

Syntaktická analýza

Definite-Clause Grammars (DCG) Gramatiky uspořádaných klauzulí

Významná aplikace Prologu: syntaktická analýza

- `sentence --> noun_phrase, verb_phrase.`
`noun_phrase --> determiner, noun.`
`noun_phrase --> noun.`
`verb_phrase --> verb, noun_phrase.`
`verb_phrase --> verb.`
`determiner --> [the].`
`determiner --> [a].`
`noun --> [student].`
`noun --> [dcg].`
`verb --> [likes].`
- `| ?- sentence([a, student, likes, dcg]).`
`yes`

Hana Rudová, Logické programování I, 20. dubna 2012

2

Gramatiky uspořádaných klauzulí

DCG a CFG

- DCG (DC gramatiky) jsou rozšířením bezkontextových gramatik (CFG)
- Implementace DCG využívá rozdílových seznamů

Formální podobnosti mezi DCG a CFG:

- CFG: pravidla tvaru $x \rightarrow y$, kde
 - $x \in N$ je neterminál
 - $y \in (N \cup T)^*$ je konečná posloupnost terminálů a neterminálů
- DCG: pravidla tvaru $\langle \text{hlava} \rangle \text{ --> } \langle \text{tělo} \rangle$
 - $\langle \text{hlava} \rangle$ je opět neterminál
 - $\langle \text{tělo} \rangle$ je opět konečná posloupnost terminálů a neterminálů
- Pravidlo $\langle \text{hlava} \rangle \text{ --> } \langle \text{tělo} \rangle$ znamená, že
 - jedním z možných tvarů $\langle \text{hlavy} \rangle$ je $\langle \text{tělo} \rangle$, neboli
 - $\langle \text{hlavu} \rangle$ je možno přepsat na $\langle \text{tělo} \rangle$

Rozdíly a rozšíření DCG oproti CFG

- **Neterminál** může být téměř libovolný term, ovšem kromě seznamu, proměnné a čísla.
 - neterminál může být složený term, tj. neterminálům lze přidat **argumenty**.
- **Terminál** může být libovolný term, s tím, že terminály a posloupnosti terminálů uzavíráme do hranatých závorek – jako **seznamy**.
 - hranaté závorky tedy odlišují terminály od neterminálů
- Pravá strana pravidla může obsahovat **dodatečné podmínky** v podobě prologovských podcílů. Tyto podmínky uzavíráme do složených závorek.
 - podmínky slouží jen pro testování, negenerují žádnou větnou formu
- Levá strana pravidla může dokonce vypadat i tak, že neterminál je následován posloupností terminálů.
- Tělo pravidla smí obsahovat řez.
 - nepodporováno všemi Prology

Příklad: gramatika

- sentence --> noun_phrase, verb_phrase.
noun_phrase --> determiner, noun_phrase2.
noun_phrase --> noun_phrase2.
noun_phrase2 --> noun.
noun_phrase2 --> adjective, noun_phrase2.
verb_phrase --> verb.
verb_phrase --> verb, noun_phrase.
determiner --> [the]. noun --> [boy].
determiner --> [a]. noun --> [song].
verb --> [sings]. adjective --> [young].
- | ?- sentence(S, []).
S = [the,song,sings] ? ;
S = [the,song,sings,the,song] ?
| ?- sentence([the, young, boy, sings, a, song], []).
yes

Příklad: binární čísla

- DC gramatika number rozeznávající binární čísla:
number --> [0].
number --> [1].
number --> [0], number.
number --> [1], number.
| ?- number([0,1,0,1,1], []). yes
- Napište DCG number2 pro rozpoznání binárních čísel bez vedoucích nul.
- Napište DCG number3 rozpoznávající binární čísla, které jsou mocninou dvojky.
number2 --> [1].
number2 --> [1], number.
number3 --> [1], zeros.
zeros --> [].
zeros --> [0], zeros.

Příklad: neterminály s argumentem

- DC gramatika digits generuje binární čísla zapsaná jedinou číslicí:
digits --> same(0). | ?- digits([1,1,0,1], []).
digits --> same(1). no
same(N) --> [N]. | ?- digits([1,1,1], []).
same(N) --> [N], same(N). yes
- Upravte kód tak, aby byly akceptovány jen korektní věty:
s --> np, vp.
np --> [zeny].
np --> [muzi].
vp --> [pracovali].
vp --> [pracovaly].
?- s([zeny, pracovali], []).
yes

Nápověda: přidejte proměnnou pro rod (pro np a vp)

Řešení: neterminály s argumentem

Původně:

```
s --> np, vp.  
np --> [zeny].  
np --> [muzi].  
vp --> [pracovali].  
vp --> [pracovaly].
```

Řešení:

```
s --> np(Rod), vp(Rod).  
np(z) --> [zeny].  
np(mz) --> [muzi].  
vp(mz) --> [pracovali].  
vp(z) --> [pracovaly].
```

Generativní/rozpoznávací síla DCG: větší než CFG

- DCG dokáže generovat/rozpoznávat jazyky typu 0
- Cvičení: napište DCG gramatiku generující jazyk $a^n b^n c^n$
- ?- abc(X, []).
 - X = [] ;
 - X = [a, b, c] ;
 - X = [a, a, b, b, c, c] ;
 - X = [a, a, a, b, b, b, c, c, c] ;

```
Nápověda: využijte a(s(s(s(0)))) , b(s(s(s(0)))) , c(s(s(s(0))))
abc --> a(N), b(N), c(N).
a(0) --> [].
a(s(N)) --> [a], a(N).
b(0) --> [].
b(s(N)) --> [b], b(N).
c(0) --> [].
c(s(N)) --> [c], c(N).
```

Komplexní vyhodnocování výrazů

```
E -> T + E | T - E | T
T -> F * T | F / T | F
F -> (E) | f
```

```
expr(X) --> term(Y), [+], expr(Z), {X is Y+Z}.
expr(X) --> term(Y), [-], expr(Z), {X is Y-Z}.
expr(X) --> term(X).
term(X) --> factor(Y), [*], term(Z), {X is Y*Z}.
term(X) --> factor(Y), [/], term(Z), {X is Y/Z}.
term(X) --> factor(X).
factor(X) --> ['('], expr(X), [')'].
factor(X) --> [X], {integer(X)}.
```

```
% vyhodnocení výrazu 3+(4/2)-(2*6/3)
?- expr(X, [3,+, '(', 4, /, 2, ')', -, '(', 2, *, 6, /, 3, ')'], []). X = 1
```

Argument neterminálu je použit jako výstupní proměnná, která v sobě nese hodnotu příslušného aritmetického výrazu.

Pomocné podmínky v těle pravidel

Vyhodnocování výrazů

```
E -> T + E | T
T -> num

expr(X) --> term(A), [+], expr(B), {X is A+B}.
expr(X) --> term(X).
term(X) --> [X], {number(X)}.

?- expr(X, [1,+,2,+,2], []). X = 5
```

- Cvičení: přidejte operaci násobení

```
E -> N + E | N
N -> T * N | T
T -> num

expr(X) --> expr2(A), [+], expr(B), {X is A+B}.
expr(X) --> expr2(X).
expr2(X) --> term(A), [*], expr2(B), {X is A*B}.
expr2(X) --> term(X).
term(X) --> [X], {number(X)}.
```

Přepis do Prologu

Přepis do prologovského programu pomocí append/3:

- Větu reprezentujeme seznamem slov [the, young, boy, sings, a, song]
- **Pravidlová část** – neterminál chápeme jako unární predikát, jehož argumentem je ta větná složka, kterou daný neterminál popisuje


```
sentence(S) :- append(NP, VP, S),
                  noun_phrase(NP), verb_phrase(VP).
...

```
- **Slovníková část** – zapisujeme ji pomocí faktů:


```
determiner([the]).      noun([boy]).
determiner([a]).      ...

```

Predikát append/3 zde *nedeterministicky* rozděluje aktuální větnou část na dva díly, což je velký zdroj neefektivnosti.

Lepší řešení poskytují *rozdílové seznamy*.

Přepis do Prologu pomocí rozdílových seznamů

- **Rozdílové seznamy** reprezentovány dvěma argumenty, první představuje neúplný seznam a druhý jeho zbytek
`append(S-S1, S1-S0, S-S0)`
- Při volání predikátu `S-S0` je spojením: `S-S3, S3-S2, S2-S1, S1-S0`
`sentence/2` je druhý argument prázdný; neúplný seznam tím uzavíráme tj. `S0=[]`

```

sentence(S,S0):- noun_phrs(S,S1), verb_phrs(S1,S0).
noun_phrs(S,S0):- determiner(S,S1), noun_phrs2(S1,S0).
noun_phrs(S,S0):- noun_phrs2(S,S0).
noun_phrs2(S,S0):- adjective(S,S1), noun_phrs2(S1,S0).
noun_phrs2(S,S0):- noun(S,S0).
verb_phrs(S,S0):- verb(S,S0).
verb_phrs(S,S0):- verb(S,S1), noun_phrs(S1,S0).

determiner([the|S],S).      noun([boy|S],S).
determiner([a|S],S).      noun([song|S],S).
verb([sings|S],S).      adjective([young|S],S).

```

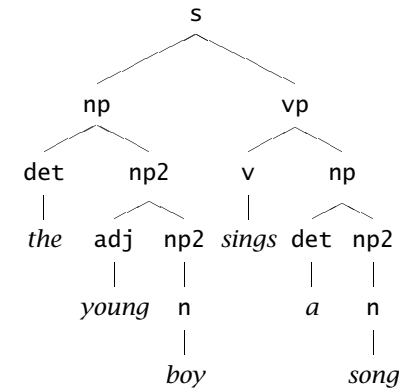
?- sentence([the,young,boy,sings,a,song],[]). yes

Derivační strom analýzy

```

?- sentence(Tree, [the,young,boy,sings,a,song],[ ]).
Tree=s( np( det(the), np2( adj(young), np2(n(boy)) ) ) ),
        vp( v(sings), np( det(a), np2( n(song)) ) ) ) )

```



Konstrukce derivačního stromu

- Neterminály opatříme argumentem:


```

sentence(s(NP,VP)) --> noun_phrase(NP), verb_phrase(VP).
noun(n(mama)) --> [mama].
noun(n(kralika)) --> [kralika].
verb(v(pekla)) --> [pekla].

```
- Doplňte gramatiku, aby platilo:


```

| ?- sentence(X, [mama,pekla,kralika],[ ]).
X = s(np(n(mama)),vp(v(pekla),np(n(kralika)))) yes

```

```

sentence(s(N,V)) --> noun_phrase(N), verb_phrase(V).
noun_phrase(np(N)) --> noun(N).
verb_phrase(vp(V)) --> verb(V).
verb_phrase(vp(V, N)) --> verb(V), noun_phrase(N).

noun(n(mama)) --> [mama].
noun(n(kralika)) --> [kralika].
verb(v(pekla)) --> [pekla].

```

Konstrukce derivačního stromu II.

Pokud však rozšíříme slovník:

```

noun(n(tata)) --> [tata].
verb(v(pekla)) --> [pekla].

```

Narazíme na problém se shodou podmětu a přísudku (mimo stávající problém „kralíka pekla máma“):

```

?- sentence(_, [tata, pekla, kralika],[ ]).
yes

```

```

?- sentence(_, [mama, pekla, kralika],[ ]).
yes

```

Proto rozšířte neterminály o další argumenty (rod, pád)

Konstrukce derivačního stromu (řešení)

Tento nový argument poskytuje pro odpovídající větnou složku informaci o jejím rodu a pádu – unifikací příslušných proměnných zajistíme shodu v této gramatické kategorii.

```
sentence(s(N,V)) --> noun_phrase(N, Rod, c1), verb_phrase(V, Rod).
noun_phrase(np(N), Rod, Pad) --> noun(N, Rod, Pad).
verb_phrase(vp(V), Rod) --> verb(V, Rod).
verb_phrase(vp(V, N), Rod) --> verb(V, Rod), noun_phrase(N, _Rod2, c4).
```

```
noun(n(mama), z, c1) --> [mama].
noun(n(tata), mz, c1) --> [tata].
noun(n(kralika), mn, c4) --> [kralika].
verb(v(pekla), z) --> [pekla].
verb(v(pekl), mz) --> [pekl].
```

Vestavěné nástroje

- operátor `-->` definován jako `?-op(1200, xfx, -->)`.
- predikáty `phrase/2`, `phrase/3`, které slouží k jednoduché *tokenizaci*

```
?- phrase(abc, [a,b,c]).           % Yes
?- phrase(abc, [a,b,c,d], [d]).   % Yes
```