Úvod do automatů

Martin Mareš

Katedra aplikované matematiky
Matematicko-fyzikální fakulta
Univerzita Karlova, Praha

Toto je stručný úvod do teorie formálních jazyků, automatů a gramatik. Vznikl jako studijní text k předmětu Algoritmy a automaty pro učitele na MFF UK, ale může se hodit i dalším zájemcům o teoretickou informatiku. Text navazuje na Průvodce labyrintem algoritmů.⟨1⟩

Milý čtenáři, buď varován, že se jedná o pracovní verzi, která jistě není dokonalá. Najdeš-li libovolnou chybu (což zahrnuje i těžko srozumitelné pasáže), dej prosím vědět autorovi na adrese *mj@ucw.cz*. Díky!

⟨1⟩ Viz *http://pruvodce.ucw.cz/*.

1 Regulární jazyky

1.1 Definice a značení

- *Abeceda* Σ je nějaká konečná množina, jejím prvkům budeme říkat *znaky* (někdy též *písmena*).
- Σ^* je množina všech *slov* neboli *řetězců* nad abecedou Σ, což jsou konečné posloupnosti znaků ze Σ.
- *Slova* budeme značit malými písmenky řecké abecedy α, β, ...
- *Znaky* abecedy označíme malými písmeny latinky x, y, ... Konkrétní znaky budeme psát `psacím strojem`. Znak budeme používat i ve smyslu jednoznakového řetězce.
- *Délka slova* $|\alpha|$ udává, kolika znaky je slovo tvořeno.
- *Prázdné slovo* značíme ε, je to jediné slovo délky 0.
- *Zřetězení* $\alpha\beta$ vznikne zapsáním slov α a β za sebe. Platí $|\alpha\beta| = |\alpha|+|\beta|$, $\alpha\varepsilon = \varepsilon\alpha = \alpha$.
- *Mocnina* řetězce α^k pro $k \in \mathbb{N}$[1] vznikne zřetězením k kopií řetězce α. Tedy $\alpha^0 = \varepsilon$, $\alpha^{k+1} = \alpha^k\alpha$.
- $\alpha[k]$ je k-tý znak slova α, indexujeme od 0 do $|\alpha| - 1$.
- $\alpha[k : \ell]$ je *podslovo* začínající k-tým znakem a končící těsně před ℓ-tým. Tedy $\alpha[k : \ell] = \alpha[k]\alpha[k + 1] \dots \alpha[\ell - 1]$. Pokud $k \geq \ell$, je podslovo prázdné. Pokud některou z mezí vynecháme, míní se $k = 0$ nebo $\ell = |\alpha|$.
- $\alpha[: \ell]$ je *prefix* (předpona) tvořený prvními ℓ znaky řetězce.
- $\alpha[k :]$ je *suffix* (přípona) od k-tého znaku do konce řetězce.
- $\#x \in \alpha$ znamená počet výskytů znaku x v řetězci α. Je to tedy počet všech i takových, že $\alpha[i] = x$.
- *Otočení* α^{R} je slovo α „čtené pozpátku“. Tedy pro $\alpha = x_1 \dots x_n$ máme $\alpha^{\mathrm{R}} = x_n \dots x_1$.
- *Jazyk* říkáme jakékoliv množině slov. Jazyky jsou tedy podmnožiny Σ^*.

[1] V tomto textu považujeme 0 za přirozené číslo.

Pozorování: Jazyky jsou podobné *rozhodovacím problémům,* které definujeme[2] jako funkce ze Σ^* do $\{0,1\}$. Rozhodovacímu problému P můžeme přiřadit jazyk $L_P = \{\alpha \in \Sigma^* \mid P(\alpha) = 1\}$. Naopak jazyku $L \subseteq \Sigma^*$ přiřadíme problém P_L takový, že $P_L(\alpha) = 1 \Leftrightarrow \alpha \in L$.[3]

1.2 Konečné automaty

Definice: *Deterministický konečný automat* (jinak řečený DFA[4]) je uspořádaná pětice $(Q, \Sigma, \delta, q_0, F)$, kde:

- Q je konečná neprázdná množina *stavů,*
- Σ je konečná neprázdná množina znaků – *abeceda,*
- $\delta : Q \times \Sigma \to Q$ je *přechodová funkce,*[5] která pro každý stav automatu a znak ze vstupu určí, do jakého stavu má automat přejít,
- $q_0 \in Q$ je *počáteční stav,*
- $F \subseteq Q$ je množina *přijímacích (neboli koncových) stavů.*

Ve zbytku tohoto oddílu budeme říkat prostě *automat.*

Definice: *Výpočet* automatu pro vstup $\alpha \in \Sigma^*$ je posloupnost stavů $s_0, s_1, \ldots, s_{|\alpha|}$ taková, že:

- $s_0 = q_0$,
- $s_{i+1} = \delta(s_i, \alpha[i])$.

Poznámka: Automat si také můžeme představit jako orientovaný graf. Jeho vrcholy odpovídají stavům, hrany přechodům mezi stavy. Každá hrana je označena jedním znakem abecedy, přičemž platí, že z každého vrcholu vede pro každý znak abecedy právě jedna hrana. Výpočet pro vstup α je pak sled[6] začínající v počátečním stavu, přičemž znaku na hranách dávají po řadě slovo α. Tento sled je jednoznačně určen slovem α.

---

[2] Viz Průvodce, kapitola Těžké problémy.

[3] Bystrý čtenář v tom poznává charakteristickou funkci podmnožiny.

[4] deterministic finite-state automaton

[5] Zde se omlouváme za nekonsistenci: malá řecká písmena jsme si vyhradili pro slova, ale toto značení pro přechodovou funkci je natolik zažité, že ho je marno měnit.

[6] Připomínáme, že *sled* v grafu je posloupnost na sebe navazujících vrcholů a hran. Od cesty se liší tím, že se v něm mohou vrcholy i hrany opakovat.

Definice: *Rozšířená přechodová funkce* $\delta^* : Q \times \Sigma^* \to Q$ je definována takto:

- $\delta^*(q, \varepsilon) = q$ pro všechna $q \in Q$,
- $\delta^*(q, \alpha x) = \delta(\delta^*(q, \alpha), x)$ pro všechna $q \in Q$, $\alpha \in \Sigma^*$, $x \in \Sigma$.

Pozorování: Pro výpočet automatu platí $s_i = \delta^*(q_0, \alpha[:i])$.

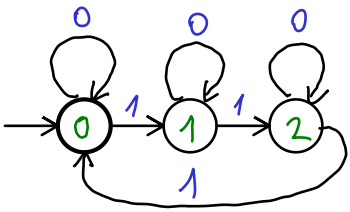
Definice: Slovo α je *přijímáno automatem*, pokud příslušný výpočet skončí v přijímacím stavu, tedy $\delta^*(q_0, \alpha) \in F$.

Definice: *Jazyk přijímaný (rozpoznávaný) automatem* je množina všech přijímaných slov, tedy $\{\alpha \in \Sigma^* \mid \delta^*(q_0, \alpha) \in F\}$. Pro automat A tento jazyk označíme $L(A)$.

Definice: Jazyk L je *regulární,* pokud je rozpoznávaný nějakým konečným automatem. Tedy existuje-li konečný automat A takový, že $L = L(A)$.

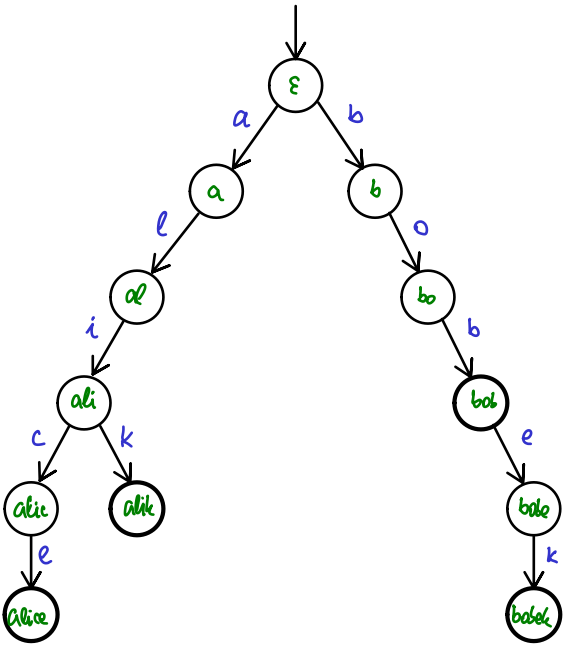
Notace: V obrázcích značíme počáteční stav šipkou z okolního prostoru do stavu, koncové stavy mají tučný okraj.

Příklad (počítání jedniček): Uvažme jazyk $L_3 = \{\alpha \in \{0, 1\} \mid (\#1 \in \alpha) \bmod 3 = 0\}$, tedy jazyk slov, jejichž počet jedniček je dělitelný třemi. Tento jazyk je regulární. O tom se snadno přesvědčíme sestrojením automatu: bude mít stavy $\{0, 1, 2\}$ odpovídající možným zbytkům po dělení počtu zatím přečtených jedniček třemi. Stav 0 bude jak počáteční, tak jediný přijímací. Viz obrázek 1.1.



Obrázek 1.1: Automat pro počet jedniček dělitelný 3

Příklad (konečné jazyky): Ukážeme, že libovolný konečný jazyk L je regulární. Automat bude mít tvar písmenkového stromu (trie) pro množinu L. Stavy tedy budou prefixy všech slov z L a navíc jeden univerzální zamítací stav z. Přechodová funkce $\delta(\alpha, x)$ pro prefix α a znak x povede do prefixu αx, pokud existuje, jinak do z. Ze z povedou všechny přechody zase do z. Počátečním stavem bude prázdný prefix ε, přijímacími stavy všechna slova z L. Viz obrázek 1.2.



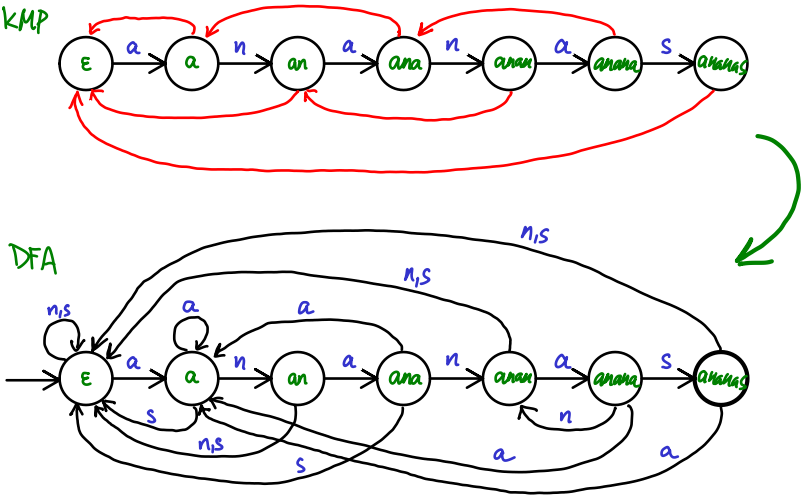
Obrázek 1.2: Automat rozpoznávající jazyk {alice, alik, bob, bobek}

Příklad (vyhledávací automaty): Vyhledávací automat⟨7⟩ typu Knuth-Morris-Pratt jde upravit na konečný automat. Množinu stavů zachováme, první stav automatu (prázdný prefix) se stane počátečním, poslední stav (prefix rovný jehle) jediným přijímacím. Přechodovou funkci definujeme pomocí funkce na jeden krok automatu, která se sama postará o použití dopředných a zpětných hran. Jazyk rozpoznávaný tímto automatem bude tvořen všemi slovy, která končí jehlou. Podobně můžeme upravit automat typu Aho-Corasicková. Viz obrázek 1.3.

Příklad (neregulární jazyk): Jazyk $L_{01} = \{0^n 1^n \mid n \in \mathbb{N}\}$ není regulární. Dokážeme to sporem. Předpokládejme, že existuje automat rozpoznávající tento jazyk. Označme t počet jeho stavů. Automat spustíme na vstupech 0^k pro $k = 0, \ldots, t$ a nazveme $s_0, \ldots, s_t$ stavy, ve kterých jednotlivé výpočty skončí. Bude tedy $s_i = \delta^*(q_0, 0^i)$.

Posloupnost $s_0, \ldots, s_t$ má $t+1$ prvků, ale ty nabývají nejvýše t hodnot. Proto se některá nutně zopakuje: máme $s_i = s_j$ pro $0 \leq i < j \leq t$. Výpočty pro slova 0^i a 0^j tedy oba

⟨7⟩ Viz Průvodce, kapitola Vyhledávání v textu.



Obrázek 1.3: Převod KMP pro slovo `ananas` na konečný automat

shodně dojdou do nějakého stavu s. Pokud za tato dvě slova přidáme libovolný suffix β, musí tedy pokračovat shodně do $\delta^*(s, \beta)$.

Takže slova $\mathtt{0}^i\mathtt{1}^i$ a $\mathtt{0}^j\mathtt{1}^i$ buďto automat obě přijme, nebo obě zamítne. Jenže první do jazyka L_{01} patří, zatímco druhé nikoliv. To je spor s předpokladem, že automat rozpoznává jazyk L_{01}.

Iterační lemma

Obrat s opakováním stavů se hodí v důkazu následujícího lemmatu:

Lemma (iterační; angl. pumping lemma): Mějme regulární jazyk L. Potom existuje číslo $n \in \mathbb{N}$ takové, že každé slovo $\omega \in L$ délky alespoň n můžeme rozložit na $\omega = \alpha\beta\gamma$, přičemž:

1. $\beta \neq \varepsilon$ ◁ *prostřední část není prázdná*
2. $|\alpha\beta| \leq n$ ◁ *první a druhá část jsou krátké*
3. Slova $\alpha\beta^t\gamma$ pro $t \geq 0$ leží všechna v L. ◁ *druhou část lze libovolně opakovat*

Důkaz: Jelikož jazyk L je regulární, existuje nějaký automat A rozpoznávající L. Za n zvolíme počet stavů tohoto automatu.

Uvažme nyní nějaké slovo $\omega \in L$ délky $m \geq n$. Označíme $s_0, \ldots, s_m$ výpočet automatu pro toto slovo, tedy $s_i = \delta^*(q_0, \omega[:i])$. Tato posloupnost má délku větší než n, ale vyskytuje se v ní nejvýše n různých stavů. Proto se nějaký stav musí opakovat: bude $s_i = s_j = s$ pro nějaké $i < j$ a stav s. Navíc k opakování nutně dojde v prvních $n+1$ prvcích, takže $j \leq n$.

Nyní zvolíme $\alpha = \omega[:i]$, $\beta = \omega[i:j]$, $\gamma = \omega[j:]$. Jelikož $\delta^*(q_0, \alpha) = s$ a $\delta^*(s, \beta) = s$, musí být $\delta^*(q_0, \alpha\beta^t) = s$ pro každé t. Proto jsou všechny stavy $\delta^*(q_0, \alpha\beta^t\gamma)$ stejné a jelikož ten pro $t = 1$ je přijímací, musí být pro všechna t přijímací. □

Příklad: Neregularitu jazyka L_{01} z předchozího příkladu můžeme snadno dokázat pumpováním. Kdyby byl regulární, uvažme slovo $\omega = \mathtt{0}^n\mathtt{1}^n$, kde n je konstanta z lemmatu. Podle lemmatu existuje rozklad $\omega = \alpha\beta\gamma$. Jelikož α a β mají dohromady nanejvýš n znaků, skládají se jenom z nul. Přidání další kopie β (nebo její odstranění) by mělo vytvořit jiné slovo jazyka. Jenže přidání β zvýší počet nul, ale zachová počet jedniček, takže slovo v jazyku ležet nemůže. To je spor.

Příklad (prvočísla): Jazyk $L_P = \{0^p \mid p \text{ je prvočíslo}\}$ také nemůže být regulární. Kdyby byl, uvažme konstantu n z lemmatu a zvolme libovolné prvočíslo $p \geq n+2$. Slovo $\mathtt{0}^p \in L_P$ tedy můžeme rozložit na části $\alpha = \mathtt{0}^i$, $\beta = \mathtt{0}^j$, $\gamma = 0^k$, kde i, j a k jsou čísla splňující $i + j + k = p$, $j > 0$, $i + j \leq n$ a $k \geq 2$.

Všechna slova $\mathtt{0}^i\mathtt{0}^{jt}\mathtt{0}^k$ pro $t \geq 0$ mají také ležet v L_P, takže $i + jt + k$ musí být pro všechna t prvočíslo. Jenže zvolíme-li $t = i + k$, je $i + jt + k = (i + k)(1 + t)$. Jelikož $i + k$ i $1 + t$ jsou větší než 1 (to první díky tomu, že $k \geq 2$), nemůže se jednat o prvočíslo. Došli jsme ke sporu.

Součin automatů

Už jsme zjistili, že otázka, zda počet jedniček ve slově je dělitelný třemi, je regulární. Podobně je regulární otázka, zda je počet nul sudý. Co kdybychom chtěli rozpoznávat slova, která splňují obě podmínky současně? K tomu se hodí představa, že oba automaty spustíme paralelně a slovo přijmeme v případě, že ho oba přijaly. To můžeme formálně popsat následující konstrukcí.

Definice: Mějme dva automaty se společnou abecedou Σ: $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ a $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$. Jejich *součin* $A_1 \times A_2$ je automat $A = (Q, \Sigma, \delta, q_0, F)$ definovaný takto:

- $Q = Q_1 \times Q_2$, ◁ *ve stavu si pamatujeme si stav obou automatů*
- $\delta((s_1, s_2), x) = (\delta_1(s_1, x), \delta_2(s_2, x))$, ◁ *simulujeme jeden krok každého automatu*
- $q_0 = (q_{01}, q_{02})$, ◁ *oba automaty začínají ve svých počátečních stavech*
- $F = F_1 \times F_2$. ◁ *přijmeme, pokud oba přijaly*

Snadno nahlédneme, že automat A přijme právě ta slova, která jsou přijata jak automatem A_1, tak A_2. Proto $L(A) = L(A_1) \cap L(A_2)$.

Důsledek: Průnik dvou regulárních jazyků je zase regulární jazyk.

Cvičení

Pro tento a následující jazyky rozhodněte, zda jsou regulární – kladnou odpověď můžete zdůvodnit sestrojením automatu, zápornou třeba pomocí iteračního lemmatu.

1. Jazyk $\{x\alpha x \mid x \in \Sigma, \alpha \in \Sigma^*\}$ pro abecedu $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Tedy jazyk slov délky aspoň 2, jejichž první a poslední znak je stejný.

2. Jazyk všech neklesajících posloupností číslic `0` ... `3`.

3. Jazyk dvojkových zápisů přirozených čísel dělitelných 5.

4. Jazyk slov nad abecedou $\{\mathtt{a}, \mathtt{b}, \mathtt{r}\}$, která začínají na některý z řetězců `abba`, `ara`, `bar`.

5. Jazyk slov nad abecedou $\{\mathtt{a}, \mathtt{b}, \mathtt{r}\}$, která končí na některý z řetězců `ara`, `bar`, `arab`.

6. Jazyk *čtverců* $\{\alpha\alpha \mid \alpha \in \{\mathtt{0}, \mathtt{1}\}^*\}$.

7. Jazyk $\{\mathtt{0}^{n^2} \mid n \in \mathbb{N}\}$.

8. Jazyk $\{\mathtt{0}^{2^n} \mid n \in \mathbb{N}\}$.

9. Dokažte, že doplněk regulárního jazyka je zase regulární. Tedy pro každý regulární jazyk $L \subseteq \Sigma^*$ je $\Sigma^* \setminus L$ také regulární.

10. Dokažte, že sjednocení regulárních jazyků je regulární.

11. Dokažte, že jazyk $\{\alpha \in \{\mathtt{0}, \mathtt{1}\}^* \mid (\#\mathtt{0} \in \alpha) = (\#\mathtt{1} \in \alpha)\}$ není regulární. Využijte toho, že $\{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$ není regulární, a že průnik dvou regulárních jazyků je regulární.

12. Definujme *uzávorkování* jako posloupnost závorek `(` a `)`, které lze spárovat tak, aby se páry nekřížily a v každém páru byla `(` před `)`. Ukažte, že jazyk všech uzávorkování není regulární.

13. Co kdybychom automatu dovolili mít nekonečně mnoho stavů? Jak by se změnilo, které jazyky můžeme rozpoznávat?

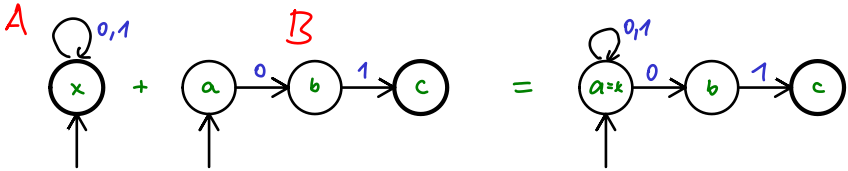
14. Vymyslete algoritmus, který pro daný automat A rozhodne, zda jazyk $L(A)$ je neprázdný.

15.* Vymyslete algoritmus, který pro daný automat A rozhodne, zda jazyk $L(A)$ je konečný.

16. Ukažte, že převod KMP na DFA z našeho příkladu lze provést v čase $\Theta(S \cdot |\Sigma|)$, kde S je počet stavů KMP.

1.3 Nedeterministické automaty

Uvažme následující příklad. Chceme popsat jazyk všech slov nad abecedou $\{0, 1\}$, která končí na 01. Taková slova můžeme složit ze dvou částí α a β, kde α je libovolný řetězec nul a jedniček a $\beta = 01$. Obě tyto části umíme rozpoznat konečnými automaty A a B, takže by bylo pěkné umět tyto automaty složit dohromady a získat tak automat pro požadovaný jazyk.



Obrázek 1.4: Slepení dvou automatů za stav

Nabízí se ztotožnit přijímací stav automatu A s počátečním stavem automatu B (jako na obrázku 1.4) a dovolit tak výpočtu přejít z A do B. Jenže vzniklý „slepený“ stav má dva přechody pro znak 0, což naše definice konečného automatu nedovoluje. Co kdybychom definici zobecnili, aby to bylo možné, a během výpočtu si pak z možných přechodů jeden vybrali? To vede k myšlence nedeterminismu.

Definice: *Nedeterministický konečný automat* (NFA) je uspořádaná pětice $(Q, \Sigma, \delta, Q_0, F)$, kde:

- Q je konečná neprázdná množina *stavů,*
- Σ je konečná neprázdná množina znaků – *abeceda,*
- $\delta : Q \times \Sigma \to 2^Q$ je *přechodová funkce,* která každé dvojici (*stav*, *znak*) přiřadí množinu všech stavů, do kterých je možné přejít,
- $Q_0 \subseteq Q$ je množina *počátečních stavů,*
- $F \subseteq Q$ je množina *přijímacích stavů.*

Z jednoho stavu tedy může vést více přechodů označených stejným znakem abecedy, ale také nemusí vést žádný.

Výpočet automatu opět odpovídá nějakému sledu v grafu. Sled začíná některým z počátečních stavů a jeho hrany jsou označeny znaky vstupního slova. Oproti deterministickému automatu ale tento sled není jednoznačně určen a nemusí ani existovat. Totéž můžeme popsat jako posloupnost stavů.

Definice: *Výpočet* nedeterministického automatu pro vstup $\alpha \in \Sigma^*$ je jakákoliv posloupnost stavů $s_0, s_1, \ldots, s_{|\alpha|}$, pro níž platí:

- $s_0 \in Q_0$,
- $s_{i+1} \in \delta(s_i, \alpha[i])$.

Definice: Slovo α je *přijímáno automatem*, pokud existuje alespoň jeden výpočet se vstupem α, který končí v jednom z přijímacích stavů. Stejně jako u DFA říkáme množině všech přijímaných slov *jazyk rozpoznávaný automatem* a značíme ho $L(A)$.

Výpočtů pro jedno slovo může být exponenciálně mnoho. Ale všimneme si, že pokud pro nějaký prefix vstupu dva výpočty skončí ve stejném stavu, mají stejnou množinu možných pokračování. Proto je nemusíme rozlišovat – stačí pro každý prefix vstupu určit množinu stavů, v nichž se může automat nacházet. To popíšeme pomocí rozšířené přechodové funkce.

Definice: *Rozšířená přechodová funkce* $\delta^* : 2^Q \times \Sigma^* \to 2^Q$ je definována takto:

- $\delta^*(S, \varepsilon) = S$ pro všechny $S \subseteq Q$,
- $\delta^*(S, \alpha x) = \bigcup_{s \in \delta^*(S,\alpha)} \delta(s, x)$ pro všechny $S \subseteq Q$, $\alpha \in \Sigma^*$, $x \in \Sigma$.

Poznámka: Slovo α je přijato automatem právě tehdy, když $\delta^*(Q_0, \alpha) \cap F \neq \emptyset$.

Příklad ze začátku oddílu ukazuje, že někdy je pohodlnější sestrojit nedeterministický automat. Nyní ukážeme, že nedeterminismu se vždy můžeme zbavit:

Věta: Ke každému NFA A existuje DFA A' takový, že $L(A') = L(A)$.

Důkaz: Budeme simulovat rozšířenou přechodovou funkci automatu A automatem A'. Stavy A' tedy budou odpovídat množinám stavů automatu A a do přechodů mezi nimi zakódujeme indukční krok z definice rozšířené přechodové funkce. Nyní precizně.

Mějme NFA $A = (Q, \Sigma, \delta, Q_0, F)$. Sestrojíme DFA $A' = (Q', \Sigma, \delta', q'_0, F')$, přičemž:

- $Q' = 2^Q$,
- $\delta'(S, x) = \bigcup_{s \in S} \delta(s, x)$ pro všechny $S \in Q'$ a $x \in \Sigma$,

- $q'_0 = Q_0$,
- $F' = \{S \in Q' \mid S \cap F \neq \emptyset\}$.

Nyní si všimneme, že výpočet automatu A' pro vstup α skončí ve stavu, který je roven $\delta^*(Q_0, \alpha)$. Tento stav je přijímací právě tehdy, když automat A přijme slovo α. □

Důsledek: Jazyk je rozpoznatelný nedeterministickým automatem právě tehdy, je-li regulární.

Příklad: Větu vyzkoušíme na NFA ze začátku kapitoly (obrázek 1.4). Stavy DFA budou

$$Q' = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\},$$

přičemž stav $\{a\}$ je počáteční a stavy $\{c\}$, $\{a,c\}$, $\{b,c\}$ a $\{a,b,c\}$ přijímací. Přechodová funkce bude vypadat následovně:

S	$\delta(S, \mathtt{0})$	$\delta(S, \mathtt{1})$
$\emptyset$	$\emptyset$	$\emptyset$
$\{a\}$	$\{a,b\}$	$\{a\}$
$\{b\}$	$\emptyset$	$\{c\}$
$\{c\}$	$\emptyset$	$\emptyset$
$\{a,b\}$	$\{a,b\}$	$\{a,c\}$
$\{a,c\}$	$\{a,b\}$	$\{a\}$
$\{b,c\}$	$\emptyset$	$\{c\}$
$\{a,b,c\}$	$\{a,b\}$	$\{a,c\}$

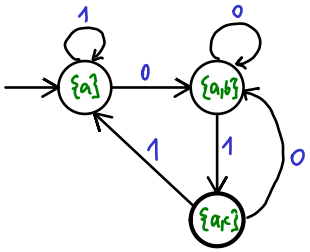
Přitom pouze stavy $\{a\}$, $\{a,b\}$ a $\{a,c\}$ budou dosažitelné z počátečního stavu, takže všechny ostatní stavy můžeme vynechat. Výsledný automat vidíte na obrázku 1.5.

Sestrojenému automatu můžeme rozumět tak, že aktuální stav obsahuje a vždy, b jen tehdy, končí-li vstup na `0`, a c jen tehdy, končí-li vstup na `01`.

Lambda-přechody

Zapojení dvou automatů „sériově“ ztotožněním stavů má jeden potenciální háček: pokud z přijímacího stavu prvního automatu vedly nějaké přechody, může výpočet přejít z druhého automatu zpátky do prvního. Lepší by bylo zavést z konce prvního automatu do začátku druhého nějaký speciální přechod, který nepoužije žádný znak ze vstupu. Takovým přechodům se říká λ-přechody a můžeme o ně definici NFA rozšířit.

Definice: *Nedeterministický konečný automat s λ-přechody (λ-NFA)* je definován stejně jako klasický NFA, jen přechodovou funkci rozšíříme na $\delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$.



Obrázek 1.5: Převod NFA z obrázku 1.4 na DFA

Výpočet automatu bude opět sled v grafu, na jehož hranách přečteme po vynechání všech λ vstupní slovo. Pozor na to, že je-li v grafu cyklus z λ-hran, může pro jeden vstup existovat nekonečně mnoho výpočtů. Popis rozšířenou přechodovou funkcí ale stále funguje a dá se zavést pomocí λ-uzávěrů:

Definice: *λ-uzávěr stavu $s \in Q$* značíme $U_\lambda(s)$ a je to množina všech stavů, do kterých se dá ze stavu s dostat po λ-hranách. Uzávěr rozšíříme na množiny stavů: $U_\lambda(S) = \bigcup_{s \in S} U_\lambda(s)$. (Všimněte si, že vždy platí $S \subseteq U_\lambda(S)$. Proto se tomu říká uzávěr.)

Pozorování: Stav je vždy dosažitelný sám ze sebe, takže vždy platí $s \in U_\lambda(s)$ a $S \subseteq U_\lambda(S)$.

Definice: *Rozšířená přechodová funkce pro λ-NFA* $\delta^* : 2^Q \times \Sigma \to 2^Q$ je definována takto:

- $\delta^*(S, \varepsilon) = U_\lambda(S)$,
- $\delta^*(S, \alpha x) = U_\lambda\left(\bigcup_{s \in \delta^*(S,\alpha)} \delta(s, x)\right)$.

Jinými slovy na samém začátku výpočtu a po průchodu každou obyčejnou hranou jsme přidali průchod po libovolně mnoha λ-hranách.

Přijímání slova a jazyk rozpoznávaný automatem definujeme stejně jako pro NFA. Důležité je, že λ-přechody je možné eliminovat a získat tak obyčejný NFA.

Věta: Pro každý λ-NFA A existuje NFA A' takový, že $L(A') = L(A)$.

Důkaz: Množinu stavů ponecháme. Množinu počátečních stavů nahradíme jejím λ-uzávěrem. Přechodovou funkci také zkombinujeme s λ-uzávěrem, tedy $\delta'(S, x) = U_\lambda(\delta(S, x))$. Koncové stavy ponecháme.

Rozšířené přechodové funkce obou automatů se budou rovnat, takže se automaty shodnou na přijetí každého slova. □

Důsledek: Jazyk je rozpoznatelný λ-NFA právě tehdy, je-li regulární.

Lemma: Každý λ-NFA je možné (při zachování rozpoznávaného jazyka) upravit tak, aby měl jediný počáteční a jediný přijímací stav. Navíc do počátečního stavu ani z přijímacího stavu nepovedou žádné přechody.

Důkaz: Přidáme nový počáteční stav, z nějž povedou λ-přechody do původních počátečních stavů. Také přidáme nový přijímací stav, do kterého zavedeme λ-přechody z původních přijímacích stavů. □

Algoritmické otázky

Jak těžké je zjistit, zda slovo patří do regulárního jazyka? Jak to závisí na délce slova n a počtu stavů automatu p? A jak na druhu automatu?

- DFA můžeme odsimulovat v čase $\mathcal{O}(n)$.
- U NFA můžeme simulovat rozšířenou přechodovou funkci v čase $\mathcal{O}(p^2)$ na krok, celkem tedy $\mathcal{O}(p^2 n)$. Nebo můžeme NFA převést podmnožinovou konstrukcí na DFA – to potrvá $\mathcal{O}(2^p \cdot p^2)$, ale pak už vstup zpracujeme v čase $\mathcal{O}(n)$.
- λ-NFA převedeme v čase $\mathcal{O}(p^2)$ na klasický NFA.

Cvičení

1. U DFA platilo, že prohodíme-li přijímací a nepřijímací stavy (tedy nahradíme F za $Q \setminus F$), dostaneme automat přijímací doplněk původního jazyka. Platí to i pro NFA?

2.* Podmnožinová konstrukce produkuje automaty s exponenciálně mnoha stavy vzhledem k počtu stavů původního automatu. I když často budou některé z nich nedosažitelné, nemusí tomu tak být vždy. Sestrojte pro každé t jazyk, který lze rozpoznat pomocí NFA s $\mathcal{O}(t)$ stavy, ale každý DFA, který ho rozpoznává, má aspoň 2^t stavů.

1.4 Regulární výrazy

Hlavní motivací pro zavádění NFA byla touha po vytváření komplikovaných regulárních jazyků z jednodušších. Ukážeme, že s λ-NFA je možné sestrojit praktickou „stavebnici regulárních jazyků“. Dokonce pak zjistíme, že z ní jdou sestavit úplně všechny regulární jazyky.

Operace s jazyky

Definice: Pro jazyky X a Y nad abecedou Σ definujeme:

- *sjednocení* $X \cup Y$ a *průnik* $X \cap Y$ jako běžné množinové operace
- *doplněk* $\overline{X} = \Sigma^* \setminus X$
- *zřetězení (konkatenaci)* $X \cdot Y = \{\alpha\beta \mid \alpha \in X \wedge \beta \in Y\}$, často tečku vynecháváme a píšeme prostě XY.
- *mocninu* X^k: $X^0 = \{\varepsilon\}$, $X^{k+1} = X^k \cdot X$. (Zjevně $X^1 = X$, $X^2 = XX$, $X^3 = XXX$, přičemž uzávorkování není třeba určit, neboť zřetězení je asociativní.)
- *iteraci* $X^* = \bigcup_{n \geq 0} X^n$
- *pozitivní iteraci* $X^+ = \bigcup_{n \geq 1} X^n$. (Platí tedy $X^* = \{\varepsilon\} \cup X^+$.)
- *otočení* $X^{\mathrm{R}} = \{\alpha^{\mathrm{R}} \mid \alpha \in X\}$.

Věta: Pokud X a Y jsou regulární, všechny operace produkují opět regulární jazyky.

Důkaz: Automat pro *průnik* regulárních jazyků už umíme získat součinovou konstrukcí. *Doplněk* jsme vyřešili v cvičení 1.2.9, *otočení* vyřešíme ve cvičení 1.

Pro zbývající operace ukážeme, že z automatů pro X a Y umíme sestrojit automat pro výsledný jazyk. Budou se nám k tomu hodit λ-automaty s jednoznačným počátečním i koncovým (přijímacím) stavem. Automat pro X označme A_X, jeho počáteční stav a_X a koncový stav z_X. Podobně pro Y.

Pro *sjednocení* vytvoříme nový počáteční stav a a nový koncový z. Přidáme λ-přechody $a \to a_X$, $a \to a_Y$, $z_X \to z$, $z_Y \to z$.

Pro *zřetězení* stačí přidat λ-přechod ze z_X do a_Y. Počátečním stavem bude a_X, koncovým z_Y.

Mocninu realizujeme jako k-násobné zřetězení (pro $k = 0$ stačí užít fakt, že každý konečný jazyk je regulární).

Pro *pozitivní iteraci* stačí přidat λ-přechod z koncového stavu do počátečního. Obecná *iterace* potřebuje přijímat navíc slovo ε: vytvoříme nový počáteční stav a a koncový z, přidáme λ-přechody $a \to z$, $a \to a_X$, $z_X \to z$, $z_X \to a_X$. □

Regulární výrazy

Postup, jak jazyk získat z jednodušších pomocí jazykových operací, můžeme popsat regulárním výrazem. To je buďto konstanta vyjadřující nějaký elementární jazyk, nebo opera-

ce aplikovaná na jednodušší regulární výrazy. Každému regulárnímu výrazu pak můžeme přiřadit nějaký jazyk *generovaný výrazem*, který budeme značit obvyklým $L(\ldots)$.

Výčet operací najdete na obrázku 1.6.

$\emptyset$	*prázdný jazyk*	$L(\emptyset) = \emptyset$
ε	*prázdné slovo*	$L(\varepsilon) = \{\varepsilon\}$
x	*znak abecedy*	$L(x) = \{x\}$
$X \mid Y$	*sjednocení*	$L(X \mid Y) = L(X) \cup L(Y)$
XY	*zřetězení*	$L(XY) = L(X) \cdot L(Y)$
X^*	*iterace*	$L(X^*) = L(X)^*$
X^+	*pozitivní iterace*	zkratka za XX^*
$X?$	*možnost*	zkratka za $X \mid \varepsilon$

Obrázek 1.6: Regulární výrazy a jimi generované jazyky

Příklad: Slova z $\{0,1\}^*$ končící na `01` můžeme popsat regulárním výrazem $(0 \mid 1)^*01$.

Příklad: Slova z $\{0,1\}^*$, ve kterých se pravidelně střídají `0` a `1`, můžeme popsat výrazem $(0 \mid 1)^* \mid (1 \mid 0)^* \mid (0 \mid 1)^*0 \mid (1 \mid 0)^*1$, případně jednodušeji $1?(0 \mid 1)^*0?$.

Už víme, že jazyky generované regulárnimi výrazy jsou vždy regulární. Překvapivě tak jde vygenerovat úplně každý regulární jazyk:

Věta (Kleeneho): Jazyk je generovaný regulárním výrazem právě tehdy, je-li regulární.

Důkaz: Zbývá dokázat implikaci zprava doleva. Mějme nějaký regulární jazyk L a DFA A, který tento jazyk rozpoznává. Automat budeme postupně zmenšovat, přičemž hrany mezi stavy budou ohodnoceny nejen znaky abecedy, ale obecnými regulárními výrazy. Výpočet může hranou projít, kdykoliv přečteme ze vstupu slovo vyhovující danému regulárnímu výrazu.

Nejprve přidáme nový počáteční stav a a přijímací stav z tak, aby do a ani ze z nevedly žádné hrany. To provedeme stejně jako u λ-NFA, přičemž roli λ-přechodu zde má hrana ohodnocená regulárním výrazem ε.

Nyní na automat budeme aplikovat dva typy úprav:

- *Sloučení hran* – pokud mezi nějakými dvěma stavy vede více paralelních hran, nahradíme je jedinou hranou ohodnocenou sjednocením regulárních výrazů z původních hran.
- *Eliminace stavu* – vybereme si nějaký stav s různý od počátečního a přijímacího. Tento stav odstraníme a zařídíme, aby všechny výpočty, které procházely tímto stavem,

použily nějakou „zkratku“, které ho obejde. Pro každou dvojici stavů x a y takovou, že z x vede do s hrana s ohodnocením R_x a z s do y hrana s ohodnocením y, přidáme zkratku z x do y:

- Pokud ve stavu s není smyčka (hrana z s do s), bude zkratka ohodnocena výrazem $R_x R_y$.
- Pokud ve stavu s je smyčka s ohodnocením R_s, zkratku ohodnotíme výrazem $R_x R_s^* R_y$.

Tyto kroky budeme střídat, než z automatu zbude pouze počáteční a přijímací stav, mezi nimiž povede jediná hrana. Jelikož oba druhy úprav zachovávají jazyk rozpoznávaný automatem, ohodnocením zbývající hrany bude regulární výraz generující jazyk automatu.

Ještě dodejme, že stejný postup by fungoval i pro NFA nebo λ-NFA. □

Příklad: Postup si vyzkoušíme na jazyku dvojkových čísel dělitelných třemi. Sestrojit regulární výraz pro tento jazyk není přímočaré, tak to zkusíme pomocí Kleeneho věty. Na obrázku 1.7 vidíme automat rozpoznávající tento jazyk (inspiraci nacházíme v cvičení 1.2.3) a jednotlivé kroky z důkazu věty. Dostáváme regulární výraz

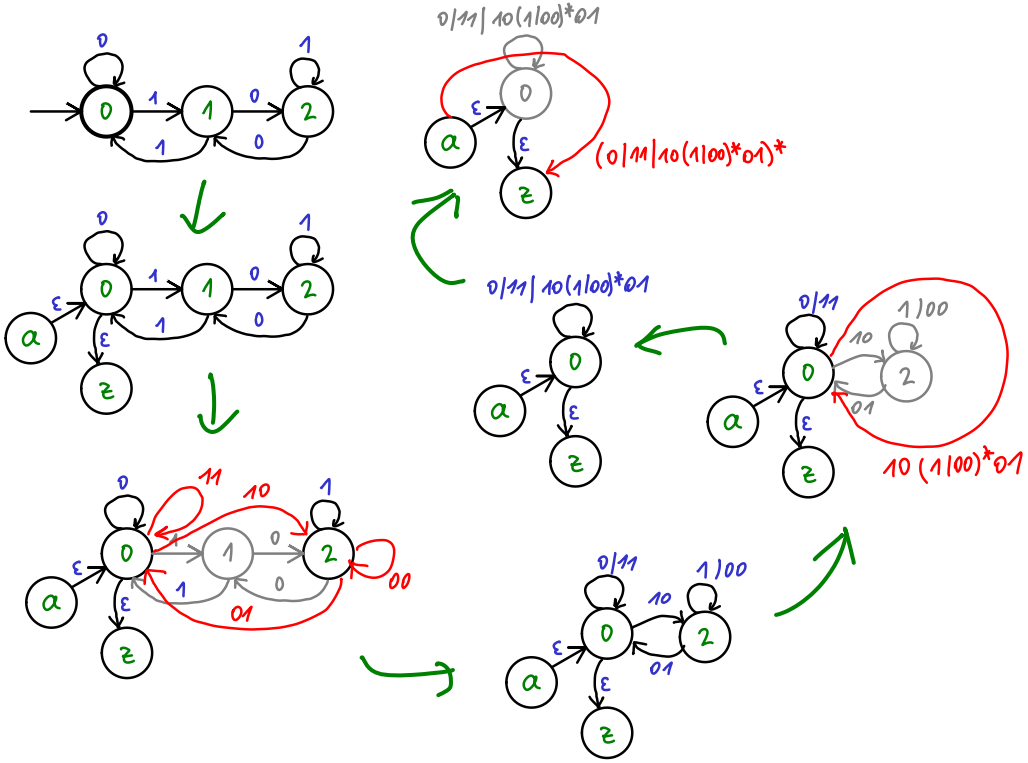
$$(\mathtt{0} \mid \mathtt{11} \mid \mathtt{10}(\mathtt{1} \mid \mathtt{00})^*\mathtt{01})^*.$$

Zkuste najít vysvětlení jednotlivých částí výrazu, aniž byste se odkázali na automat.

Poznámka: UNIXové nástroje (například `grep` a `sed` používají nejrůznější varianty regulárních výrazů, které se od těch našich liší pouze detaily notace. Jinde (například v Perlu) ale najdeme „regulární“ výrazy vybavené i schopností zpětných odkazů a rekurze. Ty dokáží popsat i mnohé neregulární jazyky, ale algoritmy používané k jejich vyhodnocování nemají polynomiální časovou složitost.

Cvičení

1. Dokažte, že pro regulární jazyk X je jeho otočení X^{R} také regulární.
2. Jak vypadá jazyk X^{**}?
3. Napište regulární výraz, který generuje jazyk všech slov nad abecedou $\{\mathtt{0}, \mathtt{1}\}$, jejichž počet jedniček je sudý nebo dělitelný třemi. Sestrojte odpovídající λ-NFA, ten převeďte na NFA a ten konečně na DFA.
4. Napište regulární výraz pro jazyk všech desítkových zápisů přirozených čísel (nedovolujeme nestandardní zápisy typu `0123` nebo ε). Podobně pro desetinná čísla (např. `3.1415`) a desetinná s periodou (značíme `0.0[01]`).



Obrázek 1.7: Převod automatu pro dělitelnost 3 na regulární výraz

5. Napište regulární výraz pro jazyk všech slov nad abecedou $\{0, \ldots, 9\}$, která jsou neklesající (1377 v tomto jazyce leží, 735 nikoliv).

6. Vytvořte regulární výraz pro dvojkové zápisy čísel dělitelných třemi. Doporučujeme nejprve sestrojit DFA a pak použít konstrukci z důkazu Kleeneho věty.

7. Uvažme jazyk všech slov nad abecedou $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, která neobsahují podslovo abc. Popište tento jazyk regulárním výrazem. Pokuste se o to přímo. Pak zkuste vytvořit DFA pro jazyk slov obsahujících abc, podle cvičení 1.2.9 vyrobit DFA pro jeho doplněk a na ten použít konstrukci z důkazu Kleeneho věty.

8. Navrhněte co nejefektivnější algoritmus, který zjistí, zda zadané slovo je generované zadaným regulárním výrazem.

9.* V důkazu Kleeneho věty jsme použili automat, který má na hranách regulární výrazy. Tento koncept můžeme zobecnit: *automat 2. řádu* má na každé hraně nějaký jazyk, hranou je možné projít, pokud ze vstupu přečteme slovo z jejího jazyka. To zahrnuje nedeterminismus i automaty s λ-přechody. Definujte pro tyto automaty výpočet a rozšířenou přechodovou funkci. Dokažte, že pokud jazyky hran jsou regulární, jazyk rozpoznávaný automatem je také regulární.

10.* *Homomorfismus* je libovolné zobrazení $h : \Sigma^* \to \Delta^*$ takové, že $h(\varepsilon) = \varepsilon$ a $h(\alpha\beta) = h(\alpha)h(\beta)$. Je tedy jednoznačně určen obrazy jednoznakových řetězců, tj. funkcí $h_1 : \Sigma \to \Delta^*$. Homomorfismus můžeme rozšířit na jazyky: $h(L) = \{h(\alpha) \mid \alpha \in L\}$. Dokažte, že je-li L regulární, pak $h(L)$ také.

11.* *Substituce* je zobecnění homomorfismu, které znakům nepřiřazuje slova, ale rovnou jazyky. Je tedy určený nějakou funkcí $s : \Sigma \to 2^{\Delta^*}$, kterou můžeme rozšířit na slova: $s(\alpha\beta) = s(\alpha) \cdot s(\beta)$, a dokonce na jazyky: $s(L) = \bigcup_{\alpha \in L} s(\alpha)$. Tedy pokud L je jazyk nad abecedou Σ, je $s(L)$ jazyk nad abecedou Δ. Dokažte, že je-li L regulární a všechny $s(x)$ regulární, tak $s(L)$ je také regulární.

12. Použití substituce: Jazyky $\{\mathtt{a}, \mathtt{b}\}$, $\{\mathtt{ab}\}$ a $\{\mathtt{a}^*\}$ jsou regulární. Pomocí substituce dokažte, že sjednocení, zřetězení a iterace libovolných regulárních jazyků jsou opět regulární.

13.* *Jiný důkaz Kleeneho věty:* Inspirujte se Floydovým-Warshallovým algoritmem na výpočet matice vzdáleností v grafu (kapitola 6.4 v Průvodci). Stavy očíslujeme od 1 do n. Pak pro k od 0 do n definujeme regulární výrazy R^k_{ij} popisující všechny sledy ze stavu i do stavu j, jejichž vnitřní stavy leží v množině $\{1, \ldots, k\}$. Ukažte, jak tyto výrazy počítat indukcí podle k a jak z nich získat regulární výraz pro jazyk automatu.

2 Bezkontextové jazyky

Jedním z přístupů k popisu syntaxe lidských jazyků jsou *generativní gramatiky*. Ty popisují strukturu věty pravidly, která říkají, jak větu rozložit na čím dál jednodušší části. Pro fragment češtiny by to mohlo vypadat třeba takto:

(1) *věta* → *podmět přísudek předmět*

(2) *podmět* → *atributy podstatné-jméno*

(3) *atributy* → ε | *atributy přídavné-jméno*

(4) *přísudek* → *sloveso*

(5) *předmět* → ε | *atributy podstatné-jméno*

(6) *podstatné-jméno* → `pán` | `vlk` | `dýmka` | `domovník`

(7) *přídavné-jméno* → `šedý` | `voňavý` | `nevrlý`

(8) *sloveso* → `nese` | `žere` | `spí`

Svislou čárou značíme výběr z několika možností.

Platnými větami podle této gramatiky jsou například (po správném doplnění koncovek):

- Pán spí.
- Vlk nese dýmku.
- Nevrlý pán nese šedého vlka.
- Nevrlý vlk žere šedou voňavou dýmku.

Gramatiku také můžeme použít na vytvoření platné věty. Začneme symbolem *věta* a postupně nahrazujeme symboly podle pravidel, až nám zbudou samá slova. Například:

- *věta*
- *podmět přísudek předmět* ◁ *použitím (1)*
- *podmět sloveso předmět* ◁ *použitím (4)*
- *podmět sloveso* ◁ *použitím (5), varianta 1*
- *atributy podstatné-jméno sloveso* ◁ *použitím (2)*
- *atributy přídavné-jméno podstatné-jméno sloveso* ◁ *použitím (3), varianta 2*

- *přídavné-jméno podstatné-jméno sloveso* ◁ *použitím (3), varianta 1*
- `nevrlý` *podstatné-jméno sloveso* ◁ *použitím (7)*
- `nevrlý` *podstatné-jméno* `žere` ◁ *použitím (8)*
- `nevrlý vlk žere` ◁ *použitím (6)*

Lidské jazyky se nakonec pro tento přístup ukázaly být příliš komplikované a nepravidelné. Ale systém generativních gramatik se osvědčil při popisování formálních jazyků, jako jsou třeba jazyky programovací.

2.1 Gramatiky a derivace

Definice: *Gramatika* je uspořádaná čtveřice (V, T, S, P), kde:

- V je konečná neprázdná množina *proměnných* (neterminálních symbolů),
- T je konečná neprázdná množina *terminálních symbolů* (neboli terminálů),
- $S \in V$ je počáteční proměnná,
- P je konečná množina *pravidel* typu $\alpha \to \beta$, kde $\alpha, \beta \in (V \cup T)^*$ a α obsahuje aspoň jednu proměnnou.

Pravidla tedy říkají, jak proměnné (nebo nějaká delší slova obsahující proměnné) přepisovat na další proměnné a terminály. Nakonec se všech proměnných zbavíme a zbude slovo tvořené pouze terminály. Postup přepisování nyní popíšeme formálně:

Definice: Slovo α se *přímo přepíše* na slovo β (značíme $\alpha \Longrightarrow \beta$), pokud existuje rozklad těchto slov $\alpha = \lambda\gamma\pi$ a $\beta = \lambda\delta\pi$, kde $(\gamma \to \delta) \in P$ je pravidlo gramatiky. Přitom všechna slova $\alpha, \beta, \gamma, \delta, \lambda, \pi$ leží v $(V \cup T)^*$.

Definice: Slovo α se *přepíše* na slovo β (značíme $\alpha \overset{*}{\Longrightarrow} \beta$), pokud existuje posloupnost slov $\beta_0, \ldots, \beta_n \in (V \cup T)^*$ taková, že $\beta_0 = \alpha$, $\beta_n = \beta$ a pro všechna i je $\beta_i \Longrightarrow \beta_{i+1}$. Posloupnosti $\beta_0, \ldots, \beta_n$ říkáme *odvození* neboli *derivace* slova β z α.

Jinak řečeno: $\alpha \Longrightarrow \beta$ znamená, že existuje pravidlo, kterým se dá nějaké podslovo slova α přepsat tak, aby z α vzniklo β. A $\alpha \overset{*}{\Longrightarrow} \beta$ znamená, že postupným přepisováním podle pravidel se dá z α vyrobit β. Jelikož derivace může být jednoprvková, platí vždy $\alpha \overset{*}{\Longrightarrow} \alpha$.

Definice: Slovo $\alpha \in T^*$ je *generované gramatikou,* pokud $S \overset{*}{\Longrightarrow} \alpha$. Množině všech takových slov říkáme *jazyk generovaný gramatikou* a značíme ho $L(G)$.

Příklad: Uvažme gramatiku s $V = \{S\}$, $T = \{\mathtt{0}, \mathtt{1}\}$ a pravidly:

- $S \to \varepsilon$
- $S \to \mathtt{0}S$
- $S \to \mathtt{1}S$

Všechny derivace z S vypadají tak, že k S postupně připisujeme nuly a jedničky zleva, až nakonec S vypustíme přepsáním na ε. Gramatika tedy generuje jazyk $\{\mathtt{0},\mathtt{1}\}^*$.

Příklad: Nyní gramatiku upravíme, aby generovala pouze posloupnosti se sudým počtem jedniček. Terminály budou opět $T = \{\mathtt{0},\mathtt{1}\}$, proměnné $V = \{S, L\}$ a pravidla:

- $S \to \varepsilon$
- $S \to \mathtt{0}S$
- $S \to \mathtt{1}L$
- $L \to \mathtt{0}L$
- $L \to \mathtt{1}S$

Derivace z S opět obsahují řetězce tvořené nulami, jedničkami a jednou proměnnou na konci. Tato proměnná je S, pokud jsme zatím zapsali sudý pocet jedniček, a L, pokud lichý. Tím pádem pouze S smíme přepsat na ε a tím derivaci ukončit.

Příklad: Gramatikou můžeme generovat i neregulární jazyky, například náš oblíbený protipříklad $\{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$. Postačí nám proměnná $P = \{S\}$, terminály $T = \{\mathtt{0},\mathtt{1}\}$ a dvě pravidla:

- $S \to \varepsilon$
- $S \to \mathtt{0}S\mathtt{1}$

Možné derivace z S jsou postupně $\mathtt{0}S\mathtt{1}$, $\mathtt{00}S\mathtt{11}$, $\mathtt{000}S\mathtt{111}$, ...

Obecná definice pravidla připouští, aby levé strany byly libovolně dlouhé. Zatím nám ale stačila pravidla mnohem jednoduššího tvaru:

Definice:

- Gramatika je *bezkontextová,* pokud všechna její pravidla jsou tvaru $X \to \alpha$, kde X je proměnná. Těmto gramatikám se říká *CFG (context-free grammar).*
- Gramatika je *pravá lineární,* pokud všechna její pravidla jsou tvaru buď $X \to \varepsilon$ nebo $X \to \alpha Y$, kde $X, Y \in V$ a $\alpha \in T^*$. Těmto gramatikám budeme říkat *RLG (right-linear grammar).*

Také můžeme definovat různé třídy jazyků podle toho, jakými gramatikami je možné je vygenerovat:

Definice:

- $\mathcal{L}$ je třída všech jazyků.
- $\mathcal{L}_0$ je třída jazyků, které se dají vygenerovat gramatikou.
- $\mathcal{L}_2$ je třída *bezkontextových jazyků,* tedy těch, které se dají vygenerovat bezkontextovou gramatikou.
- $\mathcal{L}_3$ je třída jazyků, které se dají vygenerovat pravou lineární gramatikou.

Pozorování: $\mathcal{L}_3 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}_0 \subseteq \mathcal{L}$. Časem ukážeme, že všechny tři inkluze jsou ostré.

Poznámka: Třídy $\mathcal{L}_0$, $\mathcal{L}_2$, $\mathcal{L}_3$ jsou součástí takzvané Chomského hierarchie jazyků. V té figuruje i třída $\mathcal{L}_1$, k níž se vrátíme v cvičení 3.5.3.

Cvičení
Vygenerujte gramatikou následující jazyky.

1. Jazyk *sudých palindromů* $\{\alpha\alpha^{\mathrm{R}} \mid \alpha \in \{\mathtt{a},\mathtt{b}\}^*\}$.
2. Jazyk *palindromů* $\{\alpha \in \{\mathtt{a},\mathtt{b}\}^* \mid \alpha = \alpha^{\mathrm{R}}\}$.
3. Jazyk $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in N\}$.

   Pozor, zde nestačí bezkontextová pravidla.

4.* Jazyk $\{\mathtt{0}^{2^n} \mid n \in N\}$.

5.* Jazyk *čtverců* $\{\alpha\alpha \mid \alpha \in \{\mathtt{a},\mathtt{b}\}^*\}$.

6. Ukažte, že každý regulární výraz se dá přeložit na gramatiku, která generuje tentýž jazyk. Lze to provést i přímo bez převodu na automaty a zpět.

2.2 Lineární gramatiky

V obou příkladech generování regulárního jazyka jsme použili pravou lineární gramatiku (RLG). To není náhoda: tyto gramatiky vždy generují regulární jazyky, a dokonce všechny takové.

Lemma: Každý regulární jazyk se dá vygenerovat pomocí RLG.

Důkaz: Mějme jazyk L rozpoznávaný nějakým DFA $(Q, \Sigma, \delta, q_0, F)$. Sestrojíme gramatiku (V, T, S, P) s proměnnými $V = Q$, terminály $T = \Sigma$, počáteční proměnnou $S = q_0$. Bude obsahovat pravidla:

- $X \to aY$ pro všechna $X, Y \in Q$, $a \in \Sigma$ taková, že $\delta(X, a) = Y$
- $X \to \varepsilon$ pro všechna $X \in F$

Indukcí dokážeme, že z S můžeme odvozovat slova tvaru αX, kde $\alpha \in T^*$ a $X = \delta^*(q_0, \alpha)$. Proměnné na konci se můžeme zbavit pouze tehdy, je-li $X \in F$, tedy pokud slovo α patří do jazyka L. Gramatika tedy generuje jazyk L. □

Lemma: Každá RLG generuje regulární jazyk.

Důkaz: Nabízí se použít opačný postup k převodu RLG na automat. Ale musíme se vypořádat s několika překážkami:

- V gramatice mohou současně pravidla $X \to aY$ i $X \to aY'$. To nevadí, prostě vyrobíme nedeterministický automat a posléze se osvědčeným způsobem nedeterminismu zbavíme.
- Pravidla tvaru $X \to \alpha Y$ pro α jiné než jednoznakové:
  - $X \to \varepsilon Y$ – můžeme přeložit na λ-přechody a následně převést λ-NFA na NFA.
  - $X \to a_1 a_2 \dots a_n Y$ pro $n > 1$ – zavedeme pomocné proměnné $X_1, \dots, X_{n-1}$ a pravidlo nahradíme množinou pravidel $X \to a_1 X_1$, $X_1 \to a_2 X_2$, …, $X_{n-2} \to a_{n-1} X_{n-1}$, $X_{n-1} \to a_n X$. Rozmyslete si, že touto úpravou nezměníme jazyk generovaný gramatikou.

Gramatiku (V, T, S, P) tedy nejprve upravíme, aby neobsahovala „dlouhá“ pravidla. Pak vytvoříme λ-NFA $(Q, \Sigma, \delta, Q_0, F)$, kde:

- $Q = V$
- $\Sigma = T$
- $\delta(X, a) = \{Y \in V \mid (X \to aY) \in P\}$
- $\delta(X, \lambda) = \{Y \in V \mid (X \to Y) \in P\}$
- $Q_0 = \{S\}$
- $F = \{X \in V \mid (X \to \varepsilon) \in P\}$

Výpočty NFA odpovídají derivacím z gramatiky, takže NFA rozpoznává jazyk generovaný gramatikou. □

Důsledek: Třída $\mathcal{L}_3$ jazyků generovaných RLG je rovna třídě všech regulárních jazyků. Jelikož jsme pro neregulární jazyk všech $\mathtt{0}^n\mathtt{1}^n$ našli bezkontextovou gramatiku, třídy $\mathcal{L}_2$ a $\mathcal{L}_3$ jsou různé.

Poznámka: Postup použitý při převodu RLG na automat je hodně typický: Nejprve gramatiku *normalizujeme,* tedy převedeme do nějakého speciálního tvaru s co nejjednoduššími pravidly. Další úvahy se tím výrazně zjednoduší.

Existují i jiné typy lineárních gramatik:

Definice:

- Gramatika je *levá lineární (LLG),* pokud všechna její pravidla jsou buď tvaru $X \to \varepsilon$ nebo $X \to Y\alpha$ pro $X, Y \in V$ a $\alpha \in T^*$.
- Gramatika je *lineární,* pokud všechna její pravidla jsou buď tvaru $X \to \varepsilon$ nebo $X \to \alpha Y \beta$ pro $X, Y \in V$ a $\alpha, \beta \in T^*$.

Pozorování: Díky symetrii mezi levými a pravými lineárními gramatikami platí, že slovo x lze vygenerovat pomocí LLG právě tehdy, když x^{R} lze vygenerovat pomocí RLG. Jelikož otočením regulárního jazyka je zase regulární jazyk (cvičení 1.4.1), generují LLG také regulární jazyky.

Pozorování: Jinak to je s obecnými lineárními gramatikami. Jazyk všech $\mathtt{0}^n\mathtt{1}^n$ jsme totiž generovali pravidly $S \to \mathtt{0}S\mathtt{1}$ a $S \to \varepsilon$, čili lineární gramatikou. Vidíme, že lineární gramatiky umí vygenerovat i některé bezkontextové jazyky (ale ne všechny – viz cvičení 2.3.13).

Cvičení

1. Jakou třídu jazyků generují gramatiky, které mohou obsahovat jak levá lineární pravidla, tak pravá lineární?

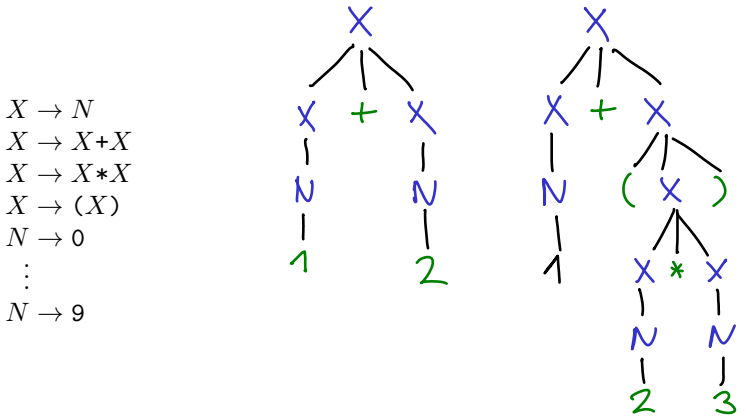
2.3 Bezkontextové gramatiky

Většina gramatik, které se v praxi používají, je bezkontextová. Má to svůj důvod: tyto gramatiky jsou dostatečně silné, aby se pomocí nich daly popsat běžné jazyky, ale stále dostatečně omezené, aby s nimi spojené algoritmy byly efektivní. Pojďme je prostudovat trochu blíže.

Derivační stromy

Především si všimneme, že derivaci slova z bezkontextové gramatiky lze popsat pomocí stromu. Do kořene umístíme počáteční proměnnou. Děti kořene budou symboly, na které jsme počáteční proměnnou přepsali. Některé z nich jsou opět proměnné a ty jsme museli časem také přepsat – jako jejich děti nakreslíme, na co jsme je přepsali, a tak dále.

Podívejme se na příklad derivačních stromů na obrázku 2.1. Stromy trochu připomínají větné rozbory z hodin češtiny – to není náhoda, stromová struktura věty je vytvořena stejným způsobem.



Obrázek 2.1: Gramatika pro výrazy a derivační stromy výrazů 1+2 a 1+(2*3)

Definice: *Derivační (neboli syntaktický) strom* je zakořeněný strom s uspořádanými dětmi každého vrcholu,[⟨1⟩] přičemž:

- Vnitřní vrcholy stromu jsou ohodnoceny proměnnými gramatiky, v kořeni je počáteční proměnná.
- Listy stromu jsou ohodnoceny terminály, proměnné a symbolem ε.
- Pro každý vrchol s ohodnocením X, jehož synové jsou ohodnoceni $Y_1, \ldots, Y_m \neq \varepsilon$ platí, že existuje pravidlo gramatiky $X \to Y_1 \ldots Y_m$. Pokud má některý ze synů ohodnocení ε, je to jediný syn a pravidlem gramatiky je $X \to \varepsilon$.

[⟨1⟩] Kombinatorici takovým stromům říkají *pěstované*.

Derivační strom *odvozuje* slovo $\alpha \in (V \cup T)^*$, pokud projdeme-li listy stromu „zleva doprava“ (v pořadí daném prohledáváním do hloubky) a zřetězíme jejich ohodnocení, získáme slovo α. Přitom ε se chová jako prázdný řetězec.

Poznámka: Generování gramatikou se obvykle definuje jen pro posloupnosti terminálů, ale odvození derivačním stromem jsme zavedli i pro „nehotová“ slova, ve kterých ještě zbývají proměnné. To nám usnadní následující úvahy.

Věta: Gramatika generuje slovo $\alpha \in T^*$ právě tehdy, když existuje derivační strom, jenž odvozuje α.

Důkaz: Implikace zleva doprava: Nechť α je slovo generované gramatikou a $\beta_0, \ldots, \beta_n$ je jeho derivace ($\beta_0 = S$, $\beta_n = \alpha$). Budeme postupně vytvářet derivační stromy $\mathcal{S}_0, \ldots, \mathcal{S}_n$ takové, že $\mathcal{S}_i$ odvozuje slovo β_i. Strom $\mathcal{S}_0$ je pouze kořen ohodnocený proměnnou S. Nyní chceme z $\mathcal{S}_i$ vytvořit $\mathcal{S}_{i+1}$. Slovo α_{i+1} vznikne z α_i přepsáním nějaké proměnné X podle pravidla $X \to Y_1 \ldots Y_m$. Najdeme tedy v $\mathcal{S}_i$ list ohodnocený X a pod něj připojíme nové listy ohodnocené $Y_1, \ldots, Y_m$. Tím vznikne strom odvozující slovo α_{i+1}. (Okrajový případ: pravidlo může také znít $X \to \varepsilon$ – v tom případě připojíme pod X nový list s ε.)

Implikaci zprava doleva dokážeme indukcí podle hloubky derivačního stromu. Pokud má strom nulovou hloubku, je tvořen pouze kořenem, takže odvozuje slovo S. To je bezpochyby generované gramatikou. Nyní uvažujme nějaký strom $\mathcal{S}$ hloubky $h > 0$, který odvozuje slovo α. Odřízneme-li z něj všechny listy, vznikne strom T' hloubky $h-1$ odvozující nějaké slovo α', tvořené výlučně proměnnými. Podle indukčního předpokladu existuje derivace slova α' z gramatiky. Tuto derivaci upravíme na derivaci slova α: za každý list stromu T' přidáme přepsání proměnné z tohoto listu na potomky, které měl v T. □

Překladače a jednoznačnost gramatik

Syntaktická analýza pomocí gramatik a derivačních stromů se často používá v překladačích programovacích jazyků. Nejprve text programu projde *lexikální analýzou,* která identifikuje „slovní druhy“, jako třeba čísla, identifikátory, operátory apod. a každému z nich přiřadí terminál gramatiky. Následuje *syntaktická analýza* podle gramatiky, jejímž výsledkem je syntaktický (derivační) strom. Ze stromu se pak odvozuje *sémantika* (význam) programu.

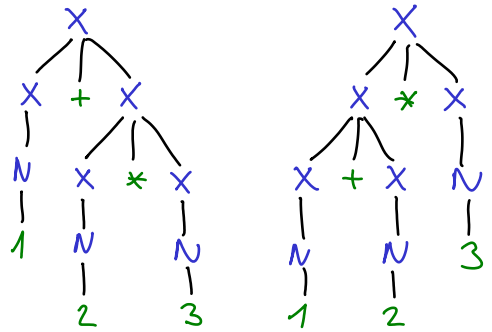
V našem příkladu s výrazy například můžeme prohledáním stromu do hloubky vyčíslit hodnotu výrazu: stačí si z každého podstromu vracet jeho hodnotu.

Tento přístup ovšem naráží na problém: Pokud není gramatika sestavena šikovně, může pro jeden řetězec existovat více různých derivačních stromů (kterým pak je přiřazena různá sémantika). Takové gramatice se říká *nejednoznačná.*

Nejednoznačnost se ostatně projevuje i v našem příkladu, viz obrázek 2.2. Potíž je v tom, že naše gramatika neví nic o tom, že násobení má přednost před sčítáním a že sčítání

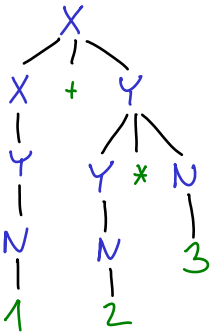
i násobení se vyhodnocují zleva doprava. Obojí lze do gramatiky zabudovat a vznikne gramatika z obrázku 2.3, která už je jednoznačná.

$X \to N$
$X \to X\texttt{+}X$
$X \to X\texttt{*}X$
$X \to \texttt{(}X\texttt{)}$
$N \to \texttt{0}$
$\vdots$
$N \to \texttt{9}$



Obrázek 2.2: Dva různé derivační stromy výrazu 1+2*3

$X \to Y$
$X \to X\texttt{+}Y$
$Y \to N$
$Y \to Y\texttt{*}N$
$N \to \texttt{(}X\texttt{)}$
$N \to \texttt{0}$
$\vdots$
$N \to \texttt{9}$



Obrázek 2.3: Jednoznačná verze gramatiky a derivační strom výrazu 1+2*3

Poznámka: Existují dokonce *nejednoznačné bezkontextové jazyky,* ke kterým nelze sestrojit jednoznačnou gramatiku. Příkladem takového jazyka je

$$\{\texttt{a}^n\texttt{b}^m\texttt{c}^m\texttt{d}^n \mid m,n>0\} \cup \{\texttt{a}^n\texttt{b}^n\texttt{c}^m\texttt{d}^m \mid m,n>0\}.$$

Tento jazyk je bezkontextový (zkuste sestrojit gramatiku), ale pro každou jeho gramatiku mají řetězce tvaru $\texttt{a}^n\texttt{b}^n\texttt{c}^n\texttt{d}^n$ více derivačních stromů. To nicméně nebudeme dokazovat.

Chomského normální forma

Občas se hodí uvažovat gramatiky s co nejjednodušší strukturou pravidel.

Definice: Gramatika je v *Chomského normální formě (ChNF),* pokud obsahuje pouze pravidla tvarů $X \to t$, $X \to YZ$ a $S \to \varepsilon$, kde X, Y, Z jsou proměnné a t terminál. Počáteční proměnná S se navíc nevyskytuje na pravé straně žádného pravidla.

Poznámka: Každá gramatika v ChNF je bezkontextová. Pravidlo $S \to \varepsilon$ jsme museli dovolit, aby se dalo vygenerovat prázdné slovo. Toto pravidlo ovšem neinteraguje s ostatními pravidly, protože S lze získat pouze na začátku derivace.

Definice: Dvě gramatiky jsou *ekvivalentní,* pokud generují tentýž jazyk.

Věta: Pro každou bezkontextovou gramatiku existuje ekvivalentní gramatika v Chomského normální formě.

Důkaz: Převod do ChNF provedeme v několika krocích. Každý odstraňuje jeden typ pravidel, který je v normální formě zakázaný. Pokaždé snadno ověříme, že se jazyk generovaný gramatikou nezměnil.

1. S na pravé straně. Vytvoříme novou proměnnou S'. Všechny výskyty S na pravé straně pravidla nahradíme za S'. Kdykoliv má pravidlo S na levé straně, vytvoříme jeho kopii s S' na levé straně.

2. Terminály na netriviální pravé straně. Tím myslíme případy, kdy pravá strana nějakého pravidla obsahuje buď více terminálů, nebo kombinaci terminálů a proměnných. Pro každý terminál a vytvoříme proměnnou T_a a pravidlo $T_a \to a$. Všechny výskyty a na netriviálních pravých stranách nahradíme za T_a.

3. Dlouhé pravé strany. Nyní všechny pravé strany obsahují buďto jeden terminál nebo libovolně mnoho proměnných. Pokud jsou proměnné více než 2, potřebujeme pravidlo rozdělit. Pravidlo $X \to Y_1 \dots Y_m$ nahradíme pravidly $X \to Y_1 Z_1$, $Z_1 \to Y_2 Z_2$, …, $Z_{m-2} \to Y_{m-2} Z_{m-1}$, $Z_{m-1} \to Y_{m-1} Y_m$, kde $Z_1, \dots, Z_{m-1}$ jsou nové proměnné.

4. Nulová pravidla, tedy pravidla s prázdnou pravou stranou. Nejprve najdeme množinu všech *nulovatelných proměnných* – to jsou proměnné X, pro něž $X \overset{*}{\Longrightarrow} \varepsilon$. Určitě mezi nulovatelné patří levé strany pravidel typu $X \to \varepsilon$. Pak opakovaně hledáme pravidla, jejichž pravá strana obsahuje pouze nulovatelné proměnné, a levé strany také prohlašujeme za nulovatelné.

Poté projdeme všechna pravidla. Kdykoliv pravá strana obsahuje nulovatelnou proměnnou, vyrobíme kopii pravidla s touto proměnnou vynechanou. (Pokud máme pravidlo $X \to AB$ a A i B jsou nulovatelné, vyrobíme postupně $X \to A$, $X \to B$ a $X \to \varepsilon$.) Nakonec smažeme všechna nulová pravidla.

5. Jednotková pravidla. To jsou pravidla typu $X \to Y$. Vytvoříme graf jednotkových pravidel: vrcholy jsou proměnné a pro každé jednotkové pravidlo $X \to Y$ vytvoříme orientovanou hranu z X do Y. Pro každou dvojici proměnných X, Y se podíváme, zda v grafu vede cesta z X do Y, čili zda je možné X použitím posloupnosti jednotkových pravidel přepsat na Y. Pokud tomu tak je, vytvoříme ke každému nejednotkovému pravidlu typu $Y \to \alpha$ jeho kopii $X \to \alpha$. Pak všechna jednotková pravidla smažeme.

Nakonec se ujistíme, že žádný krok převodu nevytváří problémy, kterých jsme se v předchozích krocích zbavili. □

Algoritmus CYK

Nyní ukážeme, jak pro dané slovo $\alpha \in T^*$ délky n efektivně rozhodnout, zda je generováno bezkontextovou gramatikou převedenou do ChNF. Tento algoritmus popsali v 60. létech nezávisle na sobě Sakai, Cocke, Young a Kasami, proto se mu (poněkud nepřesně) říká algoritmus CYK.

Použijeme dynamické programování. Pro každé podslovo $\alpha[i : j]$ spočítáme množinu $D[i, j]$ všech proměnných, které toto podslovo generují. Budeme postupovat indukcí podle délky podslova.

Pro jednoznaková podslova $\alpha[i]$ spočítáme $D[i, i+1]$ jako množinu všech proměnných X, pro které existuje pravidlo $X \to \alpha[i]$. (Pravidla $X \to YZ$ nelze použít, protože jak Y, tak Z se časem rozepíší na neprázdné řetězce terminálů.)

Každé delší podslovo $\alpha[i : j]$ zkusíme rozdělit všemi možnými způsoby na části $\alpha[i : k]$ a $\alpha[k : j]$. Kdykoliv existuje pravidlo $X \to YZ$ takové, že $Y \in D[i, k]$ a $Z \in D[j, k]$, přidáme X do $D[i, j]$.

Až sestrojíme množinu $D[0, n]$, podíváme se, zda v ní leží počáteční proměnná S. Podle toho rozhodneme, zda je slovo α generováno gramatikou.

Nyní algoritmus popíšeme detailně. Nezapomeneme zvlášť ošetřit případ $\alpha = \varepsilon$.

Algorithm CYK
Vstup: Gramatika $G = (V, T, P, S)$ v ChNF, slovo $\alpha \in T^*$
Výstup: ANO, pokud $\alpha \in L(G)$, jinak NE

1. $n \leftarrow |\alpha|$
2. Pokud $n = 0$:
3. Je-li $(S \to \varepsilon) \in P$, odpovíme ANO, jinak NE.
4. Pro $i = 0, \ldots, n-1$:
5. $D[i, i+1] \leftarrow \{X \mid (X \to \alpha[i]) \in P\}$
6. Pro $\ell = 2, \ldots, n$: ◁ *délka podslova*

```
7.    Pro i = 0, ..., n − ℓ:                          ◁ začátek podslova
8.        j ← i + ℓ                                   ◁ konec podslova
9.        D[i, j] ← ∅
10.       Pro k = j + 1, ..., i − 1:                  ◁ dělící bod
11.           Pro všechna pravidla (X → Y Z) ∈ P:
12.               Pokud Y ∈ D[i, k] a Z ∈ D[k, j]:
13.                   Přidáme X do D[i, j].
14. Je-li S ∈ D[0, n], odpovíme ANO, jinak NE.
```

Pokud považujeme velikost gramatiky za konstantu, všechny operace s množinou pravidel a množinami proměnných mají konstantní časovou složitost nezávisle na reprezentaci množin. Algoritmus pak má složitost $\mathcal{O}(n^3)$.

Dodejme ještě, že algoritmus lze snadno upravit, aby odpověď ANO doprovodil jedním z možných derivačních stromů. Stačí si pro každou proměnnou v $D[i,j]$ zapamatovat, jakým pravidlem s jakým k tam byla přidaná. Pak strom rekurzivně vytváříme od kořene S: pokaždé se podíváme, jaké pravidlo bylo v aktuálním vrcholu použito, podle jeho pravé strany vytvoříme děti vrcholu a rekurzivně se na ně zavoláme, přičemž rozdělení slova α určíme podle zapamatovaného k. Časová složitost se tím nezhorší.

Poznámka: Existují i efektivnější algoritmy. Valiantův algoritmus redukuje problém na násobení matic a dosahuje tak složitost $\mathcal{O}(n^c)$ pro $c \in (2,3)$. Earleyho algoritmus je sice v nejhorším případě také kubický, ale v typických případech (např. je-li gramatika jednoznačná) výrazně rychlejší.

Iterační lemma

I bezkontextové jazyky mají svou verzi iteračního (pumpovacího) lemmatu.

Lemma (iterační pro bezkontextové jazyky): Pro každý bezkontextový jazyk L existuje číslo n takové, že každé slovo $\omega \in L$ délky aspoň n lze rozložit na části $\omega = \alpha\beta\gamma\delta\lambda$ a pro každé $k \geq 0$ je $\alpha\beta^k\gamma\delta^k\lambda \in L$. Přitom $|\beta\gamma\delta| \leq n$ a $|\beta\delta| > 0$.

Důkaz: Gramatiku generující L nejprve převedeme do Chomského normální formy. Pak si všimneme, že je-li α dost dlouhé, musí být jeho derivační strom dost hluboký (strom je binární, takže hloubka roste s počtem listů aspoň logaritmicky). A je-li dost hluboký, musí existovat cesta z kořene do listu, na níž se nějaká proměnná vyskytuje vícekrát (stačí cesta o $|V|$ hranách, tím pádem potřebujeme $n \geq 2^{|V|}$).

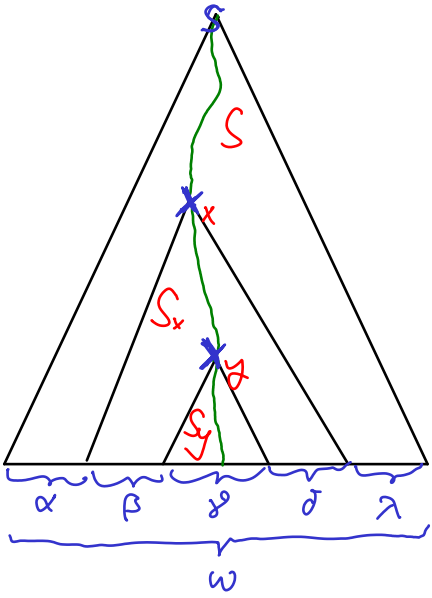
Označíme nyní x a y vrcholy, v nichž se opakovaná proměnná vyskytuje; x bude nad y. Nechť dále $\mathcal{S}$ je celý strom a $\mathcal{S}_x$ a $\mathcal{S}_y$ podstromy zakořeněné v x a y. Budeme procházet strom $\mathcal{S}$ v do hloubky a zaznamenávat navštívené listy. Ty dohromady dávají slovo ω, které rozdělíme takto:

- část α: listy před první návštěvou x,
- část β: listy mezi první návštěvou x a první návštěvou y,
- část γ: listy mezi první a poslední návštěvou y (listy podstromu $\mathcal{S}_y$),
- část δ: listy mezi poslední návštěvou y a poslední návštěvou x,
- část λ: listy po poslední návštěvě y.

Nyní si všimneme, že nahrazením podstromu $\mathcal{S}_x$ za podstrom $\mathcal{S}_y$ získáme derivační strom slova $\alpha\gamma\delta$. Naopak pokud nahradíme podstrom $\mathcal{S}_y$ další kopií celého $\mathcal{S}_x$, získáme derivační strom slova $\alpha\beta^2\gamma\delta^2\lambda$. Takto můžeme pokračovat libovolně-krát.

Pak ověříme, že se nemůže stát, že by části β a δ byly obě prázdné. Pokud cesta z x do y začíná pravou hranou, vede z x levá hrana do podstromu, jehož listy leží všechny v β, takže β není prázdné. Podobně pro levou hranu a δ.

Ještě potřebujeme splnit nerovnost $|\beta\gamma\delta| \leq n$. To zařídíme tak, že zvolíme nejdelší cestu mezi kořenem a listem a na ní nejnižší opakovaný výskyt proměnné. Tím pádem strom $\mathcal{S}_x$ bude mít hloubku nejvýše $|V|$, čili maximálně $2^{|V|}$ listů. □



Obrázek 2.4: Situace v důkazu iteračního lemmatu

Další vlastnosti

Bezkontextové iterační lemma můžeme použít k důkazu, že jazyk $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$ není bezkontextový (cvičení 5). Tento jazyk ovšem lze generovat složitější gramatikou (cvičení 2.1.3). Proto inkluze tříd jazyků $\mathcal{L}_2 \subset \mathcal{L}_0$ je ostrá.

Třída bezkontextových jazyků $\mathcal{L}_2$ je uzavřená na sjednocení, zřetězení, iteraci a otočení (cvičení 7), není uzavřená na průniky (cvičení 6) ani na doplňky (cvičení 8).

Některé algoritmické otázky jsou pro bezkontextové jazyky jednoduché, jiné zase těžké:

- *příslušnost slova do jazyka* je možné testovat v polynomiálním čase algoritmem CYK,
- *neprázdnost jazyka generovaného gramatikou* je polynomiální (cvičení 10),
- *generování prázdného slova* lze triviálně rozpoznat z Chomského normální formy,
- *generování všech slov z T^** je algoritmicky nerozhodnutelné (bez důkazu),
- *ekvivalence gramatik* (rovnost generovaných jazyků) je také nerozhodnutelná (mohli bychom převést předchozí otázku na ekvivalenci s gramatikou generující celé T^*),
- *jednoznačnost gramatiky* je také nerozhodnutelná (bez důkazu).

Bezkontextové jazyky se také dají rozpoznávat pomocí *nedeterministických zásobníkových automatů.* Těm se v tomto textu budeme věnovat pouze okrajově (cvičení 3.2.7), ale prozradíme, že rozpoznávají právě bezkontextové jazyky a že na rozdíl od konečných automatů zde nedeterminismus zvyšuje výpočetní sílu.

Cvičení

1. Doplňte do gramatiky pro výrazy operátor `^` (umocňování, vyhodnocuje se zprava doleva), unární `-` a unární postfixový `!` (faktoriál). Snažte se, aby gramatika byla nadále jednoznačná.

2. Řetězec levých a pravých závorek nazveme *závorkováním,* pokud se závorky dají rozdělit do nekřížících se párů tak, že v každém páru je nalevo `(` a napravo `)`. Sestrojte bezkontextovou gramatiku, která generuje všechna závorkování.

3. Sestrojte derivační strom řetězce `(()())` pro gramatiku z cvičení 2.

4. Gramatiku z cvičení 2 převeďte do Chomského normální formy. Jak nyní vypadá derivační strom řetězce `(()())`? Odsimulujte algoritmus CYK pro tento řetězec.

5. Dokažte, že jazyk $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$ není bezkontextový. Může se hodit bezkontextové iterační lemma.

6. Jazyk $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$ je průnikem jazyků $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m \mid n, m \in \mathbb{N}\}$ a $\{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^m \mid n, m \in \mathbb{N}\}$. První z jazyků podle předchozího cvičení není bezkontextový. Ukažte, že zbylé dva jsou, takže třída bezkontextových jazyků není uzavřená na průnik.

7. Ukažte, že třída bezkontextových jazyků je uzavřená na sjednocení, zřetězení, iteraci a otočení.

8. Ukažte, že když je třída jazyků uzavřená na sjednocení, a nikoliv na průnik, nemůže být uzavřená na doplněk.

9. U jazyků z cvičení v oddílu 2.1 rozhodněte, zda jsou bezkontextové.

10. Proměnná v gramatice je *dosažitelná,* pokud figuruje v aspoň jedné derivaci z S. Proměnná je *produktivní,* pokud z ní lze odvodit aspoň jeden řetězec terminálů. Proměnná je *potřebná,* pokud se vyskytuje v aspoň jedné derivaci řetězce terminálů z S. Dokažte, že všechny proměnné jsou potřebné, právě když jsou všechny proměnné produktivní a současně potřebné. Navrhněte polynomiální algoritmus, který gramatiku upraví tak, aby všechny proměnné byly potřebné. Pomocí toho zjistěte, zda gramatika generuje neprázdný jazyk.

11. *Velikost gramatiky* zavedeme jako součet délek všech levých a pravých stran pravidel. Jak se velikost změní převodem do ChNF? Jak na velikosti gramatiky závisí časová složitost algoritmu CYK?

12.* *Lineární iterační lemma.* Bezkontextové iterační lemma pochopitelně platí i pro lineární jazyky, ale nerovnost $|\beta\gamma\delta| \leq n$ můžeme nahradit $|\alpha\beta\delta\lambda| \leq n$. Dokažte. Stačí lehce upravit stávající důkaz, ale je potřeba domyslet analogii ChNF pro lineární gramatiky.

13. Dokažte, že jazyk $\{\mathtt{a}^n\mathtt{b}^n\mathtt{a}^m\mathtt{b}^m\}$ je bezkontextový, ale není lineární. Může se hodit tvrzení z předchozího cvičení.

14.* V cvičení 1.4.11 jsme dokázali, že třída regulárních jazyků je uzavřená na substituci. Ukažte, že to platí i pro třídu bezkontextových jazyků. To dává jiný důkaz uzavřenosti CFL na sjednocení, zřetězení a iteraci.

3 Rozhodnutelné jazyky

Počátkem 20. století se matematici zabývali otázkami „mechanické“ řešitelnosti různých problémů – kořeny celočíselných polynomiálních rovnic, dokazatelnost tvrzení v logice apod. Zásadním problémem se ale ukázala sama definice mechanického výpočtu. Podali ji až ve 30. letech Alonzo Church (λ-kalkulus), Stephen Kleene (kalkulus rekurzivních funkcí) a Alan Turing (stroj s páskou). Právě Turingovou definicí výpočtu se nyní budeme zabývat. Ostatní modely výpočtu jsou s Turingovým strojem ekvivalentní, alespoň co se týče toho, na jaké otázky dovedou odpovídat.

3.1 Turingovy stroje

Turingův stroj je motivovaný představou matematika, který má k dispozici tabuli a svou mysl. Zatímco tabule je potenciálně nekonečně velká, do mysli se vejde pouze konečné množství informací.

Tabuli budeme modelovat oboustranně nekonečnou *páskou* rozdělenou na *políčka.* Na každém políčku se vyskytuje jeden znak z konečné abecedy. Po pásce se pohybuje *hlava* stroje, která se vždy dívá na jedno políčko a umí znak z tohoto políčka přečíst, přepsat na jiný a posunout se o jedno políčko doleva nebo doprava.

Matematikovu mysl si budeme představovat jako *řídicí jednotku* stroje, která se v každém okamžiku nachází v jednom z konečně mnoha *stavů* (stejně jako konečný automat). V každém kroku výpočtu se jednotka podle svého stavu a znaku, který přečte hlava, rozhodne, jakou instrukci stroje provést. Instrukce určí, jaký znak na aktuální políčko pásky zapsat, do jakého stavu řídicí jednotky se přepnout a zda se hlava posune doleva nebo doprava, případně zůstane na místě. Rozhodování řídicí jednotky popíšeme *přechodovou funkcí.*

Na počátku výpočtu je na pásce napsán vstup, zbývající políčka pásky jsou vyplněna speciálním symbolem $\sqcup$ (*mezera*). Hlava se dívá na první znak vstupu. Řídicí jednotka se nachází v počátečním stavu q_0.

Výpočet končí tím, že se řídicí jednotka přepne do jednoho z *koncových stavů.* Ty jsou dva (q_+ a q_-). Jeden odpovídá přijetí vstupu, druhý jeho odmítnutí.

Nyní stroj a jeho výpočet nadefinujeme podobně, jako jsme to udělali u konečných automatů.

Definice: *Turingův stroj* (Turing Machine, zkráceně TM) se skládá z následujících částí:

- Q je konečná neprázdná množina *stavů* stroje,

- $q_0 \in Q$ je *počáteční stav,*
- $q_+, q_- \in Q$ jsou *koncové stavy: přijímací a odmítací,*
- Σ je konečná neprázdná *vstupní abeceda* (v ní je zadán vstup),
- $\Gamma \supset \Sigma$ je konečná *pracovní abeceda* znaků používaných na pásce,
- $\sqcup \in \Gamma \setminus \Sigma$ je znak pro *mezeru,*
- $\delta : (Q \setminus \{q_+, q_-\}) \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \bullet, \rightarrow\}$ je *přechodová funkce.*

Definice: *Konfigurace* Turingova stroje je uspořádaná trojice (s, λ, π), kde $s \in Q$ je stav stroje, $\lambda \in \Gamma^*$ obsah pásky nalevo od hlavy a $\pi \in \Gamma^*$ obsah pásky od políčka s hlavou dále doprava. Mezery zleva a zprava budeme ořezávat, tedy λ nezačíná mezerou a π nekončí mezerou.

Definice: *Krok stroje* v konfiguraci (s, λ, π), kde $s \neq q_+, q_-$, definujeme takto:

- Nejprve na začátek λ a na konec π připíšeme mezeru. Tím zajistíme, že pod hlavou i před hlavou máme explicitní znak. Platí tedy $\lambda = \lambda_1 y$ a $\pi = x\pi_1$ pro nějaké znaky x, y a slova λ_1, π_1.
- Podle $\delta(s, x)$ zjistíme, jakou instrukci (s', x', pohyb) má stroj vykonat:
  - Pokud $\mathit{pohyb} = \bullet$, položíme $\lambda' = \lambda$, $\pi' = x'\pi_1$.
  - Pokud $\mathit{pohyb} = \rightarrow$, položíme $\lambda' = \lambda x'$, $\pi' = \pi_1$.
  - Pokud $\mathit{pohyb} = \leftarrow$, položíme $\lambda' = \lambda_1$, $\pi' = yx'\pi_1$.
- Odstraníme z λ všechny počáteční mezery a z π všechny koncové mezery.
- Přejdeme do konfigurace (s', λ', π').

Definice: *Výpočet stroje* pro vstup $\alpha \in \Sigma^*$ je konečná nebo nekonečná posloupnost konfigurací $K_0, K_1, K_2, \ldots$, kde $K_0 = (q_0, \varepsilon, \alpha)$ a pro všechna i platí, že K_{i+1} vznikne jedním krokem výpočtu z K_i. Je-li výpočet konečný, poslední konfigurace obsahuje jeden z koncových stavů stroje. V žádné jiné konfiguraci se koncový stav nevyskytuje.

Výpočet je jednoznačně určen vstupem stroje. Buďto je konečný a stroj *se zastaví,* nebo je nekonečný a stroj *diverguje.* Možnost divergence stroje způsobuje, že jazyk rozpoznávaný strojem můžeme definovat dvěma způsoby:

Definice: Stroj *přijme* slovo $\alpha \in \Sigma^*$, pokud se jeho výpočet se vstupem α zastaví ve stavu q_+. Slovo tedy *odmítne,* pokud se zastaví v q_- nebo diverguje. Množině slov přijímaných strojem M říkáme *jazyk přijímaný strojem* M a značíme ho $L(M)$.

Definice: Jazyk L je *částečně rozhodnutelný* neboli *rekurzivně spočetný,* pokud existuje Turingův stroj přijímající jazyk L. Třídu všech takových jazyků značíme RE.[1]

Definice: Stroj *rozhoduje* jazyk L, pokud se pro každý vstup $\alpha \in \Sigma^*$ zastaví a přijímá jazyk L. (Pro každé slovo α tedy výpočet skončí ve stavu q_+, pokud $\alpha \in L$, a jinak skončí v q_-.)

Definice: Jazyk L je *rozhodnutelný* neboli *rekurzivní,* pokud existuje Turingův stroj rozhodující jazyk L. Třídu všech takových jazyků značíme R.

Poznámka: Zjevně je $\mathsf{R} \subseteq \mathsf{RE}$. Časem dokážeme, že inkluze je ostrá, a také prozkoumáme vztahy s třídami jazyků z předchozích kapitol.

Turingův stroj můžeme také použit k výpočtu funkcí:

Definice: Funkce $f : \Sigma^* \to \Sigma^*$ je *vyčíslitelná* neboli *rekurzivní,* pokud existuje Turingův stroj, který se pro každé $\alpha \in \Sigma^*$ zastaví a vydá výstup $f(\alpha)$. *Výstupem* stroje myslíme obsah pásky $\lambda\pi$ z poslední konfigurace výpočtu.

Definice: Funkce $f : \Sigma^* \to \Sigma^* \cup \{\uparrow\}$ je *částečně vyčíslitelná* neboli *částečně rekurzivní,* pokud existuje stroj, který se pro vstup $\alpha \in \Sigma^*$ zastaví právě tehdy, když $f(\alpha) \neq \uparrow$, a pokud se zastaví, je jeho výstupem $f(\alpha)$.

Pozorování: Jazyk L je rozhodnutelný právě tehdy, když je vyčíslitelná jeho charakteristická funkce. Jazyk L je částečně rozhodnutelný právě tehdy, když je částečně vyčíslitelná funkce $f(\alpha) = 1$ pro $\alpha \in L$ a $f(\alpha) = \uparrow$ pro $\alpha \notin L$.

Příklad: Sestrojíme Turingův stroj rozpoznávající jazyk $\{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N}\}$. Stroj bude opakovaně odebírat znak `0` ze začátku slova a `1` z konce slova. Pokud se tím podaří slovo vyprázdnit, stroj přijme. Pokud odebírání selže (na začátku nenajde `0` nebo na konci `1`), stroj odmítne.

Stroj bude mít vstupní abecedu $\Sigma = \{\mathtt{0}, \mathtt{1}\}$, pracovní abecedu $\Gamma = \{\mathtt{0}, \mathtt{1}, \sqcup\}$, množinu stavů $Q = \{q_0, q_+, q_-, r, j, \ell\}$ a přechodovou funkci definovanou tabulkou na obrázku 3.1.

Na rozdíl od konečných automatů mohou být různé TM pro tentýž vstup rozdílně rychlé (některé se ani nemusí zastavit). Hodí se proto umět efektivitu stroje měřit:

Definice: *Čas výpočtu* definujeme jako počet konfigurací, kterými výpočet stroje projde. *Prostor výpočtu* je počet políček pásky, která během výpočtu navštívila hlava stroje.

[1] Terminologie je zde trochu zavádějící. Nejde o rekurzi v dnešním obvyklém smyslu, nýbrž o Kleeneho kalkulus rekurzivních funkcí. Rekurzivně spočetné jsou anglicky *recursively enumerable,* což znamená spíš rekurzivně vyjmenovatelné. Tento vztah dále zkoumáme v cvičení 3.4.9.

stav/znak	0	1	␣
q_0	$(r, ␣, \rightarrow)$	q_-	q_+
r	$(r, 0, \rightarrow)$	$(r, 1, \rightarrow)$	$(j, ␣, \leftarrow)$
j	q_-	$(\ell, ␣, \leftarrow)$	q_-
ℓ	$(\ell, 0, \leftarrow)$	$(\ell, 1, \leftarrow)$	$(q_0, ␣, \rightarrow)$

Obrázek 3.1: Turingův stroj rozpoznávající jazyk $\{0^n1^n \mid n \in \mathbb{N}\}$. Tam, kde přecházíme do stavu q_+ nebo q_-, nezáleží na novém znaku ani pohybu hlavy.

Definice: *Časová složitost* stroje je funkce, která každé délce vstupu n přiřadí maximální čas výpočtu pro vstupy z Σ^n. Podobně prostorová složitost přiřadí délce vstupu maximální prostor výpočtu. Pokud se některý výpočet nezastaví, časová složitost bude nekonečná a prostorová možná také.

Příklad (proměnné ve stavu): Často se hodí, aby si stroj pamatoval několik „proměnných". Pokud mají omezený rozsah, můžeme je všechny zakódovat do stavu stroje: stav bude uspořádaná k-tice, jejíž složky budou odpovídat hodnotám jednotlivých proměnných.

Příklad (vícestopá paska): Podobně můžeme na jedno políčko pásky uložit několik různých druhů informací, když jako znaky pracovní abecedy použijeme uspořádané ℓ-tice. Můžeme si to představit jako *ℓ-stopou pásku,* jejíž stopy používáme nezávisle. Všechny stopy ovšem sdílí polohu hlavy. Pozor na to, že na začátku výpočtu musíme překódovat vstupní abecedu do ℓ-tic, a na konci zase ℓ-tice výstupu dekódovat.

Cvičení

Sestrojte Turingovy stroje řešící následující úlohy. Pokaždé stanovte jejich časovou a prostorovou složitost.

1. Rozhodnout jazyk všech slov nad abecedou $\{0, 1\}$, v nichž je stejně nul jako jedniček.
2. K zadanému řetězci α spočítat jeho otočení α^{R}.
3. Rozhodnout jazyk všech závorkování z cvičení 2.3.2.
4. Pro slovo 0^n spočítat zápis čísla n ve dvojkové soustavě.
5. Pro číslo n zapsané ve dvojkové soustavě vytvořit slovo 0^n.
6. Rozhodnout jazyk $a^n b^n c^n$.

3.2 Varianty Turingových strojů

Výhodou definice Turingova stroje je, že se snadno upravuje, čímž vznikají další druhy strojů.

Konečné automaty
Uvažujme stroj, jehož instrukce používají pouze pohyb doprava. Navíc přečte-li stroj mezeru, přejde vždy do stavu q_+ nebo q_-. Takové stroje jsou zjevně ekvivalentní s konečnými automaty, takže rozhodují a také rozpoznávají právě regulární jazyky.

Obousměrné automaty
Obousměrný konečný automat je Turingův stroj, který má zakázáno měnit obsah pásky – instrukce tedy musí vždy zapsat ten symbol, který byl z pásky přečten. Vstup je navíc stroji dodán ohraničený: pro vstup α je počátečním obsahem pásky slovo $\texttt{<}\alpha\texttt{>}$, kde < a > jsou znaky pracovní abecedy zvané *zarážky*. Hlava začíná na prvním znaku slova α. Pokud stroj narazí na zarážku <, musí vždy vykonat pohyb doprava; pokud narazí na >, musí jít doleva.

Obousměrné automaty se tedy mohou po vstupu libovolně pohybovat, ale nesmí ho měnit. Na rozdíl od obyčejných automatů mohou divergovat. Překvapivě opět rozpoznávají jenom regulární jazyky (toto tvrzení ponecháme bez důkazu).

Vícepáskové stroje
Zajímavější je vybavit Turingův stroj více páskami. Mějme tedy k oboustranně nekonečných pásek. Každá má svou vlastní hlavu, která se pohybuje nezávisle na ostatních hlavách. Přechodová funkce se rozhoduje podle stavu a symbolů přečtených všemi k hlavami. Instrukce stroje zapíše znaky na všech k pásek a každé hlavě řekne, kam se má pohnout. Přechodová funkce tedy vede z $Q \times \Gamma^k$ do $Q \times \Gamma^k \times \{\leftarrow, \bullet, \rightarrow\}^k$.

Konfigurace stroje se skládá ze stavu řídicí jednotky a obsahů všech pásek; každou pásku opět rozdělíme na část nalevo a napravo od příslušné hlavy. Krok stroje a výpočet se rozšíří zjevným způsobem.

Při spuštění stroje je vstup napsán na první pásce. Pokud stroj vrací výstup, napíše ho opět na první pásku.

Někdy se také určuje speciální *vstupní a výstupní páska.* Vstupní pásku je povoleno pouze číst (stroj tedy musí zapsat ten znak, který právě přečetl), na výstupní pásku je povoleno pouze zapisovat (přechodová funkce musí pro všechny možné znaky z této pásky vracet stejnou instrukci). Ostatním páskám se říká *pracovní* a do využitého prostoru počítáme jenom je.

Věta: Vícepáskový stroj je možné převést na jednopáskový, který přijímá/rozpoznává tentýž jazyk a vyčísluje tutéž funkci.

Náčrt důkazu: Budeme k-páskový stroj simulovat jednopáskovým strojem, jehož pásku rozdělíme na $2k$ stop. Stopy budou tvořit k párů. Každý pár bude odpovídat jedné pásce simulovaného stroje a bude v něm *datová stopa* s obsahem pásky a *řídicí stopa*, v níž bude vyznačena pozice hlavy simulovaného stroje. Stav simulovaného stroje si budeme pamatovat ve stavu nového stroje.

Na začátku výpočtu překódujeme vstup do $2k$-stopé abecedy, vyznačíme počáteční polohy všech hlav a přejdeme do počátečního stavu simulovaného stroje.

Jeden krok stroje odsimulujeme takto: Nejprve projdeme celou pásku zleva doprava a kdykoliv v nějaké řídicí stopě najdeme značku polohy hlavy, zapamatujeme si ve stavu znak z příslušné datové stopy. Po přečtení všech k znaků vyhodnotíme přechodovou funkci (bude zakódovaná do naší přechodové funkce), zjistíme, jakou instrukci máme vykonat, a zapamatujeme si to ve stavu. Pak znovu projdeme celou pásku a kdykoliv narazíme v řídicí stopě na značku hlavy, zapíšeme do odpovídající datové stopy nový znak a posuneme značku hlavy správným směrem. Nakonec se přepneme do nového stavu simulovaného stroje.

Na konci výpočtu dekódujeme obsah pásky z $2k$-stopé abecedy do původní.

Zbývá dořešit jeden problém: jak při procházení celé pásky poznat, kde začíná a končí. Mohli bychom si v další stopě udržovat zarážky na začátku/konci využitého úseku pásky a podle potřeby je posouvat. Ale jednodušší je pamatovat si ve stavu stroje, kolik simulovaných hlav zrovna leží nalevo od aktuální pozice na pásce. □

Nedeterministické stroje

Podobně jako jsme zavedli nedeterministický konečný automat, můžeme definovat nedeterministickou verzi Turingova stroje. Jeho přechodová funkce bude místo jedné instrukce přiřazovat množinu instrukcí. Jeden krok výpočtu pro jednu konfiguraci určí množinu možných následujících konfigurací. Pro jeden vstup pak může existovat mnoho různých výpočtů, dokonce některé konečné a jiné nekonečné.

Stroj přijme slovo, pokud se alespoň jeden z možných výpočtů zastaví v přijímacím stavu. Později dokážeme, že nedeterministické stroje přijímají tytéž jazyky jako deterministické stroje.

Výpočty nedeterministického stroje můžeme popsat stromem konfigurací: v kořeni je počáteční konfigurace, potomci každé konfigurace jsou ty, do kterých se dá dostat jedním krokem výpočtu. Listy stromu odpovídají zastavení výpočtu v koncovém stavu, ale strom může mít i nekonečné větve.

Randomizované stroje

Stroj můžeme vybavit generátorem náhodných bitů. Pořídíme mu *náhodnou pásku,* která bude na začátku výpočtu obsahovat nekonečnou posloupnost nezávislých náhodných bitů. Po této pásce bude povoleno pohybovat se pouze doprava.

Podobné jako u nedeterministických strojů není ani zde výpočet jednoznačně určen: k jedné konfiguraci mohou existovat dvě následující. Každému výpočtu pak můžeme přiřadit pravděpodobnost toho, že bude proveden (to je 2^{-t}, kde t je počet přečtených náhodných bitů). Sečtením všech přijímajících výpočtů pak můžeme stanovit pravděpodobnost přijetí slova.

Stroje s orákulem

Schopnosti stroje můžeme rozšířit o vyhodnocování libovolné funkce. Té se obvykle říká *orákulum.* S orákulem se komunikuje tak, že vstup funkce zapíšeme na speciální *orákulovou pásku,* přejdeme do speciálního stavu a na začátku dalšího kroku výpočtu bude obsah orákulové pásky nahrazen odpovědí orákula. Do času výpočtu se dotaz na orákulum počítá jako jeden krok.

Interaktivní stroje

Někdy se hodí vybavit algoritmy schopností interagovat s okolím, tedy v průběhu výpočtu vypisovat výstup a získávat další vstup. I to se dá do Turingova stroje dodělat, a to podobně jako orákulum.

Výstup uložíme na výstupní pásku a pak přejdeme do speciálního stavu, čímž je vypsán. Když chceme přečíst vstup, přejdeme do jiného speciálního stavu a na vstupní pásce se objeví další slovo ze vstupu.

Cvičení

1. Pro všechna cvičení z oddílu 3.1 uvažte, zda pomocí vícepáskového stroje není možné úlohu vyřešit s lepší časovou a/nebo prostorovou složitostí.

2.* Rozhodněte jazyk závorkování (cvičení 2.3.2) v čase $\mathcal{O}(n)$ a prostoru $\mathcal{O}(\log n)$.

3. Uvažujme TM, který umí měnit pásku jenom uděláním „kaňky“. A jakmile je na políčku kaňka, už ho nelze přepsat na jiný symbol. Dokažte, že tyto stroje rozpoznávají/rozhodují tytéž jazyky jako standardní TM.

4. Dokažte, že TM s jednostranně nekonečnou páskou dokáže rozpoznávat/rozhodovat tytéž jazyky jako standardní TM. Pokud stroj chce udělat pohyb doleva na začátku pásky, stroj automaticky přejde do stavu q_- a zastaví se.

5.* Dokažte, že každý TM je možné upravit na ekvivalentní (rozhodující tentýž jazyk, vydávající tentýž výstup), který má pouze 2 stavy kromě q_+ a q_-.

6.* Dokažte, že každý TM se vstupní abecedou $\Sigma = \{0, 1\}$ lze upravit na ekvivalentní, jehož pracovní abeceda je $\Gamma = \{0, 1, \sqcup\}$.

7.** *Zásobníkový automat* je TM s jednou vstupní páskou (na níž nelze zapisovat ani se pohybovat doleva) a jednou pracovní páskou používanou jako zásobník (při pohybu doleva musíme aktuální znak přepsat na mezeru). Dokažte, že jazyk přijímaný každým zásobníkovým automatem je bezkontextový. Aby platila i druhá implikace, musíme nicméně uvažovat nedeterministické zásobníkové automaty.

8. *Dvojzásobníkový automat* je definován podobně jako v minulém cvičení, ale místo jednoho zásobníku má dva. Dokažte, že tyto automaty přijímají/rozpoznávají stejné jazyky jako standardní TM.

9. Jak se při převodu vícepáskového stroje na jednopáskový změní časová a prostorová složitost?

10.** Navrhněte převod vícepáskového stroje na dvojpáskový tak, aby zpomaloval pouze $\mathcal{O}(\log n)$-krát.

11.* Navrhněte převod nedeterministického vícepáskového stroje na dvojpáskový tak, aby zpomaloval pouze konstanta-krát.

3.3 Vztahy s ostatními modely

Ukážeme, že Turingův stroj je ekvivalentní s některými dalšími modely výpočtu. Takzvaná *Churchova-Turingova teze* dokonce říká, že všechny realistické výpočetní modely jsou navzájem ekvivalentní. Jinými slovy že je jedno, jaký model použijeme k definici algoritmu, protože pokaždé vyjde (až na vhodný isomorfismus) totéž. Churchovu tezi nicméně není možné dokázat, protože neumíme definovat, co znamená realistický výpočetní model.

Gramatiky

Věta: Třída $\mathcal{L}_0$ jazyků generovaných gramatikami je rovná třídě RE rekurzivně spočetných jazyků.

Důkaz rozdělíme do dvou lemmat.

Lemma: Každý částečně rozhodnutelný jazyk je generovaný nějakou gramatikou.

Důkaz: Musíme se vypořádat s tím, že „výpočet“ gramatiky probíhá v opačném směru než výpočet Turingova stroje. Gramatika vyjde z počáteční proměnné a postupně přepisuje,

až vygeneruje slovo jazyka. Stroj naopak začne nějakým slovem,[2] to postupně upravuje a nakonec odpoví, zda slovo patří do jazyka.

Tento rozpor vyřešíme tak, že nejprve gramatikou vygenerujeme dvě kopie slova. Jedna (té budeme říkat *originál*) bude složena z terminálů, druhá (řečená *pracovní*) bude obsahovat aspoň jednu proměnnou. Na pracovní kopii budeme simulovat výpočet stroje. Pokud výpočet skončí v přijímacím stavu, smažeme celou pracovní kopii, čímž zbude jen originál slova z terminálů a přepisování se zastaví.

Jelikož gramatikou se snáz než $\alpha\alpha$ generuje $\alpha\alpha^{\mathrm{R}}$, dovolíme si drobný trik: stroj před převáděním na gramatiku upravíme, aby rozpoznával slova zapsaná pozpátku. Na to stačí přesunout hlavu na konec slova a pak spustit původní výpočet s prohozenými směry doleva a doprava.[3]

Nyní konkrétněji. Terminály gramatiky budou znaky pracovní abecedy stroje Γ. Proměnné gramatiky budou tvořeny:

- počáteční proměnnou I,
- kopií Q' množiny stavů stroje Q (pro každý stav $t \in Q$ vytvoříme jeho kopii $T \in Q'$ různou od ostatních proměnných a terminálů),
- zarážkami `<` a `>`,
- pomocnou proměnnou J.

V průběhu přepisování bude mít slovo tvar $\alpha\texttt{<}\beta\texttt{>}$, kde $\alpha \in \Sigma^*$ je originál slova a β aktuální obsah pásky stroje tvořený terminály z Γ, do nějž je před pozici hlavy vložen stav stroje v podobě proměnné z Q'.

Pravidla gramatiky rozdělíme do několika skupin:

- Inicializace: má za úkol vytvořit z I řetězec $\alpha\texttt{<}Q_0\beta\texttt{>}$, kde β je α^{R} přepsané do pracovních proměnných.
  - $I \to J\texttt{>}$
  - $J \to xJx$ (pro všechna $x \in \Sigma$)
  - $J \to \texttt{<}Q_0$
- Výpočet stroje: kdykoliv $\delta(x, s) = (y, s', \mathit{pohyb})$, přidáme pravidlo:

[2] Inu, i zde platí, že na počátku bylo slovo.

[3] Tím jsme mimochodem dokázali, že třída rozhodnutelných i částečně rozhodnutelných jazyků jsou obě uzavřené na otočení.

  - $Sx \to S'x'$, pokud $pohyb = \bullet$,
  - $Sx \to x'S'$, pokud $pohyb = \rightarrow$,
  - $ySx \to S'yx'$ pro každé $y \in \Gamma$, pokud $pohyb = \leftarrow$.
- Expanze pásky o další mezeru na levém a pravém okraji:
  - $< \to <_\sqcup$
  - $> \to \sqcup>$
- Smazání pracovní části, pokud se stroj dostane do stavu q_+ (v tomto stavu už výpočet nepokračuje, neboť přechodová funkce tam není definovaná):
  - $xQ_+ \to Q_+$ pro každé $x \in \Gamma$
  - $Q_+x \to Q_+$ pro každé $x \in \Gamma$
  - $<Q_+> \to \varepsilon$

Snadno ověříme, že vygenerovat jdou právě ta slova, která stroj přijímá. □

Lemma: Jazyk generovaný jakoukoliv gramatikou je částečně rozhodnutelný.

Důkaz: Mějme libovolnou gramatiku. Vytvoříme stroj, který bude vyjmenovávat všechna slova (z proměnných i terminálů) získatelná postupným přepisováním počáteční proměnné gramatiky: nejprve počáteční proměnnou, pak slova získatelná jedním přepsáním, pak dvěma, a tak dále. Pokud slovo na vstupu stroje leží v jazyce generovaném gramatikou, stroj ho najde mezi vyjmenovanými slovy a přijme. Neleží-li v jazyce, stroj bude buď vyjmenovávat další a další slova, nebo (je-li generovaný jazyk konečný), slova mu dojdou a vstup zamítne.

Stroj sestrojíme jako vícepáskový. Vstupní pásku nebude měnit. Na první pracovní pásce bude vyjmenovávat slova a oddělovat je znakem `#`. Na začátku tam bude jen slovo S. Pokaždé vezmeme další slovo a porovnáme ho se vstupem; v případě shody se zastavíme v přijímacím stavu. Jinak ve slovu zkusíme najít všechny výskyty levých stran pravidel gramatiky (tvar pravidel si budeme pamatovat v přechodové funkci stroje). Kdykoliv nějaký najdeme, zapíšeme na druhou pracovní pásku kopii aktuálního slova s levou stranou pravidla vyměněnou za pravou. Nakonec všechna slova z druhé pracovní pásky přepíšeme na konec první pásky a pokračujeme dalším slovem. □

Nedeterministické stroje

Myšlenku postupného vyjmenovávání možných výpočtů bychom mohli použít i k důkazu, že nedeterministické Turingovy stroje přijímají tytéž jazyky jako deterministické stroje. Vlastně bychom prohledávali do šířky strom možných výpočtů. Stejný výsledek ale můžeme získat jednodušeji.

Věta: Nedeterministické Turingovy stroje přijímají částečně rozhodnutelné jazyky.

Důkaz: Všimneme si, že převod Turingova stroje na gramatiku funguje i pro nedeterministické stroje: stačí vytvořit pravidla gramatiky pro všechny instrukce $(s', x', \mathit{směr}) \in \delta(s, x)$. Proto označíme-li NRE třídu jazyků přijímaných nedeterministickými stroji, platí $\mathsf{RE} \subseteq \mathsf{NRE} \subseteq \mathcal{L}_0 = \mathsf{RE}$. □

Random Access Machine

I výpočetní model RAM (Random Access Machine), ve kterém obvykle studujeme algoritmy, je ekvivalentní s TM (Turingovým strojem). Budeme používat definici RAMu z Průvodce labyrintem algoritmů (kapitola o časové a prostorové složitosti). Musíme se nicméně vyrovnat s tím, že TM zpracovávají řetězce, zatímco RAM čísla a jejich posloupnosti. Ekvivalenci tedy dokážeme jen pro vstupy ve tvaru řetězce bitů. Ty může TM dostat přímo a RAMu bity uložíme do posloupnosti paměťových buněk a ukončíme číslem 2.[4]

RAM můžeme použít k přijímání jazyka (výpočet pro daný vstup se zastaví), k rozhodování jazyka (výpočet se vždy zastaví a ve smluvené buňce paměti vydá nulu nebo jedničku) i k vyčíslování funkcí (ze smluveného místa v paměti přečteme výsledek funkce). Dostaneme stejné třídy jazyků a funkcí jako pro TM: dokážeme, že výpočet TM lze simulovat na RAMu a opačně.

Simulace TM na RAMu je přímočará: do paměťových buněk RAMu uložíme jednotlivá políčka pásky. V dalších buňkách si budeme pamatovat index prvního a posledního použitého políčka a pozici hlavy. Stav stroje budeme reprezentovat pozicí v programu RAMu.

Zajímavější je simulace v opačném směru. Budeme vytvářet vícepáskový TM.

- *Reprezentace čísel:* RAM počítá s celými čísly, na TM je budeme kódovat ve dvojkové soustavě se samostatným znakem pro znaménko.
- *Aritmetické operace:* pro každou aritmetickou a bitovou operaci RAMu sestrojíme část TM, která ji bude počítat. Pro vstupy a výstup operace použijeme samostatné pracovní pásky.

[4] Na reprezentaci vstupu zde nezáleží – jak TM, tak RAM jsou dost silné na to, aby převáděly mezi libovolnými „rozumnými“ reprezentacemi. Nějakou jsme nicméně museli zvolit.

- *Reprezentace paměti:* na další pracovní pásce si budeme pamatovat použitou část paměti RAMu v podobě posloupnosti dvojkových čísel oddělených symbolem #. Dalším znakem vyznačíme nultou buňku a ve stavu stroje si budeme pamatovat, zda jsme zrovna před ní nebo za ní, takže se budeme umět kdykoliv k nulté buňce vrátit.

- *Čtení z paměti:* adresu požadované buňky dostaneme na speciální pásce. Nejprve tuto buňku najdeme: pokud je adresa kladná, vydáme se po paměťové pásce doprava a za každý # odečteme od adresy jedničku. Až se adresa vynuluje, zkopírujeme obsah buňky na další pracovní pásku. Je-li adresa záporná, postupujeme doleva a jedničky přičítáme. Pokud během hledání buňky opustíme inicializovanou část paměti, budeme podle potřeby přidávat prázdné buňky.

- *Zápis do paměti* dostane adresu a hodnotu na dvou pracovních páskách. Buňku najdeme podobně jako při čtení a pak do ní začneme kopírovat hodnotu. Pokud se hodnota do buňky nevejde, rozšíříme buňku o další znaky, což vyžaduje posunout všechny následující znaky.

- *Aritmetické instrukce:* nejprve přečteme operandy (to jsou konstanty, přímo adresované buňky nebo nepřímo adresované buňky), pak zavoláme podprogram pro výpočet aritmetické operace, a nakonec výsledek zapíšeme do paměti (přímo nebo nepřímo adresované).

- *Chod programu a řídicí instrukce:* posloupnost instrukcí programu bude zakódovaná do přechodové funkce stroje, pozici v programu si budeme pamatovat ve stavu stroje. Nepodmíněný skok pouze změní pozici v programu. Podmíněný skok předtím vyhodnotí podmínku podobně jako aritmetickou operaci. Instrukce zastavení programu způsobí vydání výsledku a zastavení stroje.

RAM je tedy stejně silný jako TM.

Cvičení

1.* Nedeterministický stroj rozhoduje jazyk L, pokud se pro každé vstupní slovo α všechny výpočty stroje zastaví a $\alpha \in L$ právě tehdy, když aspoň jeden výpočet skončí ve stavu q_+. Dokažte, že každý takový jazyk je rozhodovaný i nějakým deterministickým strojem.

2. Doplňte detaily do simulace gramatiky Turingovým strojem.

3.* Doplňte detaily do simulace RAMu Turingovým strojem.

4. Ukažte, jak simulovat generování slova gramatikou nedeterministickým Turingovým strojem. Nedeterminismus použijte na výběr pravidla, které se má v daném kroku přepisování použít, a výběr místa v řetězci, kde se má pravidlo aplikovat.

3.4 Nerozhodnutelné problémy

V tomto oddílu ukážeme, že existují jazyky, které nejsou rozhodnutelné, a to ani částečně. K tomu se bude hodit vyjmenovat všechny Turingovy stroje.

Definice: Zavedeme *kódování* Turingových strojů, které každému stroji s jednou páskou a vstupní abecedou $\{\mathtt{0},\mathtt{1}\}$ přiřadí nějaké slovo z $\{\mathtt{0},\mathtt{1}\}^*$:

- Očíslujeme stavy stroje: $Q = \{s_1, s_2, \ldots, s_{|Q|}\}$, přičemž $s_1 = q_0$, $s_2 = q_+$, $s_3 = q_-$.
- Očíslujeme pracovní abecedu: $\Gamma = \{x_1, \ldots, x_{|\Gamma|}\}$, přičemž $x_1 = \mathtt{0}$, $x_2 = \mathtt{1}$, $x_3 = \sqcup$.
- Očíslujeme směry: $d_1 = \leftarrow$, $d_2 = \rightarrow$, $d_3 = \bullet$.
- Zakódujeme přechody: $\delta(s_i, x_j) = (s_k, x_\ell, d_m)$ zapíšeme řetězcem $\mathtt{0}^i\mathtt{1}\mathtt{0}^j\mathtt{1}\mathtt{0}^k\mathtt{1}\mathtt{0}^\ell\mathtt{1}\mathtt{0}^m\mathtt{1}\mathtt{1}$. Všimneme si, že uvnitř řetězce nejsou dvě jedničky za sebou, takže podle $\mathtt{1}\mathtt{1}$ bezpečně poznáme konec.
- Kód stroje získáme jako zřetězení všech přechodů v libovolném pořadí.

Pozorování: Různé stroje mohou dostat stejný kód, pokud se liší pouze pojmenováním stavů a znaků pracovní abecedy. Takové stroje nicméně počítají totéž (rozpoznávají i rozhodují tytéž jazyky), takže je není třeba rozlišovat.

Definice: Pro slovo $\alpha \in \{\mathtt{0},\mathtt{1}\}^*$ definujeme M_α jako stroj s kódem α. Pokud slovo α není korektním kódem stroje, bude M_α stroj, který se hned zastaví ve stavu q_-, a tedy přijímá i rozpoznává prázdný jazyk.

Poznámka: Stejným způsobem můžeme zakódovat i všechny částečně rozhodnutelné jazyky: L_α bude jazyk přijímaný strojem M_α, tedy $L_\alpha = L(M_\alpha)$. Jen pozor na to, že každý jazyk $L \in \mathsf{RE}$ nalezneme v tomto kódování nekonečně-krát ($L = L_\alpha$ pro nekonečně mnoho různých kódů α), protože je přijímán nekonečně mnoha stroji – například můžeme do stroje libovolně přidávat nedosažitelné stavy.

Nyní nadefinujeme univerzální jazyk, který v jistém smyslu obsahuje všechny částečně rozhodnutelné jazyky.

Definice: *Kódování dvojic* přiřadí každé uspořádané dvojici (α, β) řetězců $\alpha, \beta \in \{\mathtt{0},\mathtt{1}\}^*$ nějaký řetězec $\langle\alpha, \beta\rangle \in \{\mathtt{0},\mathtt{1}\}^*$, z nějž lze dvojici jednoznačně rekonstruovat. Navíc jak zakódování, tak dekódování jsou vyčíslitelné funkce.

Příklad: Kódování dvojic můžeme sestrojit třeba takto:

$$\langle a_1 \ldots a_n, b_1 \ldots b_m\rangle = \mathtt{0}a_1\mathtt{0}a_2 \ldots \mathtt{0}a_n\mathtt{1}\mathtt{0}b_1\mathtt{0}b_2 \ldots \mathtt{0}b_m.$$

Snadno ověříme, že je jednoznačně dekódovatelné.

Definice: *Univerzální jazyk* $L_u \subseteq \{0,1\}^*$ obsahuje všechny dvojice $\langle \alpha, \beta \rangle$, kde $\alpha, \beta \in \{0,1\}^*$ a $\beta \in L(M_\alpha)$.

Lemma: Univerzální jazyk je částečně rozhodnutelný.

Náčrt důkazu: Sestrojíme *Univerzální Turingův stroj (UTM),* který dovede simulovat libovolný jiný Turingův stroj. Na vstupu dostane dvojici $\langle \alpha, \beta \rangle$ a bude krok po kroku simulovat výpočet stroje M_α na vstupu β. Pokud se stroj M_α zastaví, UTM se také zastaví a vydá stejný verdikt. Pokud se M_α nezastaví, simulace bude pokračovat do nekonečna.

Zhruba popíšeme konstrukci UTM. Bude to vícepáskový stroj, který pak redukujeme na jednopáskový.

- Na pásce K bude uložen kód α simulovaného stroje.
- Na pásce P budeme udržovat obsah pásky simulovaného stroje. Jelikož UTM musí mít pevnou pracovní abecedu, a přitom umět simulovat stroj s libovolně velkou abecedou, je potřeba symboly pásky kódovat. UTM si z kódu α zjistí počet znaků k v pracovní abecedě a hodnotu k si zapíše na pomocnou pásku. Na pásce P pak bude mít uložené k-znakové *krabičky* oddělené znakem `#`. Krabička tvaru $\mathtt{1}^i\mathtt{0}^{k-i}$ kóduje i-tý znak abecedy simulovaného stroje.
- Polohu hlavy simulovaného stroje si budeme pamatovat v poloze hlavy UTM na pásce P. Hlava UTM se bude nacházet někde uvnitř příslušné krabičky, podle symbolu `#` umíme kdykoliv najít začátek krabičky.
- Na pásce S bude uložen aktuální stav simulovaného stroje v jedničkové soustavě. (Opět nelze ukládat tento stav do stavu UTM, protože simulovaný stroj může mít libovolně mnoho stavů.)
- Na začátku výpočtu UTM dekóduje dvojici $\langle \alpha, \beta \rangle$. Zkontroluje, zda α je platný kód stroje, a jinak vstup odmítne. Přepíše slovo β do krabiček na pásce P. Na pásku S zapíše počáteční stav 1.
- Jeden krok stroje bude UTM simulovat takto: projde kód α zleva doprava a najdeme přechod, který odpovídá aktuálnímu stavu a znaku na pásce. Jelikož máme vše kódované v jedničkové soustavě, porovnání je triviální. Až přechod najde, přepíše krabičku na pásce P (to jde na místě) i stav na pásce S a přesune hlavu na pásce P do sousední krabičky. Pokud sousední krabička ještě neexistuje, založí ji a vyplní znakem číslo 3 (mezerou).
- UTM kroky opakuje, dokud simulovaný stroj nepřejde do stavu 2 (přijímací) nebo 3 (odmítací). Podle toho sám buď přijme, nebo odmítne. □

Univerzální jazyk tedy je částečně rozhodnutelný. Za chvíli se ovšem ukáže, že jeho doplněk není částečně rozhodnutelný. Nejprve to ale dokážeme o jiném jazyku.

Definice: *Diagonální jazyk* L_d je jazyk nad abecedou $\{0, 1\}$, který obsahuje všechna slova α taková, že $\alpha \notin L(M_\alpha)$.

Lemma: Diagonální jazyk není částečně rozhodnutelný.

Důkaz: Pro spor předpokládejme, že L_d je částečně rozhodnutelný. Tedy je přijímán nějakým Turingovým strojem. Nechť tento stroj má kód α. Platí tedy $L_d = L(M_\alpha)$.

Položme si otázku, zda slovo α leží v L_d: podle definice L_d je to právě tehdy, když $\alpha \notin L(M_\alpha)$. Jenže $L(M_\alpha) = L_d$, takže je to právě tehdy, když $\alpha \notin L_d$. Čili α leží v L_d právě tehdy, když tam neleží, což je spor. □

Důsledek: Doplněk univerzálního jazyka není částečně rozhodnutelný.

Důkaz: Ukážeme, že kdyby existoval stroj přijímající $\overline{L_u}$, mohli bychom ho upravit, aby přijímal diagonální jazyk L_d. Nový stroj dostane na vstupu slovo α. Vytvoří z něj dvojici $\langle \alpha, \alpha \rangle$ a na ni spustí stroj pro $\overline{L_u}$. To funguje, jelikož $\langle \alpha, \alpha \rangle \in \overline{L_u} \Leftrightarrow \alpha \notin L(M_\alpha) \Leftrightarrow \alpha \in L_d$. Dostali jsme spor s tím, že L_d není částečně rozhodnutelný. □

Poznámka: Tady je vidět, proč se jazyku L_d říká diagonální: Představíme si nekonečnou tabulku, která bude mít na jedné ose všechny kódy strojů α, na druhé ose budou všechny vstupy β a okénka tabulky budou vyplněna `0` a `1` podle toho, zda $\langle \alpha, \beta \rangle \in L_u$. Pokud na diagonále tabulky prohodíme nuly a jedničky, dostaneme jazyk L_d. Tím pádem se L_d nemůže vyskytovat v žádném řádku tabulky: došli bychom ke sporu v místě, kde řádek protíná diagonálu. Neexistuje tedy žádný stroj přijímající L_d.

Důsledek: Univerzální jazyk není rozhodnutelný.

Důkaz: Kdyby L_u byl rozhodnutelný, byl by i $\overline{L_u}$ rozhodnutelný – stačilo by ve stroji prohodit stavy q_+ a q_-. Tím pádem by byl $\overline{L_u}$ i částečně rozhodnutelný, což je ve sporu s předchozím důsledkem. □

Poznámka: Zjistili jsme tedy, že jazyk L_u je částečně rozhodnutelný, ale není rozhodnutelný. Jazyk $\overline{L_u}$ a jazyk L_d nejsou ani částečně rozhodnutelné. Inkluze tříd $\mathsf{R} \subset \mathsf{RE} \subset \mathcal{L}$ jsou tedy všechny ostré.

Poznámka: Jiný pohled na totéž je tzv. *problém zastavení (halting problem):* máme najít algoritmus, který dostane kód programu a jeho vstup a ma rozhodnout, zda se program pro daný vstup zastaví. Takový algoritmus ovšem nemůže existovat, protože by rozhodoval jazyk L_u.

Věta (Postova): Jazyk L je rozhodnutelný právě tehdy, když L i $\overline{L}$ jsou částečně rozhodnutelné.

Důkaz: Jednu implikaci jsme už použili v důkazu nerozhodnutelnosti jazyka L_u: pokud L je rozhodnutelný, pak i $\overline{L}$ je rozhodnutelný (prohodíme stavy q_+ a q_-). Tím pádem jsou oba částečně rozhodnutelné.

Obrácená implikace je zajímavější: mějme stroj A rozhodující L a stroj B rozhodující $\overline{L}$. Představíme si, že oba stroje spustíme současně na tomtéž vstupu (podobně jako u součinu konečných automatů). Vytvoříme nový stroj M se dvěma páskami, z nichž jedna odpovídá stroji A a druhá stroji B. Stav stroje M bude určovat stavy obou původních strojů A a B. Stroj M v jednom kroku provede jak krok stroje A, tak krok stroje B. Pokud stroj A přejde do stavu q_+, stroj M také. Pokud stroj B přejde do q_+, stroj M do q_-. A jelikož každý vstup leží buď v L, nebo v $\overline{L}$, určitě se časem buď A nebo B zastaví. Stroj M tedy rozhoduje jazyk L. □

Cvičení

1. Dokažte, že třídy R a RE jsou uzavřené na sjednocení, průnik, zřetězení a iteraci. Třída R je uzavřená na doplňky, třída RE nikoliv.

2.* Zvolme pevnou abecedu Σ o aspoň dvou znacích. Dokažte, že množina všech jazyků nad Σ je nespočetná, zatímco množina všech částečně rozhodnutelných jazyků nad Σ spočetná. Z toho plyne nejen to, že některé jazyky nejsou ani částečně rozhodnutelné, ale že takové jsou skoro všechny jazyky.

3.* Doplňte detaily univerzálního Turingova stroje.

4. Kolik času a prostoru spotřebuje univerzální Turingův stroj? Porovnejte s časovou a prostorovou složitostí simulovaného stroje.

5. *Převody jazyků.* Důkaz $L_u \notin \mathsf{RE}$ pomocí $L_d \notin \mathsf{RE}$ připomíná převody NP-úplných problémů, jen roli převodu hraje obecná vyčíslitelná funkce (ne nutně počítaná v polynomiálním čase). Jazyk A nad abecedou Σ lze převést na jazyk B nad abecedou Δ (značíme $A \to B$), pokud existuje vyčíslitelná funkce $f : \Sigma^* \to \Delta^*$ taková, že pro všechna $\alpha \in \Sigma^*$ platí $\alpha \in A \Leftrightarrow f(\alpha) \in B$. Dokažte, že pokud $A \to B$ a B je (částečně) rozhodnutelný, pak A je také (částečně) rozhodnutelný. Takže není-li A (částečně) rozhodnutelný, nemůže být ani B (částečně) rozhodnutelný.

6.* Uvažme jazyk všech $\alpha \in \{0,1\}^*$ takových, že M_α s prázdným vstupem se zastaví. Dokažte, že tento jazyk je částečně rozhodnutelný, ale není rozhodnutelný.

7.* Uvažme jazyk všech dvojic $\langle \alpha, \beta \rangle$ takových, že stroje M_α a M_β přijímají tentýž jazyk. Dokažte, že tento jazyk není částečně rozhodnutelný.

8.* Definujme funkci $f : \mathbb{N} \to \mathbb{N}$. Číslo $f(n)$ bude udávat maximální počet kroků, za který se zastaví Turingův stroj s kódem délky nejvýše n na vstupu délky nejvýše n

(stroje, které se nezastaví, nezapočítáváme). Dokažte, že funkce f není vyčíslitelná. Dokažte, že funkce f roste rychleji než každá vyčíslitelná funkci z $\mathbb{N}$ do $\mathbb{N}$. (Čísla kódujeme ve dvojkové soustavě.)

9.* Dokažte, že jazyk je částečně vyčíslitelný právě tehdy, když lze všechna jeho slova algoritmicky vyjmenovat. Tedy existuje interaktivní Turingův stroj, který pro spuštění postupně vypisuje všechna slova jazyka tak, že každé slovo jazyka je v konečném čase vypsáno.

10.* Dokažte, že jazyk je vyčíslitelný právě tehdy, když lze všechna jeho slova vypsat v délkově-lexikografickém pořadí (slova porovnáváme podle délky, v rámci téže délky pak lexikograficky).

3.5 Inventura jazyků

V předchozích kapitolách jsme zavedli několik tříd jazyků:

- $\mathcal{L}_3$ je třída regulárních jazyků. Ty jsou rozpoznávané konečnými automaty (deterministickými i nedeterministickými), generované regulárními výrazy a levými i pravými lineárními gramatikami.
- $\mathcal{L}_2$ je třída bezkontextových jazyků, generovaných bezkontextovými gramatikami.
- R je třída rozhodnutelných (rekurzivních) jazyků. Ty jsou rozpoznávány Turingovým strojem, který se vždy zastaví.
- RE je třída částečně rozhodnutelných (rekurzivně spočetných) jazyků, které Turingův stroj přijímá zastavením.
- $\mathcal{L}_0$ je třída jazyků generovaných obecnými gramatikami.
- $\mathcal{L}$ je třída všech jazyků.

Pojďme shrnout, co jsme o nich zjistili:

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathsf{R} \subset \mathsf{RE} = \mathcal{L}_0 \subset \mathcal{L}.$$

Vztahy zleva doprava:

- Lineární gramatiky jsou speciálním případem bezkontextových. Jazyk $\mathtt{0}^n\mathtt{1}^n$ je bezkontextový, ale není regulární.
- Bezkontextové jazyky jsou rozpoznatelné algoritmem CYK. Jazyk $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$ je rozhodnutelný, ale není bezkontextový.
- Jazyky generované gramatikami jsou právě ty částečně rozhodnutelné.
- Rozhodnutelný jazyk je i částečně rozhodnutelný. Jazyk L_u leží v $\mathsf{RE} \setminus \mathsf{R}$.
- Jazyky $\overline{L_u}$ a L_d nejsou částečně rozhodnutelné.

Složitostní třídy

Další zajímavé třídy jazyků získáme omezením časové nebo prostorové složitosti Turingova stroje, který je rozhoduje:

Definice: Nechť f je funkce z $\mathbb{N}$ do $\mathbb{N}$. Potom:

- $L \in \mathsf{TIME}(f)$ právě tehdy, když je rozhodován Turingovým strojem, který se pro vstup délky n zastaví do $f(n)$ kroků.
- $L \in \mathsf{SPACE}(f)$ právě tehdy, když je rozhodován Turingovým strojem, který pro vstup délky n spotřebuje prostor nejvýše $f(n)$.
- $\mathsf{NTIME}(f)$ a $\mathsf{NSPACE}(f)$ jsou definovány analogicky přes nedeterministické stroje.
- P je sjednocení tříd $\mathsf{TIME}(n^k)$ přes všechna $k \geq 0$.
- NP je sjednocení tříd $\mathsf{NTIME}(n^k)$ přes všechna $k \geq 0$.

Všechny jazyky v těchto třídách jsou rozhodnutelné.

Cvičení

1. Dokažte, že $\mathsf{SPACE}(c)$ a $\mathsf{NSPACE}(c)$ jsou pro všechny konstanty $c \in \mathbb{N}$ rovny třídě regulárních jazyků $\mathcal{L}_1$.

2.* Třídy P a NP jsme už jednou zavedli v kapitole Průvodce o těžkých problémech (ale pouze pro binární abecedu, zatímco zde připouštíme obecnou). Třídu NP jsme v Průvodci definovali pomocí certifikátů. Dokažte, že je to ekvivalentní se zdejší definicí pomocí nedeterministického stroje.

3.* Původní Chomského hierarchie obsahovala ještě třídu $\mathcal{L}_1$. Ta se dá zavést například jako jazyky generované *nezkracujícími gramatikami*. Všechna pravidla těchto gramatik mají pravou stranu aspoň tak dlouhou jako levou. Výjimka se připouští pouze

pro pravidlo $S \to \varepsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla. Bezkontextové gramatiky v ChNF jsou nezkracující. Dokažte, že jazyk všech $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$ leží v $\mathcal{L}_1$, takže inkluze $\mathcal{L}_2 \subset \mathcal{L}_1$ je ostrá. Dále dokažte, že $\mathcal{L}_1 = \mathsf{NSPACE}(n)$. Z toho speciálně plyne, že $\mathcal{L}_1 \subseteq \mathsf{R}$. Uměli byste dokázat, že tato inkluze je ostrá?