UTEI

Urban Tadeáš, Cichra Radek

August 2023

Předmluva

Vážení studenti,
na základě našich vlastních zkušeností jsme pro Vás připravili tuto učební pomůcku, která je psána především pro praktické využití. Text si nečiní ambice obsáhnout celou problematiku automatů a proto jej, prosím, považujte za jeden, nikoli však jediný, zdroj informací nutných k přípravě na zkoušku z UTEI.
Děkujeme.

Přejeme Vám hodně úspěchů ve studiu!

Obsah

Formální abeceda, Formální jazyk, Práce s formálními jazyky

1.1 (Formální) abeceda

Definice

Jako abecedu definujeme **libovolnou konečnou množinu prvků Σ**.
Prvky abecedy nejčastěji nazýváme symboly, znaky, nebo písmena.

Příklady

$\mathbf{\Sigma_{bin}} = \{\mathbf{0}, \mathbf{1}\}$ | abeceda binární soustavy

$\mathbf{\Sigma_{dec}} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}\}$ | abeceda desítkové soustavy

$\mathbf{\Sigma_{ASCII}} = \{< \mathbf{SPACE} >, !, '', \#, \$, \%, \ldots\ldots\ldots, \mathbf{y}, \mathbf{z}, \{, \ , \}, < \mathbf{DEL} >\}$ | abeceda printable znaků ASCII standardu

1.2 Slovo

Definice

Slovo **w** nad abecedou **Σ** definujeme jako libovolnou konečnou posloupnost znaků z abecedy **Σ**.
Znaky ve slově se mohou opakovat.

Slovo neobsahující žádný znak nazveme **prázdné slovo**, dle konvence označované symbolem ε.

Příklady

Pro $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$ je slovo $\mathbf{w} = \mathbf{100110}$ slovem nad **Σ** (pro všechny znaky ve **w** platí, že $\in \mathbf{\Sigma}$).

Naopak pro $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$ slovo $\mathbf{u} = \mathbf{38}$(w v desítkové soustavě) není slovem nad **Σ**.

1.3 Konvence značení abeced a slov

Σ^* množina všech slov nad abecedou Σ | Σ^* = { w | w je slovem nad Σ }

Σ^+ množina všech slov and abecedou Σ bez prázdného slova ($\Sigma^* \backslash \{\varepsilon\}$)

$|\mathbf{w}|$ délka slova **w**

$|\mathbf{w}|_x$ počet výskytů znaku **x** ve slově **w**

$\mathbf{a^n}$ kompaktní zápis řetězce opakujících se znaků za sebou | $\mathbf{a^3}$ = **aaa**

1.4 Operace se slovy

Zřezězení

Zřetězení slov $\mathbf{u} = \mathbf{u_1 u_2 \ldots u_n}$ a $\mathbf{v} = \mathbf{v_1 v_2 \ldots v_m}$ značíme **u.v** (nebo jen **uv**) a definujeme jako připojení **v** za **u**, neboli $\mathbf{u.v} = \mathbf{uv} = \mathbf{u_1 u_2 \ldots u_n v_1 v_2 \ldots v_m}$.

Logicky platí, že |u| + |v| = |uv|. Zřetězení je asociativní.

1.5 Prefix, Sufix, Podslovo

Prefix slovo **t** je prefixem slova **z**, pokud platí, že existuje slovo **u** takové, že platí **z** = **t**.**u**. Pokud z = FJFI, pak například t = FJ je prefix, protože existuje u = FI

Sufix slovo t je sufixem slova z, pokud platí, že existuje slovo u takové, že platí z = u.t Pokud z = FJFI, pak například t = FI je sufix, protože existuje u = FJ

Podslovo slovo t je podslovem slova z, pokud platí, že existuje slovo u a slovo v takové, že platí z = utv Pokud z = FJFI, pak například t = JF je podslovo, protože existují u = F a v = I

1.6 Jazyk

Definice

Jazyk **L** nad abecedou **Σ** je libovolná podmnožina slov nad **Σ**.(neboli $\mathbf{L} \subseteq \mathbf{\Sigma}^*$)

Příklady

Nechť **L** je jazykem všech neprázdných slov délky přesně 3 znaky nad abecedou $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$. Pak **L** bude obsahovat všechny permutace čísel obsahujících pouze čísla **0** a **1** o délce právě 3 cifer $\Rightarrow$ $\mathbf{L} = \{\mathbf{000}, \mathbf{111}, \mathbf{011}, \mathbf{001}, \mathbf{100}, \mathbf{110}, \mathbf{101}, \mathbf{010}\}$.

1.7 Operace s jazyky

Definice

Hlavní 2 operace, se kterými se v kontextu jazyků při příkladech setkáme, jsou:

Zřetězení Zřetězení jazyků **K** a **L** značíme **KL** a definujeme jako jazyk všech slov **w** tvaru $\mathbf{w} = \mathbf{uv}$, kde $\mathbf{u} \in \mathbf{K}$ a $\mathbf{v} \in \mathbf{L}$.
$\Rightarrow \mathbf{KL} = \{\mathbf{uv} \mid (\mathbf{u} \in \mathbf{K}) \wedge (\mathbf{v} \in \mathbf{L})\}$

Iterace Iteraci jazyka **L** značíme $\mathbf{L}^*$ a definujeme jako rekurentí zřetězení **L** se sebou $\Rightarrow \varepsilon, \mathbf{L}, \mathbf{LL}, \mathbf{LLL}, ..., \mathbf{L^n}$.

Příklady

Pro jazyky $\mathbf{K} = \{\mathbf{a}, \mathbf{ab}, \varepsilon\}$ a $\mathbf{L} = \{\mathbf{1}, \mathbf{3}\}$ bude $\mathbf{KL} = \{\mathbf{a1}, \mathbf{a3}, \mathbf{ab1}, \mathbf{ab3}, \mathbf{1}, \mathbf{3}\}$

Pro jazyk $\mathbf{L} = \{\mathbf{1}, \mathbf{0}\}$ bude $\mathbf{L}^* = \{\varepsilon, \mathbf{0}, \mathbf{1}, \mathbf{00}, \mathbf{11}, \mathbf{01}, \mathbf{10}, \mathbf{000}, \mathbf{111}, \mathbf{001}, \mathbf{011}, ...\}$
$\Rightarrow$ nekonečná (ale spočetná) množina všech kladných celých čísel vyjádřených ve dvojkové soustavě.

Konečné automaty

2.1 Konečný automat

Definice

Konečný automat (značíme KA) definujeme jako uspořádanou pětici $\mathbf{A} = \{\mathbf{Q}, \mathbf{\Sigma}, \delta, \mathbf{q_0}, \mathbf{F}\}$, kde:

$\boxed{\mathbf{Q}}$ Konečná neprázdná množina stavů

$\boxed{\mathbf{\Sigma}}$ Vstupní abeceda

$\boxed{\delta}$ Přechodová funkce pro dané znaky abecedy mezi stavy | $\mathbf{Q} \times \mathbf{\Sigma} \rightarrow \mathbf{Q}$

$\boxed{\mathbf{q_0}}$ Počáteční stav KA | $\mathbf{q_0} \in \mathbf{Q}$

$\boxed{\mathbf{F}}$ množina koncových stavů KA | $\mathbf{F} \subseteq \mathbf{Q}$

KA může "zpracovat" slovo $\mathbf{v} \in \mathbf{\Sigma}^*$ a následně rozhodnout, jestli toto slovo přijímá, nebo ne.

Zpracováním slova se rozumí to, že automat začné ve svém počátečním stavu $\mathbf{q_0}$ a čte jednotlivé znaky vstupního slova, přičemž se řídí přechodovou funkcí δ. Přechodová funkce definuje v každém stavu přechod do dalšího stavu podle toho, jaký znak zrovna čte.

Pokud se po zpracování celého slova automat nachází v nějakém stavu z množiny koncových stavů ($\in \mathbf{F}$) $\rightarrow$ KA slovo **akceptuje**. V opačném případě ($\notin \mathbf{F}$) slovo **neakceptuje**.

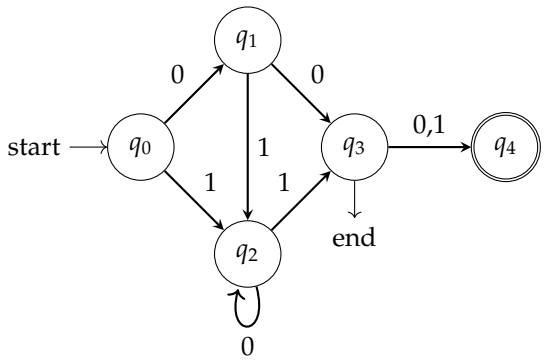
2.2 Graf KA (Stavový diagram)

Definice

Pro intuitivnější vyobrazení a vyjádření struktury a funkce KA používáme **stavový diagram**. Ten ukazuje množinu stavů $\mathbf{Q}$ jako vrcholy grafu a členy přechodové funkce δ jako hrany mezi vrcholy. Počáteční stav $\mathbf{q_0}$ je vyobrazen jako vrchol, do kterého směřuje hrana z prázdna. Koncové stavy množiny $\mathbf{F}$ jsou vrcholy grafu v kruhu (někdy také vrcholy, ze kterých vychází hrana do prázdna).

Příklad

Následuje příklad stavového diagramu KA, jehož abeceda je $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$. Stav $\mathbf{q_0}$ je vstupní stav. Stavy $\mathbf{q_3}$ a $\mathbf{q_4}$ ukazují dva nejčastější způsoby vyobrazení stavů koncových $\in \mathbf{F}$.



2.3 Deterministický a nedeterminisitcký KA (DKA a NDKA)

Definice

Podle požadavků kladených na přechodovou funkci δ můžeme rozdělit KA na :

Deterministický KA Z každého stavu by měl být definován právě jeden přechod pro každý znak ze vstupní abecedy $\mathbf{\Sigma}$.

Toto často striktně neplatí kvůli potenciálně obrovskému množství ($\mathbf{Q}.\mathbf{\Sigma}$) hran ve statovém diagramu. Tento problém se někdy řeší tzv. **error** nebo **trap stavem**, do kterého automat přejde, jakmile nemá definovaný přechod pro daný znak z daného stavu.

$\Rightarrow$ pro dané slovo $\mathbf{v} \in \mathbf{\Sigma}^*$ existuje právě jedna cesta průchodu stavovým diagramem automatu určená jeho znaky (proto deterministický).

Nedeterministický KA Z každého stavu může existovat více než jeden přechod pro daný znak.

$\Rightarrow$ pro dané slovo $\mathbf{v} \in \mathbf{\Sigma}^*$ může existovat více možných průchodů a ne všechna musí skončit v nějákém koncovém stavu.

$\Rightarrow$ pro NDKA platí, že slovo akceptuje $\iff$ existuje minimálně jeden průchod stavovým grafem končící v nějakém koncovém stavu.

2.4 Jazyk akceptovaný automatem

Definice

Jazyk automatu **A** definujeme jako množinu všech slov, která automat akceptuje, neboli $\mathbf{L(A)} = \{\mathbf{v} \mid (\mathbf{v} \in \Sigma^*) \wedge (\mathbf{A}$ **akceptuje** $\mathbf{v})\}$.

2.5 Ekvivalence automatů

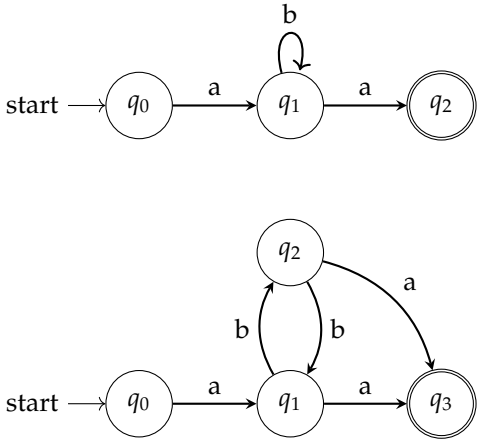
Definice

Automaty **A** a **B** jsou ekvivalentní, pokud akceptují právě stejný jazyk ($\mathbf{L(A)} = \mathbf{L(B)}$).

Platí, že **libovolný NDKA lze převést na ekvivalentní DKA**.

Příklad

Následující automaty jsou ekvivalentní ($\mathbf{L(A)} = \mathbf{L(B)} = \{\mathbf{ab^n a}\}$):



2.6 Minimální automat

Definice

Minimální automat definujeme jako **automat s nejmenším počtem stavů, který akceptuje daný jazyk**.

2.7 Minimalizace

Definice

Minimalizace automatu **A** definujeme jako jeho upravení na ekvivalentní minimální automat $\mathbf{A^{min}}$. Motivace za minimalizací je logicky jednodušší práce s automatem a jeho stavovým diagramem.

Algoritmus minimalizace

Minimalizace se skládá ze 3 kroků:

Odstranění nedosažitelných stavů Nedosažitelné stavy jsou stavy, do kterých nevedou žádné hrany ve stavovém diagramu ⇒ nelze se do nich dostat.

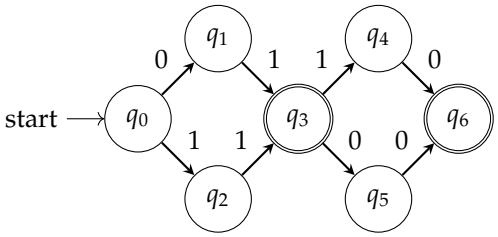
Odstranění nadbytečných stavů Nadbytečné stavy jsou stavy, do kterých sice vedou hrany, ale nelze se z nich dostat do koncových stavů diagramu.

Sloučení ekvivalentních stavů Ekvivalentní stavy jsou stavy, které generují stejný jazyk, pokud z nich uděláme počáteční stav automatu.

Nedosažitelné a nadbytečné stavy bývá snadné identifikovat a odstranit (v příkladech, se kterými se setkáme my, jsou zřejmé), proto jejich odstranění v popisu algoritmu přeskočíme.

Nalezení a sloučení ekvivalentních stavů

Nechť máme automat **A** popsaný tímto stavovým diagramem:



Tento diagram nemá žádné nedosažitelné ani nadbytečné stavy. Pro jeho minimalizaci je tedy třeba pouze nalézt a sloučit ekvivalentní stavy. To uděláme následovně:

1. Rozdělíme si množinu stavů **Q** na 2 množiny $\mathbf{S_1}$ a $\mathbf{S_2}$, kde $\mathbf{S_1} = \mathbf{F}$ a $\mathbf{S_2} = \mathbf{Q \backslash F}$.
   $\Rightarrow$
   $\mathbf{S_1} = \mathbf{F} = \{\mathbf{q_3}, \mathbf{q_6}\}$
   $\mathbf{S_2} = \mathbf{Q \backslash F} = \{\mathbf{q_0}, \mathbf{q_1}, \mathbf{q_2}, \mathbf{q_4}, \mathbf{q_5}\}$
2. Vytvoříme takzvanou **tabulku přechodů** → tabulka ukazující pro každý přechod z každého stavu, do jakého stavu vede:

Množina	**Ze stavu**	**0 vede do**	**1 vede do**
$\mathbf{S_1}$	q_3	q_5	q_4
	q_6	**X**	**X**
$\mathbf{S_2}$	q_0	q_1	q_2
	q_1	**X**	q_3
	q_2	**X**	q_3
	q_4	q_6	**X**
	q_5	q_6	**X**

3. V této tabulce nyní upravíme sloupce s přechody → místo cílového stavu uvedeme množinu, do které patří:

Množina	**Ze stavu**	**0 vede do**	**1 vede do**
$\mathbf{S_1}$	q_3	S_2	S_2
	q_6	**X**	**X**
$\mathbf{S_2}$	q_0	S_2	S_2
	q_1	**X**	S_1
	q_2	**X**	S_1
	q_4	S_1	**X**
	q_5	S_1	**X**

4. Přerozdělíme stavy do nových množin tak, že dáme dohromady stavy se stejnými množinami ve sloupcích s přechody:

Množina	**Ze stavu**	**0 vede do**	**1 vede do**
A	q_3	S_2	S_2
B	q_1	**X**	S_1
	q_2	**X**	S_1
C	q_6	**X**	**X**
D	q_0	S_2	S_2
E	q_4	S_1	**X**
	q_5	S_1	**X**

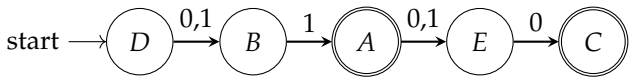
   (Pozn.: stavy $\mathbf{q_3}$ a $\mathbf{q_0}$ nejsou ve stejné množině, protože $\mathbf{q_3}$ je koncový stav, ale $\mathbf{q_0}$ ne)

5. Kroky 2 až 4 opakujeme, dokud lze stavy přerozdělovat do nových množin:

Množina	Ze stavu	0 vede do	1 vede do
A	q_3	E	E
B	q_1	**X**	A
	q_2	**X**	A
C	q_6	**X**	**X**
D	q_0	B	B
E	q_4	C	**X**
	q_5	C	**X**

   ⇒ nevzniklo nové rozdělení

6. Máme hotovo. Nyní stačí nakreslit stavový diagram dle tabulky se sloučenými stavy:



2.8 ε-přechody

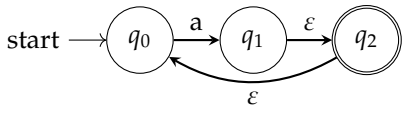
Definice

ε-přechod definujeme jako **přechod mezi stavy, při kterém se ze vstupního slova nečte žádný znak** ⇒ Automat může přejít mezi stavy bez čtení znaku ze vstupního slova.

ε-přechodů se můžeme zbavit pomocí ε-uzávěru a nebo při determinizaci.

Příklad

Následuje stavový diagram automatu **A**, pro který platí $\mathbf{L(A)} = \{\mathbf{a^n} \mid \mathbf{n} \in \mathbb{N}\}$



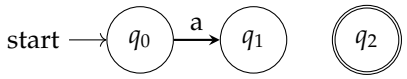
2.9 Odstranění ε-přechodů (ε-uzávěr)

Na příkladě z předchozí části si popíšeme postup vytvoření takzvaných ε-uzávěrů a jejich využití při odstraňování ε-přechodů. Postupujeme následovně:

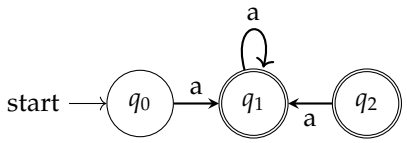
1. Pro každý stav automatu definujeme ε-uzávěr → **množina stavů, do kterých se z daného stavu dá dostat bez zpracování znaku vstupního slova** (neboli přes ε-přechody, vždy obsahuje aspoň stav samotný):

Uzávěr stavu	**Stavy v uzávěru**
$\varepsilon(q_0)$	q_0
$\varepsilon(q_1)$	q_1, q_2, q_0
$\varepsilon(q_2)$	q_2, q_0

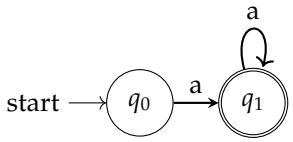
2. Překreslíme stavový diagram **BEZ** ε-přechodů:



3. Ke všem stavům diagramu doplníme přechody stavů z jejich ε-uzávěru (ne ε-přechody):



(Pozn.: $\mathbf{q_1}$ se stává koncovým stavem, neboť v jeho uzávěru byl koncový stav $\mathbf{q_2}$, $\mathbf{q_2}$ je nedosažitelný)
⇒ Automat je ekvivalentní s:

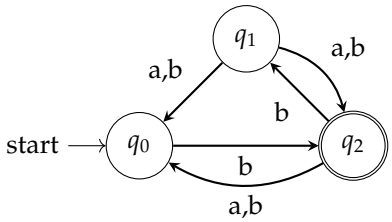


Tím máme hotovo. Stále platí $\mathbf{L(A)} = \{\mathbf{a^n} \mid \mathbf{n} \in \mathbb{N}\}$.

2.10 Determinizace

Definice

Determinizace je proces převodu NDKA na ekvivalentní DKA. Postup determinizace si ukážeme na příkladu. Nechť máme následující stavový diagram popisující automat **A**:

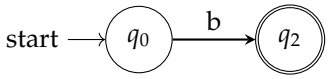


Automat **A** determinizujeme následovně:

1. Začneme u vstupního stavu a pro každý znak vytvoříme množinu stavů, do kterých se počátečního stavu jejich čtením dostaneme:

Ze stavu	**a vede do**	**b vede do**
q_0	**X**	q_2

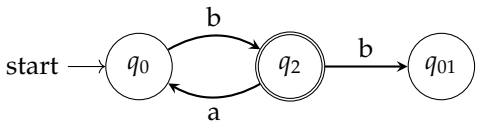
   Zatím tedy máme:

2. opakujeme krok 1 pro $\mathbf{q_2}$ a budeme opakovat pro další nově vzniklé stavy, dokud budou vznikat:

Ze stavu	**a vede do**	**b vede do**
q_2	q_0	q_0, q_1

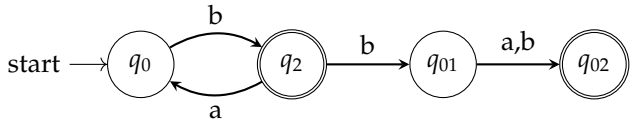
   Doplníme nový stav množiny $\{\mathbf{q_0}, \mathbf{q_1}\}$ jako $\mathbf{q_{01}}$ a přechody:

Vznikl nový stav ⇒ aplikujeme na něj krok 1 (budeme se koukat na přechody ze všech stavů v množině stavu $\mathbf{q_{01}}$):

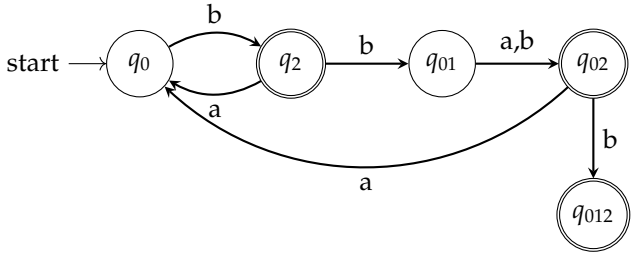
Ze stavu	**a vede do**	**b vede do**
q_{01}	q_0, q_2	q_0, q_2

⇒



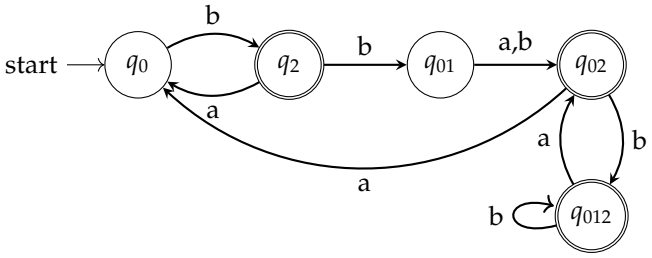
Ze stavu	**a vede do**	**b vede do**
q_{02}	q_0	q_0, q_1, q_2

⇒



Ze stavu	**a vede do**	**b vede do**
q_{012}	q_0, q_2	q_0, q_1, q_2

⇒



Již nevznikly nové stavy ⇒ máme hotovo. Můžeme si všimnout, že automat je nyní opravdu deterministický.

REGEX

3.1 Regulární výrazy

Definice

Regulární výrazy jsou mocným nástrojem pro popis vzoru nebo určité množiny textových řetězců. Používají se při vyhledávání v textu a vyjádření vzoru, podle kterého řetězec hledat. V kontextu KA jsou regulární výrazy důležité, protože generují takzvané regulární jazyky. Regulární jazyky jsou právě ty stejné jazyky, které mohou generovat a rozpoznávat KA. Tím se pro nás stávají alternativním vyjádřením těchto jazyků. Formálně bychom regulární výraz definovali asi nějak takto:

Nechť existuje abeceda $\mathbf{\Sigma}$. Pak nad $\mathbf{\Sigma}$ definujeme:

$\boxed{\theta}$ Regulární výraz označující prázdný jazyk

$\boxed{\varepsilon}$ Regulární výraz označující jazyk $\{\varepsilon\}$

$\boxed{\mathbf{a}}$ Regulární výraz označující jazyk {a}, kde $\mathbf{a} \in \mathbf{\Sigma}$

Nechť existují libovolné regulární výrazy α, β. Pak platí, že i $(\alpha + \beta)$, $(\alpha \cdot \beta)$ a (α^*) jsou regulární výrazy, kde:

$\boxed{(\alpha + \beta)}$ Operace sjednocení jazyků $\{\alpha\}$ a $\{\beta\}$ (je asociativní)

$\boxed{(\alpha \cdot \beta)}$ Operace zřetězení $\{\alpha\}$ a $\{\beta\}$ (často bez $'\cdot' \rightarrow \alpha\beta$, je asociativní)

$\boxed{(\alpha^*)}$ Operace iterace $\{\alpha\}$

Tyto pravidla nám dovolují popsat celou množinu slov akceptovaných KA (neboli jeho jazyk) popsáním souvislostí ve formátu slov daného jazyka.

Příklady

a(a + b)b První regulární výraz '**a**' nám udává, že všechna slova v generovaném jazyce **J** budou začínat písmenem z jazyka {a}. Druhý regulární výraz '**(a + b)**' udává druhý znak slov v **J**, a to sice znak ze sjednocení jazyků {a} ∪ {b} → {a,b}. Analogicky k prvnímu znaku, poslední reg. výraz udává všem slovům z **J** koncovku **b**. Dohromady tedy můžeme určit, že všechna slova v **J** budou mít délku 3 znaků, přičemž jejich prefix je '**a**', sufix '**b**' a mezi nimi je znak z jazyka {a,b} ⇒ Výraz generuje **J** = {**aab**, **abb**}.

a(b + ε)c Výraz řešíme analogicky k prvnímu příkladu. Rozdíl je různá délka generovaných slov kvůli druhému regulárnímu výrazu '(**b** + ε)'. Podle něj budou mít slova generovaného jazyka **J** na druhé pozici buď znak z jazyka {b} nebo {ε} ⇒ **J** = {**abc**, **a**ε**c** = **ac**}.

(**a***)**b**(**c***) První a poslední reg. výraz jsou iterace jazyků {a} a {c}. Z definice Iterace jazyka víme, že se jedná o rekurentní zřetězení:
⇒ (**a***) = {ε, **a**, **aa**, ..., **a**$^{\mathbf{n}}$} a (**c***) = {ε, **c**, **cc**, ..., **c**$^{\mathbf{n}}$}

Prostřední výraz '**b**' je jasný ⇒ Jazyk **J** bude tedy obsahovat slova obsahující prázdný nebo libovolně dlouhý prefix znaků '**a**' následovaný právě jedním znakem '**b**' a prázdným nebo libovolně dlouhým sufixem znaků '**c**' ⇒ **J** = {**b**, **ab**, **bc**, **aab**, **abc**, **bcc**, **aaab**, **aabc**, **abcc**, **bccc**, ...}

3.2 Převod KA na regulární výraz (eliminace stavů)

Definice

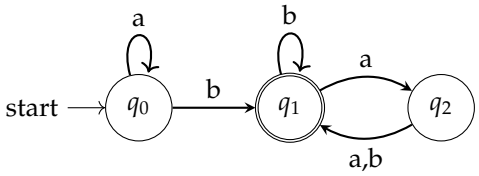
Jak bylo již řečeno, existuje vztah mezi jazykem generovaným KA a regulárními výrazy. Platí totiž, že **oba mohou generovat stejné jazyky** (regulární jazyky). Logicky tedy **lze KA převést na regulární výraz** (a naopak).

Následuje postup převodu KA na regulární výraz skrze takzvanou **eliminaci stavů**. Tato metoda je spíše intuitivní než algoritmická, ale pro naše účely postačuje. Pro doplnění je vhodné uvést, že stejného lze dosáhnout metodou regulárních rovnic. Postup pro převod KA je následující:

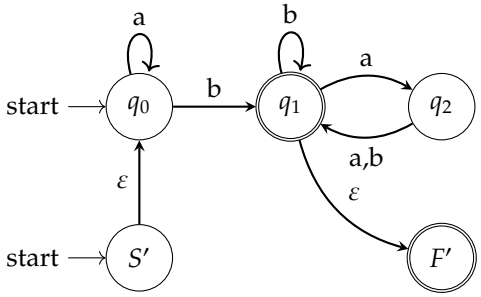
1. Vytvoříme dva nové stavy - počáteční **S'** a koncový **F'**.
   (a) ze stavu **S'** povedeme ε-přechody do všech počátečních stavů automatu.
   (b) do stavu **F'** povedeme ε-přechody ze všech koncových stavů automatu.
2. Postupně eliminujeme stavy automatu (v libovolném pořadí) a nahrazujeme zaniklé přechody ekvivalentními reg. výrazy.
3. Konec převodu nastane, když máme pouze stavy **S'** a **F'**. Hrana mezi těmito stavy bude regulární výraz ekvivalentní s původním KA.

Příklad

Ukažme si převod automatu **A** s následujícím stavovým diagramem:



1. Přidáme nové stavy **S′** a **F′** a připojíme je:



2. Začneme odstraňovat stavy a nahrazovat hrany. První odstraníme stav $\mathbf{q_0} \rightarrow$ je třeba přidat hrany, které nahradí přechody přes $\mathbf{q_0}$:

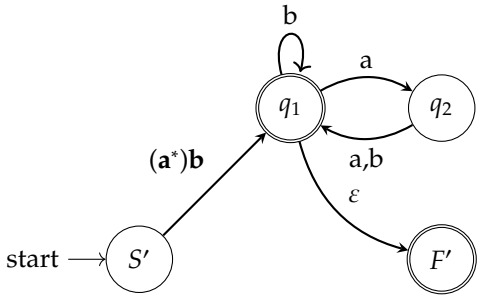
   - Do $\mathbf{q_0}$ přichází pouze 1 hrana 'ε' ze stavu $\mathbf{S'}$.
   - Z $\mathbf{q_0}$ naopak odchází 2 hrany '$\mathbf{a}$' zpět do $\mathbf{q_0}$ a '$\mathbf{b}$' do $\mathbf{q_1}$.

   Jelikož do $\mathbf{q_0}$ vede hrana s prázdným slovem 'ε' $\rightarrow$ v regulárním výrazu se nijak neprojeví.

   Zamysleme se nyní nad tím, jak funguje smyčka (hrana $\mathbf{q_0} \rightarrow \mathbf{q_0}$) v automatu. Slovo akceptované automatem $\mathbf{A}$ může začínat prázdným nebo libovolně dlouhým sufixem znaků '$\mathbf{a}$', neboť při zpracování 'a' zůstáváme ve stavu $\mathbf{q_0}$ a posouváme se až po přečtení znaku '$\mathbf{b}$' $\rightarrow$ sufix akceptovaných slov může být tvaru $\{\varepsilon, \mathbf{a}, \mathbf{aa}, ..., \mathbf{a^n}\}$. Nyní je snad zřejmá ekvivalence s reg. výrazem ($\mathbf{a}^*$).

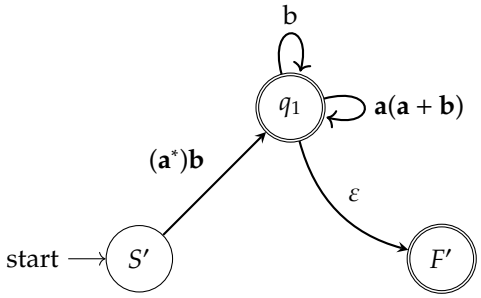
   Hrana $\mathbf{q_0} \rightarrow \mathbf{q_1}$ se symbolem '$\mathbf{b}$' je logicky ekvivalentní s regulárním výrazem ($\mathbf{b}$).

   Po odstranění stavu $\mathbf{q_0}$ tedy spojíme $\mathbf{S'} \rightarrow \mathbf{q_1}$ hranou s regulárním výrazem $(\mathbf{a}^*)\mathbf{b}$ (pozor na správné pořadí!), čímž pokryjeme jak smyčku $\mathbf{q_0} \rightarrow \mathbf{q_0}$, tak přechod $\mathbf{q_0} \rightarrow \mathbf{q_1}$.



3. Odstraňujeme stavy, dokud nezůstane pouze $\mathbf{S'} \rightarrow \mathbf{F'}$. Odstraníme tedy $\mathbf{q_2}$.

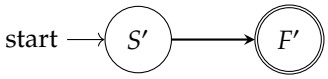
   - Do $\mathbf{q_2}$ vede 1 hrana se znakem '$\mathbf{a}$' $\rightarrow$ ($\mathbf{a}$).
   - Z $\mathbf{q_2}$ vedou 2 hrany do $\mathbf{q_1}$. Je důležité si uvědomit, že při přechodu $\mathbf{q_2} \rightarrow \mathbf{q_1}$ se přečte jen jeden znak $\rightarrow$ proto ($\mathbf{a} + \mathbf{b}$), ne ($\mathbf{ab}$).



Poslední odstraníme $\mathbf{q_1}$.

- z $\mathbf{q_1}$ vedou 3 hrany, z čehož 2 jsou smyčky a poslední je ε-přechod do $\mathbf{F'}$.

Jak jsme si již ukázali, smyčky jsou ekvivalentní s iterací → máme $(\mathbf{b}^*)$ a $(\mathbf{a}(\mathbf{a}+\mathbf{b}))^*$. Obě smyčky končí ve stejném stavu → akceptovaná slova je mohou nejenom libovolně opakovat, ale také střídat:
$\rightarrow ((\mathbf{b}^*)+(\mathbf{a}(\mathbf{a}+\mathbf{b}))^*)^* = (\mathbf{b}+(\mathbf{a}(\mathbf{a}+\mathbf{b})))^* = \boxed{(\mathbf{b}+(\mathbf{aa}+\mathbf{ab}))^*}$
Vnější iterací výrazu jsme zároveň pokryli možnost ε-přechodu $\mathbf{q_1} \rightarrow \mathbf{F'}$.



$(\mathbf{a}^*)\mathbf{b}(\mathbf{b}+(\mathbf{aa}+\mathbf{ab}))^*$

Tím máme hotovo. Regulární výraz na hraně $\mathbf{S'} \rightarrow \mathbf{F'}$ generuje stejný jazyk jako původní automat **A**.

3.3 Lemma o vkládání (Pumping lemma)

Definice

Pro libovolný **regulární** (akceptovaný konečným automatem) jazyk **L** existuje nějaké číslo $\mathbf{p} > \mathbf{0}$ takové, že pro každé slovo $(\mathbf{w} \in \mathbf{L}) \wedge (|\mathbf{w}| \geq \mathbf{p})$ platí následující:

1. **w** lze zapsat jako $\mathbf{w} = \mathbf{xyz}$
2. pro slova **x**, **y**, **z** platí, že:
   (a) $|\mathbf{xy}| \leq \mathbf{p}$
   (b) $|\mathbf{y}| > \mathbf{0}$
   (c) $(\mathbf{xy^iz}) \in \mathbf{L} \mid \mathbf{i} \geq \mathbf{0}$

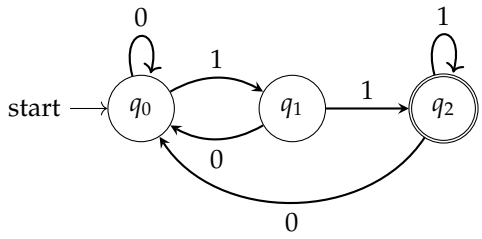
Tato definice nám říká, že v regulárním jazyce platí následující:

Dostatečně dlouhé ($|\mathbf{w}| \geq \mathbf{p}$) slovo z tohoto jazyka můžeme rozdělit takovým způsobem, který nám dovoluje libovolně opakovat jeho část ($\mathbf{y^i}$), aniž by nová slova přestala ležet v daném jazyce.

V praxi **Pumping lemma používáme při důkazu sporem, kdy chceme ukázat, že nějaký jazyk NENÍ regulární**.

Příklady

Ukažme si lemma na jazyce **L**, který obsahuje všechna slova nad abecedou $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, která končí na '**11**'. Víme, že **L** je regulární, protože můžeme celkem snadno navrhnout automat, který ho akceptuje:



Nyní musíme určit číslo **p**, pro které budeme hledat části slov k opakování. Možná už lze vydedukovat, že vhodné číslo by mohlo být 3, tedy počet stavů našeho automatu. Je tomu tak proto, že opakovaní části slov bude v automatu ekvivalentní s opakováním nějaké smyčky nebo řady opakujících se přechodů. Další důvod je to, že pokud bude mít slovo délku ≥ počet stavů automatu, nutně muselo nějakým stavem projít víckrát → potenciální část slova k opakování.

Zkusme se tedy podívat na nějaká slova délky ≥ 3:

w = 011 Existuje hned několik způsobů, jak toto slovo správně "napasovat" do tvaru $\mathbf{w} = \mathbf{xyz}$. Intuitivní je zkusit $\mathbf{x} = \mathbf{0}, \mathbf{y} = \mathbf{1}, \mathbf{z} = \mathbf{1}$. Pokud zkusíme $\mathbf{y} \to \mathbf{y^i}$, vidíme, že $\mathbf{xy^iz} = \mathbf{01^i1}$. Toto je skoro dobře, ale problém nastává pro $\mathbf{i} = \mathbf{0} \to \mathbf{xy^0z} = \mathbf{01} \notin \mathbf{L}$.

Zkusme se podívat na průchod stavovým diagramem při čtení slova **w**. Vidíme, že první znak **w** je přečten při průchodu smyčkou. zkusme tedy napasovat $\mathbf{y} = \mathbf{0}$. Potom nutně $\mathbf{x} = \varepsilon, \mathbf{z} = \mathbf{11} \to \mathbf{xy^iz} = \mathbf{0^i11}$. Zde již vše vyhovuje, neboť $\mathbf{0^i11} \in \mathbf{L}$ pro libovolné **i**.

w = 1011 Při zpracování tohoto slova automat projde potenciálně se opakující řadou stavů $\mathbf{q_0} \to \mathbf{q_1} \to \mathbf{q_0}$ při zpracování prvních dvou znaků '**10**'. Nechť tedy $\mathbf{y} = \mathbf{10} \to \mathbf{x} = \varepsilon, \mathbf{z} = \mathbf{11}$. Tím pádem $\mathbf{xy^iz} = \mathbf{(10)^i11} \in \mathbf{L}$.

w = 0110011 Zde při průchodu automatem vidíme několik potenciálních úseků k opakování. Abychom opět neměli $\mathbf{x} = \varepsilon$, zkusíme opakovat smyčku $\mathbf{q_0} \to \mathbf{q_0}$ vzniklou při zpracování pátého znaku '**0**' slova. Nechť tedy $\mathbf{x} = \mathbf{0110}, \mathbf{y} = \mathbf{0}, \mathbf{z} = \mathbf{11} \to \mathbf{xy^iz} = \mathbf{01100^i11} \in \mathbf{L}$. Validní by bylo ovšem i jakékoliv jiné řešení splňující podmínky lemmatu.

Příklad důkazu

Výše uvedené příklady na regulárním jazyce sloužili k lepšímu porozumění lemmatu. Jak již ale bylo řečeno, v praxi toto lemma používáme k dokázání, že nějaký jazyk **NENÍ** regulární . Využíváme k tomu důkaz sporem, který si nyní ukážeme na příkladu:

Mějme jazyk **L** všech slov nad abecedou $\Sigma = \{1, 0\}$, která mají tvar $\mathbf{w} = 0^n 1^n$ pro $\mathbf{n} \geq 0$. Rozhodněte o regulárnosti **L**.

Jako první můžeme zkusit navrhnout automat, který by **L** generoval. Pak by **L** byl jednoznačně regulární jazyk. Žádný takový automat se nám ale navrhnout nepodaří, což je první důvod k podezření, že **L** není regulární.

Zkusme tedy použít Pumping lemma v důkazu sporem → Předpokládejme, že **L** je regulární ⇒ všechny slova délky ≥ **p** musí jít rozdělit na části, které splňují podmínky lemmatu. Nám stačí najít pouze jeden příklad, kdy toto neplatí, abychom sporem dokázali, že **L** není regulární.

Nechť máme slovo $\mathbf{w} = 0^p 1^p$, které logicky splňuje podmínku $|\mathbf{w}| \geq \mathbf{p}$. Slovo **w** můžeme rozdělit podle $\mathbf{w} = \mathbf{xyz}$ třemi způsoby:

$\boxed{\mathbf{x} = \varepsilon, \mathbf{y} = 0^p, \mathbf{z} = 1^p}$ Toto rozdělení nebude fungovat, protože $\mathbf{xy^iz} = 0^{p+i}1^p$. Potom ale $|0| \neq |1| \Rightarrow \mathbf{w} \notin \mathbf{L}$.

$\boxed{\mathbf{x} = 0^p, \mathbf{y} = 1^p, \mathbf{z} = \varepsilon}$ Analogicky k předchozí variantě → $\mathbf{xy^iz} = 0^p 1^{p+i} \notin \mathbf{L}$.

$\boxed{\mathbf{x} = \varepsilon, \mathbf{y} = 0^p 1^p, \mathbf{z} = \varepsilon}$ Může se zdát, že jsme našli řešení, protože $|0| = |1| = \mathbf{p} + \mathbf{i}$. Je ale třeba si uvědomit, že opakování **y** způsobí změnu tvaru slova, protože budeme přidávat řetězec '**01**' → budeme znaky střídat. Tvar nových slov bude $\mathbf{xy^iz} = 0^p 1^p (01)^i \notin \mathbf{L}$.

Našli jsme slovo **w** délky aspoň **p**, pro které lemma neplatí ⇒ Spor s naším předpokladem a tedy definitní důkaz, že **L není regulární**.

Bezkontextové gramatiky a jazyky

Tato kapitola se budě věnovat obecnějším jazykům, které nelze zachytit KA ($\Rightarrow$ ani reg. výrazy). Bezkontextové jazyky nám dovolují používat takzvaná **odvozovací pravidla** k dosažení komplexních syntaktických pravidel (například správný zápis, pořadí a uzavíraní závorek), která KA vytvořit nemohou. Setkáváme se s nimi při syntaktické analýze a rozboru kódu programovacího jazyka.

Bezkontextové jazyky jsou generované bezkontextovými gramatikami, které obsahují odvozovací pravidla pro generování slov jazyka.

4.1 Bezkontextová gramatika (BG)

Definice

Bezkontextová gramatika **G** je definovaná čtveřicí $\mathbf{G} = \{\mathbf{N}, \mathbf{T}, \mathbf{S}, \mathbf{P}\}$, kde:

N Množina **neterminálních znaků** (neterminálů) $\rightarrow$ neukončují generování slova.

T Množina **terminálních znaků** (terminálů) $\rightarrow$ ukončují generování slova, platí $\mathbf{N} \cap \mathbf{T} = \theta$.

S Počáteční neterminální znak, ze kterého začínáme generovat slova.

P Množina pravidel psaných '$\mathbf{A} \rightarrow \gamma$', kde $\mathbf{A} \in \mathbf{N}$, $\gamma \in (\mathbf{N} \cup \mathbf{T})^*$.

Při generování slov vždy začneme počátečním neterminálním znakem **S**. Pak podle toho, jaká máme pro **S** definovaná odvozovací pravidla v množině **P**, generujeme slovo přidáváním znaků z množiny **N**. Pro daný znak z **N** mohou být v **P** definovaná vlastní pravidla. Eventuálně generace skončí pravidlem generujícím terminální znak z **T**.

(Pozn.: BG mohou generovat i jazyky generované KA nebo regulárními výrazy, neboť BG generují obecnější množinu)

Příklad

Nechť máme bezkontextovou gramatiku **G**, kde **N** = {**S**, **A**}, **T** = {**a**, **b**, ε} a množina **P** obsahuje tyto pravidla:

S → **A**

A → **aAb** | ε (Pozn.: znak '|' odděluje různá pravidla definovaná pro stejný neterminální znak, aby byl zápis stručnější)

Podívejme se nyní, jak se z **G** generuje nějaké slovo **w** → začínáme počátečním znakem **S**, který má definované jediné pravidlo **S** → **A**. Tedy **w** = **S** ⇒ **A**. **A** není terminální znak, podíváme se tedy na jeho pravidla, která jsou dvě. Můžeme buď zvolit řetězec obsahující neterminální znak **aAb** a pokračovat v generování, nebo zvolit řetězec obsahující **pouze terminální** znak ε, čímž generování skončí. Zvolme první možnost, abychom neměli jen prázdné slovo. Tedy **w** = **S** ⇒ **A** ⇒ **aAb**. Nyní stojíme před stejným rozhodnutím. Pro stručnost zvolme ε. Pak **w** = **S** ⇒ **A** ⇒ **aAb** ⇒ **a**ε**b** = **ab**.

Už možná vidíme, jaká slova bude **G** generovat. Neterminál **A** totiž obsahuje ve svých pravidlech sám sebe, což nám dovoluje v námi demonstrovaném cyklu přidávat do **w** opakovaně řetězec **aAb**. **G** bude tedy generovat i slova {**aabb**, **aaabbb**, ..., $\mathbf{a^n b^n}$}, neboť můžeme opakovat pravidlo **A** → **aAb** dle libosti.

Tuto gramatiku jsme si ukázali, protože generuje jazyk, který regulárními výrazy ani KA vygenerovat nelze. To jsme si přímo dokázali při demonstraci **Pumping lemmatu** v důkazu sporem.

4.2 Jazyk generovaný BG

Definice

Jazyk **L** generovaný gramatikou **G** značíme **L(G)** a definujeme jako množinu všech slov, která může gramatika nějakou sekvencí pravidel vygenerovat, neboli $\mathbf{L(G)} = \{\mathbf{w} \in \mathbf{T}^* \mid \exists(\mathbf{S} \Rightarrow ... \Rightarrow \mathbf{w})\}$.

Dvě gramatiky $\mathbf{G_1}$ a $\mathbf{G_2}$ jsou ekvivalentní, pokud $\mathbf{L(G_1)} = \mathbf{L(G_2)}$.

Jazyky generované BG nazýváme bezkontextové jazyky.

Příklady

Jaký jazyk generuje gramatika $S \to aS \mid Sb$?

Předpokládáme tedy gramatiku $G = \{\{S\}, \{a, b\}, S, \{S \to aS \mid Sb\}\}$. Chceme popsat generovaný jazyk $L = L(G)$. Můžeme vypozorovat několik poznatků:

1. G nemůže generovat prázdná slova.
2. G nemá žádná pravidla obsahující **pouze** terminální symboly $\to$ slova generovaná G jsou nutně nekonečná.
3. G může generovat slova bez znaku a a také slova bez znaku b tím, že vždy zvolí pravidlo s druhým znakem.

L bude tedy nekonečný jazyk slov tvaru $a^m b^n \mid m, n \geq 0$, která budou nekonečně dlouhá. Zároveň platí, že pokud $m = 0 \Rightarrow n \neq m$ a $n = 0 \Rightarrow m \neq n$.

Generuje gramatika $S \to abSa \mid \varepsilon$ stejný jazyk jako $S \to aSa \mid bS \mid \varepsilon$?

Máme tedy:

$G_1 = \{\{S\}, \{a, b, \varepsilon\}, S, \{S \to abSa \mid \varepsilon\}\}$
$G_2 = \{\{S\}, \{a, b, \varepsilon\}, S, \{S \to aSa \mid bS \mid \varepsilon\}\}$

Chtěli bychom zjistit, zda $L(G_1) \stackrel{?}{=} L(G_2)$. Zkusme tedy opět vypozorovat vlastnosti gramatik:

1. Obě gramatiky mohou generovat prázdná slova.
2. G_1 generuje slova ve tvaru $(ab)^n a^n$.
3. G_2 také umí generovat $(ab)^n a^n$, ale dokáže i slova jiných tvarů, třeba b^n, a^{n+1}, $a^n b^m a^n$, ...

Tyto poznatky už stačí k odpovědi. G_2 určitě generuje slova, která jazyk $L(G_1)$ neobsahuje, například už aa. Už jen jeden protipříklad nám stačí k odpovědi, že $L(G_1) \neq L(G_2)$. Můžeme si uvědomit, že $L(G_1) \subseteq L(G_2)$.

4.3 BG bez ε-pravidel

Definice

Jako ε − **pravidlo** v dané **BG** definujeme pravidlo tvaru $A \to \varepsilon$, kde $A \in N$.

BG nazveme **bez** ε − **pravidel** $\iff$ žádná taková pravidla neobsahuje, **NEBO** obsahuje jediné ε − **pravidlo** s počátečním neterminálem na levé straně $S \to \varepsilon$ a S pak nikde samo nevystupuje na pravé straně nějakého pravidla ($A \to S$).

Platí, že libovolnou **BG** s ε − **pravidly** lze převést na ekvivalentní **BG** bez ε − **pravidel**.

Odstranění ε − pravidel

Obecný postup pro odstranění ε − **pravidel** je následující:

1. Nalezneme množinu $\mathbf{N}_\varepsilon$, která obsahuje všechny neterminály, z nichž bychom se přímo či nepřímo mohli dopracovat k ε (neboli všechny neterminály s pravidly tvaru $\mathbf{A} \Rightarrow ... \Rightarrow \varepsilon$).
2. Přepíšeme pravidla, nyní bez ε − **pravidel**.
3. Upravíme ostatní pravidla, která byla odstraněním ε − **pravidel** ovlivněna tak, abychom zachovali ekvivalenci.

   Pokud počáteční neterminál obsahoval ε − **pravidlo**, Vytvoříme nový počáteční neterminál **S'**, který bude mít pouze dvě pravidla $\mathbf{S'} \rightarrow \mathbf{S} \mid \varepsilon$.

Přesný popis toho, jak pravidla upravíme, se nejlépe ukazuje na příkladě. Na něm si odstranění ukážeme.

Příklad

Mějme **G** definovanou $\mathbf{G} = \{\{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}, \{\mathbf{0}, \mathbf{1}\}, \mathbf{S}\}$ s následujícími pravidly:

$\mathbf{S} \rightarrow \mathbf{BSB} \mid \mathbf{DC} \mid \mathbf{B} \mid \varepsilon$

$\mathbf{A} \rightarrow \mathbf{11} \mid \mathbf{00}$

$\mathbf{B} \rightarrow \mathbf{00} \mid \varepsilon$

$\mathbf{C} \rightarrow \mathbf{1} \mid \mathbf{S}$

$\mathbf{D} \rightarrow \mathbf{CB} \mid \mathbf{S0C1} \mid \mathbf{A}$

Vidíme, že **G** obsahuje několik přímých i nepřímých ε − **pravidel**. Jsou to:

$\mathbf{S} \rightarrow \varepsilon$

$\mathbf{B} \rightarrow \varepsilon$

$\mathbf{C} \rightarrow \mathbf{S} \rightarrow \varepsilon$

$\mathbf{D} \rightarrow \mathbf{CB} \rightarrow \mathbf{S}\varepsilon \rightarrow \varepsilon\varepsilon = \varepsilon$

$\mathbf{S} \rightarrow \mathbf{B} \rightarrow \varepsilon$

$\mathbf{S} \rightarrow \mathbf{BSB} \rightarrow \varepsilon\mathbf{SB} \rightarrow \varepsilon\mathbf{S}\varepsilon \rightarrow \varepsilon\varepsilon\varepsilon = \varepsilon$

$\mathbf{S} \rightarrow \mathbf{DC} \rightarrow \mathbf{CBD} \rightarrow \mathbf{CBCB} \rightarrow ... \rightarrow \mathbf{SBSB} \rightarrow ... \rightarrow \varepsilon\varepsilon\varepsilon\varepsilon = \varepsilon$

Pro odstranění musíme nalézt prvky množiny $\mathbf{N}_\varepsilon$. Budou to neterminály, ze kterých se dá dostat k prázdnému slovu ε. Z našeho výpisu pravidel vidíme, že to budou neterminály na levé straně, tedy $\mathbf{N}_\varepsilon = \{\mathbf{S}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$.

Nyní navrhněme novou gramatiku **G′ bez** ε **− pravidel**. Budeme muset přidat pravidlo pro každý dřívější možný výskyt ε, abychom zachovali ekvivalenci s **G**. V praxi to znamená, že pokud nějaké pravidlo obsahuje neterminály z $\mathbf{N}_\varepsilon$, pak musíme přidat pravidlo pro každé možné nahrazení těchto neterminálů terminálem ε. Ukažme si postup přímo na pravidlech pro neterminál **S**:

$\boxed{\mathbf{S} \to \mathbf{BSB}}$ Jelikož $\mathbf{B}, \mathbf{S} \in \mathbf{N}_\varepsilon$, musíme přidat pravidla pro všechny různé kombinace nahrazení neterminálu prázdným slovem. Mohli jsme například nahradit $\mathbf{BSB} \to \varepsilon\mathbf{SB} = \mathbf{SB}$, nebo $\mathbf{BSB} \to \mathbf{B}\varepsilon\mathbf{B} = \mathbf{BB}$. Jelikož **G′** nebude mít ε **− pravidla**, musíme všechny tyto situace vystihnout novými pravidly, abychom zachovali ekvivalenci s **G**. Přidáme tedy pravidla:
S → **BS** | **BB** | **SB** | **S** | **B**.
Situaci, kdy se všechny neterminály stanou ε samozřejmě nepíšeme, neboť bychom tak přidali ε **− pravidlo**.

$\boxed{\mathbf{S} \to \mathbf{DC}}$ Řešíme analogicky k předchozímu pravidlu. Přidáme **S** → **D** | **C**.

$\boxed{\mathbf{S} \to \mathbf{B}}$ Není co přidat, protože bychom přidali ε **− pravidlo**.

$\boxed{\mathbf{S} \to \varepsilon}$ Toto pravidlo odstraníme, protože se jedná o ε **− pravidlo**.

Celkem tedy pro **S** máme pravidla: **S** → **BSB** | **BS** | **BB** | **SB** | **S** | **B** | **DC** | **D** | **C**. Úpravy pro ostatní pravidla provedeme analogicky. U neterminálů **A** a **C** nedojde k žádné úpravě. U **B** pouze odstraníme ε **− pravidlo**. U **D** pak přidáme několik pravidel: **D** → **CB** | **C** | **B** | **S0C1** | **0C1** | **10** | **S01** | **A**.

Jelikož původní **G** měla počáteční neterminál s ε **− pravidlem**, musíme ještě dodefinovat nový počáteční neterminál **S′** → **S** | ε. Tak zaručíme akceptování prázdného slova u **G′** a úplnou ekvivalenci s **G**.

Tím jsme skončili a máme $\mathbf{G}' = \{\{\mathbf{S}', \mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}, \{\mathbf{0}, \mathbf{1}\}, \mathbf{S}'\}$ s pravidly:

S′ → **S** | ε

S → **BSB** | **BS** | **BB** | **SB** | **S** | **B** | **DC** | **D** | **C**

A → **11** | **00**

B → **00**

C → **1** | **S**

D → **CB** | **C** | **B** | **S0C1** | **0C1** | **10** | **S01** | **A**

4.4 BG bez jednotkových pravidel

Definice

Jako **jednotkové pravidlo** (někdy označované jako jednoduché pravidlo) definujeme pravidlo tvaru $\mathbf{A} \rightarrow \mathbf{B}$, kde $\mathbf{A}, \mathbf{B} \in \mathbf{N}$.

BG nazveme **bez jednotkových pravidel** $\iff$ žádná taková pravidla neobsahuje.

Platí, že libovolnou **BG** (ideálně bez ε − **pravidel**) lze převést na ekvivalentní **BG** bez **jednotkových pravidel**.

Odstranění jednotkových pravidel

Postup pro odstranění jednotkových pravidel z nějaké gramatiky **G** je následující:

1. Pro každý neterminál **A** vytvoříme množinu $\mathbf{N_A}$, která bude obsahovat neterminály dosažitelné z **A** přes jednotková pravidla. $\mathbf{N_A}$ bude vždy obsahovat alespoň **A**.
2. Z pravidel neterminálu **A** odstraním jednoduchá pravidla a přidám k nim pravidla (opět ne jednoduchá) všech neterminálů z $\mathbf{N_A}$.

Příklad

Máme gramatiku $\mathbf{G'} = \{\{\mathbf{S'}, \mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}, \{\mathbf{0}, \mathbf{1}\}, \mathbf{S'}\}$ s pravidly:

$\mathbf{S'} \rightarrow \mathbf{S} \mid \varepsilon$

$\mathbf{S} \rightarrow \mathbf{BSB} \mid \mathbf{BS} \mid \mathbf{BB} \mid \mathbf{SB} \mid \mathbf{S} \mid \mathbf{B} \mid \mathbf{DC} \mid \mathbf{D} \mid \mathbf{C}$

$\mathbf{A} \rightarrow \mathbf{11} \mid \mathbf{00}$

$\mathbf{B} \rightarrow \mathbf{00}$

$\mathbf{C} \rightarrow \mathbf{1} \mid \mathbf{S}$

$\mathbf{D} \rightarrow \mathbf{CB} \mid \mathbf{C} \mid \mathbf{B} \mid \mathbf{S0C1} \mid \mathbf{0C1} \mid \mathbf{10} \mid \mathbf{S01} \mid \mathbf{A}$

(Výsledná gramatika z minulého příkladu)

Nyní si pro každý neterminál vytvoříme množinu tak, jak jsme definovali v postupu:

S' Kromě **S'** samotného je zde jednotkové pravidlo $\mathbf{S'} \rightarrow \mathbf{S}$. Z **S** ale vedou další nová jednotková pravidla, která je třeba zahrnout. Jsou to $\mathbf{S} \rightarrow \mathbf{B} \mid \mathbf{D} \mid \mathbf{C} \mid \mathbf{S}$. Z **B** a **C** nevedou žádná nová jednotková pravidla, ale z **D** vede $\mathbf{D} \rightarrow \mathbf{A}$. Celkem máme tedy $\mathbf{N_{S'}} = \{\mathbf{S'}, \mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$.

S Řešíme analogicky k **S'**. Získáme $\mathbf{N_S} = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$.

A $\mathbf{N_A} = \{\mathbf{A}\}$

B $\mathbf{N_B} = \{\mathbf{B}\}$

C $\mathbf{N_C} = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\} = \mathbf{N_S}$

D $\mathbf{N_D} = \{\mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\} = \mathbf{N_S} = \mathbf{N_C}$

Nyní odstraníme jednotková pravidla a přidáme ke každému neterminálu pravidla neterminálů z jeho množiny. Získáme tak:

$\mathbf{S'} \rightarrow \varepsilon \mid \mathbf{BSB} \mid \mathbf{BS} \mid \mathbf{BB} \mid \mathbf{SB} \mid \mathbf{DC} \mid \mathbf{11} \mid \mathbf{00} \mid \mathbf{1} \mid \mathbf{CB} \mid \mathbf{S0C1} \mid \mathbf{0C1} \mid \mathbf{10} \mid \mathbf{S01}$

$\mathbf{S} \rightarrow \mathbf{BSB} \mid \mathbf{BS} \mid \mathbf{BB} \mid \mathbf{SB} \mid \mathbf{DC} \mid \mathbf{11} \mid \mathbf{00} \mid \mathbf{1} \mid \mathbf{CB} \mid \mathbf{S0C1} \mid \mathbf{0C1} \mid \mathbf{10} \mid \mathbf{S01}$

$\mathbf{A} \rightarrow \mathbf{11} \mid \mathbf{00}$

$\mathbf{B} \rightarrow \mathbf{00}$

$\mathbf{C} \rightarrow \mathbf{BSB} \mid \mathbf{BS} \mid \mathbf{BB} \mid \mathbf{SB} \mid \mathbf{DC} \mid \mathbf{11} \mid \mathbf{00} \mid \mathbf{1} \mid \mathbf{CB} \mid \mathbf{S0C1} \mid \mathbf{0C1} \mid \mathbf{10} \mid \mathbf{S01}$

$\mathbf{D} \rightarrow \mathbf{BSB} \mid \mathbf{BS} \mid \mathbf{BB} \mid \mathbf{SB} \mid \mathbf{DC} \mid \mathbf{11} \mid \mathbf{00} \mid \mathbf{1} \mid \mathbf{CB} \mid \mathbf{S0C1} \mid \mathbf{0C1} \mid \mathbf{10} \mid \mathbf{S01}$

Takto bychom měli mít BG ekvivalentní s **G'** a bez jednotkových pravidel.

4.5 BG v Chomského normálním tvaru

Definice

BG v Chomského normálním tvaru definujeme jako gramatiku obsahující pouze pravidla následujících tvarů:

$\boxed{\mathbf{A} \to \mathbf{a}}$; $\mathbf{A} \in \mathbf{N}, \mathbf{a} \in \mathbf{T}$

$\boxed{\mathbf{A} \to \mathbf{BC}}$; $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbf{N}$

$\boxed{\mathbf{S} \to \varepsilon}$; **S** je počáteční neterminál, nikdy není na pravé straně pravidla

Při pohledu na povolená pravidla je jasné, že **BG v Chomského normálním tvaru** je zároveň i **BG bez** ε **− pravidel** a **jednotkových pravidel**. Jejich odstranění je součástí převodu na Chomského tvar.

Převod na Chomského normální tvar

Postup pro převod nějaké **BG** do Chomského tvaru je následovný:

1. Odstraníme ε **− pravidla**.
2. Odstraníme **jednotková pravidla**.
3. Odstraníme **zbytečné neterminály** (nedosažitelné žádnou sekvencí pravidel z **S**).
4. Pravidla nesplňující Chomského normálního tvar upravíme tak, aby mu vyhovovali. Při úpravách běžně vznikne potřeba dodefinování nových neterminálů a pravidel.

Dodefinování pravidel je těžké jednoduše obecně popsat, ale je vcelku intuitivní a přímočaré, pokud si ho ukážeme na příkladě.

Příklad

Vezměme si výsledek z předchozího příkladu, protože se už jedná o **BG bez jednotkových** a ε − **pravidel**.

Máme tedy $\mathbf{G} = \{\{\mathbf{S'}, \mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}, \{\mathbf{1}, \mathbf{0}\}, \mathbf{S'}\}$ s pravidly:

S′ → ε | BSB | BS | BB | SB | DC | 11 | 00 | 1 | CB | S0C1 | 0C1 | 10 | S01

S → BSB | BS | BB | SB | DC | 11 | 00 | 1 | CB | S0C1 | 0C1 | 10 | S01

A → 11 | 00

B → 00

C → BSB | BS | BB | SB | DC | 11 | 00 | 1 | CB | S0C1 | 0C1 | 10 | S01

D → BSB | BS | BB | SB | DC | 11 | 00 | 1 | CB | S0C1 | 0C1 | 10 | S01

Gramatika obsahuje pravidla, která je třeba upravit. Ukažme si na nich, jaké obecně úpravy a dodefinováni při převodu na Chomského tvar používáme:

S′ → BSB Pravidlo obsahuje jen neterminály, chtěli bychom ho tedy dostat do tvaru **A → BC**. Běžný způsob, jak řešit úpravu pravidel, která mají jen neterminály a jsou delší než dva je rozdělením na dvojice neterminálů a dodefinováním tzv. binárních neterminálů.

Uvažujme pravidlo **A → BCDEFG**. Abychom redukovali délku pravé strany pravidla, zadefinujeme si nový neterminál s pravidlem pro každou dvojici následovně:

(**BC**) → **BC**

(**DE**) → **DE**

(**FG**) → **FG**

Získáváme **A** → (**BC**)(**DE**)(**FG**). Pravá strana má pořád víc než dva neterminály, opakujeme tedy předchozí krok. Protože má pravá strana lichý počet neterminálů, (**BC**) necháme a krok provedeme pro zbývající (teď nutně sudý počet) neterminály:

(**DEFG**) → (**DE**)(**FG**)

Analogicky vyřešíme zadané pravidlo a získáme nový neterminál (**BS**). Celkem tedy upravíme pravidlo na **S′** → (**BS**)**B**.

S′ → 11 Toto pravidlo budeme muset opět dostat do tvaru **A → BC**.

Opět si pomůžeme dodefinováním nového neterminálu (**1**) s pravidlem (**1**) → **1**, které splňuje tvar **A → a**. Získáváme tak **S′** → (**1**)(**1**), což splňuje naše podmínky.

$\boxed{\mathbf{S'} \to \mathbf{00}}$ Řešíme analogicky k předchozímu pravidlu. Dodefinujeme $(\mathbf{0}) \to \mathbf{0}$.

$\boxed{\mathbf{S'} \to \mathbf{S0C1}}$ Pravidlo obsahuje terminály i neterminály, což nechceme, neboť žádné pravidlo v Chomského tvaru neobsahuje terminály i neterminály.

Potřebovali bychom na pravé straně pouze neterminály. Tento problém bychom řešili dodefinováním nových neterminálů tak, aby pravá strana obsahovala jenom je. Naštěstí pro nás už jsme při řešení jiných pravidel definovali neterminály $(\mathbf{0})$ a $(\mathbf{1})$. Máme tedy $\mathbf{S'} \to \mathbf{S}(\mathbf{0})\mathbf{C}(\mathbf{1})$. Dále řešíme analogicky k prvnímu řešenému pravidlu:

$(\mathbf{S}(\mathbf{0})) \to \mathbf{S}(\mathbf{0})$

$(\mathbf{C}(\mathbf{1})) \to \mathbf{C}(\mathbf{1})$

Získáváme $\mathbf{S'} \to (\mathbf{S}(\mathbf{0}))(\mathbf{C}(\mathbf{1}))$.

$\boxed{\mathbf{S'} \to \mathbf{0C1}}$ Vyřešíme již zadefinovanými neterminály. Získáme $\mathbf{S'} \to (\mathbf{0})(\mathbf{C}(\mathbf{1}))$.

$\boxed{\mathbf{S'} \to \mathbf{10}}$ Převedeme na $\mathbf{S'} \to (\mathbf{1})(\mathbf{0})$.

$\boxed{\mathbf{S'} \to \mathbf{S01}}$ Převedeme na $\mathbf{S'} \to (\mathbf{S}(\mathbf{0}))(\mathbf{1})$.

Tím jsme vyřešili nejen pravidla počátečního neterminálu, ale i všech ostatních. Nikde jinde se pravidlo, které bychom nevyřešili, nevyskytuje. Po dodefinování všech nových neterminálů a pravidel máme tedy výslednou gramatiku $\mathbf{G} = \{\{\mathbf{S'}, \mathbf{S}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, (\mathbf{BS}), (\mathbf{1}), (\mathbf{0}), (\mathbf{S}(\mathbf{0})), (\mathbf{C}(\mathbf{1})), \{\mathbf{0}, \mathbf{1}\}, \mathbf{S'}\}\}$ s pravidly:

$\mathbf{S'} \rightarrow \varepsilon \mid (\mathbf{BS})\mathbf{B} \mid \mathbf{BS} \mid \mathbf{BB} \mid \mathbf{SB} \mid \mathbf{DC} \mid (\mathbf{1})(\mathbf{1}) \mid (\mathbf{0})(\mathbf{0}) \mid \mathbf{1} \mid \mathbf{CB} \mid (\mathbf{S}(\mathbf{0}))(\mathbf{C}(\mathbf{1})) \mid (\mathbf{0})(\mathbf{C1})$
$(\mathbf{1})(\mathbf{0}) \mid (\mathbf{S}(\mathbf{0}))(\mathbf{1})$

$\mathbf{S} \rightarrow (\mathbf{BS})\mathbf{B} \mid \mathbf{BS} \mid \mathbf{BB} \mid \mathbf{SB} \mid \mathbf{DC} \mid (\mathbf{1})(\mathbf{1}) \mid (\mathbf{0})(\mathbf{0}) \mid \mathbf{1} \mid \mathbf{CB} \mid (\mathbf{S}(\mathbf{0}))(\mathbf{C}(\mathbf{1})) \mid (\mathbf{0})(\mathbf{C1})$
$(\mathbf{1})(\mathbf{0}) \mid (\mathbf{S}(\mathbf{0}))(\mathbf{1})$

$\mathbf{A} \rightarrow (\mathbf{1})(\mathbf{1}) \mid (\mathbf{0})(\mathbf{0})$

$\mathbf{B} \rightarrow (\mathbf{0})(\mathbf{0})$

$\mathbf{C} \rightarrow (\mathbf{BS})\mathbf{B} \mid \mathbf{BS} \mid \mathbf{BB} \mid \mathbf{SB} \mid \mathbf{DC} \mid (\mathbf{1})(\mathbf{1}) \mid (\mathbf{0})(\mathbf{0}) \mid \mathbf{1} \mid \mathbf{CB} \mid (\mathbf{S}(\mathbf{0}))(\mathbf{C}(\mathbf{1})) \mid (\mathbf{0})(\mathbf{C1})$
$(\mathbf{1})(\mathbf{0}) \mid (\mathbf{S}(\mathbf{0}))(\mathbf{1})$

$\mathbf{D} \rightarrow (\mathbf{BS})\mathbf{B} \mid \mathbf{BS} \mid \mathbf{BB} \mid \mathbf{SB} \mid \mathbf{DC} \mid (\mathbf{1})(\mathbf{1}) \mid (\mathbf{0})(\mathbf{0}) \mid \mathbf{1} \mid \mathbf{CB} \mid (\mathbf{S}(\mathbf{0}))(\mathbf{C}(\mathbf{1})) \mid (\mathbf{0})(\mathbf{C1})$
$(\mathbf{1})(\mathbf{0}) \mid (\mathbf{S}(\mathbf{0}))(\mathbf{1})$

$(\mathbf{0}) \rightarrow \mathbf{0}$

$(\mathbf{1}) \rightarrow \mathbf{1}$

$(\mathbf{BS}) \rightarrow \mathbf{BS}$

$(\mathbf{S}(\mathbf{0})) \rightarrow \mathbf{S}(\mathbf{0})$

$(\mathbf{C}(\mathbf{1})) \rightarrow \mathbf{C}(\mathbf{1})$

Tato gramatika splňuje podmínky Chomského normálního tvaru.

Zásobníkové automaty

5.1 Definice

Zásobníkový automat (ZA) je struktura podobná konečnému automatu, který jsme již popisovali. Oproti **KA** má ale ještě takzvaný **zásobník** (stack), který dovoluje generovat širší množinu jazyků než **KA**. Zásobník je forma paměti, která dovoluje **ZA** se při zpracování znaku slova rozhodovat nejenom na základě právě přečteného znaku, ale i na základě dříve přečtených znaků. Zásobník si detailněji popíšeme při jeho definici.

ZA definujeme jako šestici množin $(\mathbf{Q}, \mathbf{\Sigma}, \mathbf{\Gamma}, \delta, \mathbf{q_0}, \mathbf{Z_0})$, pro které platí:

$\boxed{\mathbf{Q}}$ Konečná neprázdná množina stavů automatu

$\boxed{\mathbf{\Sigma}}$ Konečná neprázdná množina znaků automatu (vstupní abeceda)

$\boxed{\mathbf{\Gamma}}$ konečná neprázdná množina zásobníkových znaků (zásobníková abeceda)

$\boxed{\delta}$ Přechodová funkce $\mathbf{Q} \times (\mathbf{\Sigma} \cup \{\varepsilon\}) \times \mathbf{\Gamma} \rightarrow \mathbf{Q} \times \mathbf{\Gamma}^*$. Značíme $(\mathbf{q}, \mathbf{a}, \mathbf{z}) \rightarrow (\mathbf{q'}, \mathbf{w})$ (pokud ve stavu $\mathbf{q}$ čtu znak $\mathbf{a} \in \mathbf{\Sigma}$ a na vrcholu zásobníku je znak $\mathbf{z} \in \mathbf{\Gamma} \rightarrow$ přecházím do stavu $\mathbf{q'}$, ze zásobníku odeberu $\mathbf{z}$ a přidám na něj $\mathbf{w} \in \mathbf{\Gamma}^*$)

Pokud $\mathbf{z} = \mathbf{Z_0} \rightarrow$ zásobník je prázdný.

$\boxed{\mathbf{q_0}}$ Počáteční stav

$\boxed{\mathbf{Z_0}}$ Počáteční zásobníkový symbol (nejčastěji ε)

Zásobník si tedy můžeme představit velmi podobně jako stejnojmennou datovou strukturu. Můžeme na něj psát, či z něj odebírat znaky, přičemž vždy vidíme jeho poslední přidaný znak (analogie s funkcí top()). Přečtením znaku ze zásobníku ho odstraníme.Tato forma paměti nám dovoluje generovat složitější jazyky, než bylo možné s **KA**.

Aby **ZA** akceptoval nějaké slovo $\mathbf{w} \in \mathbf{\Sigma}^*$, musí platit:

---

1. Zpracování **w** začíná v počátečním stavu $(\mathbf{q_0}, \mathbf{w}, \mathbf{Z_0})$
2. Zpracuje se celé slovo
3. Zásobník je po zpracování slova prázdný (obsahuje pouze $\mathbf{Z_0}$)

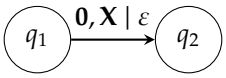
Tedy na konci musíme skončit v libovolném stavu $(\mathbf{q}, \varepsilon, \mathbf{Z_0})$

Jeden z hlavních rozdílů mezi **KA** a **ZA** je **absence koncových stavů v ZA**. O akceptování slova rozhoduje jeho úspěšné zpracování a stav zásobníku.

5.2 Stavový diagram ZA

Stejně jako u **KA**, i zásobníkové automaty lze graficky vyjádřit. Hlavní rozdíl je, že u hran píšeme kromě znaku čteného ze slova také **znak čtený ze zásobníku** a **znak vepsaný do zásobníku při přechodu**.

Příklad přechodu mezi stavy $\mathbf{q_1} \rightarrow \mathbf{q_2}$, který přečte znak **0** ze vstupního slova, pokud je **X** na vrcholu zásobníku a nic do něj nepřidá (tedy přidá ε), je následující:



Tímto způsobem vyjadřujeme **ZA** velmi podobným způsobem jako **KA** stavovým diagramem.

5.3 Příklady

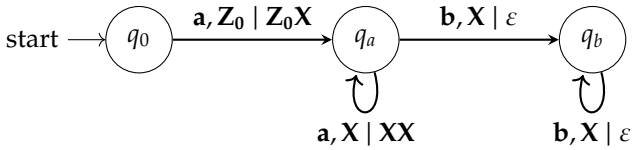
Příklad 1

Ukažme si návrh a diagram zásobníkového automatu, který akceptuje jazyk $\mathbf{L} = \{\mathbf{a^n b^n} \mid \mathbf{n} \geq \mathbf{0}\}$. V důkazu **pumping lemmatem** jsme dokázali, že **L** není regulární jazyk → nelze ho tedy vyjádřit s pomocí **KA**. Navrhněme tedy zásobníkový automat, který bude **L** generovat.

Potřebovali bychom automat akceptující slova se stejným počtem znaků **a** a **b**. Logicky bychom tedy potřebovali nějak mít přehled o tom, jestli zrovna máme stejný počet těchto znaků, nebo ne. K tomu použijeme zásobník.

Řekněme, že za každý zpracovaný znak **a** přidáme na zásobník **X**. Naopak za každý znak **b** na zásobník nepřidáme nic, ale přečteme z něho znak **X**. Takto zajistíme, že zpracované slovo **w** skončí s prázdným zásobníkem právě pokud $\mid \mathbf{w} \mid_{\mathbf{a}} = \mid \mathbf{w} \mid_{\mathbf{b}}$.

Pokud se při návrhu přechodů v našem automatu budeme řídit těmito pravidly, měli bychom vytvořit automat, který pro každé slovo tvaru $\mathbf{a^n b^n} \mid \mathbf{n} \geq \mathbf{0}$ skončí průchod akceptováním. Ke generování jazyka **L** by nám měli stačit tři stavy.



(pozn. při zapisování na zásobník píšeme dva znaky, protože přečtením daný znak odstraňujeme)

Tento automat by měl generovat jazyk **L**. Zároveň jsme ho navrhli tak, že akceptuje pouze slova, kde první je sekvence znaků **a**, pak **b**. Jakmile v libovolném stavu nastane situace, že slovo je zpracované a zásobník prázdný → slovo je akceptované.

Popišme si průběh zpracování několika slov:

$\boxed{\mathbf{w = aaabbb}}$ Zpracování slova začíná ve stavu $\mathbf{q_0}$ a s prázdným zásobníkem. Máme tedy $(\mathbf{q_0}, \mathbf{aaabbb}, \mathbf{Z_0}) \rightarrow (\mathbf{q_a}, \mathbf{aabbb}, \mathbf{Z_0X})$. Nyní jsme ve stavu $\mathbf{q_a}$, na zásobníku je $\mathbf{X}$ a na vstupu $\mathbf{a}$. Tudíž budeme teď procházet smyčkou ve stavu $\mathbf{q_a}$, dokud nenarazíme na znak $\mathbf{b}$.

Tedy $(\mathbf{q_a}, \mathbf{aabbb}, \mathbf{Z_0X}) \rightarrow (\mathbf{q_a}, \mathbf{abbb}, \mathbf{Z_0XX}) \rightarrow (\mathbf{q_a}, \mathbf{bbb}, \mathbf{Z_0XXX})$.
Nyní máme na vstupu znak $\mathbf{b}$ a na zásobníku $\mathbf{X}$, tedy přecházíme do dalšího stavu $(\mathbf{q_a}, \mathbf{bbb}, \mathbf{Z_0XXX}) \rightarrow (\mathbf{q_b}, \mathbf{bb}, \mathbf{Z_0XX}) \rightarrow (\mathbf{q_b}, \mathbf{b}, \mathbf{Z_0X}) \rightarrow (\mathbf{q_b}, \varepsilon, \mathbf{Z_0})$. Tím jsme se dostali do akceptujícího stavu, tudíž slovo je **akceptované**.

$\boxed{\mathbf{w} = \varepsilon}$ Tento příklad je jednoduchý, neboť již na začátku jsme v akceptujícím stavu $(\mathbf{q_0}, \varepsilon, \mathbf{Z_0})$.

$\boxed{\mathbf{w = abb}}$ Začínáme $(\mathbf{q_0}, \mathbf{abb}, \mathbf{Z_0}) \rightarrow (\mathbf{q_a}, \mathbf{bb}, \mathbf{Z_0X}) \rightarrow (\mathbf{q_b}, \mathbf{b}, \mathbf{Z_0})$. Zde jsme ale skončili, protože nemáme definovaný přechod ze stavu $\mathbf{q_b}$, pokud čteme znak $\mathbf{b}$ a zásobník **není** prázdný. To znamená, že toto slovo nemá stejný počet znaků $\mathbf{a}$ a $\mathbf{b}$ → automat ho **neakceptuje**.

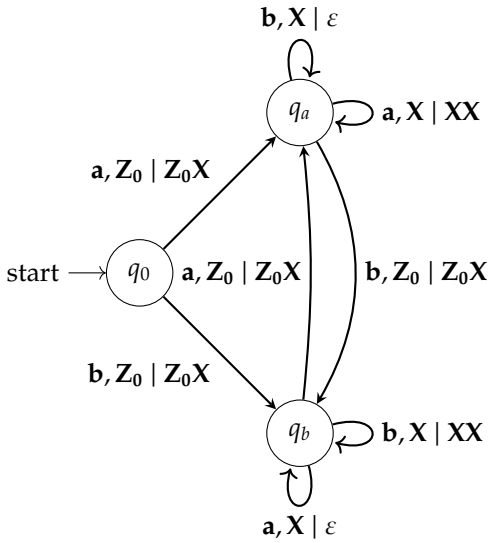
Vidíme tedy, že automat pracuje tak, jak bychom chtěli.

Příklad 2

Ukažme si ještě jeden automat podobný tomu předchozímu. Navrhněme **ZA** generující jazyk $\mathbf{J} = \{\mathbf{w} \in \{\mathbf{a}, \mathbf{b}\}^*; |\ \mathbf{w}\ |_{\mathbf{a}} = |\ \mathbf{w}\ |_{\mathbf{b}}\}$, neboli jazyk slov se stejným počtem znaků **a** jako **b**. Rozdíl od předešlého příkladu je ten, že nám nezáleží na pořadí znaků ani na tom, jak se budou střídat. Zajímá nás čistě jen jejich počet.

Zásobník bude v tomto automatu fungovat podobně jako v minulém příkladu. Je ale třeba si uvědomit, že musíme počítat výskyt zrovna toho znaku, kterého je v daný moment víc. Například řetězec **aaabbbba** patří do **J**, ale pokud by zásobník fungoval stejně jako v minulém příkladě, automat by slovo neakceptoval. Budeme tedy muset přechody navrhnout tak, abychom přidávali na zásobník **X** za zrovna častější znak a odebírali za druhý.

V praxi může automat generující **J** navrhnout třeba takto:



Tento automat v závislosti na prvním znaku slova přejde do $\mathbf{q_a}$ nebo $\mathbf{q_b}$ → za tento znak začne přidávat na zásobník **X** a za druhý znak odebírat **X**. Pokud se znaky vyrovnají (zásobník se vyprázdní) a začne převládat druhý znak, automat přejde do jeho stavu a začne přidávat **X** za něj.

Podívejme se na již zmíněné slovo **aaabbbba**. **ZA** ho zpracuje následovně:

$(\mathbf{q_0}, \mathbf{aaabbbba}, \mathbf{Z_0}) \rightarrow (\mathbf{q_a}, \mathbf{aabbbba}, \mathbf{Z_0X}) \rightarrow (\mathbf{q_a}, \mathbf{abbbba}, \mathbf{Z_0XX}) \rightarrow$
$(\mathbf{q_a}, \mathbf{bbbba}, \mathbf{Z_0XXX}) \rightarrow (\mathbf{q_a}, \mathbf{bbba}, \mathbf{Z_0XX}) \rightarrow ... \rightarrow (\mathbf{q_a}, \mathbf{ba}, \mathbf{Z_0}) \rightarrow$
$(\mathbf{q_b}, \mathbf{a}, \mathbf{Z_0X}) \rightarrow (\mathbf{q_b}, \varepsilon, \mathbf{Z_0}) \rightarrow$ **akceptuje**

Turingův stroj

Turingův stroj je zásadním nástrojem v oboru informatiky a matematiky. Jedná se o stroj nacházející se na vrcholu hierarchie abstraktních výpočetních modelů, což ho staví nad již probrané **konečné a zásobníkové automaty**. Turingův stroj je totiž model schopný simulovat jakýkoliv jiný algoritmický proces. Jinými slovy je schopen se chovat jako **KA**, **ZA**, nebo jako jakýkoliv jiný model řešící nějaký problém. Automaty definované v předchozích kapitolách jsou pouze zjednodušenými specifickými verzemi **Turingova stroje**.

6.1 Turingův stroj

Definice

Před formální definicí si **Turingův stroj** definujeme trochu abstraktně, čímž lépe pochopíme jeho jednotlivé složky.

Představme si Turingův stroj jako dva předměty - **Stroj se čtecí hlavou** a **nekonečnou vstupní pásku**. Stroj má několik vnitřních stavů, mezi kterými přechází v závislosti na aktuálním stavu a znacích čtených z pásky. Čtecí hlava stroje může vždy přečíst jen jeden znak z pásky a **může se po ní pohybovat v obou směrech**. Stroj také může znak čtený na pásce přepsat na jiný.

Pokud správně dodefinujeme vnitřní stavy stroje a to, za jakých podmínek bude stroj stavy měnit, jsme schopni simulovat jakýkoliv algoritmický proces. Nyní si **TS** zadefinujeme formálně:

Turingův stroj je definován šesticí $\mathbf{M} = \{\mathbf{Q}, \mathbf{\Gamma}, \mathbf{\Sigma}, \delta, \mathbf{q_0}, \mathbf{F}\}$, kde:

$\boxed{\mathbf{Q}}$ Konečná neprázdná množina stavů

$\boxed{\mathbf{\Gamma}}$ Pásková abeceda

$\boxed{\mathbf{\Sigma}}$ Vstupní abeceda, obecně platí $\mathbf{\Sigma} = \mathbf{\Gamma}\backslash\{\mathbf{b}\}$, kde **b** je prázdný (**blank**) znak

$\boxed{\delta}$ Přechodová funkce $(\mathbf{Q}\backslash\mathbf{F}) \times \mathbf{\Gamma} \rightarrow \mathbf{Q} \times \mathbf{\Gamma} \times \{-, \mathbf{0}, +\}$, neboli funkce, která v závislosti na přečteném nekoncovém stavu q_a a znaku x na pásce přejde do q_b, případně změní x na pásce a/nebo se posune.

(pozn: **-** doleva, **0** nic, **+** doprava)

$\boxed{\mathbf{q_0}}$ Počáteční stav, platí $\mathbf{q_0} \in \mathbf{Q}$

$\boxed{\mathbf{F}}$ množina koncových stavů, platí $\mathbf{F} \subseteq \mathbf{Q}$. Pokud se **TS** dostane zpracováním slova **w** na vstupní pásce do stavu $q \in \mathbf{F}$, pak **TS** slovo **w** akceptuje.

Nekonečná zpracování pak **TS** neakceptuje. Často také pro **TS** definujeme koncové stavy q_{acc} a q_{rej}, přičemž zpracování končící v q_{acc} akceptuje a ty v q_{rej} ne.

Slova ke zpracování **TS** tedy klademe na nekonečnou pásku, přičemž všechno 'nevyužité' místo na pásce zaplníme prázdným znakem **b**.

Slovo zapsané na pásku a pozice čtecí hlavy **TS** definujeme jako jeho **počáteční konfiguraci**.

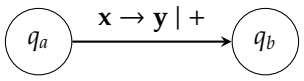
Krok **TS** budeme vnímat asi takto: Pokud **TS** je ve stavu q_a a přečte znak **x** z pásky, přejde do stavu q_b, místo **x** vepíše na pásku **x'** a posune se po pásce o $\mathbf{d} \in \{-, \mathbf{0}, +\}$. Je důležité si uvědomit, že se nemusí posunout, nic přepsat ani přejít do jiného stavu (tedy $q_a = q_b, \mathbf{x} = \mathbf{x}', \mathbf{d} = \mathbf{0}$). Notace jednoho kroku **TS** tedy je $\boxed{\delta(\mathbf{q_a}, \mathbf{x}) \to (\mathbf{q_b}, \mathbf{x}', \mathbf{d})}$.

Také je důležité si uvědomit, že pokud při úspěšném zpracování slova **w** dojde k jeho změně automatem, do generovaného jazyka přidáváme původní slovo **w**.

Ačkoliv takto se může zdát definice **TS** zmatečná, v praxi s ním pracujeme velmi podobně jako s **KA** nebo **ZA**, což si ukážeme v následujících sekcích.

6.2 Graf TS

Stejně jako **KA** nebo **ZA**, i **Turingův stroj** vyjadřujeme nejčastěji v grafické podobě stavového diagramu či grafu. V této podobě jsou pak jednotlivé stavy stroje vyobrazené jako stavy grafu. Hrany mezi stavy budou popisovat pravidla definovaná přechodovou funkcí. Představme si tedy situaci, kdy jsme ve stavu q_a, čteme **x**, máme zapsat **y**, přesunout se po pásce o znak vpřed a přejít do stavu q_b (neboli $\delta(\mathbf{q_a}, \mathbf{x}) \to (\mathbf{q_b}, \mathbf{y}, +)$). Tento krok vyobrazíme následovně:



Tímto způsobem jsme schopni popsat stavy stroje a přechodové funkce, které je spojují.

6.3 Příklad

Zkusme si navrhnout **TS**, který zadané slovo **w** nad abecedou $\{\mathbf{0}, \mathbf{1}\}$ invertuje, neboli převede $\mathbf{0} \to \mathbf{1}, \mathbf{1} \to \mathbf{0}$. Nechť náš **TS** začíná zpracování s čtecí hlavou na prvním znaku zleva.

Jako první je dobré si projít, čeho chceme dosáhnout a jaké situace nás mohou potkat. Jelikož má naše abeceda pouze 2 znaky, které se invertují jeden na druhý, bude to celkem snadné. Existují pouze 3 znaky, které může naše hlava přečíst:

$\boxed{\mathbf{0}}$ Chceme přepsat na **1**

$\boxed{\mathbf{1}}$ Chceme přepsat na **0**

$\boxed{\mathbf{b}}$ Prázdný znak

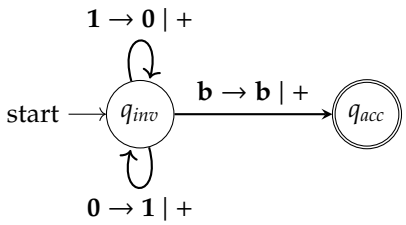
Z tohoto je snad vidět, že musíme definovat aspoň 2 přechodové funkce - jednu pro invertování $\mathbf{0} \to \mathbf{1}$, druhou pro invertování $\mathbf{1} \to \mathbf{0}$. Jinak se ve splnění zadání neposuneme.

Jelikož počáteční konfigurace hlavy je na prvním znaku vlevo, můžeme si uvědomit, že nám bude stačit postupovat pouze vpravo slovem a '*invertovat*' jednotlivé znaky.

Poslední věc, kterou si musíme uvědomit, je konec zpracování slova. Asi je jasné, že všechna slova $\mathbf{w} \in \{\mathbf{0}, \mathbf{1}\}^*$ lze invertovat. Tedy každé zpracování skončí v nějakém akceptujícím koncovém stavu q_{acc}. Logicky bychom se do něj měli dostat jen poté, co jsme invertovali celé slovo **w**. To nastane tehdy, když se čtecí hlava dostane na první prázdný znak **b** vpravo od slova (za předpokladu, že se budeme posouvat pouze doprava, jak jsme si stanovili).

Pokud spojíme tyto naše poznatky dohromady, jsme schopni navrhnout **TS**. Definujme si tedy 2 stavy $\{\mathbf{q_{inv}}, \mathbf{q_{acc}}\}$. Stav $\mathbf{q_{inv}}$ bude náš počáteční a zároveň '*pracující*' stav. Pokud se v něm nacházíme a čteme nějaký znak slova **w**, tak ho invertujeme, posuneme se o jeden znak doprava a zůstáváme v $\mathbf{q_{inv}}$. Jinými slovy definujeme $\delta(\mathbf{q_{inv}}, \mathbf{0}) \to (\mathbf{q_{inv}}, \mathbf{1}, +)$ a $\delta(\mathbf{q_{inv}}, \mathbf{1}) \to (\mathbf{q_{inv}}, \mathbf{0}, +)$. Musíme také definovat konec zpracování slova, neboli přechod do $\mathbf{q_{acc}}$. Ten nastane, pokud jsme zpracovali celé slovo, neboli pokud čteme znak **b**. Tedy definujeme $\delta(\mathbf{q_{inv}}, \mathbf{b}) \to (\mathbf{q_{acc}}, \mathbf{b}, +)$.

Tímto jsme skončili. Pokud si nyní nakreslíme to, co jsme definovali, získáme:



Zkusme si napsat postup pro slovo **w** = **0110**:

1. $\delta(\mathbf{q_{inv}}, \mathbf{0}) \rightarrow (\mathbf{q_{inv}}, \mathbf{1}, +)$ | **0110** → **1110**
2. $\delta(\mathbf{q_{inv}}, \mathbf{1}) \rightarrow (\mathbf{q_{inv}}, \mathbf{0}, +)$ | **1110** → **1010**
3. $\delta(\mathbf{q_{inv}}, \mathbf{1}) \rightarrow (\mathbf{q_{inv}}, \mathbf{0}, +)$ | **1010** → **1000**
4. $\delta(\mathbf{q_{inv}}, \mathbf{0}) \rightarrow (\mathbf{q_{inv}}, \mathbf{1}, +)$ | **1000** → **1001**
5. $\delta(\mathbf{q_{inv}}, \mathbf{b}) \rightarrow (\mathbf{q_{acc}}, \mathbf{b}, +)$ | **1001**

Snad je z tohoto popisu běh **TS** zřejmý.

Zajímavé zdroje a odkazy

Následuje výpis odkazů na různé internetové zdroje, které nám sloužili jako inspirace. V případě nejasností v tomto textu či zvídavosti čtenáře doporučujeme relevantní zdroj navštívit:

---

https://www.cs.vsb.cz/sawa/uti/2021/slides/uti-02-cz.pdf

https://www.cs.vsb.cz/sawa/uti/materialy/uti.pdf

https://www.cs.vsb.cz/sawa/uti/2020/slides/uti-07-cz.pdf

https://www.youtube.com/watch?v=47awv0bU3Hc

https://www.youtube.com/watch?v=C5KerRqrC8c

https://www.youtube.com/watch?v=WpdhNNtYMJY

https://youtu.be/uzFVAmk3uXc?si=_1bOM0sAgmmqIZHv

https://www.youtube.com/watch?v=8Defwmq5X5E

https://youtu.be/uzFVAmk3uXc?si=cwy9RR9Z8F2nBCYU

https://youtu.be/QAacOfpV5GY?si=-hJniGDLQQDMJnGd

https://youtu.be/Y7FZdvqISDU?si=EkUb7p-ZF_zhAdfI

https://youtu.be/kFx0FnPaRF8?si=yzkMGPpEmHik_sNp

https://www.youtube.com/watch?v=6YCrUSMdwbk

https://www.youtube.com/watch?v=YQDfJmKLKsc

https://youtu.be/MMFBhaWeUDY?si=6ajzGvw1JzaqWdZh

https://youtu.be/yCaweOE8VIo?si=Yhp68i8bTmaKoSYc

https://youtu.be/44se2RSyXx0?si=O7K6hSC0zeXmWfYh

https://youtu.be/4uVAwwjQnJk?si=5a4abUFn4HwKo83X
