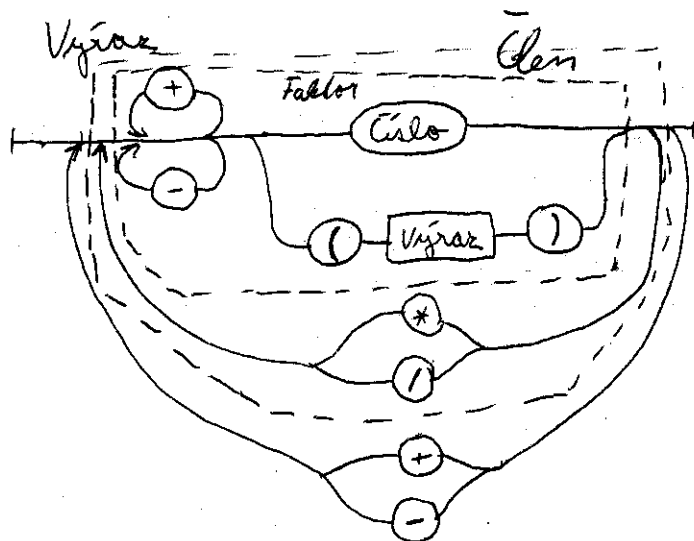
Vylepšení kalkulátory - násobení, dělení, záorky

jazyk nebude regulární
nebo použít KA

Kalk



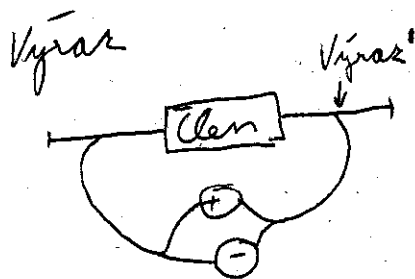
$Kalk \rightarrow Výraz =$



$Výraz \rightarrow +Výraz$
 $\rightarrow -Výraz$
 $\rightarrow \text{číslo Zbytek}$
 $\rightarrow (Výraz) Zbytek$

$Zbytek \rightarrow *Výraz$
 $\rightarrow /Výraz$
 $\rightarrow +Výraz$
 $\rightarrow -Výraz$
 $\rightarrow \epsilon$

Tato gramatika popisuje syntaxi, pro další operace nic nepřináší
 (ohledně priorit operací)



$Výraz \rightarrow \bar{Člen} \, Výraz'$
 $Výraz' \rightarrow \epsilon$
 $\rightarrow + \bar{Člen} \, Výraz'$
 $\rightarrow - \bar{Člen} \, Výraz'$

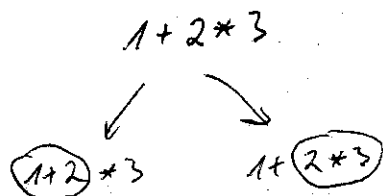
(3)

Výraz - operátory

- typy zápisu

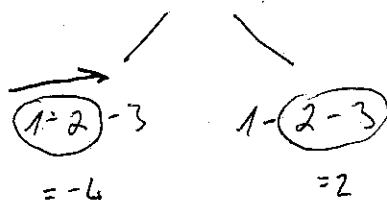
- unární
 - prefix: 0- !, ~, -, x, &, ++, --
 - postfix: -0 ++, --
 - outfix: (-) (závorhy)
- binární
 - infix: 0- +, <, &&, &, =
 - mixfix: [-], (-) (indexace, předyporání)
 - prefix: 0--
 - postfix: --0
- ternární: - ? - : - (podm. výraz)

- priorita
 určuje, kterému operátoru náleží společný operand



- asociativita

- levá: +, *, ---
- pravá: =, x = y = 0
- řádná: 1-2-3 by byla chyba



Pr Konst. aritm. výrazy

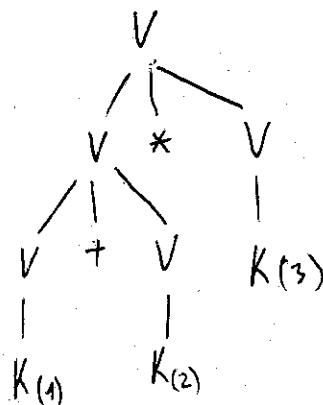
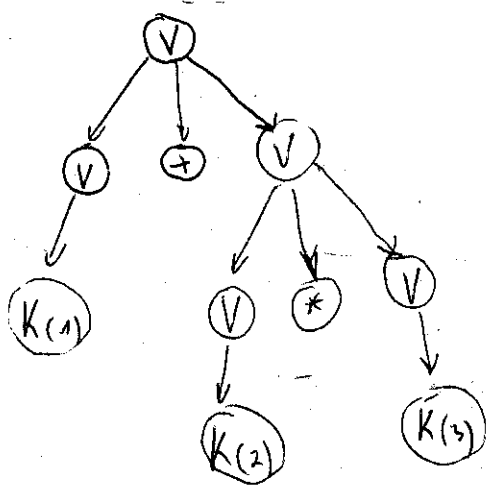
- konst
- un. -
- bin +, *
- záorky

1) $V \rightarrow K \mid -V \mid V+V \mid V*V \mid (V)$

derivací strom

kóřenový orientovaný strom s uzly hodnotenými symboly gramatiky
s kořenem startovací symbol

$1+2*3$



v listech terminály

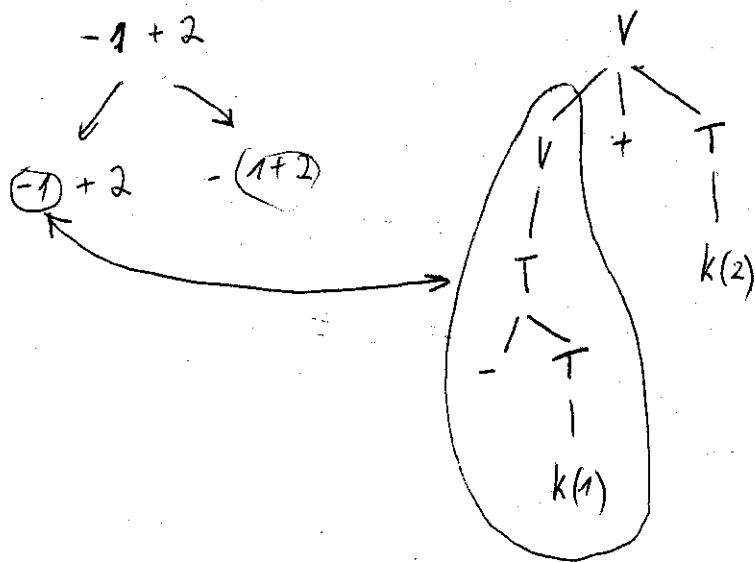
jednoznačná gramatika - ke každému řetězci jazyka
existuje právě jeden derivací strom

2)

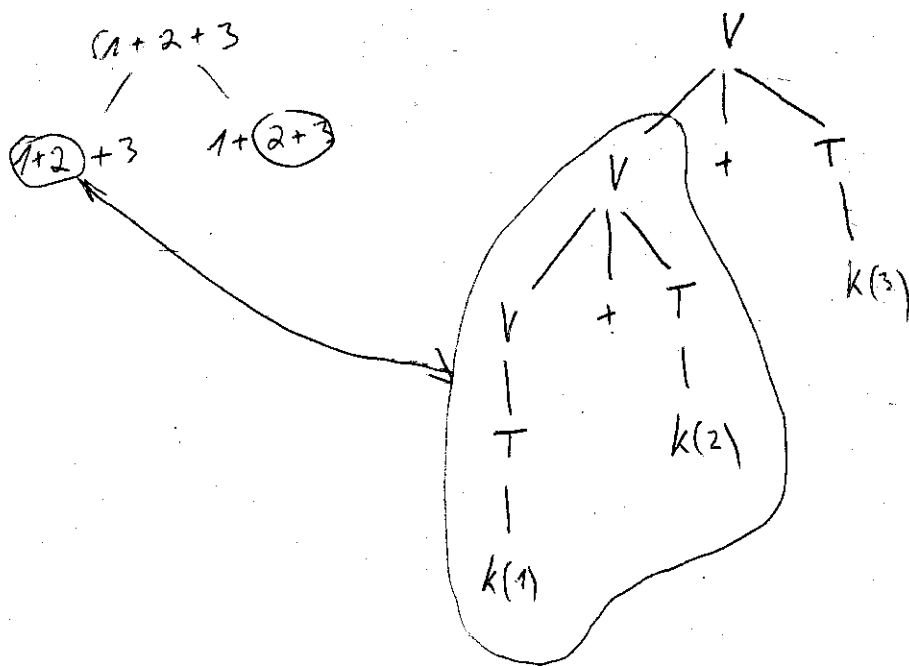
$$(V \rightarrow k \mid -V \mid V+T \mid V*T \mid (V)) \quad V \rightarrow T \mid V+T \mid V*T$$

$$(T \rightarrow k \mid -V \mid (V)) \quad T \rightarrow k \mid -T \mid (T)$$

solo je jednoznačná gramatika
(ale je obtížně dokázat)



→ unární = má vyšší prioritu než +, *



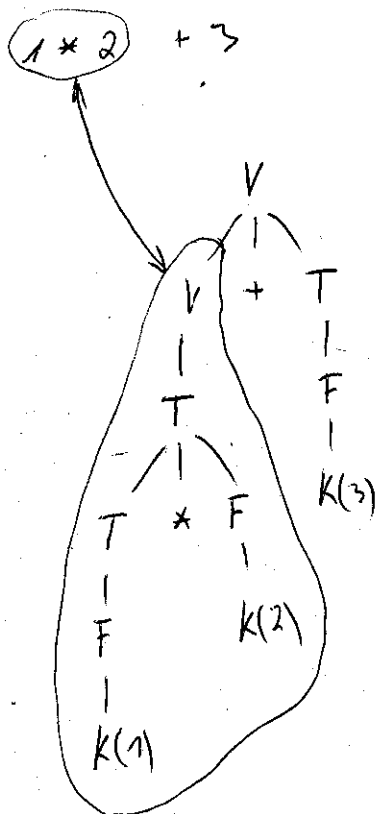
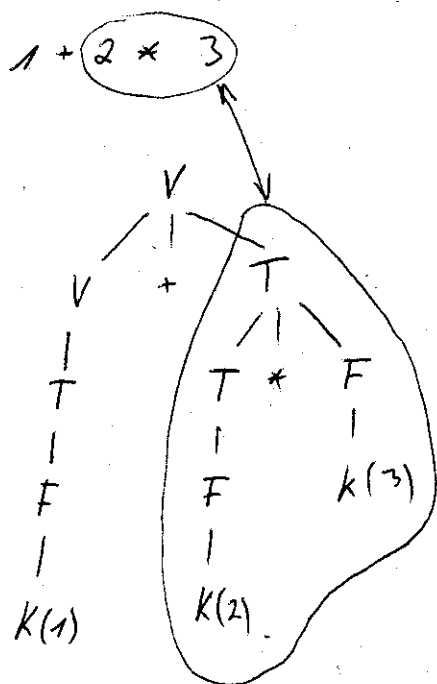
→ levá asociativita

na jedné úrovni priorit musí být stejná asociativita

3)

$$\begin{aligned} V &\rightarrow T \mid V + T \\ T &\rightarrow F \mid T * F \\ F &\rightarrow k \mid -F \mid (V) \end{aligned}$$

rozlišení priorit operatorů



Pr. konst. aritm. výraz
konst., un. prefix -, bin. & prefix -

1) $V \rightarrow k \mid -V \mid -VV$

-- 12

$-(-12) - (-1)2$

$$\begin{aligned} V &\rightarrow k \mid -V \mid -TV \\ T &\rightarrow k \mid -TV \end{aligned}$$

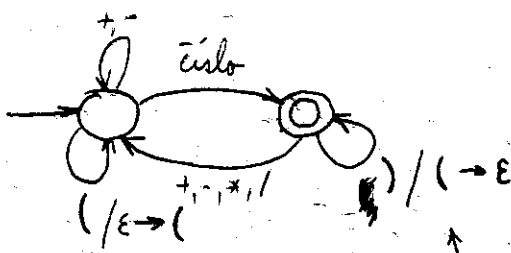
Podobně

$\text{Přikaz} \rightarrow x=y; \mid \text{if}(\bar{()}) \text{ Přikaz}$

$(\text{if}(\bar{()}) \text{ Přikaz} \text{ else Přikaz})$

$\text{if}(\bar{()}) \text{ if}(\bar{()}) x=0; \text{ else } x=1$

halk. k cv. 3



→ determ. kās. automa

reex. algoritmus na převod bezkon. gram. na det. kās. automa
 ↑
 na károbních kávoch
 ↑
 ke károbních odebrat kávoch, musí tam být

PH

Přikaz → if (podm) Přikaz Elsečís

> FIRST x FIRST ✓ OK

Přikaz → ;

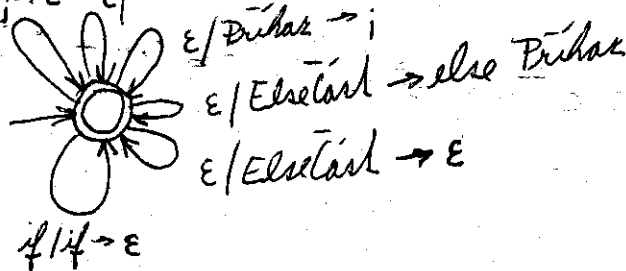
Elsečís → else Přikaz

> FIRST x FIRST ✓ OK

Elsečís! → ε

provaní exace

$j/i \rightarrow \epsilon$ $\epsilon/Přikaz \rightarrow if (Podm) Přikaz Elsečís$



$if/i \rightarrow \epsilon$

$\epsilon/Přikaz \rightarrow ;$

$\epsilon/Elsečís \rightarrow else Přikaz$

$\epsilon/Elsečís! \rightarrow \epsilon$

provede se podle dalšího symbolu na vstupu

→ nedeterm. károb. automa

na kakov. stor. / zgodov.

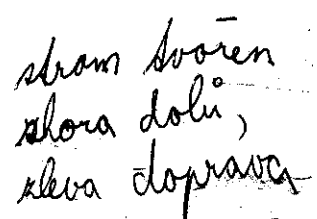
_____ 11 _____

3

1. 1-

Char

)



ref. bank
waiver

větná forma

$$\text{FIRST}(X) = \{a : a \in T, X \Rightarrow^* a\beta, \alpha, \beta \in (NUT)^*\} \cup \{\epsilon : X \Rightarrow^* \epsilon\}$$

$$\text{FOLLOW}(X) = \{ a : a \in T, S \xRightarrow{*} \alpha X a \beta; \alpha, \beta \in (N \cup T)^* \} \cup \{ \epsilon : S \xRightarrow{*} \alpha X; \alpha \in (N \cup T)^* \}$$

(very bady FOLLOW(S) = ϵ , κ $S \xrightarrow{*} \epsilon S$)

počrtať Pr.

	FIRST	FOLLOW
Príkaz	<u>if</u> i	<u>else</u> ε
ElseČasť	<u>else</u> ε	<u>else</u> ε

holice FIRST x FOLLOW

horec okupce (už nie mení)!

FOLLOW (Príkaz)

Pr Podm. príkaz PHP

if (podm):

... } Príhazy

{ elseif (podm): }

[else :]

endif

Príkaz → if (podm): Príhazy ElseifČasť ElseČasť endif !;

Príhazy → Príkaz Príhazy

Príhazy → ε

ElseifČasť → ε

ElseifČasť → elseif (podm): Príhazy ElseifČasť

ElseČasť → ε

ElseČasť → else : Príhazy

	FIRST	FOLLOW
Príkaz	<u>if</u> i	<u>if</u> i ε <u>endif</u> <u>else</u> <u>endif</u>
Príhazy	<u>if</u> i ε	~~endif~~ <u>else</u> <u>elseif</u>
ElseifČasť	<u>elseif</u> ε	<u>else</u> <u>endif</u>
ElseČasť	<u>else</u> ε	<u>endif</u>

6
přeměna návrh gram. příkazu - aritm. výraz (priority, assoc.),
 případně transform. na LL(1), rozhl. tabulka

P_R $P \rightarrow \overset{1}{\text{if}} (\text{podm}) \overset{2}{P} \overset{3}{E} \overset{4}{|};$
 $E \rightarrow \text{else } P \mid \epsilon$

je LL(1)? kolise FIRST-FOLLOW

$\text{FOLLOW}(E) = \{ \epsilon, \text{else} \}$

gramatika musí být jednoznačná, aby se dala
 transformovat na LL(1)

$P \rightarrow \text{if} (\text{podm}) R \text{ else } P \mid \text{if} (\text{podm}) P \mid ;$
 $R \rightarrow \text{if} (\text{podm}) R \text{ else } P \mid ;$

[$P \rightarrow \text{if} (\text{podm}) S \mid ;$

$S \rightarrow R \text{ else } P \mid P$

[$R \rightarrow \text{if} (\text{podm}) R \text{ else } P \mid ;$

$S \rightarrow \text{if} (\text{podm}) R \text{ else } P \text{ else } P \mid ; \text{ else } P \mid$

$\text{if} (\text{podm}) S \mid ;$

~~TH~~ $\rightarrow$

[$S \rightarrow \text{if} (\text{podm}) T \mid ; U$

$T \rightarrow R \text{ else } P \text{ else } P \mid S$

$U \rightarrow \text{else } P \mid \epsilon$

nebo převést na LL(1)

poslední verze

	if	;	else	ϵ
7	1	2		
E			3	4

byli ϵ a FOLLOW
reprezentují konec vstupu
POZOR!

kolice FIRST-FOLLOW
dáme přednost pravidlu 3
není to rozkladová tabulka gramatiky,
ale tedy to funguje

Pr Podm. příkaz PHP (x minuta)

- (1) Příkaz $\rightarrow$ if (podm): Příkaz Else část Else část endif
- (2) Příkaz $\rightarrow$ Příkaz Příkaz
- (3) Příkaz $\rightarrow \epsilon$
- (4) Else část $\rightarrow \epsilon$
- (5) Else část $\rightarrow$ elseif (podm): Příkaz Else část
- (6) Else část $\rightarrow \epsilon$
- (7) Else část $\rightarrow$ else: Příkaz
- (8) Příkaz $\rightarrow ;$

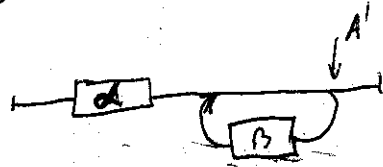
	if	elseif	elseif	endif	ϵ	;
Příkaz	1					8
Příkaz	2	3	3	3		2
Else část		5	4	4		
Else část			7	6		

$A \rightarrow AB | \alpha$ odstranění levé rekurze

kolize FIRST-FIRST

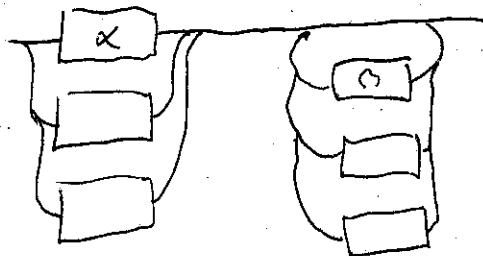
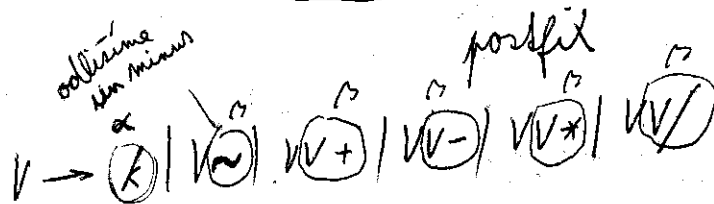
potřebujeme α ~~levo~~ vyderivovat předtím než B ,
pak i kolize FIRST-FOLLOW

$$A \Rightarrow^* \alpha \{B\}$$



$$\begin{cases} A \rightarrow \alpha A' \\ A' \rightarrow \epsilon | BA' \end{cases}$$

Př konst., un., bin., $+, -, *, /$



Odstraníme levou rekurzi, něco vyřkneme
a jsme doma

$$V \rightarrow KV'$$

$$V' \rightarrow \epsilon | \sim V' | V+V' | V-V' | V*V' | V/V'$$

$$\begin{aligned} (1) \quad V &\rightarrow KV' \\ V' &\rightarrow \overset{(2)}{\epsilon} | \overset{(3)}{\sim V'} | \overset{(4)}{VV''} \\ V'' &\rightarrow \overset{(5)}{+V'} | \overset{(6)}{-V'} | \overset{(7)}{*V'} | \overset{(8)}{/V'} \end{aligned}$$

$$\text{FOLLOW}(V') = \{ \epsilon, +, -, *, / \}$$

	ϵ	+	-	*	/	k	$\sim$
V						1	
V'	2	2	2	2	2	4	3
V''		5	6	7	8		

Př na doma

deklarace v C (část)

Dekl $\rightarrow$ ident

Dekl $\rightarrow$ *Dekl

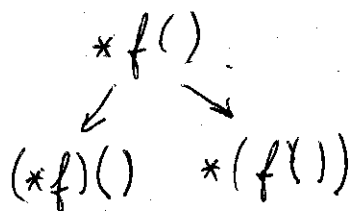
Dekl $\rightarrow$ Dekl []

Dekl $\rightarrow$ Dekl ()

Dekl $\rightarrow$ (Dekl)

nejednoznačný

(deklarátor funkce)



odstranit nejednoznačnost

pok levou rekurzi, pak snad $((f))$

a rozkl. tabulku

7

Pr

~~D~~ $\rightarrow$ ident

~~D~~ $\rightarrow$ *D

D $\rightarrow$ D()

D $\rightarrow$ D[]

D $\rightarrow$ (D)

pouze odstranění levé rekurze

Postfix větší priorita

~~D~~

D $\rightarrow$ id | *D | E() | E[] | (D)

E $\rightarrow$ id | E() | E[] | (D)

D $\rightarrow$ E | *D

E $\rightarrow$ id | E() | E[] | (D)

$\rightarrow$ jednoznačná gram.
(lze převést na LL(1))

D $\rightarrow$ E | *D

E $\rightarrow$ id | T | (D)T

T $\rightarrow$ ϵ | ()T | []T

E je vpravo jen jednou,
dosadíme za něj

(1)

1) D $\rightarrow$ *D

2) D $\rightarrow$ idT

3) D $\rightarrow$ (D)T

4) T $\rightarrow$ ϵ

5) T $\rightarrow$ ()T

6) T $\rightarrow$ []T

FOLLOW(T) = { }, ϵ }

(2) $D \rightarrow *DT$ *oddělník*
 $D \rightarrow id T$
 $D \rightarrow (D) T$
 $T \rightarrow \epsilon$
 $T \rightarrow () T$
 $T \rightarrow [] T$

$FOLLOW(T) = \{), \epsilon, (, [\}$

$\rightarrow$ holíme FIRST-FOLLOW

Rozkladová tabulka pro (1)

	id	(	)	[	]	*	ϵ
D	2	3				1	
T		5	4	6			4

`typedef enum TERM { ID, LKZ, PKZ, LHZ, PHZ, HV, EOI } TERM;`

`TERM insymbol;`

`void chisymbol();`

`void srovnani (TERM ocekavan) {`

`if (ocekavan == insymbol) chisymbol();`

`else chyba(ocekavan); }`

*vypisani chyby a konec
exit() ne nula*

`void D(void) {`

`switch (insymbol) {`

`case HV: // D  $\rightarrow$  *D`

`srovnani(HV);`

`D();`

`break;`

`case ID: // D  $\rightarrow$  id T`

`srovnani(ID);`

`T();`

`break;`

case LK2 : // $D \rightarrow (D)T$

 Grownani (LK2);

 D();

 Grownani (PK2);

 T();

 break;

default: Chyba (); }

}

case LK2 : // $T \rightarrow \epsilon$

case EOI: break;

int main()

{

 insymbol();

 D();

 if (insymbol != EOI) Chyba(...);

 return 0;

8

Překlád výrazů
Interpretace (výhodnocení)

	Prefix	Infix	Postfix
Prefix	—	✓ ² kávorky	✓
Infix	x	—	✓
Postfix	x	x	—

Překládová gramatika - zajímá nás pouze LL(1)

- přidáv^{konc.} množinám vstupních symbolů disjunkční a množ. vstupních symbolů
- tyto vst. symboly přidány do pravých stran pravidel

Prefix → Postfix

	vst.	①, ~, +, -
$V \rightarrow k$	$V \rightarrow k①$	KONST-
$V \rightarrow \sim V$	$V \rightarrow \sim V \sim$	UNMINUS
$V \rightarrow + VV$	$V \rightarrow + VV \oplus$	PLUS
$V \rightarrow - VV$	$V \rightarrow + VV \ominus$	MINUS

void V() {

switch (insymbol)

case KONST:

~~cout << srovnani * KONST;~~

cout << srovnani - KONST() << " "; break;

case UNMINUS:

srovnani (UNMINUS);

V();

cout << "~" break;

case PLUS:

return (PLUS);

V();

V();

cout << "+";

break;

ca

Prefix $\rightarrow$ Infix

$V \rightarrow k \textcircled{+}$

$V \rightarrow \sim \textcircled{-} V$

$V \rightarrow +(V \textcircled{+} V)$

$V \rightarrow -(V \textcircled{-} V)$

výst. a. $\textcircled{+}, \textcircled{-}, \textcircled{+}, (,), \textcircled{*}$

$V \rightarrow *(V \textcircled{*} V)$

výstup

* ~ + 1 2 3

* ~ + 1 2 3

~ + 1 2 3

~ + 1 2 3

~ + 1 2 3

+ 1 2 3

+ 1 2 3

+ 1 2 3

1 2 3

1 2 3

1 2 3

2 3

2 3

2 3

2 3

násobník
V

$*(V \textcircled{*} V)$

$(V \textcircled{*} V)$

$V \textcircled{*} V$

$\sim \textcircled{-} V \textcircled{*} V$

$\textcircled{-} V \textcircled{*} V$

$V \textcircled{*} V$

$+(V \textcircled{+} V) \textcircled{*} V$

$(V \textcircled{+} V) \textcircled{*} V$

$V \textcircled{+} V) \textcircled{*} V$

$k \textcircled{+} V) \textcircled{*} V$

$\textcircled{+} V) \textcircled{*} V$

$\textcircled{+} V) \textcircled{*} V$

$V) \textcircled{*} V$

$k \textcircled{+}) \textcircled{*} V$

výstup

ϵ

ϵ

ϵ

$($

$($

$($

$($

$($

$($

$($

$($

$($

$($

$($

$($

$($

3	(k ₂) ⊗ V)	(-(1+2)
3	) ⊗ V)	(-(1+2
3	⊗ V)	(-(1+2)
3	V)	(-(1+2)*
3	k ⊗)	(-(1+2)*
ε	(k ₂)	(-(1+2)*
ε	)	(-(1+2)* 3
ε	ε	(-(1+2)* 3)

Infix → Postfix

$$\begin{pmatrix} V \rightarrow TV' \\ V' \rightarrow \epsilon \mid +TV' \mid -TV' \end{pmatrix}$$

$$V \rightarrow V+T \oplus \mid V-T \ominus \mid T$$

$$T \rightarrow T * F \otimes \mid T / F \oslash \mid E$$

$$F \rightarrow k \otimes \mid -F \odot \mid (V)$$

výst: ⊕, ⊖, ⊗, ①, ②

odstranění levé rekurze → LL(1)

Infix → Prefix

$$V \rightarrow \oplus V+T \mid \ominus V-T \mid T$$

$$T \rightarrow \otimes T * F \mid \oslash T / F \mid F$$

$$F \rightarrow k \otimes \mid - \odot F \mid (V)$$

někde převést na LL(1)

šlo by jako nedeterministický automat -
exponenciální složitost

řešení - přetvořit vnitřní formu jako strom a inorder průchod

Vyhodnocení

$V \rightarrow TV'$

$V' \rightarrow \varepsilon \mid +T \odot V' \mid -T \odot V'$

⋮

$F \rightarrow k \odot \mid -F \odot \mid (V)$

```
void push(int);  
int pop();
```

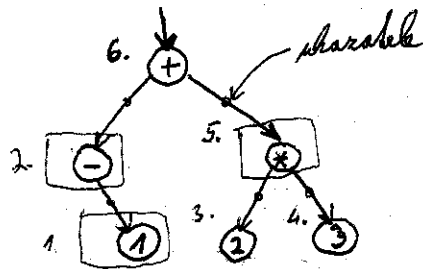
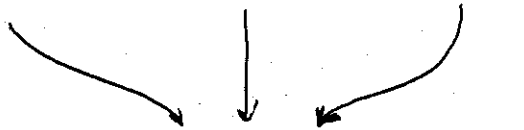
... sémantický strojník

```
void F() {  
    switch (insymbol) {  
        case KONST:  
            push(srovnani_KONST());  
            break;  
        case MINUS:  
            srovnani(MINUS)  
            F();  
            push(-pop());  
            break;  
        case LZAV:  
            srovnani(LZAV);  
            V();  
            srovnani(PZAV);  
            break;  
    }  
}
```

na konci předání vstupu a na kás. pouze výsledek

9

$+ \sim 1 * 2 3$ $-1 + 2 * 3$ $1 \sim 2 3 * +$



1... 6. pořadí vytvoření
→ postorder
jako v postfixu

Uzel (abstr. třída)

Konst
int hodn
Konst (int h)

UnOp
Uzel * Op
UnOp (Uzel * O)
UnMinus

BinOp
Uzel * Lev, * Prav
BinOp (Uzel * L, Uzel * P)
Plus Krat Deleno

při přidání operace
se nemusí upravovat
stávající kód

každé třídě přistupují tabulka virtuálních metod, na
ni ukazatel v každé instanci (role uzlu)
konstruktory se nestědí

Uzel * koren = new Plus (new UnMinus (new Konst (1)),
new Krat (new Konst (2), new Konst (3))) ;

při vytvoření stromu jeho překlad do postfixu

$V \rightarrow \tilde{k} \oplus$
 $V \rightarrow \sim \tilde{V} \ominus$
 $V \rightarrow + V V \oplus$
 $V \rightarrow - V V \ominus$
 $V \rightarrow * V V \otimes$
 $V \rightarrow / V V \odot$

Uzel * V() {
 switch (insymb) {

case KONST:

~~return new Konst (insymb);~~

return new Konst (srovnani_KONST);

case UNMINUS:

srovnani (UNMINUS);

return new UnMinus (V());

case PLUS:

srovnani (PLUS); Uzel * Pom = V();

return new Plus (~~Pom~~, V());

case MINUS:

srovnani (MINUS); Uzel * Pom = V();

return Minus (~~Pom~~, V());

}}

• vrácením přes parametry
 také lze dávat na sémantický
 násobník

nevíme, jak by překladač
 vyhodnotoval (V(), V())

int main () {
 try { chSymbol();

Uzel * koren = V();

if (insymb != EOI) throw " --- ";

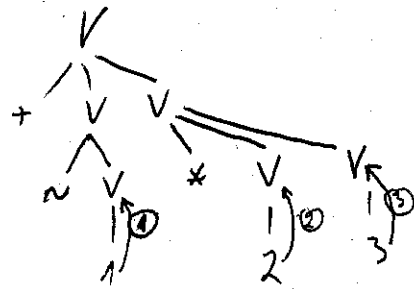
< cout << koren -> vyhodnot ();

koren -> doPostfix (cout);

} catch (...) { ... }

pokračování příště ...

+ ~ 1 * 2 3



add.

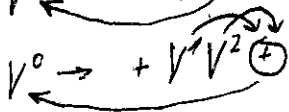
symbol	Typ	Dat
V	s	/
k	rod	/



$V.s = \text{new Konst (k.rod)}$

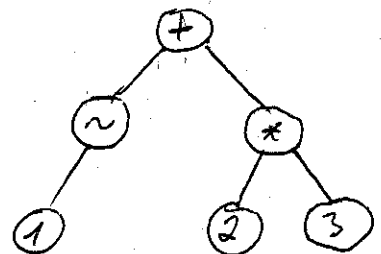
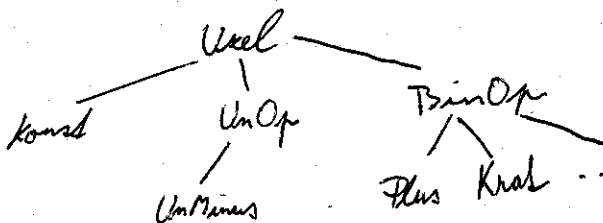


$V^0.s = \text{new UnMinus (V^1.s)}$



$V^0.s = \text{new Plus (V^1.s, V^2.s)}$

(10)



- konstruktor
- destruktor
- vyhodnot
- prefix
- infix

- generuj $\Rightarrow$ ASM

bliv.
abstract class Uzel {
 abstract public int vyhodnost();
 abstract public String prefix();
 abstract public String infix();
}

class Konst extends Uzel {

int hodnota;

public Konst (int hodnota) {

this.hodnota = hodnota;

}

public int vyhodnost () {

return hodnota;

}

public String prefix () {

return String.valueOf(hodnota);

}

public String infix () {

return prefix();

}

}

```

public abstract class BinOp extends Uzel {
protected Uzel levy, pravy;
public BinOp (Uzel ll, Uzel p) {
    levy = l;
    pravy = p;
}
}

```

```

public class Plus extends BinOp {
public Plus (Uzel l, Uzel p) {
    super (l, p);
}
public intint vyhodnot () {
    return levy.vyhodnot () + pravy.vyhodnot ();
}
}

```

```

public String prefix () {
    return "+" + levy.prefix () + " " + pravy.prefix ();
}
}

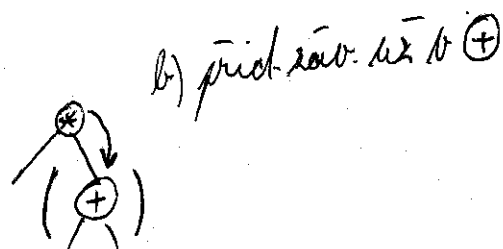
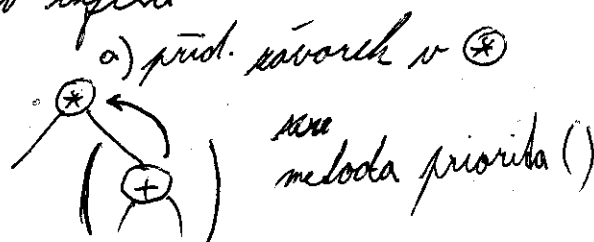
```

```

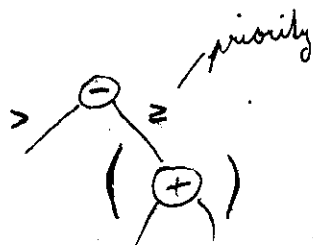
public String infix () {
    return levy.infix () + "+" pravy.infix ();
}
}

```

návorky v infixu



asociativita



"ADD AX, BX"

$AX \leftarrow AX + BX$

class Uzel

abstract public void generuj();

class Plus

public void generuj() {

levy.generuj();

System.out.println("MOV PUSH AX");

pravy.generuj();

... ("MOV BX, AX");

... ("POP AX");

... ("ADD AX, BX");

}

že není nutné

~ lze odstranit vložným
pořadím generuj()

Postfix

$$V \rightarrow k V'$$

$$V' \rightarrow \varepsilon$$

$$V'^0 \rightarrow \neg V'^1$$

$$V' \rightarrow V V''$$

$$V'' \rightarrow + \oplus V'$$

$$V'' \rightarrow - V'$$

$$V'' \rightarrow * V'$$

$$V'' \rightarrow / V'$$

$$V'.oper = k.hodn, V.hodn = V'.hodn$$

$$V'.hodn = V'.oper$$

$$V'^1.oper = - V'^0.oper, V'^0.hodn = V'^1.hodn$$

$$V''.lezy = V'.oper, V''.pravy = k.hodn, V'.hodn = V''.hodn,$$

$$V'.oper = V''.lezy + V''.pravy, V'.hodn = V''.hodn$$

	synt.	titl
V	hodn	
V'	hodn	oper
V''	hodn	lezy, pravy
k	hodn	

doporučeno vytvářet rekursivní sestup

$$V'.oper = \text{new Plus}(V''.lezy, V''.pravy)$$

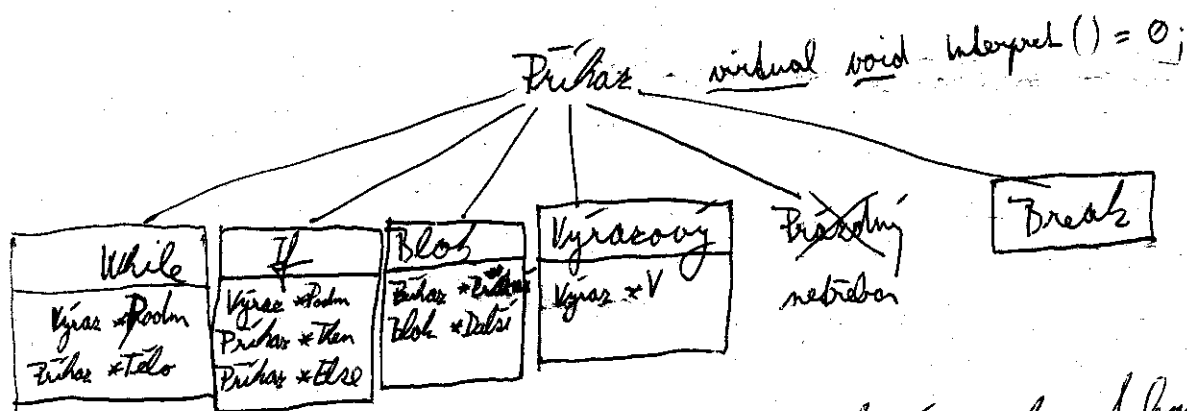
- při generování vnitřní formy

Príkazy

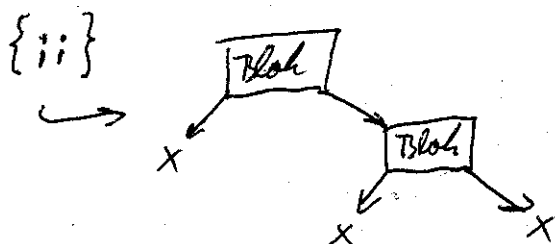
- syntetická forma
- atributová gramatika
- interpret
- generování ASM

AST (Abstract Syntax Tree)

- while
- if
- blok
- výrazový př. (končí ;)
- prázdňový př.
- break



Prázdňový příkaz reprezentován nulovým znakem



$\text{Přikaz} \rightarrow \text{while (Výraz) Přikaz}$ While
 $\text{Přikaz} \rightarrow \text{if (Výraz) Přikaz Else}$ If
 $\text{Else} \rightarrow \text{else Přikaz}$
 $\text{Else} \rightarrow \epsilon \quad \emptyset = \text{NULL}$
 $\text{Přikaz} \rightarrow \{ \text{Přikazy} \}$
 $\text{Přikazy} \rightarrow \text{Přikaz Přikazy}$ Block
 $\text{Přikazy} \rightarrow \epsilon \quad \emptyset = \text{NULL}$
 $\text{Přikaz} \rightarrow \text{Výraz};$ Výrazový
 $\text{Přikaz} \rightarrow i \quad \emptyset = \text{NULL}$
 $\text{Přikaz} \rightarrow \text{break};$ Break

```

void While::Interpret() {
    try { while (Podm → Vyhodnot())
        if (Telo) Telo → Interpret();
    } catch (exitBreak) {}
}

void If::Interpret() {
    if (Podm → Vyhodnot()) {
        if (Then) Then → Interpret();
    } else {
        if (Else) Else → Interpret();
    }
}

```

break na špatném místě → výjimka projde
 až do main → oznámení chyby

```

void Blok::Interpret () {
    for (Blok *pom = this; pom != null; pom = pom->Dalsi)
        if (pom->Prvni) pom->Prvni->Interpret();
}

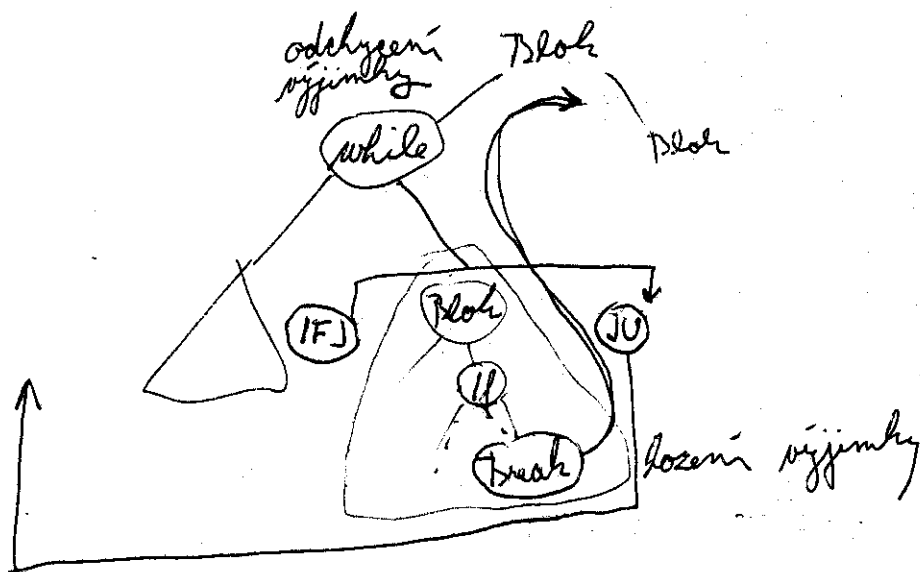
```

```

void Vyrazovj ::
{ V -> Vyhodnot (); }

```

vedlejší efekt
 - přičtení
 - inkrement
 - volání ~~for~~



```

void Break::
{ throw excBreak; }

```

časemka 90 min

papír, tuška, klava, skripty přednášky, slajdy přednášky

transformace na $LL(1)$, reh. sestup,

lehké atributy (treba AST), interpretace, vyhodnocování,
překlad (tedy všechno)

12

3.1.06

Deklarace

identifikátory

- typová kontrola

- přidělování paměti (adresace)

Tabulka symbolů

- jméno

- typ

- adresa(ce)

- délka

- druh

proměnná

funkce, procedura, metoda

parametr

konstanta, výřet. literál

položka (sámanus, třídy...)

- úroveň vnoření

pr.

char s[10][5]

čtyři řádky v tabulce symbolů

char předem, pak s, pole 10 polí, pole 5 znaků

přidávání do tabulky symbolů - házíme deklarace

odebírání - konec platnosti prom. (lokální prom. v bloku)

tabulka prohledávána od konce - od nejhlubší

úrovně kanonění - lokální prom. přebíhá globální

PA

přítaz bloku

deklarace $\left\{ \begin{array}{l} \text{int (4B)} \\ \text{double (8B)} \end{array} \right.$

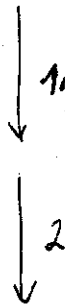
{ int x, y ;

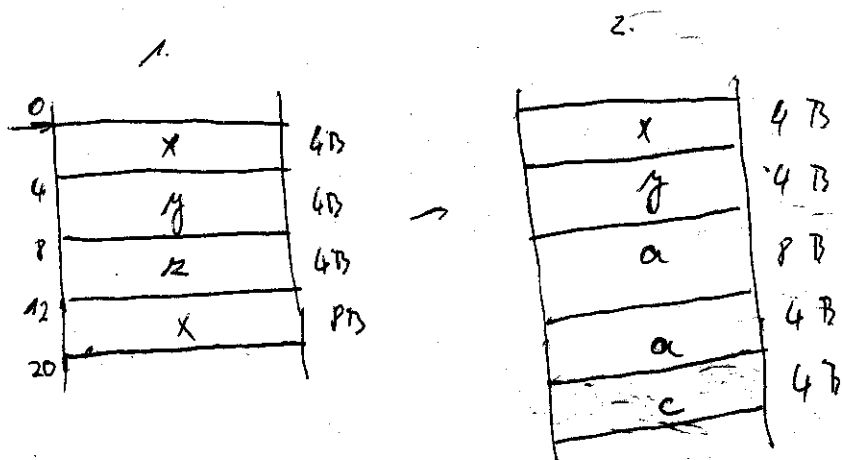
{ int x, double x ; }

double a ;

{ int a, c ; }

}





jméno	adresa	délka	úroveň
x	0	4	1
y	4	4	1
z	8	4	2
x	12	8	2
a	8	8	1
a	16	4	2
c	20	4	2

klíč (jméno, adresa, délka, úroveň)
 zruš (úroveň)

Blok $\rightarrow \{ \text{Primary} \}$ Zrús

Primary $\rightarrow \varepsilon$

$\rightarrow$ Dekl Primary

$\rightarrow$ Blok Primary

Dekl $\rightarrow$ int Ident Vloz Zbytek

double Ident Vloz Zbytek

Zbytek $\rightarrow$, Ident Vloz Zbytek

$\rightarrow$;

	dekl	zbytek
Blok	úroveň, da	sa
Primary	úroveň, da	sa
Dekl	úroveň, da	sa
Zbytek	úroveň, dekl, da	sa
Vloz	jméno, dekl, dekl, úroveň	
Zrús	úroveň	
Ident		jméno

$B \rightarrow \{P\}$ zruš

$P.\text{úroveň} = B.\text{úroveň} + 1$, $Z.\text{úroveň} = B.\text{úroveň} + 1$, $P.da = B.da$,
 $B.sa = P.sa$

$P \rightarrow \epsilon$

$P.sa = P.da$

$P^0 \rightarrow D P^1$

$D.\text{úroveň} = P^0.\text{úroveň}$, $P^1.\text{úroveň} = P^0.\text{úroveň}$, $D.da = P^0.sa$, $P^1.da = D.sa$,
 $P^0.sa = P^1.sa$

$P^0 \rightarrow B P^1$

$B.\text{úroveň} = P^0.\text{úroveň}$, $P^1.\text{úroveň} = P^0.\text{úroveň}$, $B.da = P^0.da$, $P^1.da = P^0.da$,
 $P^0.sa = \max(B.sa, P^1.sa)$

$D \rightarrow \text{int ident} \text{ (vloz) } Z$ ⁺⁴

$\text{vloz.jmeno} = \text{ident.jmeno}$, $\text{vloz.delka} = 4$, $\text{vloz.úroveň} = D.\text{úroveň}$
 $Z.\text{úroveň} = D.\text{úroveň}$, $Z.delka = 4$

$D \rightarrow \text{double ident} \text{ (vloz) } Z$ ⁺⁸

$Z \rightarrow , \text{ ident} \text{ (vloz) } Z^1$ ^{+Z⁰.delka}

$\text{vloz.jmeno} = \text{ident.jmeno}$, $\text{vloz.delka} = Z^0.delka$, $\text{vloz.úroveň} = Z^0.\text{úroveň}$
 $Z^1.delka = Z^0.delka$, $Z^1.\text{úroveň} = Z^0.\text{úroveň}$

$Z \rightarrow ;$

pro start. symbol

$B.\text{úroveň} = 0$

$B.da = 0$

přítě možnost odebrat semestrálku, jinak nepovinné
 nárok v 310+V okolí