Formální jazyky a překladače

IFJ

Studijní opora
Řešené příklady s ilustrací

Seznam již nahlášených chyb naleznete v Errata na
https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IFJ-IT/texts/OporaIFJ-cvic-Errata.txt

Alexander Meduna
Roman Lukáš

Verze: 1.2006

Tento učební text vznikl za podpory projektu „Zvýšení konkurenceschopnosti IT odborníků – absolventů pro Evropský trh práce“, reg. č. CZ.04.1.03/3.2.15.1/0003. Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

Obsah

1. Formální jazyky a operace nad nimi

Příklad 1.
Vypište prvních 5 prvků následujících jazyků nad abecedou $\Sigma = \{a\}$ v uspořádání podle délky řetězce:

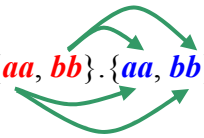
a) $L = \{ a^{2^n} : n \geq 0 \}$
b) $L = \{a^n: n \text{ je prvočíslo}\}$
(*Poznámka*: 1 není prvočíslo)

Řešení:
a) Jsou to řetězce: *a*, *aa*, *aaaa*, *aaaaaaaa*, *aaaaaaaaaaaaaaaa*
b) Jsou to řetězce: *aa*, *aaa*, *aaaaa*, *aaaaaaa*, *aaaaaaaaaaa*

Příklad 2.
Vytvořte konkatenaci následujících jazyků:
a) $\{aa, bb\}.\{aa, bb\}$
b) $\{aa, bb\}.\{\varepsilon\}$
c) $\{aa, bb\}.\varnothing$

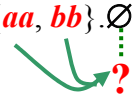
Řešení:
Konkatenace jazyků L_1 a L_2 je definována: $L_1.L_2 = \{xy: x \in L_1, y \in L_2\}$. Což neformálně znamená, že výsledný jazyk $L_1.L_2$ obsahuje všechny řetězce, které vzniknou spojením (= konkatencí) řetězců *x* a *y*, přičemž řetězec *x* je obsažen v jazyce L_1 a řetězec *y* je obsažen v jazyce L_2.

a) $\{aa, bb\}.\{aa, bb\} = \{aaaa, aabb, bbaa, bbbb\}$



b) $\{aa, bb\}.\{\varepsilon\} = \{aa\varepsilon, bb\varepsilon\} = \{aa, bb\}$

c) $\{aa, bb\}.\varnothing = \varnothing$



Konkatenace $\{aa, bb\}.\varnothing = \varnothing$, neboť podle definice musí tento jazyk obsahovat všechny řetězce tvaru *xy*, přičemž řetězec *x* je obsažen v jazyce $L_1 = \{aa, bb\}$ a řetězec *y* je obsažen v jazyce $L_2 = \varnothing$. Protože $L_2 = \varnothing$, nemůžeme najít žádný řetězec *y*, který by byl obsažen v tomto jazyce, tím spíše tedy nemůžeme ani vytvořit žádný řetězec *xy*. Konkatenace jazyků $\{aa, bb\}$ a $\varnothing$ tedy neobsahuje žádný řetězec a proto $\{aa, bb\}.\varnothing = \varnothing$.

Příklad 3.
Vytvořte průnik jazyků $L_1 = \{a^{2n}: n \geq 1\}$ a $L_2 = \{a^{3n}: n \geq 1\}$.

Řešení:
- Jazyk L_1 obsahuje řetězce samých symbolů *a*, které jsou sudé délky:

$$L_1 = \{aa, aaaa, aaaaaa, \ldots\}$$

- Jazyk L_2 obsahuje řetězce samých symbolů a, které mají délku dělitelnou číslem 3:
$$L_2 = \{aaa, aaaaaa, aaaaaaaaa, \ldots\}$$

- Průnikem těchto dvou jazyků je tedy jazyk, jehož řetězce obsahují samé symboly a, a navíc mají sudou délku dělitelnou číslem 3, což znamená, že jejich délka je dělitelná číslem 6. Formálně můžeme tento jazyk zapsat jako:
$$L_1 \cap L_2 = \{a^{6n}: n \geq 1\}$$

Příklad 4.

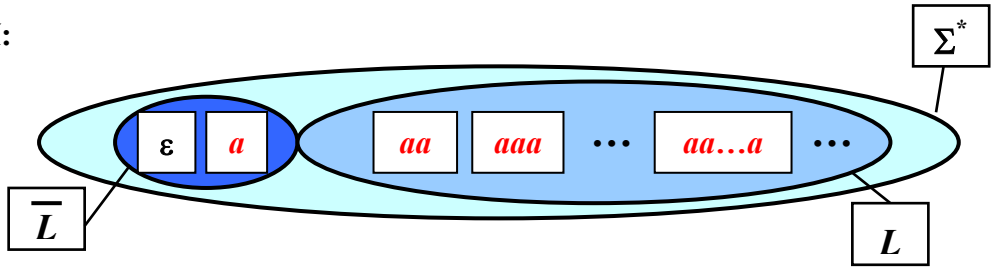
Určete, zda doplněk jazyka $L = \{a^n: n \geq 2\}$ je jazyk konečný, pokud

a) L je definován nad abecedou $\Sigma = \{a\}$

b) L je definován nad abecedou $\Sigma = \{a, b\}$

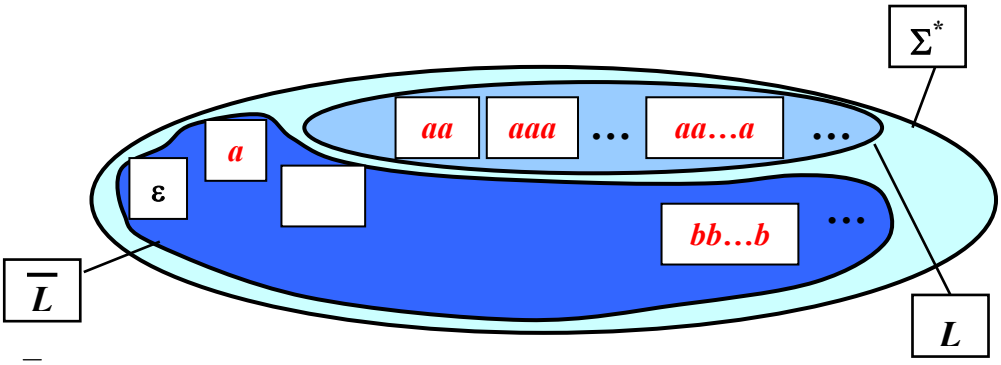
Řešení:

a)



Z obrázku je patrné, že $\overline{L} = \{\varepsilon, a\}$. $\overline{L}$ obsahuje právě 2 řetězce, $\overline{L}$ je tedy konečný jazyk.

b)



Jazyk $\overline{L}$ obsahuje všechny řetězce nad abecedou $\Sigma = \{a, b\}$, které nepatří do jazyka L. Zřejmě například libovolný řetězec, který obsahuje pouze symboly b, nepatří do jazyka L, tedy patří do jazyka $\overline{L}$. Již těchto řetězců je nekonečně mnoho, proto jazyk $\overline{L}$ je nekonečný.

Příklad 6.
Určete počet všech jazyků nad abecedou $\Sigma = \{a, b\}$, jejichž iterací vznikne konečný jazyk.

Řešení:

- Iterace jazyka L, L^*, je definována:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^i \cup \ldots = \bigcup_{i=0}^{\infty} L^i$$

a) Uvažujme jazyk $L = \varnothing$:

$L^0 = \{\varepsilon\}$
$L^1 = L.L^0 = \varnothing.\{\varepsilon\} = \varnothing$
$L^2 = L.L^1 = \varnothing.\varnothing = \varnothing$
…
$L^i = L.L^{i-1} = \varnothing.\varnothing = \varnothing$
…

$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^i \cup \ldots = \{\varepsilon\} \cup \varnothing \cup \varnothing \cup \ldots \cup \varnothing \cup \ldots = \{\varepsilon\}$

Pro $L = \varnothing$ je $L^* = \{\varepsilon\}$ a to je konečný jazyk**.**

(**Poznámka:** všimněte si, že $\varnothing^* = \{\varepsilon\}$ a nikoliv prázdný jazyk!)

b) Uvažujme jazyk $L = \{\varepsilon\}$:

$L^0 = \{\varepsilon\}$
$L^1 = L.L^0 = \{\varepsilon\}.\{\varepsilon\} = \{\varepsilon\}$
$L^2 = L.L^1 = \{\varepsilon\}.\{\varepsilon\} = \{\varepsilon\}$
…
$L^i = L.L^{i-1} = \{\varepsilon\}.\{\varepsilon\} = \{\varepsilon\}$
…

$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^i \cup \ldots = \{\varepsilon\} \cup \{\varepsilon\} \cup \{\varepsilon\} \cup \ldots \cup \{\varepsilon\} \cup \ldots = \{\varepsilon\}$

Pro $L = \{\varepsilon\}$ je $L^* = \{\varepsilon\}$ a to je konečný jazyk**.**

c) Uvažujme jakýkoliv jiný jazyk L nad abecedou $\Sigma = \{a, b\}$, který tedy obsahuje aspoň jeden neprázdný řetězec (označme jej x). Potom zřejmě L^* obsahuje řetězce $x^1, x^2, x^3, \ldots, x^i, \ldots$, kterých je nekonečně mnoho, proto L^* je **nekonečný jazyk**.

Závěr:
Nad abecedou $\Sigma = \{a, b\}$, existují pouze **2** jazyky jejichž iterací vznikne konečný jazyk a to: **∅,** $\{\varepsilon\}$.

Příklad 7.
Pomocí operací nad konečnými jazyky popište následující jazyk nad abecedou $\Sigma = \{a, b, \ldots, z\}$:

$L = \{baba, prababa, praprababa, prapraprababa, \ldots\}$

Řešení:

$$L = \{pra\}^*.\{baba\}$$

Příklad 8.
Pomocí operací nad konečnými jazyky popište jazyk L nad abecedou $\Sigma = \{0, 1, \ldots, 9, +, -\}$ obsahující řetězce reprezentující celá čísla bez přebytečných nul. Například: 100, +100, –100 jsou řetězce jazyka L, ale 001 není řetězec jazyka L, neboť obsahuje přebytečné nuly.

Řešení:

$$L = \{+, -, \varepsilon\}.\{1, 2, ..., 9\}.\{0, 1, 2, ..., 9\}^* \cup \{0\}$$

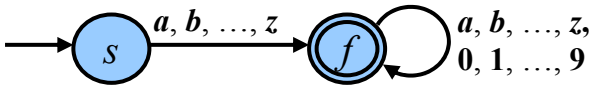
Pokud bychom uvažovali i –0 a +0 jako korektně zapsaná celá čísla, potom:

$$L = \{+, -, \varepsilon\}.\{1, 2, ..., 9\}.\{0, 1, 2, ..., 9\}^* \cup \{+, -, \varepsilon\}.\{0\}$$

2. Konečné automaty, regulární výrazy

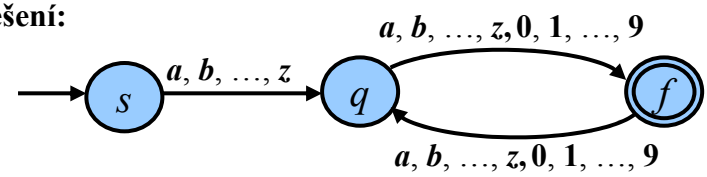
Příklad 1.
Vytvořte konečný automat, který přijímá všechny identifikátory.

Řešení:



Příklad 2.
Vytvořte konečný automat, který přijímá všechny identifikátory sudé délky.

Řešení:



Příklad 3.
Vytvořte konečný automat M přijímající všechny řetězce nad abecedou $\Sigma = \{a, b\}$, které neobsahují podřetězec ab a zároveň jejich délka je dělitelná třemi.

Množina všech koncových stavů bude obsahovat ty stavy, ve kterých řetězec neobsahoval podřetězec *ab* a dále pokud je současná délka přečteného řetězce dělitelná třemi, tedy:
$F = \{\langle —, 0\rangle, \langle a, 0\rangle\}$

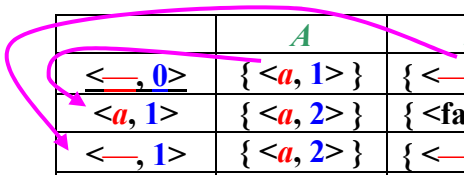
Celý automat je potom vhodné popsat tabulkou:

	a	*b*
<—, 0>	{ <a, 1> }	{ <—, 1> }
<—, 1>	{ <a, 2> }	{ <—, 2> }
<—, 2>	{ <a, 0> }	{ <—, 0> }
<a, 0>	{ <a, 1> }	{ <ab, 1> }
<a, 1>	{ <a, 2> }	{ <ab, 2> }
<a, 2>	{ <a, 0> }	{ <ab, 0> }
<ab, 0>	{<ab, 1> }	{<ab, 1> }
<ab, 1>	{<ab, 2> }	{<ab, 2> }
<ab, 2>	{<ab, 0> }	{<ab, 0> }

Způsob efektivnějšího řešení:

- Nezavedeme na začátku komplet všechny stavy. Může se někdy stát, že touto metodou zavedeme i stavy, které nikdy nebudou dostupné. Nové stavy budeme zavádět tehdy, pokud do těchto stavů vznikne v průběhu vytváření tabulky nový přechod.
- Pokud již v průběhu čtení řetězce víme, že řetězec nebude přijat, necháme přejít automat do stavu nekoncového **<false>**, ve kterým přetrvá až do konce čtení řetězce a tím řetězec nebude přijat.

Popis automatu s méně stavy:



	A	*b*
<—, 0>	{ <a, 1> }	{ <—, 1> }
<a, 1>	{ <a, 2> }	{ <false> }
<—, 1>	{ <a, 2> }	{ <—, 2> }
<a, 2>	{ <a, 0> }	{ < false> }
<—, 2>	{ <a, 0> }	{ <—, 0> }
<a, 0>	{ <a, 1> }	{ < false> }
< false>	{ < false> }	{ < false> }

Příklad 4.
Vytvořte konečný automat M přijímající všechny řetězce nad abecedou $\Sigma = \{a, b, 0\}$, které jsou sudé délky, neobsahují tři po sobě jdoucí písmena a obsahují lichý počet výskytů číslice 0.

Řešení:
Postupovat budeme obdobně jako v příkladu 3:
Všechny stavy označíme ve tvaru **<*x*, *y*, *z*>**, kde část ***x*** bude hlídat, zda dosud přijatý řetězec je sudé délky, část *y* bude hlídat, zda dosud přijatý řetězec neobsahoval tři po sobě jdoucí písmena a část *z* bude hlídat, zda dosud přijatý řetězec obsahoval lichý počet nul.

část ***x*** bude nabývat jedné z hodnot:

- ***ld*** ... délka dosud přijatého řetězce je lichá.
- ***sd*** ... délka dosud přijatého řetězce je sudá.

část ***y*** bude nabývat jedné z hodnot:

- ***0p*** … dosud přečtený řetězec nekončí písmenem.
- ***1p*** … dosud přečtený řetězec končí jedním písmenem.
- ***2p*** … dosud přečtený řetězec končí dvěma písmeny.

část ***z*** bude nabývat jedné z hodnot:

- ***s0*** … dosud přečtený řetězec obsahuje sudý počet nul.
- ***l0*** … dosud přečtený řetězec obsahuje lichý počet nul.

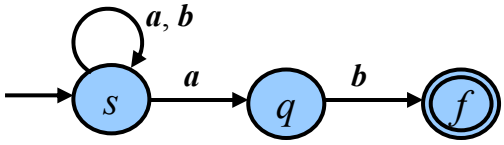
Celý automat potom můžeme popsat následovně:

	a, b	***0***
<***sd, 0p, s0***>	{ <***ld, 1p, s0***> }	{ <***ld, 0p, l0***> }
<***ld, 1p, s0***>	{ <***sd, 2p, s0***> }	{ <***sd, 0p, l0***> }
<***ld, 0p, l0***>	{ <***sd, 1p, l0***> }	{ <***sd, 0p, s0***> }
<***sd, 2p, s0***>	{ <***false***> }	{ <***ld, 0p, 10***> }
<***sd, 0p, l0***>	{ <***ld, 1p, l0***> }	{ <***ld, 0p, s0***> }
<***sd, 1p, l0***>	{ <***ld, 2p, l0***> }	{ <***ld, 0p, s0***> }
<***ld, 1p, l0***>	{ <***sd, 2p, l0***> }	{ <***sd, 0p, s0***> }
<***ld, 0p, s0***>	{ <***sd, 1p, s0***> }	{ <***sd, 0p, l0***> }
<***ld, 2p, l0***>	{ <***false***> }	{ <***sd, 0p, s0***> }
<***sd, 2p, l0***>	{ <***false***> }	{ <***ld, 0p, s0***> }
<***sd, 1p, s0***>	{ <***ld, 2p, s0***> }	{ <***ld, 0p, l0***> }
<***ld, 2p, s0***>	{ <***false***> }	{ <***sd, 0p, l0***> }
<***false***>	{ <***false***> }	{ <***false***> }

Příklad 5.

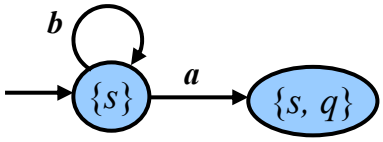
Vytvořte konečný automat M přijímající všechny řetězce nad abecedou $\Sigma = \{a, b\}$, které končí řetězcem ab. Formálně: $L(M) = \{x: ab \text{ is a suffix of } x\}$. Daný automat převeďte na deterministický.

Řešení:

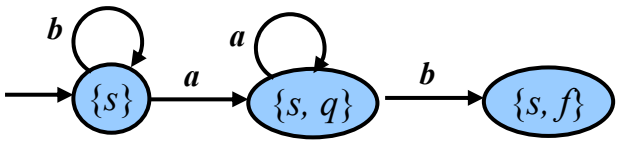


Postupná konstrukce deterministického automatu:

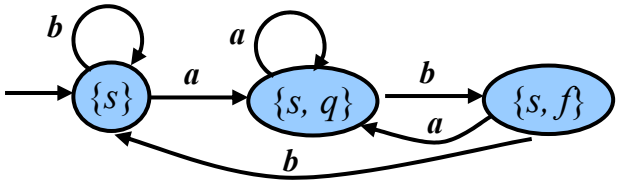
- Začneme konstruovat automat ze startovacího stavu s, který označme jako $\{s\}$. Ze stavu $\{s\}$ můžeme přejít při přečtení symbolu b do pouze do stavu $\{s\}$ ale při přečtení symbolu a do dvou stavů s a q. Vytvoříme tedy nový stav $\{s, q\}$, do kterého vstoupíme ze stavu $\{s\}$ při přečtení symbolu a:



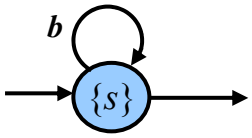
- Nyní musíme pro nový stav $\{s, q\}$ vytvořit všechny přechody a to tak, že postupně pro všechny symboly vstupní abecedy = $\{a, b\}$ budeme vytvářet nové stavy, které vniknou sloučením jednoho a více stavů z původního nedeterministickém automatu a to následovně:
  Vezmeme první symbol ze vstupní abecedy = a. Stav $\{s, q\}$ se skládá ze stavů s, q, tedy zjistíme, do kterých stavů jsme se mohli dostat v původním nedeterministickém automatu ze stavů s, q při přečtení symbolu a. Zjistíme, že ze stavu s jsme mohli přejít do stavů s, q a ze stavu q nemůžeme nikam. Celkem jsme se tedy mohli dostat opět do stavů s, q, vytvoříme tedy v automatu novou hranu označenou symbolem a do stavu $\{s, q\}$, který již existuje. Obdobně pro b zjistíme, že ze stavů s, q můžeme přejít do stavů s, f, stav $\{s, f\}$ ale ještě neexistuje, proto jej musíme vytvořit jako nový:



- Obdobným způsobem přidáme hrany novému stavu $\{s, f\}$:



- Až dospějeme do situace, kdy jsme ke všem stavům již všechny hrany přidali a přitom nevznikl žádný nový stav (což nastalo nyní), máme konstrukci hotovou. Koncové stavy jsou ty stavy, které „obsahují“ aspoň jeden koncový stav z původního automatu. Výsledný deterministický automat tedy je:



Příklad 8.
Vytvořte regulární výraz r nad abecedou $\Sigma = \{a, b\}$, který popisuje jazyk všech řetězců obsahující podřetězec aa. Formálně: $L(r) = \{x: aa \text{ is substring of } x\}$

Řešení:
Regulární výraz $r = (a + b)^* aa(a + b)^*$ popisuje tento jazyk.

Příklad 9.
Vytvořte regulární výraz r nad abecedou $\Sigma = \{a, b\}$, který popisuje jazyk všech řetězců neobsahující aa. Formálně: $L(r) = \{x: aa \text{ is not substring of } x\}$

Řešení:
Je zřejmé, že můžeme libovolně za sebe kopírovat symbol (respektive řetězec) b, aniž bychom vytvořili podřetězec aa. Pokud chceme vložit symbol a, musíme nejprve zaručit, že za symbolem a se nebude nacházet další symbol a, tedy bude se nacházet symbol b. To zaručíme tak, že za symbol a pevně „vnutíme“ symbol b. Tedy můžeme jakýmsi způsobem za sebe libovolně skládat řetězce b a ab, aniž bychom vytvořili řetězec se dvěma symboly a za sebou. Regulárním výrazem můžeme popsat tuto skutečnost jako: $(b + ab)^*$
Tento regulární výraz však ještě není přesný – pokud chceme vložit symbol a, pevně vnutí symbol b. Tímto regulárním výrazem tedy ještě nepopisujeme žádný řetězec, který končí symbolem a. Například ba je zřejmě korektně vytvořený řetězec neobsahují podřetězec aa. To můžeme ošetřit například tak, že za již vytvořený regulární výraz přidáme $(\varepsilon + a)$, což nám říká, že na konec vytvořeného řetězce máme právo buď přidat pouze jeden! symbol a a nebo nic.
Výsledný regulární výraz tedy je: $r = (b + ab)^*(\varepsilon + a)$

Příklad 10.
Vytvořte regulární výraz r nad abecedou $\Sigma = \{a, b, c\}$, který popisuje jazyk všech řetězců neobsahující aba. Formálně: $L(r) = \{x: aba \text{ is not substring of } x\}$

Řešení:
Postupovat budeme obdobně jako v příkladu 8:

Řešení „hlavního těla“ výrazu:
- Řetězece b a c můžeme libovolně za sebe kopírovat, aniž by vznikl podřetězec aba.
- Pokud je vložen jeden nebo i více po sobě symbolů a, musí za nimi následovat: řetězec c nebo řetězec bb nebo řetězec bc

Regulárním výrazem můžeme popsat tuto skutečnost jako: $(b + c + a^+c + a^+bb + a^+bc)^*$

Řešení „konce“ výrazu:
Nyní procházejme jednotlivé elementy těla výrazu:
- a^+c – vnuceno c, ale můžeme zakončit pouze sekvencí symbolů a, tj. a^+
- a^+bb – vnuceno bb, ale můžeme zakončit pouze sekvencí symbolů a ukončenou jedním nebo žádným symbolem b, tj. a^+b nebo a^+
- a^+bc – vnuceno bc, ale můžeme zakončit pouze sekvencí symbolů a ukončenou jedním nebo žádným symbolem b, tj. a^+b nebo a^+

Tedy řetězec můžeme zakončit I. prázdným řetězcem, II. sekvencí symbolů a, III. Sekvencí symbolů a následovanou jedním symbolem b. Regulárním výrazem můžeme popsat tuto skutečnost jako: $(\varepsilon + a^+ + a^+b)$, což lze zjednodušit na: $(a^* + a^+b)$.
Výsledný regulární výraz tedy je: $r = (b + c + a^+c + a^+bb + a^+bc)^*(a^* + a^+b)$

3. Pumping lemma, uzávěrové vlastnosti

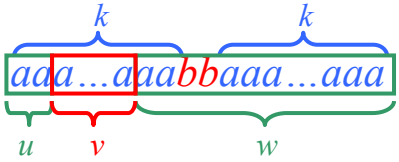
Slabé pumping lemma:
Nechť L je regulární jazyk. Potom existuje nějaká konstanta $k \geq 1$, že pro každý řetězce $z \in L$, kde $|z| \geq k$ platí, že jej můžeme rozdělit na podřetězce u, v, w, tedy $z = uvw$, které splňují následující podmínky: **1)** $v \neq \varepsilon$ **2)** $|v| \leq k$ **3)** $uv^m w \in L$ pro všechna $m \geq 0$.

Silné pumping lemma: *(učí se na přednáškách IFJ od roku 2005; silnější 2) podmínka)*
Nechť L je regulární jazyk. Potom existuje nějaká konstanta $k \geq 1$, že pro každý řetězce $z \in L$, kde $|z| \geq k$ platí, že jej můžeme rozdělit na podřetězce u, v, w, tedy $z = uvw$, které splňují následující podmínky: **1)** $v \neq \varepsilon$ **2)** $|uv| \leq k$ **3)** $uv^m w \in L$ pro všechna $m \geq 0$.

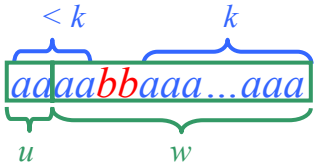
Příklad 1.
Pomocí pumping lemma dokažte, že jazyk $L_1 = \{a^n bba^n : n \geq 0\}$ není regulární.

Řešení (pomocí slabého PL):
1. Předpokládejme, že L_1 je regulární. Potom pro jazyk L_1 platí pumping lemma.
2. Nechť $\boldsymbol{k}$ je konstanta z pumping lemmy. Nyní vyberme „vhodný“ řetězec z, který splňuje podmínky: **a)** $z \in L_1$ **b)** $|z| \geq \boldsymbol{k}$.
Položme tedy například $z = a^k bba^k$. Zřejmě $a^k bba^k \in L_1$ a $|a^k bba^k| = \boldsymbol{k} + 1 + 1 + \boldsymbol{k} = 2\boldsymbol{k} + 2 > \boldsymbol{k}$.
Obě podmínky **a)** i **b)** tedy jsou splněny.
3. Podle pumping lemma můžeme tedy řetězec $z = a^k bba^k$ „nějak“ rozložit na tři podřetězce u, v, w, tak, aby platilo $z = uvw$. Nyní musíme projít **všechny možnosti** rozložení řetězce z a u každé z nich najít **aspoň jeden** případ, který vede ke sporu.

I. možnost:

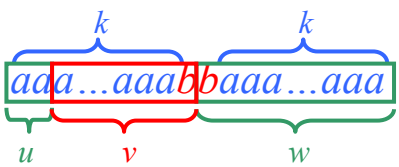


Formálně: $v = a^i$, $i \geq 1$; bb is substring of w

- Například pro řetězec $uv^0w = uw =$ 

- Podle pumping lemma $uv^0w \in L_1$
- Podle definice jazyka L_1 ale $uv^0w \notin L_1$, protože nesouhlasí počet „a“ (počet „a“ před podřetězcem bb je menší než počet „a“ za podřetězcem bb.
- **SPOR**

II. možnost:

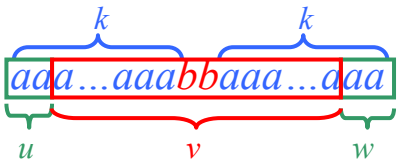


Formálně: $v = a^i b$, $i \geq 0$;

- Například pro řetězec $uv^2w = uvvw =$ 

- Podle pumping lemma $uv^2w \in L_1$
- Podle definice jazyka L_1 ale $uv^2w \notin L_1$, protože tento řetězec obsahuje tři symboly „*b*“.
- **SPOR**

III. možnost:

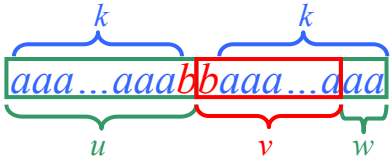


Formálně: $v = a^i bba^j$, $i, j \geq 0$;

- Například pro řetězec $uv^0w = uw =$ 

- Podle pumping lemma $uv^0w \in L_1$
- Podle definice jazyka L_1 ale $uv^0w \notin L_1$, protože tento řetězec neobsahuje podřetězec *bb*.
- **SPOR**

IV. možnost:

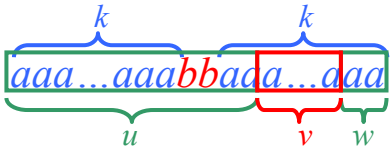


Formálně: $v = ba^i$, $i \geq 0$;

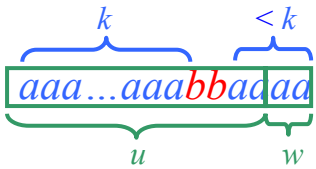
- Například pro řetězec $uv^2w = uvvw =$ 

- Podle pumping lemma $uv^2w \in L_1$
- Podle definice jazyka L_1 ale $uv^2w \notin L_1$, protože tento řetězec obsahuje tři symboly „*b*“.
- **SPOR**

V. možnost:



Formálně: $v = a^i$, $i \geq 1$; *bb* is substring of *u*

- Například pro řetězec $uv^0w = uw =$ 

- Podle pumping lemma $uv^0w \in L_1$
- Podle definice jazyka L_1 ale $uv^0w \notin L_1$, protože nesouhlasí počet „*a*“ (počet „*a*“ před podřetězcem *bb* je větší než počet „*a*“ za podřetězcem *bb*.
- **SPOR**

- Dokázali jsme sporem, že neexistuje žádné rozložení řetězce z na podřetězce u, v, w, kde $z = uvw$ tak, aby pro všechna $m \geq 0$ platilo $uv^m w \in L_1$. Předpoklad, že L_1 je regulární, je nesprávný. (Jinak by toto rozložení podle pumping lemma muselo existovat)
  Proto tedy L_1 není regulární jazyk.

Poznámka: Použitím silného pumping lemma je možné provést pouze první možnost dekompozice, protože ostatní ihned porušují 2. podmínku: $|uv| \leq k$.

Příklad 2.
Pomocí výsledku z příkladu 1) a pomocí uzávěrových vlastností dokažte, že jazyk $L_2 = \{xy: x, y \in \{a, b\}^* \wedge y = \text{reversal}(x)\}$ není regulární. Neformálně, jazyk L_2 obsahuje všechny řetězce obsahující pouze symboly a, b, pro které platí, že přečtená druhá polovina řetězce od konce tvoří první polovinu řetězce.

Řešení:
Důkaz sporem:

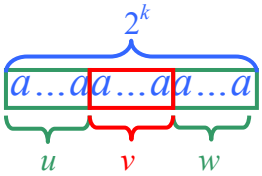
- Předpokládejme, že jazyk L_2 je regulární
- Regulární výraz $r = a^*bba^*$ zřejmě popisuje jistý regulární jazyk $L(r)$
- Regulární jazyky jsou uzavřeny vůči průniku, což znamená: Pro **libovolné** dva regulární jazyky L_a a L_b platí: $L_a \cap L_b$ je opět **jazyk regulární**.
- Konkrétně: L_2 je regulární (předpoklad), $L(r)$ je regulární (je popsán regulárním výrazem), jazyk $L_2 \cap L(r) = \{a^n bba^n: n \geq 0\}$ je tedy regulární.
  Ale v příkladu 1. jsme dokázali, že $\{a^n bba^n: n \geq 0\}$ není regulární $\Rightarrow$ **SPOR**.
- Předpoklad, že jazyk L_2 je regulární, je nesprávný.
- **Proto tedy L_2 není regulární jazyk.**

Poznámka: Nejsložitější částí příkladu je „vhodné“ zvolení regulárního výrazu r, tak aby po průniku s jazykem v zadání vznikl „jednodušší“ jazyk, u kterého lze efektivně pomocí pumping lemma dokázat, že není regulární.

Příklad 3.
Pomocí pumping lemma dokažte, že jazyk $L_3 = \{a^{2^n} : n \geq 0\}$ není regulární.

Řešení (Objevitelkou tohoto hezkého řešení je Jana Brychová, studentka 1MIT):
1. Předpokládejme, že L_3 je regulární. Potom pro jazyk L_3 platí pumping lemma.
2. Nechť $\boldsymbol{k}$ je konstanta z pumping lemmy. Nyní vyberme „vhodný“ řetězec z, který splňuje podmínky: **a)** $z \in L_3$ **b)** $|z| \geq \boldsymbol{k}$.
Položme tedy například $z = a^{2^k}$. Zřejmě $a^{2^k} \in L_3$ a $|a^{2^k}| = 2^k \geq k$. Obě podmínky **a)** i **b)** tedy jsou splněny.
3. Podle pumping lemma můžeme tedy řetězec $z = a^{2^k}$, „nějak“ rozložit na tři podřetězce u, v, w, tak, aby platilo $z = uvw$. Nyní musíme projít **všechny možnosti** rozložení řetězce z a u každé z nich najít **aspoň jeden** případ, který vede ke sporu.

I. možnost (jediná):



Formálně: $v = a^i$, $i \geq 1$;

Zřejmě $|uw| \geq 1$, neboť $|uw| = 0$ by implikovalo, že $|v| = 2^k$, což ihned dává spor s podmínkou $|v| \leq k$ u slabého pumping lemma nebo s podmínkou $|uv| \leq k$ u silného pumping lemma.

- $|z| = |uvw| = |uw| + |v| = 2^k$. Vynásobením této rovnice číslem 2 obdržíme: $2|uw| + 2|v| = 2.2^k$, tedy: $2|uw| + 2|v| = 2^{k+1}$ …[1]
- Podle pumping lemma $uv^2w \in L_3$, tedy $|uv^2w| = |uw| + 2|v| = 2^n$ …[2] kde $n > k$, neboť $|uv^2w| > |uvw|$.
- Odečtením [1] – [2] dostáváme: $2|uw| + 2|v| - (|uw| + 2|v|) = 2^{k+1} - 2^n$, tedy: $|uw| = 2^{k+1} - 2^n$
  Protože $|uw| \geq 1$, musí platit $2^{k+1} > 2^n$, čili $k+1 > n$ → **Spor! (neexistuje přirozené číslo *n* mezi čísly *k* a *k*+1)**

- Dokázali jsme sporem, že neexistuje žádné rozložení řetězce z na podřetězce u, v, w, kde $z = uvw$ tak, aby pro všechna $m \geq 0$ platilo $uv^mw \in L_3$. Předpoklad, že L_3 je regulární, je nesprávný (Jinak by toto rozložení podle pumping lemma muselo existovat).
  Proto tedy L_3 není regulární jazyk.

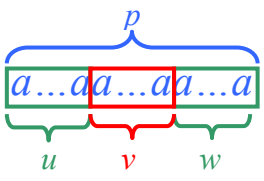
Příklad 4.
Pomocí výsledku z příkladu 3) a pomocí uzávěrových vlastností dokažte, že jazyk $L_4 = \{x: x \in \{a, b, c\}^* \wedge |x| = 2^n, n \geq 0\}$ není regulární. Neformálně, jazyk L_4 obsahuje všechny řetězce obsahující pouze symboly a, b, c, pro které platí, že jejich délka je ve tvaru 2^n, $n \geq 0$.

Řešení:
Důkaz sporem:
- Předpokládejme, že jazyk L_4 je regulární
- Regulární výraz $r = a^*$ zřejmě popisuje jistý regulární jazyk $L(r)$
- Regulární jazyky jsou uzavřeny vůči průniku, což znamená: Pro **libovolné** dva regulární jazyky L_a a L_b platí: $L_a \cap L_b$ je opět **jazyk regulární**.
- Konkrétně: L_4 je regulární (předpoklad), $L(r)$ je regulární (je popsán regulárním výrazem), jazyk $L_4 \cap L(r) = \{a^{2^n} : n \geq 0\}$ je tedy regulární.
  Ale v příkladu 3. jsme dokázali, že $\{a^{2^n} : n \geq 0\}$ není regulární ⇒ **SPOR**.
- Předpoklad, že jazyk L_4 je regulární, je nesprávný.
- **Proto tedy L_4 není regulární jazyk.**

Příklad 5.
Pomocí pumping lemma dokažte, že jazyk $L_5 = \{a^n: n$ je prvočíslo$\}$ není regulární.

Řešení:
1. Předpokládejme, že L_5 je regulární. Potom pro jazyk L_5 platí pumping lemma.
2. Nechť k je konstanta z pumping lemmy. Nyní vyberme „vhodný“ řetězec z, který splňuje podmínky: **a)** $z \in L_5$ **b)** $|z| \geq k$.
Položme tedy například $z = a^p$, kde p je první prvočíslo splňující podmínku $p > k$.
Zřejmě $a^p \in L_5$ a $|a^p| = p \geq k$. Obě podmínky **a)** i **b)** tedy jsou splněny.
3. Podle pumping lemma můžeme tedy řetězec $z = a^p$, kde p je prvočíslo, „nějak“ rozložit na tři podřetězce u, v, w, tak, aby platilo $z = uvw$. Nyní musíme projít **všechny možnosti** rozložení řetězce z a u každé z nich najít **aspoň jeden** případ, který vede ke sporu.

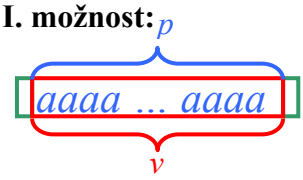
Mohli bychom postupovat zcela obecně a vzít pouze jedinou možnost (jako v příkladu 3.):



Formálně: $v = a^i$, $i \geq 1$;

Takto důkaz sice je možné provést, ale je příliš komplikovaný, proto tento případ rozdělme na následující tři možnosti:

I. možnost:

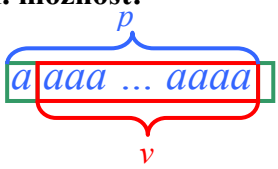


Formálně: $v = a^i$, $i \geq 1$;
$u, w = \varepsilon$, tedy: $|uw| = 0$

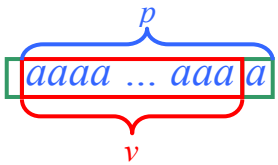
- Například pro řetězec $uv^0w = uw =$  $= \varepsilon$:

- Podle pumping lemma $uv^0w \in L_5$
- $uv^0w = \varepsilon$, $|uv^0w| = 0$. Podle definice jazyk L_5 obsahuje pouze řetězce, které mají délku prvočísla. 0 ale není prvočíslo, tedy $uv^0w \notin L_5$.
- **SPOR**

II. možnost:



nebo

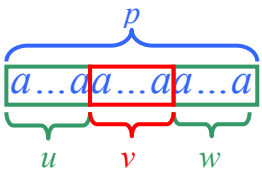


Formálně: $v = a^i$, $i \geq 1$;
$u = a$, $w = \varepsilon$ nebo $u = \varepsilon$, $w = a$,
tedy: $|uw| = 1$

- Například pro řetězec $uv^0w = uw =$  nebo  $= a$:

- Podle pumping lemma $uv^0w \in L_5$
- $uv^0w = a$, $|uv^0w| = 1$. Podle definice jazyk L_5 obsahuje pouze řetězce, které mají délku prvočísla. 1 ale není prvočíslo, tedy $uv^0w \notin L_5$.
- **SPOR**

III. možnost (všechny ostatní případy):



, kde $|uw| \geq 2$,

Formálně: $v = a^i$, $i \geq 1$;
$|uw| \geq 2$

Položme $m = |uw|$.

- Podle pumping lemma $uv^mw \in L_5$. (Platí pro libovolné $m \geq 0$, tedy i pro $m = |uw|$).
- $|uv^mw| = \underbrace{|uw|}_{m} + |v^m| = m + m|v| = \underbrace{m}_{= |uw| \geq 2} . \underbrace{(1 + |v|)}_{\geq 2}$, neboť $|v| \geq 1$

Je tedy vidět, že $|uv^m w|$ můžeme rozložit na součin dvou přirozených čísel, z nichž obě jsou větší nebo rovny číslu 2. Zřejmě tedy $|uv^m w|$ není prvočíslo, neboť prvočísla mají dělitele pouze 1 a sama sebe. Podle definice jazyk L_5 obsahuje pouze řetězce, které mají délku prvočísla. $|uv^m w|$ ale není prvočíslo, tedy $uv^m w \notin L_5$.

- **SPOR**
- Dokázali jsme sporem, že neexistuje žádné rozložení řetězce z na podřetětězce u, v, w, kde $z = uvw$ tak, aby pro všechna $m \geq 0$ platilo $uv^m w \in L_5$. Předpoklad, že L_5 je regulární, je nesprávný (Jinak by toto rozložení podle pumping lemma muselo existovat).
  Proto tedy L_5 není regulární jazyk.

Příklad 6.
Přednášky ukázaly pomocí pumping lemma, že jazyk $\{a^n b^n : n \geq 0\}$ není regulární. S využitím tohoto poznatku a uzávěrových vlastností dokažte, že ani jazyk $L_6 = \{x : x \in \{a, b\}^*, \#_a x = \#_b x\}$ není regulární.

Poznámka: $\#_a x$ obecně znamená počet výskytů symbolů a v řetězci x, tedy podmínka $\#_a x = \#_b x$ popisuje skutečnost, že počty výskytů symbolů a i b jsou v řetězci x stejné.

Řešení:
Důkaz sporem:

- Předpokládejme, že jazyk L_6 je regulární
- Regulární výraz $r = a^* b^*$ zřejmě popisuje jistý regulární jazyk $L(r)$
- Regulární jazyky jsou uzavřeny vůči průniku, což znamená: Pro **libovolné** dva regulární jazyky L_a a L_b platí: $L_a \cap L_b$ je opět **jazyk regulární**.
- Konkrétně: L_6 je regulární (předpoklad), $L(r)$ je regulární (je popsán regulárním výrazem). Jazyk $L_6 \cap L(r) = \{a^n b^n : n \geq 0\}$ je tedy regulární.
  Ale v přednášky ukázali, že $\{a^n b^n : n \geq 0\}$ není regulární $\Rightarrow$ **SPOR**.
- Předpoklad, že jazyk L_6 je regulární, je nesprávný.
- **Proto tedy L_6 není regulární jazyk.**

Příklad 7.
Nechť L_7 je jazyk nad abecedou $\Sigma = \{$ „(“ , „)“ $\}$ obsahující všechny řetězce, které vzniknou z aritmetických výrazů vypuštěním všech symbolů kromě závorek. Například **( )(( )( ))** $\in L_7$, neboť tento řetězec vznikl například z aritmetického výrazu **(**$a + b$**)**.**((**$a + b$**)** – **(**c + d**))**.

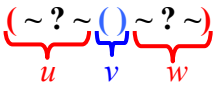
a) Dokažte, že tento jazyk není regulární
b) Ukažte, že tento jazyk splňuje podmínky slabého pumping lemma pro regulární jazyky
c) Jaká značná nepříjemnost vyplývá z podmínek a) a b)?

Řešení:
a) Důkaz sporem:

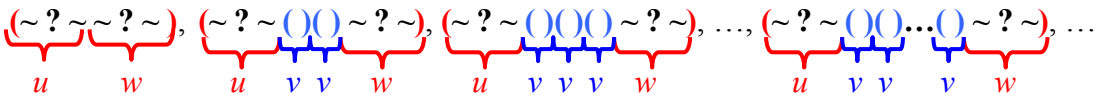
- Předpokládejme, že jazyk L_7 je regulární
- Regulární výraz $r =$ „(“* „)“* zřejmě popisuje jistý regulární jazyk $L(r)$
- Regulární jazyky jsou uzavřeny vůči průniku, což znamená: Pro **libovolné** dva regulární jazyky L_a a L_b platí: $L_a \cap L_b$ je opět **jazyk regulární**.
- Konkrétně: L_7 je regulární (předpoklad), $L(r)$ je regulární (je popsán regulárním výrazem). Jazyk $L_7 \cap L(r) = \{$„(“^{n}„)“$^n : n \geq 0\}$ je tedy regulární.
- Přednášky ale ukázaly, že $\{a^n b^n : n \geq 0\}$ není regulární, tedy ani $\{$„(“^{n}„)“$^n : n \geq 0\}$ není regulární $\Rightarrow$ **SPOR**.

- Předpoklad, že jazyk L_7 je regulární, je nesprávný.
- **Proto tedy L_7 není regulární jazyk.**

b) Dokažme, že pro jazyk L_7 platí slabší pumping lemma: Zvolme konstantu $k = 2$. Zřejmě musí každý „závorkový“ řetězec patřící do jazyka L_7 délky větší nebo rovny číslu 2 obsahovat podřetězec „( )“ jakožto nějaký pár závorek, které již v sobě jiné závorky neobsahují. Proveďme následující dekompozici daného řetězce na podřetězce $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$ tak, aby řetězec v = „( )“:



Zřejmě i řetězce:



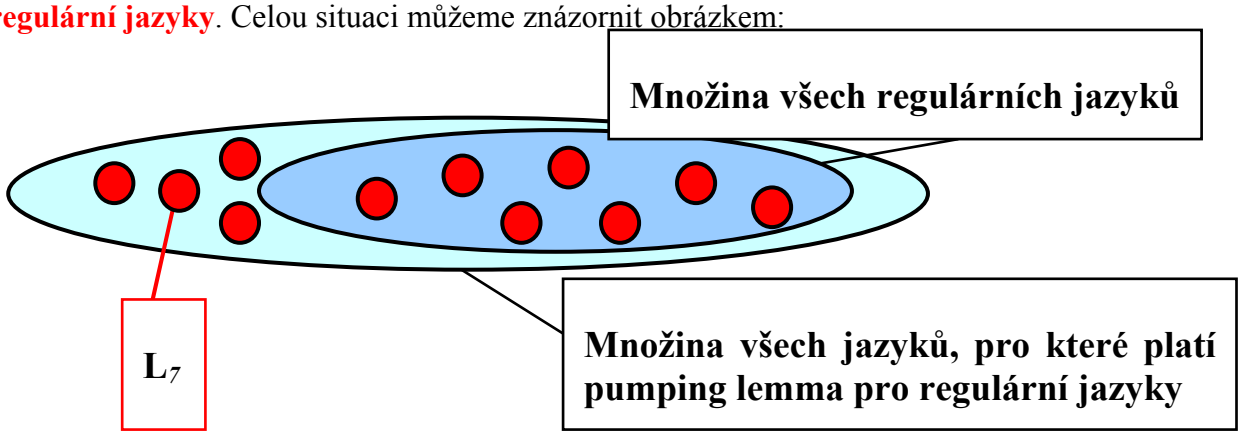
jsou obsaženy v jazyce L_7, tedy platí:

1) $v \neq \varepsilon$
2) $|v| \leq k$ } neboť $|v| = 2$, $k = 2$
3) $uv^m w \in L_7$ pro všechna $m \geq 0$.

- Pro daný jazyk L_7 tedy platí pumping lemma pro regulární jazyky.

c) Zřejmě pro každý regulární jazyk platí pumping lemma pro regulární jazyky (viz. přednášky). Jazyk L_7 není regulární (viz. část a) a přesto pro něj pumping lemma pro regulární jazyky platí (viz. část b).

- **Existují tedy jazyky, které nejsou regulární, a přesto pro ně platí pumping lemma pro regulární jazyky**. Celou situaci můžeme znázornit obrázkem:



Nepříjemný důsledek: Pumping lemma pro regulární jazyky tedy pouze říká:

- **POKUD** L je regulární jazyk, **POTOM** pro něj platí podmínky PL

A NE:

- ~~**POKUD** pro daný jazyk L platí podmínky PL **POTOM** L je regulární jazyk~~

Například jazyk L_7 splňuje podmínky pumping lemma pro regulární jazyky a přesto tento jazyk není regulární !!!

Závěr:
Pomocí pumping lemma nemůžeme nikdy dokázat, že daný jazyk je regulární!!!

4. Bezkontextové gramatiky, zásobníkové automaty

Příklad 1.
Vytvořte bezkontextovou gramatiku, která generuje jazyk $L_1 = \{a^n b^n : n \geq 1\}$. Pomocí této gramatiky proveďte derivaci řetězce ***aaabbb***.

Řešení:
Gramatika $G = (N, T, P, S)$, kde:

- $N = \{S\}$,
- $T = \{a, b\}$,
- $P = \{1: S \to aSb,\ 2: S \to ab\}$

generuje tento jazyk. Řetězec ***aaabbb*** potom můžeme vygenerovat pomocí následující posloupností derivací:

$$S \Rightarrow aSb\ [1] \Rightarrow aaSbb\ [1] \Rightarrow aaabbb\ [2]$$

Nebo zkráceně, pokud nás explicitně nezajímá použité pravidlo, můžeme jejich označení vynechat:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$$

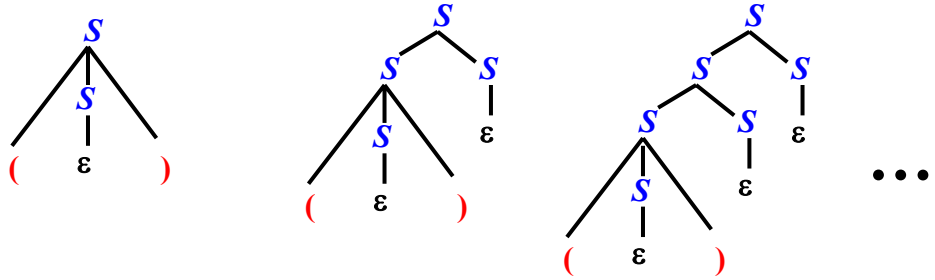
Příklad 2.
Nechť L_2 je jazyk nad abecedou Σ = { „(“ , „)“ } obsahující všechny řetězce, které vzniknou z aritmetických výrazů vypuštěním všech symbolů kromě závorek. Například **( )(( )( ))** $\in L_2$, neboť tento řetězec vznikl například z aritmetického výrazu **(***a* + *b***)**.**((***a* + *b***)** – **(**c + d**))**. Je vámi navržená gramatika jednoznačná? Pokud ne, zkuste ji zmodifikovat tak, aby jednoznačná byla.

Řešení:
Gramatika $G = (N, T, P, S)$, kde:

- $N = \{S\}$,
- $T = \{(,)\}$,
- $P = \{1: S \to SS,\ 2: S \to (S),\ 3: S \to \varepsilon\}$

generuje tento jazyk.

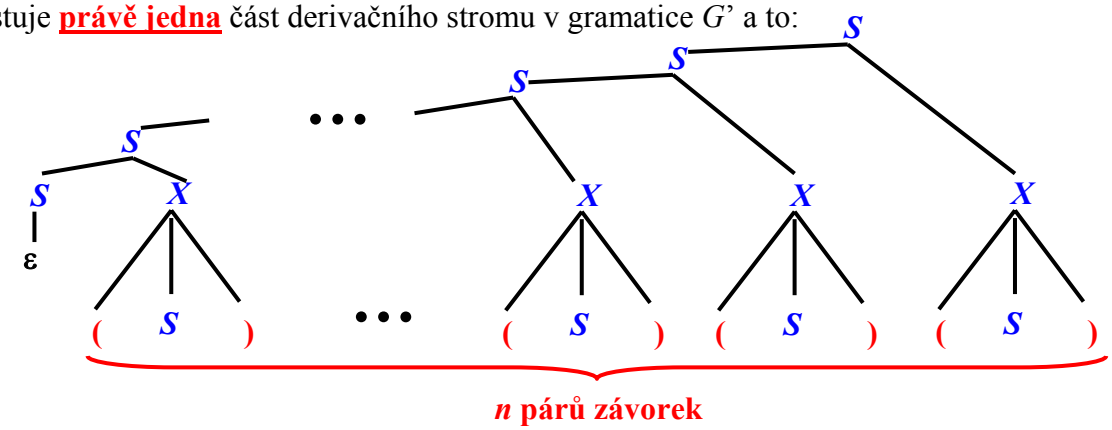
Tato gramatika **není jednoznačná**, protože již například pro řetězec **( )** existuje nekonečně mnoho derivačních stromů (pro nejednoznačnost gramatiky by stačily pouze dva):



Ekvivalentní jednoznačná gramatika k vytvořené gramatice G je například:
$G' = (N, T, P, S)$, kde:

- $N = \{S, X\}$,
- $T = \{(,)\}$,
- $P = \{1: S \to SX,\ 2: X \to (S),\ 3: S \to \varepsilon\}$

Uvažujme, že závorková struktura obsahuje na nejvyšší úrovni právě n párů závorek. Potom existuje **právě jedna** část derivačního stromu v gramatice G' a to:



***n* párů závorek**

Tyto derivační stromy můžeme do sebe zanořovat a tím budou vznikat libovolné závorkové struktury různých úrovní. Na závěr budou přepsány všechny zbývající nonterminální symboly pomocí pravidla 3: $S \to \varepsilon$.
Je tedy vidět, že pro libovolnou závorkovou strukturu můžeme vytvořit v G' pouze **jeden** derivační strom. Gramatika G' je tedy jednoznačná. Jazyk L_2 **není jazyk s inherentní nejednoznačností**, protože existuje jednoznačná gramatika G', která jej generuje.

Příklad 3.
Vytvořte bezkontextovou gramatiku, která generuje jazyk $L_3 = \{xy: x, y \in \{a, b\}^* \wedge y = \text{reversal}(x)\}$. Neformálně, jazyk L_3 obsahuje všechny řetězce obsahující pouze symboly a, b, pro které platí, že přečtená druhá polovina řetězce od konce tvoří první polovinu řetězce.

Řešení:
Gramatika $G = (N, T, P, S)$, kde:

- $N = \{S\}$,
- $T = \{a, b\}$,
- $P = \{1: S \to aSa,\ 2: S \to bSb,\ 3: S \to \varepsilon\}$

generuje tento jazyk.

Poznámka: Řetězec, který se čte stejně zepředu i pozpátku, se nazývá palindrom. Jazyk L_3 tedy obsahuje všechny palindromy sudé délky nad abecedou $\Sigma = \{a, b\}$.

Příklad 4.
Vytvořte bezkontextovou gramatiku, která generuje jazyk $L_4 = \{x: x \in \{a, b\}^* \wedge \#_a x > \#_b x\}$.

Poznámka: $\#_a x$ obecně znamená počet výskytů symbolů a v řetězci x, tedy podmínka $\#_a x > \#_b x$ popisuje skutečnost, že počet výskytů symbolů a v řetězci x je větší než počet symbolů b.

Řešení:
Gramatika $G = (N, T, P, S)$, kde:

- $N = \{S\}$,
- $T = \{a, b\}$,
- $P = \{1: S \to a,\ 2: S \to aS,\ 3: S \to bSS\ \ 4: S \to SbS,\ 5: S \to SSb\}$

generuje tento jazyk.

Pravidla 3, 4 a 5 vygenerují libovolnou větnou formu, která obsahuje pouze nonterminální symboly S a terminální symboly b, přičemž počet nonterminálních symbolů S je o jedničku větší než počet terminálních symbolů b. Potom pravidla 1 a 2 dokončí derivaci řetězce tak, že každý nonterminál S je přepsán sekvencí terminálních symbolů a, přičemž tato sekvence obsahuje aspoň jeden terminální symbol a. Celkem můžeme tedy vygenerovat všechny řetězce, které obsahují více symbolů a než symbolů b.

Příklad 5.
Sestrojte zásobníkový automat, který přijímá jazyk $L_1 = \{a^n b^n : n \geq 1\}$ z příkladu 1. s vyprázdněním zásobníku a s přechodem do koncového stavu.

Řešení:
Základní myšlenka činnosti tohoto automatu:

1) Zásobníkový automat bude číst symboly a ze vstupu a všechny je ukládat na zásobník. Tuto skutečnost zařídíme pravidly:
   $Ssa \rightarrow Sas$, $asa \rightarrow aas$
   (**Poznámka:** první pravidlo je použito při čtení prvního symbolu a, protože na zásobníku je uložen pouze startující zásobníkový symbol S. Při dalším čtení budou již na zásobníku nějaké symboly a a použito bude tedy druhé pravidlo.)

2) Jakmile automat přečte na vstupu první symbol b, vytáhne jeden symbol a ze zásobníku a přejde do jiného stavu q. Přechodem do jiného stavu q zabezpečíme, že automat již nikdy nebude číst ze vstupu symboly a. Při dalším čtení symbolu b opět vytáhneme jeden symbol a ze zásobníku ale již setrváme ve stavu q. Tuto skutečnost zařídíme pravidly:
   $asb \rightarrow q$, $aqb \rightarrow q$

3) Jakmile je celý řetězec přečtený, zkontrolujeme stav zásobníku a to následovně. Pokud řetězec patří do daného jazyka L_1, musí opět zásobník obsahovat pouze startující zásobníkový symbol. (Zásobník totiž obsahoval přesně tolik symbolů a, kolik jich bylo v první části řetězce. Při čtení symbolů b druhé části řetězce se vždy jeden symbol a ze zásobníku odstranil, tedy pokud jsou tyto počty stejné, nesmí být na něm žádný symbol a) Tuto kontrolu můžeme provést následujícím pravidlem, které dostane automat do koncového stavu f a vyprázdní mu zásobník:
   $Sq \rightarrow f$

Formální popis tohoto zásobníkového automatu:
$M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- $Q = \{s, q, f\}$,
- $\Sigma = \{a, b\}$,
- $\Gamma = \{S, a, b\}$,
- $R = \{Ssa \rightarrow Sas,\ asa \rightarrow aas,\ asb \rightarrow q,\ aqb \rightarrow q,\ Sq \rightarrow f\}$,
- $F = \{f\}$

Příklad 6.
Sestrojte zásobníkový automat, který přijímá jazyk L_2 z příkladu 2. s vyprázdněním zásobníku a přechodem do koncového stavu.

Řešení:

Základní myšlenka činnosti tohoto automatu:

1) Zásobníkový automat bude číst symboly (ze vstupu a všechny je ukládat na zásobník. Tuto skutečnost zařídíme pravidly:
   $Ss(\rightarrow S(s$, $(s(\rightarrow ((s$
2) Pokud zásobníkový automat přečte symbol), musí mít na zásobníku symbol (, který tvoří s tímto přečteným symbolem) závorkový pár. Tato závorka tedy bude odstraněna ze zásobníku pravidlem:
   $(s) \rightarrow s$

3) Jakmile je celý řetězec přečtený, zkontrolujeme, zda zásobník obsahuje pouze startující zásobníkový symbol. Pouze v tomto případě byly závorky správně popárovány. Přidáme tedy pravidlo:
   $Ss \rightarrow f$

Formální popis tohoto zásobníkového automatu:

$M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- $Q = \{s, f\}$,
- $\Sigma = \{(,)\}$,
- $\Gamma = \{S, (,)\}$,
- $R = \{Ss(\rightarrow S(s,\ (s(\rightarrow ((s,\ (s) \rightarrow s,\ Ss \rightarrow f\}$,
- $F = \{f\}$

Příklad 7.

Sestrojte rozšířený zásobníkový automat, který přijímá jazyk L_3 z příkladu 3. s vyprázdněním zásobníku a přechodem do koncového stavu.

Řešení:

Základní myšlenka činnosti tohoto automatu:

Automat bude pracovat nedeterministicky a to následujícím způsobem:

1) Rozšířený zásobníkový automat bude číst symboly a, b ze vstupu a všechny je ukládat na zásobník. Tuto skutečnost zařídíme pravidly:
   $sa \rightarrow as$, $sb \rightarrow bs$
   (**Poznámka:** Všimněte si, že u rozšířeného zásobníkového automatu nemusíme číst žádný symbol ze zásobníku)

2) Až bude mít automat přečtenu přesně polovinu řetězce, vloží na zásobník symbol C. POZOR, všimněte si, že rozšířený zásobníkový automat nemá žádný prostředek pro najití středu řetězce, neví, jak je ještě nepřečtená část dlouhá, proto je tento krok proveden !!!**NEDETERMINISTIKY!!!** pomocí pravidla:
   $s \rightarrow Cs$

3) Pokud má být druhá polovina řetězce stejná jako byla první reverzovaná první polovina, stačí postupně porovnávat následující čtené symboly se symboly na zásobníku a ty odstraňovat. Pozor, na vrcholu zásobníku je uložen symbol C, který nám říká, že jsme za polovinou čteného řetězce. První skutečný symbol je na zásobníku až pod tímto symbolem. Porovnávání můžeme tedy provést následujícími pravidly:
   $aCsa \rightarrow Cs$, $bCsb \rightarrow Cs$

4) Jakmile je celý řetězec přečtený, zkontrolujeme, zda zásobník obsahuje pod symbolem ***C*** pouze startující zásobníkový symbol pomocí pravidla:
   SCs → ***f***

Formální popis tohoto rozšířeného zásobníkového automatu:
$M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- $Q = \{s, f\}$,
- $\Sigma = \{a, b\}$,
- $\Gamma = \{S, a, b\}$,
- $R = \{sa \to as,\ sb \to bs,\ s \to Cs,\ aCsa \to Cs,\ bCsb \to Cs,\ SCs \to f\}$,
- $F = \{f\}$

Příklad 8.
Sestrojte zásobníkový automat, který přijímá jazyk L_4 z příkladu 4. s vyprázdněním zásobníku a přechodem do koncového stavu.

Řešení:
Základní myšlenka činnosti tohoto automatu:
Automat bude v každém kroku obsahovat na zásobníku kromě startujícího symbolu pouze symboly ***a*** nebo pouze symboly ***b***, ale nikdy nebude obsahovat oba druhy symbolů současně! A to následujícím způsobem: Nechť bylo do teď přečteno ze vstupu právě m symbolů ***a*** a n symbolů ***b***. Potom, pokud je $m > n$, bude zásobník obsahovat právě $m - n$ symbolů ***a***, pokud je $m < n$, bude zásobník obsahovat právě $n - m$ symbolů ***b*** a konečně pro případ $m = n$ nebude zásobník obsahovat žádné symboly ***a***, ***b***. Stručně a neformálně řečeno, zásobník obsahuje ty symboly, kterých bylo doposud přečteno více a tolik, o kolik jich bylo přečteno více než druhých. Toho dosáhneme následujícím způsobem:

1) Pokud je přečten symbol ze vstupu symbol ***a*** nebo ***b*** a zásobník je prázdný, vložíme tento symbol na zásobník pomocí pravidel:
   Ssa → ***Sas***, ***Ssb*** → ***Sbs***

2) Pokud je přečten ze vstupu symbol ***a*** a na vrcholu zásobníků již je symbol ***a***, přidáme symbol ***a*** na zásobník pomocí pravidla:
   asa → ***aas***

3) Pokud je přečten ze vstupu symbol ***b*** a na vrcholu zásobníků již je symbol ***b***, přidáme symbol ***b*** na zásobník pomocí pravidla:
   bsb → ***bbs***

4) Pokud je přečten ze vstupu symbol ***a*** a na vrcholu zásobníků je symbol ***b***, ubereme jeden symbol ***b*** ze zásobníku pro vyrovnání počtů pomocí pravidla:
   bsa → ***s***

5) Pokud je přečten ze vstupu symbol ***b*** a na vrcholu zásobníků je symbol ***a***, ubereme jeden symbol ***a*** ze zásobníku pro vyrovnání počtů pomocí pravidla:
   asb → ***s***

6) Jakmile je celý řetězec přečtený, zkontrolujeme, zda zásobník obsahuje „nějaké“ symboly ***a***. Jedině tehdy bylo totiž přečteno více symbolů ***a*** než ***b***. To zkontrolujeme následujícím pravidlem, pomocí kterého přejde automat do koncového stavu ***f***:
as → ***f***

7) Pokud je automat v koncovém stavu, necháme jej vyprázdnit zásobník užitím následujících dvou pravidel:
af → ***f***, ***Sf*** → ***f***,

Formální popis tohoto zásobníkového automatu:
$M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- $Q = \{s, f\}$,
- $\Sigma = \{a, b\}$,
- $\Gamma = \{S, a, b\}$,
- $R = \{Ssa \to Sas,\ Ssb \to Sbs,\ asa \to aas,\ bsb \to bbs,\ bsa \to s,\ asb \to s,\ as \to f,$
  $af \to f, Sf \to f\}$,
- $F = \{f\}$

(**Poznámka:** Pokud by stačilo, aby automat přijímal jazyk pouze přechodem do koncového stavu, mohli bychom poslední dvě pravidla odstranit)

Příklad 9.
Sestrojte gramatiku, která generuje jazyk $L_5 = \{a^m b^n c^n : m, n \geq 1\}$. Ke gramatice vytvořte:
a) Zásobníkový automat přijímající jazyk L_5 s vyprázdněním zásobníku simulující syntaktickou analýzu shora dolů.
b) Rozšířený zásobníkový automat přijímající jazyk L_5 s přechodem do koncového stavu simulující syntaktickou analýzu zdola nahoru.
Demonstrujte přijetí řetězce *abbcc* oběma automaty.

Řešení:
a) Gramatika $G = (N, T, P, S)$, kde:

- $N = \{S, A\}$,
- $T = \{a, b, c\}$,
- $P = \{1: S \to aS,\ 2: S \to aA,\ 3: A \to bAc,\ 4: A \to bc\}$

generuje tento jazyk.

b) Výsledný zásobníkový automat simulující syntaktickou analýzu shora dolů bude vypadat následovně:

1. Vytvoříme **porovnávací pravidla** následujícím způsobem: Pro každý terminální symbol ***t*** gramatiky *G* vytvoříme pravidlo tvaru ***tst*** → ***s***. Pro danou gramatiku *G* tedy vytvoříme pravidla:
   asa → ***s***, ***bsb*** → ***s***, ***csc*** → ***s***

2. Vytvoříme **expanzivní pravidla** následujícím způsobem: Pro každé pravidlo gramatiky tvaru ***A*** → ***x*** vytvoříme odpovídající pravidlo ***As*** → ***ys*** pro zásobníkový automat, přičemž ***y*** je reverzovaný řetězec ***x***, formálně: ***y*** = *reversal*(***x***). Pro danou gramatiku *G* tedy vytvoříme pravidla:
   Ss → ***Sas***, ***Ss*** → ***Aas***, ***As*** → ***cAbs***, ***As*** → ***cbs***

Formální popis tohoto zásobníkového automatu:
$M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- $Q = \{s\}$,
- $\Sigma = T = \{a, b, c\}$,
- $\Gamma = N \cup T = \{S, A, a, b\}$,
- $R = \{asa \to s, bsb \to s, csc \to s, Ss \to Sas, Ss \to Aas, As \to cAbs, As \to cbs\}$
- $F = \varnothing$

Poznámka: Všimněte si, že množina koncových stavů je prázdná, protože automat přijímá jazyk s vyprázdněním zásobníku a tedy koncové stavy tu nemají žádný význam

Simulace přijetí řetězce *abbcc* tímto zásobníkovým automatem:
$Ssabbcc \vdash Aasabbcc \vdash Asbbcc \vdash cAbsbbcc \vdash cAsbcc \vdash ccbsbcc \vdash ccscc \vdash csc \vdash s$

b) Výsledný rozšířený zásobníkový automat simulující syntaktickou analýzu zdola nahoru bude vypadat následovně:

1. Vytvoříme **shiftovací pravidla** následujícím způsobem: Pro každý terminální symbol t gramatiky G vytvoříme pravidlo tvaru $st \to ts$. Pro danou gramatiku G tedy vytvoříme pravidla:
   $sa \to as$, $sb \to bs$, $sc \to cs$

2. Vytvoříme **redukční pravidla** následujícím způsobem: Pro každé pravidlo gramatiky tvaru $A \to x$ vytvoříme odpovídající pravidlo $xs \to As$ pro rozšířený zásobníkový automat. Pro danou gramatiku G tedy vytvoříme pravidla:
   $aSs \to Ss$, $aAs \to Ss$, $bAcs \to As$, $bcs \to As$

3. Vytvoříme speciální pravidlo, které uvede automat do koncového stavu:
   $\#Ss \to f$

Formální popis tohoto rozšířeného zásobníkového automatu:
$M = (Q, \Sigma, \Gamma, R, s, S, F)$, kde

- $Q = \{s, f\}$,
- $\Sigma = T = \{a, b, c\}$,
- $\Gamma = N \cup T \cup \{\#\} = \{\#, S, A, a, b\}$,
- $R = \{sa \to as, sb \to bs, sc \to cs, aSs \to Ss, aAs \to Ss, bAcs \to As, bcs \to As, \#Ss \to f\}$
- $F = \{f\}$

Simulace přijetí řetězce *abbcc* tímto rozšířeným zásobníkovým automatem:
$\#sabbcc \vdash \#asbbcc \vdash \#absbcc \vdash \#abbscc \vdash \#abbcsc \vdash \#abAsc \vdash \#abAcs \vdash \#aAs \vdash \#Ss \vdash f$

5. Syntaktická analýza shora dolů

Příklad 1.
Uvažujte jednoduchý programovací jazyk, jehož syntaxi popisuje následující bezkontextová gramatika:
$G = (N, T, P, \langle prog \rangle)$, kde:

- N = {**<prog>**, **<st-list>**, **<stat>**, **<it-list>**, **<item>**},
- T = {**begin**, **end**, **;**, **read**, **write**, **add**, **:=**, **int**, **id**},
- P = {
  1: <prog> → begin <st-list> end
  2: <st-list> → <stat> ; <st-list>
  3: <st-list> → ε
  4: <stat> → read id
  5: <stat> → write <item>
  6: <stat> → id := add <item> <it-list>
  7: <stat> → ε *// prázdný příkaz*
  8: <it-list> → <item> <it-list>
  9: <it-list> → ε
  10: <item> → int
  11: <item> → id }

a) Sestrojte LL-tabulku pro danou gramatiku
b) Proveďte pomocí prediktivního syntaktického analyzátoru řízeného tabulkou z bodu a) syntaktickou analýzu vstupního řetězce **begin write int ; end**

Řešení:
a) Konstrukci LL-tabulky provedeme následujícím způsobem:

1) Nejprve vypočítáme množiny $First(X)$ a $Empty(X)$ pro každé $X \in N \cup T$ pomocí algoritmu:
   I) Pro každý terminální symbol $a \in T$ nejprve polož:
   - $First(a) = \{a\}$; $Empty(a) = \varnothing$

   II) Pro každý neterminální symbol $A \in N$ nejprve polož:
   - $First(A) = \varnothing$
   - Pokud existuje pravidlo $A \to \varepsilon \in P$ potom $Empty(A) = \{\varepsilon\}$ jinak $Empty(A) = \varnothing$

   III) Procházej postupně „ve vhodném pořadí“ jednotlivá pravidla (ne tvaru $A \to \varepsilon$) a dělej: nechť aktuální pravidlo má tvar $A \to X_1X_2\ldots X_{n-1}X_n$, potom:
   - Přidej všechny prvky z množiny $First(X_1)$ do množiny $First(A)$
   - Pokud $Empty(X_1) = \{\varepsilon\}$, potom:
     přidej všechny prvky z množiny $First(X_2)$ do množiny $First(A)$
   - Pokud $Empty(X_1) = Empty(X_2) = \{\varepsilon\}$, potom:
     přidej všechny prvky z množiny $First(X_3)$ do množiny $First(A)$
     …
   - Pokud $Empty(X_1) = Empty(X_2) = \ldots = Empty(X_{n-1}) = \{\varepsilon\}$, potom:
     přidej všechny prvky z množiny $First(X_n)$ do množiny $First(A)$
   - Pokud $Empty(X_1) = Empty(X_2) = \ldots = Empty(X_{n-1}) = Empty(X_n) = \{\varepsilon\}$, potom:
     polož **!!!** $Empty(A) = \{\varepsilon\}$ **!!!**

   IV) Pokud byla některá z množin změněna, znovu proveď krok III. pro všechna pravidla

Poznámky k algoritmu:

- Je vidět, že pro každý terminální symbol $a \in T$ platí i na závěr celého algoritmu $First(a) = \{a\}$; $Empty(a) = \varnothing$. Proto tyto množiny většinou nevypisujeme, ale pro výpočet tento fakt musíme znát. Stačí se tedy omezit na výpočet množin $First(A)$, $Empty(A)$ pro každý neterminální symbol $A \in N$.
- Doporučuji nejdříve kompletně určit množiny $Empty(A)$ pro každý neterminální symbol $A \in N$, potom až určit množiny $First(A)$ pro každý neterminální symbol A. Podstatně to urychlí výpočet těchto množin.
- Doporučuji pravidla seřadit do posloupnosti tak, aby platilo: Pokud i-té pravidlo má na levé straně neterminál A, pak každé následující pravidlo neobsahuje nikde na pravé straně neterminál A. Ne vždy to jde! Pokud to jde, pak *vhodné pořadí* pro krok III) je procházení této posloupnosti od posledního pravidla k prvnímu. Podstatně to urychlí výpočet množin $First(A)$. Pravidla gramatiky v tomto příkladu jsou tak seřazena.

Výpočet množin *First(X)* a *Empty(X)* pro výše uvedený příklad:

- Po provedení kroků I) a II) jsou tyto množiny následující:

First(**begin**)	= {**begin**}	*First*(**add**)	= {**add**}	*Empty*(**<prog>**)	= ∅
First(**end**)	= {**end**}	*First*(**:=**)	= {**:=**}	*Empty*(**<st-list>**)	= {ε}
First(**;**)	= {**;**}	*First*(**int**)	= {**int**}	*Empty*(**<stat>**)	= {ε}
First(**read**)	= {**read**}	*First*(**id**)	= {**id**}	*Empty*(**<it-list>**)	= {ε}
First(**write**)	= {**write**}			*Empty*(**<item>**)	= ∅

(Vypsány jsou pouze množiny *First* pro terminální symboly a množiny *Follow* pro neterminální symboly, neboť všechny ostatní množiny jsou po provedení kroků I) a II) vždy prázdné!)

- Dále provedeme krok III):
  - Nejprve dopočítáme kompletně množiny $Empty(A)$ pro všechna $A \in N$. Je vidět, že neexistuje již žádné pravidlo tvaru $A \to X_1X_2\ldots X_{n-1}X_n$, pro které by platilo $Empty(X_1) = Empty(X_2) = \ldots = Empty(X_{n-1}) = Empty(X_n) = \{\varepsilon\}$. Množiny $Empty(A)$ pro všechna $A \in N$ jsou již ve finálním tvaru.
  - Nyní dopočítáme kompletně množiny $First(A)$ pro všechna $A \in N$. Budeme tedy procházet postupně všechna pravidla gramatiky (ne ε-pravidla) od posledního k prvnímu (viz. třetí poznámka k algoritmu):

11: <item> → id {Přidáme *First*(**id**) do *First*(**<item>**)}
10: <item> → int {Přidáme *First*(**int**) do *First*(**<item>**)}

Celkové změny: **First(<item>) = {id, int}**

8: <it-list> → <item> <it-list> {Přidáme *First*(**<item>**) do *First*(**<it-list>**)}
Empty(**<item>**) ≠ {ε}, nic jiného tedy přidávat nebudeme

Celkové změny: **First(<it-list>) = {id, int}**

6: <stat> → id := add <item> <it-list> {Přidáme *First*(**id**) do *First*(**<stat>**)}
Empty(**id**) ≠ {ε}, nic jiného tedy přidávat nebudeme

5: **<stat>** → **write** **<item>** {Přidáme *First*(**write**) do *First*(**<stat>**)}
Empty(**write**) ≠ {ε}, nic jiného tedy přidávat nebudeme
4: **<stat>** → **read** **id** {Přidáme *First*(**read**) = do *First*(**<stat>**)}
Empty(**read**

přidej všechny prvky z množiny *First*(X_n) do množiny *First*($X_1X_2\ldots X_{n-1}X_n$)

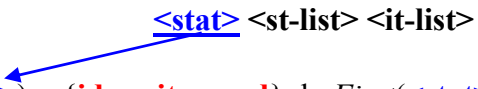
nechť $x = X_1X_2\ldots X_{n-1}X_n$, potom *Empty*($X_1X_2\ldots X_{n-1}X_n$) určíme:

- Pokud *Empty*(X_1) = *Empty*(X_2) = … = *Empty*(X_{n-1}) = *Empty*(X_n) = {ε}, potom: *Empty*($X_1X_2\ldots X_{n-1}X_n$) = {ε} jinak *Empty*($X_1X_2\ldots X_{n-1}X_n$) = ∅

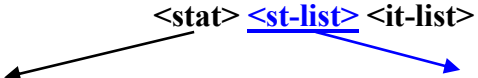
Poznámka: Pro prázdný řetězec ε dodefinujeme: *Empty*(ε) = {ε}; *First*(ε) = ∅

***Ilustrační příklad*:**
Určeme množiny *First*(<stat> <st-list> <it-list>) a *Empty*(<stat> <st-list> <it-list>).

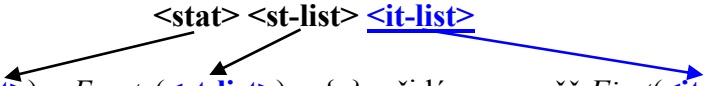
- *First*(<stat> <st-list> <it-list>) = ∅



- Přidáme *First*(<stat>) = {id, write, read} do *First*(<stat> <st-list> <it-list>)
  Celkové změny: *First*(<stat> <st-list> <it-list>) = {id, write, read}



- Protože *Empty*(<stat>) = {ε}, přidáme rovněž *First*(<st-list>) = {id, write, read, ;}
  do *First*(<stat> <st-list> <it-list>), tedy
  Celkové změny: *First*(<stat> <st-list> <it-list>) = {id, write, read, ;}



- Protože *Empty*(<stat>) = *Empty*(<st-list>) = {ε}, přidáme rovněž *First*(<it-list>) = {id, int}
- do *First*(<stat> <st-list> <it-list>), tedy
  Celkové změny: *First*(<stat> <st-list> <it-list>) = {id, write, read, ;, int}

  Výsledek: *First*(<stat> <st-list> <it-list>) = {id, write, read, ;, int}

- Protože *Empty*(<stat>) = *Empty*(<st-list>) = *Empty*(<it-list>), platí:
  Empty(<stat> <st-list> <it-list>) = {ε}

Poznámka: Na tomto příkladě je pouze ilustrován výpočet množin *First*(*x*) a *Empty*(*x*) pro konkrétní řetězec *x* = <stat> <st-list> <it-list>, což bude potřeba určovat pro různé řetězce při výpočtu množin *Follow* a *Predict* (viz. dále). Tento řetězec byl zvolen zcela náhodně pro vhodnou ilustraci výpočtu těchto množin, nemá tedy žádný konkrétní význam pro konstrukci *LL*-tabulky.

3) Nyní již můžeme vypočítat množinu *Follow*(***A***) pro každé $A \in N$ pomocí algoritmu:
I) Pro startující neterminální symbol ***S*** položíme *Follow*(***S***) = {$},
pro ostatní neterminály $A \in N$ položíme *Follow*(***A***) = ∅.

II) Procházej postupně „ve vhodném pořadí“ jednotlivá pravidla (obsahující na pravé straně aspoň jeden neterminální symbol) a pro každé z nich udělej všechny možné dekompozice na tvar: $A \to xBy$, kde $x, y \in (N \cup T)^*$, $B \in N$:

- Pokud $y \neq \varepsilon$, potom:

přidej všechny prvky z množiny *First*(***y***) do množiny *Follow*(***B***)

- Pokud *Empty*(***y***) = {ε} (tj. zahrnuje i možnost ***y*** = ε), potom:
  přidej všechny prvky z množiny *Follow*(***A***) do množiny *Follow*(***B***)

III) Pokud byla některá z množin změněna, znovu proveď krok II. pro všechna pravidla

Poznámky k algoritmu:

- Všimněte si, že množiny *First*(***y***) a *Empty*(***y***) musejí být určeny obecně pro **!řetězec!** ***y*** (ne pouze symbol).
- Pokud máme pravidla seřazena tak, jak bylo doporučeno v poznámce pro algoritmus výpočtu množin *First*(*X*) a *Empty*(*X*) pro každé $X \in N \cup T$, pak *vhodné pořadí* pro krok II. je procházení této posloupnosti pravidel od prvního k poslednímu.

Výpočet množin *Follow*(*A*) pro výše uvedený příklad:

- Po provedení kroku I) jsou tyto množiny následující:

Follow(**<prog>**)	= {**$**}
Follow(**<st-list>**)	= ∅
Follow(**<stat>**)	= ∅
Follow(**<it-list>**)	= ∅
Follow(**<item>**)	= ∅

- Nyní dopočítáme kompletně množiny *Follow*(***A***) pro všechna $A \in N$. Budeme tedy procházet postupně všechna pravidla gramatiky (ne pravidla, které neobsahují žádný neterminální symbol) od prvního k poslednímu (viz. druhá poznámka k algoritmu):

1: **<prog>** → **begin** **<st-list>** **end**
Veškeré dekompozice toho pravidla:

- **<prog>** → **begin** **<st-list>** **end**
  end ≠ ε
  {Přidáme *First*(**end**) = {**end**}do *Follow*(**<st-list>**)}

Celkové změny: *Follow*(**<st-list>**) = {**end**}

2: **<st-list>** → **<stat>** **;** **<st-list>**
Veškeré dekompozice toho pravidla:

- **<st-list>** → **<stat>** **;** **<st-list>**
  Empty(**ε**) = {ε}
  {Přidáme *Follow*(**<st-list>**) do *Follow*(**<st-list>**) // *nemá žádný význam* }
- **<st-list>** → **<stat>** **;** **<st-list>**
  ≠ ε
  {Přidáme *First*(**;** **<st-list>**) = {**;**} do *Follow*(**<stat>**) }

Celkové změny: *Follow*(**<stat>**) = {**;**}

5: **<stat>** → **write** **<item>**
Veškeré dekompozice toho pravidla:

- <stat> → **write** <item>

  Empty(ε) = {ε}

  {Přidáme *Follow*(<stat>) = {;} do *Follow*(<item>) }

Celkové změny: *Follow*(<item>) = {;}

6: **<stat> → id := add <item> <it-list>**

Veškeré dekompozice toho pravidla:

- <stat> → **id := add <item> <it-list>**

  Empty(ε) = {ε}

  {Přidáme *Follow*(<stat>) = {;} do *Follow*(<it-list>) }
- <stat> → **id := add <item>** <it-list>

  ≠ ε

  {Přidáme *First*(<it-list>) = {id, int} do *Follow*(<item>) }
- <stat> → **id := add <item>** <it-list>

  Empty(<it-list>) = {ε}

  {Přidáme *Follow*(<stat>) = {;} do *Follow*(<item>) // *ten tam již je* }

Celkové změny: *Follow*(<it-list>) = {;}, *Follow*(<item>) = {id, int, ; }

8: **<it-list> → <item> <it-list>**

Veškeré dekompozice toho pravidla:

- <it-list> → **<item> <it-list>**

  Empty(ε) = {ε}

  {Přidáme *Follow*(<it-list>) do *Follow*(<it-list>) // *nemá žádný význam* }
- <it-list> → **<item>** <it-list>

  ≠ ε

  {Přidáme *First*(<it-list>) = {id, int} do *Follow*(<item>) // *ty tam již jsou* }
- <it-list> → **<item>** <it-list>

  Empty(<it-list>) = {ε}

  {Přidáme *Follow*(<it-list>) = {;} do *Follow*(<item>) // *ten tam již je* }

Jednotlivé množiny pro neterminální symboly po provedení jednoho cyklu ve II. kroku algoritmu:

Follow(<prog>)	= {$}
Follow(<st-list>)	= {end}
Follow(<stat>)	= { ; }
Follow(<it-list>)	= { ; }
Follow(<item>)	= {id, int, ; }

Pokud bychom provedli další cyklus II. kroku algoritmu, zjistili bychom, že žádná jiná množina již změněna nebyla. Hodnoty jednotlivých množin jsou tedy výsledné.

4) Jako poslední můžeme spočítat množinu *Predict*(***r***) pro každé !**pravidlo!** ***r*****:** ***A*** → ***x***. (ne pro symboly a řetězce!) podle následující algoritmu:
   - Pro každé pravidlo tvaru ***r*****:** ***A*** → ***x*** urči množinu *Predict*(***r***) jako:
     Pokud *Empty*(***x***) = {ε} polož *Predict*(***r***) = *First*(***x***) ∪ *Follow*(***A***);
     jinak polož *Predict*(***r***) = *First*(***x***).

Poznámka k algoritmu:

- Všimněte si, že množina *First*(***x***) musí být obecně určena pro **!řetězec!** ***x*** (ne pouze symbol).

Výpočet množin *Predict*(*r*) pro výše uvedený příklad:

1: <prog> → begin <st-list> end
Empty(**begin <st-list> end**) ≠ {ε}, tedy *Predict*(1) = *First*(**begin <st-list> end**) = {**begin**}

2: <st-list> → <stat> ; <st-list>
Empty(**<stat> ; <st-list>**) ≠ {ε}, tedy *Predict*(2) = *First*(**<stat> ; <st-list>**) = {**id**, **write**, **read**, **;**}

3: <st-list> → ε
Empty(**ε**) = {ε}, tedy *Predict*(3) = *First*(**ε**) ∪ *Follow*(**<st-list>**) = ∅ ∪ {**end**} = {**end**}

4: <stat> → read id
Empty(**read id**) ≠ {ε}, tedy *Predict*(4) = *First*(**read id**) = {**read**}

5: <stat> → write <item>
Empty(**write <item>**) ≠ {ε}, tedy *Predict*(5) = *First*(**write <item>**) = {**write**}

6: <stat> → id := add <item> <it-list>
Empty(**id := add <item> <it-list>**) ≠ {ε}, tedy *Predict*(6) = *First*(**id := add <item> <it-list>**) = {**id**}

7: <stat> → ε
Empty(**ε**) = {ε}, tedy *Predict*(7) = *First*(**ε**) ∪ *Follow*(**<stat>**) = ∅ ∪ { **;** } = { **;** }

8: <it-list> → <item> <it-list>
Empty(**<item> <it-list>**) ≠ {ε}, tedy *Predict*(8) = *First*(**<item> <it-list>**) = {**id**, **int**}

9: <it-list> → ε
Empty(**ε**) = {ε}, tedy *Predict*(9) = *First*(**ε**) ∪ *Follow*(**<it-list>**) = ∅ ∪ { **;** } = { **;** }

10: <item> → int
Empty(**int**) ≠ {ε}, tedy *Predict*(10) = *First*(**int**) = {**int**}

11: <item> → id
Empty(**id**) ≠ {ε}, tedy *Predict*(11) = *First*(**id**) = {**id**}

Jednotlivé množiny *Predict* pro všechna pravidla gramatiky:

Predict(1)	= {**begin**}
Predict(2)	= {**id**, **write**, **read**, **;**}
Predict(3)	= {**end**}
Predict(4)	= {**read**}
Predict(5)	= {**write**}
Predict(6)	= {**id**}
Predict(7)	= { **;** }
Predict(8)	= {**id**, **int**}
Predict(9)	= { **;** }
Predict(10)	= {**int**}
Predict(11)	= {**id**}

5) Nyní již můžeme zkonstruovat výslednou *LL*-tabulku. Tabulka má záhlaví sloupců popsané ***terminálními symboly*** a speciálním symbolem **$**, záhlaví řádků ***neterminálními symboly***. Jednotlivá políčka tabulky α[*A*, ***a***], kde $A \in N$, $a \in T \cup \{\$\}$ potom vyplníme následujícím algoritmem:
   - α[*A*, ***a***] obsahuje pravidlo ***r***: $A \rightarrow x$, pokud $a \in Predict(r)$
   - α[*A*, ***a***] je prázdné pro všechny ostatní případy

Poznámka k algoritmu:

- **Pokud by nějaké políčko α[*A*, ***a***] mělo obsahovat více jak jedno pravidlo, pak daná gramatika NENÍ LL !!!**

Konstrukce LL-tabulky pro výše uvedený příklad:

- Konstrukce LL-tabulky pro sloupec označený terminálem **begin**:

	begin
<prog>	1
<st-list>	
<stat>	
<it-list>	
<item>	

Pravidla *r*: $A \rightarrow x$, jejichž množiny *Predict*(*r*) obsahující begin:
1: **<prog>** → **begin <st-list> end** // *Predict*(1) = {**begin**}

- Konstrukce LL-tabulky pro sloupec označený terminálem **id**:

	id
<prog>	
<st-list>	2
<stat>	6
<it-list>	8
<item>	11

Pravidla *r*: $A \rightarrow x$, jejichž množiny *Predict*(*r*) obsahující id:
2: **<st-list>** → **<stat> ; <st-list>** // *Predict*(2) = {**id**, **write**, **read**, **;** }
6: **<stat>** → **id := add <item> <it-list>** //*Predict*(6) = {**id**}
8: **<it-list>** → **<item> <it-list>** //*Predict*(8) = {**id**, **int**}
11: **<item>** → **id** //*Predict*(11) = {**id**}

- Konstrukce LL-tabulky pro sloupec označený terminálem **int**:

	int
<prog>	
<st-list>	
<stat>	
<it-list>	8
<item>	10

Pravidla *r*: $A \rightarrow x$, jejichž množiny *Predict*(*r*) obsahující int:
8: **<it-list>** → **<item> <it-list>** //*Predict*(8) = {**id**, **int**}
10: **<item>** → **int** //*Predict*(10) = {**int**}

…
Ostatní sloupce tabulky by se sestavily analogicky.

Výsledná *LL*-tabulka pro výše uvedený příklad:

	begin	end	read	write	id	int	;	:=	add	$
<prog>	1									
<st-list>		3	2	2	2		2			
<stat>			4	5	6		7			
<it-list>					8	8	9			
<item>					11	10				

b) Nejprve si uvedeme algoritmus:

Algoritmus pro prediktivní syntaktickou analýzu používající LL-tabulku:

- Vlož na zásobník symboly $***S***, kde ***S*** je startující neterminální symbol
- Hlavní cyklus
  - Nechť ***a*** je aktuální vstupní symbol, ***X*** je nejvrchnější symbol na zásobníku
    - ***X*** = $: Pokud ***a*** = $, **úspěch syntaktické analýzy**;
      jinak **chyba!**
    - ***X*** ∈ *T*: Pokud ***X*** = ***a***, přečti symbol ***a*** ze vstupu a odstraň symbol ***a*** ze zásobníku;
      jinak **chyba!**
    - ***X*** ∈ *N*: Pokud *LL*-tabulka na souřadnicích [***X***, ***a***] obsahuje pravidlo ***r***: ***X*** → ***x***,
      odstraň symbol ***X*** ze zásobníku a vlož na něj **reverzovaně** řetězec ***x***;
      jinak **chyba!**
- Proveď další smyčku cyklu

Syntaktická analýza řetězce begin write int ; end :

Zásobník	*Vstup*	**Pravidlo**	**Odpovídající derivace**
$ <prog>	begin write int ; end $	1	<prog> ⇒ begin <st-list> end
$ end <st-list> begin	begin write int ; end $		
$ end <st-list>	write int ; end $	5	⇒ begin <stat> ; <st-list> end
$ end <st-list> ; <stat>	write int ; end $	2	⇒ begin write <item> ; <st-list> end
$ end <st-list> ; <item> write	write int ; end $		
$ end <st-list> ; <item>	int ; end $	10	⇒ begin write int ; <st-list> end
$ end <st-list> ; int	int ; end $		
$ end <st-list> ;	; end $		
$ end <st-list>	end $	3	⇒ begin write int ; end
$ end	end $		
$	$		

ÚSPĚCH! Levý rozbor: 1 5 2 10 3

6. Syntaktická analýza zdola nahoru

Příklad 1.

a) Vytvořte precedenční tabulku pro gramatiku $G = (N, T, P, \boldsymbol{E})$, kde:

- $N = \{\boldsymbol{E}\}$,
- $T = \{+, -, *, /, \hat{}, \boldsymbol{i}, (,) \}$,
- $P = \{\ 1: \boldsymbol{E} \to \boldsymbol{E}+\boldsymbol{E},\ 2: \boldsymbol{E} \to \boldsymbol{E}-\boldsymbol{E},\ 3: \boldsymbol{E} \to \boldsymbol{E}*\boldsymbol{E},\ 4: \boldsymbol{E} \to \boldsymbol{E}/\boldsymbol{E},\ 5: \boldsymbol{E} \to \boldsymbol{E}\hat{}\boldsymbol{E},$
  $6: \boldsymbol{E} \to (\boldsymbol{E}),\ 7: \boldsymbol{E} \to \boldsymbol{i}\ \}$

b) pomocí precedenční tabulky proveďte syntaktickou analýzu zdola nahoru pro řetězec **(*i* * *i*) ^ *i*** a uveďte jeho pravý rozbor.

Význam jednotlivých operací, jejich asociativita a precedence:

zvyšující se precedence ↓

\+ … sčítání, − … odčítání (obě levě asociativní)
* … násobení, / … dělení (obě levě asociativní)
^ … mocnina (pravě asociativní)

Poznámka:

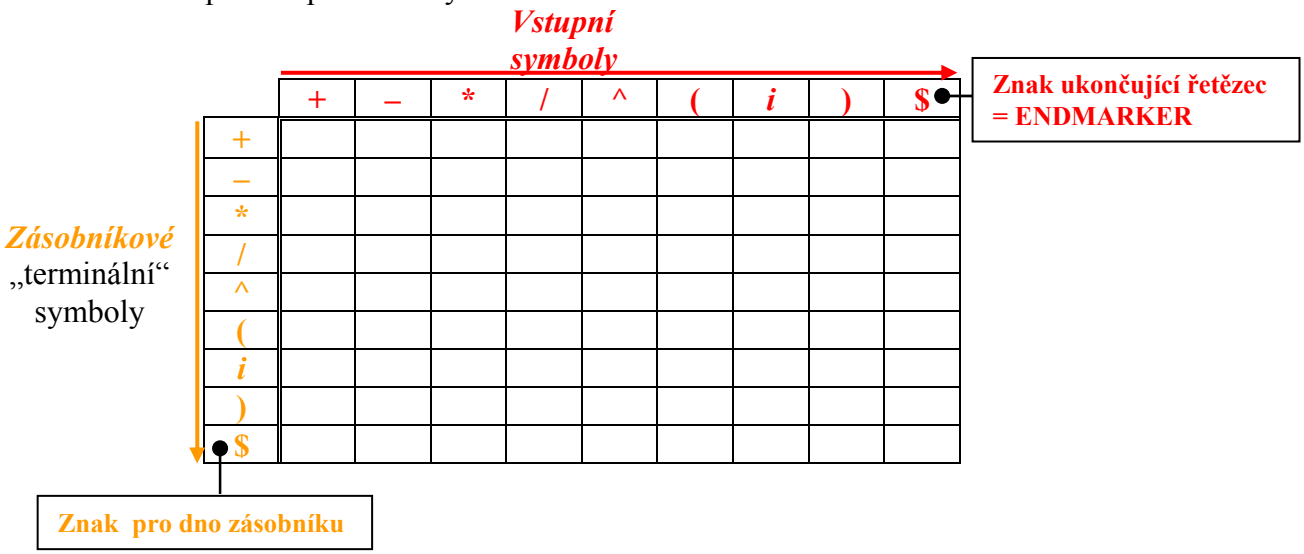
Pro levě asociativní operaci • obecně platí: $a \bullet b \bullet c = (a \bullet b) \bullet c$

Pro pravě asociativní operaci • obecně platí: $a \bullet b \bullet c = a \bullet (b \bullet c)$

Řešení:

Záhlaví precedenční tabulky vytvoříme tak, že sloupce i řádky označíme terminálními symboly gramatiky a speciálním symbolem $. Pozor! Označení řádků i sloupců je sice stejné, ale pokaždé má jiný význam. Označení sloupců bude reprezentovat vstupní symboly, speciálně $ reprezentuje znak, kterým bude každý řetězec ukončen (=*ENDMARKER*). Označení řádků bude reprezentovat některé zásobníkové symboly, speciálně $ reprezentuje symbol, který je na dně zásobníku (=*STARTUJÍCÍ ZÁSOBNÍKOVÝ SYMBOL*).

Precedenční tabulka pro náš příklad tedy bude ve tvaru:



Tabulku nyní postupně vyplníme symboly: =, <, >, „*prázdné políčko*“ následujícím způsobem:

I. Vyplnění částí týkajících se operací:

a) pokud operace op_x má **vyšší prioritu** než operace op_y, potom platí:

$$op_x > op_y \; ; \; op_y < op_x$$

b) pokud operace op_x má **stejnou prioritu** jako operace op_y a jsou-li navíc obě operace **levě asociativní**, potom platí:

$$op_x > op_y \; ; \; op_y > op_x$$

c) pokud operace op_x má **stejnou prioritu** jako operace op_y a jsou-li navíc obě operace **pravě asociativní**, potom platí:

$$op_x < op_y \; ; \; op_y < op_x$$

Operační část tabulky pro výše uvedený příklad tedy bude vypadat následovně:

	+	–	*	/	^
+	>	>	<	<	<
–	>	>	<	<	<
*	>	>	>	>	<
/	>	>	>	>	<
^	>	>	>	>	<

Mnemotechnická pomůcka pro vyplnění operační části tabulky:

- Pro různé priority jsou zápisy dostatečně mnemotechnické: + < *, * > +, * < ^, ^ > *, …
- Pro stejné priority a levou asociativitu má „*jako by větší prioritu*“ operace na prvním místě, protože bude provedena dříve: + > +, – > –, – > +, + > –, …
- Pro stejné priority a pravou asociativitu má „*jako by větší prioritu*“ operace na druhém místě, protože bude provedena dříve: ^ < ^.

II. Vyplnění částí týkajících se identifikátorů:

a) Pro každý terminální symbol *a*, který se může vyskytovat hned **před** identifikátorem *i*, platí:

$$a < i$$

b) Pro každý terminální symbol *a*, který se může vyskytovat hned **za** identifikátorem *i*, platí:

$$i > a$$

- V libovolném výrazu se může **před** identifikátorem nacházet: **libovolná operace**, **levá závorka** a **!!!POZOR!!!**, jakmile později uvidíme, budou se načtené symboly vkládat na zásobník, tedy speciálně v zásobníku se může vyskytovat hned před identifikátorem také startující symbol zásobníku $. Platí tedy: + < *i*, – < *i*, * < *i*, / < *i*, ^ < *i*, (< *i*, $ < *i*.

- V libovolném výrazu se může **za** identifikátorem nacházet: **libovolná operace**, **pravá závorka** a speciálně ve vstupním řetězci také ukončovač řetězce $. Platí tedy: *i* > +, *i* > –, *i* > *, *i* > /, *i* > ^, *i* >) , *i* > $.

Odpovídající část tabulky pro výše uvedený příklad tedy bude vypadat následovně:

	+	–	*	/	^	(	*i*	)	$
+	>	>	<	<	<		<		
–	>	>	<	<	<		<		
*	>	>	>	>	<		<		
/	>	>	>	>	<		<		
^	>	>	>	>	<		<		
(							<		
i	>	>	>	>	>			>	>
)									
$							<		

III. Vyplnění částí týkajících se závorek:

a) Platí:

(=)

b) Pro každý terminální symbol *a* různý od), $ platí:

(< *a*

c) Pro každý terminální symbol *a* různý od (, $ platí:

a >)

d) Pro každý terminální symbol *a*, který se může vyskytovat hned **před** levou závorkou (, platí:

a < (

e) Pro každý terminální symbol *a*, který se může vyskytovat hned **za** pravou závorkou), platí:

) > *a*

Poznámka: Řešení této části je analogické jako u části II. Také je tedy potřeba dát pozor na problém ukončovače řetězce a na startující symbol v zásobníku!

Odpovídající část tabulky pro výše uvedený příklad tedy bude vypadat následovně:

	+	–	*	/	^	(	*i*	)	$
+	>	>	<	<	<	<	<	>	
–	>	>	<	<	<	<	<	>	
*	>	>	>	>	<	<	<	>	
/	>	>	>	>	<	<	<	>	
^	>	>	>	>	<	<	<	>	
(	<	<	<	<	<	<	<	=	
i	>	>	>	>	>			>	>
)	>	>	>	>	>			>	>
$						<	<		

Poznámka: Všimněte si, že řádky se záhlavím *i*,) jsou stejné a sloupce se záhlavím (, *i* také.

IV. Vyplnění částí týkajících se symbolu $:

Tabulku dokončíme následujícím způsobem. Pro libovolný operátor *op* platí: $ < *op*; *op* > $

Výsledná tabulka tedy vypadá následovně:

	+	–	*	/	^	(	*i*	)	$
+	>	>	<	<	<	<	<	>	>
–	>	>	<	<	<	<	<	>	>
*	>	>	>	>	<	<	<	>	>
/	>	>	>	>	<	<	<	>	>
^	>	>	>	>	<	<	<	>	>
(	<	<	<	<	<	<	<	=	
i	>	>	>	>	>			>	>
)	>	>	>	>	>			>	>
$	<	<	<	<	<	<	<		

Algoritmus pro syntaktickou analýzu používající precedenční tabulku:

- Vlož na zásobník symbol $
- Hlavní cyklus:
  - Nechť ***a*** je aktuální vstupní symbol, ***b*** je nejvrchnější **terminální** symbol na zásobníku. Podle obsahu políčka precedenční tabulky na souřadnicích [***b***, ***a***] rozhodni:
    - = : Přečti symbol ***a*** ze vstupu a dej jej na vrchol zásobníku.
    - < : Najdi na zásobníku nejvrchnější **terminální** symbol–***b***. Hned **za** tento symbol vlož do zásobníku symbol < (Pozor, nemusí být na vrcholu zásobníku!). Přečti symbol ***a*** ze vstupu a dej jej na vrchol zásobníku.
    - > : Najdi na zásobníku nejvrchnější symbol <. Mezi tímto symbolem a vrcholem zásobníku najdi **pravou stranu** jistého pravidla *r*. Odstraň tuto část ze zásobníku včetně symbolu <. Vlož na zásobník levou stranu pravidla *r* popřípadě zapiš na výstup, že byla provedena redukce podle pravidla *r*.
    - *prázdné políčko*: **syntaktická chyba ve vstupním řetězci**
- Pokud ***a*** = $ a ***b*** = $ syntakt. analýza proběhla v pořádku, jinak proveď další smyčku cyklu.

Syntaktická analýza řetězce (*i* * *i*) ^ *i*:

Zásobník	**Operátor**	***Vstup***	**Redukce podle pravidla**
$	<	(*i* * *i*) ^ *i* $	
$ < (	<	*i* * *i*) ^ *i* $	
$ < (< *i*	>	* *i*) ^ *i* $	7: *E* → *i*
$ < (*E*	<	* *i*) ^ *i* $	
$ < (< *E* *	<	*i*) ^ *i* $	
$ < (< *E* * < *i*	>	) ^ *i* $	7: *E* → *i*
$ < (< *E* * *E*	>	) ^ *i* $	3: *E* → *E***E*
$ < (*E*	=	) ^ *i* $	
$ < (*E*)	>	^ *i* $	6: *E* → (*E*)
$ *E*	<	^ *i* $	
$ < *E* ^	<	*i* $	
$ < *E* ^ < *i*	>	$	7: *E* → *i*
$ < *E* ^ *E*	>	$	5: *E* → *E*^*E*
$ *E*	>	$	**ÚSPĚCH! Pravý rozbor:** 773675

Příklad 2.
Uvažujte následující LR-tabulku pro gramatiku $G = (N, T, P, E)$, kde:

- $N = \{E, T, F\}$,
- $T = \{+, -, *, /, i, (,)\}$,
- $P = \{ 1: E \to E+T,\ 2: E \to E-T,\ 3: E \to T,\ 4: T \to T*F,\ 5: T \to T/F,\ 6: T \to F,$
  $7: F \to (E),\ 8: F \to i \}$

LR-tabulka:

	Akční část								Přechodová část		
	i	+	–	*	/	(	)	$	E	*T*	*F*
0	s5					s4			1	2	3
1		s6	s7					☺			
2		r3	r3	s8	s9		r3	r3			
3		r6	r6	r6	r6		r6	r6			
4	s5					s4			10	2	3
5		r8	r8	r8	r8		r8	r8			
6	s5					s4				11	3
7	s5					s4				12	3
8	s5					s4					13
9	s5					s4					14
10		s6	s7				s15				
11		r1	r1	s8	s9		r1	r1			
12		r2	r2	s8	s9		r2	r2			
13		r4	r4	r4	r4		r4	r4			
14		r5	r5	r5	r5		r5	r5			
15		r7	r7	r7	r7		r7	r7			

Pomocí této LR-tabulky proveďte syntaktickou analýzu zdola nahoru pro řetězec **(*i* + *i*) / *i*** a uveďte jeho pravý rozbor.

Algoritmus pro syntaktickou analýzu používající LR tabulku:

- Vlož na zásobník dvojici symbolů <**$**, q_0>, nastav aktuální stav *s* na q_0. (q_0 je první stav)
- Hlavní cyklus:
  - Nechť ***a*** je aktuální vstupní symbol, *s* je aktuální stav. Podle obsahu políčka LR-tabulky **akční části** na souřadnicích [*s*, ***a***] rozhodni:
    - **s***q* : Přečti symbol ***a*** ze vstupu a dej dvojici <***a***, *q*> na zásobník. Aktuální stav *s* nastav na *q*.
    - **r***p* : Nechť pravidlo s návěštím *p* je tvaru $A \to X_1 X_2 \ldots X_d$ Potom:
      1) Zkontroluj, zda je prvních *d* symbolů na zásobníku ve tvaru: $<X_1, ?> <X_2, ?> \ldots <X_d, ?>$, jinak **chyba!**
      2) Nechť ještě **hned** před prvkem $<X_1, ?>$ se nachází prvek tvaru <?, *q*>, kde *q* je nějaký stav. Nastav aktuální stav *s* na hodnotu, kterou obsahuje LR-tabulka **přechodové části** na souřadnicích [*q*, *A*] (*A* je levá strana pravidla *p*), odstraň symboly $<X_1, ?> <X_2, ?> \ldots <X_d, ?>$ ze zásobníku a vlož na něj dvojici <*A*, *s*>.
    - ☺: **Konec: úspěch syntaktické analýzy!**
    - *Prázdné políčko*: **chyba!**
- Proveď další smyčku cyklu

Syntaktická analýza řetězce (*i* + *i*) / *i*:

Zásobník	**Stav**	***Vstup***	***Použití tabulky***	**Redukce podle pravidla**
<$, 0>	0	(*i* + *i*) / *i* $	α[0, (] = s4	
<$, 0> <(, 4>	4	*i* + *i*) / *i* $	α[4, *i*] = s5	
<$, 0> <(, 4> <*i*, 5>	5	+ *i*) / *i* $	α[5, +] = r8 β[4, *F*] = 3	8: *F* → *i*
<$, 0> <(, 4> <*F*, 3>	3	+ *i*) / *i* $	α[3, +] = r6 β[4, *T*] = 2	6: *T* → *F*
<$, 0> <(, 4> <*T*, 2>	2	+ *i*) / *i* $	α[2, +] = r3 β[4, *E*] = 10	3: *E* → *T*
<$, 0> <(, 4> <*E*, 10>	10	+ *i*) / *i* $	α[10, +] = s6	
<$, 0> <(, 4> <*E*, 10> <+, 6>	6	*i*) / *i* $	α[6, *i*] = s5	
<$, 0> <(, 4> <*E*, 10> <+, 6> <*i*, 5>	5	) / *i* $	α[5,)] = r8 β[6, *F*] = 3	8: *F* → *i*
<$, 0> <(, 4> <*E*, 10> <+, 6> <*F*, 3>	3	) / *i* $	α[3,)] = r6 β[6, *T*] = 11	6: *T* → *F*
<$, 0> <(, 4> <*E*, 10> <+, 6> <*T*, 11>	11	) / *i* $	α[11,)] = r1 β[4, *E*] = 10	1: *E* → *E*+*T*
<$, 0> <(, 4> <*E*, 10>	10	) / *i* $	α[10,)] = s15	
<$, 0> <(, 4> <*E*, 10> <), 15>	15	/ *i* $	α[15, /] = r7 β[0, *F*] = 3	7: *F* → (*E*)
<$, 0> <*F*, 3>	3	/ *i* $	α[3, /] = r6 β[0, *T*] = 2	6: *T* → *F*
<$, 0> <*T*, 2>	2	/ *i* $	α[2, /] = s9	
<$, 0> <*T*, 2> </, 9>	9	*i* $	α[9, *i*] = s5	
<$, 0> <*T*, 2> </, 9> <*i*, 5>	5	$	α[5, $] = r8 β[9, *F*] = 14	8: *F* → *i*
<$, 0> <*T*, 2> </, 9> <*F*, 14>	14	$	α[14, $] = r5 β[0, *T*] = 2	5: *T* → *T*/*F*
<$, 0> <*T*, 2>	2	$	α[2, $] = r3 β[0, *E*] = 1	3: *E* → *T*
<$, 0> <*E*, 1>	1	$	α[1, $] = ☺	

ÚSPĚCH! Pravý rozbor: 86386176853