Syntaktická analýza a Prolog

1

Metody pro syntaktickou analýzu jazyka

Notace: **Abeceda** - konečná množina znaků (symbolů).

Slovo z abecedy **A** je libovolný konečný řetězec symbolů z **A**.

A* = množina všech slov vytvořených ze symbolů abecedy **A**.

Gramatika G je definována pomocí čtveřice **<N, T, P, S>,** kde **S** (*start*) je jeden z prvků **N** a označuje počáteční symbol a ostatní symboly označují množiny, které obsahují

- **N** neterminální symboly,
- **T** terminální symboly,
- **P** přepisovací pravidla

a pro které platí:

- **N** a **T** jsou disjunktní.
- Přepisovací pravidlo má vždy tvar **L --> R**,

  kde **L**, **R** jsou slova v abecedě **N ∪ T**, pro která platí, že **L** obsahuje alespoň 1 neterminál (tato podmínka je označována ***ntl***). Přesněji **P** odpovídá množině dvojic slov takové, že
- **P** je podmnožinou **(N ∪ T)* N (N ∪ T)* x (N ∪ T)***

Jazyky a gramatiky

Jazyk L_G generovaný gramatikou G tvoří všechna slova z abecedy terminálů, t.j. z **T***, která lze získat z počátečního symbolu **S** postupnou aplikací přepisovacích pravidel **P**. Označíme-li zřetězení konečného počtu aplikací pravidel jako -->*, pak L_G = { **ω T* : S -->* ω** }

Chomského klasifikace gramatik podle tvaru přepisovacích pravidel

Značení: Nechť **α, β** jsou libovolná slova z (N ∪ T)*

γ je neprázdné slovo z (N ∪ T)*

G. typu 0: Nejobecnější typ, jehož přepisovací pravidla musí splňovat pouze podmínku ***ntl***. Jazyky generované gramatikami typu 0 jsou právě jazyky rozpoznávané Turingovými stroji.

G. typu 1: Kontextové gramatiky. Libovolné přepisovací pravidlo musí být tvaru **α X β --> α γ β**, kde **X** je nějaký symbol z **N**.

G. typu 2: Bezkontextové gramatiky. Libovolné přepisovací pravidlo musí být tvaru **X --> α**, kde **X** je nějaký symbol z **N**.

G. typu 3: Regulární gramatiky. Libovolné přepisovací pravidlo musí být tvaru **X --> aY** nebo **X --> a**, kde **X, Y** jsou nějaké symboly z **N** a **a** je nějaký symbol z **T**.

Notace: **Nechť** a, b, c **předst. terminály,** X, Y **a** S **neterminály.**

Příklad 1:

Bezkontextová gramatika **G1** , která má pouze 2 pravidla: **S --> a S b, S --> ab**

L_{G1} = {**$a^n b^n$: n** je libovolné přirozené číslo }

Příklad 2:

Bezkontextová gramatika **G2** , která má pouze následující typy pravidel:
S --> a S a, S --> b S b, S --> aa, S --> bb, S --> a, S --> b,

L_{G2} = { α $α^R$: α je lib. slovo ze symbolů **a**,**b**, $α^R$ je slovo α čtené pozpátku }

Tedy L_{G2} obsahuje právě **palindromy** = slova, která se čtou zleva doprava i zprava doleva stejně.

Příklad 3:

Najděte nejjednodušší gramatiku, která generuje právě všechna slova tvaru $a^n b^n c^n$, kde ***n*** je libovolné přirozené číslo.

DCG a prostředky Prologu

```
a --> [].
a --> [z],a.
```

Zkrácený zápis:

```
a --> []|[z],a.
```

`-->` je **zabudovaný** binární operátor,

`[Výraz]` je chápán jako **terminální symbol, ostatní** (např.a) jsou **neterminální symboly**.

Slova generovaná touto gramatikou získáme pomocí zabudovaného predikátu **phrase/3**:

```
phrase(neterminál, řetězec,
  zbytek_který_se_nepodařilo_analyzovat_jako_neterminál)
?- phrase(a,Y, []).
Y=[z]; Y=[z,z]; Y=[z,z,z];…
```


Prolog a příklady 1, 2

Příklad 1: **G1** tvoří 2 pravidla: **S --> a S b, S --> ab**

```
s --> [a],s,[b].
s --> [a],[b].
```

```
s --> [a],[b].
s --> [a],s,[b].
```

```
?- phrase(s,W,[]).   [a,b];[a,a,b,b]; …
```

Příklad 2 **(G2):** R --> **aa, R --> bb, R --> a, R --> b,** R --> **a R a, R --> b R b**

```
?- phrase(r,W,[]).   [a,a];[b,b];[a];[b]; [a|T]..

?- phrase(r,[b,a,a,b],[]).      more
```

NLP a prostředky Prologu

Bezkontextové gramatiky čistě formálně připomínají pravidla Prologu. Pravidlo

```
sentence  --> noun_phrase, verb_phrase.
```

by bylo možné zapsat v klasickém Prologovském zapisu jako

```
sentence(S):-
    noun_phrase(N),verb_phrase(V),append(N,V,S).
```

Definite Clause Grammars (DCG)

Prolog obsahuje zabudovaný operátor "-->", k jehož použití je doplněno několik užitečných predikátů, např.

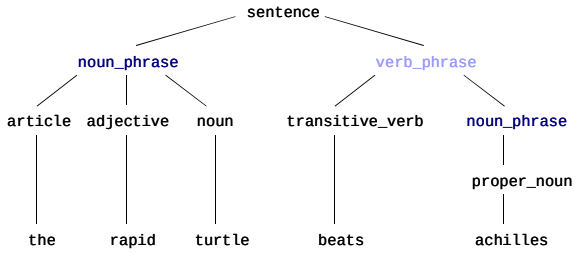
phrase(Neterminál_gramatiky, Seznam_slov, Zbytek).

Úspěch, pokud Seznam_slov *je správně utvořený syntaktický útvar pro odpovídající* Neterminál_gramatiky a Zbytek =[]

…

NLP a prostředky Prologu

```
sentence     --> noun_phrase, verb_phrase.
noun_phrase  --> proper_noun.
noun_phrase  --> article,adjective,noun.
noun_phrase  --> article,noun.
verb_phrase  --> itransitive_verb.
verb_phrase  --> transitive_verb,noun_phrase.
article      --> [the].
adjective    --> [lazy];[rapid].
proper_noun  --> [achilles].
noun         --> [turtle];[president];[queen];[cat];[fish].
itransitive_verb    --> [sleeps].
transitive_verb     --> [beats].
?- phrase(sentence,Y, []). Y = [achilles,sleeps]
?- phrase(sentence,[achilles, beats, the, lazy, turtle], []).
```

Jak řešit shodu přísudku s podmětem?

A)

sentence --> noun_phrase_singular, verb_phrase_singular.
sentence --> noun_phrase_**plural**, verb_phrase_plural.

B)

sentence --> noun_phrase(cislo), verb_phrase(cislo).

DCG gramatiky dovolují řešení B) s novým argumentem: je mnohem elegantnější a umožňuje další modifikaci!!!


```
sentence          --> noun_phrase(N),verb_phrase(N).
noun_phrase(N)    --> article(N),noun(N).
verb_phrase(N)    --> intransitive_verb(N).
article(singular) --> [a].
article(singular) --> [the].
article(plural)   --> [the].
noun(singular)    --> [turtle].
noun(plural)      --> [turtles].
intransitive_verb(singular)  --> [sleeps].
intransitive_verb(plural)    --> [sleep].
```

phrase(sentence,X,[]).
X = [a,turtle,sleeps], [the,turtles,sleep], …

[frogs,and,the,snake,sleep] ?

```
sentence             --> noun_phrase(N),verb_phrase(N).
noun_phrase(N)       --> article(N),noun(N).
noun_phrase(plural)  --> noun(N),[and],noun(N).
verb_phrase(N) --> intransitive_verb(N).
article(singular)    --> [a].
article(singular)    --> [the].
article(plural)      --> [the].
noun(singular)  --> [turtle].
noun(singular)  --> [frog].
noun(plural)    --> [turtles].
noun(plural)    --> [snakes].
intransitive_verb(singular)  --> [sleeps].
intransitive_verb(plural)    --> [sleep].
```

Oprava!

```
sentence                --> noun_phrase(N),verb_phrase(N).
    noun_phrase(N)      --> article(N),noun(N).
    noun_phrase(plural) --> noun(N),[and],noun(M).
verb_phrase(N) --> intransitive_verb(N).
    article(singular)   --> [a].
    article(singular)   --> [the].
    article(plural)     --> [the].
noun(singular)  --> [turtle].
noun(singular)  --> [frog].
noun(plural)    --> [turtles].
noun(plural)    --> [snakes].
    intransitive_verb(singular)   --> [sleeps].
    intransitive_verb(plural)     --> [sleep].
```


Příklad: Achilles - pokračování

```
?- phrase(sentence(X), [the, rapid, cat, sleeps], []).
X = s(np(art(the), adj(rapid), n(cat)), vp(iv(sleeps)))

?- phrase(sentence(X), [the, rapid, cat, beats, achilles,
   with, bag], []).
No (0.00s cpu)

?- phrase(sentence(X), [the, rapid, cat, beats, achilles,
   with, bag], Z).
X = s(np(art(the), adj(rapid), n(cat)), vp(tv(beats),
   np(pn(achilles))))
Z = [with, bag]
```

Poslední argument vrací informaci o zbytku vstupního řetězce, který se nepodařilo analyzovat.

Příklad: Achilles

```
sentence(s(NP,VP)) --> noun_phrase(NP),verb_phrase(VP).
noun_phrase(np(N)) --> proper_noun(N).
noun_phrase(np(Art,Adj,N))--> article(Art),adjective(Adj),noun(N).
noun_phrase(np(Art,N)) --> article(Art),noun(N).
verb_phrase(vp(IV))    --> itransitive_verb(IV).
verb_phrase(vp(TV,NP))--> transitive_verb(TV),noun_phrase(NP).
article(art(the)) --> [the].
adjective(adj(lazy)) --> [lazy].
adjective(adj(rapid)) --> [rapid].
proper_noun(pn(achilles)) --> [achilles].
noun(n(turtle)) --> [turtle].
noun(n(president)) --> [president].   noun(n(cat)) --> [cat].
itransitive_verb(iv(sleeps)) --> [sleeps].
transitive_verb(tv(beats)) --> [beats].
?- phrase(sentence(X),S,[]).
X = s(np(pn(achilles)),vp(tv(beats), np(art(the), adj(lazy), n(turtle))))
S = [achilles, beats, the, lazy, turtle]
```


Použití argumentů DCG

?- phrase(sentence (X), Veta).

X = s(np (pn (achilles)), vp(iv (sleeps)))

Veta = [achilles, sleeps], …

Argumenty pomáhají např. při identifikaci vybrané části věty.

?- phrase(sentence (T), [achilles,beats,the, lazy,turtle]).

```
T = s(np(pn(achilles)),
        vp(tv(beats),
           np(art(the),
              adj(lazy),
              n(turtle))))
```

Lze dále doplnit o ilustrativní tisk:

```
-----s------np------pn---achilles
      -----vp------tv------beats
            -----np-------art------the
                  -------adj-----lazy
                  ---------n----turte
```

Argument může mít i číselnou hodnotu – viz příklad „čísla slovy"

Příklad: Achilles - pokračování

```
?- phrase(sentence(X), [the, rapid, cat, sleeps], []).
X = s(np(art(the), adj(rapid), n(cat)), vp(iv(sleeps)))

?- phrase(sentence(X), [the, rapid, cat, beats, achilles,
   with, bag], []).
No (0.00s cpu)

?- phrase(sentence(X), [the, rapid, cat, beats, achilles,
   with, bag], Z).
X = s(np(art(the), adj(rapid), n(cat)), vp(tv(beats),
   np(pn(achilles))))
Z = [with, bag]
```

Poslední argument vrací informaci o zbytku vstupního řetězce, který se nepodařilo analyzovat.

Problémy strojového překladu

Spirit is high but flesh is weak

A->R

R->A

Vodka is good but meat is rotten

Příklad: čísla slovy

```
numeral(N) --> n_999(N).
numeral(N) --> n1_9(N1),[thousand], n_999(N2),{N is N1*1000 + N2}.
n_999(N) --> n_99(N).
n_999(N) --> n1_9(N1), [hundred], n_99(N2), {N is N1*100 + N2}.
n_99(N)   --> n0_9(N).
n_99(N)   --> n10_19(N).
n_99(N)   --> n20_90(N).
n_99(N)   --> n20_90(N1), n1_9(N2), {N is N1 + N2}.
n0_9(0) --> [].
n0_9(N) --> n1_9(N).
n1_9(1) --> [one].  n1_9(2) --> [two].
n10_19(10) --> [ten]. n10_19(11) --> [eleven].
n20_90(20) --> [twenty].   n20_90(30) --> [thirty].
?- phrase(numeral(X), Y, []). X = 1 Y = [one];X = 2 Y = [two]
```


Příklad: čísla slovy - pokračování

```
?- phrase(numeral(X), [one, thousand, two, hundred, eleven], []).
X = 1211                More (0.00s cpu)      No
?- phrase(numeral(X), [one, thousand, two], []).
X = 1002
?- phrase(numeral(X), [thousand, two, hundred, one], []).
No
?- phrase(numeral(X), [one, thousand,eleven],V).
N=0,V=[one, thousand,eleven]; N=1,V=[thousand,eleven];
   N=1000,V=[eleven]; N=1011,V=[]; No


?- phrase(numeral(2011), C, []).
C = [two, thousand, eleven]; No
?- phrase(numeral(1000), C, []).
C = [one, thousand]; No
X = 2000
```