1 DC gramatiky v Prologu (DCG)

Příklad 1.1: Chceme vytvořit gramatiku rozpoznávající/generující jazyk a^{2n} pro $n > 0$. Rozeberte chování následujících gramatik.

```
1.
s --> s, [a,a].
s --> [].

2.
s --> [a,a].
s --> s, [a,a].

3.
s --> [a,a].
s --> [a,a], s.
```

Jak bude vypadat prologovský program reprezentující správnou verzi této gramatiky?

Řešení 1.1:
Gramatika č. 1 vždy cyklí (pro rozpoznávání i generování).
Gramatika č. 2 dokáže jazyk generovat a správně skončí pro slova patřící do jazyka, ale cyklí pro slova, která do jazyka nepatří.
Gramatika č. 3 funguje vždy korektně.
Program pro správnou verzi:

```
s([a, a|A], A).
s([a, a|A], B) :- s(A, B).
```

□

Příklad 1.2: Napište DC gramatiku pro rozpoznávání i generování (kontextového) jazyka $a^n b^n c^n$ pro $n \geq 0$ tak, aby k vygenerovanému/rozpoznanému slovu vracela i příslušné n.

Řešení 1.2:

```
abc(N) --> a(0,N), b(N), c(N).
a(N,N) --> [].
a(N,V) --> [a], {N1 is N + 1}, a(N1,V).
b(0) --> [].
b(N) --> [b], {N > 0, N1 is N - 1}, b(N1).
c(0) --> [].
c(N) --> [c], {N > 0, N1 is N - 1}, c(N1).
```

□

Příklad 1.3: Napište DC gramatiku pro rozpoznávání korektních postfixových aritmetických výrazů obsahujících operátory +, − a nezáporná celá čísla. Pro zjednodušení je možné, aby gramatika rozpoznala i samostatné číslo.

Gramatika by například měla rozpoznat výraz 5 2 − 4 3 2 − + +. Předpokládejte, že výrazy jsou již reprezentovány jako odpovídající seznamy terminálů, tj. pro uvedený výraz `[5,2,'-',4,3,2,'-','+','+']`.

Rozšiřte vytvořenou gramatiku tak, aby rozpoznaný výraz i vyhodnotila. Dále rozšiřte gramatiku tak, aby vrátila jako jeden z argumentů i syntaktický strom provedené analýzy (parse tree).

Řešení 1.3: Snadno navrhneme levorekurzivní gramatiku (která by v Prologu nefungovala):

```
e --> f.
e --> e, e, ['+'].
e --> e, e, ['-'].
f --> [X], {integer(X)}.
```

a z ní dostaneme odstraněním levé rekurze požadovanou funkční gramatiku pro rozpoznávání:

```
e --> f, e1.
e1 --> [].
e1 --> e, ['+'], e1.
e1 --> e, ['-'], e1.
f --> [X], {integer(X)}.
```

Rozšíření gramatiky o vyhodnocování:

```
e(Y) --> f(X), e1(X,Y).
e1(X,Y) --> [], {X=Y}.
e1(X,Y) --> e(V), ['+'], {W is X+V}, e1(W,Y).
e1(X,Y) --> e(V), ['-'], {W is X-V}, e1(W,Y).
f(X) --> [X], {integer(X)}.
```

Rozšíření gramatiky o argument zachycující strom analýzy:

```
e(T,Y) --> f(Z,X), e1(Z,T,X,Y).
e1(Z,T,X,Y) --> [], {Z=T,X=Y}.
e1(Z,T,X,Y) --> e(T1,V), ['+'], {W is X+V, T2=plus(Z,T1)}, e1(T2,T,W,Y).
e1(Z,T,X,Y) --> e(T1,V), ['-'], {W is X-V, T2=minus(Z,T1)}, e1(T2,T,W,Y).
f(Z,X) --> [X], {integer(X),Z=leaf(X)}.
```

□