PROGRAMOVÁNÍ – FORMÁLNÍ – JAZYKY – ÚVOD – 1

$$\begin{tikzcd}
\text{Jazyk} \arrow[dr, bend right, "\text{Automat přijímá jazyk}"'] & & \text{Gramatika} \arrow[ll, bend right, "\text{gramatika generuje jazyk}"'] \arrow[dl, dotted, dash, "\text{Automat reprezentuje gramatiku}"] \\
& \text{Automat} \arrow[dl, dash] \arrow[dr, dash] & \\
\text{vyhovuje?} & & \text{nevyhovuje?}
\end{tikzcd}$$

Jazyk

- množina vět (některých řetězců) nad danou abecedou
- pro jeho popis používáme gramatiku nebo automat

Gramatika

- matematická struktura pro generování správných vět jazyka

Automat

- algoritmický stroj rozhodující, zda daný text odpovídá gramatice daného jazyka
- kromě rozhodování může vykonávat i jinou činnost (překlad)

Motivace

- Základním nástrojem pro reprezentaci a zpracování informací jsou jazyky (lidské, matematické i programovací)
- Každý algoritmický problém lze popsat nějakým jazykem
- Překladače = jazyky, gramatiky a automaty jsou základní stavební kameny pro tvorbu překladačů

PROGRAMOVÁNÍ – FORMÁLNÍ – JAZYKY – 02 – SYNTAKTICKÉ – DIAGRAMY

$A = \{a\}$

- A je množina obsahující $\overset{\text{entitu}}{\text{prvek}}$ a
- $A \not\equiv a$

$A \rightarrow B$

- zobrazení z množiny A do množiny B

$\{\{\}, \{a\}\}$

- množina množin, množina může být prvkem jiné množiny
- třída

$A = B \setminus C$

- rozdíl množin B a C

$A \subset B$

- A je podmnožinou B, ale není ekvivalentní B

$A \times B \times C$

- Kartézský součin
- množina uspořádaných n-tic (zde trojic) prvků z daných množin
- $[a, b, c]$ – konkrétní trojice, kde $a \in A$, $b \in B$, $c \in C$

Terminologie

- **Abeceda** = je konečná množina základních symbolů
  - terminální symboly ⇒ konečné, nereprezentují nic dalšího
  - značí se symbolem Σ (velké sigma)
  - př. $\Sigma_1 = \{x \mid x \text{ je číslice}\}$
- **Věta** = řetězec symbolů, který patří do jazyka
  - NEPOUŽÍVAT „SLOVO"
  - prázdná věta, či symbol ε
    - používá se v mnoha gramatikách (i „nic" má svoji hodnotu)
  - věta PATŘÍ DO JAZYKA (řetězec je jen sled znaků, symbolů abecedy)

Formální jazyk

- je množina vět nad konečnou abecedou Σ
- značen symbolem L

Σ^*

- jazyk všech možných vět nad abecedou Σ (včetně prázdné věty)
- nekonečný jazyk (i pro jednoprvkovou abecedu)
- jakýkoliv řetězec jazyka $\Sigma^* \rightarrow$ věta

$L \subseteq \Sigma^*$

- každý jazyk je podmnožinou množiny Σ^*

Značení jazyků

- A^* = jazyk všech možných slov nad abecedou A včetně prázdného slova
- A^+ = jazyk všech možných slov nad abecedou A kromě prázdného slova
- A^n = jazyk všech slov o délce n nad abecedou A (př. A^2 všechna 2znaková slova nad abecedou A)

semafor

- konečné jazyky = lze vyjmenovat
- nekonečné jazyky = nejde to vše vyjmenovat

↳ generativní popis množiny
- popsány formálním (gramatika, automat či jiným způsobem)

- deklarativní ⇒ vyjmenování slov
- generativní ⇒ gramatika

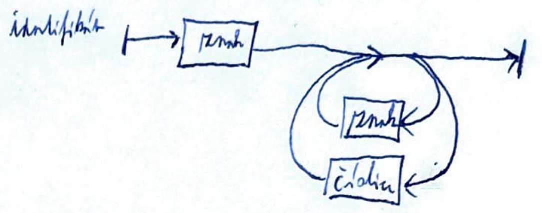
Syntaxe X sémantika

- Syntaxe = soubor pravidel pro zápis jazyka
  - vyjádřena gramatikou, syn. diag., EBNF, automat
  - v programu ji kontroluje překladač
- Sémantika = význam vět a slov (logika)
  - program dává význam čtenáři – programátorovi

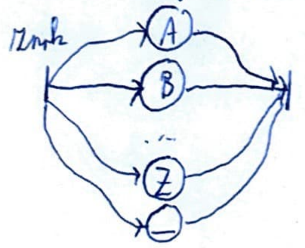
Syntaktický diagram

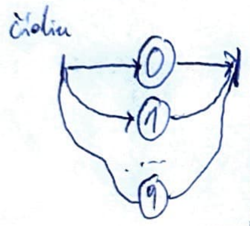
- grafický popis syntaxe
- symboly:
  - Nonterminály = odkazy pro další syntaktický diagramy $\boxed{\text{znak}}$
  - Terminály = symboly nebo klíčová slova, které se už dále nerozgenerovávají Ⓐ

Příklad: Diagram popisující identifikátor jazyka C

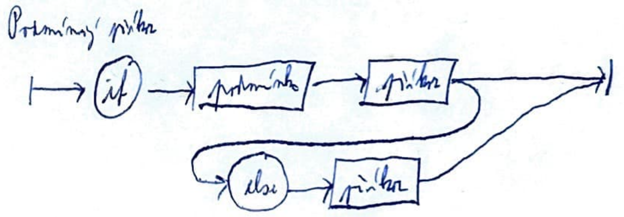


Příklad: Diagramy popisující znaky a číslice

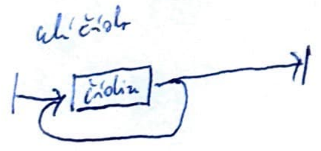




Příklad: Diagram podmíněný příkaz jazyka C

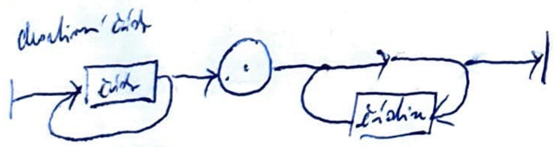


Příklad: celé číslo



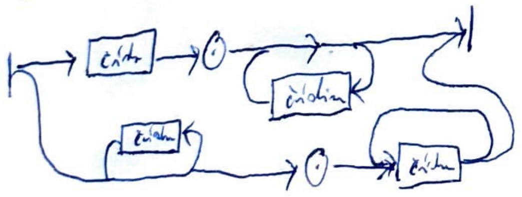
1
123

Příklad: desetinné číslo verze 1



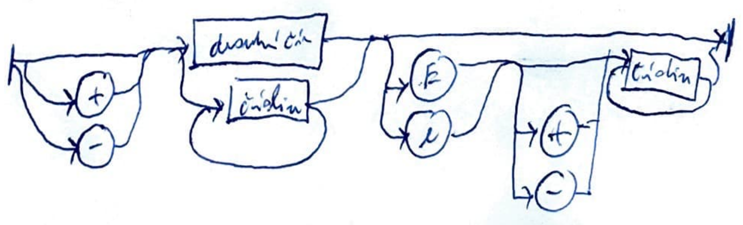
123.
123.45

Příklad: desetinné číslo verze 2



Příklad: desetinné číslo, verze 3

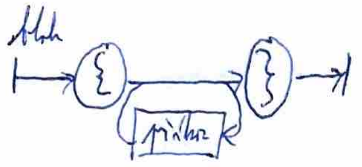
Reálné číslo



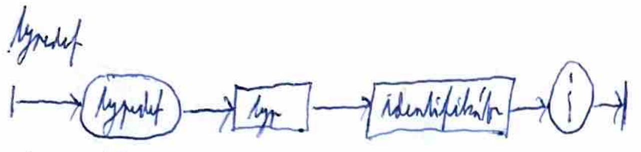
EBNF – textový zápis syntaktického diagramu

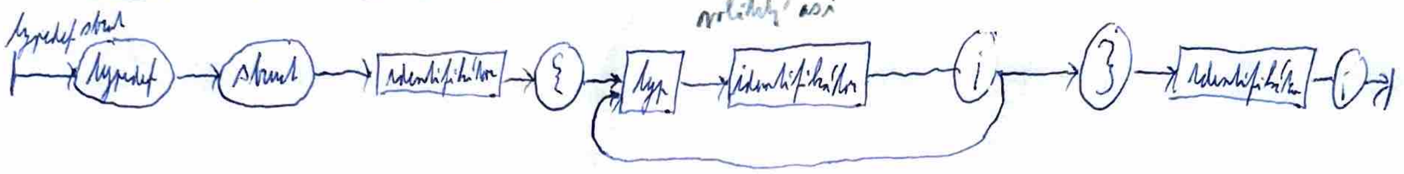
Testy – syntaktické – diagramy

B Příklad: popiš zápis blokového příkazu jazyka C ({}). Použij neterminál "příkaz".

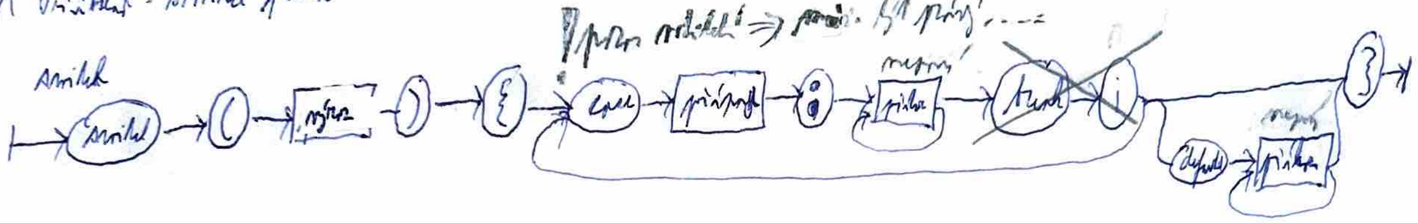


G Příklad: popiš typedef a struct

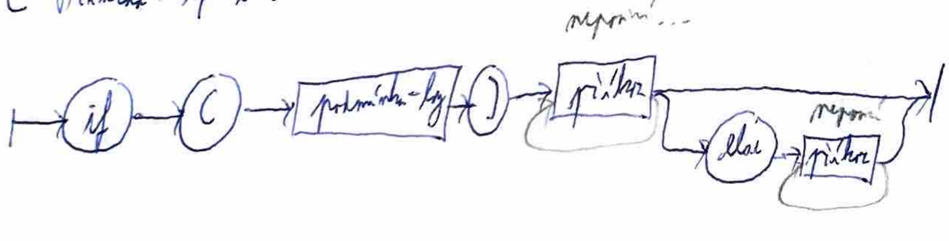




A Příklad: switch příkaz



C Příklad: if – else



REGEX

- textové vzory pro popis syntaxe
  - pro vyhledávání a manipulaci s textem
  - Regular Expression Pattern → RegEx
- hledání podle vzoru (jméno@server.koncovka)

- součástí programovacích jazyků (Perl, JavaScript, PHP, sed)
  C, C++: regex.h
  Java: balík java.util.regex
- různé odchylky v jiných jazycích

př. celé číslo ⇒ [0-9]+ opakuj to 1x a vícekrát

Používané symboly

- literály (/hello/ ⇒ hello), výrazy (znaky abecedy)
- Metaznaky – zvláštní symboly, konec řádku
- kvantifikátory
- nepovinné [illegible]

Kvantifikátory

- X? – X se vyskytuje nula nebo jedenkrát
  - nepovinný výskyt
- X* – X se vyskytuje nula nebo vícekrát
  - libovolné opakování
- X+ – X se vyskytuje jednou nebo vícekrát
  - alespoň jedno opakování
- X{n} – X se vyskytuje právě n krát
- X{n,} – X se vyskytuje n krát nebo vícekrát
- X{n,m} – X se vyskytuje n až m krát
  - alespoň n opakování, ale nikdy ne více než m krát

Vyhledávání „metaznaků"

- \\ – hledej znak \
- \. – hledej znak .
- platí podobně i pro ostatní...

Zápis regulárních výrazů

- Znak jde vyjádřit množinou
  - [abc] – najdi jeden z vyjmenovaných znaků
  - [^abc] – najdi libovolný znak kromě těch vyjmenovaných
  - [0-9a-zA-Z] – najdi znak z daného rozsahu
- např. vzor [Pp]etr hledá v textu slovo Petr nebo petr
- Ve vzoru jde používat metaznaky
  - . – reprezentuje jeden libovolný znak
  - ^ – reprezentuje začátek řádku (nebo textu)
    - ^Ahoj – najdi slovo Ahoj, jen pokud leží na začátku řádku
  - $ – Reprezentuje konec řádku (nebo textu)
    - ^Ahoj$ – najdi slovo Ahoj, jen pokud leží na řádku samo
  - \b – reprezentuje okraj slova
    - \bko – ko kosa kokos okolin
    - ko\b – ko kosa kokos okolin
    - \bko\b – ko kosa kokos okolin
- Další metaznaky reprezentují zkratky
  - \d – reprezentuje cifru (totéž co [0-9])
  - \D – reprezentuje negaci cifry [^0-9]
  - \s – reprezentuje bílý znak [\t\n\x0B\f\r]
  - \S – reprezentuje nebílý znak [^\s]
- Logické nebo |
  - Petr|Pavel – hledej Petr nebo Pavel
- Kulaté závorky seskupují části vzoru
  - P(etr|avel) – hledej Petr nebo Pavel
  - zapsání závorek jako znaku – \(a \)

REGEX

př. identifikátor

```
[_a-zA-Z](_a-zA-Z|[0-9])*
```

př. desetinné číslo

```
                  |\.     )?\.        opravit chyby
[0-9]+(\.[0-9]*)?|([0-9]*\.)?[0-9]+   volitelný
```

```
\d+(\.\d*)?|(\d*\.)?\d+
```

```
-?(?:\d+\.\d+|\.|\.d+)(?:[eE][-+]?\d+)?
```

↳ tvoří nezachytávací skupiny

A př. binární číslo (kdyby jen nula ⇒ [illegible])

```
^0*|^1[0-1]*
```

př. email adresa (BONUS příklad -03)

```
[a-zA-Z]+\.[a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]+)
([a-zA-Z\.]+[0-9a-zA-Z\.]*   -||-
```

s čárkami

př. najdi první slova v souboru

```
^[^\s]+|[\.\?!][^\s]+
                ^
              [\s]+
```

př. přímá řeč

```
"[^"]*"
```

CBD př. deklarace proměnných typu int, short, long ; proměnných

```
(\t(short|int|long)\s+\w+\s*(=\s*[+-]?[0-9]+([+-/*][+-]?[0-9]+)*\s*)?;)+
```

F př. zápis void funkcí

```
void\s+\w+\s*\([^()]*\)\s*\{[^{}]*\}
```

G př. zápis hex čísla

```
[0-9A-Fa-f]
```

F. př. hlavičky obsahující argumenty v prototypu

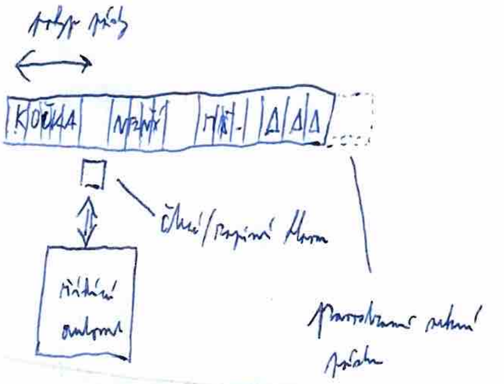
```
\s*void\s*[a-zA-Z]+[a-zA-Z0-9]?\s*\((\((int|char|short|long)\s*[a-zA-Z]+[a-zA-Z0-9]?(,)?)|
\s*void ) \s*;?)+
```

AUTOMATY

- Automat je algoritmický matematický stroj
  - rozhoduje, zda vstupní text je větou daného jazyka
  - realizující funkce z překladu, modifikace vstupního textu, interpretace vstupního textu

Turingův stroj

- automat navržen Alanem Turingem
- teoretický model nejobecnějšího počítače (umí vše co umí počítač)
  - existují problémy, které TS, ani jiný počítač vyřešit neumí (viz Halting Problem)
- používá se pro dokazování řešitelnosti problémů teoretické informatiky



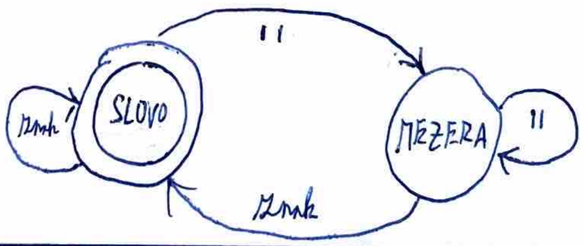
Konečný automat

- jednoduchý automat s konečným počtem stavů
- využívá se pro
  - zpracování regulárních výrazů (regex)
  - řešení problémů, které lze formulovat tak, že program prochází pevně rozlišitelnými stavy
    - zpracování událostí
    - zpracování logicky organizovaných souborů
    - hry, simulace
- grafická reprezentace
  - stavy – uzly – kolečka
  - koncové stavy – dvojité kolečko
  - přechody – šipky ohodnocené symboly z abecedy Σ
- Fungování automatu
  - čte vstupní větu po symbolech a přitom mění svůj stav
  - automat přijímá větu, pokud zpracování skončí v jednom z koncových stavů
- Algoritmus konečného automatu

```
- stav = výchozí stav
- dokud není konec vstupu opakuj
  - symbol = přečti symbol ze vstupu
  - proveď podle aktuálního stavu (switch)
    • stav 1:
        - zpracuj symbol
        - stav = přejdi na další stav
    • stav 2: ....
```

- je uspořádaná pětice $A = (S, \Sigma, \sigma, s, F)$
  - S – konečná množina stavů automatu
  - Σ – (velká sigma) konečná abeceda, resp. množina vstupních symbolů
  - σ – (malá sigma) přechodová funkce – popisuje, do jakého stavu se automat dostane při vstupu dalšího vstupního symbolu
  - $s \in S$ – počáteční stav automatu
  - $F \subseteq S$ – množina koncových stavů

př. Automat přijímající slova oddělená mezerami. Za posledním slovem nesmí být žádná mezera.



- Speciální druh automatu = Překladač (překládání věty jednoho jazyka do druhého a zachování významu)
- Kompilátor = Překladač jazyka do jazyka stroje – instrukcí, které dokáže přímo vykonávat počítačový procesor ⇒ vytváří spustitelný soubor
  - → překladač do bytekódu ⇒ cílem je nezávislost na HW (stroji)
- Interpret = zpracování programu v daném jazyce a jeho příkazy rovnou vykonávání

ALGORITMUS KONEČNÉHO AUTOMATU

```
//Výčet možných stavů
enum Estavy {TEXT, KOMENTAR};

//inicializace výchozího stavu
int stav = TEXT;

//proměnná pro uchování vstupních znaků
int znak;
```

```
while ( ( znak=getchar() ) ≠ EOF){// čti do konce
  switch (stav) {                 // pracuj podle stavu
    case TEXT:                    // ve stavu TEXT
      if (znak≠'{') { putchar(znak); }
      else { stav = KOMENTAR; }   // přepnutí stavu
      break;
    case KOMENTAR:                // ve stavu KOMENTAR
      if (znak=='}') { putchar(' '); stav = TEXT; }
      break;
} }
if (stav≠TEXT) Chyba("Neuzavřený komentář!");
```