Shrnutí minulé přednášky (+dva řádky a sloupce navíc)

- Podmnožinová konstrukce DFA z ϵNFA
- Regulární výrazy
- Kleeneho věta
  - Jazyk je přijímaný konečným automatem právě když lze napsat jako regulární výraz,
  - tj. z $\emptyset$ a $\{\mathbf{a}\}$ pro $a \in \Sigma$
  - a konečného počtu aplikací iterace, zřetězení a sjednocení.
- Uzávěrové vlastnosti
  - dnes jen 'regulární' sloupec.

jazyk	regulární (RL)	bezkontextové	deterministické CFL
sjednocení	ANO	ANO	NE
průnik	ANO	NE	NE
$\cap$ s RL	ANO	ANO	ANO
doplněk	ANO	NE	ANO
homomorfizmus	ANO	ANO	NE
inverzní hom.	ANO	ANO	ANO

Shrnutí minulé přednášky (+dva řádky a sloupce navíc)

- Podmnožinová konstrukce DFA z ϵNFA
- Regulární výrazy
- Kleeneho věta
  - Jazyk je přijímaný konečným automatem právě když lze napsat jako regulární výraz,
  - tj. z $\emptyset$ a $\{\mathbf{a}\}$ pro $a \in \Sigma$
  - a konečného počtu aplikací iterace, zřetězení a sjednocení.
- Uzávěrové vlastnosti
  - dnes jen 'regulární' sloupec.

jazyk	regulární (RL)	bezkontextové	deterministické CFL
sjednocení	ANO	ANO	NE
průnik	ANO	NE	NE
$\cap$ s RL	ANO	ANO	ANO
doplněk	ANO	NE	ANO
homomorfizmus	ANO	ANO	NE
inverzní hom.	ANO	ANO	ANO

Substituce jazyků

Definition 4.1 (Substituce jazyků)

Mějme konečnou abecedu Σ. Pro každé $x \in \Sigma$ budiž $\sigma(x)$ jazyk v nějaké abecedě Y_x. Dále položme

$$\sigma(\epsilon) = \{\epsilon\}$$
$$\sigma(u.v) = \sigma(u).\sigma(v)$$

- Zobrazení $\sigma : \Sigma^* \to P(Y^*)$, kde $Y = \bigcup_{x \in \Sigma} Y_x$ se nazývá **substituce**.
- Pro jazyk L definujeme: $\sigma(L) = \bigcup_{w \in L} \sigma(w)$, podobně sjednocení.
- **nevypouštějící substituce** je substituce, kde žádné $\sigma(x)$ neobstahuje ϵ.

Example 4.1 (substituce)

1) $\Sigma = \{k, p, m, c, t\}$, $L = (kmp)(ckmp)^* t$,
   k slovník křestních jmen, p slovník příjmení, m mezera, c čárka, t tečka.
2) Pokud $\sigma(0) = \{a^i b^j, i, j \geq 0\}, \sigma(1) = \{cd\}$
   tak $\sigma(010) = \{a^i b^j cda^k b^l, i, j, k, l \geq 0\}$.

Substituce jazyků

Definition 4.1 (Substituce jazyků)

Mějme konečnou abecedu Σ. Pro každé $x \in \Sigma$ budiž $\sigma(x)$ jazyk v nějaké abecedě Y_x. Dále položme

$$\sigma(\epsilon) = \{\epsilon\}$$
$$\sigma(u.v) = \sigma(u).\sigma(v)$$

- Zobrazení $\sigma : \Sigma^* \to P(Y^*)$, kde $Y = \bigcup_{x \in \Sigma} Y_x$ se nazývá **substituce**.
- Pro jazyk L definujeme: $\sigma(L) = \bigcup_{w \in L} \sigma(w)$, podobně sjednocení.
- **nevypouštějící substituce** je substituce, kde žádné $\sigma(x)$ neobstahuje ϵ.

Example 4.1 (substituce)

1) $\Sigma = \{k, p, m, c, t\}$, $L = (kmp)(ckmp)^* t$,
   k slovník křestních jmen, p slovník příjmení, m mezera, c čárka, t tečka.
2) Pokud $\sigma(0) = \{a^i b^j, i, j \geq 0\}, \sigma(1) = \{cd\}$
   tak $\sigma(010) = \{a^i b^j cda^k b^l, i, j, k, l \geq 0\}$.

Substituce jazyků

Definition 4.1 (Substituce jazyků)

Mějme konečnou abecedu Σ. Pro každé $x \in \Sigma$ budiž $\sigma(x)$ jazyk v nějaké abecedě Y_x. Dále položme

$$\sigma(\epsilon) = \{\epsilon\}$$
$$\sigma(u.v) = \sigma(u).\sigma(v)$$

- Zobrazení $\sigma : \Sigma^* \to P(Y^*)$, kde $Y = \bigcup_{x \in \Sigma} Y_x$ se nazývá **substituce**.
- Pro jazyk L definujeme: $\sigma(L) = \bigcup_{w \in L} \sigma(w)$, podobně sjednocení.
- **nevypouštějící substituce** je substituce, kde žádné $\sigma(x)$ neobstahuje ϵ.

Example 4.1 (substituce)

1) $\Sigma = \{k, p, m, c, t\}$, $L = (kmp)(ckmp)^* t$,
   k slovník křestních jmen, p slovník příjmení, m mezera, c čárka, t tečka.
2) Pokud $\sigma(0) = \{a^i b^j, i, j \geq 0\}, \sigma(1) = \{cd\}$
   tak $\sigma(010) = \{a^i b^j cda^k b^l, i, j, k, l \geq 0\}$.

Homomorfizmus a inverzní homomorfizmus jazyků

Definition 4.2 (homomorfizmus (jazyků), inverzní homomorfizmus)

Homomorfizmus h je speciální případ substituce, kde obraz je vždy jen jednoslovný jazyk (vynecháváme u něj závorky), tj. $(\forall x \in \Sigma)\ h(x) = w_x$.
Pokud $\forall x : w_x \neq \epsilon$, jde o **nevypouštějící homomorfizmus**.
Inverzní homomorfizmus $h^{-1}(L) = \{w | h(w) \in L\}$.

Example 4.2 (homomorfizmus)

- Znaky nahradíme TEXzápisem, $h(\mu) = \backslash mu$ a podobně.
- Homomorfizmus h definujeme: $h(0) = ab$, a $h(1) = \epsilon$. Pak $h(0011) = abab$. Pro $L = \mathbf{10^*1}$ je $h(L) = (ab)^*$.

Theorem 4.1 (uzavřenost na homomorfizmus)

Je–li jazyk L i $\forall x \in \Sigma$ jazyk $\sigma(x), h(x)$ regulární, pak je regulární i $\sigma(L), h(L)$.

Homomorfizmus a inverzní homomorfizmus jazyků

Definition 4.2 (homomorfizmus (jazyků), inverzní homomorfizmus)

Homomorfizmus h je speciální případ substituce, kde obraz je vždy jen jednoslovný jazyk (vynecháváme u něj závorky), tj. $(\forall x \in \Sigma)\ h(x) = w_x$.
Pokud $\forall x : w_x \neq \epsilon$, jde o **nevypouštějící homomorfizmus**.
Inverzní homomorfizmus $h^{-1}(L) = \{w | h(w) \in L\}$.

Example 4.2 (homomorfizmus)

- Znaky nahradíme TEXzápisem, $h(\mu) = \backslash mu$ a podobně.
- Homomorfizmus h definujeme: $h(0) = ab$, a $h(1) = \epsilon$. Pak $h(0011) = abab$. Pro $L = \mathbf{10^*1}$ je $h(L) = (ab)^*$.

Theorem 4.1 (uzavřenost na homomorfizmus)

Je–li jazyk L i $\forall x \in \Sigma$ jazyk $\sigma(x), h(x)$ regulární, pak je regulární i $\sigma(L), h(L)$.

Uzavřenost na substituci, homomorfizmus.

Strukturální indukcí 'probubláváním' algebraickým popisem jazyka základních, sjednocení, zřetězení a iterace. Pro sjednocení a zřetězení z definice substituce a uzavřenosti regulárních jazyků na sjednocení a zřetězení.

$$\begin{aligned} \underline{\sigma}(\alpha+\beta) &= \sigma(L(\alpha)) \cup \sigma(L(\beta)) \\ \underline{\sigma}(\alpha\beta) &= \{w | \exists u \in L(\alpha) \exists v \in L(\beta) : \sigma(u)\sigma(v) = w\} \end{aligned}$$

Pro iteraci rozložíme na nekonečné sjednocení, pro každý konkrétní počet iterací σ aplikované na konečné zřetězení.

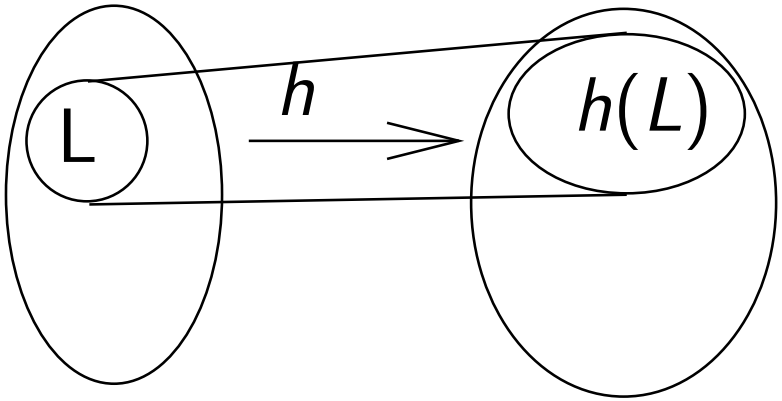
$$\begin{aligned} \sigma(L(\alpha)^*) &= \sigma(L(\alpha)^0) \cup \sigma(L(\alpha)^1) \cup \ldots \cup \sigma(L(\alpha)^n) \cup \ldots \\ &= \underline{\sigma}(\alpha)^0 \cup \underline{\sigma}(\alpha)^1 \cup \ldots \cup \underline{\sigma}(\alpha)^n \cup \ldots \\ &= L(\underline{\sigma}(\alpha)^*). \end{aligned}$$

□

Inverzní homomorfizmus

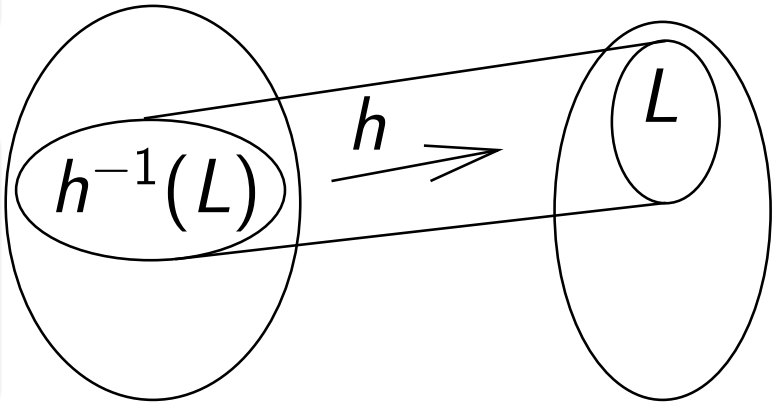
Definition ((4.2) Inverzní homomorfizmus)

Nechť h je homomorfizmus abecedy T do slov nad abecedou Σ. Pak $h^{-1}(L)$ *'h inverze L'* je množina řetězců
$h^{-1}(L) = \{w | w \in T^*;\ h(w) \in L\}$.



Example 4.3

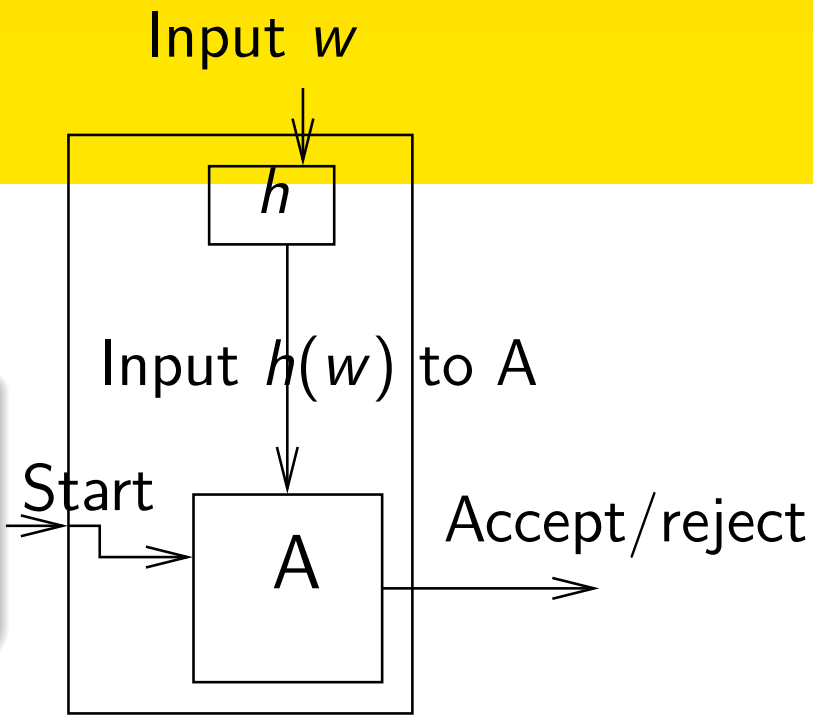
Nechť $L = (\{00\} \cup \{1\})^*$, $h(a) = 01$ a $h(b) = 10$.
Pak $h^{-1}(L) = (\{ba\})^*$.



Důkaz: $h((\{ba\})^*) \in L$ snadno.
Ostatní w generují izolované 0 (rozbor případů).

Homomorfizmus aplikovaný dopředně a zpětně.

Inverzní homomorfizmus DFA



Theorem 4.2

Je–li h homomorfizmus abecedy T do abecedy Σ a L je regulární jazyk abecedy Σ, pak $h^{-1}(L)$ je také regulární jazyk.

Proof:

- pro L máme DFA $A = (Q, \Sigma, \delta, q_0, F)$
- $h : T \to \Sigma^*$
- definujeme ϵ-NFA $B = (Q', T, \delta', [q_0, \epsilon], F \times \{\epsilon\})$ kde

$$Q' = \{[q, u] \,|\, q \in Q, u \in \Sigma^*, \exists(a \in T)\exists(v \in \Sigma^*)h(a) = vu\} \quad u \text{ je buffer}$$
$$\delta'([q, \epsilon], a) = [q, h(a)] \quad \text{naplňuje buffer}$$
$$\delta'([q, bv], \epsilon) = [p, v] \text{ kde } \delta(q, b) = p \quad \text{čte buffer.}$$

□

Příklad: Navštiv všechny stavy

Example 4.4

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je DFA. Definujme jazyk $L = \{w \in \Sigma^*;\ \delta^*(q_0, w) \in F$ a pro každý stav $q \in Q$ existuje prefix x_q slova w tak, že $\delta^*(q_0, x_q) = q\}$.
Tento jazyk L je regulární.

Příklad: Navštiv všechny stavy

Example 4.4

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je DFA. Definujme jazyk $L = \{w \in \Sigma^*;\ \delta^*(q_0, w) \in F$ a pro každý stav $q \in Q$ existuje prefix x_q slova w tak, že $\delta^*(q_0, x_q) = q\}$. Tento jazyk L je regulání.

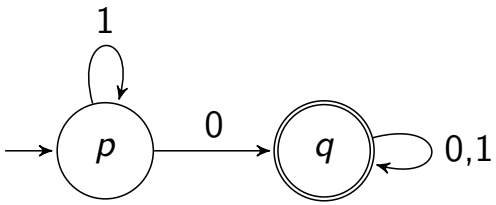
M Označme $M = L(A)$.

T Definujeme novou abecedu T trojic $\{[paq]; p, q \in Q, a \in \Sigma, \delta(p, a) = q\}$.

h Definujeme homomorfizmus $(\forall\ p, q, a)\ h([paq]) = a$.

L_1 Jazyk $L_1 = h^{-1}(M)$ je regulární, protože M je regulární (DFA inverzní homomorfizmus).

- $h^{-1}(101)$ obsahuje $2^3 = 8$ řetězců, např. $[p1p][q0q][p1p] \in \{[p1p], [q1q]\}\{[p0q], [q0q]\}\{[p1p], [q1q]\}$.
- Dále zkonstruujeme L z L_1 (další slide).



Příklad: Navštiv všechny stavy

Example 4.4

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je DFA. Definujme jazyk $L = \{w \in \Sigma^*;\ \delta^*(q_0, w) \in F$ a pro každý stav $q \in Q$ existuje prefix x_q slova w tak, že $\delta^*(q_0, x_q) = q\}$.
Tento jazyk L je regulání.

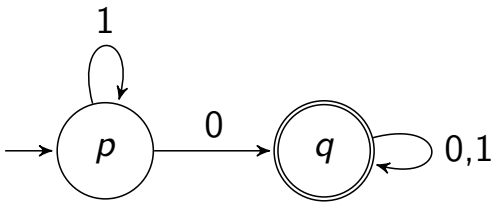
M Označme $M = L(A)$.

T Definujeme novou abecedu T trojic $\{[paq]; p, q \in Q, a \in \Sigma, \delta(p, a) = q\}$.

h Definujeme homomorfizmus $(\forall\ p, q, a)\ h([paq]) = a$.

L_1 Jazyk $L_1 = h^{-1}(M)$ je regulární, protože M je regulární (DFA inverzní homomorfizmus).

- $h^{-1}(101)$ obsahuje $2^3 = 8$ řetězců, např. $[p1p][q0q][p1p] \in \{[p1p], [q1q]\}\{[p0q], [q0q]\}\{[p1p], [q1q]\}$.
- Dále zkonstruujeme L z L_1 (další slide).



Příklad: Navštiv všechny stavy

Example 4.4

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je DFA. Definujme jazyk $L = \{w \in \Sigma^*;\ \delta^*(q_0, w) \in F$ a pro každý stav $q \in Q$ existuje prefix x_q slova w tak, že $\delta^*(q_0, x_q) = q\}$.
Tento jazyk L je regulání.

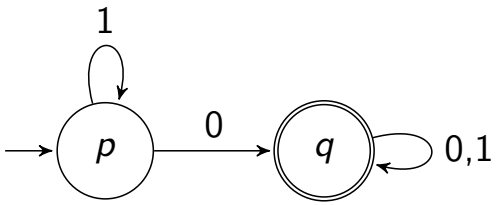
M Označme $M = L(A)$.

T Definujeme novou abecedu T trojic $\{[paq]; p, q \in Q, a \in \Sigma, \delta(p, a) = q\}$.

h Definujeme homomorfizmus $(\forall\ p, q, a)\ h([paq]) = a$.

L_1 Jazyk $L_1 = h^{-1}(M)$ je regulární, protože M je regulární (DFA inverzní homomorfizmus).

- $h^{-1}(101)$ obsahuje $2^3 = 8$ řetězců, např. $[p1p][q0q][p1p] \in \{[p1p], [q1q]\}\{[p0q], [q0q]\}\{[p1p], [q1q]\}$.
- Dále zkonstruujeme L z L_1 (další slide).



Příklad: Navštiv všechny stavy

Example 4.4

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je DFA. Definujme jazyk $L = \{w \in \Sigma^*;\ \delta^*(q_0, w) \in F$ a pro každý stav $q \in Q$ existuje prefix x_q slova w tak, že $\delta^*(q_0, x_q) = q\}$.
Tento jazyk L je regulání.

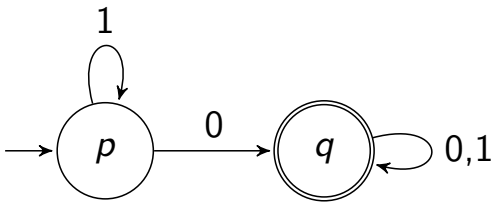
M Označme $M = L(A)$.

T Definujeme novou abecedu T trojic $\{[paq]; p, q \in Q, a \in \Sigma, \delta(p, a) = q\}$.

h Definujeme homomorfizmus $(\forall\ p, q, a)\ h([paq]) = a$.

L_1 Jazyk $L_1 = h^{-1}(M)$ je regulární, protože M je regulární (DFA inverzní homomorfizmus).

- $h^{-1}(101)$ obsahuje $2^3 = 8$ řetězců, např. $[p1p][q0q][p1p] \in \{[p1p], [q1q]\}\{[p0q], [q0q]\}\{[p1p], [q1q]\}$.
- Dále zkonstruujeme L z L_1 (další slide).



Příklad: Navštiv všechny stavy

Example 4.4

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je DFA. Definujme jazyk $L = \{w \in \Sigma^*;\ \delta^*(q_0, w) \in F$ a pro každý stav $q \in Q$ existuje prefix x_q slova w tak, že $\delta^*(q_0, x_q) = q\}$.
Tento jazyk L je regulání.

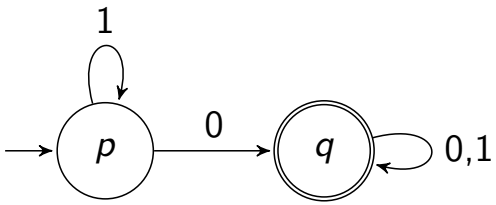
M Označme $M = L(A)$.

T Definujeme novou abecedu T trojic $\{[paq]; p, q \in Q, a \in \Sigma, \delta(p, a) = q\}$.

h Definujeme homomorfizmus $(\forall\ p, q, a)\ h([paq]) = a$.

L_1 Jazyk $L_1 = h^{-1}(M)$ je regulární, protože M je regulární (DFA inverzní homomorfizmus).

- $h^{-1}(101)$ obsahuje $2^3 = 8$ řetězců, např. $[p1p][q0q][p1p] \in \{[p1p], [q1q]\}\{[p0q], [q0q]\}\{[p1p], [q1q]\}$.
- Dále zkonstruujeme L z L_1 (další slide).



Příklad: Navštiv všechny stavy

Example 4.4

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je DFA. Definujme jazyk $L = \{w \in \Sigma^*;\ \delta^*(q_0, w) \in F$ a pro každý stav $q \in Q$ existuje prefix x_q slova w tak, že $\delta^*(q_0, x_q) = q\}$.
Tento jazyk L je regulání.

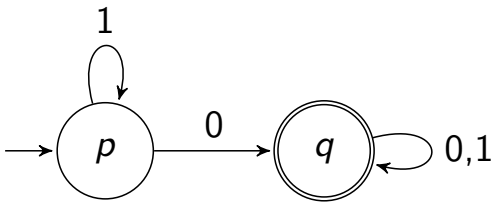
M Označme $M = L(A)$.

T Definujeme novou abecedu T trojic $\{[paq]; p, q \in Q, a \in \Sigma, \delta(p, a) = q\}$.

h Definujeme homomorfizmus $(\forall\ p, q, a)\ h([paq]) = a$.

L_1 Jazyk $L_1 = h^{-1}(M)$ je regulární, protože M je regulární (DFA inverzní homomorfizmus).

- $h^{-1}(101)$ obsahuje $2^3 = 8$ řetězců, např. $[p1p][q0q][p1p] \in \{[p1p], [q1q]\}\{[p0q], [q0q]\}\{[p1p], [q1q]\}$.
- Dále zkonstruujeme L z L_1 (další slide).



L_2 Vynutíme začátek q_0. Definujeme
$E_1 = \bigcup_{a\in\Sigma, q\in Q}\{[q_0 a q]\} =$
$E_1 = \{[q_0 a_1 q_0], [q_0 a_2 q_1], \ldots, [q_0 a_m q_n]\}$.
Pak $L_2 = L_1 \cap L(E_1.T^*)$.

L_3 Vynutíme stejné sousedící stavy.
Definujeme ne–odpovídající dvojice
$E_2 = \bigcup_{q\neq r, p,q,r,s\in Q, a,b\in\Sigma}\{[paq][rbs]\}$.
Definujeme $L_3 = L_2 - L(T^*.E_2.T^*)$,

- Končí v přijímajícím stavu, protože jsme začali z jazyku M přijímaném DFA A.

L_4 Všechny stavy. $\forall\ q \in Q$ definujeme E_q jako regulární výraz sjednocení všech symbolů T takových, že q není ani na první, ani na poslední pozici. Odečteme $L(E_q^*)$ od L_3. $L_4 = L_3 - \bigcup_{q\in Q}\{E_q^*\}$.

L Odstraníme stavy, necháme symboly.
$L = h(L_4)$. Tedy L je regulární.

Přehled:

M $= L(A)$
Inverzní homom.

L_1 $h^{-1}(M) \subseteq \{[qap]\}^*$
průnik RJ

L_2 $+\ q_0$
rozdíl RJ

L_3 + sousední stavy rovny
rozdíl RJ

L_4 + všechny stavy
homomorfismus

L $h([qap]) = a$

L_2 Vynutíme začátek q_0. Definujeme
$E_1 = \bigcup_{a\in\Sigma, q\in Q}\{[q_0 a q]\} =$
$E_1 = \{[q_0 a_1 q_0], [q_0 a_2 q_1], \ldots, [q_0 a_m q_n]\}$.
Pak $L_2 = L_1 \cap L(E_1.T^*)$.

L_3 Vynutíme stejné sousedící stavy.
Definujeme ne–odpovídající dvojice
$E_2 = \bigcup_{q\neq r, p,q,r,s\in Q, a,b\in\Sigma}\{[paq][rbs]\}$.
Definujeme $L_3 = L_2 - L(T^*.E_2.T^*)$,

- Končí v přijímajícím stavu, protože jsme začali z jazyku M přijímaném DFA A.

L_4 Všechny stavy. $\forall\ q \in Q$ definujeme E_q jako regulární výraz sjednocení všech symbolů T takových, že q není ani na první, ani na poslední pozici. Odečteme $L(E_q^*)$ od L_3. $L_4 = L_3 - \bigcup_{q\in Q}\{E_q^*\}$.

L Odstraníme stavy, necháme symboly.
$L = h(L_4)$. Tedy L je regulární.

Přehled:

M $= L(A)$
Inverzní homom.

L_1 $h^{-1}(M) \subseteq \{[qap]\}^*$
průnik RJ

L_2 $+\ q_0$
rozdíl RJ

L_3 + sousední stavy rovny
rozdíl RJ

L_4 + všechny stavy
homomorfismus

L $h([qap]) = a$

L_2 Vynutíme začátek q_0. Definujeme
$E_1 = \bigcup_{a\in\Sigma, q\in Q}\{[q_0 a q]\} =$
$E_1 = \{[q_0 a_1 q_0], [q_0 a_2 q_1], \ldots, [q_0 a_m q_n]\}$.
Pak $L_2 = L_1 \cap L(E_1.T^*)$.

L_3 Vynutíme stejné sousedící stavy.
Definujeme ne–odpovídající dvojice
$E_2 = \bigcup_{q\neq r, p,q,r,s\in Q, a,b\in\Sigma}\{[paq][rbs]\}$.
Definujeme $L_3 = L_2 - L(T^*.E_2.T^*)$,

- Končí v přijímajícím stavu, protože jsme začali z jazyku M přijímaném DFA A.

L_4 Všechny stavy. $\forall\ q \in Q$ definujeme E_q jako regulární výraz sjednocení všech symbolů T takových, že q není ani na první, ani na poslední pozici. Odečteme $L(E_q^*)$ od L_3. $L_4 = L_3 - \bigcup_{q\in Q}\{E_q^*\}$.

L Odstraníme stavy, necháme symboly.
$L = h(L_4)$. Tedy L je regulární.

Přehled:

M $= L(A)$
Inverzní homom.

L_1 $h^{-1}(M) \subseteq \{[qap]\}^*$
průnik RJ

L_2 $+\ q_0$
rozdíl RJ

L_3 + sousední stavy rovny
rozdíl RJ

L_4 + všechny stavy
homomorfismus

L $h([qap]) = a$

L_2 Vynutíme začátek q_0. Definujeme $E_1 = \bigcup_{a\in\Sigma, q\in Q}\{[q_0 a q]\} =$ $E_1 = \{[q_0 a_1 q_0], [q_0 a_2 q_1], \ldots, [q_0 a_m q_n]\}$. Pak $L_2 = L_1 \cap L(E_1.T^*)$.

L_3 Vynutíme stejné sousedící stavy. Definujeme ne–odpovídající dvojice $E_2 = \bigcup_{q\neq r, p,q,r,s\in Q, a,b\in\Sigma}\{[paq][rbs]\}$. Definujeme $L_3 = L_2 - L(T^*.E_2.T^*)$,

- Končí v přijímajícím stavu, protože jsme začali z jazyku M přijímaném DFA A.

L_4 Všechny stavy. $\forall\ q \in Q$ definujeme E_q jako regulární výraz sjednocení všech symbolů T takových, že q není ani na první, ani na poslední pozici. Odečteme $L(E_q^*)$ od L_3. $L_4 = L_3 - \bigcup_{q\in Q}\{E_q^*\}$.

L Odstraníme stavy, necháme symboly. $L = h(L_4)$. Tedy L je regulární.

Přehled:

M $= L(A)$
Inverzní homom.

L_1 $h^{-1}(M) \subseteq \{[qap]\}^*$
průnik RJ

L_2 $+\ q_0$
rozdíl RJ

L_3 + sousední stavy rovny
rozdíl RJ

L_4 + všechny stavy
homomorfismus

L $h([qap]) = a$

L_2 Vynutíme začátek q_0. Definujeme $E_1 = \bigcup_{a\in\Sigma, q\in Q}\{[q_0 a q]\} =$ $E_1 = \{[q_0 a_1 q_0], [q_0 a_2 q_1], \ldots, [q_0 a_m q_n]\}$. Pak $L_2 = L_1 \cap L(E_1.T^*)$.

L_3 Vynutíme stejné sousedící stavy. Definujeme ne–odpovídající dvojice $E_2 = \bigcup_{q\neq r, p,q,r,s\in Q, a,b\in\Sigma}\{[paq][rbs]\}$. Definujeme $L_3 = L_2 - L(T^*.E_2.T^*)$,

- Končí v přijímajícím stavu, protože jsme začali z jazyku M přijímaném DFA A.

L_4 Všechny stavy. $\forall\ q \in Q$ definujeme E_q jako regulární výraz sjednocení všech symbolů T takových, že q není ani na první, ani na poslední pozici. Odečteme $L(E_q^*)$ od L_3. $L_4 = L_3 - \bigcup_{q\in Q}\{E_q^*\}$.

L Odstraníme stavy, necháme symboly. $L = h(L_4)$. Tedy L je regulární.

Přehled:

M $= L(A)$
Inverzní homom.

L_1 $h^{-1}(M) \subseteq \{[qap]\}^*$
průnik RJ

L_2 $+\ q_0$
rozdíl RJ

L_3 + sousední stavy rovny
rozdíl RJ

L_4 + všechny stavy
homomorfismus

L $h([qap]) = a$

Rozhodovací problémy pro regulární jazyky!

Lemma (Test ne–prázdnosti regulárního jazyka)

Lze algoritmicky rozhodnout, zda jazyk přijímaný DFA, NFA, ϵ-NFA je prázdný.

Jazyk je prázdný právě když žádný z koncových stavů není dosažitelný.
Dosažitelnost lze testovat $O(|Q|^2)$.

Lemma (Test náležení do regulárního jazyka)

Pro daný řetězec w; $|w| = n$ a regulární jazyk L. Lze algoritmicky rozhodnout, zda je $w \in L$.

Rozhodovací problémy pro regulární jazyky!

Lemma (Test ne–prázdnosti regulárního jazyka)

Lze algoritmicky rozhodnout, zda jazyk přijímaný DFA, NFA, ϵ-NFA je prázdný.

Jazyk je prázdný právě když žádný z koncových stavů není dosažitelný.
Dosažitelnost lze testovat $O(|Q|^2)$.

Lemma (Test náležení do regulárního jazyka)

Pro daný řetězec w; $|w| = n$ a regulární jazyk L. Lze algoritmicky rozhodnout, zda je $w \in L$.

- DFA: Spusť automat; pokud $|w| == n$, při dobré reprezentaci a konstatním čase přechodu $O(n)$.
- NFA o s stavech: čas $O(ns^2)$. Každý vstupní symbol algoritmu má na vstupu s stavy předchozího kroku, kterých je nejvýš s.
- ϵ–NFA – nejdříve určíme ϵ–uzávěr, pak aplikujeme přechodové funkce a ϵ–uzávěr na výsledek.

Rozhodovací problémy pro regulární jazyky!

Lemma (Test ne–prázdnosti regulárního jazyka)

Lze algoritmicky rozhodnout, zda jazyk přijímaný DFA, NFA, ϵ-NFA je prázdný.

Jazyk je prázdný právě když žádný z koncových stavů není dosažitelný.
Dosažitelnost lze testovat $O(|Q|^2)$.

Lemma (Test náležení do regulárního jazyka)

Pro daný řetězec w; $|w| = n$ a regulární jazyk L. Lze algoritmicky rozhodnout, zda je $w \in L$.

- DFA: Spusť automat; pokud $|w| == n$, při dobré reprezentaci a konstatním čase přechodu $O(n)$.
- NFA o s stavech: čas $O(ns^2)$. Každý vstupní symbol aplikujeme na všechny stavy předchozího kroku, kterých je nejvýš s.
- ϵ–NFA – nejdříve určíme ϵ–uzávěr. Pak aplikujeme předchozí algoritmus a ϵ–uzávěr na výsledek.

Rozhodovací problémy pro regulární jazyky!

Lemma (Test ne–prázdnosti regulárního jazyka)

Lze algoritmicky rozhodnout, zda jazyk přijímaný DFA, NFA, ϵ-NFA je prázdný.

Jazyk je prázdný právě když žádný z koncových stavů není dosažitelný. Dosažitelnost lze testovat $O(|Q|^2)$.

Lemma (Test náležení do regulárního jazyka)

Pro daný řetězec w; $|w| = n$ a regulární jazyk L. Lze algoritmicky rozhodnout, zda je $w \in L$.

- DFA: Spusť automat; pokud $|w| = n$, při dobré reprezentaci a konstatním čase přechodu $O(n)$.
- NFA o s stavech: čas $O(ns^2)$. Každý vstupní symbol aplikujeme na všechny stavy předchozího kroku, kterých je nejvýš s.
- ϵ–NFA - nejdříve určíme ϵ–uzávěr. Pak aplikujeme přechodovou funkci a ϵ–uzávěr na výsledek.

Rozhodovací problémy pro regulární jazyky!

Lemma (Test ne–prázdnosti regulárního jazyka)

Lze algoritmicky rozhodnout, zda jazyk přijímaný DFA, NFA, ϵ-NFA je prázdný.

Jazyk je prázdný právě když žádný z koncových stavů není dosažitelný. Dosažitelnost lze testovat $O(|Q|^2)$.

Lemma (Test náležení do regulárního jazyka)

Pro daný řetězec w; $|w| = n$ a regulární jazyk L. Lze algoritmicky rozhodnout, zda je $w \in L$.

- DFA: Spusť automat; pokud $|w| = n$, při dobré reprezentaci a konstatním čase přechodu $O(n)$.
- NFA o s stavech: čas $O(ns^2)$. Každý vstupní symbol aplikujeme na všechny stavy předchozího kroku, kterých je nejvýš s.
- ϵ–NFA - nejdříve určíme ϵ–uzávěr. Pak aplikujeme přechodovou funkci a ϵ–uzávěr na výsledek.

Rozhodovací problémy pro regulární jazyky!

Lemma (Test ne–prázdnosti regulárního jazyka)

Lze algoritmicky rozhodnout, zda jazyk přijímaný DFA, NFA, ϵ-NFA je prázdný.

Jazyk je prázdný právě když žádný z koncových stavů není dosažitelný. Dosažitelnost lze testovat $O(|Q|^2)$.

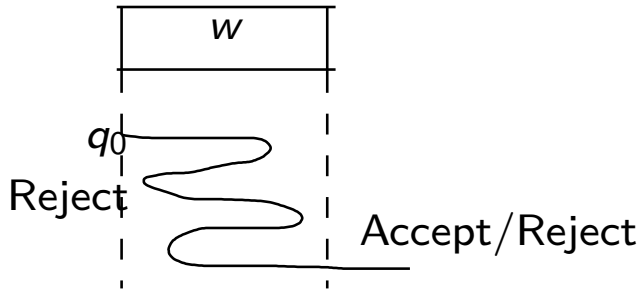
Lemma (Test náležení do regulárního jazyka)

Pro daný řetězec w; $|w| = n$ a regulární jazyk L. Lze algoritmicky rozhodnout, zda je $w \in L$.

- DFA: Spusť automat; pokud $|w| = n$, při dobré reprezentaci a konstatním čase přechodu $O(n)$.
- NFA o s stavech: čas $O(ns^2)$. Každý vstupní symbol aplikujeme na všechny stavy předchozího kroku, kterých je nejvýš s.
- $\epsilon-$NFA - nejdříve určíme ϵ–uzávěr. Pak aplikujeme přechodovou funkci a ϵ–uzávěr na výsledek.

Dvousměrné (dvoucestné) konečné automaty

- Konečný automat provádí následující činnosti:
  - přečte písmeno
  - změní stav vnitřní jednotky
  - posune čtecí hlavu doprava
- Čtecí hlava se nesmí vracet.



Definition 5.1 (Deterministické Dvousměrné konečné automaty)

Deterministickým dvousměrným konečným automatem nazýváme pětici $A = (Q, \Sigma, \delta, q_0, F)$, kde

1. Q je konečná množina stavů,
2. Σ je konečná množina vstupních symbolů
3. přechodové funkce δ je zobrazení $Q \times \Sigma \to Q \times \{-1, 1\}$ rozšířené o pohyb hlavy
4. $q_0 \in Q$ počáteční stav
5. množina přijímajících stavů $F \subseteq Q$.

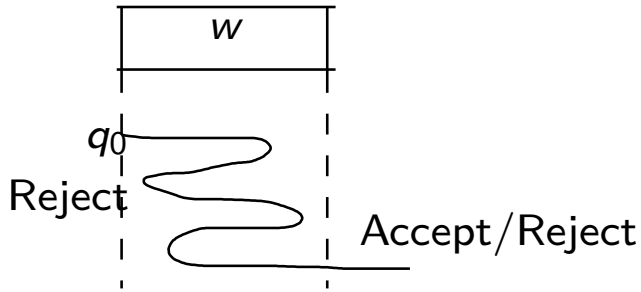
Pozn.: Je deterministický, nedeterministický $\delta_N : Q \times \Sigma \to \mathcal{P}(Q \times \{-1, 1\})$.

Pozn.2: Nulový pohyb hlavy lze, jen trochu zkomplikuje důkaz dále.

- Reprezentujeme opět stavovým diagramem, tabulkou

Dvousměrné (dvoucestné) konečné automaty

- Konečný automat provádí následující činnosti:
  - přečte písmeno
  - změní stav vnitřní jednotky
  - posune čtecí hlavu doprava
- Čtecí hlava se nesmí vracet.



Definition 5.1 (Deterministické Dvousměrné konečné automaty)

Deterministickým dvousměrným konečným automatem nazýváme pětici $A = (Q, \Sigma, \delta, q_0, F)$, kde

1. Q je konečná množina stavů,
2. Σ je konečná množina vstupních symbolů
3. přechodové funkce δ je zobrazení $Q \times \Sigma \to Q \times \{-1, 1\}$ rozšířené o pohyb hlavy
4. $q_0 \in Q$ počáteční stav
5. množina přijímajících stavů $F \subseteq Q$.

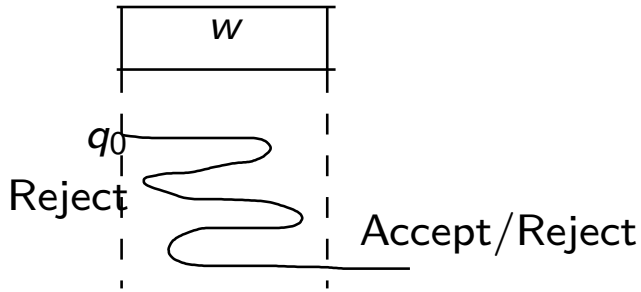
Pozn.: Je deterministický, nedeterministický $\delta_N : Q \times \Sigma \to \mathcal{P}(Q \times \{-1, 1\})$.

Pozn.2: Nulový pohyb hlavy lze, jen trochu zkomplikuje důkaz dále.

- Reprezentujeme opět stavovým diagramem, tabulkou

Dvousměrné (dvoucestné) konečné automaty

- Konečný automat provádí následující činnosti:
  - přečte písmeno
  - změní stav vnitřní jednotky
  - posune čtecí hlavu doprava
- Čtecí hlava se nesmí vracet.



Definition 5.1 (Deterministické Dvousměrné konečné automaty)

Deterministickým dvousměrným konečným automatem nazýváme pětici $A = (Q, \Sigma, \delta, q_0, F)$, kde

1. Q je konečná množina stavů,
2. Σ je konečná množina vstupních symbolů
3. přechodové funkce δ je zobrazení $Q \times \Sigma \to Q \times \{-1, 1\}$ rozšířené o pohyb hlavy
4. $q_0 \in Q$ počáteční stav
5. množina přijímajících stavů $F \subseteq Q$.

Pozn.: Je deterministický, nedeterministický $\delta_N : Q \times \Sigma \to \mathcal{P}(Q \times \{-1, 1\})$.

Pozn.2: Nulový pohyb hlavy lze, jen trochu zkomplikuje důkaz dále.

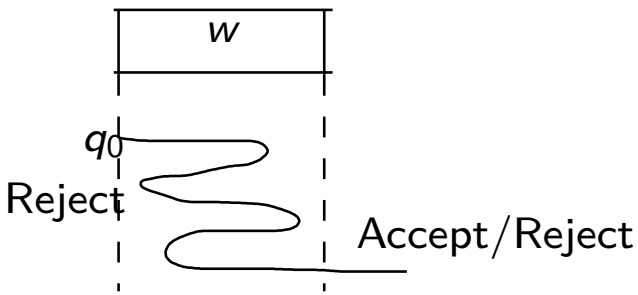
- Reprezentujeme opět stavovým diagramem, tabulkou

Výpočet dvousměrného automatu

Definition 5.2 (Výpočet dvousměrného automatu)

Slovo w je **přijato dvousměrným konečným automatem**, pokud:

- výpočet začal na prvním písmenu slova w vlevo v počátečním stavu
- čtecí hlava poprvé opustila slovo w vpravo v některém přijímajícím stavu
- mimo čtené slovo není výpočet definován (výpočet zde končí a slovo není přijato).



- Ke slovům si můžeme přidat speciální koncové znaky $\# \notin \Sigma$
- funkce $\partial_\#$ odstraní $\#$ zleva, $\partial_\#^R$ zprava.

Lemma

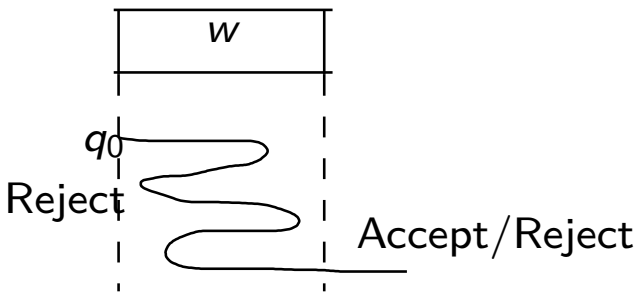
Je–li $L(A) = \{\#w\# | w \in L \subseteq \Sigma^*\}$ *regulární, potom je regulární i* $L = \partial_\# \partial_\#^R (L(A) \cap \#\Sigma^*\#)$.

Výpočet dvousměrného automatu

Definition 5.2 (Výpočet dvousměrného automatu)

Slovo w je **přijato dvousměrným konečným automatem**, pokud:

- výpočet začal na prvním písmenu slova w vlevo v počátečním stavu
- čtecí hlava poprvé opustila slovo w vpravo v některém přijímajícím stavu
- mimo čtené slovo není výpočet definován (výpočet zde končí a slovo není přijato).



- Ke slovům si můžeme přidat speciální koncové znaky $\# \notin \Sigma$
- funkce $\partial_\#$ odstraní $\#$ zleva, $\partial_\#^R$ zprava.

Lemma

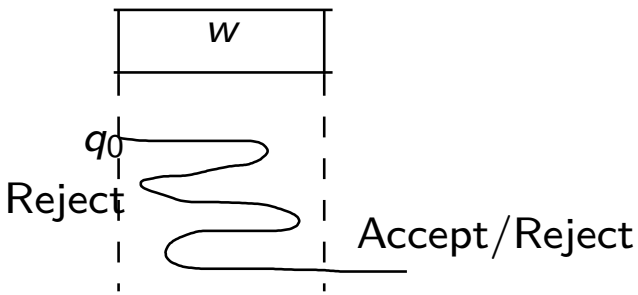
Je–li $L(A) = \{\#w\# | w \in L \subseteq \Sigma^*\}$ *regulární, potom je regulární i* $L = \partial_\# \partial_\#^R (L(A) \cap \#\Sigma^*\#)$.

Výpočet dvousměrného automatu

Definition 5.2 (Výpočet dvousměrného automatu)

Slovo w je **přijato dvousměrným konečným automatem**, pokud:

- výpočet začal na prvním písmenu slova w vlevo v počátečním stavu
- čtecí hlava poprvé opustila slovo w vpravo v některém přijímajícím stavu
- mimo čtené slovo není výpočet definován (výpočet zde končí a slovo není přijato).



- Ke slovům si můžeme přidat speciální koncové znaky $\# \notin \Sigma$
- funkce $\partial_\#$ odstraní $\#$ zleva, $\partial_\#^R$ zprava.

Lemma

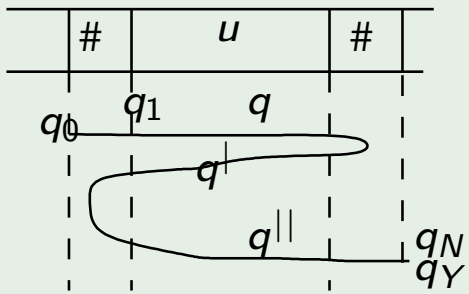
Je–li $L(A) = \{\#w\# | w \in L \subseteq \Sigma^*\}$ *regulární,*
potom je regulární i $L = \partial_\# \partial_\#^R (L(A) \cap \#\Sigma^*\#)$.

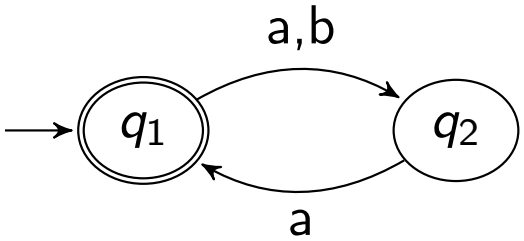
Příklad dvousměrného automatu

Example 5.1 (Příklad dvousměrného automatu)

Nechť $A = (Q, \Sigma, \delta, q_1, F)$. Dvousměrný konečný automat $B = (Q \cup Q^{|} \cup Q^{||} \cup \{q_0, q_N, q_Y\}, \Sigma \cup \{\#\}, \delta^{|}, q_0, \{q_Y\})$ přijímající jazyk $L(B) = \{\#u\# | uu \in L(A)\}$ (toto NENÍ levý ani pravý kvocient!) definujeme následovně:

$\delta^{\|}$	$x \in \Sigma$	$\#$	poznámka
q_0	$q_N, -1$	$q_1, +1$	q_1 je počátek A
q	$p, +1$	$q^{\|}, -1$	$p = \delta(q, x)$
$q^{\|}$	$q^{\|}, -1$	$q^{\|\|}, +1$	
$q^{\|\|}$	$p^{\|\|}, +1$	$q_Y, +1$	$q \in F, p = \delta(q, x)$
$q^{\|\|}$	$p^{\|\|}, +1$	$q_N, +1$	$q \notin F, p = \delta(q, x)$
q_N	$q_N, +1$	$q_N, +1$	
q_Y	$q_N, +1$	$q_N, +1$	





Dvousměrné a jednosměrné konečné automaty

Theorem 5.1

Jazyky přijímané dvousměrnými konečnými automaty jsou právě regulární jazyky.

Proof: konečný automat → dvousměrný automat

- Konečný automat předeveme na dvousměrný přidáním posunu hlavy vpravo
- $A = (Q, \Sigma, \delta, q_0, F) \rightarrow 2A = (Q, \Sigma, \delta^{|}, q_0, F)$, kde $\delta^{|}(q, x) = (\delta(q, x), +1)$.

□

- Možnost pohybovat čtecí hlavou po pásce nezvětšila sílu konečného automatu (dokud na pásku nic nepíšeme!).
- Pro důkaz potřebujeme přípravu.

Dvousměrné a jednosměrné konečné automaty

Theorem 5.1

Jazyky přijímané dvousměrnými konečnými automaty jsou právě regulární jazyky.

Proof: konečný automat $\rightarrow$ dvousměrný automat

- Konečný automat předeveme na dvousměrný přidáním posunu hlavy vpravo
- $A = (Q, \Sigma, \delta, q_0, F) \rightarrow 2A = (Q, \Sigma, \delta^{|}, q_0, F)$, kde $\delta^{|}(q, x) = (\delta(q, x), +1)$.

□

- Možnost pohybovat čtecí hlavou po pásce nezvětšila sílu konečného automatu (dokud na pásku nic nepíšeme!).
- Pro důkaz potřebujeme přípravu.

Dvousměrné a jednosměrné konečné automaty

Theorem 5.1

Jazyky přijímané dvousměrnými konečnými automaty jsou právě regulární jazyky.

Proof: konečný automat $\rightarrow$ dvousměrný automat

- Konečný automat předeveme na dvousměrný přidáním posunu hlavy vpravo
- $A = (Q, \Sigma, \delta, q_0, F) \rightarrow 2A = (Q, \Sigma, \delta^{|}, q_0, F)$, kde $\delta^{|}(q, x) = (\delta(q, x), +1)$.

□

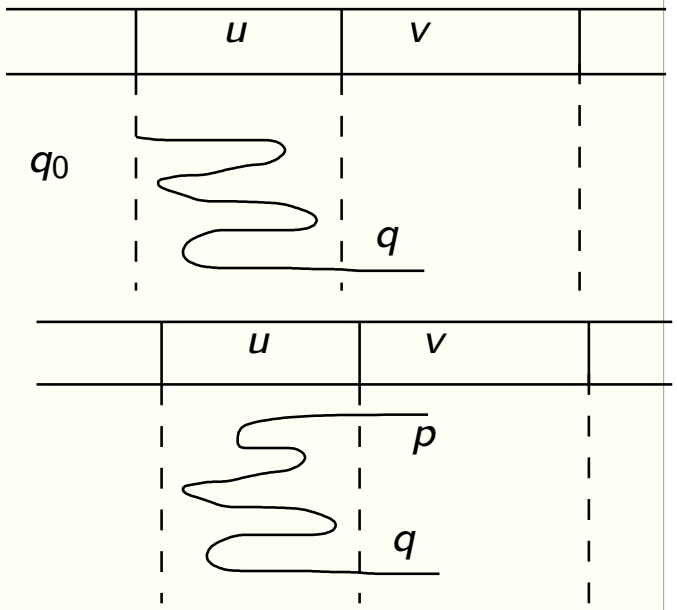
- Možnost pohybovat čtecí hlavou po pásce nezvětšila sílu konečného automatu (dokud na pásku nic nepíšeme!).
- Pro důkaz potřebujeme přípravu.

Funkce f_u popisující výpočet 2DFA nad slovem u

Algorithm: Funkce f_u popisující výpočet 2DFA nad slovem u

Definujeme funkci $f_u : Q \cup \{q_0^{\vdash}\} \to Q \cup \{0\}$

- $f_u(q_0^{\vdash})$ popisuje v jakém stavu poprvé odejdeme vpravo, pokud začneme výpočet vlevo v počátečním stavu q_0,
- symbol 0 značí, že daná situace nenastane (odejdeme vlevo nebo cyklus),
- $f_u(p); p \in Q$ v jakém stavu opět odejdeme vpravo, pokud začneme výpočet vpravo v p
- Definujeme ekvivalenci slov následovně: $u \sim w \Leftrightarrow_{def} f_u = f_w$,
  - tj. slova jsou ekvivalentní pokud mají stejné výpočetní funkce.



Regulárnost 2DFA

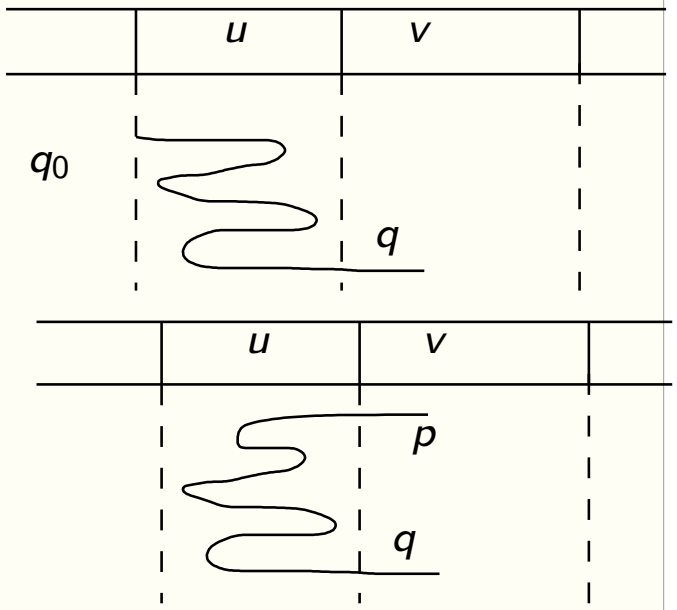
Ekvivalence $\sim$ je ekvivalence, má konečný index, je to pravá kongruence, jazyk 2DFA odpovídá sjednocení tříd $f_w(q_0^{\vdash}) \in F$.

Funkce f_u popisující výpočet 2DFA nad slovem u

Algorithm: Funkce f_u popisující výpočet 2DFA nad slovem u

Definujeme funkci $f_u : Q \cup \{q_0^{\vdash}\} \to Q \cup \{0\}$

- $f_u(q_0^{\vdash})$ popisuje v jakém stavu poprvé odejdeme vpravo, pokud začneme výpočet vlevo v počátečním stavu q_0,
- symbol 0 značí, že daná situace nenastane (odejdeme vlevo nebo cyklus),
- $f_u(p); p \in Q$ v jakém stavu opět odejdeme vpravo, pokud začneme výpočet vpravo v p
- Definujeme ekvivalenci slov následovně: $u \sim w \Leftrightarrow_{def} f_u = f_w$,
  - tj. slova jsou ekvivalentní, pokud mají stejné výsledné funkce.



Regulárnost 2DFA

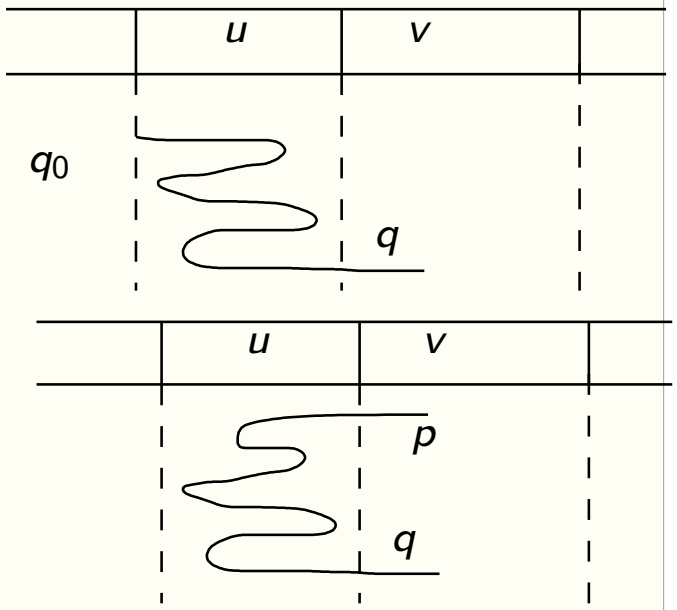
Ekvivalence $\sim$ je ekvivalence, má konečný index, je to pravá kongruence, jazyk 2DFA odpovídá sjednocení tříd $f_w(q_0^{\vdash}) \in F$.

Funkce f_u popisující výpočet 2DFA nad slovem u

Algorithm: Funkce f_u popisující výpočet 2DFA nad slovem u

Definujeme funkci $f_u : Q \cup \{q_0^{\vdash}\} \to Q \cup \{0\}$

- $f_u(q_0^{\vdash})$ popisuje v jakém stavu poprvé odejdeme vpravo, pokud začneme výpočet vlevo v počátečním stavu q_0,
- symbol 0 značí, že daná situace nenastane (odejdeme vlevo nebo cyklus),
- $f_u(p); p \in Q$ v jakém stavu opět odejdeme vpravo, pokud začneme výpočet vpravo v p



- Definujeme ekvivalenci slov následovně: $u \sim w \Leftrightarrow_{def} f_u = f_w$,
  - tj. slova jsou ekvivalentní pokud mají stejné 'výpočtové' funkce.

Regulárnost 2DFA

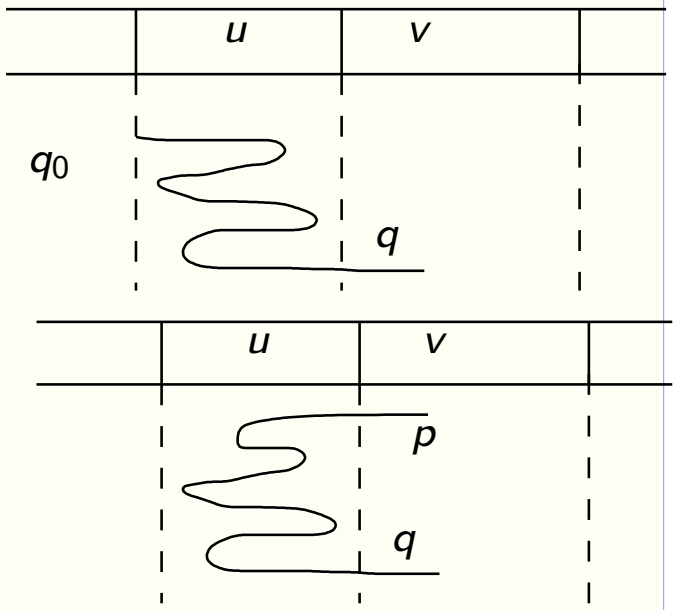
Ekvivalence $\sim$ je ekvivalence, má konečný index, je to pravá kongruence, jazyk 2DFA odpovídá sjednocení tříd $f_w(q_0^{\vdash}) \in F$.

Funkce f_u popisující výpočet 2DFA nad slovem u

Algorithm: Funkce f_u popisující výpočet 2DFA nad slovem u

Definujeme funkci $f_u : Q \cup \{q_0^{\vdash}\} \to Q \cup \{0\}$

- $f_u(q_0^{\vdash})$ popisuje v jakém stavu poprvé odejdeme vpravo, pokud začneme výpočet vlevo v počátečním stavu q_0,
- symbol 0 značí, že daná situace nenastane (odejdeme vlevo nebo cyklus),
- $f_u(p); p \in Q$ v jakém stavu opět odejdeme vpravo, pokud začneme výpočet vpravo v p



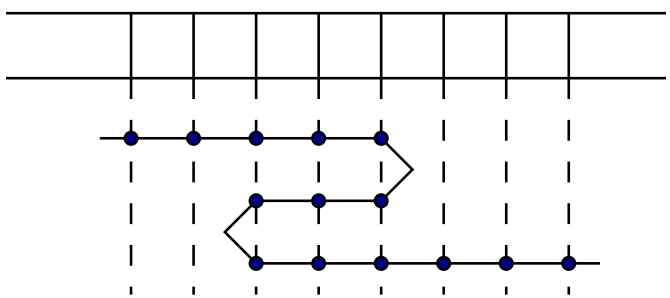
- Definujeme ekvivalenci slov následovně: $u \sim w \Leftrightarrow_{def} f_u = f_w$,
  - tj. slova jsou ekvivalentní pokud mají stejné 'výpočtové' funkce.

Regulárnost 2DFA

Ekvivalence $\sim$ je ekvivalence, má konečný index, je to pravá kongruence, jazyk 2DFA odpovídá sjednocení tříd $f_w(q_0^{\vdash}) \in F$.

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



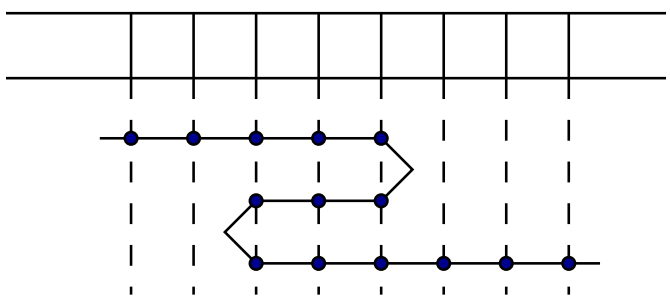
Pozorování:

- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA → NFA

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



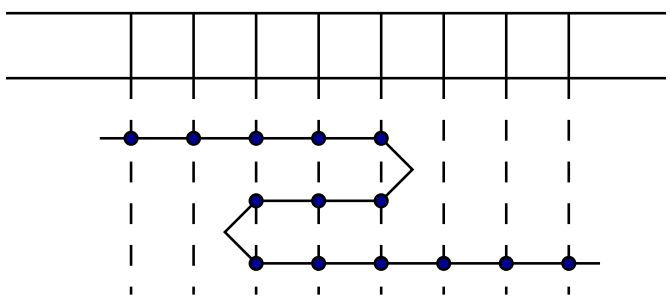
Pozorování:

- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA → NFA

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



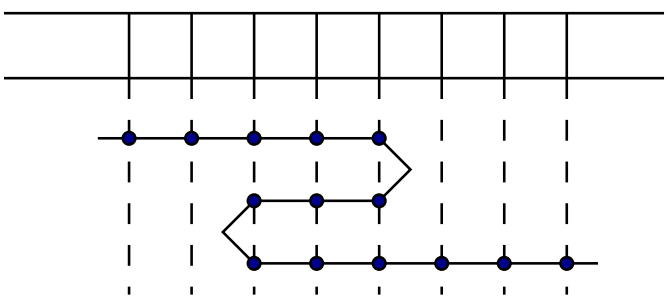
Pozorování:

- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA → NFA

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



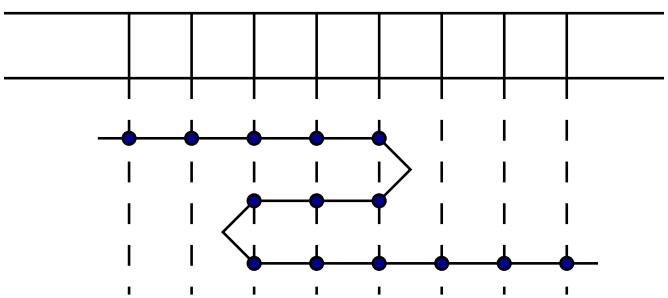
Pozorování:

- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA → NFA

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



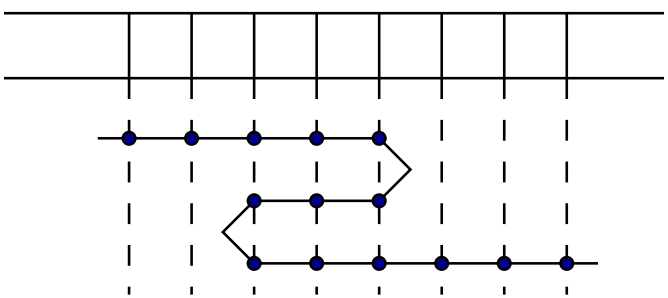
Pozorování:

- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA → NFA

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



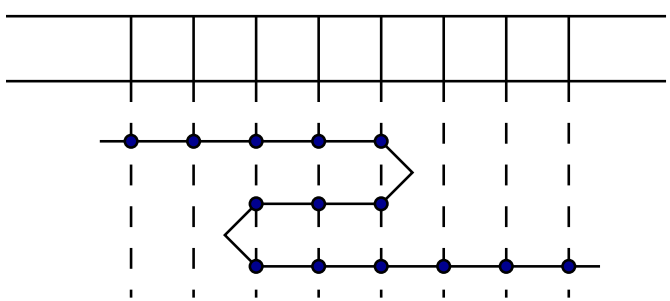
Pozorování:

- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA → NFA

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



Pozorování:

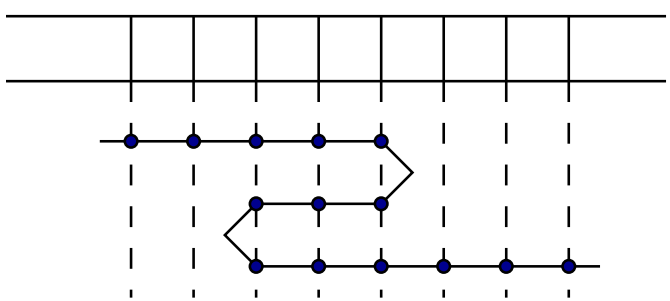
- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA → NFA

- Najdeme všechny možné řezy – posloupností stavů (je jich konečně mnoho).

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



Pozorování:

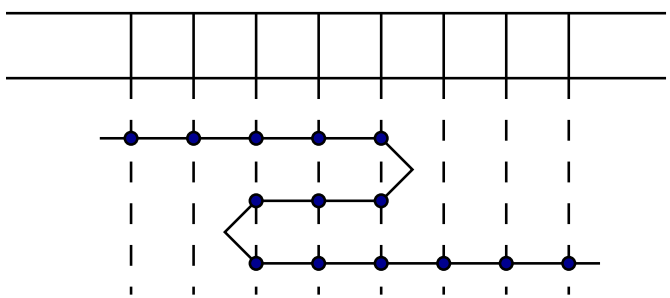
- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA → NFA

- Najdeme všechny možné řezy – posloupnosti stavů (je jich konečně mnoho).
- Mezi řezy definujeme nedeterministické přechody podle čteného symbolu.

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



Pozorování:

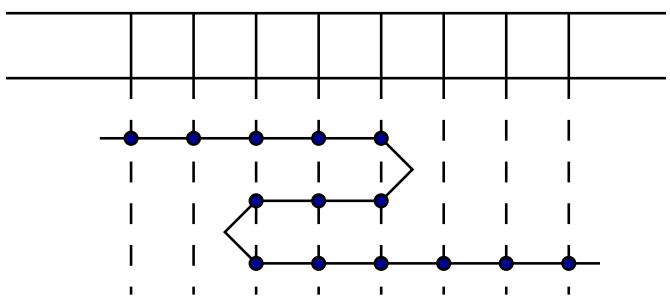
- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA $\rightarrow$ NFA

- Najdeme všechny možné řezy – posloupnosti stavů (je jich konečně mnoho).
- Mezi řezy definujeme nedeterministické přechody podle čteného symbolu.
- Rekonstruujeme výpočet skládáním řezů jako puzzle.

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



Pozorování:

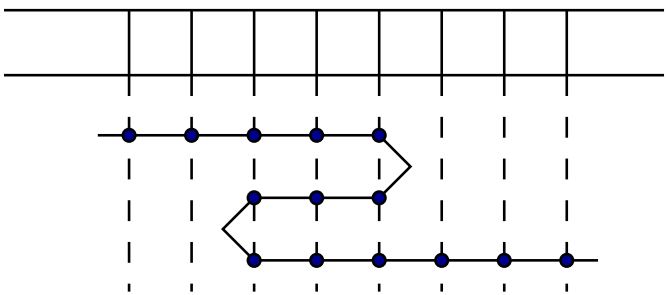
- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

Algorithm: 2DFA $\rightarrow$ NFA

- Najdeme všechny možné řezy – posloupnosti stavů (je jich konečně mnoho).
- Mezi řezy definujeme nedeterministické přechody podle čteného symbolu.
- Rekonstruujeme výpočet skládáním řezů jako puzzle.

Konstruktivní důkaz věty o 2DFA

- Potřebujeme převést návraty na lineární výpočet.
- Zajímají nás jen přijímající výpočty
- Díváme se na řezy mezi symboly (v jakém stavu přechází na další políčko)



Pozorování:

- stavy se v přechodu řezu střídají (doprava, doleva)
- první stav jde doprava, poslední také doprava
- v deterministických přijímajících výpočtech nejsou cykly
- první a poslední řez obsahují jediný stav.

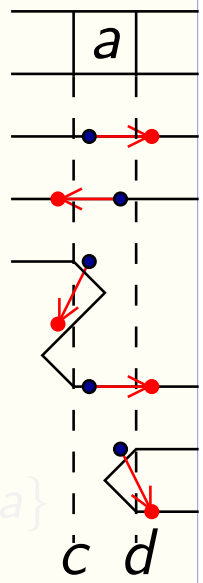
Algorithm: 2DFA $\rightarrow$ NFA

- Najdeme všechny možné řezy – posloupnosti stavů (je jich konečně mnoho).
- Mezi řezy definujeme nedeterministické přechody podle čteného symbolu.
- Rekonstruujeme výpočet skládáním řezů jako puzzle.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})\ \&\ (\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce $h : \{1, \ldots, |c|\} \to \{1, \ldots, |d|\}$ taková, že
  - zachovává uspořádání
  - pro $h(i) = i$ platí $(d[i] = 1) = (c[i], a)$
  - pro $h(i) = i + 1$ platí $(d[i] = 1) = (c[i], a)$



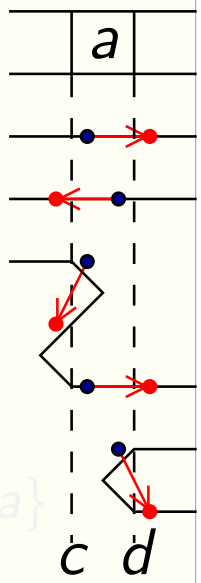
$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})\ \&\ (\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce $h : \{1, \ldots, |c|\} \to \{1, \ldots, |d|\}$ taková, že
  - zachovává uspořádání
  - pro $h(q) =$ sudá je $(d(q) - 1) = \delta(q, a)$
  - pro $h(q) =$ lichá je $(d(q) - 1) = \delta(q, a)$



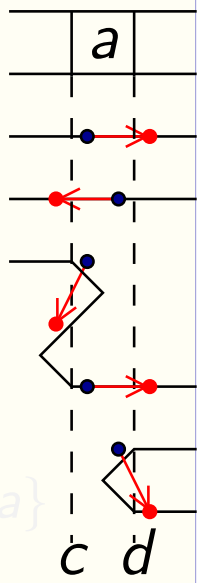
$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})\ \&\ (\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce $h : \{1, \ldots, |c|\} \to \{1, \ldots, |d|\}$ taková, že
  - zachovává uspořádání
  - pro $h(i) \in$ sudá je $(d(i) = 1) = (c(i) = a)$
  - pro $h(i) =$ lichá je $(d(i) = 1) = (c(i) = a)$



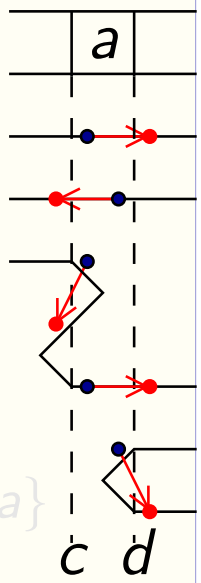
$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})$ & $(\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce $h : \{1, \ldots, k\} \to \{1, \ldots, l\}$ (kde $k = |c|$, $l = |d|$)
  - zachovává uspořádání
  - pro $h(i)$ sudé je $(d[h(i) - 1] = d[q, a])$
  - pro $h(i)$ liché je $(c[h(i) - 1] = c[q, a])$



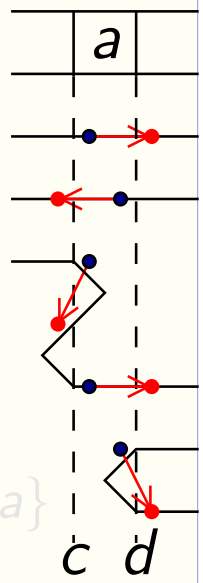
$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})\ \&\ (\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce h z $\{1, \ldots, |d|\}$ do $\{1, \ldots, |c|\}$, takže
  - zachovává uspořádání
  - pro $h(i) \in$ sudá je $(d(i) = 1) = (c(h(i), a)$
  - pro $h(i) =$ lichá je $(d(i) = 1) = (c(h(i), a)$



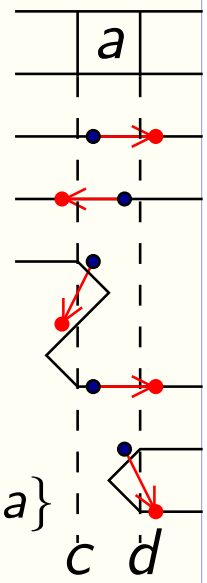
$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})$ & $(\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce: $h : c_{odd} \cup d_{even} \to c_{even} \cup d_{odd}$, tak, že:
  - zachovává uspořádání
  - pro $h(q) \in c_{even}$ je $(h(q), -1) = \delta(q, a)$
  - pro $h(q) \in d_{odd}$ je $(h(q), +1) = \delta(q, a)$.



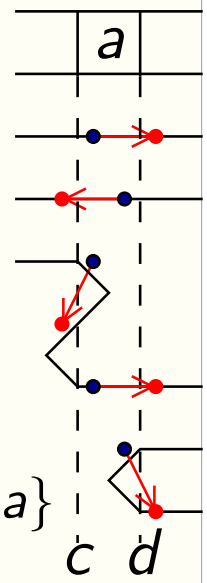
$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})$ & $(\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce: $h : c_{odd} \cup d_{even} \to c_{even} \cup d_{odd}$, tak, že:
  - zachovává uspořádání
  - pro $h(q) \in c_{even}$ je $(h(q), -1) = \delta(q, a)$
  - pro $h(q) \in d_{odd}$ je $(h(q), +1) = \delta(q, a)$.



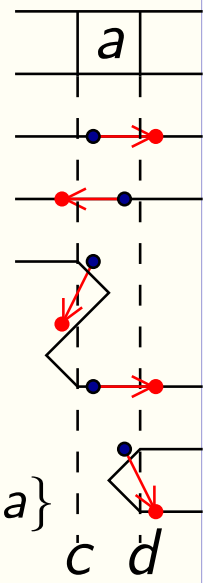
$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})$ & $(\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce: $h : c_{odd} \cup d_{even} \to c_{even} \cup d_{odd}$, tak, že:
  - zachovává uspořádání
  - pro $h(q) \in c_{even}$ je $(h(q), -1) = \delta(q, a)$
  - pro $h(q) \in d_{odd}$ je $(h(q), +1) = \delta(q, a)$.



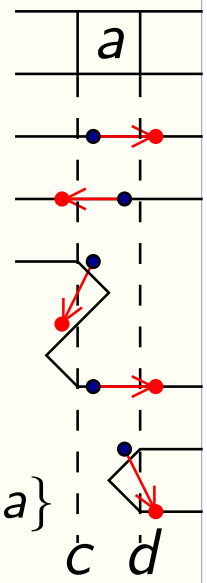
$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})$ & $(\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce: $h : c_{odd} \cup d_{even} \to c_{even} \cup d_{odd}$, tak, že:
  - zachovává uspořádání
  - pro $h(q) \in c_{even}$ je $(h(q), -1) = \delta(q, a)$
  - pro $h(q) \in d_{odd}$ je $(h(q), +1) = \delta(q, a)$.



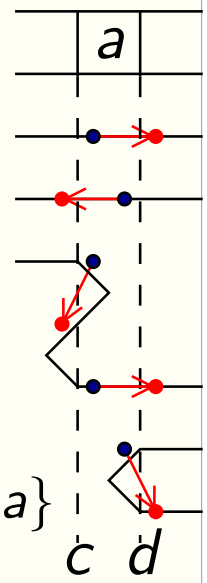
$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Algorithm: Formální převod 2DFA na NFA

Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je dvousměrný (deterministický) konečný automat. Definujeme ekvivalentní nedeterministický automat $B = (Q^{|}, \Sigma, \delta^{|}, (q_0), F^{|})$ kde:

- $Q^{|}$ jsou všechny korektní přechodové posloupnosti
  - posloupnosti stavů $(q^1, \ldots, q^k); q^i \in Q$
  - délka posloupnosti je lichá $(k = 2m + 1)$
  - žádný stav se neopakuje na liché ani na sudé pozici $(\forall i \neq j)\ (q^{2i} \neq q^{2j})$ & $(\forall i \neq j)(q^{2i+1} \neq q^{2j+1})$
- $F^{|} = \{(q) | q \in F\}$ posloupnosti délky 1
- $\delta^{|}(c, a) = \{d | d \in Q^{|} \& c \xrightarrow{a} d$ je lokálně konzistentní přechod pro $a\}$
  - existuje bijekce: $h : c_{odd} \cup d_{even} \rightarrow c_{even} \cup d_{odd}$, tak, že:
  - zachovává uspořádání
  - pro $h(q) \in c_{even}$ je $(h(q), -1) = \delta(q, a)$
  - pro $h(q) \in d_{odd}$ je $(h(q), +1) = \delta(q, a)$.



$L(A) = L(B)$

Trajektorie 2DFA A odpovídá řezům v FA B, odtud $L(A) = L(B)$.

Příklad převodu 2DFA na NKA

- Mějme následující dvousměrný konečný automat:

	a	b
$\rightarrow p$	p,+1	q,+1
$*q$	q,+1	r,-1
r	p,+1	r,-1

Možné řezy a jejich přechody

- Doleva jedině r – všechny sudé pozice r, tj. jediná sudá
- možné řezy: $(p), (q), (p, r, q), (q, r, p)$.

	a	b
$\rightarrow$ (p)	(p)	(q)
$*$(q)	(q),(q,r,p)	
(p,r,q)		
(q,r,p)		(q)

Ukázka (zacykleného, nepřijímajícího) výpočtu:

a	a	b	a	a	b	a	a	b	b
p	p	p	q	q	q				
				r					
				p	q	q	q		
						r			
						p	q		
						r	r		

Výsledný NFA:

Příklad převodu 2DFA na NKA

- Mějme následující dvousměrný konečný automat:

	a	b
$\rightarrow p$	p,+1	q,+1
$*q$	q,+1	r,-1
r	p,+1	r,-1

Možné řezy a jejich přechody

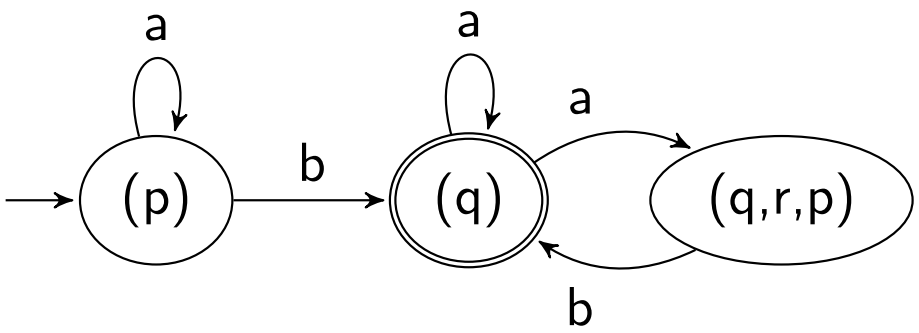
- Doleva jedině r – všechny sudé pozice r, tj. jediná sudá
- možné řezy: $(p), (q), (p, r, q), (q, r, p)$.

	a	b
$\rightarrow$ (p)	(p)	(q)
$*$(q)	(q),(q,r,p)	
(p,r,q)		
(q,r,p)		(q)

Ukázka (zacykleného, nepřijímajícího) výpočtu:

a	a	b	a	a	b	a	a	b	b
p	p	p	q	q	q				
				r					
					p	q	q	q	
							r		
								p	q
							r	r	

Výsledný NFA:



Příklad převodu 2DFA na NKA

- Mějme následující dvousměrný konečný automat:

	a	b
$\rightarrow p$	p,+1	q,+1
$*q$	q,+1	r,-1
r	p,+1	r,-1

Možné řezy a jejich přechody

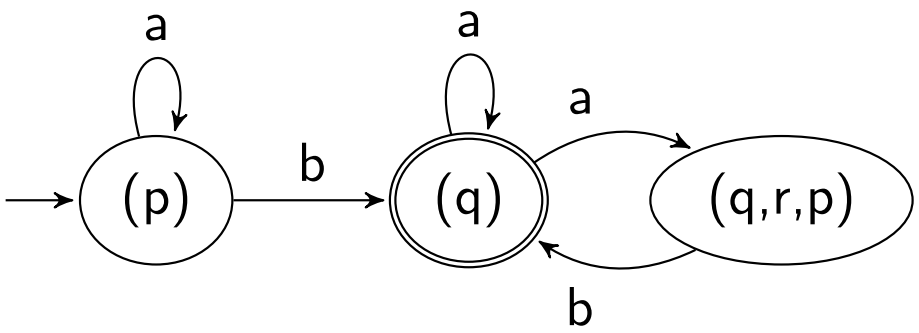
- Doleva jedině r – všechny sudé pozice r, tj. jediná sudá
- možné řezy: $(p), (q), (p, r, q), (q, r, p)$.

	a	b
$\rightarrow$ (p)	(p)	(q)
$*$(q)	(q),(q,r,p)	
(p,r,q)		
(q,r,p)		(q)

Ukázka (zacykleného, nepřijímajícího) výpočtu:

a	a	b	a	a	b	a	a	b	b
p	p	p	q	q	q				
				r					
					p	q	q	q	
							r		
								p	q
							r	r	

Výsledný NFA:



Automaty s výstupem (motivace)

- Dosud jediná zpráva z automatu: 'Jsme v přijímajícím stavu'.
- Můžeme z FA získat více informací? Můžeme zaznamenat trasu výpočtu?

Moore: indikace stavů (všech, nejen koncových)

- v každé chvíli víme, kde se automat nachází
- Příklad: různé (regulární) čítače

Mealy: indikace přechodů

- po přečtení každého symbolu víme, co automat dělal
- Příklad: regulární překlad slov

Automat už není tak docela černá skříňka.

Automaty s výstupem (motivace)

- Dosud jediná zpráva z automatu: 'Jsme v přijímajícím stavu'.
- Můžeme z FA získat více informací? Můžeme zaznamenat trasu výpočtu?

Moore: indikace stavů (všech, nejen koncových)

- v každé chvíli víme, kde se automat nachází
- Příklad: různé (regulární) čítače

Mealy: indikace přechodů

- po přečtení každého symbolu víme, co automat dělal
- Příklad: regulární překlad slov

Automat už není tak docela černá skříňka.

Automaty s výstupem (motivace)

- Dosud jediná zpráva z automatu: 'Jsme v přijímajícím stavu'.
- Můžeme z FA získat více informací? Můžeme zaznamenat trasu výpočtu?

Moore: indikace stavů (všech, nejen koncových)

- v každé chvíli víme, kde se automat nachází
- Příklad: různé (regulární) čítače

Mealy: indikace přechodů

- po přečtení každého symbolu víme, co automat dělal
- Příklad: regulární překlad slov

Automat už není tak docela černá skříňka.

Mooreův stroj

Definition 5.3 (Mooreův stroj)

Mooreovým (sekvenčním) strojem nazýváme šestici $A = (Q, \Sigma, Y, \delta, \mu, q_0)$ resp. pětici $A = (Q, \Sigma, Y, \delta, \mu)$, kde

Q je konečná neprázdná množina stavů

Σ je konečná neprázdná množina symbolů (vstupní abeceda)

Y je konečná neprázdná množina symbolů (**výstupní abeceda**)

δ je zobrazení $Q \times \Sigma \to Q$ (přechodová funkce)

μ je zobrazení $Q \to Y$ (**značkovací funkce**)

$q_0 \in Q$ (počáteční stav)

- Někdy nás nezajímá počáteční stav, ale jen práce automatu
- značkovací funkce umožňuje suplovat roli koncových stavů
  - $F \subseteq Q$ nahradíme značkovací funkcí $\mu : Q \to \{0, 1\}$ takto:
    $\mu(q) = 0$ pokud $q \notin F$,
    $\mu(q) = 1$ pokud $q \in F$.

Mooreův stroj

Definition 5.3 (Mooreův stroj)

Mooreovým (sekvenčním) strojem nazýváme šestici $A = (Q, \Sigma, Y, \delta, \mu, q_0)$ resp. pětici $A = (Q, \Sigma, Y, \delta, \mu)$, kde

Q je konečná neprázdná množina stavů

Σ je konečná neprázdná množina symbolů (vstupní abeceda)

Y je konečná neprázdná množina symbolů (**výstupní abeceda**)

δ je zobrazení $Q \times \Sigma \to Q$ (přechodová funkce)

μ je zobrazení $Q \to Y$ (**značkovací funkce**)

$q_0 \in Q$ (počáteční stav)

- Někdy nás nezajímá počáteční stav, ale jen práce automatu
- značkovací funkce umožňuje suplovat roli koncových stavů
  - $F \subseteq Q$ nahradíme značkovací funkcí $\mu : Q \to \{0, 1\}$ takto:
    $\mu(q) = 0$ pokud $q \notin F$,
    $\mu(q) = 1$ pokud $q \in F$.

Příklad Mooreova stroje

Example 5.2 (Mooreův stroj pro tenis)

Mooreův stroj pro počítání tenisového skóre.

- Vstupní abeceda: ID hráče, který uhrál bod
- Výstupní abeceda & stavy: skóre (tj. $Q = Y$ a $\mu(q) = q$)

Stav/výstup	A	B
00:00	15:00	00:15
15:00	30:00	15:15
15:15	30:15	15:30
00:15	15:15	00:30
30:00	40:00	30:15
30:15	40:15	30:30
30:30	40:30	30:40
15:30	30:30	15:40
00:30	15:30	00:40
40:00	A	40:15
40:15	A	40:30
40:30	A	shoda
30:40	shoda	B
15:40	30:40	B
00:40	15:00	B
shoda	A:40	40:B
A:40	A	shoda
40:B	shoda	B
A	15:00	00:15
B	15:00	00:15

Mealyho stroj

Definition 5.4 (Mealyho stroj)

Mealyho (sekvenčním) strojem nazýváme šestici $A = (Q, \Sigma, Y, \delta, \lambda_M, q_0)$ resp. pětici $A = (Q, \Sigma, Y, \delta, \lambda_M)$, kde

Q je konečná neprázdná množina stavů

Σ je konečná neprázdná množina symbolů (vstupní abeceda)

Y je konečná neprázdná množina symbolů (výstupní abeceda)

δ je zobrazení $Q \times \Sigma \to Q$ (přechodová funkce)

λ_M je zobrazení $Q \times \Sigma \to Y$ (**výstupní funkce**)

$q_0 \in Q$ (počáteční stav)

- Výstup je určen stavem a vstupním symbolem
  - Mealyho stroj je obecnějším prostředkem než stroj Mooreův
  - Značkovací funkci $\mu : Q \to Y$ lze nahradit výstupní funkcí $\lambda_M : Q \times \Sigma \to Y$ například takto:

    $$\forall x \in \Sigma \ \lambda_M(q, x) = \mu(q)$$

    nebo $\forall x \in \Sigma \ \lambda_M(q, x) = \mu(\delta(q, x))$

Mealyho stroj

Definition 5.4 (Mealyho stroj)

Mealyho (sekvenčním) strojem nazýváme šestici $A = (Q, \Sigma, Y, \delta, \lambda_M, q_0)$ resp. pětici $A = (Q, \Sigma, Y, \delta, \lambda_M)$, kde

Q je konečná neprázdná množina stavů

Σ je konečná neprázdná množina symbolů (vstupní abeceda)

Y je konečná neprázdná množina symbolů (výstupní abeceda)

δ je zobrazení $Q \times \Sigma \to Q$ (přechodová funkce)

λ_M je zobrazení $Q \times \Sigma \to Y$ (**výstupní funkce**)

$q_0 \in Q$ (počáteční stav)

- Výstup je určen stavem a vstupním symbolem
  - Mealyho stroj je obecnějším prostředkem než stroj Mooreův
  - Značkovací funkci $\mu : Q \to Y$ lze nahradit výstupní funkcí $\lambda_M : Q \times \Sigma \to Y$ například takto:

    $$\forall x \in \Sigma \ \lambda_M(q, x) = \mu(q)$$

    nebo $\forall x \in \Sigma \ \lambda_M(q, x) = \mu(\delta(q, x))$

Příklad Mealyho stroje

Example 5.3 (Mealyho stroj)

Automat, který dělí vstupní slovo v binárním tvaru číslem 8 (celočíselně).

- Posun o tři bity doprava
- potřebujeme si pamatovat poslední trojici bitů
- vlastně tříbitová dynamická paměť

Stav\symbol	0	1
000	000/0	001/0
001	010/0	011/0
010	100/0	101/0
011	110/0	111/0
100	000/1	001/1
101	010/1	011/1
110	100/1	101/1
111	110/1	111/1

- I když nevíme, kde automat startuje, po třech symbolech začne počítat správně.

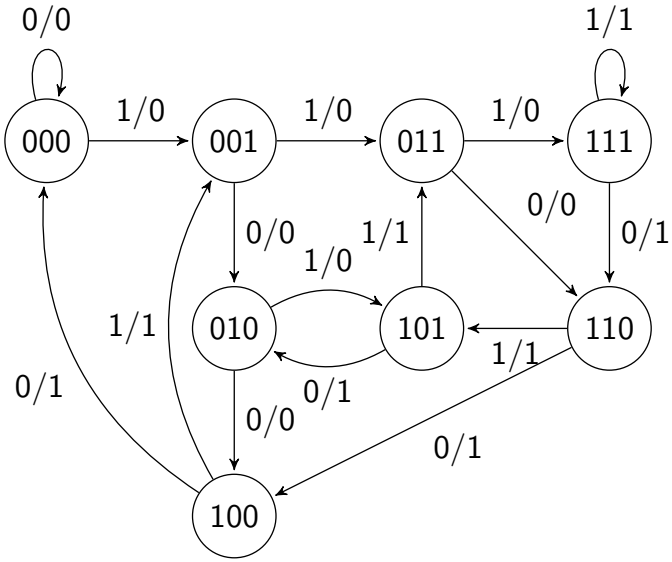
Příklad Mealyho stroje

Example 5.3 (Mealyho stroj)

Automat, který dělí vstupní slovo v binárním tvaru číslem 8 (celočíselně).

- Posun o tři bity doprava
- potřebujeme si pamatovat poslední trojici bitů
- vlastně tříbitová dynamická paměť

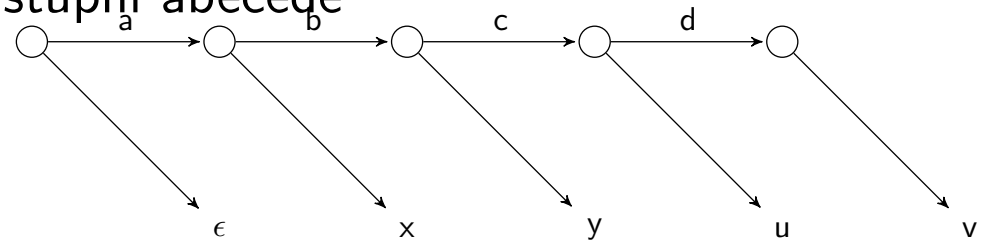
Stav\symbol	0	1
000	000/0	001/0
001	010/0	011/0
010	100/0	101/0
011	110/0	111/0
100	000/1	001/1
101	010/1	011/1
110	100/1	101/1
111	110/1	111/1



- I když nevíme, kde automat startuje, po třech symbolech začne počítat správně.

Výstup sekvenčních strojů

slovo ve vstupní abecedě $\rightarrow$ slovo ve výstupní abecedě



Mooreův stroj

značkovací funkce $\mu : Q \rightarrow Y$

$\mu^* : Q \times \Sigma^* \rightarrow Y^*$

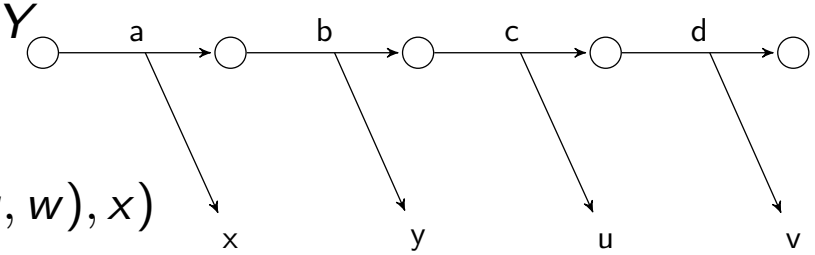
$\mu^*(q, \epsilon) = \epsilon$ (někdy $\mu^*(q, \epsilon) = q$)

$\mu^*(q, wx) = \mu^*(q, w).\mu(\delta^*(q, wx))$

Příklad: μ^*(00:00,AABA)=(00:00 .) 15:00 . 30:00 . 30:15 . 40:15

Mealyho stroj

výstupní funkce $\lambda_M : Q \times \Sigma \rightarrow Y$



$\lambda^*_M : Q \times \Sigma^* \rightarrow Y^*$

$\lambda^*_M(q, \epsilon) = \epsilon$

$\lambda^*_M(q, wx) = \lambda^*_M(q, w).\lambda_M(\delta^*(q, w), x)$

Příklad: λ^*_M(000,1101010)=0001101

Konečné automaty – shrnutí

Konečný automat

- redukovaný deterministický automat (lze definovat i jednoznačný)
- nedeterminismus ϵ–NFA, 2^n, (dvousměrný FA n^n)

Regulární výrazy

Automaty a jazyky

- regulární jazyky
- uzavřenost na množinové operace
- uzavřenost na řetězcové operace
- uzavřenost na substituci, homomorfizmus a inverzní homomorfizmus,
- automaty výše i regulární výrazy popisují stejnou třídu jazyků.

Charakteristika regulárních jazyků

- Mihyll–Nerodova věta (kongruence)
- Kleeneova věta (elementární jazyky a operace)
- Iterační (pumping) lemma (iterace podslov, jen nutná podmínka).

Dvousměrný automat, pokud nesmí psát na pásku, přijímá jen regulátrní jazyky.

Palindromy

Definition (palindrom)

Palindrom je řetězec w stejný při čtení zepředu i zezadu, tj. $w = w^R$.

- Příklady: `'otto'`; `'Madam, I'm Adam'`.

Lemma

Jazyk $L_{pal} = \{w | w = w^R, w \in \Sigma^*\}$ není regulární.

Example 6.1 (Bezkontextová gramatika pro palindromy)

$G = (\{S\}, \{0, 1\}, P, S)$, $P =$

1. $S \to \epsilon$
2. $S \to 0$
3. $S \to 1$
4. $S \to 0S0$
5. $S \to 1S1$

Proof:

- Důkaz sporem. Předpokládejme L_{pal} je regulární, nechť n je konstanta z pumping lemma, uvažujme slovo: $w = 0^n 1 0^n$.
- z pumping lemmatu lze rozložit na $w = xyz$, y obsahuje jednu nebo více z prvních n nul. Tedy xz má být v L_{pal} ale není, tj. L_{pal} není regulární.

□

Palindromy

Definition (palindrom)

Palindrom je řetězec w stejný při čtení zepředu i zezadu, tj. $w = w^R$.

- Příklady: `'otto'`; `'Madam, I'm Adam'`.

Lemma

Jazyk $L_{pal} = \{w | w = w^R, w \in \Sigma^*\}$ není regulární.

Example 6.1 (Bezkontextová gramatika pro palindromy)

$G = (\{S\}, \{0, 1\}, P, S)$, $P =$

1. $S \to \epsilon$
2. $S \to 0$
3. $S \to 1$
4. $S \to 0S0$
5. $S \to 1S1$

Proof:

- Důkaz sporem. Předpokládejme L_{pal} je regulární, nechť n je konstanta z pumping lemma, uvažujme slovo: $w = 0^n 1 0^n$.
- z pumping lemmatu lze rozložit na $w = xyz$, y obsahuje jednu nebo více z prvních n nul. Tedy xz má být v L_{pal} ale není, tj. L_{pal} není regulární.

□

Formální (generativní) gramatiky, Bezkontextové gramatiky

Definition 6.1 (Formální (generativní) gramatika)

Formální (generativní) gramatika je $G = (V, T, P, S)$ složena z

- konečné množiny **neterminálů** (variables) V
- neprázdné konečné množiny **terminálních symbolů** (**terminálů**) T
- **počáteční symbol** $S \in V$.
- konečné množiny **pravidel** (**produkcí**) P reprezentující rekurzivní definici jazyka. Každé pravidlo má tvar:
  - $\beta A \gamma \to \omega$, $A \in V, \beta, \gamma, \omega \in (V \cup T)^*$

    tj. levá strana obsahuje aspoň jeden neterminální symbol.

Definition (Bezkontextová gramatika CFG)

Bezkontextová gramatika (CFG) je $G = (V, T, P, S)$ gramatika, obsahující pouze pravidla tvaru

$$A \to \omega,\ A \in V, \omega \in (V \cup T)^*.$$

Formální (generativní) gramatiky, Bezkontextové gramatiky

Definition 6.1 (Formální (generativní) gramatika)

Formální (generativní) gramatika je $G = (V, T, P, S)$ složena z

- konečné množiny **neterminálů** (variables) V
- neprázdné konečné množiny **terminálních symbolů** (**terminálů**) T
- **počáteční symbol** $S \in V$.
- konečné množiny **pravidel** (**produkcí**) P reprezentující rekurzivní definici jazyka. Každé pravidlo má tvar:
  - $\beta A \gamma \to \omega$, $A \in V, \beta, \gamma, \omega \in (V \cup T)^*$

    tj. levá strana obsahuje aspoň jeden neterminální symbol.

Definition (Bezkontextová gramatika CFG)

Bezkontextová gramatika (CFG) je $G = (V, T, P, S)$ gramatika, obsahující pouze pravidla tvaru

$A \to \omega$, $A \in V, \omega \in (V \cup T)^*$.

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál

- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)

  - pouze pravidla ve tvaru $\alpha A \beta \to \alpha \gamma \beta$
  - s jediným výjimkou je pravidlo $S \to \varepsilon$, přičemž se zde S nesmí vyskytovat na pravé straně žádného pravidla

- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)

- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál

- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)
  - pouze pravidla ve tvaru $\gamma A \beta \to \gamma \omega \beta$
  - s jedinou výjimkou je pravidlo $S \to \varepsilon$, přičemž se zde S nesmí vyskytovat na pravé straně žádného pravidla

- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)

- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál
- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)
  - pouze pravidla ve tvaru $\gamma A \beta \to \gamma \omega \beta$

    $A \in V, \gamma, \beta \in (V \cup T)^*, \omega \in (V \cup T)^+$
  - jedinou výjimkou je pravidlo $S \to \epsilon$, potom se ale S nevyskytuje na pravé straně žádného pravidla
- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)
- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál
- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)
  - pouze pravidla ve tvaru $\gamma A \beta \to \gamma \omega \beta$

    $A \in V, \gamma, \beta \in (V \cup T)^*, \omega \in (V \cup T)^+$!
  - jedinou výjimkou je pravidlo $S \to \epsilon$, potom se ale S nevyskytuje na pravé straně žádného pravidla
- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)
- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál
- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)
  - pouze pravidla ve tvaru $\gamma A \beta \to \gamma \omega \beta$

    $A \in V, \gamma, \beta \in (V \cup T)^*, \omega \in (V \cup T)^+$**!**
  - jedinou výjimkou je pravidlo $S \to \epsilon$, potom se ale S nevyskytuje na pravé straně žádného pravidla
- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)

  pouze pravidla ve tvaru $A \to \omega$, $A \in V, \omega \in (V \cup T)^*$
- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál
- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)
  - pouze pravidla ve tvaru $\gamma A \beta \to \gamma \omega \beta$

    $A \in V, \gamma, \beta \in (V \cup T)^*, \omega \in (V \cup T)^+$**!**
  - jedinou výjimkou je pravidlo $S \to \epsilon$, potom se ale S nevyskytuje na pravé straně žádného pravidla
- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)

  pouze pravidla ve tvaru $A \to \omega, A \in V, \omega \in (V \cup T)^*$
- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál
- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)
  - pouze pravidla ve tvaru $\gamma A \beta \to \gamma \omega \beta$

    $A \in V, \gamma, \beta \in (V \cup T)^*, \omega \in (V \cup T)^+$**!**
  - jedinou výjimkou je pravidlo $S \to \epsilon$, potom se ale S nevyskytuje na pravé straně žádného pravidla
- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)

  pouze pravidla ve tvaru $A \to \omega, A \in V, \omega \in (V \cup T)^*$
- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

  pouze pravidla ve tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál
- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)
  - pouze pravidla ve tvaru $\gamma A \beta \to \gamma \omega \beta$

    $A \in V, \gamma, \beta \in (V \cup T)^*, \omega \in (V \cup T)^+$**!**
  - jedinou výjimkou je pravidlo $S \to \epsilon$, potom se ale S nevyskytuje na pravé straně žádného pravidla
- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)

  pouze pravidla ve tvaru $A \to \omega, A \in V, \omega \in (V \cup T)^*$
- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

  pouze pravidla ve tvaru $A \to \omega B, A \to \omega, A, B \in V, \omega \in T^*$

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál
- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)
  - pouze pravidla ve tvaru $\gamma A \beta \to \gamma \omega \beta$

    $A \in V, \gamma, \beta \in (V \cup T)^*, \omega \in (V \cup T)^+$**!**
  - jedinou výjimkou je pravidlo $S \to \epsilon$, potom se ale S nevyskytuje na pravé straně žádného pravidla
- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)

  pouze pravidla ve tvaru $A \to \omega, A \in V, \omega \in (V \cup T)^*$
- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

  pouze pravidla ve tvaru $A \to \omega B, A \to \omega, A, B \in V, \omega \in T^*$.

Chomského hierarchie

Definition 6.2 (Klasifikace gramatik podle tvaru přepisovacích pravidel)

- **gramatiky typu 0** (**rekurzivně spočetné jazyky** $\mathcal{L}_0$)

  pravidla v obecné formě $\alpha \to \omega$, $\alpha, \omega \in (V \cup T)^*, \alpha$ obsahuje neterminál
- **gramatiky typu 1** (**kontextové gramatiky**, jazyky $\mathcal{L}_1$)
  - pouze pravidla ve tvaru $\gamma A \beta \to \gamma \omega \beta$

    $A \in V, \gamma, \beta \in (V \cup T)^*, \omega \in (V \cup T)^+$**!**
  - jedinou výjimkou je pravidlo $S \to \epsilon$, potom se ale S nevyskytuje na pravé straně žádného pravidla
- **gramatiky typu 2** (**bezkontextové gramatiky**, jazyky $\mathcal{L}_2$)

  pouze pravidla ve tvaru $A \to \omega, A \in V, \omega \in (V \cup T)^*$
- **gramatiky typu 3** (regulární/**pravé lineární gramatiky**, regulární jazyky $\mathcal{L}_3$)

  pouze pravidla ve tvaru $A \to \omega B, A \to \omega, A, B \in V, \omega \in T^*$.

Uspořádanost Chomského hierarchie

- Chomského hierarchie definuje uspořádání tříd jazyků

$$\mathcal{L}_0 \supseteq \mathcal{L}_1 \supseteq \mathcal{L}_2 \supseteq \mathcal{L}_3$$

- dokonce vlastní podmnožiny (později)

$\mathcal{L}_0 \supseteq \mathcal{L}_1$ rekurzivně spočetné jazyky zahrnují kontextové jazyky
pravidla $\gamma A \beta \to \gamma \omega \beta$ obsahují vlevo neterminál A

$\mathcal{L}_2 \supseteq \mathcal{L}_3$ bezkontextové jazyky zahrnují regulární jazyky
pravidla $A \to \omega B, A \to \omega$ obsahují vpravo řetězec $(V \cup T)^*$

$\mathcal{L}_1 \supseteq \mathcal{L}_2$ kontextové jazyky zahrnují bezkontextové jazyky
problém je s pravidly typu $A \to \epsilon$, ale ta umíme eliminovat.

Example 6.2 (Notace)

$a, b, c, 1, *, ($	terminály
A, B, C	neterminály, proměnné
w, z	řetězec terminálů
X, Y	buď terminál nebo neterminál
α, β, γ	řetězec $(T \cup V)^*$
$A \to \alpha \mid \beta$	$\{A \to \alpha, A \to \beta\}$, OR, kompaktní zápis více pravidel.

Uspořádanost Chomského hierarchie

- Chomského hierarchie definuje uspořádání tříd jazyků

$$\mathcal{L}_0 \supseteq \mathcal{L}_1 \supseteq \mathcal{L}_2 \supseteq \mathcal{L}_3$$

- dokonce vlastní podmnožiny (později)
  $\mathcal{L}_0 \supseteq \mathcal{L}_1$ rekurzivně spočetné jazyky zahrnují kontextové jazyky
  pravidla $\gamma A \beta \to \gamma \omega \beta$ obsahují vlevo neterminál A
  $\mathcal{L}_2 \supseteq \mathcal{L}_3$ bezkontextové jazyky zahrnují regulární jazyky
  pravidla $A \to \omega B, A \to \omega$ obsahují vpravo řetězec $(V \cup T)^*$
  $\mathcal{L}_1 \supseteq \mathcal{L}_2$ kontextové jazyky zahrnují bezkontextové jazyky
  problém je s pravidly typu $A \to \epsilon$, ale ta umíme eliminovat.

Example 6.2 (Notace)

$a, b, c, 1, *, ($	terminály
A, B, C	neterminály, proměnné
w, z	řetězec terminálů
X, Y	buď terminál nebo neterminál
α, β, γ	řetězec $(T \cup V)^*$
$A \to \alpha \mid \beta$	$\{A \to \alpha, A \to \beta\}$, OR, kompaktní zápis více pravidel.

Uspořádanost Chomského hierarchie

- Chomského hierarchie definuje uspořádání tříd jazyků

$$\mathcal{L}_0 \supseteq \mathcal{L}_1 \supseteq \mathcal{L}_2 \supseteq \mathcal{L}_3$$

- dokonce vlastní podmnožiny (později)
  $\mathcal{L}_0 \supseteq \mathcal{L}_1$ rekurzivně spočetné jazyky zahrnují kontextové jazyky
  pravidla $\gamma A \beta \to \gamma \omega \beta$ obsahují vlevo neterminál A
  $\mathcal{L}_2 \supseteq \mathcal{L}_3$ bezkontextové jazyky zahrnují regulární jazyky
  pravidla $A \to \omega B, A \to \omega$ obsahují vpravo řetězec $(V \cup T)^*$
  $\mathcal{L}_1 \supseteq \mathcal{L}_2$ kontextové jazyky zahrnují bezkontextové jazyky
  problém je s pravidly typu $A \to \epsilon$, ale ta umíme eliminovat.

Example 6.2 (Notace)

$a, b, c, 1, *, ($	terminály
A, B, C	neterminály, proměnné
w, z	řetězec terminálů
X, Y	buď terminál nebo neterminál
α, β, γ	řetězec $(T \cup V)^*$
$A \to \alpha \mid \beta$	$\{A \to \alpha, A \to \beta\}$, OR, kompaktní zápis více pravidel.

Uspořádanost Chomského hierarchie

- Chomského hierarchie definuje uspořádání tříd jazyků

$$\mathcal{L}_0 \supseteq \mathcal{L}_1 \supseteq \mathcal{L}_2 \supseteq \mathcal{L}_3$$

- dokonce vlastní podmnožiny (později)
  $\mathcal{L}_0 \supseteq \mathcal{L}_1$ rekurzivně spočetné jazyky zahrnují kontextové jazyky
  pravidla $\gamma A \beta \to \gamma \omega \beta$ obsahují vlevo neterminál A
  $\mathcal{L}_2 \supseteq \mathcal{L}_3$ bezkontextové jazyky zahrnují regulární jazyky
  pravidla $A \to \omega B, A \to \omega$ obsahují vpravo řetězec $(V \cup T)^*$
  $\mathcal{L}_1 \supseteq \mathcal{L}_2$ kontextové jazyky zahrnují bezkontextové jazyky
  problém je s pravidly typu $A \to \epsilon$, ale ta umíme eliminovat.

Example 6.2 (Notace)

$a, b, c, 1, *, ($	terminály
A, B, C	neterminály, proměnné
w, z	řetězec terminálů
X, Y	buď terminál nebo neterminál
α, β, γ	řetězec $(T \cup V)^*$
$A \to \alpha \mid \beta$	$\{A \to \alpha, A \to \beta\}$, OR, kompaktní zápis více pravidel.

Uspořádanost Chomského hierarchie

- Chomského hierarchie definuje uspořádání tříd jazyků

$$\mathcal{L}_0 \supseteq \mathcal{L}_1 \supseteq \mathcal{L}_2 \supseteq \mathcal{L}_3$$

- dokonce vlastní podmnožiny (později)
  $\mathcal{L}_0 \supseteq \mathcal{L}_1$ rekurzivně spočetné jazyky zahrnují kontextové jazyky
  pravidla $\gamma A \beta \to \gamma \omega \beta$ obsahují vlevo neterminál A
  $\mathcal{L}_2 \supseteq \mathcal{L}_3$ bezkontextové jazyky zahrnují regulární jazyky
  pravidla $A \to \omega B, A \to \omega$ obsahují vpravo řetězec $(V \cup T)^*$
  $\mathcal{L}_1 \supseteq \mathcal{L}_2$ kontextové jazyky zahrnují bezkontextové jazyky
  problém je s pravidly typu $A \to \epsilon$, ale ta umíme eliminovat.

Example 6.2 (Notace)

$a, b, c, 1, *, ($	terminály
A, B, C	neterminály, proměnné
w, z	řetězec terminálů
X, Y	buď terminál nebo neterminál
α, β, γ	řetězec $(T \cup V)^*$
$A \to \alpha \mid \beta$	$\{A \to \alpha, A \to \beta\}$, OR, kompaktní zápis více pravidel.

Definition 6.3 (Derivace $\Rightarrow^*$)

Mějme gramatiku $G = (V, T, P, S)$.

- Říkáme, že α se **přímo přepíše** na ω (píšeme $\alpha \Rightarrow_G \omega$ nebo $\alpha \Rightarrow \omega$) jestliže
  $\exists \beta, \gamma, \eta, \nu \in (V \cup T)^* : \alpha = \eta\beta\nu$, $\omega = \eta\gamma\nu$ a $(\beta \to \gamma) \in P$.
- Říkáme, že α se **přepíše** na ω (píšeme $\alpha \Rightarrow^* \omega$) jestliže
  $\exists \beta_1, \ldots, \beta_n \in (V \cup T)^* : \alpha = \beta_1 \Rightarrow \beta_2 \Rightarrow \ldots \Rightarrow \beta_n = \omega$,
  tj. také $\alpha \Rightarrow^* \alpha$.
- Posloupnost $\beta_1, \ldots, \beta_n$ nazýváme **derivací (odvozením)**.
- Pokud $\forall i \neq j : \beta_i \neq \beta_j$, hovoříme o **minimálním odvození**.
- Libovolný řetězec $\omega \in (T \cup V)^*$ odvoditelný z počátečního symbolu nazýváme **sentenciální forma**.

Definition 6.4 (Jazyk generovaný gramatikou G)

Jazyk $L(G)$ generovaný gramatikou $G = (V, T, P, S)$ je množina terminálních řetězců, pro které existuje derivace ze startovního symbolu

$$L(G) = \{w \in T^* | S \Rightarrow_G^* w\}.$$

Jazyk neterminálu $A \in V$ definujeme $L(A) = \{w \in T^* | A \Rightarrow_G^* w\}$.

Definition 6.3 (Derivace $\Rightarrow^*$)

Mějme gramatiku $G = (V, T, P, S)$.

- Říkáme, že α se **přímo přepíše** na ω (píšeme $\alpha \Rightarrow_G \omega$ nebo $\alpha \Rightarrow \omega$) jestliže
  $$\exists \beta, \gamma, \eta, \nu \in (V \cup T)^* : \alpha = \eta\beta\nu,\ \omega = \eta\gamma\nu \text{ a } (\beta \to \gamma) \in P.$$
- Říkáme, že α se **přepíše** na ω (píšeme $\alpha \Rightarrow^* \omega$) jestliže
  $$\exists \beta_1, \ldots, \beta_n \in (V \cup T)^* : \alpha = \beta_1 \Rightarrow \beta_2 \Rightarrow \ldots \Rightarrow \beta_n = \omega,$$
  tj. také $\alpha \Rightarrow^* \alpha$.
- Posloupnost $\beta_1, \ldots, \beta_n$ nazýváme **derivací (odvozením)**.
- Pokud $\forall i \neq j : \beta_i \neq \beta_j$, hovoříme o **minimálním odvození**.
- Libovolný řetězec $\omega \in (T \cup V)^*$ odvoditelný z počátečního symbolu nazýváme **sentenciální forma**.

Definition 6.4 (Jazyk generovaný gramatikou G)

Jazyk $L(G)$ generovaný gramatikou $G = (V, T, P, S)$ je množina terminálních řetězců, pro které existuje derivace ze startovního symbolu

$$L(G) = \{w \in T^* | S \Rightarrow_G^* w\}.$$

Jazyk neterminálu $A \in V$ definujeme $L(A) = \{w \in T^* | A \Rightarrow_G^* w\}$.

Gramatiky typu 3 a regulární jazyky

Definition (Gramatika typu 3, pravá lineární)

Gramatika G je **pravá lineární, tj. regulární, Typu 3**, pokud obsahuje pouze pravidla tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$.

Example 6.3 (Příklad derivace gramatiky typu 3)

$$P = \{S \to 0S|1A|\epsilon,\ A \to 0A|1B,\ B \to 0B|1S\}$$

$$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$$

- Pozorování:
  - každá sentenciální forma derivace obsahuje právě jeden neterminál
  - tento neterminál je vždy umístěn zcela vpravo
  - aplikací pravidla $A \to \epsilon$ je tato derivace ukončena
  - krok derivace generuje symboly a změní neterminál
- Idea vztahu gramatiky a konečného automatu
- neterminál = stav konečného automatu
- pravidla = přechodová funkce.

Gramatiky typu 3 a regulární jazyky

Definition (Gramatika typu 3, pravá lineární)

Gramatika G je **pravá lineární, tj. regulární, Typu 3**, pokud obsahuje pouze pravidla tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$.

Example 6.3 (Příklad derivace gramatiky typu 3)

$$P = \{S \to 0S|1A|\epsilon,\ A \to 0A|1B,\ B \to 0B|1S\}$$

$$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$$

- Pozorování:
  - každá sentenciální forma derivace obsahuje právě jeden neterminál
  - tento neterminál je vždy umístěn zcela vpravo
  - aplikací pravidla $A \to w$ se derivace uzavírá
  - krok derivace generuje symboly a změní neterminál
- Idea vztahu gramatiky a konečného automatu
- neterminál = stav konečného automatu
- pravidla = přechodová funkce.

Gramatiky typu 3 a regulární jazyky

Definition (Gramatika typu 3, pravá lineární)

Gramatika G je **pravá lineární, tj. regulární, Typu 3**, pokud obsahuje pouze pravidla tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$.

Example 6.3 (Příklad derivace gramatiky typu 3)

$$P = \{S \to 0S|1A|\epsilon,\ A \to 0A|1B,\ B \to 0B|1S\}$$

$$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$$

- Pozorování:
  - každá sentenciální forma derivace obsahuje právě jeden neterminál
  - tento neterminál je vždy umístěn zcela vpravo
  - aplikací pravidla $A \to w$ se derivace uzavírá
  - krok derivace generuje symboly a změní neterminál
- Idea vztahu gramatiky a konečného automatu
- neterminál = stav konečného automatu
- pravidla = přechodová funkce.

Gramatiky typu 3 a regulární jazyky

Definition (Gramatika typu 3, pravá lineární)

Gramatika G je **pravá lineární, tj. regulární, Typu 3**, pokud obsahuje pouze pravidla tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$.

Example 6.3 (Příklad derivace gramatiky typu 3)

$$P = \{S \to 0S|1A|\epsilon,\ A \to 0A|1B,\ B \to 0B|1S\}$$

$$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$$

- Pozorování:
  - každá sentenciální forma derivace obsahuje právě jeden neterminál
  - tento neterminál je vždy umístěn zcela vpravo
  - aplikací pravidla $A \to w$ se derivace uzavírá
  - krok derivace generuje symboly a změní neterminál
- Idea vztahu gramatiky a konečného automatu
- neterminál = stav konečného automatu
- pravidla = přechodová funkce.

Gramatiky typu 3 a regulární jazyky

Definition (Gramatika typu 3, pravá lineární)

Gramatika G je **pravá lineární, tj. regulární, Typu 3**, pokud obsahuje pouze pravidla tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$.

Example 6.3 (Příklad derivace gramatiky typu 3)

$$P = \{S \to 0S|1A|\epsilon,\ A \to 0A|1B,\ B \to 0B|1S\}$$

$$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$$

- Pozorování:
  - každá sentenciální forma derivace obsahuje právě jeden neterminál
  - tento neterminál je vždy umístěn zcela vpravo
  - aplikací pravidla $A \to w$ se derivace uzavírá
  - krok derivace generuje symboly a změní neterminál
- Idea vztahu gramatiky a konečného automatu
- neterminál = stav konečného automatu
- pravidla = přechodová funkce.

Gramatiky typu 3 a regulární jazyky

Definition (Gramatika typu 3, pravá lineární)

Gramatika G je **pravá lineární, tj. regulární, Typu 3**, pokud obsahuje pouze pravidla tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$.

Example 6.3 (Příklad derivace gramatiky typu 3)

$$P = \{S \to 0S|1A|\epsilon,\ A \to 0A|1B,\ B \to 0B|1S\}$$

$$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$$

- Pozorování:
  - každá sentenciální forma derivace obsahuje právě jeden neterminál
  - tento neterminál je vždy umístěn zcela vpravo
  - aplikací pravidla $A \to w$ se derivace uzavírá
  - krok derivace generuje symboly a změní neterminál
- Idea vztahu gramatiky a konečného automatu
- neterminál = stav konečného automatu
- pravidla = přechodová funkce.

Gramatiky typu 3 a regulární jazyky

Definition (Gramatika typu 3, pravá lineární)

Gramatika G je **pravá lineární, tj. regulární, Typu 3**, pokud obsahuje pouze pravidla tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$.

Example 6.3 (Příklad derivace gramatiky typu 3)

$$P = \{S \to 0S|1A|\epsilon,\ A \to 0A|1B,\ B \to 0B|1S\}$$

$$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$$

- Pozorování:
  - každá sentenciální forma derivace obsahuje právě jeden neterminál
  - tento neterminál je vždy umístěn zcela vpravo
  - aplikací pravidla $A \to w$ se derivace uzavírá
  - krok derivace generuje symboly a změní neterminál
- Idea vztahu gramatiky a konečného automatu
- neterminál = stav konečného automatu
- pravidla = přechodová funkce.

Gramatiky typu 3 a regulární jazyky

Definition (Gramatika typu 3, pravá lineární)

Gramatika G je **pravá lineární, tj. regulární, Typu 3**, pokud obsahuje pouze pravidla tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$.

Example 6.3 (Příklad derivace gramatiky typu 3)

$$P = \{S \to 0S|1A|\epsilon,\ A \to 0A|1B,\ B \to 0B|1S\}$$

$$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$$

- Pozorování:
  - každá sentenciální forma derivace obsahuje právě jeden neterminál
  - tento neterminál je vždy umístěn zcela vpravo
  - aplikací pravidla $A \to w$ se derivace uzavírá
  - krok derivace generuje symboly a změní neterminál
- Idea vztahu gramatiky a konečného automatu
- neterminál = stav konečného automatu
- pravidla = přechodová funkce.

Gramatiky typu 3 a regulární jazyky

Definition (Gramatika typu 3, pravá lineární)

Gramatika G je **pravá lineární, tj. regulární, Typu 3**, pokud obsahuje pouze pravidla tvaru $A \to wB, A \to w, A, B \in V, w \in T^*$.

Example 6.3 (Příklad derivace gramatiky typu 3)

$$P = \{S \to 0S|1A|\epsilon,\ A \to 0A|1B,\ B \to 0B|1S\}$$

$$S \Rightarrow 0S \Rightarrow 01A \Rightarrow 011B \Rightarrow 0110B \Rightarrow 01101S \Rightarrow 01101$$

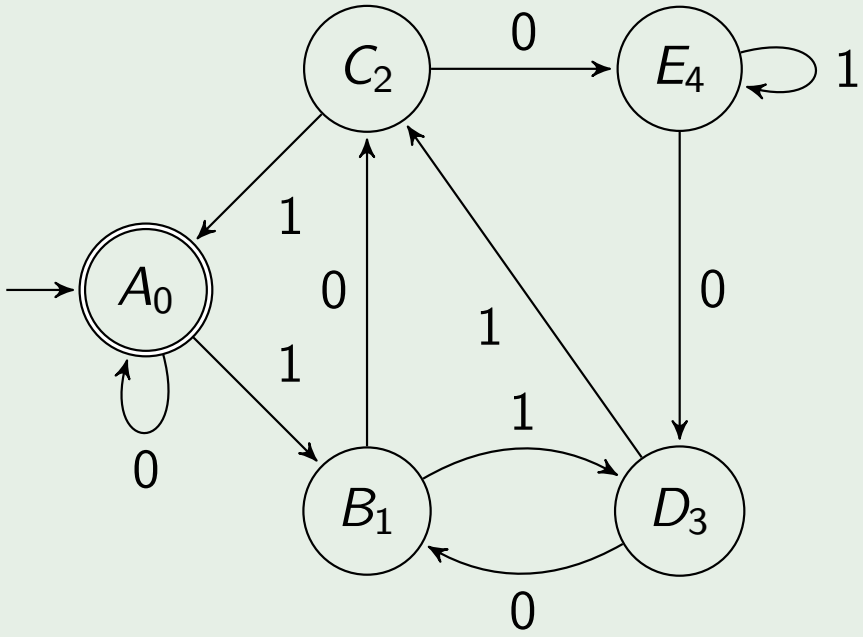
- Pozorování:
  - každá sentenciální forma derivace obsahuje právě jeden neterminál
  - tento neterminál je vždy umístěn zcela vpravo
  - aplikací pravidla $A \to w$ se derivace uzavírá
  - krok derivace generuje symboly a změní neterminál
- Idea vztahu gramatiky a konečného automatu
- neterminál = stav konečného automatu
- pravidla = přechodová funkce.

Příklad převodu FA na gramatiku

Example 6.4 (G, FA binární zápis čísla dělitelného 5)

$L = \{w | w \in \{0,1\}^* \& w$ je binární zápis čísla dělitelného 5$\}$

$$
\begin{aligned}
A &\rightarrow 1B|0A|\epsilon \\
B &\rightarrow 0C|1D \\
C &\rightarrow 0E|1A \\
D &\rightarrow 0B|1C \\
E &\rightarrow 0D|1E
\end{aligned}
$$



Příklady derivací

$$
\begin{aligned}
&A \Rightarrow 0A \Rightarrow 0 && (0)\\
&A \Rightarrow 1B \Rightarrow 10C \Rightarrow 101A \Rightarrow 101 && (5)\\
&A \Rightarrow 1B \Rightarrow 10C \Rightarrow 101A \Rightarrow 1010A \Rightarrow 1010 && (10)\\
&A \Rightarrow 1B \Rightarrow 11D \Rightarrow 111C \Rightarrow 1111A \Rightarrow 1111 && (15)
\end{aligned}
$$

Převod konečného automatu na gramatiku typu 3

Theorem 6.1 ($L \in RE \Rightarrow L \in \mathcal{L}_3$)

Pro každý jazyk rozpoznávaný konečným automatem existuje gramatika typu 3, která ho generuje.

Proof: Převod konečného automatu na gramatiku typu 3

- $L = L(A)$ pro deterministický konečný automat $A = (Q, \Sigma, \delta, q_0, F)$.
- definujme gramatiku $G = (Q, \Sigma, P, q_0)$, kde pravidla P mají tvar
  $p \to aq$, když $\delta(p, a) = q$
  $p \to \epsilon$, když $p \in F$
- je $L(A) = L(G)$?
  - $w \in L(A) \Leftrightarrow q_0 = A \Rightarrow \ldots \Rightarrow wA \Rightarrow w$
  - $a_1 \ldots a_n \in L(A) \Leftrightarrow \exists q_1, \ldots, q_n \in Q$ tak, že $\delta(q_0, a_1) = q_1, \ldots, \delta(q_{n-1}, a_n) = q_n \in F$
    $\Leftrightarrow (q_0 \Rightarrow a_1 q_1 \Rightarrow a_1 a_2 q_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_n q_n \Rightarrow a_1 \ldots a_n)$ je derivace pro $a_1 \ldots a_n$
    $\Leftrightarrow a_1 \ldots a_n \in L(G)$

□

Převod konečného automatu na gramatiku typu 3

Theorem 6.1 ($L \in RE \Rightarrow L \in \mathcal{L}_3$)

Pro každý jazyk rozpoznávaný konečným automatem existuje gramatika typu 3, která ho generuje.

Proof: Převod konečného automatu na gramatiku typu 3

- $L = L(A)$ pro deterministický konečný automat $A = (Q, \Sigma, \delta, q_0, F)$.
- definujme gramatiku $G = (Q, \Sigma, P, q_0)$, kde pravidla P mají tvar
  $p \to aq$, když $\delta(p, a) = q$
  $p \to \epsilon$, když $p \in F$
- je $L(A) = L(G)$?

□

Převod konečného automatu na gramatiku typu 3

Theorem 6.1 ($L \in RE \Rightarrow L \in \mathcal{L}_3$)

Pro každý jazyk rozpoznávaný konečným automatem existuje gramatika typu 3, která ho generuje.

Proof: Převod konečného automatu na gramatiku typu 3

- $L = L(A)$ pro deterministický konečný automat $A = (Q, \Sigma, \delta, q_0, F)$.
- definujme gramatiku $G = (Q, \Sigma, P, q_0)$, kde pravidla P mají tvar
  $p \to aq$, když $\delta(p, a) = q$
  $p \to \epsilon$, když $p \in F$
- je $L(A) = L(G)$?
  - $\epsilon \in L(A) \Leftrightarrow q_0 \in F \Leftrightarrow (q_0 \to \epsilon) \in P \Leftrightarrow \epsilon \in L(G)$
  - $a_1 \dots a_n \in L(A) \Leftrightarrow \exists q_0, \dots, q_n \in Q$ tž. $\delta(q_i, a_{i+1}) = q_{i+1}, q_n \in F$
    $\Leftrightarrow (q_0 \Rightarrow a_1 q_1 \Rightarrow \dots a_1 \dots a_n q_n \Rightarrow a_1 \dots a_n)$ je derivace pro $a_1 \dots a_n$
    $\Leftrightarrow a_1 \dots a_n \in L(G)$

□

Převod konečného automatu na gramatiku typu 3

Theorem 6.1 ($L \in RE \Rightarrow L \in \mathcal{L}_3$)

Pro každý jazyk rozpoznávaný konečným automatem existuje gramatika typu 3, která ho generuje.

Proof: Převod konečného automatu na gramatiku typu 3

- $L = L(A)$ pro deterministický konečný automat $A = (Q, \Sigma, \delta, q_0, F)$.
- definujme gramatiku $G = (Q, \Sigma, P, q_0)$, kde pravidla P mají tvar
  $p \to aq$, když $\delta(p, a) = q$
  $p \to \epsilon$, když $p \in F$
- je $L(A) = L(G)$?
  - $\epsilon \in L(A) \Leftrightarrow q_0 \in F \Leftrightarrow (q_0 \to \epsilon) \in P \Leftrightarrow \epsilon \in L(G)$
  - $a_1 \dots a_n \in L(A) \Leftrightarrow \exists q_0, \dots, q_n \in Q$ tž. $\delta(q_i, a_{i+1}) = q_{i+1}, q_n \in F$
    $\Leftrightarrow (q_0 \Rightarrow a_1 q_1 \Rightarrow \dots a_1 \dots a_n q_n \Rightarrow a_1 \dots a_n)$ je derivace pro $a_1 \dots a_n$
    $\Leftrightarrow a_1 \dots a_n \in L(G)$

□

Převod konečného automatu na gramatiku typu 3

Theorem 6.1 ($L \in RE \Rightarrow L \in \mathcal{L}_3$)

Pro každý jazyk rozpoznávaný konečným automatem existuje gramatika typu 3, která ho generuje.

Proof: Převod konečného automatu na gramatiku typu 3

- $L = L(A)$ pro deterministický konečný automat $A = (Q, \Sigma, \delta, q_0, F)$.
- definujme gramatiku $G = (Q, \Sigma, P, q_0)$, kde pravidla P mají tvar
  $p \to aq$, když $\delta(p, a) = q$
  $p \to \epsilon$, když $p \in F$
- je $L(A) = L(G)$?
  - $\epsilon \in L(A) \Leftrightarrow q_0 \in F \Leftrightarrow (q_0 \to \epsilon) \in P \Leftrightarrow \epsilon \in L(G)$
  - $a_1 \dots a_n \in L(A) \Leftrightarrow \exists q_0, \dots, q_n \in Q$ tž. $\delta(q_i, a_{i+1}) = q_{i+1}, q_n \in F$
    $\Leftrightarrow (q_0 \Rightarrow a_1 q_1 \Rightarrow \dots a_1 \dots a_n q_n \Rightarrow a_1 \dots a_n)$ je derivace pro $a_1 \dots a_n$
    $\Leftrightarrow a_1 \dots a_n \in L(G)$

Převod konečného automatu na gramatiku typu 3

Theorem 6.1 ($L \in RE \Rightarrow L \in \mathcal{L}_3$)

Pro každý jazyk rozpoznávaný konečným automatem existuje gramatika typu 3, která ho generuje.

Proof: Převod konečného automatu na gramatiku typu 3

- $L = L(A)$ pro deterministický konečný automat $A = (Q, \Sigma, \delta, q_0, F)$.
- definujme gramatiku $G = (Q, \Sigma, P, q_0)$, kde pravidla P mají tvar
  $p \to aq$, když $\delta(p, a) = q$
  $p \to \epsilon$, když $p \in F$
- je $L(A) = L(G)$?
  - $\epsilon \in L(A) \Leftrightarrow q_0 \in F \Leftrightarrow (q_0 \to \epsilon) \in P \Leftrightarrow \epsilon \in L(G)$
  - $a_1 \dots a_n \in L(A) \Leftrightarrow \exists q_0, \dots, q_n \in Q$ tž. $\delta(q_i, a_{i+1}) = q_{i+1}, q_n \in F$
    $\Leftrightarrow (q_0 \Rightarrow a_1 q_1 \Rightarrow \dots a_1 \dots a_n q_n \Rightarrow a_1 \dots a_n)$ je derivace pro $a_1 \dots a_n$
    $\Leftrightarrow a_1 \dots a_n \in L(G)$

Převod konečného automatu na gramatiku typu 3

Theorem 6.1 ($L \in RE \Rightarrow L \in \mathcal{L}_3$)

Pro každý jazyk rozpoznávaný konečným automatem existuje gramatika typu 3, která ho generuje.

Proof: Převod konečného automatu na gramatiku typu 3

- $L = L(A)$ pro deterministický konečný automat $A = (Q, \Sigma, \delta, q_0, F)$.
- definujme gramatiku $G = (Q, \Sigma, P, q_0)$, kde pravidla P mají tvar
  $p \to aq$, když $\delta(p, a) = q$
  $p \to \epsilon$, když $p \in F$
- je $L(A) = L(G)$?
  - $\epsilon \in L(A) \Leftrightarrow q_0 \in F \Leftrightarrow (q_0 \to \epsilon) \in P \Leftrightarrow \epsilon \in L(G)$
  - $a_1 \dots a_n \in L(A) \Leftrightarrow \exists q_0, \dots, q_n \in Q$ tž. $\delta(q_i, a_{i+1}) = q_{i+1}, q_n \in F$
    $\Leftrightarrow (q_0 \Rightarrow a_1 q_1 \Rightarrow \dots a_1 \dots a_n q_n \Rightarrow a_1 \dots a_n)$ je derivace pro $a_1 \dots a_n$
    $\Leftrightarrow a_1 \dots a_n \in L(G)$

Příprava převodu gramatiky typu 3 na FA

- Opačný směr
  - pravidla $A \to aB$ kódujeme do přechodové funkce
  - pravidla $A \to \epsilon$ určují koncové stavy
  - pravidla $A \to a_1 \dots a_n B, A \to a_1 \dots a_n$ s více neterminály rozepíšeme
    - zavedeme nové neterminály $Y_2, \dots, Y_n, Z_1, \dots, Z_n$
    - vytvoříme pravidla $A \to a_1 Y_2, Y_2 \to a_2 Y_3, \dots, Y_n \to a_n B$
    - resp. $Z \to a_1 Z_1, Z_1 \to a_2 Z_2, \dots, Z_{n-1} \to a_n Z_n, Z_n \to \epsilon$
  - pravidla $A \to B$ odpovídají ϵ přechodům
    - zbavíme se jich tranzitivním uzávěrem
    - nebo musíme tranzitivně uzavřít $S \to B$ pro hledání $S \to \epsilon$.

Lemma

Ke každé gramatice typu 3 existuje gramatika typu 3, která generuje stejný jazyk a obsahuje pouze pravidla ve tvaru: $A \to aB, A \to \epsilon$, $A, B \in V, a \in T$.

Příprava převodu gramatiky typu 3 na FA

- Opačný směr
  - pravidla $A \to aB$ kódujeme do přechodové funkce
  - pravidla $A \to \epsilon$ určují koncové stavy
  - pravidla $A \to a_1 \dots a_n B, A \to a_1 \dots a_n$ s více neterminály rozepíšeme
    - zavedeme nové neterminály $Y_2, \dots, Y_n, Z_1, \dots, Z_n$
    - vytvoříme pravidla $A \to a_1 Y_2, Y_2 \to a_2 Y_3, \dots, Y_n \to a_n B$
    - resp. $Z \to a_1 Z_1, Z_1 \to a_2 Z_2, \dots, Z_{n-1} \to a_n Z_n, Z_n \to \epsilon$
  - pravidla $A \to B$ odpovídají ϵ přechodům
    - zbavíme se jich tranzitivním uzávěrem
    - nebo musíme tranzitivně uzavřít $S \to B$ pro hledání $S \to \epsilon$.

Lemma

Ke každé gramatice typu 3 existuje gramatika typu 3, která generuje stejný jazyk a obsahuje pouze pravidla ve tvaru: $A \to aB, A \to \epsilon$, $A, B \in V, a \in T$.

Příprava převodu gramatiky typu 3 na FA

- Opačný směr
  - pravidla $A \to aB$ kódujeme do přechodové funkce
  - pravidla $A \to \epsilon$ určují koncové stavy
  - pravidla $A \to a_1 \dots a_n B, A \to a_1 \dots a_n$ s více neterminály rozepíšeme
    - zavedeme nové neterminály $Y_2, \dots, Y_n, Z_1, \dots, Z_n$
    - vytvoříme pravidla $A \to a_1 Y_2, Y_2 \to a_2 Y_3, \dots, Y_n \to a_n B$
    - resp. $Z \to a_1 Z_1, Z_1 \to a_2 Z_2, \dots, Z_{n-1} \to a_n Z_n, Z_n \to \epsilon$
  - pravidla $A \to B$ odpovídají ϵ přechodům
    - zbavíme se jich tranzitivním uzávěrem
    - nebo musíme tranzitivně uzavřít $S \to B$ pro hledání $S \to \epsilon$.

Lemma

Ke každé gramatice typu 3 existuje gramatika typu 3, která generuje stejný jazyk a obsahuje pouze pravidla ve tvaru: $A \to aB, A \to \epsilon$, $A, B \in V, a \in T$.

Standardizace gramatiky typu 3

Lemma

Ke každé gramatice typu 3 existuje gramatika typu 3, která generuje stejný jazyk a obsahuje pouze pravidla ve tvaru: $A \to aB, A \to \epsilon$, $A, B \in V, a \in T$.

Proof.

Pro gramatiku $G = (V, T, S, P)$ definujeme $G^| = (V^|, T, S, P^|)$, kde pro každé pravidlo zavedeme dostatečný počet nových neterminálů $Y_2, \ldots, Y_n, Z_1, \ldots, Z_n$ a definujeme

P	$P^|$
$A \to aB$ $A \to \epsilon$ $A \to a_1 \ldots a_n B$ $Z \to a_1 \ldots a_n$	$A \to aB$ $A \to \epsilon$ $A \to a_1 Y_2, Y_2 \to a_2 Y_3, \ldots Y_n \to a_n B$ $Z \to a_1 Z_1, Z_1 \to a_2 Z_2, \ldots, Z_{n-1} \to a_n Z_n, Z_n \to \epsilon$
odstraníme i pravidla: $A \to B$	tranzitivní uzávěr $U(A) = \{B \mid B \in V \& A \Rightarrow^* B\}$ $A \to \gamma$ pro všechna $Z \in U(A)$ a $(Z \to \gamma) \in P^|$

□

Standardizace gramatiky typu 3

Lemma

Ke každé gramatice typu 3 existuje gramatika typu 3, která generuje stejný jazyk a obsahuje pouze pravidla ve tvaru: $A \to aB, A \to \epsilon$, $A, B \in V, a \in T$.

Proof.

Pro gramatiku $G = (V, T, S, P)$ definujeme $G^| = (V^|, T, S, P^|)$, kde pro každé pravidlo zavedeme dostatečný počet nových neterminálů $Y_2, \ldots, Y_n, Z_1, \ldots, Z_n$ a definujeme

P	$P^|$
$A \to aB$ $A \to \epsilon$ $A \to a_1 \ldots a_n B$ $Z \to a_1 \ldots a_n$	$A \to aB$ $A \to \epsilon$ $A \to a_1 Y_2, Y_2 \to a_2 Y_3, \ldots Y_n \to a_n B$ $Z \to a_1 Z_1, Z_1 \to a_2 Z_2, \ldots, Z_{n-1} \to a_n Z_n, Z_n \to \epsilon$
odstraníme i pravidla: $A \to B$	tranzitivní uzávěr $U(A) = \{B \mid B \in V \& A \Rightarrow^* B\}$ $A \to \gamma$ pro všechna $Z \in U(A)$ a $(Z \to \gamma) \in P^|$

□

Pouze pravidla $A \to aB, A \to \epsilon$

Example 6.5

P	$P^{\mid}$
$B \to a_1$	$B \to a_1 H_1, H_1 \to \epsilon$
	$U(A) = \{A, B\}$, proto
$A \to B$	$A \to a_1 H_2, H_2 \to \epsilon$
$A \to a_2$	$A \to a_2 H_3, H_3 \to \epsilon$

Převod gramatiky typu 3 na konečný automat

Theorem 6.2 (ϵ–NFA pro gramatiku typu 3 rozpoznávající stejný jazyk)

Pro každý jazyk L generovaný gramatikou typu 3 existuje ϵ–NFA rozpoznávající L.

Proof: Převod gramatiky typu 3 na konečný automat

- Vezmeme $G = (V, T, P, S)$ obsahující jen pravidla tvaru $A \to aB$, $A \to \epsilon$, $A, B \in V, a \in T$ generující L (předchozí lemma)
- definujeme nedeterministický ϵ–NFA $A = (V, T, \delta, S, F)$, kde:
  - $F = \{A | (A \to \epsilon) \in P\}$
  - $\delta(A, a) = \{B | (A \to aB) \in P\}$
- $L(G) = L(A)$
  - $\epsilon \in L(G) \Rightarrow (S \to \epsilon) \in P \Rightarrow S \in F \Rightarrow \epsilon \in L(A)$
  - $a_1 \ldots a_n \in L(G) \Rightarrow$ existuje derivace $(S \to a_1 A_1 \to a_1 a_2 A_2 \to \ldots \to a_1 \ldots a_n A_n \to a_1 \ldots a_n)$
    $\Rightarrow \exists A_1, \ldots, A_n \in V$ tak že $A_1 = \delta(S, a_1), A_2 = \delta(A_1, a_2), \ldots, A_n \in \delta(A_{n-1}, a_n)$ a zároveň $A_n \in F$
    $\Rightarrow a_1 \ldots a_n \in L(A)$

□

Převod gramatiky typu 3 na konečný automat

Theorem 6.2 (ϵ–NFA pro gramatiku typu 3 rozpoznávající stejný jazyk)

Pro každý jazyk L generovaný gramatikou typu 3 existuje ϵ–NFA rozpoznávající L.

Proof: Převod gramatiky typu 3 na konečný automat

- Vezmeme $G = (V, T, P, S)$ obsahující jen pravidla tvaru $A \to aB$, $A \to \epsilon$, $A, B \in V, a \in T$ generující L (předchozí lemma)
- definujeme nedeterministický ϵ–NFA $A = (V, T, \delta, S, F)$, kde:

$$F = \{A | (A \to \epsilon) \in P\}$$
$$\delta(A, a) = \{B | (A \to aB) \in P\}$$

- $L(G) = L(A)$
  - $\epsilon \in L(G) \Leftrightarrow (S \to \epsilon) \in P \Leftrightarrow S \in F \Leftrightarrow \epsilon \in L(A)$
  - $a_1 \ldots a_n \in L(G) \Leftrightarrow$ existuje derivace $(S \Rightarrow a_1 B_1 \Rightarrow a_1 a_2 B_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{n-1} B_{n-1} \Rightarrow a_1 \ldots a_n)$
  - $\Leftrightarrow \exists B_1, \ldots, B_{n-1} \in V$ tak že $A_1 = S, B_1 \in A_1$, $B_2 \in \delta(B_1, a_2), \ldots, B_{n-1} \in \delta(B_{n-2}, a_{n-1})$ a $B_{n-1} \to a_n \in P$
  - $\Leftrightarrow a_1 \ldots a_n \in L(A)$

□

Převod gramatiky typu 3 na konečný automat

Theorem 6.2 (ϵ–NFA pro gramatiku typu 3 rozpoznávající stejný jazyk)

Pro každý jazyk L generovaný gramatikou typu 3 existuje ϵ–NFA rozpoznávající L.

Proof: Převod gramatiky typu 3 na konečný automat

- Vezmeme $G = (V, T, P, S)$ obsahující jen pravidla tvaru $A \to aB$, $A \to \epsilon$, $A, B \in V, a \in T$ generující L (předchozí lemma)
- definujeme nedeterministický ϵ–NFA $A = (V, T, \delta, S, F)$, kde:

  $F = \{A | (A \to \epsilon) \in P\}$

  $\delta(A, a) = \{B | (A \to aB) \in P\}$
- $L(G) = L(A)$
  - $\epsilon \in L(G) \Leftrightarrow (S \to \epsilon) \in P \Leftrightarrow S \in F \Leftrightarrow \epsilon \in L(A)$
  - $a_1 \ldots a_n \in L(G) \Leftrightarrow$ existuje derivace $(S \Rightarrow a_1 B_1 \Rightarrow a_1 a_2 B_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{n-1} B_{n-1} \Rightarrow a_1 \ldots a_n)$
  $\Leftrightarrow \exists B_1, \ldots, B_{n-1} \in V$ tak že $A_1 = S, B_{n-1} \in A$, $B_i \in \delta(B_{i-1}, a_i)$ pro $i = 2, \ldots, n-1$, $a_1 \in \delta(A_1, a_1)$
  $\Leftrightarrow a_1 \ldots a_n \in L(A)$

□

Převod gramatiky typu 3 na konečný automat

Theorem 6.2 (ϵ–NFA pro gramatiku typu 3 rozpoznávající stejný jazyk)

Pro každý jazyk L generovaný gramatikou typu 3 existuje ϵ–NFA rozpoznávající L.

Proof: Převod gramatiky typu 3 na konečný automat

- Vezmeme $G = (V, T, P, S)$ obsahující jen pravidla tvaru $A \to aB$, $A \to \epsilon$, $A, B \in V, a \in T$ generující L (předchozí lemma)
- definujeme nedeterministický ϵ–NFA $A = (V, T, \delta, S, F)$, kde:

  $F = \{A | (A \to \epsilon) \in P\}$
  $\delta(A, a) = \{B | (A \to aB) \in P\}$
- $L(G) = L(A)$
  - $\epsilon \in L(G) \Rightarrow (S \to \epsilon) \in P \Rightarrow S \in F \Rightarrow \epsilon \in L(A)$
  - $a_1 \ldots a_n \in L(G) \Rightarrow$ existuje derivace $(S \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{n-1} A_{n-1} \Rightarrow a_1 \ldots a_n)$
  $\Rightarrow \exists A_1, \ldots, A_{n-1} \in V$ tak že $S \to a_1 A_1 \in P, A_1 \to a_2 A_2 \in P, \ldots, A_{n-1} \to a_n \in P$
  $A_1 \in \delta(A, a_1), A_2 \in \delta(A_1, a_2), \ldots, A_{n-1} \in \delta(A_{n-2}, a_{n-1}), a_n \in \delta(A_{n-1}, a_n)$
  $\Rightarrow a_1 \ldots a_n \in L(A)$

□

Převod gramatiky typu 3 na konečný automat

Theorem 6.2 (ϵ–NFA pro gramatiku typu 3 rozpoznávající stejný jazyk)

Pro každý jazyk L generovaný gramatikou typu 3 existuje ϵ–NFA rozpoznávající L.

Proof: Převod gramatiky typu 3 na konečný automat

- Vezmeme $G = (V, T, P, S)$ obsahující jen pravidla tvaru $A \to aB$, $A \to \epsilon$, $A, B \in V, a \in T$ generující L (předchozí lemma)
- definujeme nedeterministický ϵ–NFA $A = (V, T, \delta, S, F)$, kde:

  $F = \{A | (A \to \epsilon) \in P\}$
  $\delta(A, a) = \{B | (A \to aB) \in P\}$
- $L(G) = L(A)$
  - $\epsilon \in L(G) \Leftrightarrow (S \to \epsilon) \in P \Leftrightarrow S \in F \Leftrightarrow \epsilon \in L(A)$
  - $a_1 \dots a_n \in L(G) \Leftrightarrow$ existuje derivace
    $(S \Rightarrow a_1 H_1 \Rightarrow \dots \Rightarrow a_1 \dots a_n H_n \Rightarrow a_1 \dots a_n)$
    $\Leftrightarrow \exists H_0, \dots, H_n \in V$ tak že $H_0 = S, H_n \in F$
    $H_{i+1} \in \delta(H_i, a_k)$ pro krok $a_1 \dots a_{k-1} H_i \Rightarrow a_1 \dots a_{k-1} a_k H_{i+1}$
    $\Leftrightarrow a_1 \dots a_n \in L(A)$

Převod gramatiky typu 3 na konečný automat

Theorem 6.2 (ϵ–NFA pro gramatiku typu 3 rozpoznávající stejný jazyk)

Pro každý jazyk L generovaný gramatikou typu 3 existuje ϵ–NFA rozpoznávající L.

Proof: Převod gramatiky typu 3 na konečný automat

- Vezmeme $G = (V, T, P, S)$ obsahující jen pravidla tvaru $A \to aB$, $A \to \epsilon$, $A, B \in V, a \in T$ generující L (předchozí lemma)
- definujeme nedeterministický ϵ–NFA $A = (V, T, \delta, S, F)$, kde:

  $F = \{A | (A \to \epsilon) \in P\}$
  $\delta(A, a) = \{B | (A \to aB) \in P\}$
- $L(G) = L(A)$
  - $\epsilon \in L(G) \Leftrightarrow (S \to \epsilon) \in P \Leftrightarrow S \in F \Leftrightarrow \epsilon \in L(A)$
  - $a_1 \dots a_n \in L(G) \Leftrightarrow$ existuje derivace
    $(S \Rightarrow a_1 H_1 \Rightarrow \dots \Rightarrow a_1 \dots a_n H_n \Rightarrow a_1 \dots a_n)$
    $\Leftrightarrow \exists H_0, \dots, H_n \in V$ tak že $H_0 = S, H_n \in F$
    $H_{i+1} \in \delta(H_i, a_k)$ pro krok $a_1 \dots a_{k-1} H_i \Rightarrow a_1 \dots a_{k-1} a_k H_{i+1}$
    $\Leftrightarrow a_1 \dots a_n \in L(A)$

□

Převod gramatiky typu 3 na konečný automat

Theorem 6.2 (ϵ–NFA pro gramatiku typu 3 rozpoznávající stejný jazyk)

Pro každý jazyk L generovaný gramatikou typu 3 existuje ϵ–NFA rozpoznávající L.

Proof: Převod gramatiky typu 3 na konečný automat

- Vezmeme $G = (V, T, P, S)$ obsahující jen pravidla tvaru $A \to aB$, $A \to \epsilon$, $A, B \in V, a \in T$ generující L (předchozí lemma)
- definujeme nedeterministický ϵ–NFA $A = (V, T, \delta, S, F)$, kde:

  $F = \{A | (A \to \epsilon) \in P\}$
  $\delta(A, a) = \{B | (A \to aB) \in P\}$
- $L(G) = L(A)$
  - $\epsilon \in L(G) \Leftrightarrow (S \to \epsilon) \in P \Leftrightarrow S \in F \Leftrightarrow \epsilon \in L(A)$
  - $a_1 \dots a_n \in L(G) \Leftrightarrow$ existuje derivace
    $(S \Rightarrow a_1 H_1 \Rightarrow \dots \Rightarrow a_1 \dots a_n H_n \Rightarrow a_1 \dots a_n)$

    $\Leftrightarrow \exists H_0, \dots, H_n \in V$ tak že $H_0 = S, H_n \in F$
    $H_{i+1} \in \delta(H_i, a_k)$ pro krok $a_1 \dots a_{k-1} H_i \Rightarrow a_1 \dots a_{k-1} a_k H_{i+1}$

    $\Leftrightarrow a_1 \dots a_n \in L(A)$

□

Převod gramatiky typu 3 na konečný automat

Theorem 6.2 (ϵ–NFA pro gramatiku typu 3 rozpoznávající stejný jazyk)

Pro každý jazyk L generovaný gramatikou typu 3 existuje ϵ–NFA rozpoznávající L.

Proof: Převod gramatiky typu 3 na konečný automat

- Vezmeme $G = (V, T, P, S)$ obsahující jen pravidla tvaru $A \to aB$, $A \to \epsilon$, $A, B \in V, a \in T$ generující L (předchozí lemma)
- definujeme nedeterministický ϵ–NFA $A = (V, T, \delta, S, F)$, kde:

  $F = \{A | (A \to \epsilon) \in P\}$
  $\delta(A, a) = \{B | (A \to aB) \in P\}$
- $L(G) = L(A)$
  - $\epsilon \in L(G) \Leftrightarrow (S \to \epsilon) \in P \Leftrightarrow S \in F \Leftrightarrow \epsilon \in L(A)$
  - $a_1 \dots a_n \in L(G) \Leftrightarrow$ existuje derivace
    $(S \Rightarrow a_1 H_1 \Rightarrow \dots \Rightarrow a_1 \dots a_n H_n \Rightarrow a_1 \dots a_n)$

  $\Leftrightarrow$ $\exists H_0, \dots, H_n \in V$ tak že $H_0 = S, H_n \in F$
  $H_{i+1} \in \delta(H_i, a_k)$ pro krok $a_1 \dots a_{k-1} H_i \Rightarrow a_1 \dots a_{k-1} a_k H_{i+1}$

  $\Leftrightarrow$ $a_1 \dots a_n \in L(A)$

□

Převod gramatiky typu 3 na konečný automat

Theorem 6.2 (ϵ–NFA pro gramatiku typu 3 rozpoznávající stejný jazyk)

Pro každý jazyk L generovaný gramatikou typu 3 existuje ϵ–NFA rozpoznávající L.

Proof: Převod gramatiky typu 3 na konečný automat

- Vezmeme $G = (V, T, P, S)$ obsahující jen pravidla tvaru $A \to aB$, $A \to \epsilon$, $A, B \in V, a \in T$ generující L (předchozí lemma)
- definujeme nedeterministický ϵ–NFA $A = (V, T, \delta, S, F)$, kde:

  $F = \{A | (A \to \epsilon) \in P\}$
  $\delta(A, a) = \{B | (A \to aB) \in P\}$
- $L(G) = L(A)$
  - $\epsilon \in L(G) \Leftrightarrow (S \to \epsilon) \in P \Leftrightarrow S \in F \Leftrightarrow \epsilon \in L(A)$
  - $a_1 \dots a_n \in L(G) \Leftrightarrow$ existuje derivace
    $(S \Rightarrow a_1 H_1 \Rightarrow \dots \Rightarrow a_1 \dots a_n H_n \Rightarrow a_1 \dots a_n)$

  $\Leftrightarrow$ $\exists H_0, \dots, H_n \in V$ tak že $H_0 = S, H_n \in F$
  $H_{i+1} \in \delta(H_i, a_k)$ pro krok $a_1 \dots a_{k-1} H_i \Rightarrow a_1 \dots a_{k-1} a_k H_{i+1}$

  $\Leftrightarrow$ $a_1 \dots a_n \in L(A)$

□

Levé (a pravé) lineání gramatiky

Definition 6.5 (Levé (a pravé) lineání gramatiky)

Gramatiky typu 3 nazýváme také **pravé lineární** (neterminál je vždy vpravo).
Gramatika G je **levá lineání**, jestliže má pouze pravidla tvaru
$A \to Bw, A \to w, A, B \in V, w \in T^*$.

Lemma

Jazyky generované levou lineání gramatikou jsou právě regulární jazyky.

Proof:

$\Rightarrow$ 'otočením' pravidel levé lineární gramatiky dostaneme pravou lineární
$A \to Bw, A \to w$ převedeme na $A \to w^R B, A \to w^R$

- získaná gramatika generuje jazyk L^R, najdeme automat
- víme, že regulární jazyky jsou uzavřené na reverzi,
L^R je regulární, tudíž i $L = (L^R)^R$ je regulární

$\Leftarrow$ takto lze získat všechny regulární jazyky

Levé (a pravé) lineání gramatiky

Definition 6.5 (Levé (a pravé) lineání gramatiky)

Gramatiky typu 3 nazýváme také **pravé lineární** (neterminál je vždy vpravo).
Gramatika G je **levá lineání**, jestliže má pouze pravidla tvaru
$A \to Bw, A \to w, A, B \in V, w \in T^*$.

Lemma

Jazyky generované levou lineání gramatikou jsou právě regulární jazyky.

Proof:

$\Rightarrow$ 'otočením' pravidel levé lineární gramatiky dostaneme pravou lineární
$A \to Bw, A \to w$ převedeme na $A \to w^R B, A \to w^R$

- získaná gramatika generuje jazyk L^R, najdeme automat
- víme, že regulární jazyky jsou uzavřené na reverzi,
L^R je regulární, tudíž i $L = (L^R)^R$ je regulární

$\Leftarrow$ takto lze získat všechny regulární jazyky

Levé (a pravé) lineání gramatiky

Definition 6.5 (Levé (a pravé) lineání gramatiky)

Gramatiky typu 3 nazýváme také **pravé lineární** (neterminál je vždy vpravo).
Gramatika G je **levá lineání**, jestliže má pouze pravidla tvaru
$A \to Bw, A \to w, A, B \in V, w \in T^*$.

Lemma

Jazyky generované levou lineání gramatikou jsou právě regulární jazyky.

Proof:

$\Rightarrow$ 'otočením' pravidel levé lineární gramatiky dostaneme pravou lineární
$A \to Bw, A \to w$ převedeme na $A \to w^R B, A \to w^R$

- získaná gramatika generuje jazyk L^R, najdeme automat
- víme, že regulární jazyky jsou uzavřené na reverzi,
  L^R je regulární, tudíž i $L = (L^R)^R$ je regulární

$\Leftarrow$ takto lze získat všechny regulární jazyky

Levé (a pravé) lineání gramatiky

Definition 6.5 (Levé (a pravé) lineání gramatiky)

Gramatiky typu 3 nazýváme také **pravé lineární** (neterminál je vždy vpravo).
Gramatika G je **levá lineání**, jestliže má pouze pravidla tvaru
$A \to Bw, A \to w, A, B \in V, w \in T^*$.

Lemma

Jazyky generované levou lineání gramatikou jsou právě regulární jazyky.

Proof:

- $\Rightarrow$ 'otočením' pravidel levé lineární gramatiky dostaneme pravou lineární
  $A \to Bw, A \to w$ převedeme na $A \to w^R B, A \to w^R$
  - získaná gramatika generuje jazyk L^R, najdeme automat
  - víme, že regulární jazyky jsou uzavřené na reverzi,
    L^R je regulární, tudíž i $L = (L^R)^R$ je regulární
- $\Leftarrow$ takto lze získat všechny regulární jazyky

Levé (a pravé) lineání gramatiky

Definition 6.5 (Levé (a pravé) lineání gramatiky)

Gramatiky typu 3 nazýváme také **pravé lineární** (neterminál je vždy vpravo).
Gramatika G je **levá lineání**, jestliže má pouze pravidla tvaru
$A \to Bw, A \to w, A, B \in V, w \in T^*$.

Lemma

Jazyky generované levou lineání gramatikou jsou právě regulární jazyky.

Proof:

$\Rightarrow$ 'otočením' pravidel levé lineární gramatiky dostaneme pravou lineární
$A \to Bw, A \to w$ převedeme na $A \to w^R B, A \to w^R$

- získaná gramatika generuje jazyk L^R, najdeme automat
- víme, že regulární jazyky jsou uzavřené na reverzi,
  L^R je regulární, tudíž i $L = (L^R)^R$ je regulární

$\Leftarrow$ takto lze získat všechny regulární jazyky

Levé (a pravé) lineání gramatiky

Definition 6.5 (Levé (a pravé) lineání gramatiky)

Gramatiky typu 3 nazýváme také **pravé lineární** (neterminál je vždy vpravo).
Gramatika G je **levá lineání**, jestliže má pouze pravidla tvaru
$A \to Bw, A \to w, A, B \in V, w \in T^*$.

Lemma

Jazyky generované levou lineání gramatikou jsou právě regulární jazyky.

Proof:

$\Rightarrow$ 'otočením' pravidel levé lineární gramatiky dostaneme pravou lineární
$A \to Bw, A \to w$ převedeme na $A \to w^R B, A \to w^R$

- získaná gramatika generuje jazyk L^R, najdeme automat
- víme, že regulární jazyky jsou uzavřené na reverzi,
  L^R je regulární, tudíž i $L = (L^R)^R$ je regulární

$\Leftarrow$ takto lze získat všechny regulární jazyky

Levé (a pravé) lineání gramatiky

Definition 6.5 (Levé (a pravé) lineání gramatiky)

Gramatiky typu 3 nazýváme také **pravé lineární** (neterminál je vždy vpravo).
Gramatika G je **levá lineání**, jestliže má pouze pravidla tvaru
$A \to Bw, A \to w, A, B \in V, w \in T^*$.

Lemma

Jazyky generované levou lineání gramatikou jsou právě regulární jazyky.

Proof:

$\Rightarrow$ 'otočením' pravidel levé lineární gramatiky dostaneme pravou lineární
$A \to Bw, A \to w$ převedeme na $A \to w^R B, A \to w^R$

- získaná gramatika generuje jazyk L^R, najdeme automat
- víme, že regulární jazyky jsou uzavřené na reverzi,
L^R je regulární, tudíž i $L = (L^R)^R$ je regulární

$\Leftarrow$ takto lze získat všechny regulární jazyky

- (FA$\Rightarrow$reverze$\Rightarrow$ pravá lineární gramatika$\Rightarrow$ levá lineární gramatika) $\square$

Levé (a pravé) lineání gramatiky

Definition 6.5 (Levé (a pravé) lineání gramatiky)

Gramatiky typu 3 nazýváme také **pravé lineární** (neterminál je vždy vpravo).
Gramatika G je **levá lineání**, jestliže má pouze pravidla tvaru
$A \to Bw, A \to w, A, B \in V, w \in T^*$.

Lemma

Jazyky generované levou lineání gramatikou jsou právě regulární jazyky.

Proof:

$\Rightarrow$ 'otočením' pravidel levé lineární gramatiky dostaneme pravou lineární
$A \to Bw, A \to w$ převedeme na $A \to w^R B, A \to w^R$

- získaná gramatika generuje jazyk L^R, najdeme automat
- víme, že regulární jazyky jsou uzavřené na reverzi,
L^R je regulární, tudíž i $L = (L^R)^R$ je regulární

$\Leftarrow$ takto lze získat všechny regulární jazyky

- (FA$\Rightarrow$reverze$\Rightarrow$ pravá lineární gramatika$\Rightarrow$ levá lineární gramatika) $\square$

Lineární gramatiky (a jazyky)

- Levá a pravá lineární pravidla dohromady jsou už silnější.

Definition 6.6 (lineární gramatika, jazyk)

Gramatika je lineární, jestliže má pouze pravidla tvaru $A \to uBw, A \to w, A, B \in V, u, w \in T^*$ (na pravé straně vždy maximálně jeden neterminál).
Lineární jazyky jsou právě jazyky generované lineáními gramatikami.

- Zřejmě platí: regulární jazyky $\subseteq$ lineární jazyky.
- Jde o vlastní podmnožinu $\subsetneq$.

Example 6.6 (lineární, neregulární jazyk)

Jazyk $L = \{0^i 1^i | i \geq 1\}$ není regulární jazyk, ale je lineární, generovaný gramatikou s pravidly $S \to 0S1|01$.

Pozorování:

- lineární pravidla lze rozložit na levě a pravě lineání pravidla: $S \to 0A, A \to S1$.

Lineární gramatiky (a jazyky)

- Levá a pravá lineární pravidla dohromady jsou už silnější.

Definition 6.6 (lineární gramatika, jazyk)

Gramatika je lineární, jestliže má pouze pravidla tvaru $A \to uBw, A \to w, A, B \in V, u, w \in T^*$ (na pravé straně vždy maximálně jeden neterminál).
Lineární jazyky jsou právě jazyky generované lineáními gramatikami.

- Zřejmě platí: regulární jazyky $\subseteq$ lineární jazyky.
- Jde o vlastní podmnožinu $\subsetneq$.

Example 6.6 (lineární, neregulární jazyk)

Jazyk $L = \{0^i 1^i | i \geq 1\}$ není regulární jazyk, ale je lineární, generovaný gramatikou s pravidly $S \to 0S1|01$.

Pozorování:

- lineární pravidla lze rozložit na levě a pravě lineání pravidla: $S \to 0A, A \to S1$.

Lineární gramatiky (a jazyky)

- Levá a pravá lineární pravidla dohromady jsou už silnější.

Definition 6.6 (lineární gramatika, jazyk)

Gramatika je lineární, jestliže má pouze pravidla tvaru $A \to uBw, A \to w, A, B \in V, u, w \in T^*$ (na pravé straně vždy maximálně jeden neterminál).
Lineární jazyky jsou právě jazyky generované lineáními gramatikami.

- Zřejmě platí: regulární jazyky $\subseteq$ lineární jazyky.
- Jde o vlastní podmnožinu $\subsetneq$.

Example 6.6 (lineární, neregulární jazyk)

Jazyk $L = \{0^i 1^i | i \geq 1\}$ není regulární jazyk, ale je lineární, generovaný gramatikou s pravidly $S \to 0S1|01$.

Pozorování:

- lineární pravidla lze rozložit na levě a pravě lineání pravidla: $S \to 0A, A \to S1$.

Lineární gramatiky (a jazyky)

- Levá a pravá lineární pravidla dohromady jsou už silnější.

Definition 6.6 (lineární gramatika, jazyk)

Gramatika je lineární, jestliže má pouze pravidla tvaru $A \to uBw, A \to w, A, B \in V, u, w \in T^*$ (na pravé straně vždy maximálně jeden neterminál).
Lineární jazyky jsou právě jazyky generované lineáními gramatikami.

- Zřejmě platí: regulární jazyky $\subseteq$ lineární jazyky.
- Jde o vlastní podmnožinu $\subsetneq$.

Example 6.6 (lineární, neregulární jazyk)

Jazyk $L = \{0^i 1^i | i \geq 1\}$ není regulární jazyk, ale je lineární, generovaný gramatikou s pravidly $S \to 0S1|01$.

Pozorování:

- lineární pravidla lze rozložit na levě a pravě lineání pravidla: $S \to 0A, A \to S1$.

Bezkontextová gramatika pro jednoduché výrazy

Definition (Bezkontextová gramatika)

Bezkontextová gramatika je gramatika, kde všechna pravidla jsou tvaru $A \to \omega, \omega \in (V \cup T)^*$.

Example 6.7 (CFG pro jednoduché výrazy)

Gramatika pro jednoduché výrazy $G = (\{E, I\}, \{+, *, (,), a, b, 0, 1\}, P, E)$, P jsou pravidla vypsaná vpravo.

- Pravidla 1–4 definují výraz.
- Pravidla 5–10 definují identifikátor I, odpovídající regulárnímu výrazu $(\mathbf{a} + \mathbf{b})(\mathbf{a} + \mathbf{b} + \mathbf{0} + \mathbf{1})^*$.

CFG pro jednoduché výrazy

1. $E \to I$
2. $E \to E + E$
3. $E \to E * E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

Bezkontextová gramatika pro jednoduché výrazy

Definition (Bezkontextová gramatika)

Bezkontextová gramatika je gramatika, kde všechna pravidla jsou tvaru $A \to \omega, \omega \in (V \cup T)^*$.

Example 6.7 (CFG pro jednoduché výrazy)

Gramatika pro jednoduché výrazy $G = (\{E, I\}, \{+, *, (,), a, b, 0, 1\}, P, E)$, P jsou pravidla vypsaná vpravo.

- Pravidla 1–4 definují výraz.
- Pravidla 5–10 definují identifikátor I, odpovídající regulárnímu výrazu $(\mathbf{a} + \mathbf{b})(\mathbf{a} + \mathbf{b} + \mathbf{0} + \mathbf{1})^*$.

CFG pro jednoduché výrazy

1. $E \to I$
2. $E \to E + E$
3. $E \to E * E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

Derivační strom

Definition 6.7 (Derivační strom)

Mějme gramatiku $G = (V, T, P, S)$. **Derivační strom** pro G je strom, kde:

- Kořen (kreslíme nahoře) je označen startovním symbolem S,
- každý vnitřní uzel je ohodnocen neterminálem V.
- Každý uzel je ohodnocen prvkem $\in V \cup T \cup \{\epsilon\}$.
- Je–li uzel ohodnocen ϵ, je jediným dítětem svého rodiče.
- Je–li A ohodnocení vrcholu a jeho děti zleva pořadě jsou ohodnoceny $X_1, \ldots, X_k$, pak $(A \to X_1 \ldots X_k) \in P$ je pravidlo gramatiky.

Notation 1 (Terminologie stromů)

Uzly, rodiče, děti, kořen, vnitřní uzly, listy, následníci, předci.

- Stromová struktura reprezentuje zdrojový program v překladači. Struktura usnadňuje překlad do strojového kódu.

Derivační strom

Definition 6.7 (Derivační strom)

Mějme gramatiku $G = (V, T, P, S)$. **Derivační strom** pro G je strom, kde:

- Kořen (kreslíme nahoře) je označen startovním symbolem S,
- každý vnitřní uzel je ohodnocen neterminálem V.
- Každý uzel je ohodnocen prvkem $\in V \cup T \cup \{\epsilon\}$.
- Je–li uzel ohodnocen ϵ, je jediným dítětem svého rodiče.
- Je–li A ohodnocení vrcholu a jeho děti zleva pořadě jsou ohodnoceny $X_1, \ldots, X_k$, pak $(A \to X_1 \ldots X_k) \in P$ je pravidlo gramatiky.

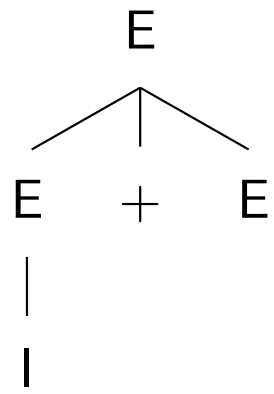
Notation 1 (Terminologie stromů)

Uzly, rodiče, děti, kořen, vnitřní uzly, listy, následníci, předci.

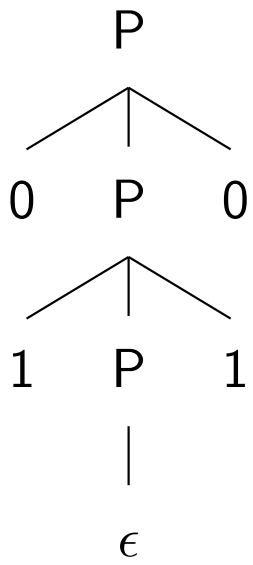
- Stromová struktura reprezentuje zdrojový program v překladači. Struktura usnadňuje překlad do strojového kódu.

Příklady stromů, Strom dává slovo (yield)

Derivační strom $E \Rightarrow^* I + E$.



Derivační strom $P \Rightarrow^* 0110$.



Definition 6.8 (Strom dává slovo (yield))

Říkáme, že **derivanční strom dává slovo** w **(yield)**, jestliže w je slovo složené z ohodnocení listů bráno zleva doprava.

Levá a pravá derivace

Definition 6.9 (Levá a pravá derivace)

Levá derivace (leftmost) $\Rightarrow_{lm}$, $\Rightarrow_{lm}^*$ v každém kroku přepisuje nejlevnější neterminál.
Pravá derivace (rightmost) $\Rightarrow_{rm}$, $\Rightarrow_{rm}^*$ v každém kroku přepisuje nejpravější neterminál.

Example 6.8 (levá derivace)

$$E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E \Rightarrow_{lm} a * (E) \Rightarrow_{lm} a * (E + E) \Rightarrow_{lm} a * (I + E) \Rightarrow_{lm}$$
$$\Rightarrow_{lm} a * (a + E) \Rightarrow_{lm} a * (a + I) \Rightarrow_{lm} a * (a + I0) \Rightarrow_{lm} a * (a + I00) \Rightarrow_{lm} a * (a + b00)$$

Pravá derivace používá stejné přepisy, jen je provádí v jiném pořadí.

Example 6.9 (rightmost derivation)

$$E \Rightarrow_{rm} E * E \Rightarrow_{rm} E * (E) \Rightarrow_{rm} E * (E + E) \Rightarrow_{rm} E * (E + I) \Rightarrow_{rm}$$
$$\Rightarrow_{rm} E * (E + I0) \Rightarrow_{rm} E * (E + I00) \Rightarrow_{rm} E * (E + b00) \Rightarrow_{rm}$$
$$\Rightarrow_{rm} E * (I + b00) \Rightarrow_{rm} E * (a + b00) \Rightarrow_{rm} I * (a + b00) \Rightarrow_{rm} a * (a + b00)$$

Levá a pravá derivace

Definition 6.9 (Levá a pravá derivace)

Levá derivace (leftmost) $\Rightarrow_{lm}$, $\Rightarrow_{lm}^*$ v každém kroku přepisuje nejlevnější neterminál.
Pravá derivace (rightmost) $\Rightarrow_{rm}$, $\Rightarrow_{rm}^*$ v každém kroku přepisuje nejpravější neterminál.

Example 6.8 (levá derivace)

$E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E \Rightarrow_{lm} a * (E) \Rightarrow_{lm} a * (E + E) \Rightarrow_{lm} a * (I + E) \Rightarrow_{lm}$
$\Rightarrow_{lm} a * (a + E) \Rightarrow_{lm} a * (a + I) \Rightarrow_{lm} a * (a + I0) \Rightarrow_{lm} a * (a + I00) \Rightarrow_{lm} a * (a + b00)$

Pravá derivace používá stejné přepisy, jen je provádí v jiném pořadí.

Example 6.9 (rightmost derivation)

$E \Rightarrow_{rm} E * E \Rightarrow_{rm} E * (E) \Rightarrow_{rm} E * (E + E) \Rightarrow_{rm} E * (E + I) \Rightarrow_{rm}$
$\Rightarrow_{rm} E * (E + I0) \Rightarrow_{rm} E * (E + I00) \Rightarrow_{rm} E * (E + b00) \Rightarrow_{rm}$
$\Rightarrow_{rm} E * (I + b00) \Rightarrow_{rm} E * (a + b00) \Rightarrow_{rm} I * (a + b00) \Rightarrow_{rm} a * (a + b00)$

Levá a pravá derivace

Definition 6.9 (Levá a pravá derivace)

Levá derivace (leftmost) $\Rightarrow_{lm}$, $\Rightarrow_{lm}^*$ v každém kroku přepisuje nejlevnější neterminál.
Pravá derivace (rightmost) $\Rightarrow_{rm}$, $\Rightarrow_{rm}^*$ v každém kroku přepisuje nejpravější neterminál.

Example 6.8 (levá derivace)

$E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E \Rightarrow_{lm} a * (E) \Rightarrow_{lm} a * (E + E) \Rightarrow_{lm} a * (I + E) \Rightarrow_{lm}$
$\Rightarrow_{lm} a * (a + E) \Rightarrow_{lm} a * (a + I) \Rightarrow_{lm} a * (a + I0) \Rightarrow_{lm} a * (a + I00) \Rightarrow_{lm} a * (a + b00)$

Pravá derivace používá stejné přepisy, jen je provádí v jiném pořadí.

Example 6.9 (rightmost derivation)

$E \Rightarrow_{rm} E * E \Rightarrow_{rm} E * (E) \Rightarrow_{rm} E * (E + E) \Rightarrow_{rm} E * (E + I) \Rightarrow_{rm}$
$\Rightarrow_{rm} E * (E + I0) \Rightarrow_{rm} E * (E + I00) \Rightarrow_{rm} E * (E + b00) \Rightarrow_{rm}$
$\Rightarrow_{rm} E * (I + b00) \Rightarrow_{rm} E * (a + b00) \Rightarrow_{rm} I * (a + b00) \Rightarrow_{rm} a * (a + b00)$

Derivace a derivační stromy

Theorem 6.3

Pro danou gramatiku $G = (V, T, P, S)$ a $w \in T^*$ jsou následující tvrzení ekvivalentní:

- $A \Rightarrow^* w$.
- $A \Rightarrow^*_{lm} w$.
- $A \Rightarrow^*_{rm} w$.
- *Existuje derivační strom s kořenem A dávající slovo w.*

Derivace a derivační stromy

Theorem 6.3

Pro danou gramatiku $G = (V, T, P, S)$ a $w \in T^*$ jsou následující tvrzení ekvivalentní:

- $A \Rightarrow^* w$.
- $A \Rightarrow^*_{lm} w$.
- $A \Rightarrow^*_{rm} w$.
- *Existuje derivační strom s kořenem A dávající slovo w.*

Derivace a derivační stromy

Theorem 6.3

Pro danou gramatiku $G = (V, T, P, S)$ a $w \in T^*$ jsou následující tvrzení ekvivalentní:

- $A \Rightarrow^* w$.
- $A \Rightarrow^*_{lm} w$.
- $A \Rightarrow^*_{rm} w$.
- *Existuje derivační strom s kořenem A dávající slovo w.*

Derivace a derivační stromy

Theorem 6.3

Pro danou gramatiku $G = (V, T, P, S)$ a $w \in T^*$ jsou následující tvrzení ekvivalentní:

- $A \Rightarrow^* w$.
- $A \Rightarrow^*_{lm} w$.
- $A \Rightarrow^*_{rm} w$.
- *Existuje derivační strom s kořenem A dávající slovo w.*

Od stromů k derivaci

Lemma

Mějme CFG $G = (V, T, P, S)$ a derivační strom s kořenem A dávající slovo $w \in T^*$.
Pak existuje levá derivace $A \Rightarrow_{lm}^* w$ v G.

Příprava důkazu: 'obalení derivace'

Mějme následující derivaci:

$$E \Rightarrow I \Rightarrow Ib \rightarrow ab.$$

Pro libovolná slova $\alpha, \beta \in (V \cup T)^*$ je také derivace:

$$\alpha E \beta \Rightarrow \alpha I \beta \Rightarrow \alpha I b \beta \Rightarrow \alpha a b \beta.$$

Proof: ∃ derivační strom pak existuje levá derivace $\Rightarrow_{lm}$

Indukcí podle výšky stomu.

- Základ: výška 1: Kořen A s dětmi dávajícími w. Je to derivační strom, proto, $A \to w$ je pravidlo $\in P$, tedy $A \Rightarrow_{lm} w$ v jednom kroku.
- Indukce: výška $n > 1$. Kořen A s dětmi $X_1, X_2, \ldots, X_k$.
  - Je-li $X_i \in T$, definujeme $w_i = X_i$.
  - Je-li $X_i \in V$, z indukčního předpokladu $X_i \Rightarrow_{lm}^* w_i$.

  Levou derivaci konstruujeme induktivně pro $i = 1, \ldots, k$ složíme $A \Rightarrow_{lm}^* w_1 w_2 \ldots w_i X_{i+1} X_{i+2} \ldots X_k$.
  - Pro $X_i \in T$ jen zvedneme čítač $i \mapsto i+1$.
  - Pro $X_i \in V$ přepíšeme derivaci $X_i \Rightarrow_{lm} \alpha_1 \Rightarrow_{lm} \alpha_2 \Rightarrow_{lm} \ldots \Rightarrow_{lm} w_i$ na

    $w_1 w_2 \ldots w_{i-1} X_i X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$

    $w_1 w_2 \ldots w_{i-1} \alpha_1 X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$

    $\ldots$

    $w_1 w_2 \ldots w_{i-1} w_i X_{i+1} X_{i+2} \ldots X_k$

  Pro $i = k$ dostaneme levou derivaci w z A.

□

Proof: ∃ derivační strom pak existuje levá derivace $\Rightarrow_{lm}$

Indukcí podle výšky stomu.

- Základ: výška 1: Kořen A s dětmi dávajícími w. Je to derivační strom, proto, $A \to w$ je pravidlo $\in P$, tedy $A \Rightarrow_{lm} w$ v jednom kroku.
- Indukce: výška $n > 1$. Kořen A s dětmi $X_1, X_2, \ldots, X_k$.
  - Je–li $X_i \in T$, definujeme $w_i \equiv X_i$.
  - Je–li $X_i \in V$, z indukčního předpokladu $X_i \Rightarrow^*_{lm} w_i$.

  Levou derivaci konstruujeme induktivně pro $i = 1, \ldots, k$ složíme $A \Rightarrow^*_{lm} w_1 w_2 \ldots w_i X_{i+1} X_{i+2} \ldots X_k$.
  - Pro $X_i \in T$ jen zvedneme čítač $i++$.
  - Pro $X_i \in V$ přepíšeme derivaci: $X_i \Rightarrow_{lm} \alpha_1 \Rightarrow_{lm} \alpha_2 \ldots \Rightarrow_{lm} w_i$ na

$$w_1 w_2 \ldots w_{i-1} X_i X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$$
$$w_1 w_2 \ldots w_{i-1} \alpha_1 X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$$
$$\ldots$$
$$\Rightarrow_{lm} w_1 w_2 \ldots w_{i-1} w_i X_{i+1} X_{i+2} \ldots X_k.$$

Pro $i = k$ dostaneme levou derivaci w z A. □

Proof: ∃ derivační strom pak existuje levá derivace $\Rightarrow_{lm}$

Indukcí podle výšky stomu.

- Základ: výška 1: Kořen A s dětmi dávajícími w. Je to derivační strom, proto, $A \to w$ je pravidlo $\in P$, tedy $A \Rightarrow_{lm} w$ v jednom kroku.
- Indukce: výška $n > 1$. Kořen A s dětmi $X_1, X_2, \ldots, X_k$.
  - Je–li $X_i \in T$, definujeme $w_i \equiv X_i$.
  - Je–li $X_i \in V$, z indukčního předpokladu $X_i \Rightarrow^*_{lm} w_i$.

  Levou derivaci konstruujeme induktivně pro $i = 1, \ldots, k$ složíme $A \Rightarrow^*_{lm} w_1 w_2 \ldots w_i X_{i+1} X_{i+2} \ldots X_k$.
  - Pro $X_i \in T$ jen zvedneme čítač $i++$.
  - Pro $X_i \in V$ přepíšeme derivaci: $X_i \Rightarrow_{lm} \alpha_1 \Rightarrow_{lm} \alpha_2 \ldots \Rightarrow_{lm} w_i$ na

$$w_1 w_2 \ldots w_{i-1} X_i X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$$
$$w_1 w_2 \ldots w_{i-1} \alpha_1 X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$$
$$\ldots$$
$$\Rightarrow_{lm} w_1 w_2 \ldots w_{i-1} w_i X_{i+1} X_{i+2} \ldots X_k.$$

Pro $i = k$ dostaneme levou derivaci w z A. □

Proof: ∃ derivační strom pak existuje levá derivace $\Rightarrow_{lm}$

Indukcí podle výšky stomu.

- Základ: výška 1: Kořen A s dětmi dávajícími w. Je to derivační strom, proto, $A \to w$ je pravidlo $\in P$, tedy $A \Rightarrow_{lm} w$ v jednom kroku.
- Indukce: výška $n > 1$. Kořen A s dětmi $X_1, X_2, \ldots, X_k$.
  - Je–li $X_i \in T$, definujeme $w_i \equiv X_i$.
  - Je–li $X_i \in V$, z indukčního předpokladu $X_i \Rightarrow^*_{lm} w_i$.

  Levou derivaci konstruujeme induktivně pro $i = 1, \ldots, k$ složíme $A \Rightarrow^*_{lm} w_1 w_2 \ldots w_i X_{i+1} X_{i+2} \ldots X_k$.
  - Pro $X_i \in T$ jen zvedneme čítač $i++$.
  - Pro $X_i \in V$ přepíšeme derivaci: $X_i \Rightarrow_{lm} \alpha_1 \Rightarrow_{lm} \alpha_2 \ldots \Rightarrow_{lm} w_i$ na

$$w_1 w_2 \ldots w_{i-1} X_i X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$$
$$w_1 w_2 \ldots w_{i-1} \alpha_1 X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$$
$$\ldots$$
$$\Rightarrow_{lm} w_1 w_2 \ldots w_{i-1} w_i X_{i+1} X_{i+2} \ldots X_k.$$

Pro $i = k$ dostaneme levou derivaci w z A. □

Proof: ∃ derivační strom pak existuje levá derivace $\Rightarrow_{lm}$

Indukcí podle výšky stomu.

- Základ: výška 1: Kořen A s dětmi dávajícími w. Je to derivační strom, proto, $A \to w$ je pravidlo $\in P$, tedy $A \Rightarrow_{lm} w$ v jednom kroku.
- Indukce: výška $n > 1$. Kořen A s dětmi $X_1, X_2, \ldots, X_k$.
  - Je–li $X_i \in T$, definujeme $w_i \equiv X_i$.
  - Je–li $X_i \in V$, z indukčního předpokladu $X_i \Rightarrow^*_{lm} w_i$.

  Levou derivaci konstruujeme induktivně pro $i = 1, \ldots, k$ složíme $A \Rightarrow^*_{lm} w_1 w_2 \ldots w_i X_{i+1} X_{i+2} \ldots X_k$.
  - Pro $X_i \in T$ jen zvedneme čítač $i++$.
  - Pro $X_i \in V$ přepíšeme derivaci: $X_i \Rightarrow_{lm} \alpha_1 \Rightarrow_{lm} \alpha_2 \ldots \Rightarrow_{lm} w_i$ na

$$w_1 w_2 \ldots w_{i-1} X_i X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$$
$$w_1 w_2 \ldots w_{i-1} \alpha_1 X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm}$$
$$\ldots$$
$$\Rightarrow_{lm} w_1 w_2 \ldots w_{i-1} w_i X_{i+1} X_{i+2} \ldots X_k.$$

Pro $i = k$ dostaneme levou derivaci w z A. □

Proof: ∃ derivační strom pak existuje levá derivace $\Rightarrow_{lm}$

Indukcí podle výšky stomu.

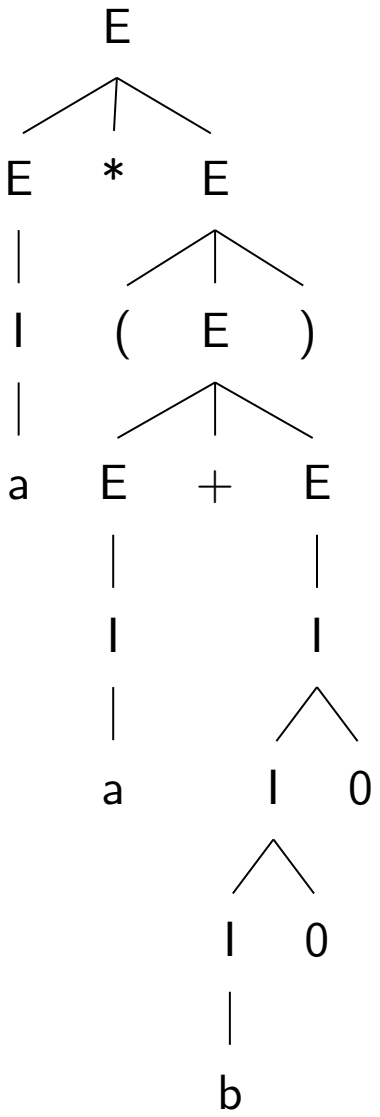
- Základ: výška 1: Kořen A s dětmi dávajícími w. Je to derivační strom, proto, $A \to w$ je pravidlo $\in P$, tedy $A \Rightarrow_{lm} w$ v jednom kroku.
- Indukce: výška $n > 1$. Kořen A s dětmi $X_1, X_2, \ldots, X_k$.
  - Je–li $X_i \in T$, definujeme $w_i \equiv X_i$.
  - Je–li $X_i \in V$, z indukčního předpokladu $X_i \Rightarrow^*_{lm} w_i$.

  Levou derivaci konstruujeme induktivně pro $i = 1, \ldots, k$ složíme $A \Rightarrow^*_{lm} w_1 w_2 \ldots w_i X_{i+1} X_{i+2} \ldots X_k$.
  - Pro $X_i \in T$ jen zvedneme čítač $i++$.
  - Pro $X_i \in V$ přepíšeme derivaci: $X_i \Rightarrow_{lm} \alpha_1 \Rightarrow_{lm} \alpha_2 \ldots \Rightarrow_{lm} w_i$ na

$$
\begin{aligned}
& w_1 w_2 \ldots w_{i-1} X_i X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm} \\
& w_1 w_2 \ldots w_{i-1} \alpha_1 X_{i+1} X_{i+2} \ldots X_k \Rightarrow_{lm} \\
& \qquad \ldots \\
& \Rightarrow_{lm} w_1 w_2 \ldots w_{i-1} w_i X_{i+1} X_{i+2} \ldots X_k .
\end{aligned}
$$

Pro $i = k$ dostaneme levou derivaci w z A. □

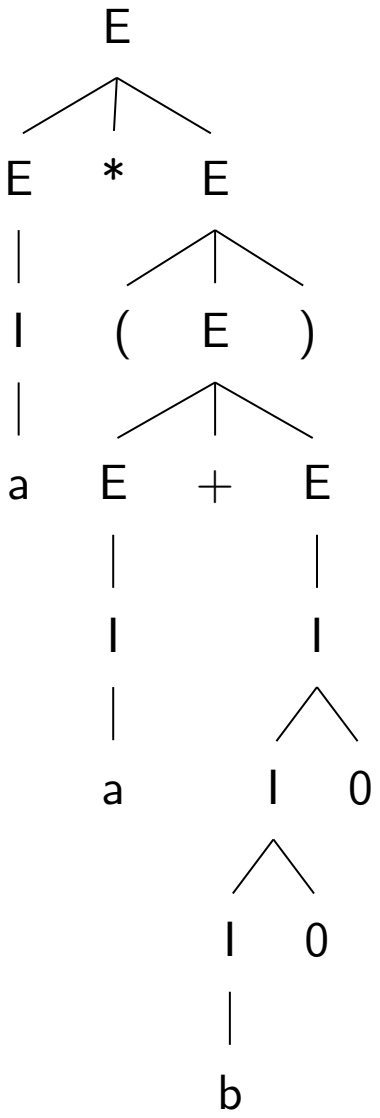
Příklad levé derivace z derivačního stromu



Je příjemnější zachytit derivaci stromem.

- Kořen: $E \Rightarrow_{lm} E * E$
- Levé dítě kořene: $E \Rightarrow_{lm} I \Rightarrow_{lm} a$
- Kořen a levé dítě: $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E$
- Pravé dítě kořene:
  $E \Rightarrow_{lm} (E) \Rightarrow_{lm} (E + E) \Rightarrow_{lm} (I + E) \Rightarrow_{lm} (a + E)$
  $\Rightarrow_{lm} (a+I) \Rightarrow_{lm} (a+I0) \Rightarrow_{lm} (a+I00) \Rightarrow_{lm} (a+b00)$
- Plná derivace:
  $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (E) \Rightarrow_{lm} a * (E + E) \Rightarrow_{lm} a * (I + E) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + E) \Rightarrow_{lm} a * (a + I) \Rightarrow_{lm} a * (a + I0) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + I00) \Rightarrow_{lm} a * (a + b00)$.

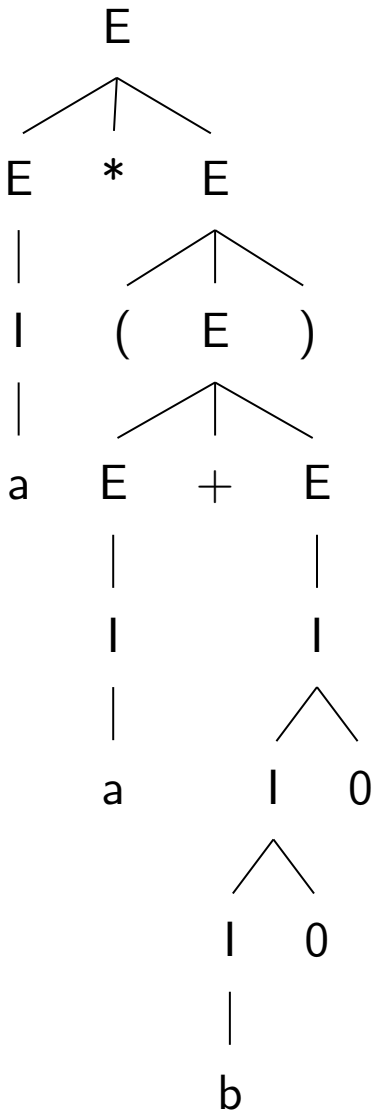
Příklad levé derivace z derivačního stromu



Je příjemnější zachytit derivaci stromem.

- Kořen: $E \Rightarrow_{lm} E * E$
- Levé dítě kořene: $E \Rightarrow_{lm} I \Rightarrow_{lm} a$
- Kořen a levé dítě: $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E$
- Pravé dítě kořene:
  $E \Rightarrow_{lm} (E) \Rightarrow_{lm} (E + E) \Rightarrow_{lm} (I + E) \Rightarrow_{lm} (a + E)$
  $\Rightarrow_{lm} (a+I) \Rightarrow_{lm} (a+I0) \Rightarrow_{lm} (a+I00) \Rightarrow_{lm} (a+b00)$
- Plná derivace:
  $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (E) \Rightarrow_{lm} a * (E + E) \Rightarrow_{lm} a * (I + E) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + E) \Rightarrow_{lm} a * (a + I) \Rightarrow_{lm} a * (a + I0) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + I00) \Rightarrow_{lm} a * (a + b00).$

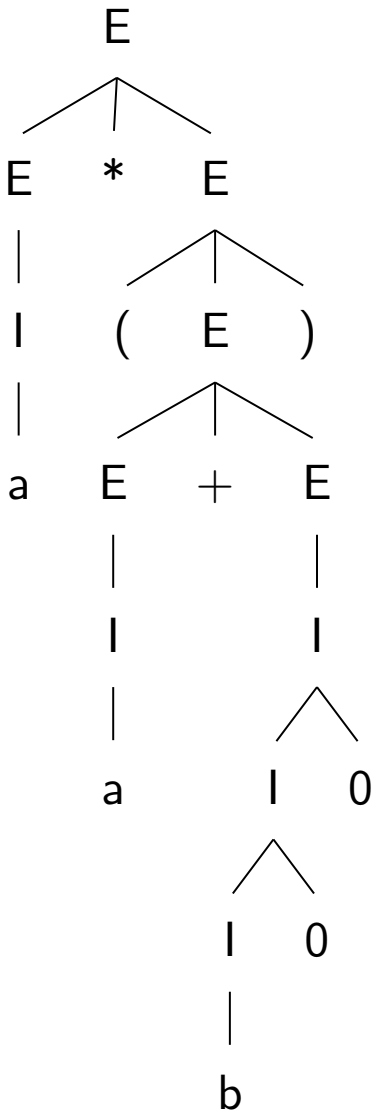
Příklad levé derivace z derivačního stromu



Je příjemnější zachytit derivaci stromem.

- Kořen: $E \Rightarrow_{lm} E * E$
- Levé dítě kořene: $E \Rightarrow_{lm} I \Rightarrow_{lm} a$
- Kořen a levé dítě: $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E$
- Pravé dítě kořene:
  $E \Rightarrow_{lm} (E) \Rightarrow_{lm} (E + E) \Rightarrow_{lm} (I + E) \Rightarrow_{lm} (a + E)$
  $\Rightarrow_{lm} (a+I) \Rightarrow_{lm} (a+I0) \Rightarrow_{lm} (a+I00) \Rightarrow_{lm} (a+b00)$
- Plná derivace:
  $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (E) \Rightarrow_{lm} a * (E + E) \Rightarrow_{lm} a * (I + E) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + E) \Rightarrow_{lm} a * (a + I) \Rightarrow_{lm} a * (a + I0) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + I00) \Rightarrow_{lm} a * (a + b00).$

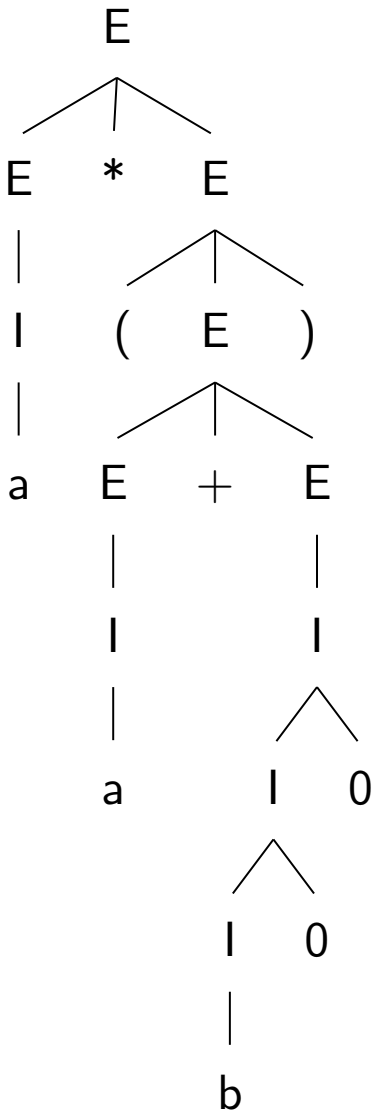
Příklad levé derivace z derivačního stromu



Je příjemnější zachytit derivaci stromem.

- Kořen: $E \Rightarrow_{lm} E * E$
- Levé dítě kořene: $E \Rightarrow_{lm} I \Rightarrow_{lm} a$
- Kořen a levé dítě: $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E$
- Pravé dítě kořene:
  $E \Rightarrow_{lm} (E) \Rightarrow_{lm} (E + E) \Rightarrow_{lm} (I + E) \Rightarrow_{lm} (a + E)$
  $\Rightarrow_{lm} (a+I) \Rightarrow_{lm} (a+I0) \Rightarrow_{lm} (a+I00) \Rightarrow_{lm} (a+b00)$
- Plná derivace:
  $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (E) \Rightarrow_{lm} a * (E + E) \Rightarrow_{lm} a * (I + E) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + E) \Rightarrow_{lm} a * (a + I) \Rightarrow_{lm} a * (a + I0) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + I00) \Rightarrow_{lm} a * (a + b00).$

Příklad levé derivace z derivačního stromu



Je příjemnější zachytit derivaci stromem.

- Kořen: $E \Rightarrow_{lm} E * E$
- Levé dítě kořene: $E \Rightarrow_{lm} I \Rightarrow_{lm} a$
- Kořen a levé dítě: $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E$
- Pravé dítě kořene:
  $E \Rightarrow_{lm} (E) \Rightarrow_{lm} (E + E) \Rightarrow_{lm} (I + E) \Rightarrow_{lm} (a + E)$
  $\Rightarrow_{lm} (a+I) \Rightarrow_{lm} (a+I0) \Rightarrow_{lm} (a+I00) \Rightarrow_{lm} (a+b00)$
- PIná derivace:
  $E \Rightarrow_{lm} E * E \Rightarrow_{lm} I * E \Rightarrow_{lm} a * E \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (E) \Rightarrow_{lm} a * (E + E) \Rightarrow_{lm} a * (I + E) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + E) \Rightarrow_{lm} a * (a + I) \Rightarrow_{lm} a * (a + I0) \Rightarrow_{lm}$
  $\Rightarrow_{lm} a * (a + I00) \Rightarrow_{lm} a * (a + b00).$

Ekvivalence gramatik

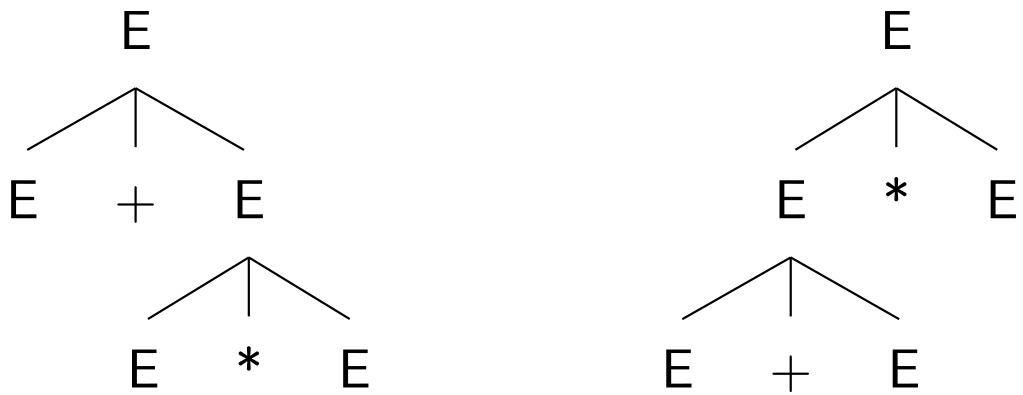
Definition 6.10 (ekvivalence gramatik)

Gramatiky G_1, G_2 jsou **ekvivalentní**, jestliže $L(G_1) = L(G_2)$, tj. generují stejný jazyk.

Víceznačnost gramatik

Dvě derivace téhož výrazu:

$E \Rightarrow E + E \Rightarrow E + E * E$ $\quad$ $E \Rightarrow E * E \Rightarrow E + E * E$



- Rozdíl je důležitý, vlevo $1 + (2 * 3) = 7$, vpravo $(1 + 2) * 3 = 9$.
- Tato gramatika může být modifikovaná na jednoznačnou.

Example 6.10

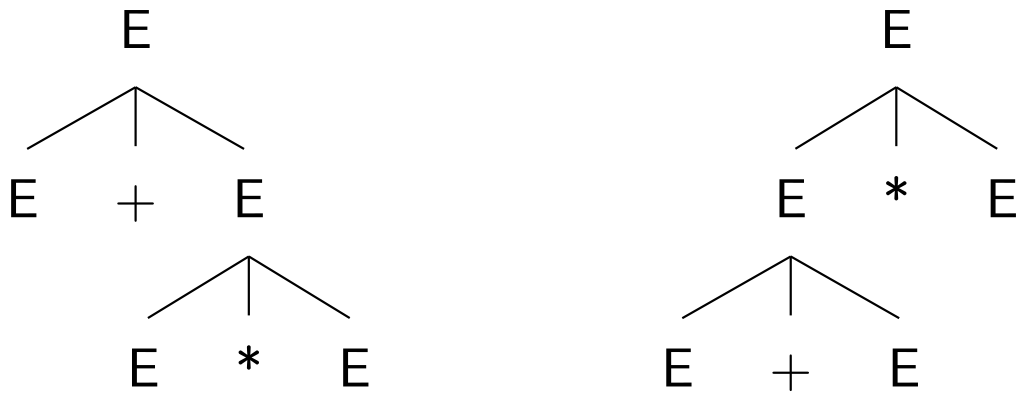
Různé derivace mohou reprezentovat stejný derivační strom, pak není problém.

1. $E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + I \Rightarrow a + b$
2. $E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b.$

Víceznačnost gramatik

Dvě derivace téhož výrazu:

$E \Rightarrow E + E \Rightarrow E + E * E$ $\quad$ $E \Rightarrow E * E \Rightarrow E + E * E$



- Rozdíl je důležitý, vlevo $1 + (2 * 3) = 7$, vpravo $(1 + 2) * 3 = 9$.
- Tato gramatika může být modifikovaná na jednoznačnou.

Example 6.10

Různé derivace mohou reprezentovat stejný derivační strom, pak není problém.

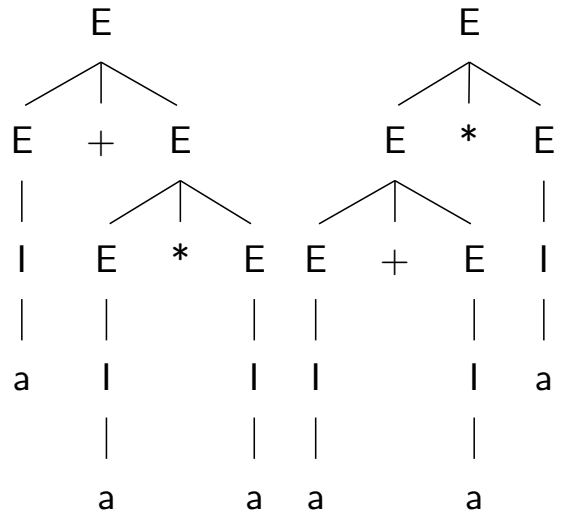
1. $E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + I \Rightarrow a + b$
2. $E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b$.

Definition 6.11 (Jednoznačnost a víceznačnost CFG)

- Bezkontextová gramatika $G = (V, T, P, S)$ je **víceznačná** pokud existuje aspoň jeden řetězec $w \in T^*$ pro který můžeme najít dva různé derivační stromy, oba s kořenem S dávající slovo w.
- V opačném případě nazýváme gramatiku **jednoznačnou**.
- Bezkontextový jazyk L je **jednoznačný**, jestliže existuje jednoznačná CFG G tak, že $L = L(G)$.
- **Bezkontextový jazyk L je (podstatně) nejednoznačný**, jestliže každá CFG G taková, že $L = L(G)$, je nejednoznačná. Takovému jazyku říkáme i **víceznačný**.

Example 6.11 (nejednoznačnost CFG)

Dva derivační stromy dávající $a + a * a$ ukazující víceznačnost gramatiky.

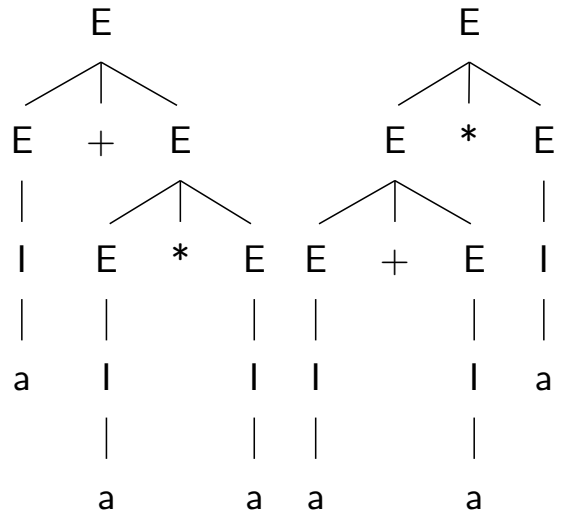


Definition 6.11 (Jednoznačnost a víceznačnost CFG)

- Bezkontextová gramatika $G = (V, T, P, S)$ je **víceznačná** pokud existuje aspoň jeden řetězec $w \in T^*$ pro který můžeme najít dva různé derivační stromy, oba s kořenem S dávající slovo w.
- V opačném případě nazýváme gramatiku **jednoznačnou**.
- Bezkontextový jazyk L je **jednoznačný**, jestliže existuje jednoznačná CFG G tak, že $L = L(G)$.
- **Bezkontextový jazyk L je (podstatně) nejednoznačný**, jestliže každá CFG G taková, že $L = L(G)$, je nejednoznačná. Takovému jazyku říkáme i **víceznačný**.

Example 6.11 (nejednoznačnost CFG)

Dva derivační stromy dávající $a + a * a$ ukazující víceznačnost gramatiky.

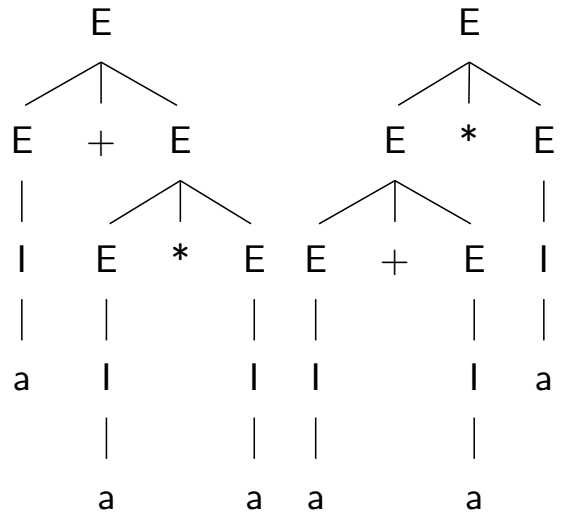


Definition 6.11 (Jednoznačnost a víceznačnost CFG)

- Bezkontextová gramatika $G = (V, T, P, S)$ je **víceznačná** pokud existuje aspoň jeden řetězec $w \in T^*$ pro který můžeme najít dva různé derivační stromy, oba s kořenem S dávající slovo w.
- V opačném případě nazýváme gramatiku **jednoznačnou**.
- Bezkontextový jazyk L je **jednoznačný**, jestliže existuje jednoznačná CFG G tak, že $L = L(G)$.
- **Bezkontextový jazyk L je (podstatně) nejednoznačný**, jestliže každá CFG G taková, že $L = L(G)$, je nejednoznačná. Takovému jazyku říkáme i **víceznačný**.

Example 6.11 (nejednoznačnost CFG)

Dva derivační stromy dávající $a + a * a$ ukazující víceznačnost gramatiky.

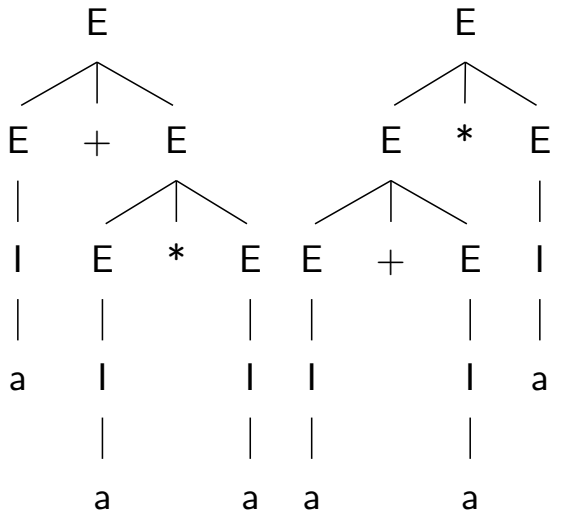


Definition 6.11 (Jednoznačnost a víceznačnost CFG)

- Bezkontextová gramatika $G = (V, T, P, S)$ je **víceznačná** pokud existuje aspoň jeden řetězec $w \in T^*$ pro který můžeme najít dva různé derivační stromy, oba s kořenem S dávající slovo w.
- V opačném případě nazýváme gramatiku **jednoznačnou**.
- Bezkontextový jazyk L je **jednoznačný**, jestliže existuje jednoznačná CFG G tak, že $L = L(G)$.
- **Bezkontextový jazyk L je (podstatně) nejednoznačný**, jestliže každá CFG G taková, že $L = L(G)$, je nejednoznačná. Takovému jazyku říkáme i **víceznačný**.

Example 6.11 (nejednoznačnost CFG)

Dva derivační stromy dávající $a + a * a$ ukazující víceznačnost gramatiky.



Příklad víceznačného jazyka

Example 6.12 (Víceznačný jazyk)

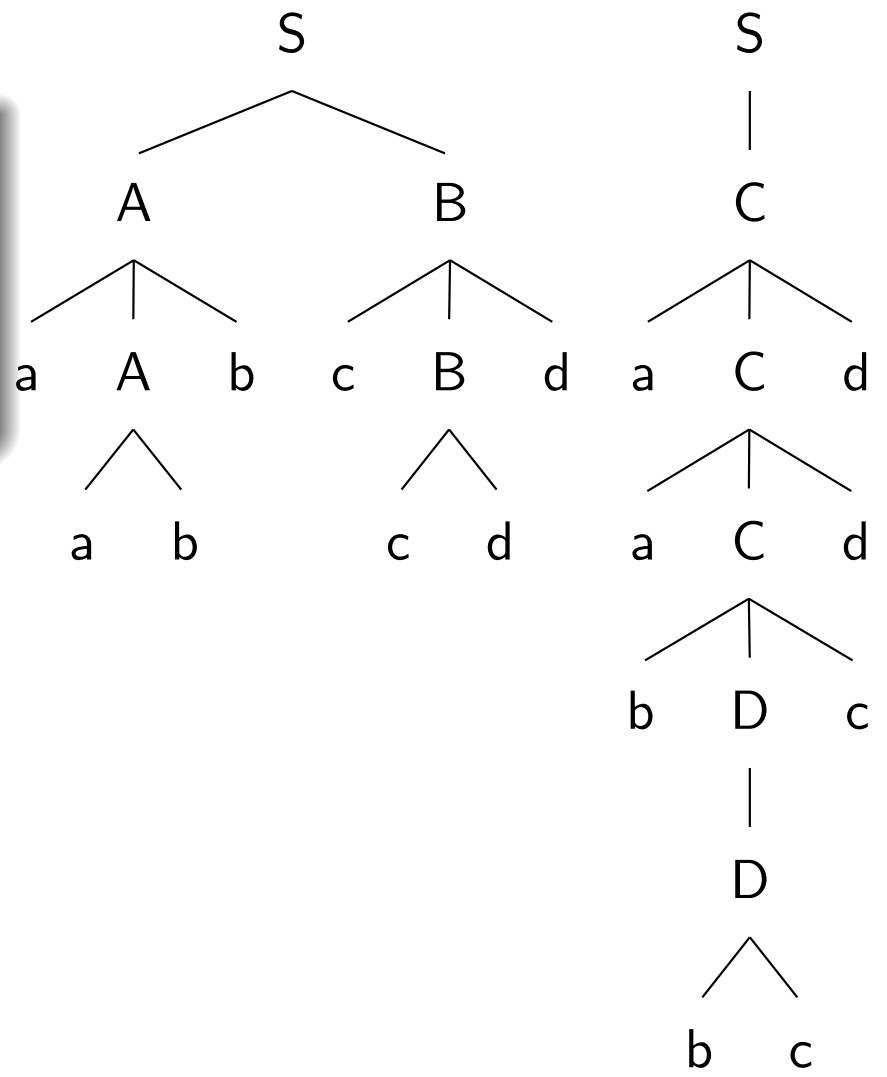
Příklad víceznačného jazyka:

$$L = \{a^n b^n c^m d^m | n \geq 1, m \geq 1\} \cup \cup \{a^n b^m c^m d^n | n \geq 1, m \geq 1\}.$$

1. $S \to AB|C$
2. $A \to aAb|ab$
3. $B \to cBd|cd$
4. $C \to aCd|aDd$
5. $D \to bDc|bc$.

Jakákoli gramatika pro daný jazyk bude generovat pro některá slova typu $a^n b^n c^n d^n$ dva různé derivační stromy.

Dva derivační stromy pro *aabbccdd*.



Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:

  - syntaktická chyba (Algol 60)
  - else patří k nejbližšímu if (preference pořadí pravidel)
  - závorky begin-end, odsazení v Python (ani nejsou součástí)

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$
  slovo 'if then if then else' má dva významy
  'if then (if then else)' nebo 'if then (if then) else'
  Řešení:
  - syntaktická dvojba (Algol 60)
  - else patří k nejbližšímu if (preferuje pořadí pravidel)
  - závorky begin-end, odsazení v Python (asi nejvíce čitelné)

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:

  - syntaktická dvojba (Algol 60)
  - blíže patří k bližšímu if (preference pořadí pravidel)
  - závorky begin-end, odsazení v Python (jasně nejvíce čitelné)

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:

  - syntaktická dvojba (Algol 60)
  - else patří k nejbližšímu if (preference pořadí pravidel)
  - závorky begin-end, odsazení v Python (jak nejlépe řešit)

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:

  - syntaktická chyba (Algol 60)
  - else patří k bližšímu if (preferuje se první pravidlo)
  - závorky begin-end, odsazení v Pythonu (zde nevidíte odsazení)

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:

  - syntaktická chyba (Algol 60)
  - else patří k nejbližšímu if (preference pořadí pravidel)
  - závorky begin-end, odsazení v Pythonu (pro nekvalifikované)

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:

  - syntaktická chyba (Algol 60)
  - else patří k nejbližšímu if (preference shora dolů pravidel)
  - závorky begin-end, odsazení v Pythonu (zavedení nového jazyka)

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:
  - syntaktická chyba (Algol 60)
  - else patří k nejbližšímu if (přednastavená sémantika pravidel)
  - závorky begin-end, odsazení v Pythonu (programátorská disciplína)

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:
  - syntaktická chyba (Algol 60)
  - else patří k bližšímu if (preference pořadí pavidel)
  - závorky begin-end,odsazení v Python (asi nejčistší řešení).

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:
  - syntaktická chyba (Algol 60)
  - else patří k bližšímu if (preference pořadí pavidel)
  - závorky begin-end,odsazení v Python (asi nejčistší řešení).

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:
  - syntaktická chyba (Algol 60)
  - else patří k bližšímu if (preference pořadí pavidel)
  - závorky begin-end,odsazení v Python (asi nejčistší řešení).

Odstanění víceznačnosti gramatiky

- Neexistuje algoritmus, který nám řekne, zda je daná gramatika víceznačná.
- Existují bezkontextové jazyky, pro které neexistuje jednoznačná bezkontextová gramatika, pouze víceznačné CFG.
- Existují určitá doporučení pro odstranění víceznačnosti.

Víceznačnost má různé příčiny:

- Není respektovaná priorita operátorů.
- Posloupnost identických operátorů lze shlukovat zleva i zprava.
- $S \rightarrow$ if then S else $S|$ if then $S|\epsilon$

  slovo 'if then if then else' má dva významy

  'if then (if then else)' nebo 'if then (if then) else'

  Řešení:
  - syntaktická chyba (Algol 60)
  - else patří k bližšímu if (preference pořadí pavidel)
  - závorky begin-end,odsazení v Python (asi nejčistší řešení).

Vynucení priority

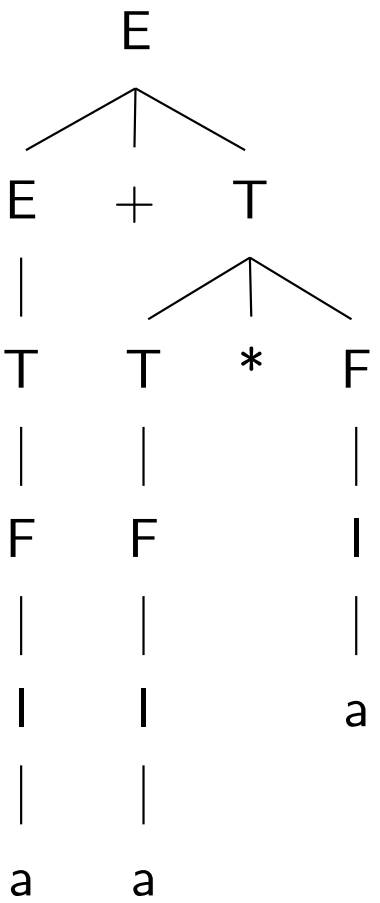
Řešením je zavést více různých proměnných, každou pro jednu úroveň 'priority'.
Konkrétně:

- **Faktor** je výraz který nesmí rozdělit žádný operátor.
  - identifikátory
  - výraz v závorkách
- **Term** je výraz, který nemůže rozdělit operátor $+$.
- **Výraz** může být rozdělen $*$ i $+$.

Jednoznačná gramatika pro výrazy:

1. $I \to a|b|Ia|Ib|I0|I1$
2. $F \to I|(E)$
3. $T \to F|T * F$
4. $E \to T|E + T$.

Jediný derivační strom pro $a + a * a$.



Jednoznačnost a kompilátory

Kompilace výrazu (zásobník na mezivýsledky + dva registry):

(1) $E \to E + T$ ... pop r1; pop r2; add r1,r2; push r2
(2) $E \to T$
(3) $T \to T * F$ ... pop r1; pop r2; mul r1,r2; push r2
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to a$ ... push a

- 'a+a*a' získáme postupnou aplikací pravidel 1,2,4,6,3,4,6,6
- posloupnost obrátíme a vybereme pouze pravidla generující kód

  6,6,3,6,1
- nyní nahradíme pravidla příslušným kódem

  push a; push a; pop r1; pop r2; mul r1,r2; push r2; push a; pop r1; pop r2; add r1,r2; push r2

Shrnutí

- Gramatiky
  - obecné
  - kontextové
  - bezkontextové
  - regulární, pravé lineární
- jazyk gramatiky, derivace, derivace dává slovo, derivační strom (pro bezkontextové gramatiky), ekvivalentní gramatiky
- ne každá lineární gramatika má ekvivalentní pravou lineární
- bezkontextové gramatiky
- jednoznačné a (podstatně) víceznačné gramatiky.

Přehled kapitol

1. Úvod, Iterační lemma pro reg. jazyky
2. Redukovaný DFA a ekvivalence automatů, stavů
3. Nedeterministické ϵ–NFA, Operace zachovávající regularitu
4. Regulární výrazy, Kleeneova věta, Substituce, Homomorfizmus
5. Dvousměrné FA, Mealy a Moore stroje
6. Gramatiky, Chomského hierarchie, víceznačnost
7. Chomského NF, Pumping Lemma pro CFL
8. CYK – náležení do CFL
9. Zásobníkové automaty, Deterministické PDA
10. Uzávěrové vlastnosti, Dykovy jazyky
11. Turingův stroj, rozšíření
12. Lineárně omezené automaty, Univerzální TM, Diagonální jazyk
13. Nerozhodnutelné problémy, Postův korespondenční p.
14. Časová složitost