

Teoretická informatika

TIN

Studijní opora

M. Češka, T. Vojnar, A. Smrčka

10. září 2010

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

Obsah

1	Úvod	5
1.1	Obsahové a metodické informace o předmětu Teoretická informatika	6
1.1.1	Cíle předmětu	6
1.1.2	Anotace předmětu	6
1.1.3	Požadované prerekvizitní znalosti a dovednosti	6
1.1.4	Osnova přednášek a přiřazení ke kapitolám opory	6
2	Jazyky, gramatiky a jejich klasifikace	8
2.1	Jazyky	8
2.2	Gramatika	12
2.3	Chomského klasifikace gramatik	16
2.3.1	Typ 0	16
2.3.2	Typ 1	16
2.3.3	Typ 2	17
2.3.4	Typ 3	17
2.4	Cvičení	18
3	Regulární jazyky	21
3.1	Jazyky přijímané konečnými automaty a deterministický konečný automat	21
3.1.1	Nedeterministický konečný automat	21
3.1.2	Lineární a regulární gramatiky	24
3.1.3	Ekvivalence třídy \mathcal{L}_3 a třídy jazyků přijímaných konečnými automaty	28
3.2	Minimalizace deterministického konečného automatu	33
3.3	Regulární množiny a regulární výrazy	36
3.3.1	Regulární množiny	36
3.3.2	Regulární výrazy	37
3.3.3	Rovnice nad regulárními výrazy	40
3.3.4	Soustavy rovnic nad regulárními výrazy	40
3.4	Převod regulárních výrazů na konečné automaty	42
3.5	Vlastnosti regulárních jazyků	46
3.5.1	Strukturální vlastnosti regulárních jazyků	47
3.5.2	Myhill-Nerodova věta	48
3.5.3	Uzávěrové vlastnosti regulárních jazyků	50
3.5.4	Rozhodnutelné problémy regulárních jazyků	51
3.6	Cvičení	52

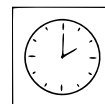
4	Bezkontextové jazyky a zásobníkové automaty	54
4.1	Derivační strom	55
4.2	Fráze větné formy	58
4.3	Víceznačnost gramatik	59
4.4	Rozklad věty	63
4.5	Transformace bezkontextových gramatik	65
4.6	Chomského normální forma	74
4.7	Greibachové normální forma	75
4.8	Cvičení	77
4.9	Základní definice zásobníkového automatu	80
4.10	Varianty zásobníkových automatů	82
4.11	Ekvivalence bezkontextových jazyků a jazyků přijímaných zásob- níkovými automaty	86
4.12	Deterministický zásobníkový automat	92
4.13	Vlastnosti bezkontextových jazyků	93
4.13.1	Strukturální vlastnosti	93
4.13.2	Uzávěrové vlastnosti	95
4.13.3	Rozhodnutelné a nerozhodnutelné problémy pro bezkontex- tové jazyky	97
4.13.4	Uzávěrové vlastnosti jazyků deterministických bezkontexto- vých jazyků	98
4.13.5	Některé další zajímavé vlastnosti bezkontextových jazyků	99
4.14	Cvičení	100
5	Turingovy stroje	102
5.1	Základní koncepce Turingových strojů	102
5.1.1	Churchova teze	102
5.1.2	Turingův stroj	102
5.1.3	Konfigurace Turingova stroje	103
5.1.4	Přechodová relace TS	103
5.1.5	Výpočet TS	104
5.1.6	Poznámka – alternativní definice TS	104
5.1.7	Grafická reprezentace TS	104
5.1.8	Modulární konstrukce TS	105
5.1.9	Kompozitní diagram TS	105
5.1.10	Základní stavební bloky TS	106
5.1.11	Příklady TS	107
5.2	Turingovy stroje jako akceptory jazyků	108
5.2.1	Jazyk přijímaný TS	108
5.3	Modifikace TS	109
5.3.1	Přijímání zvláštních konfigurací pásky	109
5.3.2	Vícepáskové Turingovy stroje	110
5.3.3	Nedeterministické Turingovy stroje	112
5.4	Jazyky rekurzivně vyčíslitelné a jazyky rekurzivní	114
5.4.1	Rekurzivní vyčíslitelnost a rekurzivnost	114
5.4.2	Rozhodovací problémy	114
5.4.3	Rozhodování problémů TS	115
5.5	TS a jazyky typu 0	115
5.5.1	Jazyky přijímané TS jsou typu 0	115
5.5.2	Jazyky typu 0 jsou přijímány TS	116
5.5.3	Jazyky typu 0 = jazyky přijímané TS	117
5.6	Vlastnosti jazyků rekurzivních a rekurzivně vyčíslitelných	117

5.6.1	Uzavřenost vůči \cup, \cap, \cdot a $*$	117
5.6.2	(Ne)uzavřenost vůči komplementu	119
5.7	Lineárně omezené automaty	119
5.7.1	Lineárně omezené automaty	120
5.7.2	LOA a kontextové jazyky	120
5.7.3	Kontextové a rekurzivní jazyky	120
5.7.4	Vlastnosti kontextových jazyků	121
5.8	Cvičení	122
6	Meze rozhodnutelnosti	123
6.1	Jazyky mimo třídu 0	123
6.1.1	Existence jazyků mimo třídu 0	123
6.2	Problém zastavení	124
6.2.1	Kódování TS	124
6.2.2	Univerzální TS	125
6.2.3	Problém zastavení TS	125
6.3	Redukce	127
6.3.1	Důkaz nerozhodnutelnosti redukcí	127
6.4	Problém náležitosti a další problémy	128
6.4.1	Problém náležitosti pro \mathcal{L}_0	128
6.4.2	Příklady dalších problémů pro TS	129
6.5	Postův korespondenční problém	129
6.5.1	Postův korespondenční problém	130
6.5.2	Nerozhodnutelnost PCP	130
6.5.3	Nerozhodnutelnost redukcí z PCP	131
6.5.4	Souhrn některých vlastností jazyků	131
6.6	Riceova věta	132
6.6.1	Riceova věta – první část	132
6.6.2	Důkaz 1. části Riceovy věty	133
6.6.3	Riceova věta – druhá část	133
6.7	Alternativy Turingova stroje	134
6.8	Vyčíslitelné funkce	134
6.8.1	Základy teorie rekurzivních funkcí	134
6.8.2	Počáteční funkce	135
6.8.3	Primitivně rekurzivní funkce	135
6.8.4	Příklady primitivně rekurzivních funkcí	137
6.8.5	Funkce mimo primitivně rekurzivní funkce	139
6.8.6	Parciálně rekurzivní funkce	139
6.9	Vztah vyčíslitelných funkcí a Turingových strojů	140
6.9.1	Turingovsky vyčíslitelné funkce	140
6.9.2	Turingovská vyčíslitelnost parciálně rekurzivních funkcí	141
6.9.3	Reprezentace Turingova stroje parciálně rekurzivními funkcemi	142
6.10	Cvičení	144
7	Složitost	146
7.1	Základní pojmy složitosti	146
7.1.1	Složitost algoritmů	146
7.1.2	Různé případy při analýze složitosti	146
7.1.3	Složitost výpočtů TS	147
7.1.4	Složitost a cena atomických operací	148
7.1.5	Složitost výpočtů na TS a v jiných prostředích	148

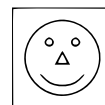
7.1.6	Asymptotická omezení složitosti	150
7.2	Třídy složitosti	151
7.2.1	Složitost problémů	151
7.2.2	Třídy složitosti	152
7.2.3	Časově/prostorově konstruovatelné funkce	152
7.2.4	Nejběžněji užívané třídy složitosti	153
7.2.5	Třídy pod a nad polynomiální složitostí	153
7.2.6	Třídy nad exponenciální složitostí	154
7.2.7	Vrchol hierarchie tříd složitosti	154
7.3	Vlastnosti tříd složitosti	154
7.3.1	Vícepáskové stroje	154
7.3.2	Determinismus a nedeterminismus	155
7.3.3	Prostor kontra čas	157
7.3.4	Uzavřenost vůči doplňku	157
7.3.5	Ostrost hierarchie tříd	158
7.3.6	Některé další zajímavé výsledky	158
7.3.7	\mathcal{R} redukce, jazyky \mathcal{C} -těžké a \mathcal{C} -úplné	159
7.3.8	Nejběžnější typy \mathcal{R} redukce a úplnosti	159
7.4	Příklady složitosti problémů	160
7.5	SAT-problém	161
7.5.1	Polynomiální redukce	161
7.5.2	Problém splnitelnosti - SAT problém	162
7.5.3	NP-úplné jazyky	163
7.5.4	Význačné NP-úplné problémy	163
7.5.5	Použití redukcí k důkazu úplnosti	164
7.6	Cvičení	165

Kapitola 1

Úvod



0:20



Teorie formálních jazyků a vyčíslitelnost představují klasické a velmi důležité oblasti teoretické informatiky. Základy novodobé historie těchto disciplín položil v roce 1956 americký matematik NOAM CHOMSKY, který v souvislosti se studiem přirozených jazyků vytvořil matematický model gramatiky jazyka.

Realizace původních představ, formalizovat popis přirozeného jazyka takovým způsobem, aby mohl být automatizován překlad z jednoho přirozeného jazyka do druhého nebo aby přirozený jazyk sloužil jako prostředek komunikace člověka s počítačem, se ukázala velmi obtížnou a teprve současné výsledky můžeme považovat alespoň za slibné.

Začala se však vyvíjet vlastní teorie jazyků, která nyní obsahuje bohaté výsledky v podobě matematicky dokázaných tvrzení – teorémů. Tato teorie pracuje se dvěma duálními matematickými entitami, s gramatikou a s automatem, představující abstraktní matematický stroj. Zatímco gramatika umožňuje většinou snáze popsat strukturu vět formálního jazyka, automat dovede tuto strukturu snáze identifikovat a zpracovávat.

Poznatky teorie formálních jazyků mají význam nejen ve všech oblastech informatiky a informačních technologií, ale jejich aplikace zasahují do téměř nespočetné řady technických i netechnických oborů. Dodávají algoritmy, jež jsou podkladem pro konstrukci reálných programů či zařízení zpracovávajících informaci ve tvaru vět formálního jazyka. Stanovují však také možnosti a omezení algoritmických postupů řešení problémů; odhalují problémy, které jsou algoritmicky nerozhodnutelné, tj. problémy, jejichž řešení nelze dosáhnout v konečném čase.

Teorie formálních jazyků našla doposud největší uplatnění v oblasti programovacích jazyků. Krátce po CHOMSKÉHO definici gramatiky formálního jazyka a klasifikaci formálních jazyků (CHOMSKÉHO hierarchii formálních jazyků), BACKUS a NAUER použili základních objektů gramatiky pro definici syntaxe programovacího jazyka *Algol 60* (ve tvaru formalismu, jež se nazývá Backus-Naueroва forma). Další vývoj pak přímočaře vedl k aplikacím teorie jazyků v oblasti překladačů programovacích jazyků. Stanovení principů syntaxí řízeného překladu a generátorů překladačů (programovacích systémů, které na základě formálního popisu syntaxe a sémantiky programovacího jazyka vytvoří jeho překladač) představuje kvalitativní skok při konstrukci překladačů umožňující automatizovat náročnou programátorskou práci spojenou s implementací programovacích jazyků. V současné době je teorie formálních jazyků spolu s matematickou logikou základem ambiciozních výzkumných programů zaměřených na formální verifikace technických i programových prostředků, vedoucí k větší spolehlivosti a bezpečnosti počítačových aplikací.

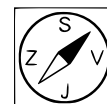
	Čas potřebný pro studium		Otázka, příklad k řešení
	Cíl		Počítačové cvičení, příklad
	Definice		Příklad
	Důležitá část		Reference
	Rozšiřující látka		Správné řešení
	Obtížná část		Souhrn
			Slovo tutora, komentář
			Zajímavé místo

Tabulka 1.1: Význam používaných piktogramů

1.1 Obsahové a metodické informace o předmětu Teoretická informatika

1.1.1 Cíle předmětu

Rozšíření znalostí teorie formálních jazyků a osvojení základů teorie výčísitelnosti a základních pojmů výpočetní složitosti.

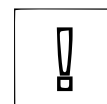


1.1.2 Anotace předmětu

Aplikace teorie formálních jazyků v informatice a informačních technologiích (překladače, modelování a analýza systémů, lingvistika, biologie atd.), modelovací a rozhodovací síla formálního modelu, regulární jazyky a jejich vlastnosti, minimalizace konečného automatu, bezkontextové jazyky a jejich vlastnosti, Turingovy stroje, vlastnosti rekurzivních a rekurzivně výčíslitelných jazyků, výčíslitelné funkce, nerozhodnutelnost, nerozhodnutelné problémy teorie formálních jazyků, úvod do výpočetní složitosti.

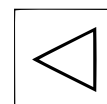
1.1.3 Požadované prerekvizitní znalosti a dovednosti

Základní znalosti z binárních relací, teorie grafů a formálních jazyků včetně konečných a zásobníkových automatů a pojmů algoritmické složitosti.



1.1.4 Osnova přednášek a přiřazení ke kapitolám opory

Kapitola 2 Úvod, aplikace teorie formálních jazyků, modelovací a rozhodovací síla formálního modelu, operace nad jazyky.



Kapitola 3 Regulární jazyky a jejich vlastnosti, Kleenova věta, Nerodova věta, věta o vkládání (Pumping theorem). Minimalizace konečného automatu, relace nerozlišitelnosti stavů, konstrukce redukovaného konečného automatu. Uzávěrové vlastnosti regulárních jazyků, regulární jazyky jako množinová Booleova algebra, rozhodnutelné problémy regulárních jazyků.

Kapitola 4 Bezkontextové jazyky a jejich vlastnosti. Normální tvary bezkontextových gramatik, jednoznačné a deterministické bezkontextové jazyky, věta o vkládání pro bezkontextové jazyky. Zásobníkové automaty, varianty zásobníkových automatů, ekvivalence bezkontextových jazyků a jazyků přijímaných zásobníkovými automaty, deterministický zásobníkový automat. Uzávěrové vlastnosti bezkontextových jazyků, uzavřenost vzhledem k substituci, důsledky, rozhodnutelné problémy bezkontextových jazyků.

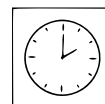
Kapitola 5 Turingovy stroje (TS), definice TS a jazyka přijímaného TS, rekurzivně vyčíslitelné a rekurzivní jazyky a problémy, TS a funkce, metody konstrukce TS. Modifikace TS, TS s obousměrně nekonečnou páskou, s více páskami, nedeterministický TS, stroj se dvěma zásobníky, stroje s čítači. TS a jazyky typu 0, diagonalizace, vlastnosti rekurzivních a rekurzivně vyčíslitelných jazyků, lineárně ohraničené automaty a jazyky typu 1. Vyčíslitelné funkce, počáteční funkce, primitivně rekurzivní funkce, mí-rekurzivní funkce, vztah vyčíslitelných funkcí a Turingových strojů. Church-Turingova téze, univerzální TS, nerozhodnutelnost, problém zastavení TS, redukce, Postův korespondenční problém.

Kapitola 6 Nerozhodnutelné problémy teorie formálních jazyků.

Kapitola 7 Úvod do výpočetní složitosti, Turingovská složitost, třída P a NP problémů.

Kapitola 2

Jazyky, gramatiky a jejich klasifikace



10:00

Cílem této kapitoly je pochopení pojmu formální jazyk, jeho matematické explikace (popisu) a universalitu a základních algebraických operací nad formálními jazyky, které nacházejí praktické uplatnění v praktických aplikacích. Dále pak pochopení základního způsobu specifikace formálního jazyka gramatikou a způsobu klasifikace formálních jazyků, založeného na klasifikaci gramatik.



2.1 Jazyky

Jedním ze základních pojmů pro vymezení jazyka jsou pojmy *abeceda* a *řetězec*.

Definice 2.1 Abecedou rozumíme neprázdnou množinu prvků, které nazýváme *symbols* *abecedy*.



V některých teoretických případech je účelné pracovat i s abecedami nekonečnými, v našich aplikacích se však omezíme na abecedy *konečné*.

Příklad 2.1 Jako příklady abeced můžeme uvést *latinskou abecedu* obsahující 52 symbolů, které reprezentují velká a malá písmena, *řeckou abecedu*, dvouprvkovou *binární abecedu* reprezentovanou množinou $\{0, 1\}$ resp. $\{\text{true}, \text{false}\}$ nebo abecedy programovacích jazyků.

Definice 2.2 *Řetězcem* (také slovem nebo větou) nad danou abecedou rozumíme každou konečnou posloupnost symbolů abecedy. Prázdnou posloupnost symbolů, tj. posloupnost, která neobsahuje žádný symbol, nazýváme *prázdný řetězec*. Prázdný řetězec budeme označovat písmenem ϵ .



Formálně lze definovat řetězec nad abecedou Σ takto:

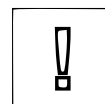
- (1) prázdný řetězec ϵ je řetězec nad abecedou Σ ,
- (2) je-li x řetězec nad Σ a $a \in \Sigma$, pak xa je řetězec nad Σ ,
- (3) y je řetězec nad Σ , když a jen když lze y získat aplikací pravidel 1 a 2.

Příklad 2.2 Je-li $A = \{a, b, +\}$ abeceda, pak

$$\epsilon \quad a \quad b \quad + \quad aa \quad a + b \quad + ab$$

jsou některé řetězce nad abecedou A . Pořadí symbolů v řetězci je významné; řetězce $a + b$, $+ab$ jsou různé. Symbol b , například, je řetězec nad A , poněvadž $\epsilon b = b$ (pravidlo (2)).

Konvence 2.1 Budeme-li pracovat s obecnými abecedami, symbols, či řetězci, pak pro jejich značení použijeme:



- velká řecká písmena pro abecedy,
- malá latinská písmena a, b, c, d, f, \dots pro symboly,
- malá latinská písmena t, u, v, w, \dots pro řetězce.

Nyní uvedeme některé důležité operace nad řetězci.

Definice 2.3 Necht x a y jsou řetězce nad abecedou Σ . *Konkatenací* (zřetěžením) řetězce x s řetězcem y vznikne řetězec xy připojením řetězce y za řetězec x . Operace konkatenace je zřejmě asociativní, tj. $x(yz) = (xy)z$, ne však komutativní, $xy \neq yx$.

DEF

Příklad 2.3 Je-li $x = ab, y = ba$, pak $xy = abba, yx = baab, x\epsilon = \epsilon x = ab$, kde ϵ je prázdný řetězec.

Definice 2.4 Necht $x = a_1a_2 \dots a_n$ je řetězec nad abecedou $\Sigma, a_i \in \Sigma$ pro $i = 1, \dots, n$. *Reverzí* (zrcadlovým obrazem) řetězce x rozumíme řetězec $x^R = a_na_{n-1} \dots a_2a_1$; tj. symboly řetězce x^R jsou vzhledem k řetězci x zapsány v opačném pořadí.

DEF

Příklad 2.4 Je-li $x = abbc$, pak $x^R = cbba$. Zřejmě $\epsilon^R = \epsilon$.

Definice 2.5 Necht w je řetězec nad abecedou Σ . Řetězec z se nazývá *podřetězcem* řetězce w , jestliže existují řetězce x a y takové, že $w = xzy$. Řetězec x_1 se nazývá *prefixem* (předponou) řetězce w , jestliže existuje řetězec y_1 takový, že $w = x_1y_1$. Analogicky, řetězec y_2 se nazývá *sufixem* (příponou) řetězce w , jestliže existuje řetězec x_2 takový, že $w = x_2y_2$. Je-li $y_1 \neq \epsilon$, resp. $x_2 \neq \epsilon$, pak x_1 je *vlastní prefix*, resp. y_2 je *vlastní suffix* řetězce w .

DEF

Příklad 2.5 Je-li $w = abbc$, pak

$$\epsilon \quad a \quad ab \quad abb \quad abbc$$

jsou všechny prefixy řetězce w (první čtyři jsou vlastní),

$$\epsilon \quad c \quad bc \quad bbc \quad abbc$$

jsou všechny suffixy řetězce w (první čtyři jsou vlastní) a

$$a \quad bb \quad abb$$

jsou některé podřetězce řetězce w .

Zřejmě, prefix i suffix jsou podřetězce řetězce w , prázdný řetězec je podřetězcem, prefixem i suffixem každého řetězce.

Definice 2.6 *Délka řetězce* je nezáporné celé číslo udávající počet symbolů řetězce. Délku řetězce x značíme symbolicky $|x|$. Je-li $x = a_1a_2 \dots a_n, a_i \in \Sigma$ pro $i = 1, \dots, n$, pak $|x| = n$. Délka prázdného řetězce je nulová, tj. $|\epsilon| = 0$.

DEF

Konvence 2.2 Řetězec nebo podřetězec, který sestává právě z k výskytů symbolu a budeme symbolicky značit a^k . Např.

$$a^3 = aaa, \quad b^2 = bb, \quad a^0 = \epsilon.$$

!

Definice 2.7 Necht Σ je abeceda. Označme symbolem Σ^* množinu všech řetězců nad abecedou Σ včetně řetězce prázdného, symbolem Σ^+ množinu všech řetězců nad Σ vyjma řetězce prázdného, tj. $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$. Množinu L , pro níž platí $L \subseteq \Sigma^*$ (případně $L \subseteq \Sigma^+$, pokud $\epsilon \notin L$), nazýváme *jazykem* L nad abecedou Σ . Jazykem tedy může být libovolná podmnožina řetězců nad danou abecedou. Řetězec x , $x \in L$, nazýváme *větou* (také někdy slovem) jazyka L .

DEF

Příklad 2.6 Necht Σ je abeceda. Potenční množina 2^{Σ^*} reprezentuje všechny možné jazyky nad abecedou Σ .

Necht jazyky L_1 a L_2 jsou definovány nad binární abecedou, tj. $L_1, L_2 \subseteq \{0, 1\}^*$, takto:

 $x + y$

$$\begin{aligned} L_1 &= \{0^n 1^n \mid n \geq 0\} \text{ tj.} \\ L_1 &= \{\epsilon, 01, 0011, 000111, \dots\} \\ L_2 &= \{xx^R \mid x \in \{0, 1\}^+\} \text{ tj.} \\ L_2 &= \{00, 11, 0000, 0110, 1001, 1111, \dots\} \end{aligned}$$

Příklad 2.7 Uvažujeme abecedu programovacího jazyka Pascal. Jazyk Pascal je nekonečná množina řetězců (programů) nad jeho abecedou, které lze odvodit z grafů syntaxe jazyka Pascal (viz. [6]). Např. řetězec

 $x + y$

`program P; begin end.`

je větou jazyka Pascal (přestože nepopisuje žádnou akci), kdežto řetězec

`procedure S; begin a := a mod b end`

nepatří do jazyka Pascal, protože reprezentuje pouze úsek možného programu (podřetězec nějaké věty).

Povšimněme si nyní operací, které lze definovat nad jazyky. Tyto operace lze rozdělit do dvou skupin. Jednu skupinu představují obvyklé množinové operace plynoucí ze skutečnosti, že jazyk je množina. Má tedy smysl mluvit o *sjednocení*, *průniku*, *rozdílu* a *komplementu* jazyků. Je-li např. L_1 jazyk nad abecedou Σ , pak jeho komplement (doplňek) L_2 je jazyk, jenž je dán rozdílem $L_2 = \Sigma^* \setminus L_1$.

Druhá skupina operací respektuje specifikum množin tvořících jazyky, tj. skutečnost, že jejich prvky jsou řetězce. Z této skupiny definujeme operaci součinu a iterace jazyků.

Definice 2.8 Necht L_1 je jazyk nad abecedou Σ_1 , L_2 jazyk nad abecedou Σ_2 . *Součinem* (konkatenací) jazyků L_1 a L_2 je jazyk $L_1 \cdot L_2$ nad abecedou $\Sigma_1 \cup \Sigma_2$, jenž je definován takto:

DEF

$$L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$$

Operace součin jazyků je definována prostřednictvím konkatenace řetězců a má stejné vlastnosti jako konkatenace řetězců – je asociativní a nekomutativní.

Příklad 2.8 Necht:

 $x + y$

$$\begin{aligned} P &= \{A, B, \dots, Z, a, b, \dots, z\}, \\ C &= \{0, 1, \dots, 9\} \text{ jsou abecedy} \\ L_1 &= P, \\ L_2 &= (P \cup C)^* \text{ jsou jazyky} \end{aligned}$$

Součinem jazyků $L_1 \cdot L_2$ je jazyk, který obsahuje všechny identifikátory, tak jak jsou obvykle definovány v programovacích jazycích.

Prostřednictvím součinu jazyka se sebou samým (mocniny jazyka) můžeme definovat důležitou operaci nad jazykem – iteraci jazyka.

Definice 2.9 Nechť L je jazyk nad abecedou Σ . *Iteraci* L^* jazyka L a *pozitivní iteraci* L^+ jazyka L definujeme takto:

$$L^0 = \{\epsilon\} \quad (1)$$

$$L^n = L \cdot L^{n-1} \text{ pro } n \geq 1 \quad (2)$$

$$L^* = \bigcup_{n \geq 0} L^n \quad (3)$$

$$L^+ = \bigcup_{n \geq 1} L^n \quad (4)$$

Věta 2.1 Je-li L jazyk, pak platí:

$$L^* = L^+ \cup \{\epsilon\} \quad \text{a} \quad (1)$$

$$L^+ = L \cdot L^* = L^* \cdot L \quad (2)$$

Důkaz: Tvrzení 1 plyne z definice iterace a pozitivní iterace:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots, \quad L^+ = L^1 \cup L^2 \cup \dots$$

$$L^* = L^0 \cup L^+ = L^+ \cup L^0 = L^+ \cup \{\epsilon\}$$

Tvrzení 2 dostaneme provedením součinu $L \cdot L^*$ resp. $L^* \cdot L$:

$$L \cdot L^* = L \cdot (L^0 \cup L^1 \cup L^2 \cup \dots) = L^1 \cup L^2 \cup L^3 \cup \dots = L^+,$$

s použitím distributivního zákona $L_1(L_2 \cup L_3) = L_1 L_2 \cup L_1 L_3$. Podobně, s využitím ekvivalentní definice mocniny $L^n = L^{n-1} L, n \geq 0$ obdržíme $L^* \cdot L = L^+$. \square

Příklad 2.9 Je-li $L_1 = \{(p)\}, L_2 = \{,p\}, L_3 = \{\}$ potom jazyk $L_1 \cdot L_2^* \cdot L_3$ obsahuje řetězce:

$$(p) \quad (p,p) \quad (p,p,p) \quad (p,p,p,p) \dots$$

Definice 2.10 Algebraická struktura $\langle A, +, \cdot, 0, 1 \rangle$ se nazývá polokruh s aditivním jednotkovým prvkem 0 a multiplikativním jednotkovým prvkem 1, jestliže

(1) $\langle A, +, 0 \rangle$ je komutativní monoid

(2) $\langle A, \cdot, 1 \rangle$ je monoid

(3) pro operaci \cdot platí distributivní zákon vzhledem k $+$:

$$a \cdot (b + c) = a \cdot b + a \cdot c \text{ pro } a, b, c \in A.$$

Věta 2.2 Algebra jazyků $\langle 2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\epsilon\} \rangle$, kde \cup je sjednocení a \cdot konkatenace jazyků tvoří polokruh.

DEF

 $x + y$

DEF

Důkaz:

1. $\langle 2^{\Sigma^*}, \cup, \emptyset \rangle$ je komutativní monoid (\cup je komutativní a asociativní operace a $L \cup \emptyset = \emptyset \cup L = L$ pro všechna $L \in 2^{\Sigma^*}$).
2. $\langle 2^{\Sigma^*}, \cdot, \{\epsilon\} \rangle$ je monoid: $L \cdot \{\epsilon\} = \{\epsilon\} \cdot L = L$ pro všechna $L \in 2^{\Sigma^*}$.
3. Pro všechny $L_1, L_2, L_3 \in 2^{\Sigma^*}$:

$$\begin{aligned} L_1(L_2 \cup L_3) &= \{xy \mid (x \in L_1) \wedge (y \in L_2 \vee y \in L_3)\} = \\ &= \{xy \mid (x \in L_1 \wedge y \in L_2) \vee (x \in L_1 \wedge y \in L_3)\} = \\ &= \{xy \mid x \in L_1 \wedge y \in L_2\} \cup \{xy \mid x \in L_1 \wedge y \in L_3\} = \\ &= L_1L_2 \cup L_1L_3. \end{aligned}$$

□

Příklad 2.10 Necht $L_1 = \{xy, z, x\}$, $L_2 = \{a, bc\}$ jsou jazyky. Specifikujte jazyky L_1L_2 , L_2^2 , L_2^* .

Řešení:

- (1) $L_1L_2 = \{xy, z, x\} \cdot \{a, bc\} = \{xya, xybc, za, zbc, xa, xbc\}$
- (2) $L_2^2 = L_2 \cdot L_2 = \{aa, abc, bca, bcbc\}$
- (3) $L_2^* = \{w \mid w \text{ je prázdný řetězec nebo řetězec nad abecedou } \{a, b, c\} \text{ v němž každému výskytu symbolu } c \text{ bezprostředně předchází výskyt symbolu } b \text{ a za každým výskytem symbolu } b \text{ bezprostředně následuje symbol } c\}.$

x + y

2.2 Gramatika

Pojem gramatika souvisí velmi úzce s problémem reprezentace jazyka. Triviální způsob reprezentace – výčet všech vět jazyka – je nepoužitelný nejen pro jazyky nekonečné, ale prakticky i pro rozsáhlé konečné jazyky. Také obvyklé matematické prostředky (použité v předchozích příkladech) jsou použitelné pouze pro reprezentaci jazyků s velmi jednoduchou strukturou.

Gramatika, jako nejznámější prostředek pro reprezentaci jazyků, splňuje základní požadavek kladený na reprezentaci konečných i nekonečných jazyků, požadavek konečnosti reprezentace. Používá dvou konečných disjunktních abeced:

1. množiny N nonterminálních symbolů
2. množiny Σ terminálních symbolů

Nonterminální symboly, krátce *nonterminály*, mají roli pomocných proměnných označujících určité syntaktické celky – syntaktické kategorie.

Množina *terminálních symbolů*, krátce *terminálů*, je identická s abecedou, nad níž je definován jazyk. Sjednocení obou množin, tj. $N \cup \Sigma$, nazýváme *slovníkem gramatiky*.

Konvence 2.3 Pro zápis terminálních a nonterminálních symbolů a řetězců tvořených těmito symboly budeme používat této konvence:

1. a, b, c, d reprezentují terminální symboly
2. A, B, C, D, S reprezentují nonterminální symboly
3. U, V, \dots, Z reprezentují terminální nebo nonterminální symboly

!

4. $\alpha, \beta, \dots, \omega$ reprezentují řetězce terminálních a nonterminálních symbolů
5. u, v, \dots, z reprezentují řetězce pouze terminálních symbolů
6. řetězec uzavřený v úhlových závorkách \langle, \rangle reprezentuje nonterminální symbol (konvence používaná v BNF).

Příklad 2.11 Slovník gramatiky pro definici jazyka identifikátorů může mít tvar:

$$N = \{\langle \text{identifikátor} \rangle, \langle \text{písmeno} \rangle, \langle \text{číslice} \rangle\}$$

$$\Sigma = \{A, B, \dots, Z, a, b, \dots, z, 0, 1, \dots, 9\}$$

Obsahuje tedy 65 symbolů.

Gramatika představuje generativní systém, ve kterém lze z jistého vyznačeného nonterminálu generovat, aplikací tzv. přepisovacích pravidel, řetězce tvořené pouze terminálními symboly. Takové řetězce reprezentují právě věty gramatikou definovaného jazyka.

Jádrem gramatiky je tak konečná množina P *přepisovacích pravidel* (nazývaných také produkce). Každé přepisovací pravidlo má tvar uspořádané dvojice (α, β) řetězců; stanovuje možnou substituci řetězce β namísto řetězce α , který se vyskytuje jako podřetězec generovaného řetězce. Řetězec α obsahuje alespoň jeden nonterminální symbol, řetězec β je prvek množiny $(N \cup \Sigma)^*$. Formálně vyjádřeno, množina P přepisovacích pravidel je podmnožinou kartézského součinu:

$$P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$$

Příklad 2.12 Uvažujme, že např. dvojice (AB, CDE) je jedním z přepisovacích pravidel gramatiky a předpokládejme, že řetězec $x = FGABH$ byl získán aplikací jiných pravidel gramatiky. Aplikujeme-li nyní na řetězec x pravidlo (AB, CDE) , obdržíme řetězec $y = FGCDEH$ (nahrazením podřetězce AB řetězce x řetězcem CDE). Říkáme, že jsme řetězec y odvodili (derivovali) z řetězce x podle přepisovacího pravidla (AB, CDE) .

Přistoupíme nyní k úplné definici gramatiky a formální definici pojmů, jejichž prostřednictvím lze definovat jazyk generovaný gramatikou.

Definice 2.11 Gramatika G je čtveřice $G = (N, \Sigma, P, S)$, kde

- N je konečná množina nonterminálních symbolů
- Σ je konečná množina terminálních symbolů, $N \cap \Sigma = \emptyset$
- P je konečná podmnožina kartézského součinu $(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$,
- $S \in N$ je výchozí (také počáteční) symbol gramatiky

Prvek (α, β) množiny P nazýváme *přepisovacím pravidlem* (krátce *pravidlem*) a budeme jej zapisovat ve tvaru $\alpha \rightarrow \beta$. Řetězec α resp. β nazýváme *levou* resp. *pravou stranou* přepisovacího pravidla.

Příklad 2.13

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

$$P = \{S \rightarrow 0A1, 0A \rightarrow 00A1, A \rightarrow \epsilon\}$$

Příklad 2.14 Gramatika definující jazyk identifikátorů může mít tvar:

$$\begin{aligned}
 G &= (N, \Sigma, P, S), \text{ kde} \\
 N &= \{\langle \text{identifikátor} \rangle, \langle \text{písmeno} \rangle, \langle \text{číslice} \rangle\} \\
 \Sigma &= \{A, B, \dots, Z, a, b, \dots, z, 0, 1, \dots, 9\} \\
 P &= \{\langle \text{identifikátor} \rangle \rightarrow \langle \text{písmeno} \rangle, \\
 &\quad \langle \text{identifikátor} \rangle \rightarrow \langle \text{identifikátor} \rangle \langle \text{písmeno} \rangle, \\
 &\quad \langle \text{identifikátor} \rangle \rightarrow \langle \text{identifikátor} \rangle \langle \text{číslice} \rangle, \\
 &\quad \langle \text{písmeno} \rangle \rightarrow A, \\
 &\quad \vdots \\
 &\quad \langle \text{písmeno} \rangle \rightarrow Z, \\
 &\quad \langle \text{číslice} \rangle \rightarrow 0, \\
 &\quad \vdots \\
 &\quad \langle \text{číslice} \rangle \rightarrow 9\} \\
 S &= \langle \text{identifikátor} \rangle
 \end{aligned}$$

Množina P obsahuje 65 přepisovacích pravidel.

Konvence 2.4 Obsahuje-li množina pravidel P přepisovací pravidla tvaru $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$, pak pro zkrácení lze použít zápisu $\alpha \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$.

Definice 2.12 Necht $G = (N, \Sigma, P, S)$ je gramatika a necht λ a μ jsou řetězce z $(N \cup \Sigma)^*$. Mezi řetězci λ a μ platí binární relace \Rightarrow_G , nazývaná *přímá derivace*, jestliže můžeme řetězce λ a μ vyjádřit ve tvaru

$$\begin{aligned}
 \lambda &= \gamma \alpha \delta \\
 \mu &= \gamma \beta \delta
 \end{aligned}$$

kde γ a δ jsou libovolné řetězce z $(N \cup \Sigma)^*$ a $\alpha \rightarrow \beta$ je nějaké přepisovací pravidlo z P .

Platí-li mezi řetězci λ a μ relace přímé derivace, pak píšeme $\lambda \Rightarrow_G \mu$ a říkáme, že řetězec μ lze přímo generovat z řetězce λ v gramatice G . Je-li z kontextu zřejmé, že jde o derivaci v gramatice G , pak nemusíme specifikaci gramatiky pod symbolem \Rightarrow uvádět.

Příklad 2.15 Uvažujme gramatiku z příkladu 2.13 a řetězce $\lambda = 000A111$ a $\mu = 0000A1111$. Položíme-li

$$\begin{aligned}
 \lambda &= \underbrace{00}_{\gamma} \underbrace{0A}_{\alpha} \underbrace{111}_{\delta} \\
 \mu &= \underbrace{00}_{\gamma} \underbrace{00A1}_{\beta} \underbrace{111}_{\delta}
 \end{aligned}$$

vidíme, že platí $000A111 \Rightarrow 0000A1111$, protože $0A \rightarrow 00A1$ je pravidlem v této gramatice.

$x + y$

!

DEF

$x + y$

Příklad 2.16 Je-li $\alpha \rightarrow \beta$ pravidlo v gramatice G , pak v této gramatice platí $\alpha \Rightarrow \beta$, jak plyne z definice 2.12, položíme-li $\gamma = \delta = \epsilon$.

Definice 2.13 Necht $G = (N, \Sigma, P, S)$ je gramatika a λ a μ jsou řetězce z $(N \cup \Sigma)^*$. Mezi řetězci λ a μ platí relace \Rightarrow^+ nazývaná *derivace*, jestliže existuje posloupnost přímých derivací $\nu_{i-1} \Rightarrow \nu_i \quad i = 1, \dots, n, \quad n \geq 1$ taková, že platí:

$$\lambda = \nu_0 \Rightarrow \nu_1 \Rightarrow \dots \Rightarrow \nu_{n-1} \Rightarrow \nu_n = \mu$$

Tuto posloupnost nazýváme *derivací délky n* . Platí-li $\lambda \Rightarrow^+ \mu$, pak říkáme, že řetězec μ lze *generovat* z řetězce λ v gramatice G . Relace \Rightarrow^+ je zřejmě tranzitivním uzávěrem relace přímé derivace \Rightarrow . Symbolem \Rightarrow^n značíme n -tou mocninu relace \Rightarrow .

Příklad 2.17 V gramatice z příkladu 2.13 platí (v důsledku pravidla $0A \rightarrow 00A1$, viz příklad 2.15) relace

$$0^n A 1^n \Rightarrow 0^{n+1} A 1^{n+1} \quad n > 0$$

a tudíž také $0A1 \Rightarrow^+ 0^n A 1^n$ pro libovolné $n > 1$.

Definice 2.14 Jestliže v gramatice G platí pro řetězce λ a μ relace $\lambda \Rightarrow^+ \mu$ nebo identita $\lambda = \mu$, pak píšeme $\lambda \Rightarrow^* \mu$. Relace \Rightarrow^* je tranzitivním a reflexivním uzávěrem relace přímé derivace \Rightarrow .

Příklad 2.18 Relaci $0A1 \Rightarrow^+ 0^n A 1^n, \quad n > 1$ z příkladu 2.17 lze rozšířit na relaci

$$0A1 \Rightarrow^* 0^n A 1^n, \quad n > 0$$

Poznámka 2.1 Dojdeme-li v posloupnosti přímých derivací k řetězci, který obsahuje pouze terminální symboly, pak již nelze aplikovat žádné přepisovací pravidlo a proces generování *končí*. Z této skutečnosti, jež vyplývá z definice pravidla, je odvozen název množiny Σ jako množiny *terminálních* symbolů.

Definice 2.15 Necht $G = (N, \Sigma, P, S)$ je gramatika. Řetězec $\alpha \in (N \cup \Sigma)^*$ nazýváme *větnou formou*, jestliže platí $S \Rightarrow^* \alpha$, tj. řetězec α je generovatelný z výchozího symbolu S . Větná forma, která obsahuje pouze terminální symboly, se nazývá *věta*. Jazyk $L(G)$, generovaný gramatikou G , je definován množinou všech vět

$$L(G) = \{w \mid S \Rightarrow^* w \wedge w \in \Sigma^*\}$$

Příklad 2.19 Jazyk generovaný gramatikou G z příkladu 2.13 je množina

$$L(G) = \{0^n 1^n \mid n > 0\},$$

protože platí

$$\begin{aligned} S &\Rightarrow 0A1 \\ S &\Rightarrow^* 0^n A 1^n \quad n > 0 \quad (\text{viz příklad 2.17 a 2.18}) \\ S &\Rightarrow^* 0^n 1^n \quad n > 0 \quad (\text{po aplikaci pravidla } A \rightarrow \epsilon) \end{aligned}$$

DEF

DEF



DEF

x + y

2.3 Chomského klasifikace gramatik

Chomského klasifikace gramatik (a jazyků), známá také pod názvem Chomského hierarchie jazyků, vymezuje čtyři typy gramatik, podle tvaru přepisovacích pravidel, jež obsahuje množina přepisovacích pravidel P . Tyto typy se označují jako typ 0, typ 1, typ 2 a typ 3.

2.3.1 Typ 0

Gramatika typu 0 obsahuje pravidla v nejobecnějším tvaru, shodným s definicí 2.11 gramatiky:

$$\alpha \rightarrow \beta, \alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*, \beta \in (N \cup \Sigma)^*.$$

Z tohoto důvodu se gramatiky typu 0 nazývají také gramatikami *neomezenými*.

Příklad 2.20 Příklad neomezené gramatiky:

$$\begin{aligned} G &= (\{A, B\}, \{a, b\}, P, A) \text{ s pravidly} \\ A &\rightarrow AbB \mid a \\ AbB &\rightarrow baB \mid BAbB \\ B &\rightarrow b \mid \epsilon \end{aligned}$$

2.3.2 Typ 1

Gramatika typu 1 obsahuje pravidla tvaru:

$$\alpha A \beta \rightarrow \alpha \gamma \beta, \quad A \in N, \quad \alpha, \beta \in (N \cup \Sigma)^*, \quad \gamma \in (N \cup \Sigma)^+$$

nebo

$$S \rightarrow \epsilon, \text{ pokud se } S \text{ neobjevuje na pravé straně žádného pravidla.}$$

Gramatiky typu 1 se nazývají také *gramatikami kontextovými*, poněvadž tvar pravidla této gramatiky implikuje, že nonterminál A může být nahrazen řetězcem γ pouze tehdy, je-li jeho pravým kontextem řetězec β a levým kontextem řetězec α .

Kontextové gramatiky neobsahují pravidla tvaru $\alpha A \beta \rightarrow \alpha \beta$, tj. nepřipouštějí, aby byl nonterminál nahrazen prázdným řetězcem. Jedinou přípustnou výjimkou je pravidlo $S \rightarrow \epsilon$, a to jen za předpokladu, že se S , tj. výchozí symbol gramatiky, neobjevuje na pravé straně žádného pravidla. Případné použití pravidla $S \rightarrow \epsilon$ umožňuje popsat příslušnost prázdného řetězce k jazyku, který je danou gramatikou generován. V důsledku této vlastnosti pravidel gramatiky typu 1 nemůže při generování věty dojít ke zkrácení generovaných řetězců, tj. platí-li $\lambda \Rightarrow \mu$, pak $|\lambda| \leq |\mu|$ (s výjimkou $S \Rightarrow \epsilon$).

Příklad 2.21 Příklad kontextové gramatiky:

$$\begin{aligned} G &= (\{A, S\}, \{0, 1\}, P, S) \text{ s pravidly} \\ S &\rightarrow 0A \mid \epsilon \\ 0A &\rightarrow 00A1 \quad (\alpha = 0, \beta = \epsilon, \gamma = 0A1) \\ A &\rightarrow 1 \end{aligned}$$

Dodejme, že v literatuře je možné se setkat s alternativní definicí gramatik typu 1, která definuje stejnou třídu jazyků. V této definici se připouští pouze pravidla tvaru $\alpha \rightarrow \beta$, kde $|\alpha| \leq |\beta|$, nebo $S \rightarrow \epsilon$, pokud se S neobjevuje na pravé straně žádného pravidla.

2.3.3 Typ 2

Gramatika typu 2 obsahuje pravidla tvaru:

$$A \rightarrow \gamma, \quad A \in N, \quad \gamma \in (N \cup \Sigma)^*.$$

Gramatiky typu 2 se nazývají také *bezkontextovými gramatikami*, protože substituci pravé strany γ pravidla za nonterminál A lze provádět bez ohledu na kontext, ve kterém je nonterminál A uložen. Na rozdíl od kontextových gramatik, bezkontextové gramatiky smí obsahovat pravidla tvaru $A \rightarrow \epsilon$. V kapitole 4 však ukážeme, že každou bezkontextovou gramatiku lze transformovat, aniž by se změnil jazyk generovaný touto gramatikou tak, že obsahuje nejvýše jedno pravidlo s prázdným řetězcem na pravé straně tvaru $S \rightarrow \epsilon$. V takovém případě, stejně jako v případě kontextových gramatik, nesmí se výchozí symbol S objevit v žádné pravé straně prepisovacího pravidla gramatiky.

Příklad 2.22 Příklad bezkontextové gramatiky:

$$\begin{aligned} G &= (\{S\}, \{0, 1\}, P, S) \text{ s pravidly} \\ S &\rightarrow 0S1 \mid \epsilon \end{aligned}$$

Poznamenejme, že jazyk generovaný touto gramatikou je stejný jako jazyk generovaný kontextovou gramatikou z příkladu 2.21:

$$L(G) = \{0^n 1^n\}, \quad n \geq 0$$

2.3.4 Typ 3

Gramatika typu 3 obsahuje pravidla tvaru:

$$A \rightarrow xB \quad \text{nebo} \quad A \rightarrow x; \quad A, B \in N, \quad x \in \Sigma^*.$$

Gramatika s tímto tvarem pravidel se nazývá *pravá lineární gramatika* (jediný možný nonterminál pravé strany pravidla stojí úplně napravo). V následující kapitole ukážeme, že k uvedené gramatice lze sestavit ekvivalentní speciální pravou lineární gramatiku s pravidly tvaru

$$A \rightarrow aB \quad \text{nebo} \quad A \rightarrow a, \quad \text{kde} \quad A, B \in N, \quad a \in \Sigma$$

nebo

$$S \rightarrow \epsilon, \quad \text{pokud se } S \text{ neobjevuje na pravé straně žádného pravidla.}$$

Tuto gramatiku budeme nazývat *regulární gramatikou*, přesněji *pravou regulární gramatikou*.

Příklad 2.23 Příklady gramatiky typu 3:

$$\begin{aligned} G &= (\{A, B\}, \{a, b, c\}, P, A) \text{ s pravidly} \\ A &\rightarrow aaB \mid ccB \\ B &\rightarrow bB \mid \epsilon \end{aligned}$$

Definice 2.16 Jazyk generovaný gramatikou typu i , $i = 0, 1, 2, 3$, nazýváme jazykem typu i . Podle názvů gramatik mluvíme také o jazycích *rekurzivně vyčíslitelných* ($i = 0$) a analogicky ke gramatikám jazycích *kontextových* ($i = 1$), *bezkontextových* ($i = 2$) a *regulárních* ($i = 3$)¹.

DEF

Věta 2.3 Necht \mathcal{L}_i , $i = 0, 1, 2, 3$ značí třídu všech jazyků typu i . Pak platí $\mathcal{L}_0 \supseteq \mathcal{L}_1 \supseteq \mathcal{L}_2 \supseteq \mathcal{L}_3$.

Důkaz: Z definice gramatiky typu i plyne, že každá gramatika typu 1 je zároveň gramatikou typu 0 a každá gramatika typu 3 je zároveň bezkontextovou gramatikou. Inkluze $\mathcal{L}_1 \supseteq \mathcal{L}_2$ plyne ze skutečnosti, že každá bezkontextová gramatika může být převedena na bezkontextovou gramatiku, jež neobsahuje, s výjimkou pravidla $S \rightarrow \epsilon$ (S je výchozí symbol), žádné pravidlo s pravou stranou totožnou s prázdným řetězcem ϵ . \square

K základním poznatkům teorie formálních jazyků patří další tvrzení, jež je zesílením věty 2.3 a jež definuje Chomského hierarchii formálních jazyků. Toto tvrzení uvádíme nyní bez důkazu.

Věta 2.4 Necht \mathcal{L}_i , $i = 0, 1, 2, 3$ jsou třídy jazyků typu i . Pak platí $\mathcal{L}_0 \supset \mathcal{L}_1 \supset \mathcal{L}_2 \supset \mathcal{L}_3$.

Formální jazyk je univerzálním prostředkem pro kódování informace a popis objektů řady aplikací, které lze reprezentovat a zpracovávat počítačovými programy. Algebraický charakter operací nad formálními jazyky umožňuje kompozitní řešení složitých problémů jejich rozkladem a skládáním výsledného řešení. Chomského hierarchie gramatik a formálních jazyků je referenčním schématem klasifikace aplikací teorie formálních jazyků a prostředkem vymezujícím popisnou a rozhodovací sílu dané třídy formálních modelů.

 Σ

2.4 Cvičení

Cvičení 2.4.1 Uspořádejte (podle množinové inkluze) tyto jazyky

- (a) $\{a, b\}^*$
- (b) $a^*b^*c^*$
- (c) $\{w \mid w \in \{a, b\}^* \text{ a počet výskytů symbolů } a \text{ a } b \text{ je roven } \}$

?

Cvičení 2.4.2 Na základě Chomského klasifikace gramatik a jazyků rozhodněte typ dané gramatiky. Uvádíme pouze pravidla gramatiky; velká písmena značí nonterminální symboly (S je výchozí symbol) a malá písmena nebo číslice značí terminální symboly.

¹Toto označení vychází z ekvivalence vyjadřovací síly gramatik typu 3 a gramatik regulárních.

- (a) $S \rightarrow bSS \mid a$
- (b) $S \rightarrow 00S1 \mid \epsilon$
- (c) $S \rightarrow bA \quad A \rightarrow eB \quad B \rightarrow gC$
 $C \rightarrow iD \quad D \rightarrow n$
- (d) $S \rightarrow \text{sur } A \quad S \rightarrow \text{in } A \quad S \rightarrow \text{bi } A$
 $S \rightarrow \text{pro } A \quad A \rightarrow \text{jekce}$
- (e) $S \rightarrow SZ_1 \quad Y_1Z_3 \rightarrow YZ_3 \quad ZZ_1 \rightarrow Z_2Z_1$
 $S \rightarrow X_1Z_1 \quad YZ_3 \rightarrow YZ \quad Z_2Z_1 \rightarrow Z_2Z$
 $X_1 \rightarrow aY_1 \quad Y_1Y \rightarrow Y_1Z_2 \quad Z_2Z \rightarrow Z_1Z$
 $X_1 \rightarrow aX_1Y_1 \quad Y_1Y_2 \rightarrow YY_2 \quad Y \rightarrow b$
 $Y_1Z_1 \rightarrow Y_1Z_3 \quad YY_2 \rightarrow YY_1 \quad Z \rightarrow c$
- (f) $S \rightarrow YXY \quad ZY \rightarrow XXY \quad YX \rightarrow YZ$
 $X \rightarrow a \quad ZX \rightarrow XXZ \quad Y \rightarrow \epsilon$

V gramatikách (a)–(f) vytvořte derivace terminálních řetězců a stanovte jazyky $L(G)$, které tyto gramatiky specifikují.

Cvičení 2.4.3 Při popisu syntaxe programovacích jazyků se často používají určité konvence, které zkracují zápisy syntaktických definic. Hranaté závorky označují volitelnost řetězců, např.

$\langle \text{statement} \rangle \rightarrow \text{if } \langle \text{expr} \rangle \text{ then } \langle \text{statement} \rangle \text{ else } [\langle \text{statement} \rangle]$

složené závorky pak iteraci řetězce včetně 0 – násobné iterace jako v příkladu definice složeného příkazu:

$\langle \text{statement} \rangle \rightarrow \text{begin } \langle \text{statement} \rangle \{; \langle \text{statement} \rangle\} \text{ end} .$

Stanovte ekvivalentní zápisy pravidly bezkontextové gramatiky pro

- (1) $A \rightarrow \alpha [\beta] \gamma$
- (2) $A \rightarrow \alpha \{\beta\} \gamma$

Cvičení 2.4.4 Vytvořte gramatiku typu 3, která generuje identifikátory jež mohou mít maximálně 6 znaků a začínají písmenem I, J, K, L, M nebo N (celočíslné proměnné v jazyce Fortran).

Cvičení 2.4.5 Vytvořte gramatiku typu 3, která generuje čísla jazyka Pascal.

Cvičení 2.4.6 Vytvořte bezkontextovou gramatiku, která generuje všechny řetězce nul a jedniček takové, že počet nul je v každém řetězci shodný s počtem jedniček.

Cvičení 2.4.7 Ukažte, že neomezená gramatika $G = (\{S, B, C\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla $S \rightarrow SaBC, Ca \rightarrow aC, S \rightarrow aBC, BC \rightarrow CB, aB \rightarrow Ba, CB \rightarrow BC, Ba \rightarrow aB, B \rightarrow b, aC \rightarrow Ca, C \rightarrow c$ generuje věty ve kterých je počet výskytů symbolů a, b, c navzájem roven.

Cvičení 2.4.8 Necht je N množina neterminálních symbolů a Σ množina terminálů. Ukažte, že pravidla tvaru $A \rightarrow r$, kde $A \in N$ a r je regulární výraz nad abecedou $(N \cup \Sigma)$ lze převést na pravidla bezkontextové gramatiky.

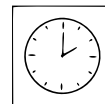
Podle navrženého postupu převedte na ekvivalentní pravidla bezkontextové gramatiky pravidlo $A \rightarrow a (C + D)$, kde symboly $*$, $+$, $($, $)$ jsou metasymbole regulárního výrazu.

Cvičení 2.4.9 Sestrojte gramatiky, které generují tyto jazyky:

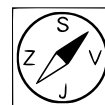
- (a) $L = \{ww^R \mid w \in \{a, b\}^*\}$
- (b) $L = \{wcw^R \mid w \in \{a, b\}^+\}$
- (c) jazyk aritmetických výrazů s operátory $+$, $*$, $/$ a $-$ ve významu unárního i binárního operátoru.
- (d) jazyk identifikátorů (vytvořte gramatiky s levou rekurzí, s pravou rekurzí a s minimálním počtem pravidel).

Kapitola 3

Regulární jazyky



20:00



Cílem této kapitoly je důkladné pochopení alternativních prostředků specifikace regulárních jazyků, důkazových technik, které verifikují jejich ekvivalenci a formálních algoritmů zápisů jejich vzájemných převodů. Vede dále k pochopení základních vlastností regulárních jazyků, jež činí tuto třídu nejvíce aplikovatelnou v řadě technických i netechnických oblasí. Kapitola stanovuje rámce teorie, které jsou využívány i v následujících částech učebního textu.

3.1 Jazyky přijímané konečnými automaty a deterministický konečný automat

3.1.1 Nedeterministický konečný automat

Definice 3.1 Nedeterministický konečný automat (NKA) je 5-tice $M = (Q, \Sigma, \delta, q_0, F)$, kde

- (1) Q je konečná množina stavů,
- (2) Σ je konečná vstupní abeceda,
- (3) δ je zobrazení $Q \times \Sigma \rightarrow 2^Q$, které nazýváme *funkcí přechodu* (2^Q je množina podmnožin množiny Q),
- (4) $q_0 \in Q$ je počáteční stav,
- (5) $F \subseteq Q$ je množina koncových stavů.

Deterministický konečný automat definujeme analogicky jako nedeterministický s tím, že přechodová funkce má tvar $\delta : Q \times \Sigma \rightarrow Q \cup \{nedef\}$, $nedef \notin Q$.

Činnost konečného automatu M je dána posloupností přechodů; přechod z jednoho stavu do druhého je řízen funkcí přechodu δ , která na základě přítomného stavu q_i a právě přečteného symbolu $a \in \Sigma$ vstupního řetězce předepisuje budoucí stav q_j automatu. Je-li M deterministický konečný automat, δ předepisuje vždy jediný budoucí stav $q_j \in Q$, nebo *nedef*, což znamená, že přechod není možný¹; v případě nedeterministického konečného automatu δ předepisuje množinu budoucích stavů Q_j , $Q_j \in 2^Q$.

Definice 3.2 Je-li $M = (Q, \Sigma, \delta, q_0, F)$ konečný automat, pak dvojici $C = (q, w)$ z $Q \times \Sigma^*$ nazýváme *konfigurací* automatu M . Konfigurace tvaru (q_0, w) je *počáteční konfigurace*, konfigurace tvaru (q, ϵ) , $q \in F$ je *koncová konfigurace*. Přechod automatu M je reprezentován binární relací \vdash_M na množině konfigurací C . Pro všechna $q, q' \in Q$ a $w, w' \in \Sigma^*$ definujeme, že platí $(q, w) \vdash_M (q', w')$, tehdy a jen tehdy, když $w = aw'$ pro nějaké $a \in \Sigma$ a $q' \in \delta(q, a)$ (tj. $\delta(q, a) = Q_j$, $Q_j \in 2^Q$, $q' \in Q_j$). Označíme symbolem \vdash_M^k , $k \geq 0$, k -tou mocninu ($C \vdash^0 C'$ právě když $C = C'$),

¹Alternativou je použít *parciální* přechodovou funkci $Q \times \Sigma \rightarrow Q$.

DEF

DEF

δ	z	c	.	e	#
q0	q_8	q_7	q_6	q_4	
q1					
q2		q_2			q_1
q3		q_2			
q4	q_3	q_2			
q5		q_5		q_4	q_1
q6		q_5			
q7		q_7	q_6	q_4	q_1
q8		q_7	q_6	q_4	

Tabulka 3.1: Funkce přechodů automatu M

symbolem \vdash_M^+ tranzitivní uzávěr a symbolem \vdash_M^* tranzitivní a reflexivní uzávěr relace \vdash_M . Bude-li zřejmé, že jde o automat M , pak uvedené relace zapíšeme pouze ve tvaru \vdash , \vdash^k , \vdash^+ , \vdash^* .

Definice 3.3 Říkáme, že vstupní řetězec w je *přijímán* konečným automatem M , jestliže $(q_0, w) \vdash^* (q, \epsilon)$, $q \in F$. Jazyk přijímaný konečným automatem M označujeme symbolem $L(M)$ a definujeme ho jako množinu všech řetězců přijímaných automatem M :

$$L(M) = \{w \mid (q_0, w) \vdash^* (q, \epsilon) \wedge q \in F\}$$

Příklad 3.1 Konečný deterministický automat

$$M = (\{q_0, q_1, q_3, q_4, q_5, q_6, q_7, q_8\}, \{c, z, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot\}, \delta, q_0, \{q_1\}),$$

jehož funkce přechodu δ je definována tabulkou 3.1, přijímá jazyk zápisů čísel, jak ho známe z některých programovacích jazyků. Symbolem c značíme prvek množiny $\{0, 1, \dots, 9\}$, symbolem z prvek množiny $\{+, -\}$; znak $\#$ ukončuje zápis čísla.

Např. vstupnímu řetězci $zc.cezc\#$ (např. číslu $+3.1e-5$) bude odpovídat tato posloupnost konfigurací:

$$\begin{aligned}
(q_0, zc.cezc\#) &\vdash (q_8, c.cezc\#) \\
&\vdash (q_7, .cezc\#) \\
&\vdash (q_6, cezc\#) \\
&\vdash (q_5, ezc\#) \\
&\vdash (q_4, zc\#) \\
&\vdash (q_3, c\#) \\
&\vdash (q_2, \#) \\
&\vdash (q_1, \epsilon)
\end{aligned}$$

Poněvadž $(q_0, zc.cezc\#) \vdash^* (q_1, \epsilon)$, patří $zc.cezc\#$ do $L(M)$.

Příklad 3.2 Nedeterministický konečný automat (NKA)

$$M_1 = (\{q_0, q_1, q_2, q_F\}, \{0, 1\}, \delta, q_0, \{q_F\})$$

má přechodovou funkci definovanou takto:

DEF

 $x + y$

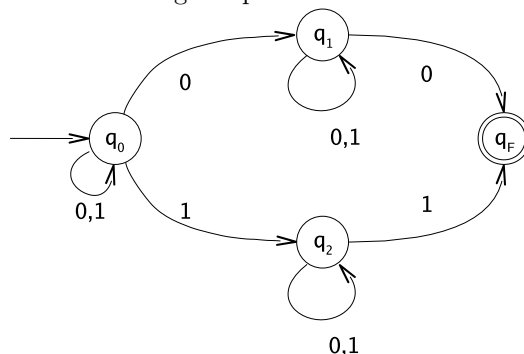
$$\begin{array}{ll} \delta : & \delta(q_0, 0) = \{q_0, q_1\} & \delta(q_0, 1) = \{q_0, q_2\} \\ & \delta(q_1, 0) = \{q_1, q_F\} & \delta(q_1, 1) = \{q_1\} \\ & \delta(q_2, 0) = \{q_2\} & \delta(q_2, 1) = \{q_2, q_F\} \\ & \delta(q_F, 0) = \emptyset & \delta(q_F, 1) = \emptyset \end{array}$$

Alternativními způsoby reprezentace funkce δ může být 1. matice přechodů nebo 2. diagram přechodů.

1. matice (přechodů)

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_1, q_F\}$	$\{q_1\}$
q_2	$\{q_2\}$	$\{q_2, q_F\}$
q_F	\emptyset	\emptyset

2. diagram přechodů



Příklad 3.3 Uvažujme NKA M_1 z příkladu 1.5. Platí: $(q_0, 1010) \vdash (q_0, 010) \vdash (q_1, 10) \vdash (q_1, 0) \vdash (q_f, \epsilon)$ a tedy: $(q_0, 1010) \vdash^* (q_f, \epsilon)$

Neplatí například $(q_0, \epsilon) \vdash^* (q_f, \epsilon)$

Vyjádření jazyka $L(M_1)$: $L(M_1) = \{w \mid w \in \{0, 1\}^* \wedge w \text{ končí symbolem, který je již v řetězci } w \text{ obsažen}\}$.

Vztah mezi nedeterministickými a deterministickými konečnými automaty patří k základním poznatkům teorie formálních jazyků.

Věta 3.1 Každý *nedeterministický* konečný automat M lze převést na *deterministický* konečný automat M' tak, že $L(M) = L(M')$.

Důkaz:

1. Nejprve nalezneme algoritmus převodu $M \rightarrow M'$:

Algoritmus 3.1 Převod nedeterministického konečného automatu na ekvivalentní deterministický konečný automat.

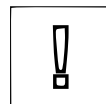
Vstup: Nedeterministický konečný automat $M = (Q, \Sigma, \delta, q_0, F)$

Výstup: Deterministický konečný automat $M' = (Q', \Sigma, \delta', q'_0, F')$

Metoda:

1. Polož $Q' = (2^Q \setminus \{\emptyset\}) \cup \{nedef\}$
2. Polož $q'_0 = \{q_0\}$
3. Pro všechna $S \in 2^Q \setminus \{\emptyset\}$ a pro všechna $a \in \Sigma$ polož: $\delta'(S, a) = \bigcup_{q \in S} \delta(q, a)$ Je-li $\delta'(S, a) = \emptyset$, polož $\delta'(S, a) = nedef$.
4. Polož $F' = \{S \mid S \in 2^Q \wedge S \cap F \neq \emptyset\}$

2. Nyní je třeba ukázat, že $L(M) = L(M')$ tj. že platí:



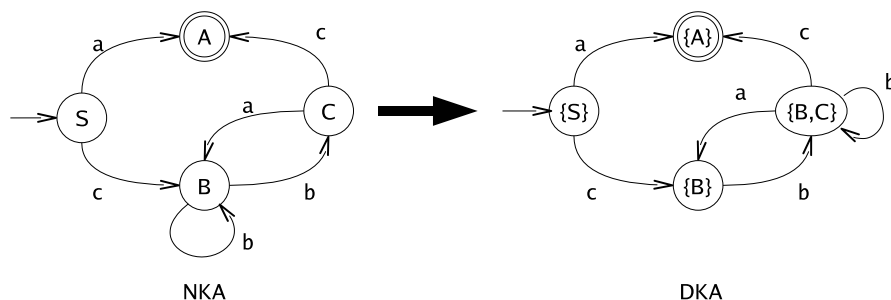
- (a) $L(M) \subseteq L(M')$ a současně,
- (b) $L(M') \subseteq L(M)$.

Tuto část důkazu ponecháme jako cvičení. □

Příklad 3.4 Uvažujme nedeterministický konečný automat $M_2 = (\{S, A, B, C\}, \{a, b, c\}, \delta, S, \{A\})$, $\delta : \delta(S, a) = \{A\}, \delta(S, c) = \{B\}, \delta(B, b) = \{B, C\}, \delta(C, a) = \{B\}, \delta(C, c) = \{A\}$.

K nalezení funkce δ' příslušného DKA aplikujeme zkrácený postup, využívající skutečnosti, že řada stavů z 2^Q může být nedostupných:

1. Počáteční stav: $\{S\}$
2. $\delta'(\{S\}, a) = \{A\}$ — koncový stav
 $\delta'(\{S\}, c) = \{B\}$
3. $\delta'(\{B\}, b) = \{B, C\}$
4. $\delta'(\{B, C\}, a) = \delta(B, a) \cup \delta(C, a) = \{B\}$
 $\delta'(\{B, C\}, b) = \{B, C\}$ $\delta'(\{B, C\}, c) = \{A\}$



3.1.2 Lineární a regulární gramatiky

Dříve, než přistoupíme k důkazu ekvivalence třídy \mathcal{L}_3 a třídy jazyků přijímaných konečnými automaty, ukážeme, že každý jazyk typu 3 může být generován speciálním typem gramatiky, než je pravá lineární gramatika.

Definice 3.4

- Gramatika $G = (N, \Sigma, P, S)$ s pravidly tvaru:

$$\begin{aligned} A &\rightarrow xB & A, B \in N, x \in \Sigma^* \text{ nebo} \\ A &\rightarrow x & x \in \Sigma^*, \end{aligned}$$

resp. tvaru:

$$\begin{aligned} A &\rightarrow Bx & A, B \in N \text{ nebo} \\ A &\rightarrow x & x \in \Sigma^* \end{aligned}$$

se nazývá *pravá lineární*, resp. *levá lineární*, gramatika.

- Gramatika $G = (N, \Sigma, P, S)$ s pravidly tvaru:

DEF

$$\begin{array}{ll} A \rightarrow aB & A, B \in N, a \in \Sigma, \\ A \rightarrow a & a \in \Sigma, \\ \text{případně } S \rightarrow \epsilon & \text{pokud se } S \text{ neobjevuje na pravé straně žádného pravidla} \end{array}$$

resp. s pravidly tvaru:

$$\begin{array}{ll} A \rightarrow Ba & A, B \in N, a \in \Sigma, \\ A \rightarrow a & a \in \Sigma, \\ \text{případně } S \rightarrow \epsilon & \text{pokud se } S \text{ neobjevuje na pravé straně žádného pravidla} \end{array}$$

se nazývá *pravá regulární*, resp. *levá regulární*, gramatika. Gramatiky pravé regulární i levé regulární můžeme též označit souhrnně jako *regulární*.

Poznámka 3.1 Gramatika $G = (N, \Sigma, P, S)$ s pravidly tvaru $A \rightarrow xBy$ nebo $A \rightarrow x$, kde $A, B \in N$ a $x, y \in \Sigma^*$ se nazývá *lineární gramatika*.

Označme:

- \mathcal{L}_{PL} – všechny jazyky generované pravými lineárními gramatikami,
- \mathcal{L}_{LL} – všechny jazyky generované levými lineárními gramatikami,
- \mathcal{L}_L – všechny jazyky generované lineárními gramatikami.

Platí: $\mathcal{L}_{PL} = \mathcal{L}_{LL}$ a $\mathcal{L}_{PL} \subset \mathcal{L}_L$.

Věta 3.2 Každá pravá lineární gramatika $G = (N, \Sigma, P, S)$, tj. gramatika obsahující pouze pravidla typu $A \rightarrow xB$ nebo $A \rightarrow x$, kde $A, B \in N$, $x \in \Sigma^*$, může být transformována na gramatiku $G' = (N', \Sigma, P', S')$, která obsahuje pouze pravidla tvaru $A \rightarrow aB$ nebo $A \rightarrow \epsilon$, přičemž $L(G) = L(G')$.

Důkaz: Množinu přepisovacích pravidel P' gramatiky G' konstruujeme takto:

- (1) všechna pravidla z P tvaru $A \rightarrow aB$ a $A \rightarrow \epsilon$ kde $A, B \in N$, $a \in \Sigma$ zařadíme do množiny P'
- (2) každé pravidlo tvaru $A \rightarrow a_1a_2 \dots a_nB$; $A, B \in N$, $a_i \in \Sigma$, $n \geq 2$ z množiny P nahradíme v množině P' soustavou pravidel:

$$\begin{array}{ll} A & \rightarrow a_1A_1 \\ A_1 & \rightarrow a_2A_2 \\ & \vdots \\ A_{n-1} & \rightarrow a_nB \end{array}$$

Nově zavedené nonterminály A_1, \dots, A_{n-1} přidáme k množině N' . Derivaci $A \xRightarrow{G} a_1 \dots a_nB$ zřejmě odpovídá právě derivace $A \xRightarrow{G'}^n a_1a_2 \dots a_nB$.

- (3) Každé pravidlo tvaru $A \rightarrow a_1 \dots a_n$, $a_i \in \Sigma$, $n \geq 2$ z množiny P nahradíme v množině P' soustavou pravidel:

$$\begin{array}{ll} A & \rightarrow a_1A'_1 \\ A'_1 & \rightarrow a_2A'_2 \\ & \vdots \\ A'_{n-1} & \rightarrow a_nA'_n \\ A'_n & \rightarrow \epsilon \end{array}$$

Derivaci $A \xRightarrow{G} a_1 a_2 \dots a_n$ zřejmě odpovídá právě derivace $A \xRightarrow{G'}^{n+1} a_1 a_2 \dots a_n$.

(4) zbývající, tzv. jednoduchá pravidla tvaru $A \rightarrow B$, $A, B \in N$, nahradíme takto:

- (a) Pro každé $A \in N$ sestrojíme množinu $N_A = \{B \mid A \Rightarrow^* B\}$.
- (b) Množinu pravidel P' rozšíříme takto: Jestliže $B \rightarrow \alpha$ je v P a není jednoduchým pravidlem, pak pro všechna A , pro něž $B \in N_A$, přidáme k P' pravidla $A \rightarrow \alpha$.

Bod 4b aplikuje obecný algoritmus odstranění jednoduchých pravidel bezkontextové gramatiky, který je uveden a dokázán v kapitole 4 (algoritmus 4.5). Jeho součástí je také výpočet množiny N_A .

□

Příklad 3.5 Na základě předchozí věty budeme transformovat pravou lineární gramatiku $G = (\{X, Y\}, \{a, b, c\}, P, X)$, kde P obsahuje pravidla:

$$\begin{aligned} X &\rightarrow abc \mid Y \mid \epsilon \\ Y &\rightarrow aY \mid cbX \end{aligned}$$

Podle 1 budou v P' pravidla:

$$X \rightarrow \epsilon, Y \rightarrow aY$$

Podle 2 nahradíme pravidlo $Y \rightarrow cbX$ pravidly

$$Y \rightarrow cZ, Z \rightarrow bX$$

Podle 3 nahradíme pravidlo $X \rightarrow abc$ pravidly

$$X \rightarrow aU, U \rightarrow bV, V \rightarrow cW, W \rightarrow \epsilon$$

Podle 4 nahradíme pravidlo $X \rightarrow Y$. Protože $N_X = \{X, Y\}$ přidáme k P' pravidla

$$X \rightarrow aY, X \rightarrow cZ$$

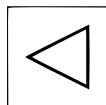
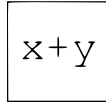
Výsledná gramatika má pak tvar $G' = (\{X, Y, Z, U, V, W\}, \{a, b, c\}, P', X)$ kde P' obsahuje pravidla:

$$\begin{aligned} X &\rightarrow \epsilon \mid aY \mid cZ \mid aU \\ Y &\rightarrow aY \mid cZ \\ Z &\rightarrow bX \\ U &\rightarrow bV \\ V &\rightarrow cW \\ W &\rightarrow \epsilon \end{aligned}$$

Věta 3.3 Každý jazyk typu 3 lze generovat pravou regulární gramatikou.

Důkaz: Pravou regulární gramatiku získáme z gramatiky zkonstruované podle věty 3.2 odstraněním pravidel s prázdným řetězcem na pravé straně. Systematicky tento postup popisuje algoritmus 4.4 v kapitole 4 (níže jej ilustrujeme na příkladě).

□



Příklad 3.6 Gramatiku z příkladu 3.5 převedeme na pravou regulární odstraněním pravidla $W \rightarrow \epsilon$ (vznikne pravidlo $V \rightarrow c$) a pravidla $X \rightarrow \epsilon$ (vznikne pravidlo $Z \rightarrow b$). Protože $\epsilon \in L(G)$ a X je na pravé straně pravidla $Z \rightarrow bX$, musíme zavést nový výchozí symbol X' a odstranit jednoduché pravidlo $X' \rightarrow X$. Výsledná gramatika bude mít tvar:

$$G'' = (\{X', X, Y, Z, U, V\}, \{a, b, c\}, P'', X'),$$

P'' obsahuje pravidla:

$$\begin{aligned} X' &\rightarrow \epsilon \mid aY \mid cZ \mid aU \\ Y &\rightarrow aY \mid cZ \\ Z &\rightarrow bX \mid b \\ X &\rightarrow aY \mid cZ \mid aU \\ U &\rightarrow bV \\ V &\rightarrow c \end{aligned}$$

Věta 3.4 Každý jazyk typu 3 může být generován levou lineární gramatikou.

Důkaz: Úplný důkaz ponecháme na cvičení. Lze postupovat tak, že nejprve z definice regulární množiny dokážeme tvrzení: je-li L regulární množina, pak L^R je také regulární množina. Dále ukážeme: je-li $G = (N, \Sigma, P, S)$ pravá lineární gramatika, pak levá lineární gramatika G' , jejíž množina pravidel má tvar $P' = \{A \rightarrow \alpha^R \mid A \rightarrow \alpha \text{ je v } P\}$, generuje jazyk $(L(G))^R$. \square

Příklad 3.7 Gramatika $G' = (\{X, Y\}, \{a, b, c\}, P', X)$, kde P' obsahuje pravidla

$$\begin{aligned} X &\rightarrow cba \mid Y \mid \epsilon \\ Y &\rightarrow Ya \mid Xbc \end{aligned}$$

je levá lineární gramatika, pro kterou $L(G') = (L(G))^R$ a G je pravá lineární gramatika z příkladu 3.5.

Věta 3.5 Každý jazyk typu 3 lze generovat levou regulární gramatikou.

Důkaz: Konstrukce levé regulární gramatiky k dané levé lineární gramatice je zcela analogická konstrukci pravé regulární gramatiky k dané pravé lineární gramatice, a je proto ponechána na čtenáři (ilustraci poskytuje následující příklad). \square

Příklad 3.8 Uvažujme gramatiku z příkladu 3.7, jež má pravidla:

$$\begin{aligned} X &\rightarrow cba \mid Y \mid \epsilon \\ Y &\rightarrow Ya \mid Xbc \end{aligned}$$

1. krok zavádí pouze pravidla typu $A \rightarrow Ba$ nebo $A \rightarrow \epsilon$:

$$\begin{aligned} X &\rightarrow Ua \mid Ya \mid Zc \mid \epsilon \\ Y &\rightarrow Ya \mid Zc \\ U &\rightarrow Vb \\ V &\rightarrow Wc \\ W &\rightarrow \epsilon \\ Z &\rightarrow Xb \end{aligned}$$

x+y

x+y



x+y

2. krok odstraňuje pravidla typu $A \rightarrow \epsilon$ a upravuje gramatiku na konečný tvar:

$$\begin{aligned} X' &\rightarrow Ua \mid Ya \mid Zc \mid \epsilon \\ Y &\rightarrow Ya \mid Zc \\ U &\rightarrow Vb \\ V &\rightarrow c \\ Z &\rightarrow Xb \mid b \\ X &\rightarrow Ua \mid Ya \mid Zc \end{aligned}$$

Povšimněme si, v čem se liší pravidla této gramatiky od pravidel pravé regulární gramatiky z příkladu 3.6, jež byla získána z pravé lineární gramatiky (příklad 3.5).

3.1.3 Ekvivalence třídy \mathcal{L}_3 a třídy jazyků přijímaných konečnými automaty

Označíme \mathcal{L}_M třídu jazyků přijímaných konečnými automaty a ukážeme, že $\mathcal{L}_3 = \mathcal{L}_M$.

Věta 3.6 Nechť L je jazyk typu 3. Pak existuje konečný automat M takový, že $L = L(M)$, tj. $\mathcal{L}_3 \subseteq \mathcal{L}_M$.

Důkaz: Podle věty 3.2 můžeme jazyk L generovat gramatikou $G = (N, \Sigma, P, S)$ typu 3, jejíž pravidla mají tvar $A \rightarrow aB$ nebo $A \rightarrow \epsilon$ ($A, B \in N, a \in \Sigma$). Sestrojíme konečný (nedeterministický) automat $M = (Q, \Sigma, \delta, q_0, F)$, kde:

- (1) $Q = N$,
- (2) $\Sigma = \Sigma$,
- (3) $\delta : \delta(A, a)$ obsahuje B pro každé pravidlo $A \rightarrow aB$ z P ,
- (4) $q_0 = S$,
- (5) $F = \{A \mid A \rightarrow \epsilon \text{ je pravidlo z } P\}$.

Ukážeme, že $L(G) = L(M)$.

Důkaz provedeme indukcí. Dokazovaná induktivní hypotéza nechť má tvar:

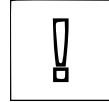
pro každé $A \in N$ platí $A \Rightarrow_G^{i+1} w, w \in \Sigma^*$, právě když $(A, w) \vdash_M^i (C, \epsilon)$
pro nějaké $C \in F$.

Nejdříve dokážeme tuto hypotézu pro $i = 0$. Zřejmě platí

$$A \Rightarrow \epsilon, \text{ právě když } (A, \epsilon) \vdash^0 (A, \epsilon) \text{ pro } A \in F.$$

Nyní předpokládejme, že dokazovaná hypotéza platí pro $i > 0$ a položíme $w = ax, a \in \Sigma, |x| = i - 1$. Pak platí $A \Rightarrow^{i+1} w$, právě když $A \Rightarrow aB \Rightarrow^i x$. Podle definice funkce δ pak $\delta(A, a)$ obsahuje B (v důsledku přímé derivace $A \Rightarrow aB$). Na základě induktivní hypotézy platí $B \Rightarrow^i x$, právě když $(B, x) \vdash^{i-1} (C, \epsilon), C \in F$. Shrňme-li tyto skutečnosti, platí:

$$A \Rightarrow aB \Rightarrow^i ax = w, \text{ právě když } (A, ax) \vdash (B, x) \vdash^{i-1} (C, \epsilon), C \in F$$



tj. $A \Rightarrow^{i+1} w$, právě když $(A, w) \vdash^i (C, \epsilon), C \in F$;

tím jsme platnost induktivního předpokladu dokázali pro všechna $i \geq 0$.

Speciálně pak platí

$$S \Rightarrow^* w, \text{ právě když } (S, w) \vdash^* (C, \epsilon), C \in F$$

a tedy $L(G) = L(M)$. \square

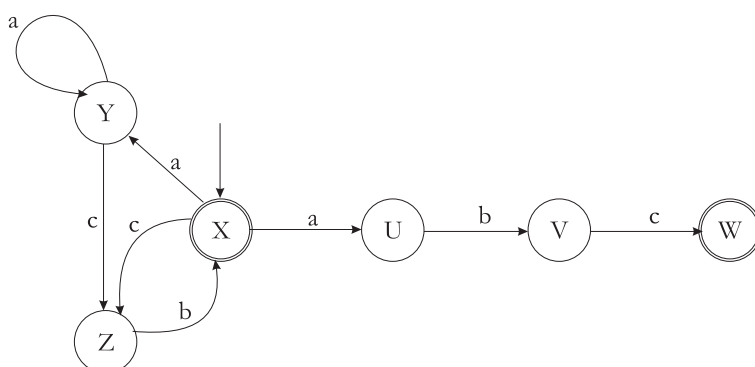
Příklad 3.9 Ke gramatice $G = (\{X, Y, Z, U, V, W\}, \{a, b, c\}, P, X)$, kde P obsahuje pravidla

$$\begin{aligned} X &\rightarrow \epsilon \mid aY \mid cZ \mid aU \\ Y &\rightarrow aY \mid cZ \\ Z &\rightarrow bX \\ U &\rightarrow bV \\ V &\rightarrow cW \\ W &\rightarrow \epsilon \end{aligned}$$

sestrojíme konečný automat, který přijímá jazyk $L(G)$.

Budeme postupovat podle věty 3.6. Funkci přechodů δ reprezentujeme diagramem přechodů (obr. 3.1), v němž koncové stavy jsou vyznačeny dvojitým kroužkem a počáteční stav malou šipkou bez výstupního vrcholu.

$$\begin{aligned} M &= (Q, \Sigma, \delta, q_0, F) \text{ kde} \\ Q &= \{X, Y, Z, U, V, W\} \\ \Sigma &= \{a, b, c\} \\ \delta &: \text{ viz obr. 3.1} \\ q_0 &= X \\ F &= \{X, W\} \end{aligned}$$



Obrázek 3.1: Diagram přechodů automatu M

Věta 3.7 Necht $L = L(M)$ pro nějaký konečný automat M . Pak existuje gramatika G typu 3 taková, že $L = L(G)$, tj. $\mathcal{L}_M \subseteq L_3$.

Důkaz: Nechť $M = (Q, \Sigma, \delta, q_0, F)$. Protože každý nedeterministický automat může být převeden na deterministický automat přijímající stejný jazyk, předpokládejme, že M je deterministický automat. Nechť G je gramatika typu 3, $G = (Q, \Sigma, P, q_0)$, jejíž množina P přepisovacích pravidel je definována takto:

(1) je-li $\delta(q, a) = r$, pak P obsahuje pravidlo $q \rightarrow ar$

(2) je-li $p \in F$, pak P obsahuje pravidlo $p \rightarrow \epsilon$

V další části probíhá důkaz zcela analogicky důkazu věty 3.6. \square

Příklad 3.10 Na základě příkladu 3.1 sestrojíme gramatiku typu 3, která generuje jazyk zápisů čísel. Přechodová funkce deterministického konečného automatu, který přijímá tento jazyk, je v tabulce 3.1. Výsledná gramatika bude tvaru:

$$G = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}, \{c, z, e, ., \#\}, P, q_0)$$

P obsahuje pravidla

$$\begin{aligned} q_0 &\rightarrow zq_8 \mid cq_7 \mid \cdot q_6 \mid e^{q_4} \\ q_1 &\rightarrow \epsilon \\ q_2 &\rightarrow cq_2 \mid \#q_1 \\ q_3 &\rightarrow cq_2 \\ q_4 &\rightarrow zq_3 \mid cq_2 \\ q_5 &\rightarrow cq_5 \mid e^{q_4} \mid \#q_1 \\ q_6 &\rightarrow cq_5 \\ q_7 &\rightarrow cq_7 \mid \cdot q_6 \mid e^{q_4} \mid \#q_1 \\ q_8 &\rightarrow cq_7 \mid \cdot q_6 \mid e^{q_4} \end{aligned}$$

Poznamenejme, že převod této gramatiky na gramatiku regulární je velice snadný, stačí dosadit za q_1 prázdný řetězec a odstranit pravidlo $q_1 \rightarrow \epsilon$.

Věta 3.8 Třída jazyků, jež jsou přijímány konečnými automaty, je totožná s třídou jazyků typu 3 Chomského hierarchie.

Důkaz: Tvrzení bezprostředně plyne z vět 3.6 a 3.7. \square

Algoritmus konstrukce nedeterministického konečného automatu z důkazu věty 3.6 může být snadno modifikován pro převod právě regulární gramatiky na ekvivalentní nedeterministický konečný automat.

Algoritmus 3.2 Konstrukce nedeterministického konečného automatu k právě regulární gramatice.

Vstup: Právě regulární gramatika $G = (N, \Sigma, P, S)$, jejíž pravidla mají tvar $A \rightarrow aB$ a $A \rightarrow a$, kde $A, B \in N$ a $a \in \Sigma$, případně $S \rightarrow \epsilon$ za předpokladu, že S se nevyskytuje na pravé straně žádného pravidla.

Výstup: Nedeterministický konečný automat $M = (N, \Sigma, \delta, q_0, F)$, pro který je $L(M) = L(G)$.

Metoda:

$x + y$

!

a b c

◁

- (1) Položíme $Q = N \cup \{q_F\}$, kde q_F reprezentuje koncový stav.
- (2) Množina vstupních symbolů automatu M je identická s množinou terminálů gramatiky G .
- (3) Funkci přechodů δ definujeme takto:
 - (a) Je-li $A \rightarrow aB$ pravidlo z P , pak $\delta(A, a)$ obsahuje stav B .
 - (b) Je-li $A \rightarrow a$ pravidlo z P , pak $\delta(A, a)$ obsahuje q_F .
- (4) $q_0 = S$
- (5) Je-li $S \rightarrow \epsilon$ pravidlo z P , pak $F = \{S, q_F\}$, v opačném případě $F = \{q_F\}$.

Věta 3.9 Necht G je pravá regulární gramatika a M konečný automat z algoritmu 3.2. Pak $L(M) = L(G)$.

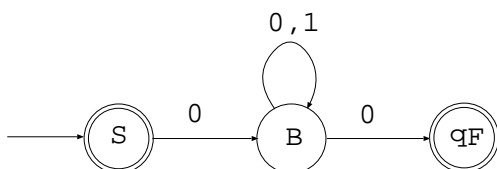
Důkaz: Důkaz této věty je modifikací důkazu věty 3.6; přenecháme ho jako cvičení. \square

Příklad 3.11 Uvažujeme gramatiku $G = (\{S, B\}, \{0, 1\}, P, S)$ s pravidly

$$\begin{aligned} S &\rightarrow 0B \mid \epsilon \\ B &\rightarrow 0B \mid 1B \mid 0 \end{aligned}$$

$$x + y$$

Diagram přechodů automatu přijímajícího jazyk $L(G)$ má, na základě konstrukce podle algoritmu 3.2, tvar:



Obrázek 3.2: Diagram přechodů

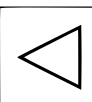
Algoritmus 3.3 Konstrukce nedeterministického konečného automatu k levé regulární gramatice.

Vstup: Levá regulární gramatika $G = (N, \Sigma, P, S)$ jejíž pravidla mají tvar $A \rightarrow Ba$ a $A \rightarrow a$ kde $A, B \in N$ a $a \in \Sigma$, případně $S \rightarrow \epsilon$, je-li $\epsilon \in L(G)$.

Výstup: Nedeterministický konečný automat $M = (N, \Sigma, \delta, q_0, F)$, pro který je $L(M) = L(G)$.

Metoda:

- (1) Položíme $Q = N \cup \{q_0\}$
- (2) Množina vstupních symbolů automatu M je identická s termální abecedou gramatiky G .
- (3) Funkci přechodu δ definujeme takto:



- (a) Je-li $A \rightarrow Ba$ pravidlo z P , pak $\delta(B, a)$ obsahuje stav A .
- (b) Je-li $A \rightarrow a$, pak $\delta(q_0, a)$ obsahuje stav A .
- (4) q_0 je počáteční stav automatu M
- (5) Je-li $S \rightarrow \epsilon$ pravidlo z P pak $F = \{S, q_0\}$, v opačném případě $F = \{S\}$

Věta 3.10 Necht G je levá lineární gramatika a M konečný automat z algoritmu 3.3. Pak $L(M) = L(G)$.

Důkaz: Nejprve dokážeme, že $\epsilon \in L(G)$, právě když $\epsilon \in L(M)$. Podle konstrukce automatu M , $q_0 \in F$ právě když v P je pravidlo $S \rightarrow \epsilon$ a tedy

$$S \Rightarrow \epsilon, \text{ právě když } (q_0, \epsilon) \overset{0}{\vdash} (q_0, \epsilon), q_0 \in F$$

Nyní dokážeme, že pro libovolné $w \in \Sigma^+$ platí

$$S \Rightarrow^+ w, \text{ právě když } (q_0, w) \overset{+}{\vdash} (S, \epsilon), S \in F$$

Položme $w = ax$, $a \in \Sigma$, $x \in \Sigma^*$. Induktivní hypotéza necht má tvar:

$$S \Rightarrow^i Ax \Rightarrow ax, \text{ právě když } (q_0, ax) \vdash^i (A, x) \vdash^i (S, \epsilon), A \in N, S \in F, |x| = i$$

Pro $i = 0$ dostaneme:

$$S \Rightarrow a, \text{ právě když } (q_0, a) \vdash (S, \epsilon),$$

což plyne přímo z definice funkce přechodů automatu M ($\delta(q_0, a)$ obsahuje S , právě když $S \rightarrow a$ je v P).

Pro $i \geq 1$ je třeba dokázat:

- (a) $Ax \Rightarrow ax$, právě když $(q_0, ax) \vdash (A, x)$
- (b) $S \Rightarrow^i Ax$ právě když $(A, x) \vdash^i (S, \epsilon)$

Důkaz tvrzení (a) a (b) ponecháme jako cvičení. □

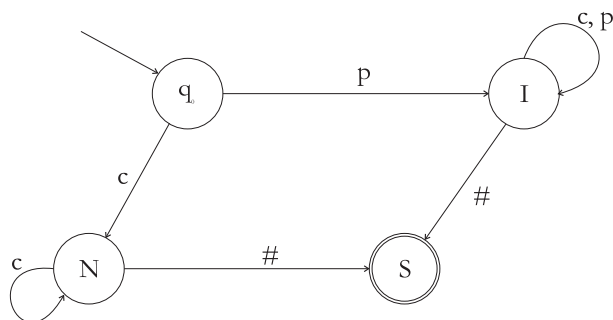
Příklad 3.12 Uvažujme gramatiku $G = (\{S, I, N\}, \{c, p, \#\}, P, S)$ s pravidly:

$$\begin{aligned} S &\rightarrow I\# \mid N\# \\ I &\rightarrow p \mid Ip \mid Ic \\ N &\rightarrow c \mid Nc \end{aligned}$$

$x + y$

Značí-li c arabskou číslici a p písmeno, pak $L(G)$ je jazyk identifikátorů (nonterminál I) a celých čísel bez znaménka (nonterminál N). Každá věta jazyka $L(G)$ končí koncovým znakem $\#$.

Diagram přechodů automatu, který přijímá jazyk $L(G)$, má na základě algoritmu 3.3 tvar:



Obrázek 3.3: Diagram přechodů

3.2 Minimalizace deterministického konečného automatu

Postup převodu deterministického konečného automatu na minimální deterministický konečný automat se skládá z několika kroků, ve kterých eliminujeme nedosažitelné stavy, redukuje nerozlišitelné stavy, převedeme automat na redukovaný deterministický konečný automat a odstraníme přebytečné stavy, které nemají vliv na přijímání vstupního řetězce.

Definice 3.5 Deterministický konečný automat $M = (Q, \Sigma, \delta, q_0, F)$ nazýváme *úplný deterministický konečný automat*, pokud pro všechna $q \in Q$ a všechna $a \in \Sigma$ platí, že $\delta(q, a) \in Q$, tj. δ je totální funkcí na $Q \times \Sigma$.

Definice 3.6 Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Stav $q \in Q$ nazveme *dosažitelný*, pokud existuje $w \in \Sigma^*$ takové, že $(q_0, w) \xrightarrow[M]{*} (q, \varepsilon)$. Stav je *nedosažitelný*, pokud není dosažitelný.

Algoritmus 3.4 Eliminace nedosažitelných stavů

Vstup: Deterministický konečný automat $M = (Q, \Sigma, \delta, q_0, F)$.

Výstup: Deterministický konečný automat M' bez nedosažitelných stavů, $L(M) = L(M')$.

Metoda:

1. $i := 0$
2. $S_i := \{q_0\}$
3. **repeat**
4. $S_{i+1} := S_i \cup \{q \mid \exists p \in S_i \exists a \in \Sigma : \delta(p, a) = q\}$
5. $i := i + 1$
6. **until** $S_i = S_{i-1}$
7. $M' := (S_i, \Sigma, \delta|_{S_i}, q_0, F \cap S_i)$

Základem algoritmu minimalizace deterministického konečného automatu je koncept nerozlišitelných stavů.

DEF

DEF

!

Definice 3.7 Necht $M = (Q, \Sigma, \delta, q_0, F)$ je úplný DKA. Říkáme, že řetězec $w \in \Sigma^*$ rozlišuje stavy $z \in Q$, jestliže $(q_1, w) \stackrel{*}{\vdash}_M (q_3, \varepsilon) \wedge (q_2, w) \stackrel{*}{\vdash}_M (q_4, \varepsilon)$ pro nějaké q_3, q_4 a právě jeden ze stavů q_3, q_4 je v F . Říkáme, že stavy $q_1, q_2 \in Q$ jsou k -nerozlišitelné a píšeme $q_1 \stackrel{k}{\equiv} q_2$, právě když neexistuje $w \in \Sigma^*$, $|w| \leq k$, který rozlišuje q_1 a q_2 . Stavy q_1, q_2 jsou *nerozlišitelné*, značíme $q_1 \equiv q_2$, jsou-li pro každé $k \geq 0$ k -nerozlišitelné.

Dá se snadno dokázat, že \equiv je relací ekvivalence na Q , tj. relací, která je reflexivní, symetrická a tranzitivní.

Definice 3.8 DKA M nazýváme *redukovaný*, jestliže žádný stav Q není nedostupný a žádné dva stavy nejsou nerozlišitelné.

Věta 3.11 Necht $M = (Q, \Sigma, \delta, q_0, F)$ je úplný DKA a $|Q| = n$, $n \geq 2$. Platí $\forall q_1, q_2 \in Q : q_1 \equiv q_2 \Leftrightarrow q_1 \stackrel{n-2}{\equiv} q_2$.

Důkaz: Část \Rightarrow důkazu je triviální, plyne přímo z definice.

Část \Leftarrow :

1. Jestliže $|F| = 0$ nebo $|F| = n$, pak vzájemně platí $q_1 \stackrel{n-2}{\equiv} q_2 \Rightarrow q_1 \equiv q_2$ (všechny stavy jsou koncové, nebo všechny stavy jsou nekoncové).
2. Necht $|F| > 0 \wedge |F| < n$. Ukážeme, že platí $\equiv = \stackrel{n-2}{\equiv} \subseteq \stackrel{n-3}{\equiv} \subseteq \dots \subseteq \stackrel{1}{\equiv} \subseteq \stackrel{0}{\equiv}$:

- Zřejmě platí:

$$(a) \quad \forall q_1, q_2 \in Q : q_1 \stackrel{0}{\equiv} q_2 \Leftrightarrow (q_1 \notin F \wedge q_2 \notin F) \vee (q_1 \in F \wedge q_2 \in F).$$

$$(b) \quad \forall q_1, q_2 \in Q \quad \forall k \geq 1 : q_1 \stackrel{k}{\equiv} q_2 \Leftrightarrow \forall a \in \Sigma : \delta(q_1, a) \stackrel{k-1}{\equiv} \delta(q_2, a).$$

- Relace $\stackrel{0}{\equiv}$ je ekvivalencí určující rozklad $\{F, Q \setminus F\}$.
- Je-li $\stackrel{k+1}{\equiv} \neq \stackrel{k}{\equiv}$, pak $\stackrel{k+1}{\equiv}$ je vlastním zjemněním $\stackrel{k}{\equiv}$, tj. obsahuje alespoň o jednu třídu více než rozklad $\stackrel{k}{\equiv}$.
- Jestliže pro nějaké k platí $\stackrel{k+1}{\equiv} = \stackrel{k}{\equiv}$, pak také $\stackrel{k+1}{\equiv} = \stackrel{k+2}{\equiv} = \stackrel{k+3}{\equiv} = \dots$ podle (b) a tedy $\stackrel{k}{\equiv}$ je hledaná ekvivalence.
- Protože F nebo $Q \setminus F$ obsahuje nejvýše $n - 1$ prvků, získáme relaci \equiv po nejvýše $n - 2$ zjemněních $\stackrel{0}{\equiv}$.

□

Algoritmus 3.5 Převod na redukovaný deterministický konečný automat.

Vstup: Úplně definovaný DKA $M = (Q, \Sigma, \delta, q_0, F)$.

Výstup: Redukovaný DKA $M' = (Q', \Sigma, \delta', q'_0, F')$, $L(M) = L(M')$.

Metoda:

1. Odstraň nedosažitelné stavy s využitím algoritmu 3.4.
2. $i := 0$
3. $\stackrel{0}{\equiv} := \{(p, q) \mid p \in F \Leftrightarrow q \in F\}$

DEF



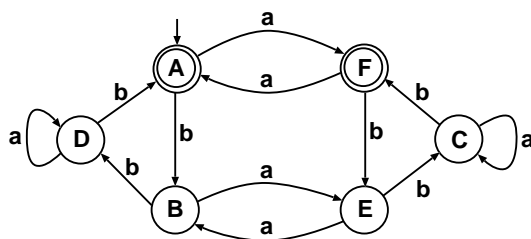
DEF



4. **repeat**
5. $\stackrel{i+1}{\equiv} := \{(p, q) \mid p \stackrel{i}{\equiv} q \wedge \forall a \in \Sigma : \delta(p, a) \stackrel{i}{\equiv} \delta(q, a)\}$
6. $i := i + 1$
7. **until** $\stackrel{i}{\equiv} = \stackrel{i-1}{\equiv}$
8. $Q' := Q / \stackrel{i}{\equiv}$
9. $\forall p, q \in Q \forall a \in \Sigma : \delta'([p], a) = [q] \Leftrightarrow \delta(p, a) = q$
10. $q'_0 = [q_0]$
11. $F' = \{[q] \mid q \in F\}$

Poznámka 3.2 Výraz $[x]$ značí ekvivalenční třídu určenou prvkem x .

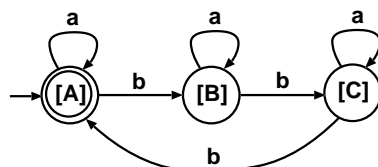
Příklad 3.13 Převeďte níže uvedený deterministický konečný automat (zadaný diagram přechodů) na odpovídající redukovaný deterministický konečný automat.

 $x + y$


Řešení: V řešení příkladu si ukážeme, jak vypadají stavy na jednotlivých řádcích algoritmu 3.5.

	$\stackrel{0}{\equiv}$	δ	a	b
1. Neobsahuje nedostupné stavy.	$I:$	A	F_I	B_{II}
		F	A_I	E_{II}
3. $\stackrel{0}{\equiv} = \{\{A, F\}, \{B, C, D, E\}\}$	$II:$	B	E_{II}	D_{II}
		C	C_{II}	F_I
5.1. $\stackrel{1}{\equiv} = \{\{A, F\}, \{B, E\}, \{C, D\}\}$		D	D_{II}	A_I
		E	B_{II}	C_{II}
	$\stackrel{1}{\equiv}$	δ	a	b
	$I:$	A	F_I	B_{II}
		F	A_I	E_{II}
5.2. $\stackrel{2}{\equiv} = \{\{A, F\}, \{B, E\}, \{C, D\}\} =$	$II:$	B	E_{II}	D_{III}
$\stackrel{1}{\equiv} = \equiv$		E	B_{II}	C_{III}
	$III:$	C	C_{III}	F_I
8. $Q' = \{[A], [B], [C]\}$, kde $[A] =$		D	D_{III}	A_I
$\{A, F\}$, $[B] = \{B, E\}$, $[C] =$				
$\{C, D\}$				

9–11.



3.3 Regulární množiny a regulární výrazy

3.3.1 Regulární množiny

Definice 3.9 Necht Σ je konečná abeceda. *Regulární množinu* nad abecedou Σ definujeme rekurzivně takto:

DEF

- (1) \emptyset (prázdná množina) je regulární množina nad Σ
- (2) $\{\epsilon\}$ (množina obsahující pouze prázdný řetězec) je regulární množina nad Σ
- (3) $\{a\}$ pro všechna $a \in \Sigma$ je regulární množina nad Σ
- (4) jsou-li P a Q regulární množiny nad Σ , pak také $P \cup Q$, $P \cdot Q$ a P^* jsou regulární množiny nad Σ
- (5) regulárními množinami jsou právě ty množiny, které lze získat aplikací 1–4.

Třída regulárních množin je tedy nejmenší třída jazyků, která obsahuje \emptyset , $\{\epsilon\}$, $\{a\}$ pro všechny symboly a a je uzavřena vzhledem k operacím sjednocení, součinu a iterace.

Příklad 3.14 $L = (\{a\} \cup \{d\}) \cdot (\{b\}^*) \cdot \{c\}$ je regulární množina nad $\Sigma = \{a, b, c, d\}$.

Ukážeme, že třída regulárních množin tvoří právě třídu \mathcal{L}_3 , tj. třídu jazyka typu 3 Chomského hierarchie.

Věta 3.12 Necht Σ je konečná abeceda. Pak

$$\emptyset \quad (1)$$

$$\{\epsilon\} \quad (2)$$

$$\{a\} \quad \text{pro všechna } a \in \Sigma \quad (3)$$

jsou jazyky typu 3 nad abecedou Σ .

Důkaz:

- (1) $G = (\{S\}, \Sigma, \emptyset, S)$ je gramatika typu 3 pro kterou $L(G) = \emptyset$
- (2) $G = (\{S\}, \Sigma, \{S \rightarrow \epsilon\}, S)$ je gramatika typu 3 pro kterou $L(G) = \{\epsilon\}$
- (3) $G_a = (\{S\}, \Sigma, \{S \rightarrow a\}, S)$ je gramatika typu 3, pro kterou $L(G_a) = \{a\}$

□

Věta 3.13 Necht L_1 a L_2 jsou jazyky typu 3 nad abecedou Σ . Pak

$$L_1 \cup L_2 \quad (1)$$

$$L_1 \cdot L_2 \quad (2)$$

$$L_1^* \quad (3)$$

jsou jazyky typu 3 nad abecedou Σ .

Důkaz: Protože L_1 a L_2 jsou jazyky typu 3, můžeme předpokládat existenci gramatik $G_1 = (N_1, \Sigma, P_1, S_1)$ a $G_2 = (N_2, \Sigma, P_2, S_2)$ typu 3 takových, že $L(G_1) = L_1$ a $L(G_2) = L_2$. Bez újmy na obecnosti dále předpokládejme, že množiny N_1 a N_2 jsou disjunktní.

- (1) Necht $G_3 = (N_1 \cup N_2 \cup \{S_3\}, \Sigma, P_1 \cup P_2 \cup \{S_3 \rightarrow S_1, S_3 \rightarrow S_2\}, S_3)$ je gramatika typu 3, kde S_3 je nový nonterminál, $S_3 \notin N_1$, $S_3 \notin N_2$. Zřejmě $L(G_3) = L(G_1) \cup L(G_2)$, poněvadž pro každou derivaci $S_3 \xRightarrow{G_3}^+ w$ existuje buď derivace $S_1 \xRightarrow{G_1}^+ w$, nebo $S_2 \xRightarrow{G_2}^+ w$ a naopak. Protože G_3 je gramatika typu 3, je $L(G_3)$ jazyk typu 3.
- (2) Necht $G_4 = (N_1 \cup N_2, \Sigma, P_4, S_1)$ je gramatika typu 3, jejíž množina přepisovacích pravidel P_4 je definována takto:
- (a) je-li $A \rightarrow xB$ v P_1 , pak $A \rightarrow xB$ je v P_4
 - (b) je-li $A \rightarrow x$ v P_1 , pak $A \rightarrow xS_2$ je v P_4
 - (c) všechna pravidla z P_2 jsou v P_4 .

Nyní, jestliže $S_1 \xRightarrow{G_1}^+ w$, pak $S_1 \xRightarrow{G_4}^+ wS_2$. Je-li dále $S_2 \xRightarrow{G_2}^+ x$, pak $S_2 \xRightarrow{G_4}^+ x$ a tedy $S_1 \xRightarrow{G_4}^+ wx$ pro libovolné w a x . Z toho plyne $L(G_1) \cdot L(G_2) \subseteq L(G_4)$.

Předpokládejme, že platí $S_1 \xRightarrow{G_4}^+ w$. Protože v P_4 nejsou žádná pravidla tvaru $A \rightarrow x$ z množiny P_1 , můžeme tuto derivaci zapsat ve tvaru $S_1 \xRightarrow{G_4}^+ x \quad S_2 \xRightarrow{G_4}^+ xy$, kde $w = xy$, přičemž všechna pravidla použitá v derivaci $S_1 \xRightarrow{G_4}^+ xS_2$ byla odvozena podle (a) a (b). Pak ale musí existovat derivace $S_1 \xRightarrow{G_1}^+ x$ a $S_2 \xRightarrow{G_2}^+ y$ a tudíž $L(G_4) \subseteq L(G_1) \cdot L(G_2)$. Z toho však plyne $L(G_4) = L(G_1) \cdot L(G_2)$.

- (3) Necht $G_5 = \{N_1 \cup \{S_5\}, \Sigma, P_5, S_5\}$, $S_5 \notin N_1$ a množina P_5 je konstruována takto:
- (a) je-li $A \rightarrow xB$ v P_1 , pak $A \rightarrow xB$ je v P_5
 - (b) je-li $A \rightarrow x$ v P_1 , pak $A \rightarrow xS_5$ a $A \rightarrow x$ jsou v P_5
 - (c) $S_5 \rightarrow S_1$ a $S_5 \rightarrow \epsilon$ jsou v P_5 .

Nyní je třeba dokázat tvrzení:
Derivace

$$S_5 \xRightarrow{G_5}^+ x_1 S_5 \xRightarrow{G_5}^+ x_1 x_2 S_5 \xRightarrow{G_5}^+ \dots \xRightarrow{G_5}^+ x_1 x_2 \dots x_{n-1} S_5 \xRightarrow{G_5}^+ x_1 x_2 \dots x_{n-1} x_n$$

existuje tehdy a jen tehdy, když existují derivace

$$S_1 \xRightarrow{G_1}^+ x_1, S_1 \xRightarrow{G_1}^+ x_2, \dots, S_1 \xRightarrow{G_1}^+ x_n.$$

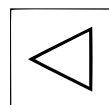
Z tohoto tvrzení, jehož důkaz ponecháme jako cvičení, bezprostředně plyne $L(G_5) = (L(G_1))^*$. Protože G_5 je gramatika typu 3, je i jazyk $L(G_5)$ typu 3. \square

3.3.2 Regulární výrazy

Obvyklou notací pro reprezentaci regulárních množin jsou regulární výrazy.

Definice 3.10 Regulární výrazy nad Σ a regulární množiny, které označují, jsou rekurzivně definovány takto:

- (1) \emptyset je regulární výraz označující regulární množinu \emptyset ,
- (2) ϵ je regulární výraz označující regulární množinu $\{\epsilon\}$,



- (3) a je regulární výraz označující regulární množinu $\{a\}$ pro všechny $a \in \Sigma$,
- (4) jsou-li p, q regulární výrazy označující regulární množiny P a Q , pak
- (a) $(p + q)$ je regulární výraz označující regulární množinu $P \cup Q$,
 - (b) (pq) je regulární výraz označující regulární množinu $P \cdot Q$,
 - (c) (p^*) je regulární výraz označující regulární množinu P^* .
- (5) Žádné jiné regulární výrazy nad Σ neexistují.

Konvence 3.1 Při používání regulárních výrazů stanovujeme následující konvence.

1. Regulární výraz p^+ značí regulární výraz pp^* .
2. Abychom minimalizovali počet používaných závorek, stanovujeme priority operátorů: $*$ a $+$ (iterace – nejvyšší priorita), \cdot (konkatenace), $+$ (alternativa).

Příklad 3.15

1. 01 odpovídá $\{01\}$.
2. 0^* odpovídá $\{0\}^*$.
3. $(0 + 1)^*$ odpovídá $\{0, 1\}^*$.
4. $(0 + 1)^*011$ značí množinu řetězců nad $\{0, 1\}$ končících 011 .
5. $(a + b)(a + b + 0 + 1)^*$ značí množinu řetězců nad $\{a, b, 0, 1\}$ začínající symbolem a nebo b .

Nyní jsme si definovali regulární výrazy. K tomu, abychom byli schopni odvodit všechny rovnice nad regulárními výrazy, potřebujeme sadu axiomů. Takovou sadou je *Kleeneho algebra*.

Definice 3.11 Kleeneho algebra sestává z neprázdné množiny se dvěma význačnými konstantami 0 a 1 , dvěma binárními operacemi $+$ a \cdot a unární operací $*$, které splňují následující axiomy:

$a + (b + c) = (a + b) + c$	asociativita $+$	[A.1]
$a + b = b + a$	komutativita $+$	[A.2]
$a + a = a$	idempotence $+$	[A.3]
$a + 0 = a$	0 je identitou pro $+$	[A.4]
$a(bc) = (ab)c$	asociativita \cdot	[A.5]
$a1 = 1a = a$	1 je identitou pro \cdot	[A.6]
$a0 = 0a = 0$	0 je anihilátorem pro \cdot	[A.7]
$a(b + c) = ab + ac$	distributivita zleva	[A.8]
$(a + b)c = ac + bc$	distributivita zprava	[A.9]
$1 + aa^* = a^*$		[A.10]
$1 + a^*a = a^*$		[A.11]
$b + ac \leq c \Rightarrow a^*b \leq c$		[A.12]
$b + ca \leq c \Rightarrow ba^* \leq c$		[A.13]

V A.12 a A.13 reprezentuje \leq uspořádání definované takto: $a \leq b \stackrel{def}{\iff} a + b = b$.

Příklad 3.16 Příklad Kleeneho algeber

 $x + y$
DEF
 $x + y$

- (1) Třída 2^{Σ^*} všech podmnožin Σ^* s konstantami \emptyset a ε a operacemi \cup , \cdot a $*$.
- (2) Třída všech regulárních podmnožin Σ^* s konstantami \emptyset a ε a operacemi \cup , \cdot a $*$.
- (3) Třída všech binárních relací nad množinou X s konstantami v podobě prázdné relace a identity a \cup , kompozicí relací a reflexivním tranzitivním uzávěrem kompozice jako operacemi.
- (4) Kleeneho algebry matic.

Poznámka 3.3 Axiomy A.12 a A.13 lze nahradit následujícími ekvivalentními vztahy (důkaz viz [3]):

$$ac \leq c \Rightarrow a^*c \leq c \quad [\text{A.14}]$$

$$ca \leq c \Rightarrow ca^* \leq c \quad [\text{A.15}]$$

Některé užitečné teoremy Kleeneho algebry, které lze odvodit z jejich axiomů jsou následující:

- (1) $0^* = 1$
- (2) $1 + a^* = a^*$
- (3) $a^* = a + a^*$
- (4) $a^*a^* = a^*$
- (5) $a^{**} = a^*$
- (6) $(a^*b)^*a^* = (a + b)^*$ pravidlo „vynořování“ [R.16]
- (7) $a(ba)^* = (ab)^*a$ pravidlo posuvu [R.17]
- (8) $a^* = (aa)^* + a(aa)^*$

Důkaz: Pro stručnost uvádíme pouze důkazy prvních dvou teorémů, ostatní viz [3].

$$(1) 0^* = 1: 0^* \stackrel{\text{A.10}}{=} 1 + 00^* \stackrel{\text{A.7}}{=} 1 + 0 \stackrel{\text{A.4}}{=} 1.$$

$$(2) 1 + a^* = a^* - \text{pro stručnost neuvádíme použití A.1, A.2:}$$

$$(a) 1 + a^* \leq a^*:$$

- i. A.10: $a^* = 1 + aa^*$
- ii. A.3: $a^* + a^* = 1 + aa^*$
- iii. A.10: $1 + aa^* + a^* = 1 + aa^*$
- iv. A.3: $1 + 1 + aa^* + a^* = 1 + aa^*$, neboli $1 + a^* + 1 + aa^* = 1 + aa^*$
- v. def. \leq : $1 + a^* \leq 1 + aa^*$
- vi. A.10: $1 + a^* \leq a^*$

$$(b) a^* \leq 1 + a^*:$$

- i. $1 + a^* = 1 + a^*$
- ii. A.3: $1 + a^* + a^* = 1 + a^*$
- iii. def. \leq : $a^* \leq 1 + a^*$

$$(c) \text{antisymetrie } \leq.$$

□

Příklad 3.17 Necht p , resp. q , označují regulární množiny P , resp. Q . Pak $p + q$ označuje $P \cup Q$ a $q + p$ označuje $Q \cup P$. $P \cup Q = Q \cup P$ (komutativita množinového sjednocení) $\Rightarrow p + q = q + p$.

Různé vlastnosti Kleeneho algeber se někdy snáze dokazují pro jednotlivé konkrétní příklady těchto algeber, např. pro Kleeneho algebru regulárních výrazů, kde lze např. využít vazby na teorii množin.




3.3.3 Rovnice nad regulárními výrazy

Definice 3.12 Rovnice, jejímiž složkami jsou koeficienty a neznámé, které reprezentují (dané a hledané) regulární výrazy, nazýváme *rovnice nad regulárními výrazy*.

Příklad 3.18 Uvažujme rovnici nad regulárními výrazy nad abecedou $\{a, b\}$

$$X = aX + b$$

Jejím řešením je regulární výraz $X = a^*b$.

Důkaz: LS = a^*b , PS = $a(a^*b) + b = a^+b + b = (a^+ + \epsilon)b = a^*b$. Ne vždy však existuje jediné řešení rovnice nad regulárními výrazy. \square

Příklad 3.19 Necht $X = pX + q$ je rovnice nad regulárními výrazy, kde p, q jsou regulární výrazy a p označuje regulární množinu P takovou, že $\epsilon \in P$. Pak

$$X = p^*(q + r)$$

je řešením této rovnice pro libovolné r (kterému nemusí ani odpovídat regulární množina, ale případně i obecnější jazyk).

Důkaz: LS = $p^*(q + r)$, PS = $p(p^*(q + r)) + q = pp^*(q + r) + q = p^*(q + r) + q = p^*(q + r)$. Je třeba si uvědomit, že $\epsilon \in P$. \square

Obvykle ale hledáme „nejmenší řešení“, tzv. *nejmenší pevný bod*, dané rovnice.

Věta 3.14 Nejmenším pevným bodem rovnice $X = pX + q$ je:

$$X = p^*q$$

Důkaz: LS = p^*q , PS = $pp^*q + q = (pp^* + \epsilon)q = p^*q$ \square

3.3.4 Soustavy rovnic nad regulárními výrazy

Příklad 3.20 Budiž dána soustava rovnic

$$\begin{aligned} X &= a_1X + a_2Y + a_3 \\ Y &= b_1X + b_2Y + b_3 \end{aligned}$$

Její řešení je:

$$\begin{aligned} X &= (a_1 + a_2b_2^*b_1)^*(a_3 + a_2b_2^*b_3) \\ Y &= (b_2 + b_1a_1^*a_2)^*(b_3 + b_1a_1^*a_3) \end{aligned}$$

Důkaz: Cvičení. \square

DEF

 $x + y$ $x + y$ $x + y$

Definice 3.13 Soustava rovnic nad regulárními výrazy je ve *standardním tvaru* vzhledem k neznámým $\Delta = \{X_1, X_2, \dots, X_n\}$, má-li soustava tvar

$$\bigwedge_{i \in \{1, \dots, n\}} X_i = \alpha_{i0} + \alpha_{i1}X_1 + \alpha_{i2}X_2 + \dots + \alpha_{in}X_n$$

kde α_{ij} jsou regulární výrazy nad nějakou abecedou Σ , $\Sigma \cap \Delta = \emptyset$.

Věta 3.15 Je-li soustava rovnic nad regulárními výrazy ve standardním tvaru, pak existuje její minimální pevný bod a algoritmus jeho nalezení.

Důkaz: Vyjadřujeme hodnotu jednotlivých proměnných pomocí řešení rovnice $X = pX + q$ jako regulární výraz s proměnnými, jejichž počet se postupně snižuje: Z rovnice pro X_n vyjádříme např. X_n jako regulární výraz nad Σ a X_1, \dots, X_{n-1} . Dosadíme za X_n do rovnice pro X_{n-1} a postup opakujeme. Jsou přitom možné (ale ne nutné) různé optimalizace tohoto pořadí.

□

Příklad 3.21 Vyřešte soustavu rovnic nad regulárními výrazy:

- (1) $X_1 = (01^* + 1)X_1 + X_2$
- (2) $X_2 = 11 + 1X_1 + 00X_3$
- (3) $X_3 = \varepsilon + X_1 + X_2$

Řešení:

1. Výraz pro X_3 dosadíme z (3) do (2). Dostaneme soustavu:

- (4) $X_1 = (01^* + 1)X_1 + X_2$
- (5) $X_2 = 11 + 1X_1 + 00(\varepsilon + X_1 + X_2) = 00 + 11 + (1 + 00)X_1 + 00X_2$

2. Ze (4) vyjádříme X_1 s využitím řešení rovnice $X = pX + q$ (věta 3.14):

$$(6) \quad X_1 = (01^* + 1)^*X_2 = (0 + 1)^*X_2$$

3. Dosazením do (5):

$$(7) \quad \begin{aligned} X_2 &= 00 + 11 + (1 + 00)(0 + 1)^*X_2 + 00X_2 = \\ &= 00 + 11 + (1 + 00)(0 + 1)^*X_2 \end{aligned}$$

4. Vypočtením X_2 jako řešení rovnice $X = pX + q$ dostaneme:

$$(8) \quad X_2 = ((1 + 00)(0 + 1)^*)^*(00 + 11)$$

5. Dosazením do (6) dostaneme:

$$(9) \quad X_1 = (0 + 1)^*((1 + 00)(0 + 1)^*)^*(00 + 11) = (0 + 1)^*(00 + 11)$$

6. Dosazením do (3) dostaneme:

$$(10) \quad \begin{aligned} X_3 &= \varepsilon + (0 + 1)^*(00 + 11) + ((1 + 00)(0 + 1)^*)^*(00 + 11) = \\ &= \varepsilon + ((0 + 1)^* + ((1 + 00)(0 + 1)^*)^*)(00 + 11) = \\ &= \varepsilon + (0 + 1)^*(00 + 11) \end{aligned}$$

Věta 3.16 Každý jazyk generovaný gramatikou typu 3 je regulární množinou: $\mathcal{L}_3 \subseteq \mathcal{L}_R$

Důkaz:

1. Nechť $L \in \mathcal{L}_3$ je libovolný jazyk typu 3. Již víme, že ho můžeme popsat konečným automatem $M = (Q, \Sigma, \delta, q_0, F)$. Nechť $Q = \{q_0, q_1, \dots, q_n\}$.

DEF

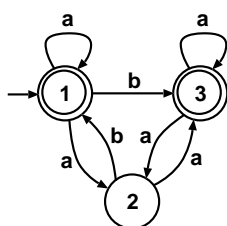
!

 $x + y$

2. Vytvoříme soustavu rovnic na regulární výrazy s proměnnými X_0, X_1, \dots, X_n ve standardním tvaru. Rovnice pro X_i popisuje množinu řetězců přijímaných ze stavu Q_i .
3. Řešením této soustavy získáme regulární výraz pro proměnnou X_0 , který reprezentuje jazyk L .

□

Příklad 3.22 Jazyk L popisuje regulární výraz, který je řešením této soustavy pro proměnnou X_1 .

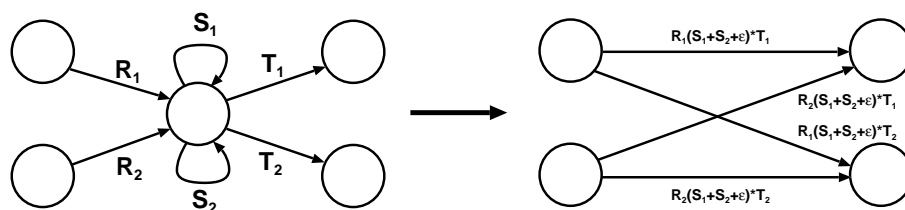


$$\begin{aligned} X_1 &= \varepsilon + aX_1 + aX_2 + bX_3 \\ X_2 &= bX_1 + aX_3 \\ X_3 &= \varepsilon + aX_2 + aX_3 \end{aligned}$$

Poznámka 3.4 Jiný převod KA na RV

Regulární přechodový graf je zobecnění konečného automatu, které umožňuje množinu počátečních stavů a regulární výrazy na hranách.

Každý regulární přechodový graf je možné převést na *regulární přechodový graf s jediným přechodem*, ze kterého odečteme hledaný regulární výraz. Zavedeme nový počáteční a koncový stav, které propojíme s původními počátečními a koncovými stavy ϵ přechody. Pak postupně odstraňujeme všechny původní stavy následujícím způsobem:



3.4 Převod regulárních výrazů na konečné automaty

Nyní si ukážeme přímý převod regulárního výrazu na deterministický konečný automat. Regulární výrazy budeme převádět nejprve na tzv. rozšířené konečné automaty a ty pak na deterministické konečné automaty.

Definice 3.14 *Rozšířený konečný automat* (RKA) je pětice $M = (Q, \Sigma, \delta, q_0, F)$, kde

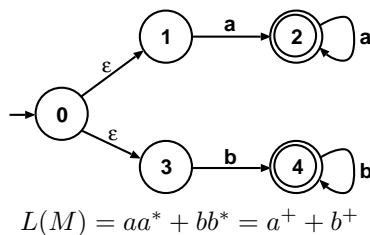
- (a) Q je konečná množina stavů,

 $x + y$


DEF

- (b) Σ je konečná vstupní abeceda,
- (c) δ je zobrazení $Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$,
- (d) $q_0 \in Q$ je počáteční stav,
- (e) $F \subseteq Q$ je množina koncových stavů.

Příklad 3.23 $M = (\{0, 1, 2, 3, 4\}, \{a, b\}, \delta, 0, \{2, 4\})$



x + y

Definice 3.15 Klíčovou funkcí v algoritmu převodu RKA na DKA má výpočet funkce, která k danému stavu určí množinu všech stavů, jež jsou dostupné po ε hranách diagramu přechodů funkce δ . Označme tuto funkci jako ε -uzávěr:

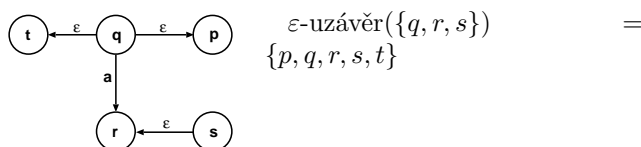
DEF

$$\varepsilon\text{-uzávěr}(q) = \{p \mid \exists w \in \Sigma^* : (q, w) \vdash^* (p, w)\}$$

Funkci ε -uzávěr zobecníme tak, aby argumentem mohla být množina $T \subseteq Q$:

$$\varepsilon\text{-uzávěr}(T) = \bigcup_{s \in T} \varepsilon\text{-uzávěr}(s)$$

Příklad 3.24



x + y

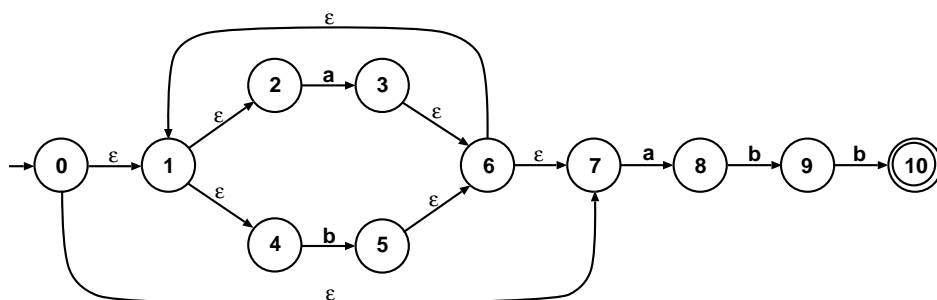
Pro popis výpočtu ε -uzávěru si zavedeme relaci $\xrightarrow{\varepsilon}$ v množině Q takto:

$$\forall q_1, q_2 \in Q : q_1 \xrightarrow{\varepsilon} q_2 \stackrel{\text{def}}{\iff} q_2 \in \delta(q_1, \varepsilon)$$

Pak $\varepsilon\text{-uzávěr}(p) = \{q \in Q \mid p \xrightarrow{\varepsilon}^* q\}$.

K výpočtu ε -uzávěru pak použijeme Warshallův algoritmus, doplníme diagonálu jedničkami a z příslušného řádku matice výsledné relace vyčteme ε -uzávěr.

Příklad 3.25



x + y

$$\begin{aligned}\varepsilon\text{-uzávěr}(3) &= \{3, 6, 7, 1, 2, 4\} \\ \varepsilon\text{-uzávěr}(\{1, 0\}) &= \{0, 1, 2, 4, 7\}\end{aligned}$$

Algoritmus 3.6 Převod rozšířeného konečného automatu na deterministický konečný automat.

Vstup: Rozšířený konečný automat $M = (Q, \Sigma, \delta, q_0, F)$.

Výstup: Deterministický konečný automat $M' = (Q', \Sigma, \delta', q'_0, F')$, $L(M) = L(M')$.

Metoda:

1. $Q' := 2^Q \setminus \{\emptyset\}$.
2. $q'_0 := \varepsilon\text{-uzávěr}(q_0)$.
3. $\delta' : Q' \times \Sigma \rightarrow Q' \cup \{nedef\}$ je vypočtena takto:
 - Necht $\forall T \in Q', a \in \Sigma : \bar{\delta}(T, a) = \bigcup_{q \in T} \delta(q, a)$.
 - Pak pro každé $T \in Q', a \in \Sigma$:
 - (a) pokud $\bar{\delta}(T, a) \neq \emptyset$, pak $\delta'(T, a) = \varepsilon\text{-uzávěr}(\bar{\delta}(T, a))$,
 - (b) jinak $\delta'(T, a) = nedef$.
4. $F' := \{S \mid S \in Q' \wedge S \cap F \neq \emptyset\}$.

Příklad 3.26 Aplikujte algoritmus 3.6 na automat z příkladu 3.25

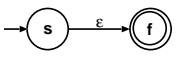
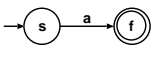

1. Počáteční stav, označíme ho A , je $A = \varepsilon\text{-uzávěr}(0) = \{0, 1, 2, 4, 7\}$.
2. $\delta'(A, a) = \varepsilon\text{-uzávěr}(\{3, 8\}) = \{1, 2, 3, 4, 6, 7, 8\} = B$.
3. $\delta'(A, b) = \varepsilon\text{-uzávěr}(\{5\}) = \{1, 2, 4, 5, 6, 7\} = C$.
4. $\delta'(B, a) = \varepsilon\text{-uzávěr}(\{3, 8\}) = B$.
5. $\delta'(B, b) = \varepsilon\text{-uzávěr}(\{5, 9\}) = \{1, 2, 4, 5, 6, 7, 9\} = D$.
6. $\delta'(C, a) = \varepsilon\text{-uzávěr}(\{3, 8\}) = B$.
7. $\delta'(C, b) = \varepsilon\text{-uzávěr}(\{5\}) = C$.
8. $\delta'(D, a) = \varepsilon\text{-uzávěr}(\{3, 8\}) = B$.
9. $\delta'(D, b) = \varepsilon\text{-uzávěr}(\{5, 10\}) = \{1, 2, 4, 5, 6, 7, 10\} = E$.
10. $\delta'(E, a) = \varepsilon\text{-uzávěr}(\{3, 8\}) = B$.
11. $\delta'(E, b) = \varepsilon\text{-uzávěr}(\{5\}) = C$.
12. Množina koncových stavů $F = \{E\}$.

Algoritmus 3.7 Převod regulárního výrazu na rozšířený konečný automat.

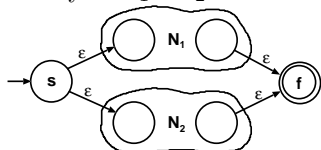
Vstup: Regulární výraz r popisující regulární množinu R nad Σ .

Výstup: Rozšířený konečný automat M takový, že $L(M) = R$.

Metoda: 1. Rozložíme r na jeho primitivní složky podle rekurzivní definice regulární množiny/výrazu.

2. (a) Pro výraz ε zkonstruujeme automat: 
- (b) Pro výraz $a, a \in \Sigma$ zkonstruujeme automat: 
- (c) Pro výraz \emptyset zkonstruujeme automat: 
- (d) Necht N_1 je automat přijímající jazyk specifikovaný výrazem r_1 a necht N_2 je automat přijímající jazyk specifikovaný výrazem r_2 .

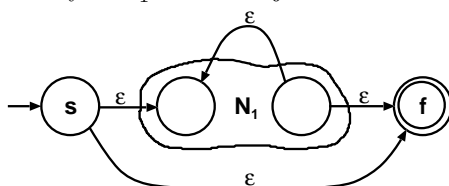
- i. Pro výraz $r_1 + r_2$ zkonstruujeme automat:



- ii. Pro výraz $r_1 r_2$ zkonstruujeme automat:



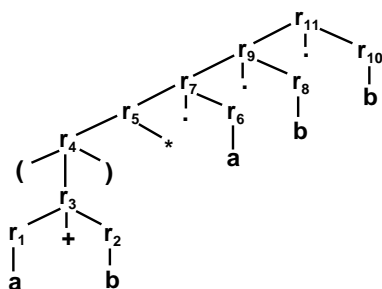
- iii. Pro výraz r_1^* zkonstruujeme automat:



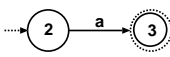
Příklad 3.27 Vytvořme rozšířený konečný automat pro regulární výraz $(a + b)^*abb$:

$x + y$

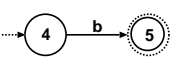
1. Rozklad regulárního výrazu vyjádříme stromem:



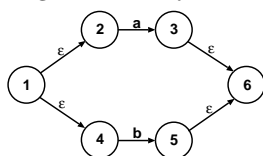
2. (a) Regulárnímu výrazu $r_1 = a$ přísluší automat N_1 :



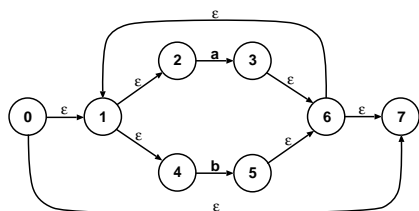
- (b) Regulárnímu výrazu $r_2 = b$ přísluší automat N_2 :



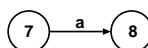
- (c) Regulárnímu výrazu $r_1 + r_2$ přísluší automat N_3 :



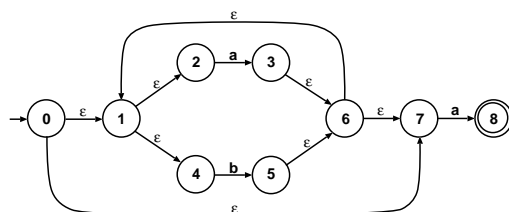
- (d) Automat N_4 pro $r_4 = (r_3)$ je stejný jako N_3 , zkonstruuujeme tedy rovnou N_5 pro výraz $r_5 = r_4^* = (a + b)^*$:



- (e) Regulárnímu výrazu $r_6 = a$ přísluší automat N_6 :



- (f) Regulárnímu výrazu $r_7 = r_5 r_6$ přísluší automat N_7 :



(...) Pokračujeme, až do získání automatu z příkladu 3.25.

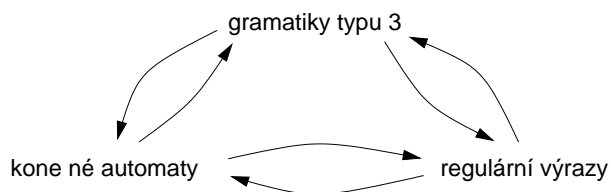
Převod RV na RKA zavádí mnoho vnitřních stavů a je proto obvykle následován použitím algoritmu *minimalizace DKA* (algoritmus 3.5).

Příklad 3.28 Ověřte, zda deterministický konečný automat z příkladu 3.26 je či není minimální.

Pro porovnávání regulárních gramatik, konečných automatů a regulárních výrazů můžeme shrnout, že

1. gramatiky typu 3 (pravé/levé regulární gramatiky, pravé/levé lineární gramatiky),
2. (rozšířené/nedeterministické/deterministické) konečné automaty a
3. regulární výrazy

mají ekvivalentní vyjadřovací sílu.



3.5 Vlastnosti regulárních jazyků

Podobně jako u dalších tříd jazyků budeme nyní zkoumat následující vlastnosti regulárních jazyků:

- (1) vlastnosti strukturální,



- (2) vlastnosti uzávěrové a
 (3) rozhodnutelné problémy pro regulární jazyky.

3.5.1 Strukturální vlastnosti regulárních jazyků

Věta 3.17 Každý konečný jazyk je regulární.

Důkaz: Necht $L = \{w_1, w_2, \dots, w_n\}$, $w_i \in \Sigma$.

Pak $L = L(G)$, kde $G = (\{S\}, \Sigma, \{S \rightarrow w_1, S \rightarrow w_2, \dots, S \rightarrow w_n\}, S)$. G je zřejmě gramatika typu 3.

□

Opak věty 3.17 zjevně neplatí:

Příklad 3.29 Sestrojte gramatiku typu 3 generující jazyk $\{0, 1\}^*$.

Řešení: $\rightarrow \textcircled{S} \rightarrow_{0,1} \Rightarrow G = (\{S\}, \{0, 1\}, \{S \rightarrow \varepsilon, S \rightarrow 0S, S \rightarrow 1S\}, S)$

Pumping lemma

Pokud se zajímáme o vlastnost, zda jazyk spadá do dané třídy jazyků, je užitečné tzv. *Pumping lemma*. Neformálně řečeno Pumping lemma tvrdí, že v každé dostatečně dlouhé větě každého regulárního jazyka jsme schopni najít poměrně krátkou sekvenci, kterou je možné vypustit, resp. zopakovat libovolný počet krát, přičemž dostáváme stále věty daného jazyka.

Věta 3.18 Necht L je nekonečný regulární jazyk. Pak existuje celočíselná konstanta $p \geq 0$ taková, že platí:

$$\begin{aligned} w \in L \quad \wedge \quad |w| \geq p \quad \Rightarrow \quad & w = x y z \quad \wedge \\ & 0 < |y| \leq p \quad \wedge \\ & x y^i z \in L \text{ pro } i \geq 0. \end{aligned}$$

Důkaz: Necht $L = L(M)$, $M = (Q, \Sigma, \delta, q_0, F)$ je konečný automat, kde $|Q| = n$. Položme $p = n$. Je-li $w \in L$ a $|w| \geq n$, pak M přijme větu w „průchodem“ alespoň $n + 1$ konfiguracemi a tudíž alespoň dvě z nich obsahují stejný stav, tedy:

$$(q_0, w) = (q_0, xyz) \stackrel{*}{\vdash} (r, yz) \stackrel{k}{\vdash} (r, z) \stackrel{*}{\vdash} (q_F, \varepsilon), \quad q_F \in F$$

pro nějaký stav $r \in Q$ a k takové, že $0 < k \leq n$.

Pak ale existuje posloupnost konfigurací:

$$\begin{aligned} (q_0, xy^i z) & \stackrel{*}{\vdash} (r, y^i z) \\ & \quad + \\ & \quad \vdash (r, y^{i-1} z) \\ & \quad \vdots \\ & \quad + \\ & \quad \vdash (r, z) \\ & \quad + \\ & \quad \vdash (q_F, \varepsilon) \end{aligned}$$

□

To ovšem znamená, že $xy^i z \in L(M)$, a to nejen pro $i > 0$, ale i pro případ $i = 0$: $(q_0, xz) \stackrel{*}{\vdash} (r, z) \stackrel{*}{\vdash} (q_F, \varepsilon)$, $q_F \in F$.

Věty 3.18 se často používá k důkazu, že daný jazyk není regulární.

Příklad 3.30 Dokažte, že jazyk $L = \{0^n 1^n \mid n \geq 1\}$ není regulární.

Důkaz: Předpokládejme, že L je regulární. Pak podle věty 3.18 může být věta $0^k 1^k$ pro dostatečně velké k zapsána ve tvaru $xyz = 0^k 1^k$, $0 < |y| \leq k$, a platí $xy^i z \in L$ pro $i \geq 0$.

Při hledání podřetězce y mohou nastat 3 možnosti:

$$\underbrace{000\dots}_y \cdot \underbrace{011}_y \underbrace{1\dots1}_y$$

- $y \in \{0\}^+$ pak ale $xy^i z \notin L$ — nesouhlasí počet 0 a 1
- $y \in \{1\}^+$ pak ale $xy^i z \notin L$ — nesouhlasí počet 0 a 1
- $y \in \{0\}^+ \cdot \{1\}^+$ pak ale $xy^i z \notin L$ (všechny 0 nepředcházejí všechny 1)

Řetězec tudíž nelze vybrat, a proto $L \notin \mathcal{L}_3$.

□

Příklad 3.31 Dokažte, že jazyk $L = \{a^q \mid q \text{ je prvočíslo}\}$ není regulární.

Důkaz: Předpokládejme, že L je regulární. Pak podle věty 3.18 existuje p takové, že každá věta $a^q \in L$, kde $q \geq p$, může být zapsána ve tvaru $a^q = xyz$, $0 < |y| \leq p$, a platí $xy^i z \in L$ pro $i \geq 0$.

Zvolme r prvočíslo větší než p .

$a^r \in L$ a $|a^r| = r \geq p$, což implikuje $a^r = xyz$, $0 < |y| \leq p$, a platí $xy^i z \in L$ pro $i \geq 0$.

Nechť $|y| = k$ a zvolme $i = r + 1$.

Pak ale $|xy^{r+1}z| = |xyz| + |y^r| = r + r \cdot k = r \cdot (k + 1)$, což však není prvočíslo, a tedy $xy^i z \notin L$. Spor.

□

3.5.2 Myhill-Nerodova věta

Myhill-Nerodova věta charakterizuje některé zásadní vztahy mezi konečnými automaty nad abecedou Σ a jistými ekvivalenčními relacemi nad řetězci ze Σ^* . Dále slouží k popisu některých z nutných a postačujících podmínek pro to, aby daný jazyk byl jazykem regulárním (používá se často k důkazu neregularity jazyka). Poskytuje také formální bázi pro elegantní důkaz existence unikátního (až na isomorfismus) minimálního DKA k danému regulárnímu jazyku.

Pravá kongruence a prefixová ekvivalence

Pro zopakování: *ekvivalence* \sim je binární relace, která je *reflexivní*, *symetrická* a *tranzitivní*. Index ekvivalence \sim je počet tříd rozkladu Σ^* / \sim . Je-li těchto tříd nekonečně mnoho, definujeme index jako ∞ .

Definice 3.16 Nechť Σ je abeceda a \sim je ekvivalence na Σ^* . Ekvivalence \sim je *pravou kongruencí* (je zprava invariantní), pokud pro každé $u, v, w \in \Sigma^*$ platí

$$u \sim v \implies uw \sim vw$$

Věta 3.19 Ekvivalence \sim na Σ^* je pravá kongruence právě tehdy, když pro každé $u, v \in \Sigma^*$, $a \in \Sigma$ platí $u \sim v \implies ua \sim va$.

Důkaz: „ \implies “ je triviální, „ \Leftarrow “ lze snadno ukázat indukcí nad délkou w .

□

x + y

DEF



Definice 3.17 Necht L je libovolný (ne nutně regulární) jazyk nad abecedou Σ . Na množině Σ^* definujeme relaci \sim_L zvanou *prefixová ekvivalence* pro L takto:

$$u \sim_L v \stackrel{def}{\iff} \forall w \in \Sigma^* : uw \in L \iff vw \in L$$

DEF

Věta 3.20 Necht L je jazyk nad Σ . Pak následující tvrzení jsou ekvivalentní:

1. L je jazyk přijímaný deterministickým konečným automatem.
2. L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ^* s konečným indexem.
3. Relace \sim_L má konečný index.

!

Myhill-
Nerodova
věta

Důkaz: Dokážeme následující implikace:

- (a) $1 \Rightarrow 2$
- (b) $2 \Rightarrow 3$
- (c) $3 \Rightarrow 1$

Z definice ekvivalence ($a \Leftrightarrow b \stackrel{def}{\iff} a \Rightarrow b \wedge b \Rightarrow a$) a ze základní tautologie výrokové logiky ($a \Rightarrow b \wedge b \Rightarrow c \Rightarrow a \Rightarrow c$) plyne tvrzení věty.

(a) Důkaz implikace $1 \Rightarrow 2$

Je-li L přijíman DKA, pak L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ^* s konečným indexem.

Pro DKA $M = (Q, \Sigma, \delta, q_0, F)$ zavedme *zobecněnou přechodovou funkci* $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ tak, že $\forall q_1, q_2 \in Q, w \in \Sigma^* : \hat{\delta}(q_1, w) = q_2 \Leftrightarrow (q_1, w) \stackrel{*}{\vdash}_M (q_2, \varepsilon)$.

Pro daný L přijímaný konečným automatem M zkonstruujeme \sim s potřebnými vlastnostmi:

- (1) Necht $M = (Q, \Sigma, \delta, q_0, F)$ a δ je totální.
- (2) Zvolíme \sim jako binární relaci na Σ^* takovou, že $u \sim v \iff \hat{\delta}(q_0, u) = \hat{\delta}(q_0, v)$.
- (3) Ukážeme, že \sim má potřebné vlastnosti:
 - i. \sim je *ekvivalence*: je reflexivní, symetrická a tranzitivní.
 - ii. \sim má *konečný index*: třídy rozkladu odpovídají stavům automatu.
 - iii. \sim je *pravá kongruence*: Necht $u \sim v$ a $a \in \Sigma$. Pak $\hat{\delta}(q_0, ua) = \delta(\hat{\delta}(q_0, u), a) = \delta(\hat{\delta}(q_0, v), a) = \hat{\delta}(q_0, va)$ a tedy $ua \sim va$.
 - iv. L je *sjednocením některých tříd* $\Sigma^* \setminus \sim$: těch, které odpovídají F .

(b) Důkaz implikace $2 \Rightarrow 3$

Existuje-li relace \sim splňující podmínku 2, pak \sim_L má konečný index.

Pro všechny $u, v \in \Sigma^*$ takové, že $u \sim v$, platí $u \sim_L v$: Necht $u \sim v$. Ukážeme, že také $u \sim_L v$, tj. $\forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L$. Víme, že $uw \sim vw$ a protože L je sjednocením některých tříd rozkladu $\Sigma^* \setminus \sim$, platí též $uw \in L \Leftrightarrow vw \in L$. Víme tedy, že $\sim \subseteq \sim_L$ (tj. \sim_L je největší pravá kongruence s danými vlastnostmi). Každá

třída \sim je obsažena v nějaké třídě \sim_L . Index \sim_L nemůže být větší než index \sim . \sim má konečný index a tedy i \sim_L má konečný index.

(c) Důkaz implikace $3 \Rightarrow 1$

Má-li \sim_L konečný index, pak L je přijímán nějakým konečným automatem. Zkonstruujeme $M = (Q, \Sigma, \delta, q_0, F)$ přijímající L :

- (1) $Q = \Sigma^* \setminus \sim$ (stavy jsou třídy rozkladu Σ^* relací \sim),
- (2) $\forall u \in \Sigma^*, u \in \Sigma : \delta([u], a) = [ua]$,
- (3) $q_0 = [\varepsilon]$,
- (4) $F = \{[x] \mid x \in L\}$.

Uvedená konstrukce je *korektní*, tj. $L = L(M)$: indukcí nad délkou slova v ukážeme, že $\forall v \in \Sigma^* : \hat{\delta}([\varepsilon], v) = [v]$. Navíc $v \in L \iff [v] \in F \iff \hat{\delta}([\varepsilon], v) \in F$. \square

Jak bylo řečeno dříve, Myhill-Nerodova věta také slouží pro elegantní dokázání, zda je daný jazyk regulární (resp. není regulární). Následující příklad ukazuje toto využití.

Příklad 3.32 Dokažte, že jazyk $L = \{a^n b^n \mid n \geq 0\}$ není regulární.

Důkaz: Žádné řetězce $\varepsilon, a, a^2, a^3, \dots$ nejsou \sim_L -ekvivalentní, protože $a^i b^i \in L$, ale $a^i b^j \notin L$ pro $i \neq j$. Relace \sim_L má tedy nekonečně mnoho tříd (neboli nekonečný index). Dle Myhill-Nerodovy věty tudíž nemůže být L přijímán žádným konečným automatem. \square

Další využití Myhill-Nerodovy věty je pro elegantní důkaz existence minimálního deterministického konečného automatu k danému regulárnímu výrazu.

Věta 3.21 (2. varianta Myhill-Nerodovy věty) Počet stavů libovolného minimálního

DKA přijímajícího L je roven indexu \sim_L . (Takový DKA existuje právě tehdy, když je index \sim_L konečný.)

Důkaz: Každý DKA (můžeme uvažovat DKA bez nedosažitelných stavů) určuje jistou pravou kongruenci s konečným indexem a naopak. Je-li L regulární, je \sim_L největší pravou kongruencí s konečným indexem takovou, že L je sjednocením některých tříd příslušného rozkladu. Konečný automat, který odpovídá \sim_L (viz důkaz $3 \Rightarrow 1$ Myhill-Nerodovy věty), je tedy minimální konečný automat přijímající L . \square

3.5.3 Uzávěrové vlastnosti regulárních jazyků

Věta 3.22 Třída regulárních jazyků je uzavřena (mimo jiné) vzhledem k operacím \cup (sjednocení), \cdot (konkatenace) a $*$ (iterace).

Důkaz: Plyne z definice regulárních množin a ekvivalence regulárních množin a regulárních jazyků. \square

Věta 3.23 Třída regulárních jazyků tvoří množinovou *Booleovu algebru*.

$x + y$



Důkaz:

Označme \mathcal{L}_3 třídu všech regulárních jazyků abecedou Σ . Ukážeme, že algebraická struktura $\langle \mathcal{L}_3, \cup, \cap, ', \Sigma^*, \emptyset \rangle$ je Booleova algebra.

1. Protože $\Sigma^* \in \mathcal{L}_3$ i $\emptyset \in \mathcal{L}_3$, jsou tyto jazyky zřejmě „jedničkou“, resp. „nulou“ uvažované algebry. Dále je zřejmá uzavřenost vůči \cup .
2. Dokážeme uzavřenost vzhledem ke komplementu nad abecedou Δ , $\Delta \subseteq \Sigma$. K jazyku L sestrojíme *úplně definovaný* KA M .

$$M = (Q, \Delta, \delta, q_0, F)$$

takový, že $L = L(M)$. Pak KA M'

$$M' = (Q, \Delta, \delta, q_0, Q \setminus F)$$

zřejmě přijímá jazyk $\Delta^* \setminus L$ (komplement L v Δ^*).

Komplement vzhledem k Σ^* :

$$\bar{L} = L(M') \cup \Sigma^*(\Sigma \setminus \Delta)\Sigma^*$$

což je podle věty 3.19 regulární jazyk.

3. Uzavřenost vzhledem k průniku plyne z de Morganových zákonů:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}} = \overline{\overline{L_1}} \cap \overline{\overline{L_2}}$$

a tedy $L_1, L_2 \in \mathcal{L}_3 \Rightarrow L_1 \cap L_2 \in \mathcal{L}_3$.

□

Věta 3.24 Nechť $L \in \mathcal{L}_3$ a nechť $L^R = \{w^R \mid w \in L\}$. Pak $L^R \in \mathcal{L}_3$.

Důkaz: Nechť M je konečný automat takový, že $L = L(M)$. Lze sestrojit M' takový, aby $L(M') = L^R$.

Konstrukce automatu M' se ponechává na čtenáři jako cvičení.

□

3.5.4 Rozhodnutelné problémy regulárních jazyků

Tato podkapitola se zabývá těmito základními rozhodnutelnými problémy regulárních jazyků: problém *neprázdnoti*: $L \neq \emptyset$?, problém *náležitosti*: $w \in L$? a problém *ekvivalence*: $L(G_1) = L(G_2)$?

Věta 3.25 Ve třídě \mathcal{L}_3 je rozhodnutelný problém *neprázdnoti* jazyka i problém *náležitosti* řetězce (do jazyka).

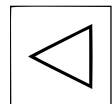
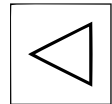
Důkaz: K jazyku $L \in \mathcal{L}_3$ sestrojíme deterministický konečný automat M , $L = L(M)$:

$$M = (Q, \Sigma, \delta, q_0, F)$$

neprázdnot: $L(M) \neq \emptyset \iff \exists q \in Q : (q \in F \wedge q \text{ je dostupný z } q_0)$

náležitost: $w \in L \iff (q_0, w) \vdash^* (q, \varepsilon) \wedge q \in F$

□



Věta 3.26 Necht $L_1 = L(G_1)$ a $L_2 = L(G_2)$ jsou dva jazyky generované regulárními gramatikami G_1 a G_2 . Pak je rozhodnutelný problém *ekvivalence*, tj. $L(G_1) = L(G_2)$ nebo $L(G_1) \neq L(G_2)$.

Důkaz: Necht $M_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1)$, resp. $M_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2)$ jsou KA přijímající jazyky L_1 , resp. L_2 takové, že $Q_1 \cap Q_2 = \emptyset$.

Vytvoříme konečný automat M takto:

$$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta_1 \cup \delta_2, q_0^1, F_1 \cup F_2)$$

a vypočítáme relaci \equiv nerozlišitelnosti stavů z $Q_1 \cup Q_2$ pro automat M .

$$\text{Pak } L(G_1) = L(G_2) \iff q_0^1 \equiv q_0^2$$

□

Regulární jazyky, zahrnující všechny konečné jazyky, představují třídu formálních jazyků s velmi výrazným počtem aplikací zvláště pak pro jejich mimořádnou rozhodovací sílu. K nejrozšířenějším prostředkům jejich specifikace patří, vedle gramatik typu 3 a nedeterministických a deterministických konečných jazyků také regulární výrazy. Regulární jazyky tvoří jednu z instancí Kleenovy algebry, která umožňuje řešit standartizované soustavy rovnic nad regulárními výrazy. Algoritmy převodů mezi ekvivalentními specifikačními prostředky jsou základem speciálních jazykových procesorů, např. konstruktorů lexikálních analyzátorů překladačů.

3.6 Cvičení

Cvičení 3.6.1 Ke konečnému nedeterministickému automatu

$$M = (\{q_0, q_1, q_2, q_3, q_F\}, \{1, 2, 3\}, \delta, q_0, \{q_F\}),$$

kde zobrazení δ je definováno touto tabulkou .

Q	Σ		
	1	2	3
q₀	$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_3\}$
q₁	$\{q_1, q_F\}$	$\{q_1\}$	$\{q_1\}$
q₂	$\{q_2\}$	$\{q_2, q_F\}$	$\{q_1\}$
q₃	$\{q_3\}$	$\{q_3\}$	$\{q_3, q_F\}$
q_F	\emptyset	\emptyset	\emptyset

sestrojte deterministický automat M' , pro který platí $L(M') = L(M)$.

Cvičení 3.6.2 Necht $L_1 = L(M_1)$ a $L_2 = L(M_2)$ jsou jazyky přijímané konečnými automaty $M_1 = (Q_1, \Sigma_1, \delta_1, q_0^1, F_1)$ a $M_2 = (Q_2, \Sigma_2, \delta_2, q_0^2, F_2)$. Analogicky větě 3.13 ukažte konstrukci automatů M_3 , M_4 a M_5 , pro které platí:

$$L(M_3) = L_1 \cup L_2, \quad L(M_4) = L_1 \cdot L_2 \quad \text{a} \quad L(M_5) = L_1^*.$$

Cvičení 3.6.3 K pravé lineární gramatice, která obsahuje pravidla

$$\begin{aligned} A &\rightarrow B \mid C \\ B &\rightarrow 0B \mid 1B \mid 011 \\ C &\rightarrow 0D \mid 1C \mid \epsilon \\ D &\rightarrow 0C \mid 1D \end{aligned}$$

vytvořte pravou lineární gramatiku, jež generuje stejný jazyk. Nonterminál A je výchozím symbolem gramatiky.

Cvičení 3.6.4 Vytvořte pravou regulární gramatiku, která generuje jazyk přijímaný automatem M ze cvičení 3.6.1.

Cvičení 3.6.5 Vytvořte konečný automat, který přijímá jazyk generovaný gramatikou ze cvičení 3.6.3.

Cvičení 3.6.6 Na základě algoritmu, jenž je „inverzní“ k algoritmu 3.3, sestrojte levou regulární gramatiku pro jazyk reálných čísel pro programovací jazyky. Automat přijímající tento jazyk je uveden v příkladě 3.1.

Cvičení 3.6.7 Na základě gramatiky ze cvičení 2.4.5 vytvořte konečný deterministický automat, který přijímá jazyk čísel programovacího jazyka Pascal.

Cvičení 3.6.8 Gramatika $G = (\{S, A_n, A_{n-1}, \dots, A_0\}, \{a, b\}, P, S)$ s pravidly

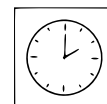
$$\begin{aligned} S &\rightarrow A_n \\ A_n &\rightarrow A_{n-1}A_{n-1} \\ A_{n-1} &\rightarrow A_{n-2}A_{n-2} \\ &\vdots \\ A_2 &\rightarrow A_1A_1 \\ A_1 &\rightarrow A_0A_0 \\ A_0 &\rightarrow a \mid b \end{aligned}$$

popisuje pro $n \geq 0$ regulární jazyk nad abecedou $\{a, b\}$.

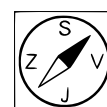
- Specifikujte jazyk $L(G)$ gramatikou typu 3.
- Převedte tuto gramatiku na pravou regulární gramatiku.
- Převedte tuto gramatiku na ekvivalentní NKA a DKA.
- Popište jazyk specifikovaný gramatikou G ve tvaru regulárního výrazu.
- K získanému regulárnímu výrazu vytvořte ekvivalentní rozšířený konečný automat a deterministický konečný automat.
- Srovnajte DKA, které jste získali v (c) a (e). Ukažte, že automat přijímající jazyk $L(G)$ nepotřebuje více než $2^n + 1$ stavů.

Kapitola 4

Bezkontextové jazyky a zásobníkové automaty



25:00



Prvním cílem této kapitoly je důkladné zvládnutí pojmů a poznatků teorie *bezkontextových gramatik*, jež jsou základem moderních aplikací v překladačích programovacích jazyků či při formální specifikaci a verifikaci (nejen) počítačových systémů. Kapitola vymezuje základní transformace bezkontextových gramatik zachovávající ekvivalenci bezkontextových jazyků a vede k algoritmizaci a důkazu těchto transformací. Druhým cílem této kapitoly je osvojení prostředků a metod alternativních popisů bezkontextových jazyků různými variantami *zásobníkových automatů*. Dokazuje se vztah mezi bezkontextovými gramatikami a nedeterministickými a deterministickými zásobníkovými automaty. Konečně třetím cílem je vést čtenáře k rigoróznímu pochopení základních *vlastností bezkontextových jazyků* a seznámit ho, méně formálně, i s některými pokročilejšími užitečnými poznatky teorie bezkontextových jazyků.

Definice 4.1 Bezkontextová gramatika G je čtveřice $G = (N, \Sigma, P, S)$

- (1) N je konečná množina nonterminálních symbolů
- (2) Σ je konečná množina terminálních symbolů
- (3) P je konečná množina přepisovacích pravidel (pravidel) tvaru $A \rightarrow \alpha$, $A \in N$ a $\alpha \in (N \cup \Sigma)^*$
- (4) $S \in N$ je výchozí symbol gramatiky.

Dále budeme gramatikou bez další specifikace rozumět bezkontextovou gramatiku.

Příklad 4.1 Gramatika $G = (\{S, A, B\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidla

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aAb \mid ab \\ B &\rightarrow cBd \mid cd \end{aligned}$$

generuje bezkontextový jazyk $L(G) = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\}$

Jazyk $L(G)$ je součinem dvou bezkontextových jazyků generovaných gramatikami

$$\begin{aligned} G_1 &= (\{A\}, \{a, b\}, \{A \rightarrow aAb, A \rightarrow ab\}, A) \\ G_2 &= (\{B\}, \{c, d\}, \{B \rightarrow cBd, B \rightarrow cd\}, B) \end{aligned}$$

Poznamenejme, že jazyk $L(G_1) = \{a^n b^n \mid n \geq 1\}$ a tudíž ani jazyk $L(G)$ není jazykem regulárním, protože konečný automat nemůže pro libovolné n „počítat“ stejný počet výskytů symbolů a a b . Na druhé straně, jazyk $\{a^n b^n c^n d^n \mid n \geq 1\}$, dokonce ani jazyk $\{a^n b^n c^n \mid n \geq 0\}$ není jazykem bezkontextovým.

DEF

$x + y$

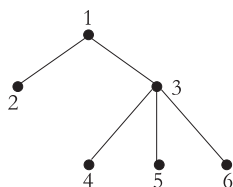
4.1 Derivační strom

Důležitým prostředkem pro grafické vyjádření struktury věty (její derivace) je kořenový, uzlově ohodnocený strom, který se nazývá derivačním nebo syntaktickým stromem.

Připomeňme, že strom je orientovaný acyklický graf s těmito vlastnostmi:

- (1) Existuje jediný uzel, tzv. *kořen stromu*, do něhož nevstupuje žádná hrana.
- (2) Do všech ostatních uzlů grafu vstupuje právě jedna hrana.

Uzly, z nichž žádná hrana nevystupuje, se nazývají *koncové uzly stromu* (listy). Při kreslení stromu je obvykle dodržována tato konvence: kořen leží nejvýše, všechny hrany jsou orientovány směrem dolů. Budeme-li tuto konvenci dodržovat, pak můžeme vynechat šipky, které označují orientaci hran. Kořenem stromu na



Obrázek 4.1: Příklad stromu

4.1 je uzel 1, uzly 2, 4, 5, 6 jsou koncovými uzly stromu.

Definice 4.2 Nechť δ je věta nebo větná forma generovaná v gramatice $G = (N, \Sigma, P, S)$ a nechť $S = \nu_0 \Rightarrow \nu_1 \Rightarrow \nu_2 \dots \Rightarrow \nu_k = \delta$ její derivace v G . *Derivační strom* příslušející této derivaci je strom s těmito vlastnostmi:

- (1) Uzly derivačního stromu jsou (ohodnoceny) symboly z množiny $N \cup \Sigma$; kořen stromu je označen výchozím symbolem S .
- (2) Přímé derivaci $\nu_{i-1} \Rightarrow \nu_i$, $i = 0, 1, \dots, k$, kde

$$\begin{aligned} \nu_{i-1} &= \mu A \lambda, \quad \mu, \lambda \in (N \cup \Sigma)^*, \quad A \in N \\ \nu_i &= \mu \alpha \lambda \text{ a} \\ A &\rightarrow \alpha, \quad \alpha = X_1 \dots X_n \text{ je pravidlo z } P, \end{aligned}$$

odpovídá právě n hran (A, X_j) , $j = 1, \dots, n$ vycházejících z uzlu A jež jsou uspořádány zleva doprava v pořadí $(A, X_1), (A, X_2), \dots, (A, X_n)$.

- (3) Označení koncových uzlů derivačního stromu vytváří zleva doprava větnou formu nebo větu δ (plyne z (1) a (2)).

Příklad 4.2 V Gramatice z příkladu 4.1 můžeme generovat řetězec $aabbcd$ např. derivací:

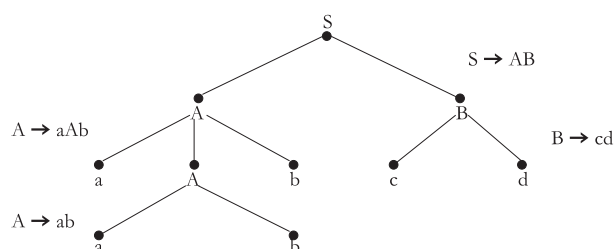
$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbcd$$

Derivační strom odpovídající této derivaci je na obrázku 4.2. Po stranách jsou uvedena použitá pravidla.

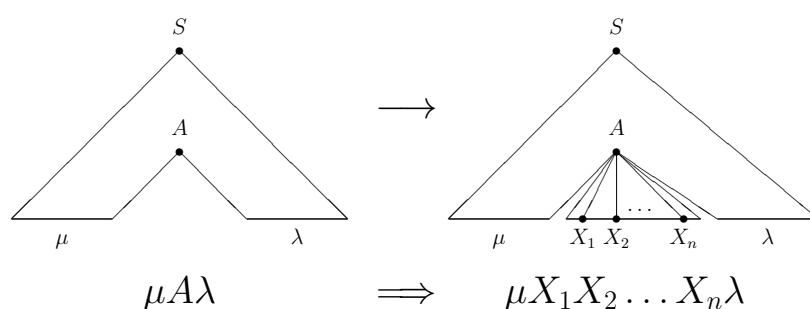
Poznámka 4.1 Uzel derivačního stromu, který je označen terminálním symbolem, musí být zřejmě koncovým uzlem derivačního stromu.

DEF

$x + y$



Obrázek 4.2: Derivační strom



Obrázek 4.3: Konstrukce derivačního stromu

Při konstrukci derivačního stromu k dané derivaci opakovaně aplikujeme bod (2) z definice 4.2. Tuto aplikaci ilustruje obr. 4.3.

Příklad 4.3 Uvažujme gramatiku

$$G = (\{\langle \text{výraz} \rangle, \langle \text{term} \rangle, \langle \text{faktor} \rangle\}, \{+, -, *, /, (,), i\}, P, \langle \text{výraz} \rangle),$$

které se často používá pro popis aritmetického výrazu s binárními operacemi $+$, $-$, $*$, $/$. Terminální symbol i odpovídá identifikátoru. Množina P obsahuje tato přepisovací pravidla:

$$\begin{aligned} \langle \text{výraz} \rangle &\rightarrow \langle \text{term} \rangle \mid \langle \text{výraz} \rangle + \langle \text{term} \rangle \mid \langle \text{výraz} \rangle - \langle \text{term} \rangle \\ \langle \text{term} \rangle &\rightarrow \langle \text{faktor} \rangle \mid \langle \text{term} \rangle * \langle \text{faktor} \rangle \mid \langle \text{term} \rangle / \langle \text{faktor} \rangle \\ \langle \text{faktor} \rangle &\rightarrow (\langle \text{výraz} \rangle) \mid i \end{aligned}$$

Jak lze snadno ukázat, větami jazyka $L(G)$ jsou například řetězce i , (i) , $i * i$, $i * i + i$, $i * (i + i)$.

Předpokládejme, že máme zkonstruovat derivační strom pro větnou formu $\langle \text{výraz} \rangle + \langle \text{term} \rangle * \langle \text{faktor} \rangle$. Nejprve ukažme, že tento řetězec je skutečně větnou formou:

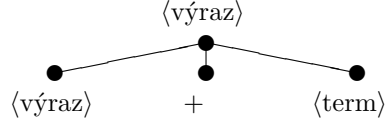
$$\langle \text{výraz} \rangle \Rightarrow \langle \text{výraz} \rangle + \langle \text{term} \rangle \Rightarrow \langle \text{výraz} \rangle + \langle \text{term} \rangle * \langle \text{faktor} \rangle$$

Derivační strom začínáme vytvářet od výchozího symbolu:

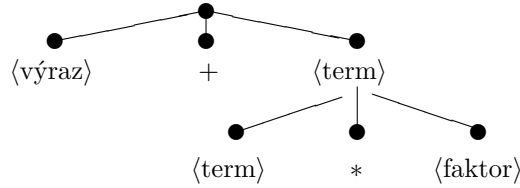
$$\langle \text{výraz} \rangle$$

První přímé derivaci odpovídá konstrukce:

$$x + y$$



Po znázornění druhé přímé derivace obdržíme výsledný derivační strom.



Je-li tedy dána derivace větné formy, pak této derivaci přísluší právě jeden derivační strom. Důkaz vyplývá z konstrukce derivačního stromu. Podívejme se nyní, zda uvedené tvrzení platí také obráceně: K derivačnímu stromu větné formy přísluší pouze jedna derivace. Uvažujme gramatiku G z příkladu 4.1.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aAb \mid ab \\ B &\rightarrow cBd \mid cd \end{aligned}$$

Na obr. 4.2 jsme znázornili derivační strom k derivaci

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbcd$$

Ukážeme nyní, že existují i jiné derivace věty $aabbcd$.

$$S \Rightarrow AB \Rightarrow Acd \Rightarrow aAbcd \Rightarrow aabbcd$$

nebo

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aAbcd \Rightarrow aabbcd$$

Uvedené derivace věty $aabbcd$ se liší v pořadí, v němž byly vybírány nonterminály pro přímé derivace. Toto pořadí však ve výsledném derivačním stromě není postiženo, a proto budou derivační stromy příslušející k 2. a 3. z uvedených derivací věty $aabbcd$ totožné s derivačním stromem na obr. 4.2. Neplatí tedy tvrzení, že k derivačnímu stromu přísluší pouze jedna derivace.

Poznamenejme, že jiné derivace věty $aabbcd$ v gramatice z příkladu 4.2 neexistují. V první a druhé z uvedených derivací byla přepisovací pravidla aplikována určitým kanonickým způsobem, který je charakteristický pro nejužívanější syntaktické analyzátoři překladačů programovacích jazyků. Zavedeme si pro tyto speciální derivace vlastní označení.

Definice 4.3 Necht $S = \alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n = \alpha$ je derivace větné formy α . Jestliže byl v každém řetězci α_i , $i = 1, \dots, n-1$ přepsán nejlevější (nejpravější) nonterminál, pak tuto derivaci nazýváme *levou* (*pravou*) derivací větné formy α .

První derivace věty $aabbcd$ je tedy derivací levou, druhá je derivací pravou.

Je-li $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n = w$ levá derivace věty w , pak je každé α_i , $i = 0, 1, \dots, n-1$ tvaru $x_i A_i \beta_i$, kde $x_i \in \Sigma^*$, $A_i \in N$ a $\beta_i \in (N \cup \Sigma^*)$. K získání větné formy α_{i+1} bude přepsán nonterminál A_i . Obrácená situace platí pro pravou derivaci.

DEF

4.2 Fráze větné formy

Definice 4.4 Nechť $G = (N, \Sigma, P, S)$ je gramatika a nechť řetězec $\lambda = \alpha\beta\gamma$ je větná forma. Podřetězec β se nazývá *frází větné formy* λ vzhledem k nonterminálu A z N , jestliže platí:

$$\begin{aligned} S &\Rightarrow^* \alpha A \gamma \\ A &\Rightarrow^+ \beta \end{aligned}$$

Podřetězec β je *jednoduchou frází větné formy* λ , jestliže platí:

$$\begin{aligned} S &\Rightarrow^* \alpha A \gamma \\ A &\Rightarrow \beta \end{aligned}$$

Příklad 4.4 Máme nalézt všechny fráze a všechny jednoduché větné formy $\langle \text{výraz} \rangle + \langle \text{term} \rangle * \langle \text{faktor} \rangle$ v gramatice, která popisuje aritmetický výraz. Jak již bylo uvedeno, derivace této větné formy má tvar:

$$\langle \text{výraz} \rangle \Rightarrow \langle \text{výraz} \rangle + \langle \text{term} \rangle \Rightarrow \langle \text{výraz} \rangle + \langle \text{term} \rangle * \langle \text{faktor} \rangle$$

Protože platí

$$\begin{aligned} \langle \text{výraz} \rangle &\Rightarrow^* \langle \text{výraz} \rangle + \langle \text{term} \rangle \quad \text{a} \\ \langle \text{term} \rangle &\Rightarrow \langle \text{term} \rangle * \langle \text{faktor} \rangle \end{aligned}$$

je řetězec $\langle \text{term} \rangle * \langle \text{faktor} \rangle$ jednoduchou frází větné formy $\langle \text{výraz} \rangle + \langle \text{term} \rangle * \langle \text{faktor} \rangle$ vzhledem k nonterminálu $\langle \text{term} \rangle$.

Dále je:

$$\begin{aligned} \langle \text{výraz} \rangle &\Rightarrow^* \langle \text{výraz} \rangle \\ \langle \text{výraz} \rangle &\Rightarrow^+ \langle \text{výraz} \rangle + \langle \text{term} \rangle * \langle \text{faktor} \rangle \end{aligned}$$

a z toho vyplývá, že větná forma $\langle \text{výraz} \rangle + \langle \text{term} \rangle * \langle \text{faktor} \rangle$ je frází sama k sobě vzhledem k výchozímu symbolu. Tato skutečnost je důsledkem triviálního případu v definici fráze, kdy jsou řetězce α a γ prázdné. Jiné fráze větné formy $\langle \text{výraz} \rangle + \langle \text{term} \rangle * \langle \text{faktor} \rangle$ neexistují.

Pojem fráze je stěžejním pojmem pro syntaktickou analýzu. Celá třída syntaktických analyzátorů je postavena na metodách hledání nejlevější jednoduché fráze větné formy (věty). Protože dále budeme pojmu nejlevější jednoduchá fráze často používat, zavedeme pro něj speciální označení, *l-fráze*.

Ve větné formě $\langle \text{výraz} \rangle + \langle \text{term} \rangle * \langle \text{faktor} \rangle$ je jediná jednoduchá fráze: $\langle \text{term} \rangle * \langle \text{faktor} \rangle$. Tato fráze je tedy zároveň l-frází.

S pojmem fráze větné formy je velmi úzce svázán pojem podstrom příslušného derivačního stromu. Podstromem derivačního stromu budeme rozumět tu část tohoto stromu, která je vymezena jistým uzlem, tzv. kořenem podstromu, spolu se všemi uzly, které jsou z kořene podstromu dostupné prostřednictvím příslušných hran, včetně těchto hran.

Příklad 4.5 Podstromy derivačního stromu věty $aabbcd$ z obr. 4.2 jsou stromy z obr. 4.4.

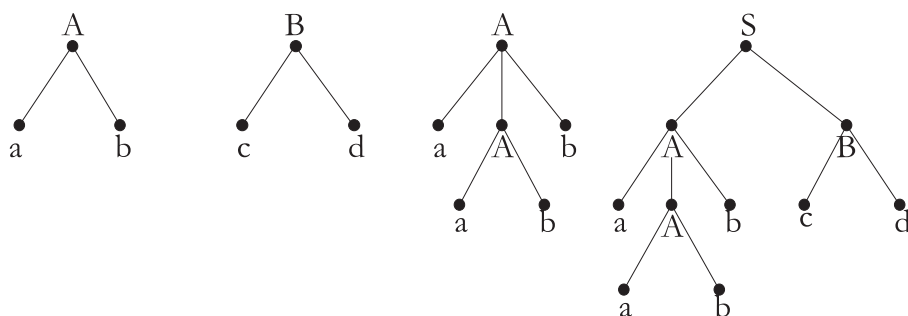
Předpokládejme nyní, že nonterminál A je kořenem podstromu derivačního stromu. Je-li β řetězec koncových uzlů tohoto podstromu, pak jistě platí $A \Rightarrow^+ \beta$.

DEF

 $x + y$

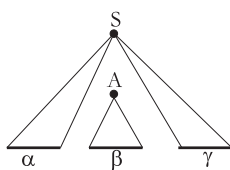
!

 $x + y$



Obrázek 4.4: Podstromy derivačního stromu

Nechť α je řetězec koncových uzlů vlevo, γ řetězec koncových uzlů derivačního stromu vpravo od β . Pak platí $S \Rightarrow^* \alpha\beta\gamma$; S je kořenem derivačního stromu. To ovšem znamená, že β je frází větné formy $\alpha\beta\gamma$ vzhledem k nonterminálu A . Situaci ilustruje obr. 4.5.



Obrázek 4.5: Fráze větné formy

Podstrom derivačního stromu tedy odpovídá frázi příslušné větné formy. Fráze je tvořena koncovými uzly podstromu. Jednoduchá fráze odpovídá podstromu, jenž je výsledkem přímé derivace $A \Rightarrow \beta$.

Příklad 4.6 V gramatice z příkladu 4.1 nalezněte fráze věty $a^2b^2c^2d^2$. Nejdříve sestavíme derivační strom. Pro jeho konstrukci vytvořme (např.) levou derivaci této věty:

$$S \Rightarrow AB \Rightarrow aAbB \Rightarrow aabbB \Rightarrow aabbcBd \Rightarrow aabbccdd$$

Vyznačené podstromy odpovídají dále uvedeným frázím definovaným k příslušným nonterminálům. Horní indexy slouží k rozlišení výskytu téhož nonterminálu (viz 4.6)

Fráze	Nonterminál
$aabbccdd$	S
$aabb$	A^1
ab	A^2
$ccdd$	B^1
cd	B^2

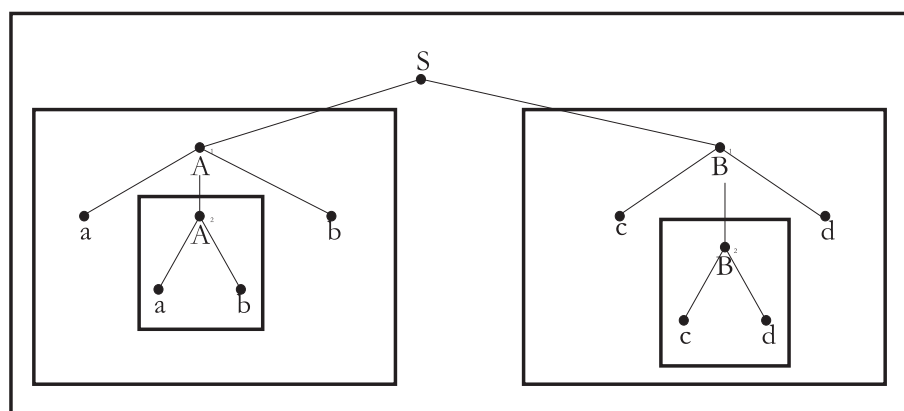
Fráze ab a cd jsou jednoduché, ab je 1-fráze.

4.3 Víceznačnost gramatik

Definice 4.5 Necht G je gramatika. Říkáme, že věta w generovaná gramatikou

$x + y$

DEF



Obrázek 4.6: Podstromy derivačního stromu

G je *víceznačná*, existují-li alespoň dva různé derivační stromy s koncovými uzly tvořícími větu w . *Gramatika G je víceznačná*, jestliže generuje alespoň jednu víceznačnou větu. V opačném případě mluvíme o jednoznačné gramatice.

Povšimněte si, že definujeme víceznačnou gramatiku, nikoli víceznačný jazyk. V mnoha případech lze vhodnými transformacemi víceznačné gramatiky odstranit víceznačnost vět, aniž se samozřejmě změní jazyk generovaný získanou jednoznačnou gramatikou. Existují však jazyky, které nelze generovat jednoznačnou gramatikou. Takové jazyky jsou pak nazývány jazyky s *inherentní víceznačností*.

Příklad 4.7 Uvažujme gramatiku $G = (\{E\}, \{+, -, *, /, (,), i\}, P, E)$, kde P je množina pravidel

$$E := E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid i$$

$x + y$

Jazyk $L(G)$ je totožný s jazykem generovaným gramatikou z příkladu 4.3 a je tvořen aritmetickými výrazy s binárními operacemi.

Tato gramatika je na rozdíl od gramatiky z příkladu 4.3 *víceznačná*. Vezměme například větu $i + i * i$ a uvažujme všechny možné derivační stromy příslušející k této větě (viz obr. 4.7).

Existence dvou různých derivačních stromů k větě $i + i * i$ dokazuje, že gramatika G je *víceznačná*. Označuje-li se $+$ sečítání a $*$ násobení, pak není jasné, zda první operací bude násobení (první derivační strom), a nebo sečítání (druhý derivační strom). Jednoznačná gramatika z příkladu 4.3 na druhé straně respektuje obvyklou prioritou operací (násobení má přednost před sečítáním), jak je patrné z jediného derivačního stromu věty $i + i * i$ na obr. 4.8

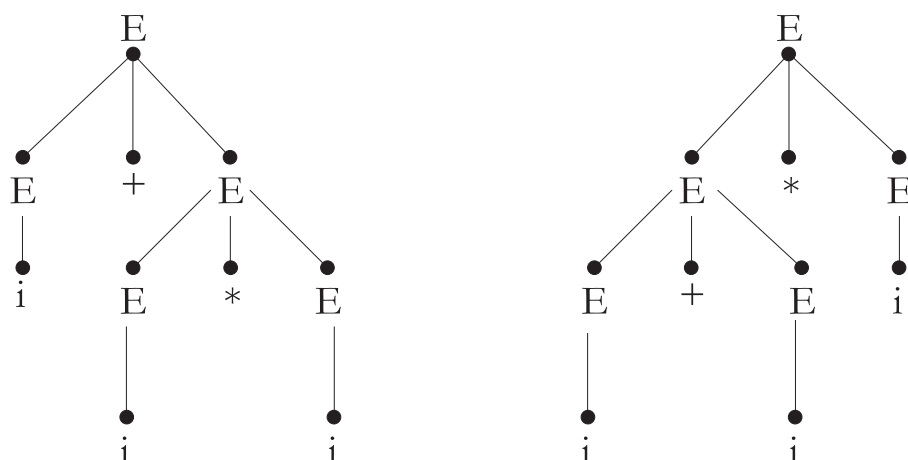
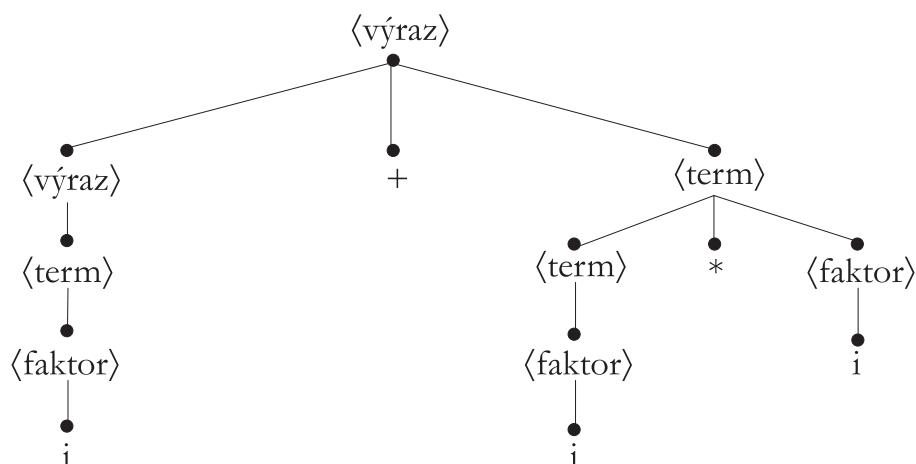
Pro bezkontextové gramatiky bylo dokázáno, že problém, zda daná gramatika je a nebo není *víceznačná*, je nerozhodnutelný, tj. neexistuje algoritmus, který by v konečném čase odhalil *víceznačnost* každé bezkontextové gramatiky.

Příklad 4.8 Gramatika obsahující pravidlo tvaru $A \rightarrow A$ je zřejmě *víceznačná*. Toto pravidlo můžeme vypustit, aniž změníme jazyk generovaný takto zredukovanou gramatikou.

$x + y$

Poznámka 4.2 *Víceznačnost gramatiky, pokud negeneruje jazyk s inherentní víceznačností, je obecně pokládána za negativní rys, poněvadž vede k větám,*



Obrázek 4.7: Derivační stromy věty $i + i * i$ Obrázek 4.8: Derivační strom věty $i + i * i$ v G_2

jež mají několik interpretací. Na druhé straně však víceznačná gramatika může být jednodušší, než odpovídající jednoznačná gramatika (viz příklad 4.7). Této skutečnosti se někdy využívá při konstrukci překladačů takovým způsobem, že se použije víceznačné gramatiky a nežádoucí interpretace víceznačné věty se vyloučí dodatečným sémantickým pravidlem.

Příklad 4.9 Jedním z nejznámějších příkladů víceznačnosti v programovacích jazycích jsou konstrukce s **then** a **else**. Skutečně, přepisovací pravidla

$$\begin{aligned}
 S &\rightarrow \text{if } b \text{ then } S \text{ else } S \\
 S &\rightarrow \text{if } b \text{ then } S \\
 S &\rightarrow p
 \end{aligned}$$

kde b značí booleovský výraz a p jiný, než podmíněný příkaz, vedou k víceznačné interpretaci podmíněného příkazu. Např. k větě

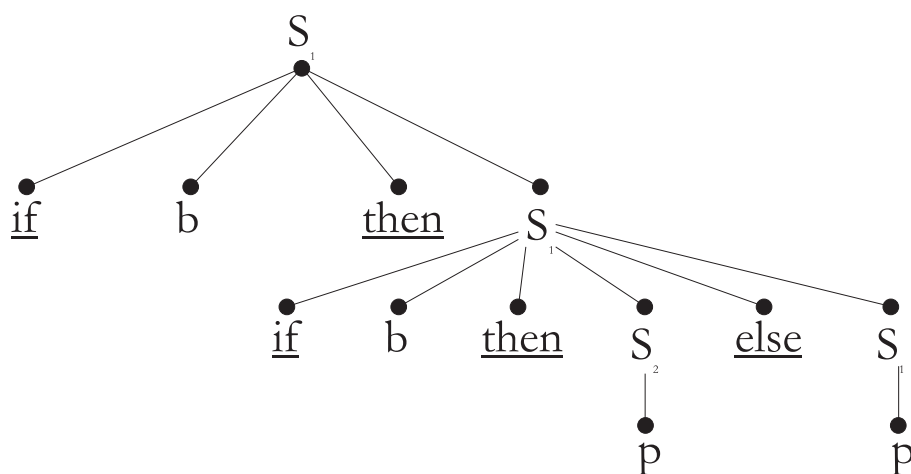
$$\text{if } b \text{ then if } b \text{ then } p \text{ else } p$$

$x + y$

existují dva různé derivační stromy. Zatímco Algol 60 tuto konstrukci nedovoloval (za **then** musel být složený příkaz), jazyk Pascal uvedenou víceznačnost řeší sémantickým pravidlem: „k danému **else** se vztahuje nejbližší předchozí **then**“. Poznamenejme, že i toto pravidlo lze postihnout také syntakticky:

$$\begin{aligned} S_1 &\rightarrow \text{if } b \text{ then } S_1 \mid \text{if } b \text{ then } S_2 \text{ else } S_1 \mid p \\ S_2 &\rightarrow \text{if } b \text{ then } S_2 \text{ else } S_2 \mid p \end{aligned}$$

S použitím těchto přepisovacích pravidel obdržíme pro větu **if** *b* **then** **if** *b* **then** *p* **else** *p* jediný derivační strom na obr. 4.9.



Obrázek 4.9: Derivační strom podmíněného příkazu

Příklad 4.10 Nalezněte bezkontextovou gramatiku, která generuje správně vytvořené regulární výrazy nad abecedou $\{a, b\}$.

Řešení: Velmi jednoduché řešení tohoto příkladu nabízí rekursivní definice regulární množiny 3.9 $G_{RV}^0 = (\{E\}, \{a, b, \epsilon, +, *, (,)\}, P, E)$, kde

E je jediný nonterminál popisující syntaktickou kategorii regulární výraz

ϵ značí prázdný řetězec jako součást výrazu

$+$ odpovídá operaci sjednocení regulárních množin

$*$ odpovídá operaci iterace regulární množiny

Konkatenace se explicitně nevyznačuje. Pravidla tvořící množinu P pak vytváříme na základě definice regulární množiny:

$$E \rightarrow a \mid b \mid \epsilon \mid E + E \mid EE \mid E^* \mid (E)$$

Sestrojená gramatika G_{RV}^0 je jednoduchá, má však vlastnost, která je obvykle nežádoucí – víceznačnost. Abychom vytvořili jednoznačnou gramatiku, budeme využívat analogii s gramatikou pro aritmetický výraz. Kromě nejvyšší syntaktické kategorie R popisující celý regulární výraz, zavedeme další nonterminály:

K pro podvýrazy tvořené operací konkatenace

I pro podvýrazy tvořené operací iterace

P pro primitivní regulární výrazy

Výsledná gramatika pak může mít tvar $G_{RV} = (\{R, K, I, P\}, \{a, b, \epsilon, +, *, (,)\}, PP, R)$, kde množina PP přepisovacích pravidel obsahuje pravidla

$$\begin{aligned} R &\rightarrow R + K \mid K \\ K &\rightarrow KI \mid I \\ I &\rightarrow I^* \mid P \\ P &\rightarrow a \mid b \mid \epsilon \mid (R) \end{aligned}$$

4.4 Rozklad věty

Konstrukci derivace či derivačního stromu pro danou větu nebo větnou formu nazýváme *rozkladem* nebo *syntaktickou analýzou* této věty nebo větné formy. Program, který provádí rozklad vět určitého jazyka, se nazývá *syntaktický analyzátor* (anglicky také parser, to parse = rozložit).

Algoritmy syntaktické analýzy lze rozdělit podle způsobu, kterým je konstruována derivace věty, tj. vytvářen derivační strom, do těchto dvou základních skupin:

- syntaktická analýza shora dolů
- syntaktická analýza zdola nahoru

x + y

Při syntaktické analýze shora dolů začínáme derivační strom budovat od výchozího symbolu (kořene derivačního stromu) a postupnými přímými derivacemi dojdeme k terminálním symbolům, které tvoří analyzovanou větu (koncovým uzlům derivačního stromu). Problém spočívá ve správnosti volby přímých derivací, tj. pořadí používání přepisovacích pravidel.

Při syntaktické analýze zdola nahoru začínáme derivační strom budovat od koncových uzlů a postupnými přímými redukcemi dojdeme ke kořenu (výchozímu symbolu gramatiky).

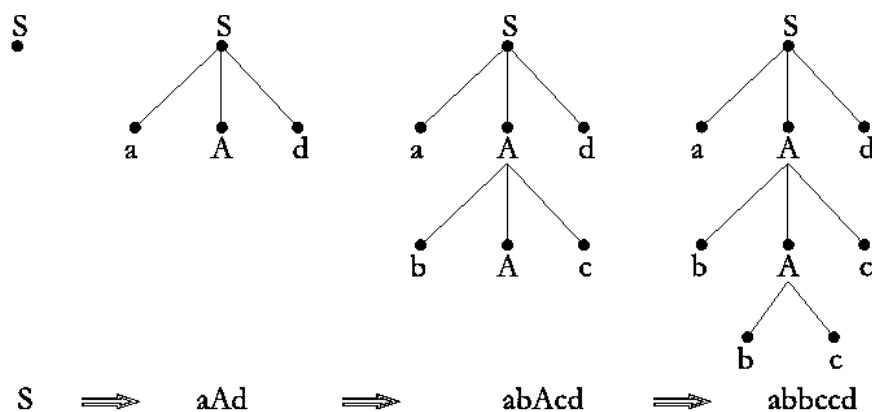
Základním problémem této třídy syntaktických analyzátorů je hledání prvního podřetězce věty (v dalších krocích větných forem), který může být redukován k jistému nonterminálu – kořenu podstromu derivačního stromu. Tento podřetězec, jak již víme, se nazývá l-fráze.

Příklad 4.11 Odlišnost obou typů syntaktické analýzy ilustrujeme na příkladě gramatiky, která generuje jazyk $L = \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$. Tato gramatika má pravidla

$$\begin{aligned} S &\rightarrow aSd \mid aAd \\ A &\rightarrow bAc \mid bc \end{aligned}$$

kde S je výchozí symbol.

Na obr. 4.10 je uvedena konstrukce derivačního stromu metodou shora dolů spolu s odpovídající levou derivací věty $abbccd$. Na obr. 4.11 je znázorněna konstrukce



Obrázek 4.10: Syntaktická analýza shora dolů

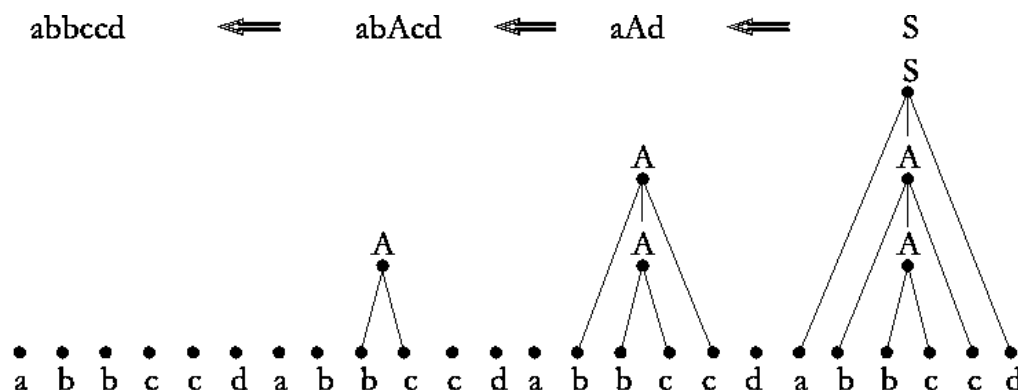
derivačního stromu metodou zdola nahoru. Odpovídající pravá derivace je zapsána zprava doleva.

Vraťme se opět k základním problémům syntaktické analýzy. Při analýze shora dolů konstruuujeme levou derivaci věty. Předpokládejme, že v jistém kroku analýzy je A nejlevějším nonterminálem, který má být přepsán. Dále předpokládejme, že gramatika obsahuje n pravidel s levou stranou A :

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

Jak poznáme, kterým řetězcem α_i je třeba nahradit nonterminál A ? Podobně při analýze zdola nahoru, kdy je v každém kroku redukována l-fráze větné formy, spočívá hlavní problém v určení začátku a konce l-fráze.

 $x + y$



Obrázek 4.11: Syntaktická analýza zdola nahoru

Jedním z řešení těchto problémů je náhodný výběr některé z možných alternativ. Ukáže-li se později, že zvolená alternativa nebyla správná, je třeba proces rozkladu „vrátit“ a uvažovat jinou alternativu. (Při syntaktické analýze shora dolů se například pokoušíme přepisovat A řetězci $\alpha_1, \alpha_2 \dots$ tak, aby se prefix získané levé derivace, který obsahuje pouze terminální symboly, shodoval s prefixem analyzované věty). Tento typ analýzy se nazývá *syntaktická analýza s návratem* (Syntax analysis with backup). I když počet návratů je omezený, je patrné, že analýza s návratem je časově náročná. Kromě toho jsou návraty zdrojem komplikací při sémantickém vyhodnocování překládaného programu.

Praktickým řešením problémů syntaktické analýzy programovacích jazyků jsou tzv. deterministické gramatiky (kapitola 6.), které umožňují na základě kontextu zpracovávaného podřetězce větné formy, určit správnou alternativu v každém kroku analýzy. Tento typ syntaktické analýzy se nazývá *deterministická syntaktická analýza* (deterministický rozklad) nebo *syntaktická analýza bez návratu*. Pro ilustraci role kontextu uvažujme větu $i + i * i$ v gramatice z příkladu 4.3. Po zpracování podřetězce $i + i$ nám kontext reprezentovaný terminálem $*$ pomůže určit frázi, kterou není $i + i$, ale podřetězec $i * i$.

4.5 Transformace bezkontextových gramatik

Obecně neexistuje algoritmická metoda, která by umožňovala sestavit gramatiku generující jazyk s libovolnou strukturou. Teorie vyčíslitelnosti dokonce ukazuje, že nekonečně mnoho formálních jazyků nelze popsat žádným konečným formálním systémem, tudíž ani gramatikou. Existuje však celá řada užitečných transformací gramatik, které dovolují modifikovat gramatiku, aniž by byl porušen generovaný jazyk. V tomto odstavci popíšeme některé z těchto transformací formou matematických zápisů příslušných algoritmů.

Definice 4.6 Říkáme, že gramatiky G_1 a G_2 jsou *ekvivalentní*, jestliže platí $L(G_1) = L(G_2)$, tj. jestliže jimi generované jazyky jsou totožné.

V některých případech se může stát, že sestavená gramatika obsahuje zbytečné symboly a nadbytečná přepisovací pravidla. Nehledě k tomu, že tato skutečnost může být důsledkem chyby v konstrukci gramatiky, je velmi pravděpodobné, že syntaktická analýza bude probíhat méně efektivně. Je proto důležité odstranit z gramatiky zbytečné symboly a nadbytečná pravidla.

DEF

Uvažujme například gramatiku $G = (\{S, A\}, \{a, b\}, P, S)$, kde $P = \{S \rightarrow a, A \rightarrow b\}$. Je zřejmé, že nonterminál A a terminál b se nemohou objevit v žádné větě. Proto je můžeme z gramatiky G odstranit spolu s nadbytečným pravidlem $A \rightarrow b$, aniž se změní jazyk $L(G)$.

Definice 4.7 Necht $G = (N, \Sigma, P, S)$ je gramatika. Říkáme, že symbol $X \in (N \cup \Sigma)$ je v gramatice G *zbytečný*, jestliže neexistuje derivace tvaru $S \Rightarrow^* wXy \Rightarrow^* wxy$. Řetězce w, x, y jsou v Σ^* .

Abychom zjistili, zda nonterminál A je zbytečný, popíšeme algoritmus určující, může-li se z nonterminálu A generovat řetězec terminálních symbolů, tj. zjišťujeme, zda $\{w \mid A \Rightarrow^* w, w \in \Sigma^*\} = \emptyset$

Tento algoritmus lze rozšířit na algoritmus, který zjišťuje, je-li jazyk, generovaný bezkontextovou gramatikou, neprázdný.

Algoritmus 4.1 Je $L(G)$ neprázdný?

Vstup: gramatika $G = (N, \Sigma, P, S)$.

Výstup: ANO je-li $L(G) \neq \emptyset$, NE v opačném případě.

Metoda: Sestrojíme množiny N_0, N_1, \dots rekurzivně takto

- (1) $N_0 = \emptyset, i = 1$
- (2) $N_i = \{A \mid A \rightarrow \alpha \text{ je v } P \wedge \alpha \in (N_{i-1} \cup \Sigma)^*\} \cup N_{i-1}$
- (3) Je-li $N_i \neq N_{i-1}$, polož $i = i + 1$ a vrať se ke kroku 2. Je-li $N_i = N_{i-1}$, polož $N_t = N_i$
- (4) Jestliže výchozí symbol S je v N_t , pak je výstup ANO, jinak NE.

Poznámka 4.3 Jestliže množina nonterminálů N má n prvků, pak, protože $N_t \subseteq N$, musí algoritmus 4.1 skončit maximálně po $n + 1$ iteracích kroku 2.

Věta 4.1 Algoritmus 4.1 má výstup ANO, právě když $S \Rightarrow^* w$ pro nějaké $w \in \Sigma^*$

Důkaz: Nejprve dokážeme indukcí pro i implikaci:

$$\text{Jestliže platí } A \in N_i, \text{ pak } A \Rightarrow^* w \text{ pro nějaké } w \in \Sigma^* \quad (1)$$

Pro $i = 0$ implikace platí, protože $N_0 = \emptyset$. Předpokládejme, že (1) platí pro i , a že A je prvkem množiny N_{i+1} . Je-li zároveň $A \in N_i$, pak je induktivní krok triviální. Jestliže A leží v $N_{i+1} - N_i$, pak existuje pravidlo $A \rightarrow X_1 \dots X_k$, kde X_j je buď terminální symbol, nebo $X_j \in N_i, j = 1, \dots, k$. Pro každé j tedy existuje řetězec $w_j \in \Sigma^*$ takový, že $X_j \Rightarrow^* w_j$ a platí tak

$$A \Rightarrow X_1 \dots X_k \Rightarrow^* w_1 X_2 \dots X_k \Rightarrow^* \dots \Rightarrow^* w_1 \dots w_k = w$$

Opačnou implikaci:

$$\text{Jestliže } A \Rightarrow^n w, \text{ pak } A \in N_i \text{ pro nějaké } i \quad (2)$$

dokážeme opět indukcí.

Je-li $n = 1$, pak zřejmě $i = 1$. Předpokládejme, že (2) platí pro n , a že $A \Rightarrow^{n+1} w$. Pak můžeme psát $A \Rightarrow X_1 \dots X_k \Rightarrow^n w$, kde $w = w_1 \dots w_k$ a $X_j \Rightarrow^{n_j} w_j$ pro $j = 1, \dots, k, n_j \leq n$. (Podřetězce w_j jsou fráze řetězce w vzhledem k nonterminálům X_j v případě, že $X_j \in N$, pak $X_j \in N_{i_j}$ pro nějaké i_j . Položme $i_j = 0$, jestliže $X_j \in \Sigma$. Necht $i = 1 + \max(i_1, \dots, i_k)$. Pak, podle definice, $A \in N_i$. Položíme-li nyní $A = S$ v implikacích (1) a (2), dostáváme důkaz věty 4.1. \square

DEF

!

◀

Definice 4.8 Říkáme, že symbol $X \in (N \cup \Sigma)$ je *nedostupný* v gramatice $G = (N, \Sigma, P, S)$, jestliže X se nemůže objevit v žádné větné formě.

Algoritmus 4.2 Odstranění nedostupných symbolů

Vstup: Gramatika $G = (N, \Sigma, P, S)$.

Výstup: Gramatika $G' = (N', \Sigma', P', S)$, pro kterou platí

- (i) $L(G') = L(G)$
- (ii) Pro všechna X z $(N' \cup \Sigma')$ existují řetězce α a β z $(N' \cup \Sigma')^*$ tak, že $S \Rightarrow^* \alpha X \beta$ v gramatice G' .

Metoda:

- (1) Položíme $V_0 = \{S\}$ a $i = 1$
- (2) Konstruujeme $V_i = \{X \mid A \rightarrow \alpha X \beta \in P \wedge A \in V_{i-1}\} \cup V_{i-1}$
- (3) Je-li $V_i \neq V_{i-1}$, polož $i = i + 1$ a vrať se ke kroku 2.
Je-li $V_i = V_{i-1}$, pak
 $N' = V_i \cap N$
 $\Sigma' = V_i \cap \Sigma$
 $P' \subseteq P$ obsahuje ta pravidla, která jsou tvořena pouze symboly z V_i .

Algoritmus 4.2 je podobný algoritmu 4.1. Protože je $V_i \subset N \cup \Sigma$, je počet opakování bodu (2) algoritmu 4.2 konečný. Důkaz algoritmu se provede indukcí pro i této ekvivalence:

$$S \Rightarrow^* \alpha X \beta, \text{ právě když } X \in V_i \text{ pro nějaké } i.$$

Nyní zformulujeme algoritmus pro odstranění zbytečných symbolů.

Algoritmus 4.3 Odstranění zbytečných symbolů.

Vstup: Gramatika $G = (N, \Sigma, P, S)$ generující neprázdný jazyk.

Výstup: Gramatika $G' = (N', \Sigma', P', S)$, pro kterou platí:

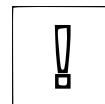
- (i) $L(G) = L(G')$
- (ii) Žádný symbol v $N' \cup \Sigma'$ není zbytečný

Metoda:

- (1) Na gramatiku G aplikuj algoritmus 4.1 s cílem získat množinu N_t . Polož $\bar{G} = (N_t, \Sigma, P_1, S)$, kde P_1 obsahuje pravidla tvořené pouze symboly z $N_t \cup \Sigma$
- (2) Algoritmus 4.2 aplikuj na gramatiku \bar{G} . Výsledkem je gramatika $G' = (N', \Sigma', P', S)$, která neobsahuje zbytečné symboly.

V kroku (1) algoritmu 4.3 jsou z G odstraněny všechny nonterminály, které nemohou generovat terminální řetězce. V kroku (2) jsou pak odstraněny všechny symboly, které jsou nedostupné. Pořadí použití algoritmů 4.1 a 4.2 je podstatné, obrácené pořadí nemusí vždy vést ke gramatice bez zbytečných symbolů.

Věta 4.2 Gramatika G' z algoritmu 4.3 je ekvivalentní gramatice G a nemá zbytečné symboly.



Důkaz: Dokážeme sporem, že gramatika G' nemá zbytečné symboly. Předpokládejme, že $A \in N'$ je zbytečný symbol. Podle definice zbytečných symbolů musíme uvažovat dva případy:

1. Neplatí $S \xRightarrow{G'}^* \alpha A \beta$ pro všechna α a β . V tomto případě se však dostáváme do sporu s krokem (2) algoritmu 4.3.
2. Platí $S \xRightarrow{G'}^* \alpha A \beta$ pro nějaká α a β , avšak neplatí $A \xRightarrow{G'}^* w, w \in \Sigma'^*$. Pak A nebude odstraněn v kroku (2) algoritmu 4.3 a navíc, platí-li $A \xRightarrow{G}^* \gamma B \delta$, pak ani B nebude odstraněn v kroku (2). Platí-li však $A \xRightarrow{G}^* w$, pak také platí $A \xRightarrow{G'}^* w$ a tedy neplatí-li $A \xRightarrow{G'}^* w$, pak neplatí ani $A \xRightarrow{G}^* w$, což je spor s krokem (1) algoritmu 4.3.

Důkaz, že G' neobsahuje ani zbytečný terminál se provede obdobně. \square

Příklad 4.12 Uvažujme gramatiku $G = (\{S, A, B\}, \{a, b\}, P, S)$, kde P obsahuje pravidla:

$$\begin{aligned} S &\rightarrow a \mid A \\ A &\rightarrow AB \\ B &\rightarrow b \end{aligned}$$

Aplikujeme-li na G algoritmus 4.3, pak v kroku 1 získáme $N_t = \{S, B\}$, takže $\bar{G} = (\{S, B\}, \{a, b\}, \{S \rightarrow a, B \rightarrow b\}, S)$. Po aplikování algoritmu 4.2 na \bar{G} obdržíme $V_2 = V_1 = \{S, a\}$. Výsledkem je tedy ekvivalentní gramatika

$$G' = (\{S\}, \{a\}, \{S \rightarrow a\}, S).$$

Kdybychom jako první aplikovali algoritmus 4.2, zjistíme, že všechny symboly v G jsou dostupné a algoritmus 4.2 tedy gramatiku G nezmění. Po aplikování algoritmu 4.1 získáme $N_t = \{S, B\}$, takže výsledná gramatika bude \bar{G} a ne G' .

Další důležitou transformací, kterou nyní popíšeme, je transformace odstraňující z gramatiky pravidla tvaru $A \rightarrow \epsilon$ (ϵ je prázdný řetězec), tzv. ϵ -pravidla. Jestliže ovšem $L(G)$ má obsahovat také prázdný řetězec, pak není možné aby G neobsahovala žádné ϵ -pravidlo. Následující definice gramatiky bez ϵ -pravidla respektuje tuto skutečnost.

Definice 4.9

Říkáme, že gramatika $G = (N, \Sigma, P, S)$ je *gramatikou bez ϵ -pravidel*, jestliže buď P neobsahuje žádné ϵ -pravidlo, nebo, v případě $\epsilon \in L(G)$, existuje jediné ϵ -pravidlo tvaru $S \rightarrow \epsilon$ a výchozí symbol S se nevyskytuje na pravé straně žádného pravidla z P .

Algoritmus 4.4 Transformace na gramatiku bez ϵ -pravidel.

Vstup: Gramatika $G = (N, \Sigma, P, S)$

Výstup: Ekvivalentní gramatika $G' = (N', \Sigma', P', S')$ bez ϵ -pravidel.

Metoda:

- (1) Sestroj $N_\epsilon = \{A \mid A \in N \text{ a } A \Rightarrow^* \epsilon\}$. Konstrukce množiny N_ϵ je analogická konstrukci N_t z 4.1.

x + y

DEF



(2) Necht P' je množina pravidel, kterou konstruujeme takto:

- a) Jestliže $A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \dots B_k \alpha_k$ je v P , $k \geq 0$ a každé B_i je v N_ϵ , $1 \leq i \leq k$, avšak žádný ze symbolů řetězců α_j není v N_ϵ , $0 \leq j \leq k$, pak k P' přidej všechna nová pravidla tvaru

$$A \rightarrow \alpha_0 X_1 \alpha_1 X_2 \dots X_k \alpha_k$$

kde X_i je buď B_i nebo ϵ . Nepřidávej ϵ -pravidlo $A \rightarrow \epsilon$, které se objeví, jsou-li všechna $\alpha_i = \epsilon$.

- b) Jestliže $S \in N_\epsilon$ pak k P' přidej pravidla

$$S' \rightarrow \epsilon \mid S$$

S' je nový výchozí symbol. Polož $N' = N \cup \{S'\}$. Jestliže $S \notin N_\epsilon$, pak $N' = N$ a $S' = S$

(3) Výsledná gramatika má tvar $G' = (N', \Sigma', P', S')$

Příklad 4.13 Uvažujme gramatiku $G = (\{S\}, \{a, b\}, P, S)$, kde P obsahuje pravidla $S \rightarrow aSbS \mid bSaS \mid \epsilon$. Po aplikování algoritmu 4.4 získáme gramatiku G' bez ϵ -pravidel. $G' = (\{S', S\}, \{a, b\}, P', S')$, kde P' obsahuje pravidla

$$\begin{aligned} S' &\rightarrow S \mid \epsilon \\ S &\rightarrow aSbS \mid abS \mid aSb \mid ab \mid bSaS \mid baS \mid bSa \mid ba \end{aligned}$$

Věta 4.3 Algoritmus 4.3 převádí vstupní gramatiku G na ekvivalentní gramatiku G' bez ϵ -pravidel. Důkaz: G' zřejmě neobsahuje ϵ -pravidla kromě případného pravidla $S \rightarrow \epsilon$, je-li $\epsilon \in L(G)$. Důkaz, že $L(G) = L(G')$ se provede indukcí pro délku řetězce w v ekvivalenci

$$A \xRightarrow{G'}^* w \text{ právě když } w \neq \epsilon \wedge A \xRightarrow{G}^* w$$

Jinou užitečnou transformací je odstranění pravidel tvaru $A \rightarrow B$. □

Definice 4.10 Přepisovací pravidlo tvaru $A \rightarrow B$, $A, B \in N$ se nazývá *jednoduché pravidlo*.

Algoritmus 4.5 Odstranění jednoduchých pravidel.

Vstup: Gramatika G bez ϵ -pravidel.

Výstup: Ekvivalentní gramatika G' bez jednoduchých pravidel.

Metoda:

(1) Pro každé $A \in N$ sestroj množinu $N_A = \{B \mid A \xRightarrow{*} B\}$ takto:

- a) $N_0 = \{A\}$, $i = 1$
b) $N_i = \{C \mid B \rightarrow C \text{ je v } P \text{ a } B \in N_{i-1}\} \cup N_{i-1}$
c) Jestliže $N_i \neq N_{i-1}$, polož $i = i + 1$ a opakuj krok b). V opačném případě je $N_A = N_i$.

(2) Sestroj P' takto: Jestliže $B \rightarrow \alpha$ je v P a není jednoduchým pravidlem, pak pro všechna A , pro něž $B \in N_A$, přidej k P' pravidla $A \rightarrow \alpha$

(3) Výsledná gramatika je $G = (N, \Sigma, P', S)$

x + y

DEF

!

Příklad 4.14 Uvažujme gramatiku s přepisovacími pravidly:

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid i \end{aligned}$$

$x + y$

Tato gramatika se liší od gramatiky z příkladu 4.3 pouze jiným značením nonterminálu a generuje tudíž stejný jazyk aritmetických výrazů

Po aplikování algoritmu 4.5 bude

$$\begin{aligned} N_E &= \{E, T, F\} \\ N_T &= \{T, F\} \\ N_F &= \{F\} \end{aligned}$$

a výsledná množina přepisovacích pravidel, neobsahující jednoduchá pravidla, bude:

$$\begin{aligned} E &\rightarrow E + T \mid T * F \mid (E) \mid i \\ T &\rightarrow T * F \mid (E) \mid i \\ F &\rightarrow (E) \mid i \end{aligned}$$

Věta 4.4 Algoritmus 4.4 převádí vstupní gramatiku G na ekvivalentní gramatiku G' bez jednoduchých pravidel. Důkaz: Gramatika G' zřejmě neobsahuje jednoduchá pravidla. Abychom ukázali, že $L(G) = L(G')$, dokážeme že platí $L(G') \subseteq L(G)$ a také $L(G) \subseteq L(G')$.

1. $L(G') \subseteq L(G)$

Nechť $w \in L(G')$. Pak existuje v G' derivace $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n = w$.

Jestliže k derivaci $\alpha_i \xRightarrow{G'} \alpha_{i+1}$ bylo použito pravidla $A \rightarrow \beta$, pak existuje nonterminál B ($A = B$ případně) takový, že $A \xRightarrow{G}^* B$ a $B \xRightarrow{G} \beta$ a tedy $A \xRightarrow{G}^* \beta$ a $\alpha_i \xRightarrow{G}^* \alpha_{i+1}$. Z toho plyne, že platí $S \xRightarrow{G}^* w$ a w je tudíž v $L(G)$.

2. $L(G) \subseteq L(G')$

Nechť $w \in L(G)$ a $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n = w$ je levá derivace řetězce w v gramatice G . Nechť i_1, i_2, \dots, i_k je posloupnost indexů tvořená pouze těmi j , pro které v levé derivaci $\alpha_{j-1} \Rightarrow \alpha_j$ nebylo použito jednoduchého pravidla. Speciálně $i_k = n$, protože poslední aplikované pravidlo v derivaci řetězce w nemůže být jednoduché. Derivace řetězce w v G je levou derivací, a proto aplikací jednoduchých pravidel nahrazujeme pouze nejlevější nonterminál jiným nonterminálem (reprezentuje levou stranu pravidla v G'). Platí tedy $S \xRightarrow{G'} \alpha_{i_1} \xRightarrow{G'} \dots \xRightarrow{G'} \alpha_{i_k} = w$ a tudíž $w \in L(G')$.

Dokázali jsme tak ekvivalenci gramatik G a G' . □

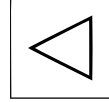
Definice 4.11 Říkáme, že G je *gramatika bez cyklu*, jestliže v ní neexistuje derivace tvaru $A \Rightarrow^+ A$ pro žádné A z N . Jestliže G je gramatika bez cyklu a bez ϵ -pravidel a nemá žádné zbytečné symboly, pak říkáme, že G je *vlastní gramatika*.

DEF

Věta 4.5 Je-li L bezkontextový jazyk, pak existuje vlastní gramatika G taková, že $L = L(G)$.

Důkaz: Vyplývá z algoritmů 4.1–4.5. Gramatika, jež neobsahuje ϵ -pravidla a jednoduchá pravidla, je zřejmě gramatikou bez cyklu. \square

Poznámka 4.4 Gramatika, která obsahuje cykly, je víceznačná a nemůže být napr. použita pro konstrukci deterministického syntaktického analyzátoru. Existence ϵ -pravidel a jednoduchých pravidel, na druhé straně, neimplikuje existenci cyklu (tj. derivace tvaru $A \Rightarrow^+ A$ pro nějaké $A \in N$).



Další transformací, kterou uvedeme, je odstranění levé rekurze. Tato transformace je důležitá pro použití analýzy typu shora-dolů.

Definice 4.12 Nechť $G = (N, \Sigma, P, S)$. Přepisovací pravidlo z P se nazývá *rekurzivní zleva* (*rekurzivní zprava*), jestliže je tvaru $A \rightarrow A\alpha(A \rightarrow \alpha A)$, $A \in N, \alpha \in (N \cup \Sigma)^*$. Jestliže v G existuje derivace $A \Rightarrow^+ \alpha A\beta$, pro nějaké $A \in N$, říkáme, že gramatika G je *rekurzivní*. Je-li $\alpha = \epsilon$, pak mluvíme o *gramatice rekurzivní zleva*, je-li $\beta = \epsilon$, pak říkáme, že G je *rekurzivní zprava*.

DEF

Poznamenejme, že je-li jazyk $L(G)$ nekonečný, pak G musí být rekurzivní.

Dříve, než zformujeme algoritmus eliminující levou rekurzi, ukážeme, jakým způsobem lze odstranit přepisovací pravidla rekurzivní zleva.

Definice 4.13 Pravidlo $A \rightarrow \alpha$, s levou stranou tvořenou nonterminálem A , budeme nazývat A -pravidlo (Neplést s ϵ -pravidlem.)

DEF

Věta 4.6 Nechť $G = (N, \Sigma, P, S)$ je gramatika a nechť $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$ jsou všechna A -pravidla. Žádný z řetězců β_i nezačíná nonterminálem A . Gramatika $G' = (N \cup \{A'\}, \Sigma, P', S)$, kde P' obsahující namísto uvedených pravidel pravidla

$$\begin{aligned} A &\rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n \mid \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A' \\ A' &\rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m \mid \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \end{aligned}$$

je ekvivalentní s gramatikou G , tj. $L(G) = L(G')$.

Důkaz: Uvedena transformace nahrazuje pravidla rekurzivní zleva pravidly, která jsou rekurzivní zprava. Označíme-li jazyky $L_1 = \{\beta_1, \beta_2, \dots, \beta_n\}$ a $L_2 = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$ vidíme, že v G lze z nonterminálu A derivovat řetězce tvořící jazyk $L_1 L_2^*$. Právě tyto řetězce můžeme však derivovat z A také v gramatice G' . Efekt popisované transformace ilustruje obrázek 4.12. \square

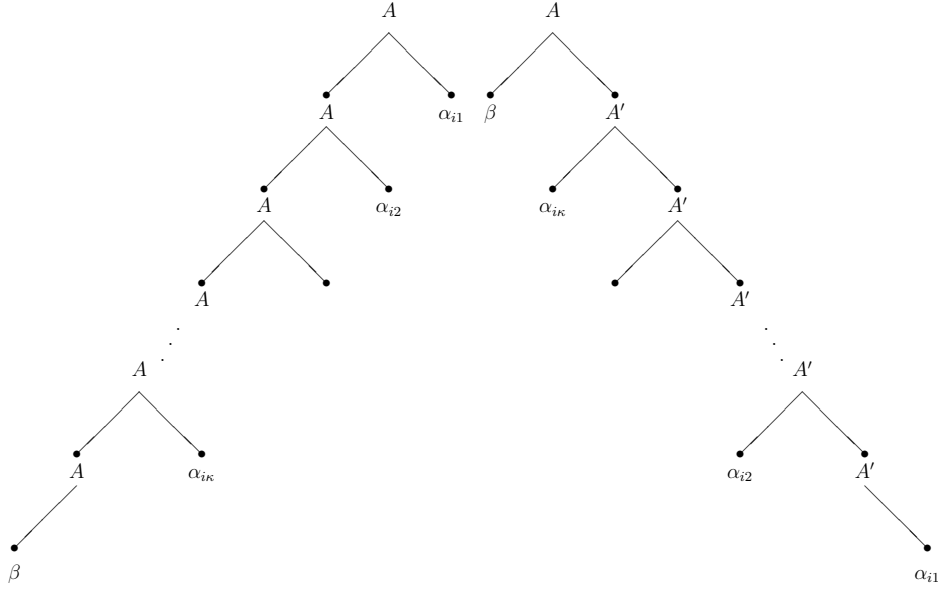
Příklad 4.15 Uvažujme gramatiku G z předcházejícího příkladu s pravidly

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid i \end{aligned}$$

x+y

Po odstranění pravidel rekurzivních zleva získáme ekvivalentní gramatiku G' s pravidly

$$\begin{aligned} E &\rightarrow T \mid TE' \\ E' &\rightarrow +T \mid +TE' \\ T &\rightarrow F \mid FT' \\ T' &\rightarrow *F \mid *FT' \\ F &\rightarrow (E) \mid i \end{aligned}$$



Obrázek 4.12: Odstranění levé rekurze: nalevo část stromu v G , napravo část stromu v G'

Věta 4.7 Necht $G = (N, \Sigma, P, S)$ je gramatika. $A \rightarrow \alpha B \beta$, $B \in N$; $\alpha, \beta \in (N \cup \Sigma)^*$ je pravidlo z P a $B \rightarrow \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_n$ jsou všechna B -pravidla v P . Necht $G' = (N, \Sigma, P', S)$, kde

$$P' = P - \{A \rightarrow \alpha B \beta\} \cup \{A \rightarrow \alpha \gamma_1 \beta \mid \alpha \gamma_2 \beta \mid \dots \mid \alpha \gamma_n \beta\}.$$

Pak $L(G) = L(G')$.

Důkaz: Výsledkem této transformace je odstranění pravidla $A \rightarrow \alpha B \beta$ z gramatiky G tím způsobem, že za nonterminál B „dosadíme“ všechny pravé strany B -pravidel. Formálně se důkaz provede tak, že se ukáže platnost konjunkce $L(G) \subseteq L(G) \wedge L(G') \subseteq L(G)$. Efekt této transformace ilustruje obr. 4.13, jenž znázorňuje derivační stromy řetězce $aabbb$ v gramatice G resp. G' . Gramatika G obsahuje pravidlo $A \rightarrow aAA$ a dvě A -pravidla $A \rightarrow aAA \mid b$. Položíme-li $\alpha = a, B = A, \beta = A$, můžeme eliminovat pravidlo $A \rightarrow aAA$ zavedením těchto A -pravidel v gramatice G' :

$$A \rightarrow aaAAA \mid abA \mid b$$

□

Algoritmus 4.6 Odstranění levé rekurze.

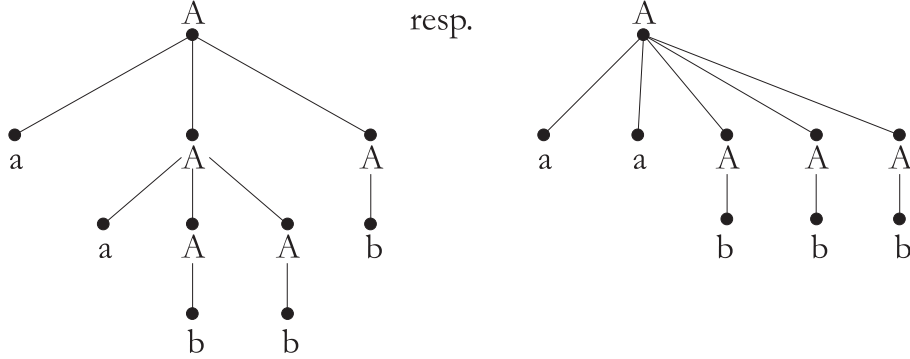
Vstup: Vlastní gramatika $G = (N, \Sigma, P, S)$

Výstup: Gramatika G' bez levé rekurze.

Metoda:

- (1) Necht $N = \{A_1, A_2, \dots, A_n\}$. Gramatiku budeme transformovat tak, že je-li $A_i \rightarrow \alpha$ pravidlo, pak α začíná buď terminálem, nebo nonterminálem A_j , $j > i$. K tomu účelu položíme $i = 1$.



Obrázek 4.13: Derivační stromy věty $aabb$ v G resp. G'

- (2) Necht $A_i \rightarrow A_i\alpha_1 \mid \dots \mid A_i\alpha_m \mid \beta_1 \mid \dots \mid \beta_p$ jsou všechna A_i -pravidla a necht žádné β_i nezačíná nonterminálem A_k , je-li $k \leq i$.

Nahraď všechna A_i -pravidla těmito pravidly:

$$\begin{aligned} A_i &\rightarrow \beta_1 \mid \dots \mid \beta_p \mid \beta_1 A'_i \mid \dots \mid \beta_p A'_i \\ A'_i &\rightarrow \alpha_1 \mid \dots \mid \alpha_m \mid \alpha_1 A'_i \mid \dots \mid \alpha_m A'_i \end{aligned}$$

kde A'_i je nový nonterminál. Takto všechna A_i -pravidla začínají buď terminálem, nebo nonterminálem $A_k, k > i$.

- (3) Je-li $i = n$, pak jsme získali výslednou gramatiku G' . V opačném případě polož $i = i + 1$ a $j = 1$.
- (4) Každé pravidlo tvaru $A_i \rightarrow A_j\alpha$ nahraď pravidly $A_i \rightarrow \beta_1\alpha \mid \dots \mid \beta_p\alpha$ kde $A_j \rightarrow \beta_1 \mid \dots \mid \beta_p$ jsou všechna A_j -pravidla. Po této transformaci budou všechna A_j -pravidla začínat buď terminálem, nebo nonterminálem $A_k, k > j$, takže také všechna A_i -pravidla budou mít tuto vlastnost.
- (5) Je-li $j = i - 1$, pak přejdi ke kroku (2). Jinak $j = j + 1$ a opakuj krok (4).

Věta 4.8 Každý bezkontextový jazyk lze generovat gramatikou bez levé rekurze. Důkaz: Necht G je vlastní gramatika. Algoritmus 4.6 používá pouze transformací z vět 4.6 a 4.7 a tudíž je $L(G) = L(G')$.

Dále musíme dokázat, že G' není gramatikou rekurzivní zleva. Formální důkaz nebudeme provádět (viz [1]). Uvědomíme si však, že krok (2) odstraňuje pravidla rekurzivní zleva. Stejně tak pravidla vzniklá aplikací kroku (4) nejsou rekurzivní zleva. Protože krok (2) je aplikován na všechny nonterminální symboly, nemůže výsledná gramatika obsahovat pravidla rekurzivní zleva. \square

Příklad 4.16 Necht G je gramatika s pravidly:

$$\begin{aligned} A &\rightarrow BC \mid a \\ B &\rightarrow CA \mid Ab \\ C &\rightarrow AB \mid CC \mid a \end{aligned}$$

$$x + y$$

Položme $A_1 = A, A_2 = B$ a $A_3 = C$. V každém kroku uvedeme pouze nová pravidla pro ty nonterminály, jejichž pravidla se mění.

krok (2), $i=1$, beze změny
 krok (4), $i=2, j=1$, $B \rightarrow CA \mid BCb \mid ab$
 krok (2), $i=2$, $B \rightarrow CA \mid ab \mid CAB' \mid abB'$
 $B' \rightarrow CbB' \mid Cb$
 krok (4), $i=3, j=1$, $C \rightarrow BCB \mid aB \mid CC \mid a$
 krok (4), $i=3, j=2$, $C \rightarrow CACB \mid abCB \mid CAB'CB \mid abB'CB \mid aB \mid CC \mid a$
 krok (2), $i=3$, $C \rightarrow abCB \mid abB'CB \mid aB \mid a \mid abCBC' \mid abB'CBC'$
 $C \rightarrow aBC' \mid aC'$
 $C' \rightarrow ACBC' \mid AB'CBC' \mid CC' \mid ACB \mid AB'CB \mid C$

4.6 Chomského normální forma

Definice 4.14 Gramatika $G = (N, \Sigma, P, S)$ je v *Chomského normální formě* (CNF), jestliže každé pravidlo z P má jeden z těchto tvarů:

- (1) $A \rightarrow BC, A, B, C \in N$ nebo
- (2) $A \rightarrow a, a \in \Sigma$ nebo
- (3) Je-li $\epsilon \in L(G)$, pak $S \rightarrow \epsilon$ je pravidlo z P a výchozí symbol S se neobjeví na pravé straně žádného pravidla.

Chomského normální forma gramatiky je prostředkem zjednodušujícím tvar reprezentace bezkontextového jazyka. Nyní popíšeme algoritmus převodu obecné bezkontextové gramatiky do Chomského normální formy.

Algoritmus 4.7 Převod do Chomského normální formy.

Vstup: Vlastní gramatika $G = (N, \Sigma, P, S)$ bez jednoduchých pravidel.

Výstup: Gramatika $G' = (N', \Sigma, P', S')$ v CNF taková, že $L(G) = L(G')$

Metoda: Z gramatiky G získáme ekvivalentní gramatiku G' v CNF takto:

- (1) Množina pravidel P' obsahuje všechna pravidla tvaru $A \rightarrow a$ z P
- (2) Množina pravidel P' obsahuje všechna pravidla tvaru $A \rightarrow BC$ z P
- (3) Je-li pravidlo $S \rightarrow \epsilon$ v P , pak $S \rightarrow \epsilon$ je také v P'
- (4) Pro každé pravidlo tvaru $A \rightarrow X_1 \dots X_k$, kde $k > 2$ z P přidej k P' tuto množinu pravidel. Symbolem X'_i značíme nonterminál X_i , je-li $X_i \in N$, nebo nový nonterminál, je-li $X_i \in \Sigma$:

$$\begin{aligned}
 A &\rightarrow X'_1 \langle X_2 \dots X_k \rangle \\
 \langle X_2 \dots X_k \rangle &\rightarrow X'_2 \langle X_3 \dots X_k \rangle \\
 &\vdots \\
 \langle X_{k-1} X_k \rangle &\rightarrow X'_{k-1} X'_k
 \end{aligned}$$

kde každý symbol $\langle X_i \dots X_k \rangle$ značí nový nonterminální symbol.

- (5) Pro každé pravidlo tvaru $A \rightarrow X_1 X_2$, kde některý ze symbolů X_1 nebo X_2 leží v Σ přidej k P' pravidlo $A \rightarrow X'_1 X'_2$.
- (6) Pro každý nový nonterminál tvaru a' přidej k P' pravidlo $a' \rightarrow a$. Výsledná gramatika je $G' = (N', \Sigma, P', S')$; množina N' obsahuje všechny nonterminály tvaru $\langle X_i \dots X_k \rangle$ a a' .

DEF

!

Věta 4.9 Necht L je bezkontextový jazyk. Pak existuje bezkontextová gramatika G v CNF taková, že $L = L(G)$.

Důkaz: Podle věty 4.5 má jazyk L vlastní gramatiku G . Stačí tedy dokázat, že výsledkem algoritmu 4.7 je ekvivalentní gramatika G' tj. $L(G) = L(G')$. Tato ekvivalence však plyne bezprostředně z aplikace věty 4.7 na každé pravidlo z P' , které má nonterminály tvaru a' a $\langle X_i \dots X_j \rangle$. Výsledkem bude gramatika G' . \square

Příklad 4.17 Necht G je gramatika s pravidly

$$\begin{aligned} S &\rightarrow aAB \mid BA \\ A &\rightarrow BBB \mid a \\ B &\rightarrow AS \mid b \end{aligned}$$

x + y

Chomského normální forma $G' = (N', \{a, b\}, P', S)$ má pravidla:

$$\begin{aligned} S &\rightarrow a' \langle AB \rangle \mid BA \\ A &\rightarrow B \langle BB \rangle \mid a \\ B &\rightarrow AS \mid b \\ \langle AB \rangle &\rightarrow AB \\ \langle BB \rangle &\rightarrow BB \\ a' &\rightarrow a \end{aligned}$$

Nová množina nonterminálů je tedy $N' = \{S, A, B, \langle AB \rangle, \langle BB \rangle, a'\}$

Příklad 4.18 Předpokládejme, že gramatika G je v CNF, a že řetězci $w \in L(G)$ přísluší derivace v G délky p . Jaká je délka věty w ?

Řešení: Necht $|w| = n$. Pak pro odvození věty bylo třeba n -krát aplikovat pravidlo tvaru $A \rightarrow a$ a $(n-1)$ -krát pravidlo tvaru $A \rightarrow BC$. Dostáváme tedy vztah

$$p = n + n - 1 = 2n - 1$$

a vidíme, že p je vždy liché. Řešení příkladu je tedy

$$|w| = \frac{p+1}{2}$$

x + y

4.7 Greibachové normální forma

Definice 4.15 Gramatika $G = (N, \Sigma, P, S)$ je v Greibachové normální formě (GNF), je-li G gramatikou bez ϵ -pravidel a jestliže každé pravidlo (s výjimkou případného pravidla $S \rightarrow \epsilon$) má tvar $A \rightarrow a\alpha$ kde $a \in \Sigma$ a $\alpha \in N^*$.

DEF

Lemma 4.1 Necht $G = (N, \Sigma, P, S)$ je gramatika bez levé rekurze. Pak existuje lineární uspořádání $<$ definované na množině nonterminálních symbolů N takové, že je-li $A \rightarrow B\alpha$ v P , pak $A < B$.

Důkaz: Necht R je relace na množině N taková, že ARB platí právě když $A \Rightarrow^+ B\alpha$ pro nějaké $\alpha \in (N \cup \Sigma)^*$. Z definice levé rekurze plyne, že R je částečné uspořádání (R je tranzitivní). Každé částečné uspořádání pak lze rozšířit na lineární uspořádání. \square

Nyní zformulujeme algoritmus převodu gramatiky G do Greibachové normální formy.

Algoritmus 4.8 Převod do Greibachové normální formy.

Vstup: Vlastní gramatika bez levé rekurze $G = (N, \Sigma, P, S)$

Výstup: Ekvivalentní gramatika G' v GNF

Metoda:

- (1) Podle lemmy 4.1 vytvoř lineární uspořádání $<$ na N takové, že každé A -pravidlo začíná buď terminálem, nebo nějakým nonterminálem B takovým, že $A < B$. Nechť

$$N = \{A_1, A_2, \dots, A_n\} \text{ a } A_1 < A_2 < \dots < A_n.$$

- (2) Polož $i = n - 1$
- (3) Je-li $i = 0$ přejdi k bodu (5), je-li $i \neq 0$ nahraď každé pravidlo tvaru $A_i \rightarrow A_j \alpha$, kde $j > i$ pravidly $A_i \rightarrow \beta_1 \alpha \mid \dots \mid \beta_m \alpha$, kde $A_j \rightarrow \beta_1 \mid \dots \mid \beta_m$ jsou všechna A_j -pravidla. (Každý z řetězců $\beta_1 \dots \beta_m$ začíná terminálem.)
- (4) Polož $i = i - 1$ a opakuj krok (3).
- (5) V tomto okamžiku všechna pravidla (s výjimkou pravidla $S \rightarrow \epsilon$) začínají terminálním symbolem. V každém pravidle $A \rightarrow aX_1 \dots X_k$ nahraď ty symboly X_j , které jsou terminálními symboly, novým nonterminálem X'_j .
- (6) Pro všechna X'_j z bodu (5) přidej pravidla $X'_j \rightarrow X_j$

Věta 4.10 Nechť L je bezkontextový jazyk. Pak existuje gramatika G v GNF taková, že $L = L(G)$.

Důkaz: Z definice uspořádání $<$ vyplývá, že všechna A_n -pravidla začínají terminály, a že po opakovaném provádění kroku (3) algoritmu 4.8 budou všechna A_i -pravidla pro $i = 1, \dots, n - 1$ začínat také terminálními symboly. Krok (5) a (6) převádí nepočáteční terminální symboly pravých stran na nonterminály, aniž se mění generovaný jazyk. \square

Příklad 4.19 Převeďme gramatiku G s pravidly

$$\begin{aligned} E &\rightarrow T \mid TE' \\ E' &\rightarrow +T \mid +TE' \\ T &\rightarrow F \mid FT' \\ T' &\rightarrow *F \mid *FT' \\ F &\rightarrow (E) \mid i \quad \text{do GNF.} \end{aligned}$$

Řešení: Podle lemmy 4.1 je $E < T < F$. Jako lineární uspořádání na množině nonterminálů vezmeme uspořádání

$$E' < E < T' < T < F$$

Všetchna F -pravidla začínají terminálem (důsledek skutečnosti, že F je největší prvek v uspořádání $<$). Předcházející symbol T má pravidla $T \rightarrow F \mid FT'$ a po aplikaci kroku (3) dostáváme pravidla $T \rightarrow (E) \mid i \mid (E)T' \mid iT'$.



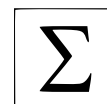
$x + y$

Výsledná gramatika v GNF má pravidla:

$$\begin{aligned}
E &\rightarrow (E)' \mid i \mid (E)'T' \mid iT' \mid (E)'E' \mid iE' \mid (E)'T'E' \mid iT'E' \\
E' &\rightarrow +T \mid +TE' \\
T &\rightarrow (E)' \mid i \mid (E)'T' \mid iT' \\
T' &\rightarrow *F \mid *FT' \\
F &\rightarrow (E)' \mid i \\
)' &\rightarrow)
\end{aligned}$$

Nevýhodou algoritmu 4.8 je velké množství nových pravidel. Existuje alternativní algoritmus převodu do GNF, [1], který nezvětšuje tak výrazně počet pravidel gramatiky, avšak zavádí zase více nonterminálů.

Bezkontextové gramatiky umožňují popisovat většinu rysů současných vyšších programovacích jazyků. Způsob získání derivačního stromu věty v dané bezkontextové gramatice je základem syntaktické analýzy a klasifikace syntaktických analyzátorů překladačů a jazykových procesorů. Řada transformací bezkontextových gramatik a existence normálních forem bezkontextových gramatik umožňuje odstraňování některých atributů gramatiky nebo derivaci v této gramatice, jež se využívá v praktických aplikacích i v rámci důkazových technik.



4.8 Cvičení



Cvičení 4.8.1 Modifikujte gramatiku G_{RV} z příkladu 4.10 tak, aby zápisy regulárních výrazů rovněž připouštěly:

- (a) explicitní použití operátoru \cdot konkatenace (např. $a \cdot b$)
- (b) použití operátoru $^+$ ve významu $r^+ = rr^*$

Cvičení 4.8.2 Gramatika $G = (\{\langle \text{deklarace} \rangle, \langle \text{options} \rangle, \langle \text{option} \rangle, \langle \text{mode} \rangle, \langle \text{scale} \rangle, \langle \text{precision} \rangle, \langle \text{base} \rangle\}, \{\text{declare, id, real, complex, fixed, floating, single, double, binary, decimal}\}, P, \langle \text{deklarace} \rangle)$ s pravidly P

$$\begin{aligned}
\langle \text{deklarace} \rangle &\rightarrow \text{declare id } \langle \text{options} \rangle \\
\langle \text{options} \rangle &\rightarrow \langle \text{option} \rangle \langle \text{options} \rangle \mid \epsilon \\
\langle \text{option} \rangle &\rightarrow \langle \text{mode} \rangle \mid \langle \text{scale} \rangle \mid \langle \text{precision} \rangle \mid \langle \text{base} \rangle \\
\langle \text{mode} \rangle &\rightarrow \text{real} \mid \text{complex} \\
\langle \text{scale} \rangle &\rightarrow \text{fixed} \mid \text{floating} \\
\langle \text{precision} \rangle &\rightarrow \text{single} \mid \text{double} \\
\langle \text{base} \rangle &\rightarrow \text{binary} \mid \text{decimal}
\end{aligned}$$

popisuje deklaraci jednoduché proměnné. Tato gramatika však dovoluje vytvářet deklarace, které obsahují redundantní nebo rozporné informace, např.

declare X real fixed real floating

Zapište gramatiku, která připouští pouze deklarace bez redundancí a rozporů. Uvažte, zda je vhodné takový problém řešit prostředky definice syntaxe jazyka.

Cvičení 4.8.3 Pro výpočet tranzitivního uzávěru binární relace definované na konečné množině (např. slovníku gramatiky) se velmi často používá reprezentace relace prostřednictvím booleovské matice a operací nad touto maticí. Je-li A booleovská matice reprezentující binární relaci R na množině M (tj. $A[p, q] = \text{true}$, je-li $(p, q) \in R$ a $A[p, q] = \text{false}$, je-li $(p, q) \notin R$, $p, q \in M$ a $A[p, q]$ je prvek matice A v řádku označeném p a sloupci označeném q), pak tranzitivní uzávěr relace R je relace R^+ , jenž je reprezentován maticí A^+ :

$$A^+ = A + A^2 + \dots + A^n, \quad (3)$$

kde $n = \min(|M| - 1, |R|)$ a operace sečítání, resp. násobení jsou interpretovány jako disjunkce, resp. konjunkce.

Pro výpočet tranzitivního uzávěru existuje efektivnější postup než podle 3, nazývaný, podle autora, *Warshallův algoritmus*.

Algoritmus 4.9 Warshallův algoritmus pro výpočet tranzitivního uzávěru binární relace.

Vstup: Booleovská matice A reprezentující binární relaci R .

Výstup: Booleovská matice B reprezentující binární relaci R^+ .

Metoda:

- (1) Polož $B = A$ a $i = 1$.
- (2) Pro všechna j , jestliže $B[j, i] = \text{true}$, pak pro $k = 1, \dots, n$ polož $B[j, k] = B[j, k] + B[i, k]$.
- (3) Polož $i = i + 1$.
- (4) Je-li $i \leq n$, vrať se ke kroku (2); v opačném případě je B výsledná matice.

Ukažte, že algoritmus 4.9 počítá skutečně tranzitivní uzávěr binární relace.

Cvičení 4.8.4 V gramatice z příkladu 4.3 vytvořte levou a pravou derivaci a derivační strom věty $i * (i + i - i)$. Nalezněte všechny fráze, jednoduché fráze a l -frázi této věty.

Cvičení 4.8.5 Necht $G = (N, \Sigma, P, S)$ je gramatika a α_1, α_2 řetězce $\alpha_1, \alpha_2 \in (N \cup \Sigma)^*$. Navrhněte algoritmus, který zjišťuje, zda platí $\alpha_1 \xRightarrow{G}^* \alpha_2$.

Cvičení 4.8.6 Je dána gramatika $G = (\{S, A, B\}, \{a, b\}, P, S)$, která má pravidla

$$\begin{aligned} S &\rightarrow bA \mid aB \\ A &\rightarrow a \mid aS \mid bAA \\ B &\rightarrow b \mid bS \mid aBB \end{aligned}$$

Ukažte, že tato gramatika je víceznačná (uvažujte např. větu $aabbab$). Pokuste se nalézt ekvivalentní jednoznačnou gramatiku.



Cvičení 4.8.7 Gramatiku s pravidly

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aB \mid bS \mid b \\ B &\rightarrow AB \mid Ba \\ C &\rightarrow AS \mid b \end{aligned}$$

transformujte na ekvivalentní gramatiku bez zbytečných symbolů.

Cvičení 4.8.8 Nalezněte gramatiku bez ϵ -pravidel, jež je ekvivalencí s gramatikou

$$\begin{aligned} S &\rightarrow ABC \\ A &\rightarrow BB \mid \epsilon \\ B &\rightarrow CC \mid a \\ C &\rightarrow AA \mid b \end{aligned}$$

Cvičení 4.8.9 Ke gramatice

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow C \mid D \\ B &\rightarrow D \mid E \\ C &\rightarrow S \mid a \mid \epsilon \\ D &\rightarrow S \mid b \\ E &\rightarrow S \mid c \mid \epsilon \end{aligned}$$

nalezněte ekvivalentní vlastní gramatiku.

Cvičení 4.8.10 Formulujte algoritmus, který testuje, zda gramatika G je a nebo není gramatikou bez cyklu.

Cvičení 4.8.11 V gramatice

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow BS \mid b \\ B &\rightarrow SA \mid a \end{aligned}$$

odstraňte levou rekurzi.

Cvičení 4.8.12 Do Chomského normální formy převedte gramatiku

$$G = (\{S, T, L\}, \{a, b, +, -, *, /, [,]\}, P, S)$$

kde P obsahuje pravidla

$$\begin{aligned} S &\rightarrow T + S \mid T - S \mid T \\ T &\rightarrow L * T \mid L / T \mid L \\ L &\rightarrow [S] \mid a \mid b \end{aligned}$$

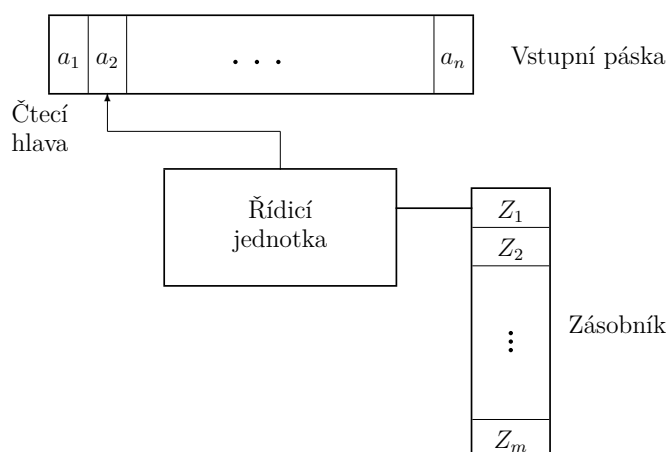
Cvičení 4.8.13 Gramatiku $G = (\{S, A, B\}, \{a, b\}, P, S)$ s pravidly

$$\begin{aligned} S &\rightarrow Ba \mid Ab \\ A &\rightarrow Sa \mid AAb \mid a \\ B &\rightarrow Sb \mid BBa \mid b \end{aligned}$$

převeďte do Greibachové normální formy.

4.9 Základní definice zásobníkového automatu

Zásobníkový automat je jednocestný nedeterministický automat, jenž je opatřen zásobníkem reprezentujícím nekonečnou paměť. Schéma zásobníkového automatu je uvedeno na obr. 4.14.



Obrázek 4.14: Schéma zásobníkového automatu

Vstupní páska je rozdělena na jednotkové záznamy, každý záznam obsahuje právě jeden vstupní symbol. Obsah jednotkového záznamu může být čten čtecí hlavou, nemůže však být přepsán (read only input tape). V určitém časovém okamžiku může čtecí hlava zůstat na daném záznamu, nebo se posune o jeden záznam doprava (jednocestný automat). Konečná řídicí jednotka realizuje operace posuvu čtecí hlavy a ukládání informace do zásobníku prostřednictvím funkce přechodů definované nad vstupní abecedou, množinou stavů řídicí jednotky a vrcholem zásobníku.

Definice 4.16 Zásobníkový automat P je sedmice

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F), \text{ kde}$$

- (1) Q je konečná množina stavových symbolů reprezentujících vnitřní stavy řídicí jednotky,
- (2) Σ je konečná vstupní abeceda; jejími prvky jsou vstupní symboly,
- (3) Γ je konečná abeceda zásobníkových symbolů,
- (4) δ je zobrazení z množiny $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ do konečné množiny podmnožin množiny $Q \times \Gamma^*$ popisující funkci přechodů,



- (5) $q_0 \in Q$ je počáteční stav řídicí jednotky,
- (6) $Z_0 \in \Gamma$ je symbol, který je na počátku uložen do zásobníku – tzn. *startovací symbol* zásobníku,
- (7) $F \subseteq Q$ je množina koncových stavů.

Definice 4.17 *Konfigurací* zásobníkového automatu P nazveme trojici

$$(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$$

kde

- (1) q je přítomný stav řídicí jednotky
- (2) w je doposud nepřčtená část vstupního řetězce; první symbol řetězce w je pod čtecí hlavou. Je-li $w = \epsilon$, pak byly všechny symboly ze vstupní pásky přčteny.
- (3) α je obsah zásobníku. Pokud nebude uvedeno jinak, budeme zásobník reprezentovat řetězcem, jehož nejlevější symbol koresponduje s vrcholem zásobníku. Je-li $\alpha = \epsilon$, pak je zásobník prázdný.

Definice 4.18 *Přechod* zásobníkového automatu P budeme reprezentovat binární relací \vdash_P (nebo \vdash bude-li zřejmé, že jde o automat P), která je definována na množině konfigurací zásobníkového automatu P . Relace

$$(q, w, \beta) \vdash (q', w', \beta')$$

platí pro $q, q' \in Q$, $w, w' \in \Sigma^*$, $\beta, \beta' \in \Gamma^*$, jestliže $w = aw'$ pro nějaké $a \in (\Sigma \cup \{\epsilon\})$, $\beta = Z\alpha$ a $\beta' = \gamma\alpha$ pro nějaké $Z \in \Gamma$, $\alpha, \gamma \in \Gamma^*$ a $\delta(q, a, Z)$ obsahuje prvek (q', γ) .

Relaci \vdash_P interpretujeme tímto způsobem. Nachází-li se zásobníkový automat P ve stavu q a na vrcholu zásobníku je uložen symbol Z , pak po přčtení vstupního symbolu $a \neq \epsilon$ může automat přejít do stavu q' , přičemž se čtecí hlava posune doprava a na vrchol zásobníku se uloží, po odstranění symbolu Z , řetězec γ . Je-li $\gamma = \epsilon$, pak je pouze odstraněn vrchol zásobníku.

Je-li $a = \epsilon$, neposouvá se čtecí hlava, což znamená, že přechod do nového stavu a nový obsah zásobníku není určován příštím vstupním symbolem. Tento typ přechodu budeme nazývat *ϵ -přechodem*. Poznamenejme, že ϵ -přechod může nastat i tehdy, když byly přčteny všechny vstupní symboly.

Relace $\vdash^i, \vdash^+, \vdash^*$ jsou definovány obvyklým způsobem. Relace \vdash^+ resp. \vdash^* je tranzitivní, resp. tranzitivní a reflexivní uzávěr relace \vdash , \vdash^i je i -tá mocnina relace \vdash , $i \geq 0$.

Počáteční konfigurace zásobníkového automatu má tvar (q_0, w, Z_0) pro $w \in \Sigma^*$, tj. automat je v počátečním stavu q_0 , na vstupní pásce je řetězec w a v zásobníku je startovací symbol Z_0 . *Koncová konfigurace* má tvar (q, ϵ, α) , kde $q \in F$ je koncový stav a $\alpha \in \Gamma^*$.

Definice 4.19 Platí-li pro řetězec $w \in \Sigma^*$ relace $(q_0, w, Z_0) \vdash^*(q, \epsilon, \alpha)$ pro nějaké $q \in F$ a $\alpha \in \Gamma^*$, pak říkáme, že řetězec w je přijímán zásobníkovým automatem P . Množinu $L(P)$ všech řetězců přijímaných zásobníkovým automatem P , který nazýváme *jazykem přijímaným zásobníkovým automatem*.

Příklad 4.20 Uvažujme jazyk $L = \{0^n 1^n \mid n \geq 0\}$. Zásobníkový automat P , který přijímá jazyk L , má tvar

$$P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{Z, 0\}, \delta, q_0, Z, \{q_0\})$$

kde

$$\begin{aligned}\delta(q_0, 0, Z) &= \{(q_1, 0Z)\} \\ \delta(q_1, 0, 0) &= \{(q_1, 00)\} \\ \delta(q_1, 1, 0) &= \{(q_2, \epsilon)\} \\ \delta(q_2, 1, 0) &= \{(q_2, \epsilon)\} \\ \delta(q_2, \epsilon, Z) &= \{(q_0, \epsilon)\}\end{aligned}$$

a pracuje tak, že kopíruje všechny nuly ze vstupního řetězce do zásobníku. Objeví-li se na vstupu symbol 1, pak odstraňuje jednu nulu ze zásobníku. Kromě toho požaduje, aby všechny nuly předcházely všechny jedničky. Například pro vstupní řetězec 0011 realizuje automat P tyto přechody:

$$\begin{aligned}(q_0, 0011, Z) &\vdash (q_1, 011, 0Z) \\ &\vdash (q_1, 11, 00Z) \\ &\vdash (q_2, 1, 0Z) \\ &\vdash (q_2, \epsilon, Z) \\ &\vdash (q_0, \epsilon, \epsilon)\end{aligned}$$

Ukažme, že skutečně platí $L(P) = L$. Z definice funkce přechodů δ a relace \vdash plyne $(q_0, \epsilon, Z) \vdash^0 (q_0, \epsilon, Z)$ a tedy $\epsilon \in L(P)$. Dále platí

$$\begin{aligned}(q_0, 0, Z) &\vdash (q_1, \epsilon, 0Z) \\ (q_1, 0^i, 0Z) &\vdash^i (q_1, \epsilon, 0^{i+1}Z) \\ (q_1, 1, 0^{i+1}Z) &\vdash (q_2, \epsilon, 0^iZ) \\ (q_2, 1^i, 0^iZ) &\vdash^i (q_2, \epsilon, Z) \\ (q_2, \epsilon, Z) &\vdash (q_0, \epsilon, \epsilon)\end{aligned}$$

což dohromady implikuje

$$(q_0, 0^n 1^n, Z) \vdash^{2n+1} (q_0, \epsilon, \epsilon) \quad \text{pro } n \geq 1$$

a tedy $L \subseteq L(P)$.

Dokázat opačnou inkluzi $L(P) \subseteq L$, tj. dokázat, že zásobníkový automat nepřijímá jiné řetězce, než $0^n 1^n$, $n \geq 1$, je obtížnější. Přijme-li zásobníkový automat P vstupní řetězec, pak musí projít posloupností stavů q_0, q_1, q_2, q_0 .

Jestliže platí $(q_0, w, Z) \vdash^i (q_1, \epsilon, \alpha)$, pak $w = 0^i$ a $\alpha = 0^i Z$. Podobně, jestliže $(q_2, w, \alpha) \vdash^i (q_2, \epsilon, \beta)$ pak $w = 1^i$ a $\alpha = 0^i \beta$. To však znamená, že relace $(q_1, w, \alpha) \vdash (q_2, \epsilon, \beta)$ platí, právě když $w = 1$ a $\alpha = 0\beta$ a relace $(q_2, w, Z) \vdash^*(q_0, \epsilon, \epsilon)$ platí, právě když $w = \epsilon$. Tedy, jestliže platí $(q_0, w, Z) \vdash^i (q_0, \epsilon, \alpha)$ pro nějaké $i > 0$, pak $w = 0^n 1^n$, $i = 2n + 1$ a $\alpha = \epsilon$, tj. $L(P) \subseteq L$.

4.10 Varianty zásobníkových automatů

V tomto odstavci uvedeme dvě varianty základní definice zásobníkového automatu, které nám usnadní objasnit vztah mezi zásobníkovými automaty a bezkontextovými jazyky.

x + y

Definice 4.20 *Rozšířeným zásobníkovým automatem* rozumíme sedmici

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

kde δ je zobrazení z konečné podmnožiny $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma^*$ do množiny podmnožin množiny $Q \times \Gamma^*$. Ostatní symboly mají stejný význam jako v základní definici 4.16.

Podobně jako u běžných ZA je relace přechodu \vdash definována jako *nejmenší* relace taková, že $(q, aw, \alpha\gamma) \vdash (q', w, \beta\gamma)$ platí, jestliže $\delta(q, a, \alpha)$ obsahuje (q', β) pro $q, q' \in Q, a \in \Sigma \cup \{\epsilon\}$ a $\alpha, \beta, \gamma \in \Gamma^*$. Tato relace odpovídá přechodu, v němž je vrcholový řetězec zásobníku odstraněn a nahrazen řetězcem β . (Připomeňme, že v základní definici je $\alpha \in \Gamma$ nikoliv $\alpha \in \Gamma^*$). Jazyk definovaný automatem P je

$$L(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \epsilon, \alpha), q \in F, \alpha \in \Gamma^*\}$$

Rozšířený zásobníkový automat může, na rozdíl od základní definice provádět přechody i v případě, že je zásobník prázdný.

Příklad 4.21 Uvažujme rozšířený zásobníkový automat P , který přijímá jazyk

$$\begin{aligned} L &= \{ww^R \mid w \in \{a, b\}^*\}, \\ P &= (\{q, p\}, \{a, b\}, \{a, b, S, Z\}, \delta, q, Z, \{p\}) \end{aligned}$$

kde zobrazení δ je definováno takto:

$$\begin{aligned} \delta(q, a, \epsilon) &= \{(q, a)\} \\ \delta(q, b, \epsilon) &= \{(q, b)\} \\ \delta(q, \epsilon, \epsilon) &= \{(q, S)\} \\ \delta(q, \epsilon, aSa) &= \{(q, S)\} \\ \delta(q, \epsilon, bSb) &= \{(q, S)\} \\ \delta(q, \epsilon, SZ) &= \{(p, \epsilon)\} \end{aligned}$$

Pro vstupní řetězec $aabbaa$ realizuje automat P tyto přechody:

$$\begin{aligned} (q, aabbaa, Z) &\vdash (q, abbaa, aZ) \\ &\vdash (q, bbaa, aaZ) \\ &\vdash (q, baa, baaZ) \\ &\vdash (q, baa, SbbaaZ) \\ &\vdash (q, aa, bSbbaaZ) \\ &\vdash (q, aa, SaaZ) \\ &\vdash (q, a, aSaaZ) \\ &\vdash (q, a, SaZ) \\ &\vdash (q, \epsilon, aSaZ) \\ &\vdash (q, \epsilon, SZ) \\ &\vdash (q, \epsilon, \epsilon) \end{aligned}$$

Vidíme, že automat P pracuje tak, že nejdříve ukládá do zásobníku prefix vstupního řetězce, pak na vrchol zásobníku uloží znak S značící střed; pak vrcholový řetězec aSa nebo bSb nahrazuje symbolem S . Tento příklad ilustruje také neterministickou povahu zásobníkového automatu, protože zobrazením není určeno,

DEF

x + y

zda má být přečten vstupní symbol a uložen na vrchol zásobníku ($\delta(q, x, \epsilon) = (q, x)$, $x \in \{a, b\}$), nebo zda má být proveden ϵ -přechod, který uloží na vrchol zásobníku středový symbol S , ($\delta(q, \epsilon, \epsilon) = (q, S)$).

Z definice rozšířeného zásobníkového automatu vyplývá, že je-li jazyk L přijímán zásobníkovým automatem, pak je také přijímán rozšířeným zásobníkovým automatem. Ukažme nyní, že platí i opačná implikace.

Věta 4.11 Necht $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je rozšířený zásobníkový automat. Pak existuje zásobníkový automat P_1 takový, že $L(P_1) = L(P)$.

Důkaz: Položme $m = \max\{|\alpha| \mid \delta(q, a, \alpha) \neq \emptyset \text{ pro nějaké } q \in Q \text{ a } a \in \Sigma \cup \{\epsilon\}\}$

Zásobníkový automat P_1 budeme konstruovat tak, aby simuloval automat P . Protože automat P neurčuje přechody podle vrcholu zásobníku, ale podle vrcholového řetězce zásobníku, bude automat P_1 ukládat m vrcholových symbolů v jakési „vyrovnávací paměti“ řídicí jednotky tak, aby na počátku každého přechodu věděl, jakých m vrcholových symbolů je v zásobníku automatu P . Nahrazuje-li automat P k vrcholových symbolů řetězcem délky l , pak se totéž provede ve vyrovnávací paměti automatu P_1 . Jestliže $l < k$, pak P_1 realizuje $k - l$ ϵ -přechodů, které přesouvají $k - l$ symbolů z vrcholu zásobníku do vyrovnávací paměti. Automat P_1 pak může simulovat další přechod automatu P . Je-li $l \geq k$, pak se symboly přesouvají z vyrovnávací paměti do zásobníku.

Formálně můžeme konstrukci zásobníkového automatu P_1 popsat takto:

$$P_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, Z_1, F_1) \quad \text{kde}$$

$$(1) \quad Q_1 = \{[q, \alpha] \mid q \in Q, \alpha \in \Gamma_1^* \text{ a } 0 \leq |\alpha| \leq m\}$$

$$(2) \quad \Gamma_1 = \Gamma \cup \{Z_1\}$$

(3) Zobrazení δ_1 je definováno takto:

a) Předpokládejme, že $\delta(q, a, X_1 \dots X_k)$ obsahuje $(r, Y_1 \dots Y_l)$.

(i) Jestliže $l \geq k$, pak pro všechna $Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| = m - k$

$$\delta_1([q, X_1 \dots X_k \alpha], a, Z) \text{ obsahuje } ([r, \beta], \gamma Z)$$

kde $\beta\gamma = Y_1 \dots Y_l \alpha$ a $|\beta| = m$.

(ii) Je-li $l < k$, pak pro všechna $Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| = m - k$

$$\delta_1([q, X_1 \dots X_k \alpha], a, Z) \text{ obsahuje } ([r, Y_1 \dots Y_l \alpha Z], \epsilon)$$

b) Pro všechna $q \in Q$, $Z \in \Gamma_1$ a $\alpha \in \Gamma_1$ taková, že $|\alpha| < m$

$$\delta_1([q, \alpha], \epsilon, Z) = \{([q, \alpha, Z], \epsilon)\}$$

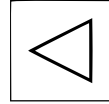
Tato pravidla vedou k naplnění vyrovnávací paměti (obsahuje m symbolů).

(4) $q_1 = [q_0, Z_0, Z_1^{m-1}]$. Vyrovnávací paměť obsahuje na počátku symbol Z_0 na vrcholu a $m - 1$ symbolů Z_1 na dalších místech. Symboly Z_1 jsou speciální znaku pro označení dna zásobníku.

$$(5) \quad F_1 = \{[q, \alpha] \mid q \in F, \alpha \in \Gamma_1^*\}$$

Lze ukázat, že

$$(q, aw, X_1 \dots X_k X_{k+1} \dots X_n) \vdash_P (r, w, Y_1 \dots Y_l X_{k+1} \dots X_n)$$



Ekvivalence
RZA a ZA

platí, právě když $([q, \alpha], aw, \beta) \vdash_{P_1}^+ ([r, \alpha'], w, \beta')$ kde

$$\begin{aligned}\alpha\beta &= X_1 \dots X_n Z_1^m \\ \alpha'\beta' &= Y_1 \dots Y_l X_{k+1} \dots X_n Z_1^m \\ |\alpha| &= |\alpha'| = m\end{aligned}$$

a mezi těmito dvěma konfiguracemi automatu P_1 není žádná konfigurace, ve které by druhý člen stavu (vyrovnávací paměť) měl délku m .

Tedy relace

$$(g_0, w, Z_0) \vdash_P (q, \epsilon, \alpha) \quad \text{pro } q \in F, \alpha \in \Gamma^*$$

platí, právě když

$$([q_0, Z_0, Z_1^{m-1}], w, Z_1) \vdash_{P_1}^* ([q, \beta], \epsilon, \gamma)$$

kde $|\beta| = m$ a $\beta\gamma = \alpha Z_1^m$. Tedy $L(P) = L(P_1)$. \square

Definice 4.21 Necht $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový nebo rozšířený zásobníkový automat. Řetězec w je přijímán s *vyprázdněním zásobníku*, jestliže platí $(q_0, w, Z_0) \vdash^+(q, \epsilon, \epsilon)$, $q \in F$. Označme $L_\epsilon(P)$ množinu všech řetězců, které jsou přijímány zásobníkovým automatem P s vyprázdněním zásobníku.

Věta 4.12 Necht L je jazyk přijímaný zásobníkovým automatem $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, $L = L(P)$. Lze zkonstruovat zásobníkový automat P' takový, že $L_\epsilon(P') = L$.

Důkaz: Opět budeme konstruovat automat P' tak, aby simuloval automat P . Kdykoli automat P dospěje do koncového stavu, přejde do speciálního stavu q_ϵ , který způsobí vyprázdnění zásobníku. Musíme však uvážit situaci, kdy automat P je v konfiguraci s prázdným zásobníkem, nikoliv však v koncovém stavu. Abychom zabránili případům, že automat P' přijímá řetězec, který nemá být přijat, přidáme k zásobníkové abecedě automatu P' znak, jenž bude označovat dno zásobníku a může být vybrán pouze tehdy, je-li automat P' ve stavu q_ϵ . Formálně vyjádřeno, necht

$$P = (Q \cup \{q_\epsilon, q'\}, \Sigma, \Gamma \cup \{Z'\}, \delta', q', Z', \emptyset), \quad \text{kde}$$

symbolem \emptyset vyznačujeme, že P přijímá řetězec s vyprázdněním zásobníku. Zobrazení δ nyní definujeme takto:

- (1) Jestliže $\delta(q, a, Z)$ obsahuje (r, γ) , pak $\delta'(q, a, Z)$ obsahuje (r, γ) pro všechna $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$ a $Z \in \Gamma$.
- (2) $\delta(q', \epsilon, Z') = \{(q_0, Z_0 Z')\}$. První přechod zásobníkového automatu P' uloží do zásobníku řetězec $Z_0 Z'$, kde Z' je speciální znak označující dno zásobníku, a přejde do počátečního stavu automatu P .
- (3) Pro všechna $q \in F$ a $Z \in \Gamma \cup \{Z'\}$ obsahuje $\delta'(q, \epsilon, Z)$ prvek (q_ϵ, ϵ) .
- (4) Pro všechna $Z \in \Gamma \cup \{Z'\}$ je $\delta(q_\epsilon, \epsilon, Z) = \{(q_\epsilon, \epsilon)\}$.

Nyní zřejmě platí

$$\begin{aligned}(q', w, Z') &\vdash_{P'} (q_0, w, Z_0 Z') \\ &\vdash_{P'}^n (q, \epsilon, Y_1 \dots Y_r) \\ &\vdash_{P'} (q_\epsilon, \epsilon, Y_2 \dots Y_r) \\ &\vdash_{P'}^{r-1} (q_\epsilon, \epsilon, \epsilon) \quad \text{kde } Y_r = Z',\end{aligned}$$

DEF



právě když

$$(q_0, w, Z_0) \vdash_P^n (q, \epsilon, Y_1 \dots Y_{r-1})$$

pro $q \in F$ a $Y_1 \dots Y_{r-1} \in \Gamma^*$. Tudíž $L_\epsilon(P') = L(P)$.

Předchozí věta platí také obráceně

□

Věta 4.13 Necht $P = (q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ je zásobníkový automat přijímající vyprázdněním zásobníku. Lze zkonstruovat zásobníkový automat P' takový, že $L(P') = L_\epsilon(P)$.

Důkaz: Zásobníkový automat P' konstruujeme tak, že má speciální symbol Z' na dně zásobníku. Jakmile je tento symbol ze zásobníku odstraněn, přechází automat P' do nového koncového stavu $q_f : F' = \{q_f\}$. Formální konstrukci automatu P' nebudeme provádět.

□

4.11 Ekvivalence bezkontextových jazyků a jazyků přijímaných zásobníkovými automaty

Nejdříve ukážeme, jak lze zkonstruovat nedeterministický syntaktický analyzátor jazyka generovaného bezkontextovou gramatikou, který pracuje metodou shora–dolů.

Věta 4.14 Necht $G = (N, \Sigma, P, S)$ je bezkontextová gramatika. Z gramatiky G můžeme zkonstruovat zásobníkový automat R takový, že $L_\epsilon(R) = L(G)$.

Důkaz: Zásobníkový automat R konstruujeme tak, aby vytvářel levou derivaci vstupního řetězce v gramatice G . Necht $R = (\{q\}, \Sigma, N \cup \Sigma, \delta, S, \emptyset)$, kde δ je definováno takto:

(1) Je-li $A \rightarrow \alpha$ pravidlo z P , pak $\delta(q, \epsilon, A)$ obsahuje (q, α) .

(2) $\delta(q, a, a) = \{(q, \epsilon)\}$ pro všechna $a \in \Sigma$.

Nyní ukážeme, že $A \Rightarrow^m w$, právě když a jen když $(q, w, \epsilon) \vdash^n (q, \epsilon, \epsilon)$ pro nějaké $m, n \geq 1$.

Část „jen když“ dokážeme indukcí pro m . Předpokládejme, že $A \Rightarrow^m w$. Je-li $m = 1$ a $w = a_1 \dots a_k, k \geq 0$ pak

$$\begin{aligned} (q, a_1 \dots a_k, A) &\vdash (q, a_1 \dots a_k) \\ &\vdash^k (q, \epsilon, \epsilon) \end{aligned}$$

Předpokládejme nyní, že platí $A \Rightarrow^m w$ pro $m > 1$. První krok této derivace musí mít tvar $A \Rightarrow X_1 X_2 \dots X_k$, přičemž $X_i \Rightarrow^{m_i} x_i$ pro $m_i < m, 1 \leq i \leq k$ a $x_1 x_2 \dots x_k = w$. Tedy

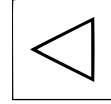
$$(q, w, A) \vdash (q, w, X_1 \dots X_k)$$

Je-li $X_i \in N$, pak podle induktivního předpokladu

$$(q, x_i, X_i) \vdash^* (q, \epsilon, \epsilon)$$

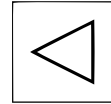
Je-li $X_i = x_i, x_i \in \Sigma$, pak

$$(q, x_i, X_i) \vdash (q, \epsilon, \epsilon)$$



$$\mathcal{L}_2 = \mathcal{L}_P$$

$$\mathcal{L}_2 \subseteq \mathcal{L}_P$$



Nyní již vidíme, že levé derivaci

$$A \Rightarrow X_1 \dots X_k \Rightarrow^{m_1} x_1 X_2 \dots X_k \Rightarrow^{m_2} \dots \Rightarrow^{m_k} x_1 x_2 \dots x_k = w$$

odpovídá tato posloupnost přechodů:

$$(q, x_1 \dots x_k, A) \vdash (q, x_1 \dots x_k, X_1 \dots X_k) \vdash^* (q, x_2 \dots x_k, X_2 \dots X_k) \dots \vdash^* (q, \epsilon, \epsilon)$$

Část „když“, tj. je-li $(q, w, A) \vdash^n (q, \epsilon, \epsilon)$, pak $\Rightarrow^+ w$ dokážeme indukcí pro n . Pro $n = 1$, $w = \epsilon$ a $A \rightarrow \epsilon$ je pravidlo v P . Předpokládejme, že dokazovaná relace platí pro všechna $n' < n$. Pak první přechod automatu R musí mít tvar

$$(q, w, A) \vdash (q, w, X_1 \dots X_k)$$

a $(q, x_i, X_i) \vdash^{n_i} (q, \epsilon, \epsilon)$ pro $1 \leq i \leq k$, kde $w = x_1 \dots x_k$. Pak $A \rightarrow X_1 \dots X_k$ je pravidlo z P a derivace $X_i \Rightarrow^+ x_i$ plyne z indukčního předpokladu. Je-li $X_i \in \Sigma$ pak $X_i \Rightarrow^0 x_i$.

Tedy $A \Rightarrow X_1 \dots X_k \Rightarrow^* x_1 X_2 \dots X_k \Rightarrow^* \dots \Rightarrow^* x_1 \dots x_{k-1} X_k \Rightarrow^* x_1 \dots x_{k-1} x_k = w$ je levá derivace řetězce w z A .

Vezmeme-li $S = A$ jako speciální případ, dostaneme $S \Rightarrow^+ w$, právě když $(q, w, s) \vdash^+ (q, \epsilon, \epsilon)$ a tedy $L_\epsilon(R) = L(G)$. □

Příklad 4.22 Ke gramatice G s pravidly

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid i \end{aligned}$$

$x + y$

sestrojíme zásobníkový automat P takový, že $L_\epsilon(P) = L(G)$

Řešení:

$$P = (\{q\}, \{+, *, (,), i\}, \{+, *, (,), i, E, T, F\}, \delta, q, E, \emptyset)$$

kde δ je zobrazení

$$\begin{aligned} \delta(q, \epsilon, E) &= \{(q, E + T), (q, T)\} \\ \delta(q, \epsilon, T) &= \{(q, T * F), (q, F)\} \\ \delta(q, \epsilon, F) &= \{(q, (E)), (q, i)\} \\ \delta(q, a, a) &= \{(q, \epsilon)\} \quad \text{pro všechna } a \in \{+, *, (,), i\} \end{aligned}$$

Pro vstupní řetězec $i + i * i$ může zásobníkový automat P realizovat tuto posloup-

nost přechodů:

$$\begin{aligned}
(q, i + i * i, E) &\vdash (q, i + i * i, E + T) \\
&\vdash (q, i + i * i, T + T) \\
&\vdash (q, i + i * i, F + T) \\
&\vdash (q, i + i * i, i + T) \\
&\vdash (q, +i * i, +T) \\
&\vdash (q, i * i, T) \\
&\vdash (q, i * i, T * F) \\
&\vdash (q, i * i, F * F) \\
&\vdash (q, i * i, i * F) \\
&\vdash (q, *i, *F) \\
&\vdash (q, i, F) \\
&\vdash (q, i, i) \\
&\vdash (q, \epsilon, \epsilon)
\end{aligned}$$

Nyní ukážeme, jakým způsobem lze k bezkontextové gramatice G definovat rozšířený zásobníkový automat reprezentující syntaktický analyzátor zdola–nahoru.

Věta 4.15 Necht $G = (N, \Sigma, P, S)$ je bezkontextová gramatika. Rozšířený zásobníkový automat $R = (\{q, r\}, \Sigma, N \cup \Sigma \cup \{\$, \delta, q, \$, \{r\}\})$, kde zobrazení δ je definováno takto:

- (1) $\delta(q, a, \epsilon) = \{(q, a)\}$ pro všechna $a \in \Sigma$.
- (2) Je-li $A \rightarrow \alpha$ pravidlo z P , pak $\delta(q, \epsilon, \alpha)$ obsahuje (q, A) .
- (3) $\delta(q, \epsilon, S\$) = \{(r, \epsilon)\}$,

přijímá jazyk $L(G)$, tj. $L(R) = L(G)$.

Důkaz: Ukážeme, že automat R pracuje tak, že vytváří pravou derivaci postupnými redukcemi l -fráze počínaje terminálním řetězcem (umístěným na vstupní pásce) a konče výchozím symbolem S . Jinými slovy automat R realizuje syntaktickou analýzu zdola–nahoru.

Změna konvence: Vrchol zásobníku budeme nyní uvádět *vpravo*. Induktivní hypotéza, kterou dokážeme indukcí pro n má tvar:

$$S \Rightarrow_{rm}^* \alpha A y \Rightarrow_{rm}^n xy \text{ implikuje } (q, xy, \$) \vdash^* (q, y, \$\alpha A), \quad \Rightarrow \text{ značí pravou derivaci.}$$

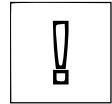
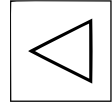
Pro $n = 0$ tato hypotéza platí, protože $\alpha A = x$ a tudíž automat R provádí přechody podle $\delta(q, a, \epsilon) = \{(q, a)\}$, $a \in \Sigma$, přesouvající řetězec x do zásobníku.

Nyní předpokládejme, že induktivní hypotéza platí pro všechny derivace, kratší než n . Můžeme psát

$$\alpha A \gamma \Rightarrow_{rm} \alpha \beta y \Rightarrow_{rm}^{n-1} xy$$

Je-li řetězec $\alpha \beta$ tvořen pouze terminálními symboly, pak $\alpha \beta = x$ a $(q, xy, \$) \vdash^* (q, y, \$\alpha \beta) \vdash (q, y, \$\alpha A)$. Není-li $\alpha \beta \in \Sigma^*$, pak můžeme psát $\alpha \beta = \gamma B z$, kde B je nejpravější nonterminál a podle induktivní hypotézy

$$S \Rightarrow_{rm}^* \gamma B z y \Rightarrow_{rm}^{n-1} xy$$



implikuje $(q, xy, \$) \vdash^* (q, zy, \$\gamma B)$. Platí tedy $(q, zy, \$\gamma B) \vdash^* (q, y, \$\gamma Bz) \vdash (q, y, \$\alpha A)$ a tudíž jsme dokázali indukativní hypotézu. Protože $(q, \epsilon, \$S) \vdash (r, \epsilon, \epsilon)$ dostáváme $L(G) \subseteq L(R)$.

Abychom dokázali, že $L(R) \subseteq L(G)$, ukažme, že platí implikace:

$$\text{Jestliže } (q, xy, \$) \vdash^n (q, y, \$\alpha A), \text{ pak } \alpha Ay \Rightarrow^* xy$$

Pro $n = 0$ tato implikace platí. Předpokládejme, že platí také pro všechny posloupnosti přechodů kratší než n . Protože vrchol zásobníku je nonterminální symbol, prováděl se poslední přechod podle předpisu (2) v zobrazení δ a můžeme psát:

$$(q, xy, \$) \vdash^{n-1} (q, y, \$\alpha\beta) \vdash (q, y, \$\alpha A)$$

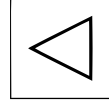
kde $A \rightarrow \beta$ je pravidlo z P . Existuje tedy derivace $\alpha Ay \Rightarrow \alpha\beta y \Rightarrow^* xy$.

Jako speciální případ uvažujme implikaci:

$$\text{je-li } (q, w, \$) \vdash^* (q, \epsilon, \$S), \text{ pak } S \Rightarrow^* w$$

Protože však platí $(q, w, \$) \vdash^* (q, \epsilon, \$S) \vdash (r, \epsilon, \epsilon)$ je $L(R) \subseteq L(G)$ a společně s první částí důkazu dostáváme $L(G) = L(R)$. □

Poznamenejme, že automat R skutečně představuje model syntaktického analyzátoru, který vytváří pravou derivaci vstupního řetězce postupnými redukcemi l -fráze větných forem (počáteční větná forma je vstupní řetězec, koncová větná forma je výchozí symbol gramatiky). Bezprostředně po přechodu automatu je pravá větná forma αAx reprezentována obsahem zásobníku (řetězec αA) a ne-zpracovanou částí vstupního řetězce (řetězec x). Následující činností automatu je přesunutí prefixu řetězce x na vrchol zásobníku a redukce l -fráze dané vrcholovým řetězcem zásobníku. Tento typ syntaktické analýzy se, jak již víme, nazývá syntaktická analýza zdola–nahoru.



Příklad 4.23 Sestrojme syntaktický analyzátor R zdola–nahoru pro gramatiku s pravidly

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid i \end{aligned}$$

Nechť R je rozšířený zásobníkový automat

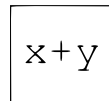
$$R = (\{q, r\}, \{+, *, (,), i\}, \{\$, E, T, F, +, *, (,), i\}, \delta, q, \$, \{r\})$$

kde δ je zobrazení

$$(1) \delta(q, a, \epsilon) = \{(q, a)\} \text{ pro všechna } a \in \{i, +, *, (,)\}.$$

(2)

$$\begin{aligned} \delta(q, \epsilon, E + T) &= \{(q, E)\} \\ \delta(q, \epsilon, T) &= \{(q, E)\} \\ \delta(q, \epsilon, T * F) &= \{(q, T)\} \\ \delta(q, \epsilon, F) &= \{(q, T)\} \\ \delta(q, \epsilon, (E)) &= \{(q, F)\} \\ \delta(q, \epsilon, i) &= \{(q, F)\} \end{aligned}$$



$$(3) \delta(q, \epsilon, \$E) = \{(r, \epsilon)\}$$

Pro vstupní řetězec $i + i * i$ může rozšířený zásobníkový automat R provést tuto posloupnost přechodů:

$$\begin{aligned}
 (q, i + i * i, \$) &\vdash (q, +i * i, \$i) \\
 &\vdash (q, +i * i, \$F) \\
 &\vdash (q, +i * i, \$T) \\
 &\vdash (q, +i * i, \$E) \\
 &\vdash (q, i * i, \$E+) \\
 &\vdash (q, *i, \$E + i) \\
 &\vdash (q, *i, \$E + F) \\
 &\vdash (q, *i, \$E + T) \\
 &\vdash (q, i, \$E + T*) \\
 &\vdash (q, \epsilon, \$E + T * i) \\
 &\vdash (q, \epsilon, \$E + T * F) \\
 &\vdash (q, \epsilon, \$E + T) \\
 &\vdash (q, \epsilon, \$E) \\
 &\vdash (r, \epsilon, \epsilon)
 \end{aligned}$$

Na závěr tohoto odstavce ukážeme, že bezkontextové jazyky jsou ekvivalentní jazykům přijímaným zásobníkovými automaty, tj., že lze také ke každému zásobníkovému automatu R vytvořit bezkontextovou gramatiku G takovou, že $L(R) = L(G)$.

Věta 4.16 Necht $R = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. Pak existuje bezkontextová gramatika G , pro kterou platí $L(G) = L(R)$.

Důkaz: Gramatiku G budeme konstruovat tak, aby levá derivace terminálního řetězce w přímo korespondovala s posloupností přechodů automatu R . Budeme používat nonterminální symboly tvaru $[qZr]$ kde $q, r \in Q, Z \in \Gamma$. Vrchol zásobníku uvádíme opět *vlevo*.

Necht gramatika $G = (N, \Sigma, P, S)$ je formálně definována takto:

- (1) $N = \{[qZr] \mid q, r \in Q, Z \in \Gamma\} \cup \{S\}$
- (2) Jestliže $\delta(q, a, Z)$ obsahuje $(r, X_1 \dots X_k)$ $k \geq 1$, pak k množině přepisovacích pravidel P přidej všechna pravidla tvaru

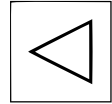
$$[qZs_k] \rightarrow a[rX_1s_1][s_1X_2s_2] \dots [s_{k-1}X_k s_k]$$

pro každou posloupnost s_1, s_2, \dots, s_k stavů z množiny Q .

- (3) Jestliže $\delta(q, a, Z)$ obsahuje (r, ϵ) , pak k P přidej pravidlo $[qZr] \rightarrow a$.

- (4) Pro každý stav $q \in Q$ přidej k P pravidlo $S \rightarrow [q_0Z_0q]$.

Indukcí lze opět dokázat, že pro všechna $q, r \in Q$ a $Z \in \Gamma$ platí $[qZr] \Rightarrow^m w$, právě když $(q, w, Z) \vdash^n (r, \epsilon, \epsilon)$. Speciální případ této ekvivalence je $S \Rightarrow [q_0Z_0q] \Rightarrow^+ w$, právě když $(q_0, w, Z_0) \vdash^+ (q, \epsilon, \epsilon), q \in F$. To však znamená, že $L_\epsilon(R) = L(G)$. \square



$$\mathcal{L}_P \subseteq \mathcal{L}_2$$

Příklad 4.24 Necht $P = (\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, \delta, q_0, Z_0, \emptyset)$ kde zobrazení δ je dáno takto:

$$x + y$$

$$\begin{aligned}\delta(q_0, 0, Z_0) &= \{(q_0, XZ_0)\} \\ \delta(q_0, 0, X) &= \{(q_0, XX)\} \\ \delta(q_0, 1, X) &= \{(q_1, \epsilon)\} \\ \delta(q_1, 1, X) &= \{(q_1, \epsilon)\} \\ \delta(q_1, \epsilon, X) &= \{(q_1, \epsilon)\} \\ \delta(q_1, \epsilon, Z_0) &= \{(q_1, \epsilon)\}\end{aligned}$$

Zkonstruujeme gramatiku $G = (N, \Sigma, P, S)$ takovou, že $L(P) = L(G)$. Množiny nonterminálů a terminálů mají tvar:

$$\begin{aligned}N &= \{S, [q_0Xq_0], [q_0Xq_1], [q_1Xq_0], [q_1Xq_1], [q_0Z_0q_0], [q_0Z_0q_1], [q_1Z_0q_0], [q_1Z_0q_1]\} \\ \Sigma &= \{0, 1\}\end{aligned}$$

Množina N obsahuje některé nonterminály, které jsou v gramatice G nedostupné. Abychom je nemuseli dodatečně odstraňovat, začneme konstruovat pravidla gramatiky G počínaje S -pravidly a přidávejme pouze ty nonterminály, které se objevují na pravých stranách konstruovaných pravidel.

Podle bodu (4) sestrojíme S -pravidla:

$$S \rightarrow [q_0Z_0q_0] \quad S \rightarrow [q_0Z_0q_1]$$

Nyní přidáme pravidla pro nonterminál $[q_0Z_0q_0]$

$$\begin{aligned}[q_0Z_0q_0] &\rightarrow 0[q_0Xq_0][q_0Z_0q_0] \\ [q_0Z_0q_0] &\rightarrow 0[q_0Xq_1][q_1Z_0q_0]\end{aligned}$$

vyplývající z $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$.

Pravidla pro nonterminál $[q_0Z_0q_1]$

$$\begin{aligned}[q_0Z_0q_1] &\rightarrow 0[q_0Xq_0][q_0Z_0q_1] \\ [q_0Z_0q_1] &\rightarrow 0[q_0Xq_1][q_1Z_0q_1]\end{aligned}$$

jsou požadována zobrazením $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$.

Pravidla pro zbývající nonterminály jsou:

$$\begin{aligned}[q_0Xq_0] &\rightarrow 0[q_0Xq_0][q_0Xq_0] \\ [q_0Xq_0] &\rightarrow 0[q_0Xq_1][q_1Xq_0] \\ [q_0Xq_1] &\rightarrow 0[q_0Xq_0][q_0Xq_1] \\ [q_0Xq_1] &\rightarrow 0[q_0Xq_1][q_1Xq_1]\end{aligned}$$

protože, $\delta(q_0, 0, X) = \{(q_0, XX)\}$;

$$\begin{aligned}[q_0Xq_1] &\rightarrow 1, \text{ protože } \delta(q_0, 1, X) = \{(q_1, \epsilon)\} \\ [q_1Z_0q_1] &\rightarrow \epsilon, \text{ protože } \delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\} \\ [q_1Xq_1] &\rightarrow \epsilon, \text{ protože } \delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\} \\ [q_1Xq_1] &\rightarrow 1, \text{ protože } \delta(q_1, 1, X) = \{(q_1, \epsilon)\}.\end{aligned}$$

Povšimněme si nyní, že pro nonterminály $[q_1Xq_0]$ a $[q_1Z_0q_0]$ nejsou definována žádná pravidla, a proto ani z nonterminálu $[q_0Z_0q_0]$ ani z $[q_0Xq_0]$ nemohou

být derivovány terminální řetězce. Odstraníme-li tedy pravidla obsahující některý z uvedených čtyř nonterminálů, dostaneme ekvivalentní gramatiku generující jazyk $L(P)$:

$$\begin{aligned}
S &\rightarrow [q_0 Z_0 q_1] \\
[q_0 Z_0 q_1] &\rightarrow 0[q_0 X q_1][q_1 Z_0 q_1] \\
[q_0 X q_1] &\rightarrow 0[q_0 X q_1][q_1 X q_1] \\
[q_1 Z_0 q_1] &\rightarrow \epsilon \\
[q_0 X q_1] &\rightarrow 1 \\
[q_1 X q_1] &\rightarrow \epsilon \\
[q_1 X q_1] &\rightarrow 1
\end{aligned}$$

4.12 Deterministický zásobníkový automat

V předchozím odstavci jsme ukázali, že ke každé bezkontextové gramatice lze sestavit zásobníkový automat, který reprezentuje syntaktický analyzátor pro věty generované danou gramatikou. Tento analyzátor je obecně nedeterministický. Z hlediska aplikací teorie formálních jazyků v překladačích jsou důležité tzv. deterministické bezkontextové jazyky, které lze analyzovat deterministickými syntaktickými analyzátory. V tomto odstavci definujeme třídu zásobníkových automatů, které v každém okamžiku mohou přijít pouze do jediné konfigurace, tzv. deterministické zásobníkové automaty, a jim odpovídající deterministické jazyky.

Definice 4.22 Zásobníkový automat $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ nazýváme deterministickým zásobníkovým automatem, jestliže pro každé $q \in Q$ a $Z \in \Gamma$ platí buď pro každé $a \in \Sigma$ obsahuje $\delta(q, a, Z)$ nanejvýš jeden prvek a $\delta(q, \epsilon, Z) = \emptyset$, nebo $\delta(q, a, Z) = \emptyset$ pro všechna $a \in \Sigma$ a $\delta(q, \epsilon, Z)$ obsahuje nejvýše jeden prvek.

Protože zobrazení $\delta(q, a, Z)$ obsahuje nejvýše jeden prvek, budeme místo $\delta(q, a, Z) = \{(r, \gamma)\}$ psát $\delta(q, a, Z) = (r, \gamma)$.

Příklad 4.25 Deterministický zásobníkový automat, který přijímá jazyk $L = \{wcw^R \mid w \in \{a, b\}^+\}$, má tvar:

$$P = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{Z, a, b\}, \delta, q_0, Z, \{q_2\})$$

kde zobrazení δ je definováno takto:

$$\begin{aligned}
\delta(q_0, X, Y) &= (q_0, XY) \quad \text{pro všechna } X \in \{a, b\} \text{ a } Y \in \{Z, a, b\} \\
\delta(q_0, c, Y) &= (q_1, Y) \quad \text{pro všechna } Y \in \{a, b\} \\
\delta(q_1, X, X) &= (q_1, \epsilon) \quad \text{pro všechna } X \in \{a, b\} \\
\delta(q_1, \epsilon, Z) &= (q_2, \epsilon)
\end{aligned}$$

Automat P pracuje tak, že dokud nepřečte středový symbol c , ukládá symboly vstupního řetězce do zásobníku. Potom přejde do stavu q_1 a srovnává další vstupní symboly se symboly zásobníku.

Mezi zásobníkovým automatem a deterministickým zásobníkovým automatem bohužel neplatí vztah jako mezi nedeterministickým a deterministickým konečným automatem, tzn., že ne každý jazyk přijímaný zásobníkovým automatem může být přijímán deterministickým zásobníkovým automatem.

DEF

$x + y$

Definice 4.23 Jazyk L se nazývá *deterministický bezkontextový jazyk*, jestliže existuje deterministický zásobníkový automat P takový, že $L(P) = L$.

DEF

Definici deterministického zásobníkového automatu můžeme rozšířit tak, aby zahrnovala rozšířené zásobníkové automaty.

Definice 4.24 Rozšířený zásobníkový automat $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ nazýváme *deterministický RZA* (DRZA), jestliže platí:

DEF

1. $\forall q \in Q \forall a \in \Sigma \cup \{\varepsilon\} \forall \gamma \in \Gamma^* : |\delta(q, a, \gamma)| \leq 1$.
2. Je-li $\delta(q, a, \alpha) \neq \emptyset$, $\delta(q, a, \beta) \neq \emptyset$ a $\alpha \neq \beta$, pak ani α není předponou β , ani β není předponou α .
3. Je-li $\delta(q, a, \alpha) \neq \emptyset$ a $\delta(q, \varepsilon, \beta) \neq \emptyset$, pak ani α není předponou β , ani β není předponou α .

Pozor na použití $\alpha \neq \beta$ v bodě 2 ve výše uvedené definici a nepoužití této podmínky v bodě 3: V bodě 2 dostáváme pro $\alpha = \beta$ tutéž trojici $(q, a, \alpha) = (q, a, \beta)$ a nedochází tedy k nedeterminismu na rozdíl od bodu 3, kde vznikají dvě různé dvojice $(q, a, \alpha) = (q, a, \beta)$ a $(q, \varepsilon, \alpha) = (q, \varepsilon, \beta)$.

!

Věta 4.17 Deterministické rozšířené zásobníkové automaty mají ekvivalentní vyjadřovací sílu jako deterministické zásobníkové automaty.

Věta 4.18 Deterministické zásobníkové automaty mají striktně menší vyjadřovací sílu než nedeterministické zásobníkové automaty.

◁

Důkaz: i (z důvodu zjednodušení je zde uvedena pouze idea důkazu) Bezkontextový jazyk $L = \{ww^R \mid w \in \Sigma^+\}$ nelze přijímat žádným DZA. Neformálně řečeno, DZA nemá možnost uhádnout, kdy končí w a začíná w^R .

□

Poznámka 4.5 Jiná možnost důkazu věty 4.18 je přes následně uvedenou uzavřenost jazyků DZA vůči doplňku a přes uvážení, že $\{a^n b^n c^n \mid n \geq 1\}$ je bezkontextový jazyk.

◁

Problém, zda daný bezkontextový jazyk je jazykem nějakého DZA, *není obecně rozhodnutelný* (podobně jako není rozhodnutelná víteznačnost).

4.13 Vlastnosti bezkontextových jazyků

V této podkapitole se budeme zabývat některými vlastnostmi bezkontextových jazyků v podobné podobě jako v případě regulárních jazyků.

4.13.1 Strukturální vlastnosti

Pumping lemma

Věta 4.19 Necht L je bezkontextový jazyk. Pak existuje konstanta k taková, že je-li $z \in L$ a $|z| \geq k$, pak lze z napsat ve tvaru:

$$z = u v w x y, vx \neq \varepsilon, |vwx| \leq k$$

a pro všechna $i \geq 0$ je $uv^iwx^iy \in L$.

!

Důkaz: Necht $L = L(G)$ a necht $G = (N, \Sigma, P, S)$ je gramatika v CNF.

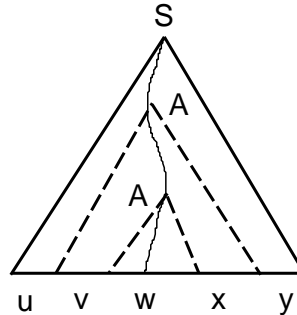
1. Nejprve dokážeme implikaci:

Jestliže $A \Rightarrow^+ w$ pro nějaké $A \in N$, $w \in \Sigma^*$, pak $|w| \leq 2^{m-2}$, kde m je počet vrcholů nejdelší cesty v odpovídajícím derivačním stromu.

Tato implikace platí, protože $|w|$ je rovno počtu přímých předchůdců listů příslušného derivačního stromu, která je maximálně rovna počtu listů plného binárního stromu, jehož všechny větve obsahují $m - 1$ uzlů, což je právě 2^{m-2} . Skutečně:

- Plný binární strom s větvemi o n uzlech, má 2^{n-1} listů, což se snadno ukáže indukcí:
 - Plný binární strom s (jedinou) větví o $n = 1$ uzlu, má $1 = 2^0 = 2^{n-1}$ listů.
 - Plný binární strom s větvemi délky $n = n' + 1$ uzlů, kde $n' \geq 1$, má $2^{n'-1} + 2^{n'-1} = 2 \cdot 2^{n'-1} = 2^{1+n'-1} = 2^{n'} = 2^{n-1}$ listů.
- Postačí tedy volit $n = m - 1$, přičemž případ neplných binárních stromů není třeba uvažovat, neboť se zajímáme o stromy s maximálním počtem listů při dané maximální délce větví.

2. Položme $k = 2^{|N|}$ a uvažujme libovolnou větu z takovou, že $|z| \geq k$. Označíme-li m počet vrcholů nejdelší cesty v odpovídajícím derivačním stromu, pak $2^{|N|} \leq 2^{m-2}$ a taková cesta pak obsahuje alespoň $|N| + 2$ vrcholů ($|N| + 2 \leq m$). Z těchto $|N| + 2$ vrcholů je jeden terminál a nutně alespoň dva jsou označeny stejným nonterminálem, řekněme A . Viz obrázek vpravo.



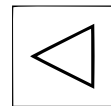
Řetězce v, x nemohou být prázdné, protože aplikované pravidlo mělo tvar $A \rightarrow BC$. Nyní uvažujme derivaci řetězce z tvaru: $S \Rightarrow^* uAy \Rightarrow^+ uvAxy \Rightarrow^+ uvwxy = z$

To pak ovšem znamená, že v G existuje rovněž derivace: $S \Rightarrow^* uAy \Rightarrow^+ uvAxy \Rightarrow^+ uvvAxy \Rightarrow^+ uv^2wx^2y$, protože $A \Rightarrow^+ w$, a tedy derivace $S \Rightarrow^* uv^iwx^i y$ pro libovolné $i > 0$, což je dokazované tvrzení. □

Lemma 4.2 Jazyk $L = \{a^n b^n c^n \mid n \geq 1\}$ není bezkontextovým jazykem.

Důkaz: Důkaz provedeme aplikací pumping teoremu: Nelze zvolit řetězce v a x tak, aby jejich iterací zůstal stejný počet symbolů a, b, c a současně pořadí symbolů a, b, c zůstalo nezměněno. □

Poznámka 4.6 Jazyk $L = \{a^n b^n c^n \mid n \geq 1\}$ je typickým kontextovým jazykem.



4.13.2 Uzávěrové vlastnosti

Substituce jazyků

Definice 4.25 Necht \mathcal{L} je třída jazyků a necht $L \subseteq \Sigma^*$ je jazykem třídy \mathcal{L} . Dále necht $\Sigma = \{a_1, a_2, \dots, a_n\}$ pro nějaké $n \in \mathbb{N}$ a necht jazyky označené $L_{a_1}, L_{a_2}, \dots, L_{a_n}$ jsou rovněž jazyky třídy \mathcal{L} . Říkáme, že třída \mathcal{L} je *uzavřena vzhledem k substituci*, jestliže pro každý výběr jazyků $L, L_{a_1}, L_{a_2}, \dots, L_{a_n}$ je také jazyk $\sigma_{L_{a_1}, L_{a_2}, \dots, L_{a_n}}(L)$

DEF

$$\sigma_{L_{a_1}, L_{a_2}, \dots, L_{a_n}}(L) = \{x_1 x_2 \dots x_m \mid b_1 b_2 \dots b_m \in L \wedge \forall i \in \{1, \dots, m\} : x_i \in L_{b_i}\}$$

ve třídě \mathcal{L} .

Příklad 4.26 Necht $L = \{0^n 1^n \mid n \geq 1\}$, $L_0 = \{a\}$, $L_1 = \{b^m c^m \mid m \geq 1\}$. Substitucí jazyků L_0 a L_1 do L dostaneme jazyk

$$L' = \{a^n b^{m_1} c^{m_1} b^{m_2} c^{m_2} \dots b^{m_n} c^{m_n} \mid n \geq 1 \wedge \forall i \in \{1, \dots, n\} : m_i \geq 1\}$$

 $x+y$

Morfismus jazyků

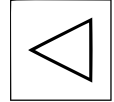
Definice 4.26 Necht Σ a Δ jsou abecedy a $L \subseteq \Sigma^*$ je jazyk nad abecedou Σ . Zobrazení $h : \Sigma^* \rightarrow \Delta^*$ nazveme *morfismem nad slovy*, platí-li $\forall w = a_1 a_2 \dots a_n \in \Sigma^* : h(w) = h(a_1) h(a_2) \dots h(a_n)$. *Morfismus jazyka* $h(L)$ pak definujeme jako $h(L) = \{h(w) \mid w \in L\}$.

Morfismus jazyků je *zvláštní případ substituce*, kde každý substituovaný jazyk má právě jednu větu.

Uzavřenost vůči substituci

Věta 4.20 Třída bezkontextových jazyků je uzavřena vůči substituci.

Důkaz:

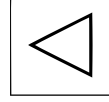


1. Ve shodě s definicí substituce necht $\Sigma = \{a_1, a_2, \dots, a_n\}$ je abeceda bezkontextového jazyka L a L_a pro $a \in \Sigma$ libovolné bezkontextové jazyky. Necht $G = (N, \Sigma, P, S)$ a $G_a = (N_a, \Sigma_a, P_a, S_a)$ pro $a \in \Sigma$ jsou gramatiky, pro které $L = L(G)$ a $L_a = L(G_a)$ pro $a \in \Sigma$.
2. Předpokládejme, že $N \cap N_a = \emptyset$ a $N_a \cap N_b = \emptyset$ pro každé $a, b \in \Sigma$, $a \neq b$. Sestrojme gramatiku $G' = (N', \Sigma', P', S)$ takto:
 - (a) $N' = N \cup \bigcup_{a \in \Sigma} N_a$.
 - (b) $\Sigma' = \bigcup_{a \in \Sigma} \Sigma_a$.
 - (c) Necht h je morfismus na $N \cup \Sigma$ takový, že
 - i. $h(A) = A$ pro $A \in N$ a
 - ii. $h(a) = S_a$ pro $a \in \Sigma$
 a necht $P' = \{A \rightarrow h(\alpha) \mid (A \rightarrow \alpha) \in P\} \cup \bigcup_{a \in \Sigma} P_a$.
3. Uvažujme libovolnou větu $a_{i_1} a_{i_2} \dots a_{i_m} \in L$ a věty $x_j \in L_{a_j}$, $1 \leq j \leq m$. Pak $S \xRightarrow[G']{*} S_{a_{i_1}} S_{a_{i_2}} \dots S_{a_{i_m}} \xRightarrow[G']{*} x_1 S_{a_{i_2}} \dots S_{a_{i_m}} \xRightarrow[G']{*} \dots \xRightarrow[G']{*} x_1 x_2 \dots x_m$ a tedy $L' \subseteq L(G')$. Podobně $L(G') \subseteq L'$.

□

Uzavřenost \mathcal{L}_2 vůči různým jazykovým operacím**Věta 4.21** Bezkontextové jazyky jsou uzavřeny vzhledem k:

1. sjednocení,
2. konkatenaci,
3. iteraci,
4. pozitivní iteraci,
5. morfismu.

Důkaz: Necht L_a a L_b jsou bezkontextové jazyky.

1. Uzavřenost vůči \cup plyne ze substituce L_a, L_b do jazyka $\{a, b\}$.
2. Uzavřenost vůči \cdot plyne ze substituce L_a, L_b do jazyka $\{ab\}$.
3. Uzavřenost vůči $*$ plyne ze substituce L_a do jazyka $\{a\}^*$.
4. Uzavřenost vůči $+$ plyne ze substituce L_a do jazyka $\{a\}^+$.
5. Necht h je daný morfismus a $L'_a = \{h(a)\}$ pro $a \in \Sigma$. Substitucí jazyků L'_a do jazyka L získáme jazyk $h(L)$.

□

Věta 4.22 Bezkontextové jazyky jsou uzavřeny vzhledem k průniku s regulárními jazyky.

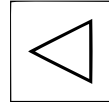
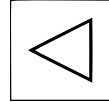
Důkaz: Snadno zkonstruujeme zásobníkový automat přijímající příslušný průnik – konstruujeme průnik na konečném řízení, zásobníkové operace zůstávají.

□

Definice 4.27 Je-li $h : \Sigma^* \rightarrow \Delta^*$ morfismus, pak definujeme *inverzní morfismus nad slovy* z Δ^* jako $h^{-1}(w) = \{x \in \Sigma^* \mid h(x) = w\}$ a *inverzní morfismus jazyka* L nad Δ jako $h^{-1}(L) = \{x \in \Sigma^* \mid h(x) \in L\}$.**Věta 4.23** Bezkontextové jazyky jsou uzavřeny vzhledem k inverznímu morfismu.Důkaz: Mějme libovolný morfismus $h : \Sigma^* \rightarrow \Delta^*$. Zavedme pomocnou abecedu $\bar{\Sigma}$, jež ke každému $a \in \Sigma$ obsahuje \bar{a} takové, že $\bar{a} \notin \Sigma$. Uzavřenost \mathcal{L}_2 vůči inverznímu morfismu plyne z uzavřenosti vůči substituci, průniku s regulárním jazykem a morfismu: $h^{-1}(L) = h_1(\sigma(L) \cap L_1^*)$, kde:

1. σ je substituce taková, že $\forall c \in \Delta : L_c = \bar{\Sigma}^* c \bar{\Sigma}^*$,
2. $L_1 = \{\bar{a}w \mid \bar{a} \in \bar{\Sigma} \wedge w \in \Delta^* \wedge h(a) = w\}$ je regulární jazyk a
3. $h_1 : (\bar{\Sigma} \cup \Delta)^* \rightarrow \Sigma^*$ je morfismus takový, že (1) $\forall \bar{a} \in \bar{\Sigma} : h_1(\bar{a}) = a$ a (2) $\forall c \in \Delta : h_1(c) = \varepsilon$.

□



Neuzavřenost \mathcal{L}_2 vůči průniku a doplňku**Věta 4.24** Bezkontextové jazyky *nejdou* uzavřeny vůči průniku a doplňku.

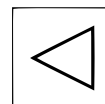
Důkaz:

1. Neuzavřenost vůči
- \cap
- :

Uvažujme jazyky $L_1 = \{a^m b^n c^n \mid n, m \geq 1\}$ a $L_2 = \{a^m b^n c^n \mid m, n \geq 1\}$, které jsou oba bezkontextové. Ovšem $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$, což není bezkontextový jazyk (lze ukázat např. pomocí Pumping lemmatu).

2. Neuzavřenost vůči doplňku: Předpokládejme, že bezk. jazyky jsou uzavřeny vůči doplňku. Z De Morganových zákonů (a z uzavřenosti vůči sjednocení) pak ovšem plyne uzavřenost vůči průniku $L_1 \cap L_2 = \overline{\overline{L_1} \cap \overline{L_2}} = \overline{\overline{L_1} \cup \overline{L_2}}$, což je spor.

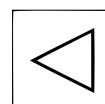
□

**4.13.3 Rozhodnutelné a nerozhodnutelné problémy pro bezkontextové jazyky****Věta 4.25** Následující problémy jsou *rozhodnutelné*, tj. jsou algoritmicky řešitelné:

1. problém *neprázdnosti jazyka* $L(G)$ pro libovolnou bezkontextovou gramatiku G ,
2. problém *příslušnosti řetězce* $w \in \Sigma^*$ *do jazyka* $L(G)$ pro libovolnou bezkontextovou gramatiku G ,
3. problém *konečnosti jazyka* $L(G)$ pro libovolnou bezkontextovou gramatiku G .

Důkaz:

1. K rozhodování neprázdnosti lze využít algoritmus iterativně určující množinu N_t nonterminálů generujících terminální řetězce uvedený v přednášce.
4. Pak $L(G) \neq \emptyset \Leftrightarrow S \in N_t$.
2. U problému příslušnosti řetězce můžeme např. určit průnik NZA s KA přijímajícím právě řetězec w a pak ověřit neprázdnost.
3. Problém konečnosti můžeme rozhodovat na základě platnosti Pumping lemma pro bezkontextové jazyky:
 - (a) Dle Pumping lemma pro bezkontextové jazyky existuje pro každý bezkontextový jazyk L konstanta $k \in \mathbb{N}$ taková, že každou větu $w \in L$, $|w| \geq k$, můžeme rozepsat jako $uvwx$, kde $vx \neq \varepsilon$ a $|vwx| \leq k$, a $\forall i \in \mathbb{N} : uv^iwx^iy \in L$.
 - (b) Pro testování konečnosti tedy postačí ověřit, že žádný řetězec ze Σ^* o délce mezi k a $2k - 1$ nepatří do daného jazyka: Pokud takový řetězec existuje, může být „napumpován“ a dostáváme nekonečně mnoho řetězců patřících do daného jazyka. Jestliže takový řetězec neexistuje, $k - 1$ je horní limit délky řetězců L . Pokud by existoval řetězec délky $2k$ nebo větší patřící do L , můžeme v něm podle Pumping lemma najít vwx a vypustit vx . Vzhledem k tomu, že $0 < |vx| \leq k$, postupným opakováním vypouštění bychom se dostali k nutné existenci řetězce z L o délce mezi k a $2k - 1$.



- (c) K určení konstanty k postačí reprezentovat L pomocí bezkontextové gramatiky v CNF s n nonterminály a zvolit $k = 2^n$ (viz důkaz Pumping lemma).

□

Nerozhodnutelné problémy pro \mathcal{L}_2

Věta 4.26 Následující problémy jsou *nerozhodnutelné*, tj. nejsou algoritmicky řešitelné:

1. problém *ekvivalence jazyků bezkontextových gramatik*, tj. otázka, zda $L(G_1) = L(G_2)$ pro dvě bezkontextové gramatiky G_1, G_2 ,
2. problém *inkluze jazyků bezkontextových gramatik*, tj. otázka, zda $L(G_1) \subseteq L(G_2)$ pro dvě bezkontextové gramatiky G_1, G_2 .

Důkaz: Využívá redukcí z nerozhodnutelného Postova korespondenčního problému – viz dále. Z nerozhodnutelnosti ekvivalence plyne nerozhodnutelnost inkluze ($A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A$).

□

V následujících kapitolách zmíníme i některé další nerozhodnutelné problémy pro \mathcal{L}_2 , mezi něž patří *univerzalita* ($L(G) \stackrel{?}{=} \Sigma^*$), *regularita* apod.

4.13.4 Uzávěrové vlastnosti jazyků deterministických bezkontextových jazyků

Věta 4.27 Deterministické bezkontextové jazyky jsou uzavřeny vůči průniku s regulárními jazyky a doplňku.

Důkaz: (idea) Bod 1 dokážeme podobně jako v případě jazyků přijímaných nedeterministickými zásobníkovými automaty. U bodu 2 postupujeme podobně jako u deterministického konečného automatu – použijeme záměnu koncových a nekoncových stavů, musíme ale navíc řešit dva okruhy problémů: (a) DZA nemusí vždy dočíst vstupní slovo až do konce (buď se dostane do konfigurace, z níž nemůže pokračovat, nebo cyklí přes ε -kroky) a (b) DZA slovo dočte do konce, ale pak ještě provede posloupnost ε -kroků jdoucích přes koncové i nekoncové stavy. Popis řešení těchto problémů je možno nalézt v doporučené literatuře.

□

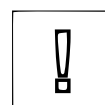
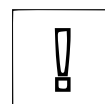
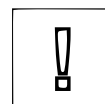
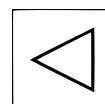
Věta 4.28 Deterministické bezkontextové jazyky *nejdou* uzavřeny vůči průniku a sjednocení.

Důkaz: U bodu 1 použijeme stejný postup jako u NZA (uvědomíme si, že $\{a^m b^m c^n \mid n, m \geq 1\}$ a $\{a^m b^n c^n \mid n, m \geq 1\}$ lze přijímat DZA). Bod 2 plyne z De Morganových zákonů.

□

Věta 4.29 Deterministické bezkontextové jazyky *nejdou* uzavřeny vůči konkatenaci a iteraci.

Důkaz: (idea) Vyjdeme z toho, že zatímco jazyky $L_1 = \{a^m b^m c^n \mid m, n \geq 1\}$ a $L_2 = \{a^m b^n c^n \mid m, n \geq 1\}$ jsou deterministické bezkontextové, jazyk $L_1 \cup L_2$ ne. (Intuitivně DZA nemůže odhadnout, zda má kontrolovat první nebo druhou rovnost, a tedy, zda na zásobník ukládat symboly a nebo b .)



1. Neuzavřenost vůči konkatenci. Jazyk $L_3 = 0L_1 \cup L_2$ je zřejmě deterministický bezkontextový. Jazyk 0^* je také deterministický bezkontextový (dokonce regulární), ovšem není těžké nahlédnout, že 0^*L_3 není deterministický bezkontextový. Stačí uvážit, že $0a^*b^*c^*$ je deterministický bezkontextový (dokonce regulární) jazyk a $0^*L_3 \cap 0a^*b^*c^* = 0L_1 \cup 0L_2 = 0(L_1 \cup L_2)$.
2. Neuzavřenost vůči iteraci. Uvážíme $(\{0\} \cup L_3)^* \cap 0a^+b^+c^+ = 0(L_1 \cup L_2)$.

□

4.13.5 Některé další zajímavé vlastnosti bezkontextových jazyků

Definice 4.28 Bezkontextová gramatika $G = (N, \Sigma, P, S)$ má vlastnost sebevlození, jestliže existují $A \in N$ a $u, v \in \Sigma^+$ takové, že $A \Rightarrow^+ uAv$ a A není zbytečný nonterminál. Bezkontextový jazyk má vlastnost sebevlození, jestliže každá gramatika, která jej generuje, má vlastnost sebevlození.

Věta 4.30 Bezkontextový jazyk má vlastnost sebevlození právě tehdy, když není regulární.

Důkaz: Můžeme využít Greibachovu normální formu (definice 4.15).

□

Uvedená věta může být užitečná při dokazování, že určitá gramatika negeneruje regulární jazyk. Zopakujme ale, že problém, zda daná bezkontextová gramatika generuje regulární jazyk, není algoritmicky rozhodnutelný.

Teorém Chomského a Schützenbergera. Tento teorém postihuje úzkou vazbu bezkontextových jazyků na závorkování.

Definice 4.29 Označme ZAV_n pro $n \geq 0$ jazyky sestávající ze všech vyvážených řetězců závorek n typů. Tyto jazyky – označované též jako Dyckovy jazyky – jsou generovány gramatikami s pravidly tvaru:

$$S \rightarrow [^1 S]^1 \mid [^2 S]^2 \mid \dots \mid [^n S]^n \mid SS \mid \varepsilon$$

Věta 4.31 (Chomsky-Schützenberger) Každý bezkontextový jazyk je morfismem průniku nějakého jazyka závorek a nějaké regulární množiny. Jinými slovy, pro každý $L \in \mathcal{L}_2$ existují $n \geq 0$, regulární množina R a morfismus h takový, že $L = h(ZAV_n \cap R)$

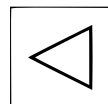
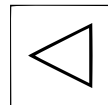
Důkaz: Viz doporučená literatura.

□

Parikhův teorém. Tento teorém opět postihuje strukturu bezkontextových jazyků – zabývá se tím, co dostaneme, pokud ve větách odhlédneme od pořadí jednotlivých symbolů a zkoumáme pouze počet jejich opakování (tj. zahrneme vlastně libovolné přeházení znaků v řetězci).



regularita



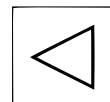
Definice 4.30 Mějme abecedu $\Sigma = \{a_1, \dots, a_k\}$. *Parikhova funkce* je funkce $\psi : \Sigma^* \rightarrow \mathbb{N}^k$ definovaná pro $w \in \Sigma^*$ jako $\psi(w) = (\#a_1(w), \dots, \#a_k(w))$, kde $\#a_i(w)$ udává počet výskytů symbolu a_i ve w .

DEF

Definice 4.31 Podmnožinu množiny vektorů \mathbb{N}^k nazveme *lineární množinou*, je-li dána bází $u_0 \in \mathbb{N}^k$ a periodami $u_1, \dots, u_m \in \mathbb{N}^k$ jako $\{u_0 + a_1 u_1 + \dots + a_m u_m \mid a_1, \dots, a_m \in \mathbb{N}\}$. Podmnožinu \mathbb{N}^k nazveme *semilineární množinou*, je-li sjednocením konečného počtu lineárních množin.

DEF

Věta 4.32 (Parikh) Pro libovolný bezkontextový jazyk L , $\psi(L)$ je semilineární množina.



Důkaz: Viz doporučená literatura. \square

Ke každé semilineární množině S můžeme najít regulární množinu $R \subseteq \Sigma^*$ takovou, že $\psi(R) = S$. Proto bývá Parikhův teorém někdy formulován takto: Komutativní obraz každého bezkontextového jazyka odpovídá nějakému regulárnímu jazyku. Semilineární množiny se navíc dají reprezentovat konečnými automaty přímo jako množiny číselných vektorů v binárním kódování (tzv. *NDDs – number decision diagrams*).

Ekvivalentním specifikačním prostředkem pro libovolný bezkontextový jazyk je nedeterministický zásobníkový automat. Existují ekvivalentní varianty těchto automatů, které se, mimo jiné, využívají jako modely různých typů syntaktických analyzátorů. Třída jazyků přijímaných deterministickými zásobníkovými automaty (tzv. deterministické bezkontextové jazyky) je vlastní podtřídou třídy bezkontextových jazyků. K uzávěrovým vlastnostem bezkontextových jazyků nepatří průnik a doplněk, není rovněž rozhodnutelný problém ekvivalence dvou bezkontextových gramatik. K rozhodnutí nepříslušnosti formálního jazyka do třídy bezkontextových jazyků se používá pumping lemma, která rovněž charakterizuje základní atribut vět nekonečného bezkontextového jazyka.



4.14 Cvičení



Cvičení 4.14.1 Sestrojte zásobníkový automat, který přijímá jazyk

$$L = \{a^n b^m \mid n \leq m \leq 2n\}$$

Cvičení 4.14.2 Pro gramatiky

(a)

$$S \rightarrow aSb \mid \epsilon$$

(b)

$$\begin{aligned} S &\rightarrow AS \mid b \\ A &\rightarrow SA \mid a \end{aligned}$$

(c)

$$\begin{aligned} S &\rightarrow SS \mid A \\ A &\rightarrow 0A1 \mid S \mid 01 \end{aligned}$$

sestrojte zásobníkové automaty modelující syntaktickou analýzu shora–dolů a zdola–nahoru.

Cvičení 4.14.3 Nalezněte gramatiku, která generuje jazyk $L(P)$, kde

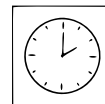
$$P = (\{q_0, q_1, q_2\}, \{a, b\}, \{Z_0, A\}, \delta, q_0, Z_0, \{q_2\});$$

zobrazení δ má tvar:

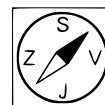
$$\begin{aligned}\delta(q_0, a, Z_0) &= (q_1, AZ_0) \\ \delta(q_0, a, A) &= (q_1, AA) \\ \delta(q_1, a, A) &= (q_0, AA) \\ \delta(q_1, \epsilon, A) &= (q_2, A) \\ \delta(q_2, b, A) &= (q_2, \epsilon)\end{aligned}$$

Kapitola 5

Turingovy stroje



15:00



Cílem kapitoly je formální vymezení Turingova stroje, stanovení pravidel a konvencí pro kompozitní vytváření Turingových strojů, pochopení mechanismu definice jazyka přijímaného a rozhodovaného Turingovým strojem a dokázání ekvivalence třídy jazyků typu 0 s třídou jazyků přijímaných Turingovými stroji. Cílem této kapitoly je také analýza vlastností příslušných jazyků a důležitá modifikace Turingova stroje specifikující kontextové jazyky.

5.1 Základní koncepce Turingových strojů

5.1.1 Churchova teze

Churchova (Church-Turingova) teze: *Turingovy stroje (a jim ekvivalentní systémy) definují svou výpočetní silou to, co intuitivně považujeme za efektivně vyčíslitelné.*

Churchova teze *není teorém*, nemůžeme formálně dokazovat, že něco odpovídá našim intuitivním představám, nicméně je podpořena řadou argumentů:

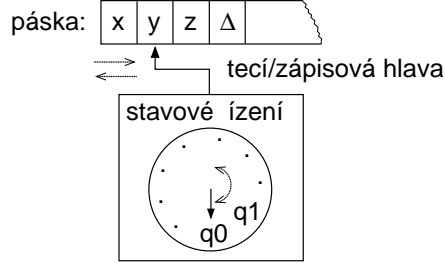
- Turingovy stroje jsou *velmi robustní* – uvidíme, že jejich různé úpravy nemění jejich výpočetní sílu (determinismus x nedeterminismus, počet pásek, ...).
- Byla navržena řada zcela *odlišných výpočetních modelů* (λ -kalkul, parciálně rekurzivní funkce, Minského stroje, ...), jejichž síla odpovídá Turingovým strojům.
- Není znám *žádný výpočetní proces*, který bychom označili za efektivně vyčíslitelný a který by nebylo možné realizovat na Turingově stroji.¹



5.1.2 Turingův stroj

Turingův stroj se skládá z konečně stavové řídicí jednotky, jednosměrně neohraňované pásky a čtecí/zapisovací hlavy. V jednom kroku výpočtu Turingův stroj nejprve přečte symbol pod hlavou. Následně v závislosti na čteném symbolu a stavu řídicí jednotky může čtený symbol přepsat, změnit stav a posunout hlavu o jedno pole doprava nebo doleva. To vše podle konečné série pravidel – přechodové relace, která je součástí řídicí jednotky. Výpočet je pak navazující sérií výpočetních kroků vycházející z počáteční konfigurace stroje, t.j. stavu řídicí jednotky, obsahu pásky a pozice hlavy.

¹Existují formalizované výpočetní procesy realizovatelné např. na TS s orákulem (nápo-vědou), rozhodujícím atomicky nějaký Turingovsky nerozhodnutelný problém (např. problém zastavení), které ale nepovažujeme za efektivní výpočetní procesy.



Definice 5.1.1 Turingův stroj (*TS*) je šestice tvaru $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, kde:

- Q je konečná množina vnitřních (řídících) stavů,
- Σ je konečná množina symbolů nazývaná vstupní abeceda, $\Delta \notin \Sigma$,
- Γ je konečná množina symbolů, $\Sigma \subset \Gamma$, $\Delta \in \Gamma$, nazývaná pásková abeceda,
- parciální funkce $\delta : (Q \setminus \{q_F\}) \times \Gamma \rightarrow Q \times (\Gamma \cup \{L, R\})$, kde $L, R \notin \Gamma$, je přechodová funkce,
- q_0 je počáteční stav, $q_0 \in Q$ a
- q_F je koncový stav, $q_F \in Q$.

DEF

5.1.3 Konfigurace Turingova stroje

Symbol Δ značí tzv. *blank* (prázdný symbol), který se vyskytuje na místech pásky, která nebyla ještě použita (může ale být na pásku zapsán i později).

Konfigurace pásky je dvojice sestávající z nekonečného řetězce reprezentujícího obsah pásky a pozice hlavy na tomto řetězci – přesněji jde o prvek množiny $\{\gamma\Delta^\omega \mid \gamma \in \Gamma^*\} \times \mathbb{N}$. *Konfiguraci pásky* zapisujeme jako $\Delta xyz \underline{z} \Delta x \Delta \Delta \dots$ (podtržení značí pozici hlavy).

Konfigurace stroje je pak dána stavem řízení a konfigurací pásky – formálně se jedná o prvek množiny $Q \times \{\gamma\Delta^\omega \mid \gamma \in \Gamma^*\} \times \mathbb{N}$.

5.1.4 Přechodová relace TS

Pro libovolný řetězec $\gamma \in \Gamma^\omega$ a číslo $n \in \mathbb{N}$ označme γ_n n -tý symbol daného řetězce a označme $s_b^n(\gamma)$ řetězec, který vznikne z γ záměnou γ_n za b .

Krok výpočtu TS M definujeme jako *nejmenší* binární relaci \vdash_M takovou, že $\forall q_1, q_2 \in Q \forall \gamma \in \Gamma^\omega \forall n \in \mathbb{N} \forall b \in \Gamma$:

- $(q_1, \gamma, n) \vdash_M (q_2, \gamma, n+1)$ pro $\delta(q_1, \gamma_n) = (q_2, R)$ – operace *posuvu doprava* při γ_n pod hlavou,
- $(q_1, \gamma, n) \vdash_M (q_2, \gamma, n-1)$ pro $\delta(q_1, \gamma_n) = (q_2, L)$ a $n > 0$ – operace *posuvu doleva* při γ_n pod hlavou a
- $(q_1, \gamma, n) \vdash_M (q_2, s_b^n(\gamma), n)$ pro $\delta(q_1, \gamma_n) = (q_2, b)$ – operace *zápisu* b při γ_n pod hlavou.

DEF

5.1.5 Výpočet TS

Výpočet TS M začínající z konfigurace K_0 je posloupnost konfigurací K_0, K_1, K_2, \dots , ve které $K_i \xrightarrow{M} K_{i+1}$ pro všechna $i \geq 0$ taková, že K_{i+1} je v dané posloupnosti, a která je buď

- *nekonečná*, a nebo
- *konečná* s koncovou konfigurací (q, γ, n) , přičemž rozlišujeme následující *typy zastavení* TS:
 1. *normální* – přechodem do koncového stavu, tj. $q = q_F$, a
 2. *abnormální*:
 - (a) pro (q, γ_n) není δ definována, nebo
 - (b) hlava je na nejlevější pozici pásky a dojde k posunu doleva, tj. $\delta(q, \gamma_n) = (q', L)$ pro nějaké $q' \in Q$.

DEF

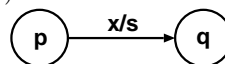
5.1.6 Poznámka – alternativní definice TS

Používají se i některé *alternativní definice TS*, u kterých se dá snadno ukázat *vzájemná převoditelnost*:

- namísto jediného q_F je povolena množina koncových stavů,
- namísto q_F je zavedena dvojice q_{accept} a q_{reject} ,
- na prvním políčku pásky je „napevno“ zapsán symbol konce pásky, z něhož není možný posun doleva,
- při zavedení obou předchozích bodů je δ obvykle definovaná jako totální funkce,
- přepis a posuv hlavy jsou spojeny do jedné operace
- apod.

5.1.7 Grafická reprezentace TS

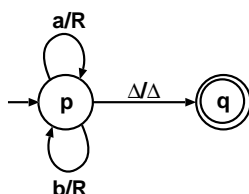
Grafická reprezentace *přechodu* (x – co se čte, s – zápis/L/R):



Grafická reprezentace *počátečního a koncového stavu*:



Příklad 5.1.1 TS, který posouvá hlavu doprava na první Δ počínaje aktuální pozicí (např. $\Delta \underline{a}ab\Delta \dots \rightarrow \Delta aab\underline{\Delta} \dots$):

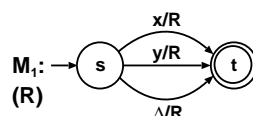
 $x + y$

5.1.8 Modulární konstrukce TS

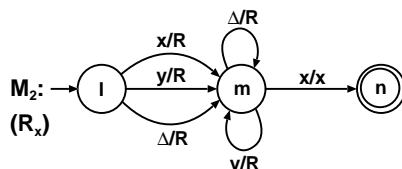
Podobně jako program je možné vytvořit kompozicí jednodušších podprogramů, procedur a funkcí, tak i Turingův stroj lze konstruovat modulárně *spojováním (kombinací)* jednodušších TS ve složitější celky.

Příklad 5.1.2 Uvažme tři následující komponenty:

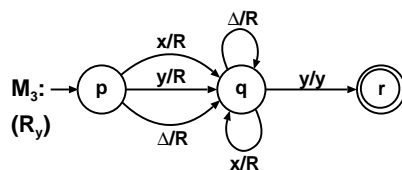
- M_1 přesune hlavu o jeden symbol doprava:



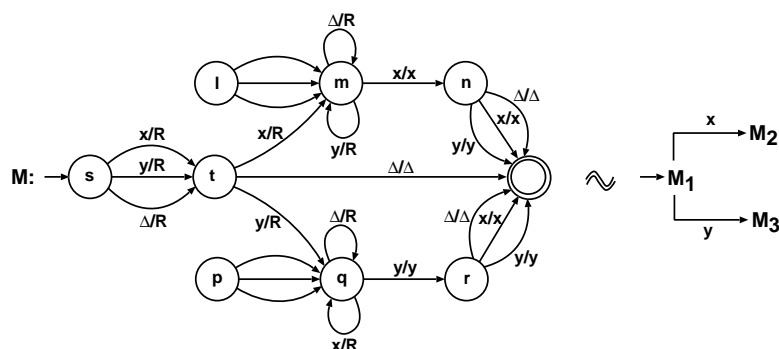
- M_2 nalezne první výskyt symbolu x vpravo (od aktuální pozice hlavy):



- M_3 nalezne první výskyt symbolu y vpravo:



Následující kombinací M_1 , M_2 a M_3 vznikne TS M , který nalezne druhý výskyt (neblankového) symbolu od počáteční pozice. V pravé části obrázku je M zapsán v podobě tzv. *kompozitního diagramu*:

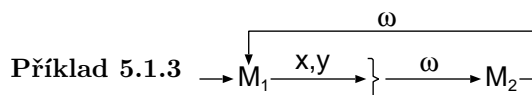


$x + y$

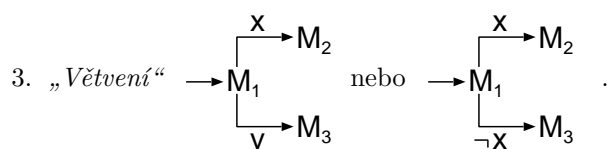
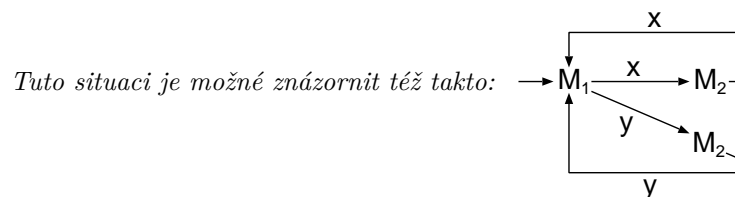
5.1.9 Kompozitní diagram TS

Konvence pro zjednodušený zápis kompozitního diagramu:

1. *Sekvenci strojů* $\rightarrow A \rightarrow B \rightarrow C$ zkracujeme na $\rightarrow ABC$.
2. „Parametrová konvence“: $\left. \begin{array}{c} x, y, z \end{array} \right\} \xrightarrow{\omega}$, kde ω nabývá hodnoty toho symbolu, který byl na pásce.



M_1 skončí s x , to dostane na vstup M_2 , a pouze pokud ten skončí s x , předá se řízení opět M_1 . Analogicky pro y .



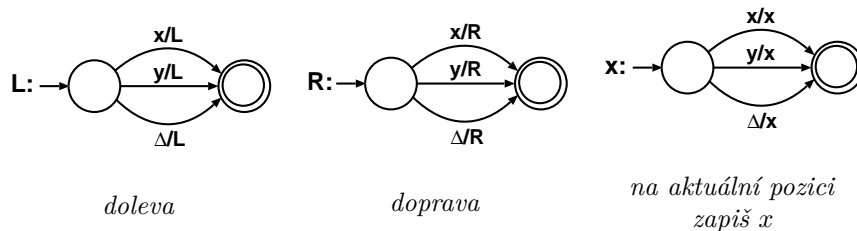
Pozor na rozdíl následujících dvou konstrukcí:

- $\rightarrow M_1 \rightarrow M_2$ – zde se přejde vždy, pokud jsou definovány příslušné hrany v M_2 (neboli doplní se hrany z q_F v M_1 pro všechny hrany z q_0 v M_2 – jedná se o skutečné ztotožnění q_F z M_1 a q_0 z M_2).
- $\rightarrow M_1 \xrightarrow{x} M_2$ – zde se doplní pouze hrana x (samozřejmě, je-li v M_2).

5.1.10 Základní stavební bloky TS

Uvažujme $\Gamma = \{x, y\}$, mezi základní stavební bloky TS obvykle patří následující stroje (lze je snadno upravit pro libovolnou jinou Γ):

1. Stroje L , R , x :

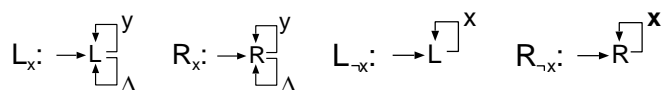


DEF

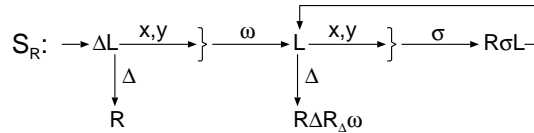
$x + y$

Příklad 5.1.4 Stroj $\rightarrow R \rightarrow y \rightarrow L$ neboli $\rightarrow RyL$ transformuje pásku $\Delta xyxy \underline{x} \Delta \Delta \dots$ na $\Delta xyxy \underline{xy} \Delta \dots$

2. Stroje L_x , R_x , $L_{\neg x}$ a $R_{\neg x}$:



3. Stroje S_R a S_L pro posuv (shift) obsahu pásky: Stroj S_R posune řetězec neblankových symbolů nacházejících se vlevo od aktuální pozice hlavy o jeden symbol doprava. Stroj S_L funguje podobně.

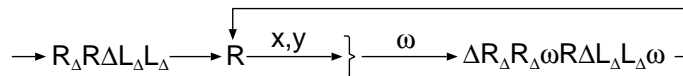


Příklad 5.1.5 Činnost strojů S_R a S_L si můžeme přiblížit na následujících příkladech:

- $\Delta xyx\Delta\Delta\Delta\ldots \xrightarrow{S_R} \Delta\Delta xyx\Delta\Delta\Delta\ldots$
- $\Delta yxy\Delta\Delta xxy\Delta\ldots \xrightarrow{S_R} \Delta\Delta yxy\Delta xxy\Delta\ldots$
- $xy\Delta yx\Delta\Delta\ldots \xrightarrow{S_R} xy\Delta\Delta x\Delta\Delta\ldots$
- $\Delta y y x x \Delta\Delta\ldots \xrightarrow{S_L} \Delta y x x \Delta\Delta\Delta\ldots$

5.1.11 Příklady TS

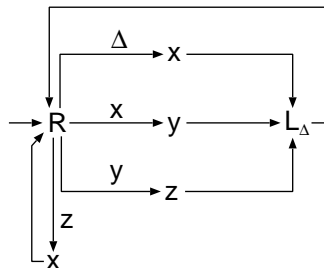
Příklad 5.1.6 Kopírovací stroj – transformuje $\Delta w \Delta$ na $\Delta w \Delta w \Delta$:



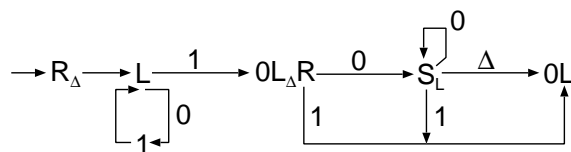
$$x + y$$

Příklad 5.1.7 Generování řetězců:

- Následující stroj generuje postupně všechny řetězce nad $\{x, y, z\}$ v uspořádání $\varepsilon, x, y, z, xx, yx, zx, xy, yy, zy, xz, yz, zz, xxx, yxx, \dots$
- Předpokládáme, že stroj začíná s konfigurací pásky $\Delta w \Delta \dots$, kde w je řetězec uведенé posloupnosti, od kterého generování započne.



Příklad 5.1.8 Stroj dekrementující kladné číslo zapsané ve dvojkové soustavě:



5.2 Turingovy stroje jako akceptory jazyků

5.2.1 Jazyk přijímaný TS

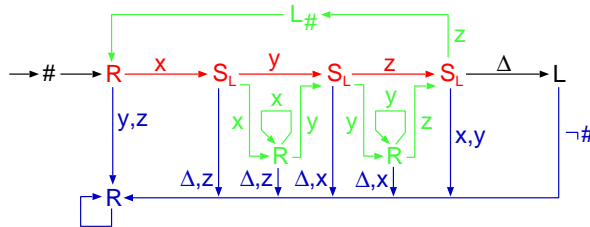
Turingův stroj normálně (přechodem do stavu q_F) zastaví jen pro některá slova množiny Σ^* zapsaná na pásce v počáteční konfiguraci. Každý stroj tedy jednoznačně vymezuje množinu slov ze Σ^* , se kterými na pásce v počáteční konfiguraci normálně zastaví – jde o *jazyk přijímaný Turingovým strojem*.

Výpočetní problémy je možno nahlížet jako jazyky. Třída jazyků, pro které existuje Turingův stroj, který je přijímá, vlastně reprezentuje třídu všech vypočítatelných problémů.

Definice 5.2.1

1. Řetězec $w \in \Sigma^*$ je přijat TS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, jestliže M při aktivaci z počáteční konfigurace pásky $\Delta w \Delta \dots$ a počátečního stavu q_0 zastaví přechodem do koncového stavu q_F , tj. $(q_0, \Delta w \Delta^\omega, 0) \vdash_M^* (q_F, \gamma, n)$ pro nějaké $\gamma \in \Gamma^*$ a $n \in \mathbb{N}$.
2. Množinu $L(M) = \{w \mid w \text{ je přijat TS } M\} \subseteq \Sigma^*$ nazýváme jazyk přijímaný TS M .

Příklad 5.2.1 Pro níže uvedený TS M platí $L(M) = \{x^n y^n z^n \mid n \geq 0\}$:



Stroj pracuje takto: $x^n y^n z^n \rightarrow x^{n-1} y^n z^n \rightarrow x^{n-1} y^{n-1} z^n \rightarrow \dots \varepsilon$.

Stroj akceptující stejný jazyk, jehož přechodová relace je definována tabulkou:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$$

$$Q = \{q_0, q_1, q_2, q_3, q_F\}$$

$$\Sigma = \{x, y, z\}$$

$$\Gamma = \{x, y, z, \times, \Delta\}$$

δ	Δ	x	y	z	\times
q_0	q_1, R				
q_1	q_F	q_2, \times			q_1, R
q_2			q_3, \times		q_2, R
q_3				q_4, \times	q_3, R
q_4	q_1, R	q_1, L	q_1, L	q_1, L	q_1, L

Stroj si postupně škrťá (symbolem \times) po jednom od každého ze symbolů x, y, z . Ve stavu q_1 hledá x , poté ve stavu q_2 y , a následně ve stavu q_3 z . Ve stavu q_4 se vrací na začátek slova, kde začne vyškrtávat další trojici. Akceptuje, jen pokud je slovo prázdné nebo pokud se mu podařilo jej po trojicích symbolů celé proškrtat.

DEF

$x + y$

5.3 Modifikace TS

Prozkoumáme několik způsobů rozšíření možností Turingova stroje. Protože je ale původní TS velmi robustní, ani zdánlivě obecnější modifikace nás nedovedou k mechanismům s větší výpočetní silou – třída jazyků akceptovaných následujícími modifikovanými Turingovými stroji zůstane stejná.

5.3.1 Přijímání zvláštních konfigurací pásky

Někdy může být vhodné, aby Turingův Stroj oznámil korektní ukončení výpočtu jinak, a to např. zapsáním informace o výsledku výpočtu na pásku.

Můžeme *přijetí řetězce TS* definovat tak, že TS začíná s konfigurací pásky $\underline{\Delta}w\Delta\dots$ a zastaví s konfigurací pásky $\underline{\Delta}Y\Delta\dots$, $Y \in \Gamma \setminus \Sigma$, (Y značí *Yes*).

Věta 5.3.1 Máme-li TS M , který přijímá $L(M)$ přechodem do q_F , můžeme vždy sestrojit stroj M'' , který bude přijímat $L(M)$ zastavením s konfigurací pásky $\underline{\Delta}Y\Delta\dots$.

Důkaz. (Idea)



- M'' začíná s páskou $\# \underline{\Delta}w * \Delta\dots$
- M' sestrojíme doplněním M s množinou stavů Q o přechody, které řeší nájezd na $*$ a $\#$:
 1. $\forall q \in Q : \delta(q, \#) = (q, L)$ – „zajištění přepadnutí“ (abnormálního ukončení), ke kterému by původně došlo.
 2. Při čtení symbolu $*$ je třeba (potenciálně) zvětšit pracovní prostor – aktivujeme podstroj $\Delta R * L$.

□

Snadno lze přejít i opačně od přijímání konfigurací pásky $\underline{\Delta}Y\Delta\dots$ k přijímání přes q_F .

Poznámka 5.3.1

- Uvědomme si, že na TS lze také nahlížet jako na výpočetní mechanismy implementující funkce $\Sigma^* \rightarrow \Gamma^*$ tím, že transformují počáteční neblankový prefix své pásky na jiný neblankový prefix při přechodu do koncového stavu.
- Vzhledem k tomu, že TS nemusí každý svůj vstup přijmout, jsou funkce jimi implementované obecně parciální.
- Blíže se budeme vyčíslováním funkcí TS zabývat dále.

5.3.2 Vícepáskové Turingovy stroje

Přímočarou cestou k zesílení Turingova stroje je zapojení ne jedné, ale několika pásek. Vícepáskový TS disponuje k jednosměrně nekonečnými páskami, z nichž každá je vybavena vlastní čtecí/zapisovací hlavou a je připojena ke společné konečné stavové jednotce. V jednom kroku stroj nejdříve na každé pásce zvlášť přečte symbol, posune hlavu nebo přepíše symbol a poté v závislosti na čtených symbolech změní stav řídicí jednotky.

Formálněji – uvažujme TS, který má k pásek s páskovými abecedami $\Gamma_1, \Gamma_2, \dots, \Gamma_k$ a k odpovídajících hlav s přechodovou funkcí tvaru

$$\delta : (Q \setminus \{q_F\}) \times \Gamma_1 \times \Gamma_2 \times \dots \times \Gamma_k \longrightarrow Q \times \bigcup_{i \in \{1, \dots, k\}} \{i\} \times (\Gamma_i \cup \{L, R\}),$$

kde i značí pásku, na kterou se zapisuje (na které se posouvá hlava).

Příklad 5.3.1 Sestrojíme dvoupáskový stroj, který bude akceptovat již známý jazyk $x^n y^n z^n$. Výhoda druhé pásky se zde projeví efektivitou stroje. Budeme předpokládat, že v počáteční konfiguraci mají pásky tvary:

– první páska: $\underline{\Delta} w \Delta \Delta \dots$

– druhá páska: $\underline{\Delta} \Delta \dots$

Stroj bude vypadat takto:

$$M = \{Q, \Sigma, \Gamma_1, \Gamma_2, \delta, q_0, q_F\} \text{ kde:}$$

$$Q = \{q_0, q_1, q_2, q_3, q_F\}$$

$$\Sigma = \{x, y, z\}$$

$$\Gamma_1 = \{x, y, z, \Delta\}$$

$$\Gamma_2 = \{X, Y, \Delta\}$$

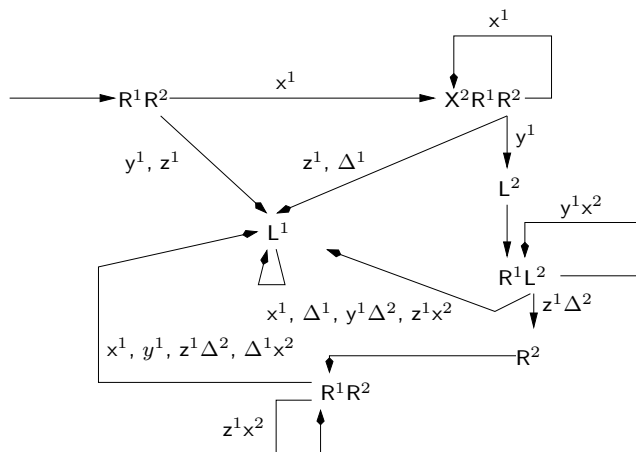
δ	Δ, Δ	x, Δ	x, X	y, Δ	y, X	y, Y	z, Δ	z, Y
q_0	q_1, R, R							
q_1	q_F	q_1, x, X	q_1, R, R	q_2, y, L				
q_2					q_2, y, Y	q_2, R, L	q_3, z, R	
q_3	q_F							q_3, R, R

Stroj slovo na první (vstupní) pásce přečte jen jednou zleva doprava. Přitom na druhé pásce kontroluje jeho tvar takto: Nejdříve zápisem symbolu X na druhou pásku zaznamenává na vstupu přečtená x . Poté za každé y přečtené na vstupu přepíše jeden symbol X na druhou pásku symbolem Y (skončí na začátku druhé pásky). Nakonec za každé přečtené z na vstupu posune hlavu druhé pásky doprava. Slovo akceptuje, pokud hlavu druhé pásky takto posune přesně za všechna Y .

DEF

 $x + y$

Stroj podobným způsobem akceptující stejný jazyk, zapsaný diagramem:



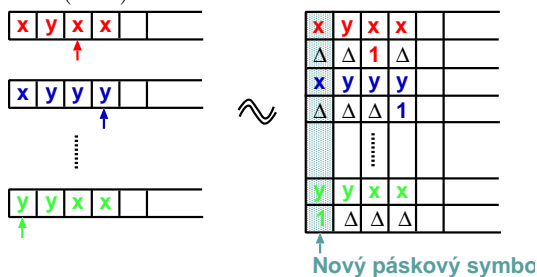
Akce první a druhé hlavy jsou zde rozlišeny horním indexem.

Příklad 5.3.2 Navrhněte vícepáskový TS pro převod binárního zápisu čísla na unární.



Věta 5.3.2 Pro každý k -páskový TS M existuje jednopáskový TS M' takový, že $L(M) = L(M')$.

Důkaz. (idea)



Důkaz provedeme tak, že ukážeme algoritmus převodu na ekvivalentní jednopáskový TS:

- Předpokládáme, že přijímaný řetězec je u k -páskového stroje na počátku zapsán na první pásce, všechny ostatní pásy jsou prázdné a všechny hlavy jsou na nejlevější pozici.
- Původních k pásek simulujeme rozšířením páskové abecedy o $2k$ -tice, v nichž vždy i -tá složka pro liché i reprezentuje obsah $(\frac{i+1}{2})$ -ní pásky a na pozici $i+1$ je Δ nebo 1 podle toho, zda se na ní nachází příslušná hlava či nikoliv.
- Počet načítaných kombinací symbolů v původním automatu je konečný a tudíž si výše uvedené rozšíření můžeme skutečně dovolit.
- Při simulaci k -páskového TS pak nejprve převedeme původní obsah první pásky na ekvivalentní obsah zakódovaný v $2k$ -ticích a pak každý krok simulujeme několika kroky.

- Při simulaci využíváme *stavy ve formě $(k+1)$ -tic*, kde první složka je stav původního TS a ostatní složky jsou aktuálně čtené symboly.
- Při rozhodování o dalším kroku pak bereme na zřetel především tento stav, aktuální načítaný symbol není důležitý a vždy se při čtení přemístíme na speciální pozici nalevo od užitečného obsahu pásky.
- Po rozhodnutí o dalším kroku „*najedeme*“ *doprava* na pozici, na které je simulovaná hlava pásky, která se má modifikovat, provedeme příslušnou změnu a vrátíme se zpět doleva.
- Za nový aktuální stav považujeme $(k+1)$ -tici danou novým stavem simulovaného TS a k -tici převzatou z původního stavu nového TS modifikovanou na místě modifikované pásky.
- Navíc je nutné korektně *simulovat „přepadnutí“ hlavy na kterékoliv pásce a převod dosud nevyužitých míst pásky s Δ na odpovídající $2k$ -tici blank symbolů.*
- Při řádné formalizaci popsaného algoritmu pak není těžké ukázat, že výsledný TS skutečně simuluje původní TS.

□

Závěr:

Zvětšení paměťových možností TS nerozšiřuje jejich schopnosti přijímat jazyky!

Příklad 5.3.3 Využijte možností *vícépáskového stroje k návrhu TS akceptujícího jazyk $x^n y^n z^n$.*



5.3.3 Nedeterministické Turingovy stroje

Definovaný deterministický Turingův stroj se o příštím kroku rozhodoval vždy jednoznačně na základě čteného symbolu a stavu. Nedeterministický TS si naopak proti tomu v dané konfiguraci vybírá z konečného počtu možností. Pro jednu vstupní konfiguraci tedy existuje více možných výpočtů a slovo je nedeterministickým strojem akceptováno, právě když alespoň jeden z možných výpočtů na něm je akceptující.²



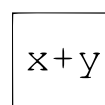
Definice 5.3.1 Nedeterministický TS je šestice $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, kde význam jednotlivých složek je shodný s deterministickým TS až na δ , jež má tvar:

$$\delta : (Q \setminus \{q_F\}) \times \Gamma \longrightarrow 2^{Q \times (\Gamma \cup \{L, R\})}$$

Definice 5.3.2 Jazyk $L(M)$ NTS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je množina řetězců $w \in \Sigma^*$ takových, že M při aktivaci z q_0 při počátečním obsahu pásky $\Delta w \Delta \dots$ může zastavit přechodem do q_F .



²jakoby stroj „hádal“, který z možných kroků (jestli nějaký) vede do koncového stavu



Příklad 5.3.4 Sestrojíme dvoupáskový nedeterministický stroj, který bude akceptovat jazyk $L = \{ww^R \mid w \in \{x, y\}^*\}$. Nedeterminismus zde využijeme k „uhádnutí“ poloviny vstupního slova. To znamená, že stroj nejprve čte slovo na první pásce a přečtené symboly zapisuje na druhou pásku. V určitém okamžiku stroj nedeterministicky přejde do stavu, kdy začne porovnávat obsah druhé pásky s dosud nepřčteným vstupem na první pásce (na druhé pásce se pohybuje zprava doleva). Budeme předpokládat, že v počáteční konfiguraci mají pásy tvary:

– první páska: $\Delta w \Delta \Delta \dots$

– druhá páska: $\Delta \Delta \dots$

Jelikož stroj je nedeterministický, jeho přechodová funkce δ vrací množinu stavů. To znamená, že v jednotlivých políčkách tabulky pro přechodovou funkci může být víc než jedna položka. Stroj bude vypadat takto:

$$M = \{Q, \Sigma, \Gamma_1, \Gamma_2, \delta, q_0, q_F\} \text{ kde:}$$

$$Q = \{q_0, q_1, q_2, q_F\}$$

$$\Sigma = \{x, y\}$$

$$\Gamma_1 = \{x, y, \Delta\}$$

$$\Gamma_2 = \{x, y, \Delta\}$$

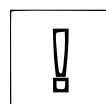
δ	Δ, Δ	x, Δ	x, x	y, Δ	y, y	x, y	y, x	Δ, x	Δ, y
q_0	q_1, R, R								
q_1	q_0, R, R	q_1, x, x	q_1, R, R q_2, R, x	q_1, y, y	q_1, R, R q_2, R, y				
q_2	q_F	q_0, x, Δ	q_2, R, L	q_0, y, Δ	q_2, R, L	q_0, x, y	q_0, y, x	q_0, Δ, x	q_0, Δ, y

Příklad 5.3.5 Navrhněte vícepáskový nedeterministický TS akceptující jazyk $\{a^p \mid \text{kde } p \text{ není prvočíslo}\}$.



Věta 5.3.3 Pro každý NTS M existuje DTS M' takový, že $L(M) = L(M')$.

Důkaz. (idea)



- NTS M budeme simulovat třípáskovým DTS. Význam jednotlivých pásek tohoto stroje je následující:
 - Páska 1 obsahuje přijímaný vstupní řetězec.
 - Páska 2 je pracovní páska. Obsahuje kopii pásky 1 ohraničenou vhodnými speciálními značkami. Po neúspěšném pokusu o přijetí je její obsah smazán a obnoven z první pásky.
 - Páska 3 obsahuje kódovanou volbu posloupností přechodů; při neúspěchu bude její obsah nahrazen jinou posloupností.
- Zvolená posloupnost přechodů je kódována posloupností čísel přiřazených přechodům simulovaného stroje.
- Jednotlivé posloupnosti přechodů na pásce 3 je možno generovat modifikovaným TS pro generování posloupností řetězců $\varepsilon, x, y, z, xx, \dots$
- Vlastní simulace probíhá takto:
 1. Okopíruj obsah pásky 1 na pásku 2.

2. Generuj příští posloupnost přechodů na pásce 3.
 3. Simuluj provedení posloupnosti z pásky 3 na obsahu pásky 2.
 4. Vede-li zkoumaná posloupnost do q_F simulovaného stroje, zastav – vstupní řetězec je přijat. V opačném případě smaž pásku 2 a vrať se k bodu 1.
- Není obtížné nahlédnout, že jazyk přijímaný takto vytvořeným strojem odpovídá jazyku přijímaným původním NTS.

□

Závěr:

Zavedením nedeterminismu do TS se nezvyšují jejich schopnosti přijímat jazyky!

5.4 Jazyky rekurzivně vyčíslitelné a jazyky rekurzivní

Zavedeme klasifikaci jazyků (problémů) vzhledem k jejich akceptovatelnosti Turingovými stroji (řešitelnosti). Dojdeme k základním třem třídám jazyků, které reprezentují problémy řešitelné, problémy řešitelné jen částečně a problémy algoritmicky neřešitelné.

5.4.1 Rekurzivní vyčíslitelnost a rekurzivnost

Turingův stroj se nazývá *úplný* (*total*), právě když pro každý vstup zastaví.

Definice 5.4.1 Jazyk $L \subseteq \Sigma^*$ se nazývá

- rekurzivně vyčíslitelný, *jestliže* $L = L(M)$ pro nějaký TS M ,
- rekurzivní, *jestliže* $L = L(M)$ pro nějaký úplný TS M .

Je-li M úplný Turingův stroj, pak říkáme, že M *rozhoduje jazyk* $L(M)$.

Ke každému rekurzivnímu jazyku existuje TS, který ho rozhoduje, tj. *zastaví pro každé vstupní slovo* – tento TS lze samozřejmě upravit tak, aby pro každý řetězec z daného jazyka zastavil s páskou $\Delta Y \Delta \Delta \dots$, a jinak zastavil s páskou $\Delta N \Delta \Delta \dots$.

TS přijímající rekurzivně vyčíslitelný jazyk L zastaví pro každé $w \in L$, ovšem pro $w \notin L$ může zastavit, ale také *může donekonečna cyklit*.

DEF

5.4.2 Rozhodovací problémy

Rozhodovací problém (*decision problem*) P může být chápán jako funkce f_P s oborem hodnot $\{true, false\}$ ³.

Rozhodovací problém je obvykle specifikován:

- definičním oborem A_P reprezentujícím množinu možných instancí problému (možných vstupů) a

DEF

³otázka s možnou odpovědí ano/ne

- podmnožinou $B_P \subseteq A_P$, $B_P = \{p \mid f_P(p) = \text{true}\}$ instancí, pro které je hodnota f_P rovna true .

V teorii formálních jazyků používáme ke kódování jednotlivých instancí problémů řetězce nad vhodnou abecedou Σ . Pak je rozhodovací problém P přirozeně specifikován jazykem $L_P = \{w \in \Sigma^* \mid w = \text{code}(p), p \in B_P\}$, kde $\text{code} : A_P \rightarrow \Sigma^*$ je injektivní funkce, která přiřazuje instancím problému příslušný řetězec (nezávisle na f_P).

$x + y$

Příklad 5.4.1 Příklady rozhodovacích problémů:

- P_1 – orientovaný graf je silně souvislý.
- P_2 – dvě bezkontextové gramatiky jsou ekvivalentní.
- P_3 – n je prvočíslo.

Poznámka: Dále budeme o rozhodovacích problémech hovořit jednoduše jako o problémech.

5.4.3 Rozhodování problémů TS

Definice 5.4.2 Necht P je problém specifikovaný jazykem L_P nad Σ . Problém P nazveme:

DEF

- rozhodnutelný, pokud L_P je rekurzivní jazyk, tj. existuje TS, který L_P rozhoduje (přijme každý řetězec $w \in L_P$ a zamítne každý řetězec $w \in \Sigma^* \setminus L_P$),
- nerozhodnutelný, když není rozhodnutelný, a
- částečně rozhodnutelný, jestliže L_P je rekurzivně vyčíslitelný jazyk, tj. existuje TS, který přijme každý řetězec $w \in L_P$ a každý řetězec $w \in \Sigma^* \setminus L_P$ zamítne, nebo je jeho výpočet na něm nekonečný.

Poznámka: Z definice 8.2 plyne, že každý rozhodnutelný problém je současně částečně rozhodnutelný. Některé nerozhodnutelné problémy nejsou ale ani částečně rozhodnutelné.

5.5 TS a jazyky typu 0

Odvodíme důležitou korespondenci mezi gramatikami typu 0 a Turingovými stroji. Jazyky generované gramatikami typu 0 jsou totiž právě jazyky akceptované Turingovými stroji (rekurzivně vyčíslitelné). Ukážeme, jak pro každý TS sestavit gramatiku typu 0, která generuje jazyk jím akceptovaný, a naopak jak pro každou gramatiku typu 0 sestavit TS akceptující jazyk jí generovaný.

5.5.1 Jazyky přijímané TS jsou typu 0

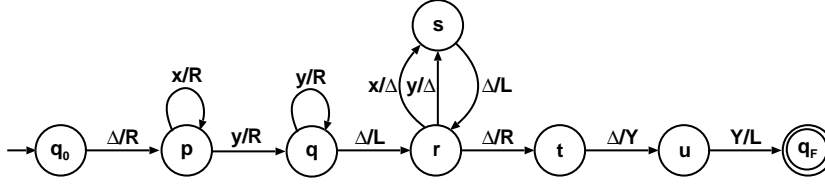
Konfigurace TS je dána (1) stavem řízení, (2) obsahem pásky a (3) pozicí hlavy.

Pro zápis konfigurace TS v řídicím stavu q a s konfigurací pásky $\Delta x \underline{y} z \Delta \dots$ zavedeme konvenci $[\Delta x q y z \Delta \dots]$.

DEF

Příklad 5.5.1 TS přijímající jazyk $\{x^m y^n \mid m \geq 0, n > 0\}$ konfigurací pásky $\underline{\Delta} Y \Delta$ při počáteční konfiguraci pásky $\underline{\Delta} w \Delta$:

$x + y$



Posloupnost konfigurací při příjetí xy :

- | | | |
|------------------------------------|--|--|
| 1. $[q_0 \Delta xxy \Delta \dots]$ | 6. $[\Delta xxy \Delta \dots]$ | 11. $[\Delta s \Delta \Delta \Delta \Delta \dots]$ |
| 2. $[\Delta p xxy \Delta \dots]$ | 7. $[\Delta xxs \Delta \Delta \dots]$ | 12. $[r \Delta \Delta \Delta \Delta \Delta \dots]$ |
| 3. $[\Delta p xxy \Delta \dots]$ | 8. $[\Delta xrx \Delta \Delta \dots]$ | 13. $[\Delta t \Delta \Delta \Delta \Delta \dots]$ |
| 4. $[\Delta xpxy \Delta \dots]$ | 9. $[\Delta xs \Delta \Delta \Delta \dots]$ | 14. $[\Delta uY \Delta \Delta \Delta \dots]$ |
| 5. $[\Delta xxyq \Delta \dots]$ | 10. $[\Delta rx \Delta \Delta \Delta \dots]$ | 15. $[q_F \Delta Y \Delta \Delta \Delta \dots]$ |

Věta 5.5.1 Každý jazyk přijímaný TS (tj. každý rekurzivně vyčíslitelný jazyk) je jazykem typu 0.

Důkaz. Necht $L = L(M)$ pro nějaký TS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$. Sestrojíme gramatiku $G = (N, \Sigma, P, S)$ typu 0 takovou, že $L(G) = L(M)$. Gramatika G bude umožňovat vytvářet právě derivace odpovídající inverzi posloupnosti konfigurací TS M při přijetí libovolného řetězce $w \in L(M)$:

1. $N = \{S\} \cup Q \cup (\Gamma \setminus \Sigma) \cup \{[,]\}$ (předpokládáme, že všechny zde sjednocené množiny jsou po dvou disjunktní).
2. P je nejmenší množina obsahující následující pravidla:
 - (a) $S \rightarrow [q_f \Delta Y \Delta]$,
 - (b) $\Delta] \rightarrow \Delta \Delta$ – doplnění Δ ,
 - (c) $qy \rightarrow px$, jestliže $\delta(p, x) = (q, y)$,
 - (d) $xq \rightarrow px$, jestliže $\delta(p, x) = (q, R)$,
 - (e) $qyx \rightarrow ypx$ pro každé $y \in \Gamma$, jestliže $\delta(p, x) = (q, L)$,
 - (f) $[q_0 \Delta \rightarrow \varepsilon, \Delta \Delta] \rightarrow \Delta], \Delta] \rightarrow \varepsilon$ – zajištění $[q_0 \Delta w \Delta \dots \Delta] \xrightarrow{+}_G w$.

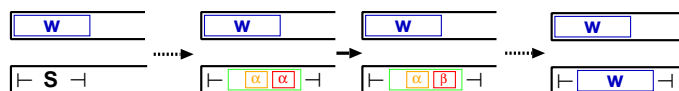
Snadno se nyní nahlédne, že $w \in L(M)$ právě tehdy, když existuje derivace $S \Rightarrow_G [q_F \Delta Y \Delta] \Rightarrow_G \dots \Rightarrow_G [q_0 \Delta w \Delta \dots] \Rightarrow_G \dots \Rightarrow_G w$, a že $L(G) = L(M)$. □

5.5.2 Jazyky typu 0 jsou přijímány TS

Věta 5.5.2 Každý jazyk typu 0 je přijímán nějakým TS (tj. je rekurzivně vyčíslitelný).

Důkaz. Necht $L = L(G)$ pro $G = (N, \Sigma, P, S)$ je jazykem typu 0. Sestrojíme nedeterministický dvoupáskový TS M takový, že $L(G) = L(M)$:

- 1. páska obsahuje přijímaný vstupní řetězec w .
- Na 2. pásce se M pokouší pomocí simulace použití přepisovacích pravidel $(\alpha \rightarrow \beta) \in P$ vytvořit derivaci w :



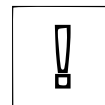
1. Stroj nejprve umístí na 2. pásku symbol S .
2. Stroj opakovaně simuluje na 2. pásce provádění pravidel $(\alpha \rightarrow \beta) \in P$. Nedeterministicky zvolí pravidlo a také výskyt α na pásce. Při přepisu α na β , $|\alpha| \neq |\beta|$, může využít posuv části užitečného obsahu pásky vlevo či vpravo.
3. Stroj srovná finální obsah 2. pásky s 1. páskou. Shodují-li se, zastaví přechodem do q_F . Jinak posouvá hlavu doleva až do abnormálního zastavení.

Snadno se nyní nahlédne, že skutečně $L(G) = L(M)$. Navíc lze M , podobně jako u vícepáskových DTS, převést na jednopáskový NTS a ten dále na jednopáskový DTS. \square

5.5.3 Jazyky typu 0 = jazyky přijímané TS

Věta 5.5.3 Třída jazyků přijímaných TS (neboli jazyků rekurzivně vyčíslitelných) je shodná se třídou jazyků typu 0.

Důkaz. Důsledek dvou předchozích vět. \square

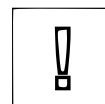


5.6 Vlastnosti jazyků rekurzivních a rekurzivně vyčíslitelných

„Příjemnou“ vlastností rekurzivních a rekurzivně spočetných jazyků je jejich uzavřenost vůči operacím průniku, sjednocení a zřetězení. Třída rekurzivních jazyků je navíc uzavřena i vůči doplňku.

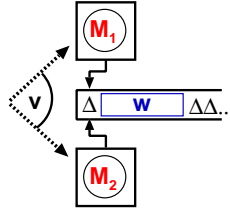
5.6.1 Uzavřenost vůči \cup , \cap , \cdot a $*$

Věta 5.6.1 Třídy rekurzivních a rekurzivně vyčíslitelných jazyků jsou uzavřeny vůči operacím \cup , \cap , \cdot a $*$.

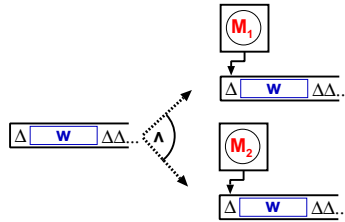


Důkaz. Necht L_1, L_2 jsou jazyky přijímané TS M_1, M_2 . Zřejmě můžeme předpokládat, že množiny stavů TS M_1, M_2 jsou disjunktní.

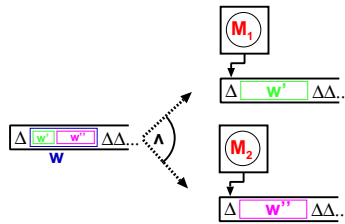
- NTS $M_{L_1 \cup L_2}$, $L(M_{L_1 \cup L_2}) = L_1 \cup L_2$, sestrojíme tak, že sjednotíme po složkách stroje M_1 a M_2 , zavedeme nový počáteční stav, z něj nedeterministické přechody přes Δ/Δ do obou původních počátečních stavů a sloučíme původní koncové stavy do jediného nového koncového stavu.



- Třípáskový TS $M_{L_1 \cap L_2}$, $L(M_{L_1 \cap L_2}) = L_1 \cap L_2$, okopíruje vstup z první pásky na druhou, na ní simuluje stroj M_1 , pokud ten přijme, okopíruje vstup z první pásky na třetí, na ní simuluje stroj M_2 , pokud i ten přijme, přijme i stroj $M_{L_1 \cap L_2}$.



- Třípáskový NTS $M_{L_1.L_2}$, $L(M_{L_1.L_2}) = L_1.L_2$, okopíruje nedeterministicky zvolený prefix vstupu z první pásky na druhou, na ní simuluje stroj M_1 , pokud ten přijme, okopíruje zbytek vstupu z první pásky na třetí, na ní simuluje stroj M_2 , pokud i ten přijme, přijme vstup i stroj $M_{L_1.L_2}$.



- Dvoupáskový NTS $M_{L_1^*}$, $L(M_{L_1^*}) = L_1^*$, je zobecněním předchozího stroje: po částech kopíruje vstup z první pásky na druhou a na ní simuluje opakovaně stroj M_1 . Obsah druhé pásky má ohraničený speciálními značkami a po každé simulaci stroje M_1 ho smaže. Umožňuje samozřejmě posuv pravé značky dále doprava při nedostatku místa.

Jsou-li stroje M_1 a M_2 úplné, je možné vybudovat stroje podle výše uvedených pravidel také jako *úplné* (u $M_{L_1 \cup L_2}$, $M_{L_1 \cap L_2}$, $M_{L_1.L_2}$ je to okamžité, u $M_{L_1^*}$ nepřípustíme načítání prázdného podřetězce vstupu z 1. na 2. pásku – pouze umožníme jednorázově přijmout prázdný vstup). To dokazuje uzavřenost vůči uvedeným operacím také u *rekurzivních jazyků*.

□

5.6.2 (Ne)uzavřenost vůči komplementu



Věta 5.6.2 Třída rekurzivních jazyků je uzavřena vůči komplementu.

Důkaz. TS M přijímající rekurzivní jazyk L vždy zastaví. Snadno upravíme M na M' , který při nepřijetí řetězce vždy přejde do unikátního stavu q_{reject} . TS \overline{M} , $L(\overline{M}) = \overline{L}$, snadno dostaneme z M' záměnou q_F a q_{reject} . \square

Třída rekurzivně vyčíslitelných jazyků není uzavřena vůči komplementu!

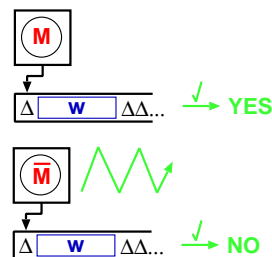
- Výše uvedené konstrukce nelze užít – cyklení zůstane zachováno.
- Důkaz neuzavřenosti bude uveden v dalších přednáškách.

Věta 5.6.3 Jsou-li jazyky L i \overline{L} rekurzivně vyčíslitelné, pak jsou oba rekurzivní.

Důkaz.

Mějme M , $L(M) = L$, a \overline{M} , $L(\overline{M}) = \overline{L}$. Úplný TS přijímající L sestojíme takto:

- Použijeme dvě pásky. Na jedné budeme simulovat M , na druhé \overline{M} . Simulace se bude provádět proloženě krok po kroku: krok M , krok \overline{M} , krok M , ...
- Přijmeme, právě když by přijal M , zamítneme abnormálním zastavením, právě když by přijal \overline{M} . Jedna z těchto situací určitě nastane v konečném počtu kroků.



Existence úplného TS pro \overline{L} plyne z uzavřenosti rekurzivních jazyků vůči komplementu. \square

Důsledkem výše uvedených vět je mj. to, že pro L a \overline{L} musí vždy nastat jedna z následujících situací:

- L i \overline{L} jsou rekurzivní,
- L ani \overline{L} nejsou rekurzivně vyčíslitelné,
- jeden z těchto jazyků je rekurzivně vyčíslitelný, ale ne rekurzivní, druhý není rekurzivně vyčíslitelný.

5.7 Lineárně omezené automaty

Poslední třídou Chomského hierarchie, ke které jsme zatím nepoznali protějšek ve formě výpočetního stroje⁴, jsou gramatiky typu 1 – kontextové gramatiky. Nyní se seznámíme s mírně omezenou formou Turingových strojů, která tuto mezeru vyplňuje.

⁴regulárním gramatikám odpovídají konečné automaty, bezkontextovým gramatikám zásobníkové automaty a gramatikám typu 0 Turingovy stroje

5.7.1 Lineárně omezené automaty

Lineárně omezený automat (LOA) je *nedeterministický* TS, který nikdy neopustí tu část pásky, na níž je zapsán jeho vstup.

Formálně můžeme LOA definovat jako NTS, jehož pásková abeceda obsahuje speciální symbol, kterým unikátně označujeme pravý konec vstupu na pásce, přičemž tento symbol není možné přepsat, ani z něj provést posun doprava.

Deterministický LOA můžeme přirozeně definovat jako (deterministický) TS, který nikdy neopustí část pásky se zapsaným vstupem.

Není známo, zda deterministický LOA je či není striktně slabší než LOA.

DEF

5.7.2 LOA a kontextové jazyky

Věta 5.7.1 Třída jazyků, kterou lze generovat kontextovými gramatikami, odpovídá třídě jazyků, které lze přijímat LOA.

Důkaz.

- Uvážíme definici bezkontextových gramatik jako gramatik s pravidly v podobě $\alpha \rightarrow \beta$, kde $|\alpha| \leq |\beta|$, nebo $S \rightarrow \varepsilon$.
- $LOA \longrightarrow G1$:
 - Použijeme podobnou konstrukci jako u $TS \longrightarrow G0$.
 - Na počátku vygenerujeme příslušný pracovní prostor, který se pak již nebude měnit: odpadá nekontextové pravidlo $\Delta\Delta] \rightarrow \Delta]$.
 - Užití nekontextových pravidel $[q_0\Delta \rightarrow \varepsilon$ a $\Delta] \rightarrow \varepsilon$ obejdeme (1) zavedením zvláštních koncových neterminálů integrujících původní informaci a příznak, že se jedná o první/poslední symbol, a (2) integrací symbolu reprezentujícího řídicí stav a pozici hlavy s následujícím páskovým symbolem.
- $G1 \longrightarrow LOA$:
 - Použijeme podobnou konstrukci jako u $G0 \longrightarrow TS$ s tím, že nepovolíme, aby rozsah druhé pásky někdy překročil rozsah první pásky.

□

5.7.3 Kontextové a rekurzivní jazyky

Věta 5.7.2 Každý kontextový jazyk je rekurzivní.

Důkaz.

- Počet konfigurací, které se mohou objevit při přijímání w příslušným LOA M je vzhledem k nemožnosti zvětšovat pracovní prostor pásky konečný: lze shora ohraničit funkcí c^n pro vhodnou konstantu c – exponenciála plyne z nutnosti uvažovat výskyt všech možných symbolů na všech místech pásky.
- Pro zápis libovolného čísla z intervalu $0, \dots, c^n - 1$ nikdy nebude třeba více než n symbolů, užijeme-li c -ární soustavu.

- Můžeme zkonstruovat úplný LOA ekvivalentní s M , který bude mít každý symbol na pásce strukturovaný jako dvojici:
 - S využitím 1. složek těchto dvojic simulujeme M .
 - V 2. složkách počítáme počet kroků; dojde-li k přetečení, odmítneme vstup.

□

Věta 5.7.3 *Ne každý rekurzivní jazyk je kontextový.*

Důkaz. (Idea) Lze užít techniku diagonalizace prezentovanou dále.

□

5.7.4 Vlastnosti kontextových jazyků

Věta 5.7.4 Třída kontextových jazyků je uzavřena vůči operacím \cup , \cap , $.$, $*$ a komplementu.

Důkaz.

- Uzavřenost vůči \cup , \cap , $.$ a $*$ lze ukázat stejně jako u rekurzivně spočetných jazyků.
- Důkaz uzavřenosti vůči komplementu je značně komplikovaný (všimněme si, že LOA je *nedeterministický* a nelze tudíž užít konstrukce použité u rekurzivních jazyků) – zájemci naleznou důkaz v doporučené literatuře.

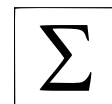
□

Poznamenejme, že již víme, že u kontextových jazyků

- lze rozhodovat členství věty do jazyka (rekurzivnost) a
- nelze rozhodovat inkluzi jazyků (neplatí ani pro bezkontextové jazyky).

Dále lze ukázat, že pro kontextové jazyky *nelze rozhodovat prázdnotu jazyka* (užije se redukce z Postova problému přiřazení – viz další kapitoly).

Turingův stroj je ekvivalentním specifikačním prostředkem pro libovolný jazyk typu 0 a v souladu s Churchovou tezí nejmocnějším formálním prostředkem (spolu s řadou dalších) pro specifikaci formálních jazyků. Představuje robustní matematický nástroj popisu jazyků a algoritmů, jehož modelovací síla není ovlivněna ani nedeterminismem či determinismem stroje, ani jeho paměťovými prostředky (více páskami). Významná podtřída Turingových strojů, lineárně omezené automaty, vymezují třídu kontextových jazyků.



5.8 Cvičení

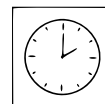
Příklad 5.8.1

1. Vysvětlete význam Churchovy teze.
2. Definujte pojmy Turingův stroj a jazyků přijímaných Turingovými stroji.
3. Definujte pojmy vícepáskový a nedeterministický Turingův stroj a rozhodněte, zda tato rozšíření zvyšují schopnost přijímat jazyky.
4. Vysvětlete rozdíl mezi úplnými a neúplnými Turingovy stroji a uveďte třídy jazyků, které jsou schopny přijímat.
5. Jaký je vztah mezi třídou jazyků přijímanou Turingovými stroji a jazyky typu 0.
6. Rozhodněte, pro které jazykové operace jsou uzavřeny třídy rekurzivních a rekurzivně vyčíslitelných jazyků.
7. Definujte lineárně omezené automaty a rozhodněte, jaký je vztah mezi třídou jazyků přijímaných lineárně omezenými automaty a třídou jazyků typu 1.



Kapitola 6

Meze rozhodnutelnosti



15:00

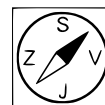
Podle Churchovy teze třídu algoritmicky řešitelných – vyčíslitelných problémů vymezují Turingovy stroje. Zavedli jsme pomocí nich základní rozdělení problémů na rozhodnutelné, částečně rozhodnutelné a nerozhodnutelné. Nyní budeme chtít blíže prozkoumat vlastnosti jednotlivých tříd a problémů v nich ležících a najít odpověď na otázky jako:

- Existují jazyky (problémy), jež nejsou rekurzivně vyčíslitelné (částečně rozhodnutelné)?

- Které jazyky (problémy) nejsou rekurzivní (rozhodnutelné)?

- Jak poznat, ve které ze tříd leží konkrétní jazyk (problém)?

Bude také zaveden prostředek specifikace jazyků (algoritmů) ekvivalentní Turingovým strojům – parciálně rekurzivní funkce.



6.1 Jazyky mimo třídu 0

Jazyky mimo třídu 0 (algoritmicky neřešitelné problémy) nejen existují, ale dokonce se dá říci, že částečná rozhodnutelnost (algoritmická řešitelnost) je mezi jazyky (problémy) výjimkou.



6.1.1 Existence jazyků mimo třídu 0

Věta 6.1.1 Pro každou abecedu Σ existuje jazyk nad Σ , který není typu 0.

Důkaz.

1. Libovolný jazyk typu 0 nad Σ může být přijat TS s $\Gamma = \Sigma \cup \{\Delta\}$: Pokud M používá více symbolů, můžeme je zakódovat jako jisté posloupnosti symbolů ze $\Sigma \cup \{\Delta\}$ a sestrojít TS M' , který simuluje M .
2. Nyní můžeme snadno systematicky vypisovat všechny TS s $\Gamma = \Sigma \cup \{\Delta\}$. Začneme stroji se dvěma stavy, pak se třemi stavy, atd.
Závěr: Množina všech takových strojů a tedy i jazyků typu 0 je spočetná.
3. Množina Σ^* ale obsahuje nekonečně mnoho řetězců a proto je množina 2^{Σ^*} , zahrnující všechny jazyky, nespočetná – důkaz viz následující lemma.
4. Z rozdílnosti mohutností spočetných a nespočetných množin plyne platnost uvedené věty.

□

Lemma 6.1.1 Pro neprázdnou a konečnou množinu Σ je množina 2^{Σ^*} nespočetná.

Důkaz. Důkaz provedeme tzv. *diagonalizací* (poprvé použitou Cantorem při důkazu rozdílné mohutnosti \mathbb{N} a \mathbb{R}).

- Předpokládejme, že 2^{Σ^*} je spočetná. Pak dle definice spočetnosti existuje bijekce $f : \mathbb{N} \longleftrightarrow 2^{\Sigma^*}$.
- Uspořádejme Σ^* do nějaké posloupnosti w_1, w_2, w_3, \dots , např. $\varepsilon, x, y, xx, xy, yy, yy, xxx, \dots$ pro $\Sigma = \{x, y\}$. Nyní můžeme f zobrazit *nekonečnou maticí*:

$$\begin{array}{cccccc} & w_0 & w_1 & w_2 & \dots & w_i & \dots \\ L_0 = f(0) & a_{00} & a_{01} & a_{02} & \dots & a_{0i} & \dots \\ L_1 = f(1) & a_{10} & a_{11} & a_{12} & \dots & a_{1i} & \dots \\ L_2 = f(2) & a_{20} & a_{21} & a_{22} & \dots & a_{2i} & \dots \\ \dots & & & & & & \end{array}, \text{ kde } a_{ij} = \begin{cases} 0, & \text{jestliže } w_j \notin L_i, \\ 1, & \text{jestliže } w_j \in L_i. \end{cases}$$

- Uvažujme jazyk $\bar{L} = \{w_i \mid a_{ii} = 0\}$. \bar{L} se liší od každého jazyka $L_i = f(i)$, $i \in \mathbb{N}$:
 - je-li $a_{ii} = 0$, pak w_i patří do jazyka,
 - je-li $a_{ii} = 1$, pak w_i nepatří do jazyka.
- Současně ale $\bar{L} \in 2^{\Sigma^*}$, f tudíž není surjektivní, což je spor.

□

6.2 Problém zastavení

Problém, zda výpočet daného TS zastaví, je prvním problémem, jehož nerekurzivnost ukážeme. Za tím účelem zavedeme (s pomocí vhodného kódování TS) *univerzální Turingův stroj*, což je TS schopný simulovat výpočet kteréhokoli jiného stroje z jakékoliv počáteční konfigurace (stroj a konfigurace jsou vstupem univerzálního stroje). Ten bude dále důležitý nejen v důkazu nerozhodnutelnosti problému zastavení.

6.2.1 Kódování TS

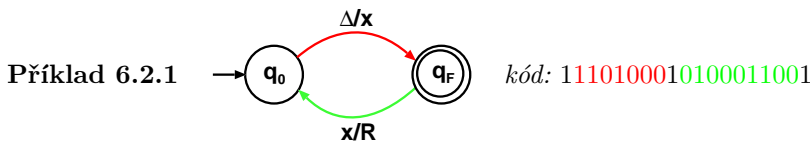
Kódovací systém pro TS zahrnuje (1) kódování stavů (tak, aby byly odlišeny všechny stavy včetně q_0 a q_F), (2) symbolů z Γ a (3) přechodové funkce δ . *Kódování stavů*: Množinu stavů Q uspořádáme do posloupnosti q_0, q_F, q, p, \dots, t . Stav q_j zakódujeme jako 0^j , přičemž indexujeme (např.) od nuly.

Kódování symbolů a příkazů L/R : Předpokládejme, že $\Gamma = \Sigma \cup \{\Delta\}$. Uspořádáme Σ do posloupnosti a_1, a_2, \dots, a_n a zvolíme tyto kódy: $\Delta \mapsto \varepsilon$, $L \mapsto 0$, $R \mapsto 00$, $a_i \mapsto 0^{i+2}$.

Přechod $\delta(p, x) = (q, y)$, kde $y \in \Gamma \cup \{L, R\}$, reprezentujeme čtveřicí (p, x, q, y) a kódujeme *zřetězením kódů* p, x, q, y při použití 1 jako *oddělovače*, tj. jako $\langle p \rangle 1 \langle x \rangle 1 \langle q \rangle 1 \langle y \rangle$, kde $\langle _ \rangle$ značí kód $_$.

Celý TS kódujeme jako *posloupnost kódů přechodů* oddělených a ohraničených 1.

DEF



6.2.2 Univerzální TS

Zavádí koncept „programovatelného“ stroje, který umožňuje ve vstupním řetězci specifikovat konkrétní TS (tj. program) i data, nad nimiž má tento stroj pracovat.

TS, který má být simulován, budeme kódovat, jak bylo uvedeno výše, vstupní řetězec budeme kódovat jako posloupnost příslušných kódů symbolů oddělených a ohraničených 1. Kód stroje a vstupního řetězce oddělíme např. #.

Příklad 6.2.2 TS z předchozí strany mající na vstupu xxx :

1110100010100011001#1000100010001

$x + y$

Univerzální TS, který zpracuje toto zadání, můžeme navrhnout jako třípáskový stroj, který

- má na 1. páске zadání (a později výstup),
- 2. pásku používá k simulaci pracovní pásky původního stroje a
- na 3. páске má zaznamenán aktuální stav simulovaného stroje a aktuální pozici hlavy (pozice hlavy i je kódována jako 0^i).

Univerzální stroj pracuje takto:

1. Stroj zkontroluje, zda vstup odpovídá zadání. Pokud ne, abnormálně zastaví.
2. Přepíše w na 2. pásku, na 3. pásku umístí kód q_0 a za něj poznačí, že hlava se nachází na levém okraji pásky.
3. Na 2. páске vyhledá aktuální symbol pod hlavou simulovaného stroje a na 1. páске vyhledá přechod proveditelný ze stavu zapsaného na začátku 3. pásky pro tento vstupní symbol. Pokud žádný přechod možný není, stroj abnormálně zastaví.
4. Stroj provede na 2. a 3. páске změny odpovídající simulovanému přechodu (přepis aktuálního symbolu, změna pozice hlavy, změna řídicího stavu).
5. Pokud nebyl dosažen stav q_F simulovaného stroje, přejdeme na bod 2. Jinak stroj vymaže 1. pásku, umístí na ní obsah 2. pásky a zastaví normálně (přechodem do koncového stavu).

Víme, že výše uvedený stroj můžeme převést na *jednopáskový univerzální TS*, který budeme v dalším značit jako T_U .

6.2.3 Problém zastavení TS

Věta 6.2.1 Problém zastavení TS (Halting Problem), kdy nás zajímá, zda daný TS M pro daný vstupní řetězec w zastaví, **není rozhodnutelný**, ale je **částečně rozhodnutelný**.

!

Důkaz.

- Problému zastavení odpovídá rozhodování jazyka $HP = \{\langle M \rangle \# \langle w \rangle \mid M \text{ zastaví při } w\}$, kde $\langle M \rangle$ je kód TS M a $\langle w \rangle$ je kód w .

- Částečnou rozhodnutelnost ukážeme snadno použitím modifikovaného T_U , který zastaví přijetím vstupu $\langle M \rangle \# \langle w \rangle$ právě tehdy, když M zastaví při w – modifikace spočívá v převedení abnormálního zastavení při simulaci na zastavení přechodem do q_F .
- Nerozhodnutelnost ukážeme pomocí diagonalizace:
 1. Pro $x \in \{0, 1\}^*$, nechť M_x je TS s kódem x , je-li x legální kód TS. Jinak ztotožníme M_x s pevně zvoleným TS, např. s TS, který pro libovolný vstup okamžitě zastaví.
 2. Můžeme nyní sestavit posloupnost $M_\varepsilon, M_0, M_1, M_{00}, M_{01}, M_{10}, M_{11}, M_{000}, \dots$ zahrnující všechny TS nad $\Sigma = \{0, 1\}$ indexované řetězcí z $\{0, 1\}^*$.

3. Uvažme nekonečnou matici

	ε	0	1	00	01	10	...
M_ε	$H_{M_\varepsilon, \varepsilon}$	$H_{M_\varepsilon, 0}$	$H_{M_\varepsilon, 1}$	$H_{M_\varepsilon, 00}$	$H_{M_\varepsilon, 01}$...	
M_0	$H_{M_0, \varepsilon}$	$H_{M_0, 0}$	$H_{M_0, 1}$	$H_{M_0, 00}$	$H_{M_0, 01}$...	
M_1	$H_{M_1, \varepsilon}$	$H_{M_1, 0}$	$H_{M_1, 1}$	$H_{M_1, 00}$	$H_{M_1, 01}$...	
M_{00}	$H_{M_{00}, \varepsilon}$	$H_{M_{00}, 0}$	$H_{M_{00}, 1}$	$H_{M_{00}, 00}$	$H_{M_{00}, 01}$...	
M_{01}	$H_{M_{01}, \varepsilon}$	$H_{M_{01}, 0}$	$H_{M_{01}, 1}$	$H_{M_{01}, 00}$	$H_{M_{01}, 01}$...	
...							

kde $H_{M_x, y} = \begin{cases} \mathbf{C}, & \text{jestliže } M_x \text{ cyklí na } y, \\ \mathbf{Z}, & \text{jestliže } M_x \text{ zastaví na } y. \end{cases}$

4. Předpokládejme, že existuje úplný TS K přijímající jazyk HP . Pak K pro vstup $\langle M \rangle \# \langle w \rangle$
 - zastaví normálně (přijme) právě tehdy, když M zastaví na w ,
 - zastaví abnormálně (odmítne) právě tehdy, když M cyklí na w .
5. Sestavíme TS N , který pro vstup $x \in \{0, 1\}^*$:
 - Sestaví M_x z x a zapíše $\langle M_x \rangle \# x$ na svou pásku.
 - Simuluje K na $\langle M_x \rangle \# x$, přijme, pokud K odmítne, a přejde do nekonečného cyklu, pokud K přijme.

Všimněme si, že N v podstatě komplementuje diagonálu uvedené matice:

	ε	0	1	00	01	10	...
M_ε	$H_{M_\varepsilon, \varepsilon}$	$H_{M_\varepsilon, 0}$	$H_{M_\varepsilon, 1}$	$H_{M_\varepsilon, 00}$	$H_{M_\varepsilon, 01}$...	
M_0	$H_{M_0, \varepsilon}$	$H_{M_0, 0}$	$H_{M_0, 1}$	$H_{M_0, 00}$	$H_{M_0, 01}$...	
M_1	$H_{M_1, \varepsilon}$	$H_{M_1, 0}$	$H_{M_1, 1}$	$H_{M_1, 00}$	$H_{M_1, 01}$...	
M_{00}	$H_{M_{00}, \varepsilon}$	$H_{M_{00}, 0}$	$H_{M_{00}, 1}$	$H_{M_{00}, 00}$	$H_{M_{00}, 01}$...	
M_{01}	$H_{M_{01}, \varepsilon}$	$H_{M_{01}, 0}$	$H_{M_{01}, 1}$	$H_{M_{01}, 00}$	$H_{M_{01}, 01}$...	
...							

6. Dostáváme, že

$$\begin{aligned} N \text{ zastaví na } x &\Leftrightarrow K \text{ odmítne } \langle M_x \rangle \# \langle x \rangle && (\text{definice } N) \\ &\Leftrightarrow M_x \text{ cyklí na } x && (\text{předpoklad o } K). \end{aligned}$$

7. To ale znamená, že N se liší od každého M_x alespoň na jednom řetězci – konkrétně na x . To je ovšem spor s tím, že posloupnost $M_\varepsilon, M_0, M_1, M_{00}, M_{01}, M_{10}, M_{11}, M_{000}, \dots$ zahrnuje všechny TS nad $\Sigma = \{0, 1\}$.

Tento spor plyne z předpokladu, že existuje TS K , který pro daný TS M a daný vstup x určí (rozhodne), zda M zastaví na x , či nikoliv.

□

Ukázali jsme, že jazyk HP je rekurzivně vyčíslitelný ale není rekurzivní, a tedy že problém zastavení TS je jen částečně rozhodnutelný. Z věty 8.6 pak plyne, že **komplement problému zastavení není ani částečně rozhodnutelný** a jazyk $co-HP = \{\langle M \rangle \# \langle w \rangle \mid M \text{ nezastaví při } w\}$ je příkladem jazyka, jenž není ani rekurzivně vyčíslitelný. S dalším příkladem takového jazyka se seznámíme v následujících kapitolách.

6.3 Redukce

Redukce je základní technikou klasifikace problémů z hlediska vyčíslitelnosti. Jedná se o algoritmický převod problému na jiný problém, tedy neformálně o vyčíslitelnou redukční funkci f , která každé instanci I problému P přiřadí instanci $f(I)$ problému Q tak, že řešení $f(I)$ je právě řešením I ¹.

6.3.1 Důkaz nerozhodnutelnosti redukcí

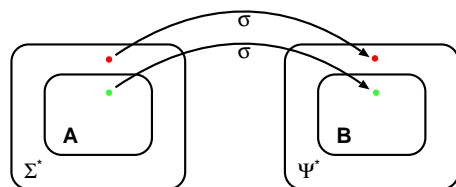
Technika *redukce* patří spolu s diagonalizací k nejpoužívanějším technikám důkazu, že problém není rozhodnutelný (částečně rozhodnutelný), neboli že jazyk není rekurzivní (rekurzivně vyčíslitelný):

- víme, že jazyk A není rekurzivní (rekurzivně vyčíslitelný),
- zkoumáme jazyk B ,
- ukážeme, že A lze úplným TS převést (redukovat) na B ,
- to ale znamená, že B rovněž není rekurzivní (rekurzivně vyčíslitelný) – jinak by šlo použít úplný TS (ne-úplný TS) přijímající B a příslušné redukce k sestavení úplného TS (ne-úplného TS) přijímajícího A , což by byl spor.

Argumentace výše samozřejmě ukazuje, že redukci lze použít i při dokazování, že určitý problém je rekurzivní (částečně rekurzivní).

Definice 6.3.1 Necht A, B jsou jazyky, $A \subseteq \Sigma^*$, $B \subseteq \Psi^*$. Redukce jazyka A na jazyk B je rekurzivně vyčíslitelná funkce $\sigma : \Sigma^* \rightarrow \Psi^*$ taková, že $w \in A \Leftrightarrow \sigma(w) \in B$.

DEF



Existuje-li redukce jazyka A na B , říkáme, že A je redukovatelný na B , což značíme $A \leq B$.

¹ P a Q jsou rozhodovací problémy

Věta 6.3.1 *Nechť $A \leq B$.*

1. *Není-li jazyk A rekurzivně vyčíslitelný, pak ani jazyk B není rekurzivně vyčíslitelný.*
2. *Není-li jazyk A rekurzivní, pak ani jazyk B není rekurzivní.*
- $\bar{1}$. *Je-li jazyk B rekurzivně vyčíslitelný, pak i jazyk A je rekurzivně vyčíslitelný.*
- $\bar{2}$. *Je-li jazyk B rekurzivní, pak i jazyk A je rekurzivní.*

Důkaz. Dokážeme, že pokud $A \leq B$, pak $(\bar{1})$ je-li jazyk B rekurzivně vyčíslitelný, pak i jazyk A je rekurzivně vyčíslitelný:

- Nechť M_R je úplný TS počítající redukci σ z A na B a M_B je TS přijímající B .
- Sestrojíme M_A přijímající A :
 1. M_A simuluje M_R se vstupem w , čímž transformuje obsah pásky na $\sigma(w)$.
 2. M_A simuluje výpočet M_B na $\sigma(w)$.
 3. Pokud M_B zastaví a přijme, M_A rovněž zastaví a přijme, jinak M_A zastaví abnormálně nebo cyklí.
- Zřejmě platí:

$$\begin{aligned} M_A \text{ přijme } w &\Leftrightarrow M_B \text{ přijme } \sigma(w) \\ &\Leftrightarrow \sigma(w) \in B \\ &\Leftrightarrow w \in A \end{aligned} \quad (\text{definice redukce}).$$

Tvrzení (1) je kontrapozicí $(\bar{1})$; tvrzení $(\bar{2})$ dokážeme podobně jako $(\bar{1})$ při použití úplného TS M_B ; tvrzení (2) je kontrapozicí² $(\bar{2})$.

□

6.4 Problém náležitosti a další problémy

Redukce z problému zastavení přináší množství „špatných zpráv“ o nerozhodnutelnosti významných problémů, jako je např. problém příslušnosti slova do daného jazyka.

6.4.1 Problém náležitosti pro \mathcal{L}_0

Věta 6.4.1 **Problém náležitosti (Membership Problem)** řetězce w do jazyka L typu 0 **není rozhodnutelný**, ale je **částečně rozhodnutelný**.

Důkaz. Částečná rozhodnutelnost je zřejmá: jednoduše použijeme T_U , který bude simulovat TS M , $L(M) = L$, nad daným řetězcem w . Nerozhodnutelnost ukážeme redukcí z problému zastavení:

²Kontrapozice: $p \rightarrow q \Leftrightarrow \neg q \rightarrow \neg p$ (Modus tollens).

- Libovolný TS M můžeme snadno upravit na M' , který přijme w právě tehdy, když M zastaví (jakkoli) na vstupu w : postačí dodat veškeré „chybějící“ přechody jejich svedením do q_F , dodat počáteční poznačení levého okraje unikátním symbolem a přechod do q_F , který bude použit, kdykoliv se hlava posune na tento symbol.
- Uvedenou transformaci lze zřejmě snadno realizovat na úrovni kódů Turingových strojů *úplným TS* – ten tak bude implementovat *redukcí jazyka HP* na jazyk $MP = \{\langle M \rangle \# \langle w \rangle \mid w \in L(M)\}$.
- Jsme tedy schopni redukovat *HP* úplným TS na *MP* a současně víme, že *HP* není rekurzivní. Dle věty 10.3 (2) tedy máme, že *MP* není rekurzivní, a tedy problém náležitosti pro jazyky typu 0 není rozhodnutelný.

□

Podobně jako u problému zastavení nyní z věty 8.6 plyne, že

- **komplement problému náležitosti** není ani částečně rozhodnutelný a
- jazyk $\text{co-}MP = \{\langle M \rangle \# \langle w \rangle \mid w \notin L(M)\}$ je dalším příkladem jazyka, jenž není ani rekurzivně vyčíslitelný.

6.4.2 Příklady dalších problémů pro TS

Konstrukcí příslušného *úplného TS* (a v případě složitější konstrukce důkazem její korektnosti) lze ukázat, že např. *následující problémy jsou rozhodnutelné*:

- Daný TS má alespoň 2005 stavů.
- Daný TS učiní více než 2005 kroků na vstupu ε .
- Daný TS učiní více než 2005 kroků na *nějakém* vstupu.

Konstrukcí příslušného (*ne-úplného*) TS a důkazem *nerekurzivnosti redukcí* lze ukázat, že např. *následující problémy jsou právě částečně rozhodnutelné*:

- Jazyk daného TS je neprázdný.
- Jazyk daného TS obsahuje alespoň 2005 slov.

Důkazem redukcí lze ukázat, že jazyky odpovídající následujícím problémům *nejsou ani parciálně rekurzivní* – *následující problémy nejsou ani částečně rozhodnutelné*:

- Jazyk daného TS je prázdný.
- Jazyk daného TS obsahuje nanejvýš 2005 slov.
- Jazyk daného TS je konečný (regulární, bezkontextový, kontextový, rekurzivní).

6.5 Postův korespondenční problém

Podobně jako problém zastavení je Postův korespondenční problém důležitý jako východisko redukce při dokazování nerozhodnutelnosti mnoha problémů.

$x + y$

6.5.1 Postův korespondenční problém

Definice 6.5.1

DEF

- Postův systém nad abecedou Σ je dán neprázdným seznamem S dvojic neprázdných řetězců nad Σ , $S = \langle (\alpha_1, \beta_1), \dots, (\alpha_k, \beta_k) \rangle$, $\alpha_i, \beta_i \in \Sigma^+$, $k \geq 1$.
- Řešením Postova systému je každá neprázdná posloupnost přirozených čísel $I = \langle i_1, i_2, \dots, i_m \rangle$, $1 \leq i_j \leq k$, $m \geq 1$, taková, že:

$$\alpha_{i_1} \alpha_{i_2} \dots \alpha_{i_m} = \beta_{i_1} \beta_{i_2} \dots \beta_{i_m}$$

(Pozor: m není omezené a indexy se mohou opakovat!)

- Postův problém (PCP) zní: *Existuje pro daný Postův systém řešení?*

Příklad 6.5.1

x + y

- Uvažujme Postův systém $S_1 = \{(b, bbb), (babbb, ba), (ba, a)\}$ nad $\Sigma = \{a, b\}$. Tento systém má řešení $I = \langle 2, 1, 1, 3 \rangle$: $babbb \ b \ b \ ba = ba \ bbb \ bbb \ a$.
- Naopak Postův systém $S_2 = \{(ab, abb), (a, ba), (b, bb)\}$ nad $\Sigma = \{a, b\}$ nemá řešení, protože $|\alpha_i| < |\beta_i|$ pro $i = 1, 2, 3$.

6.5.2 Nerozhodnutelnost PCP

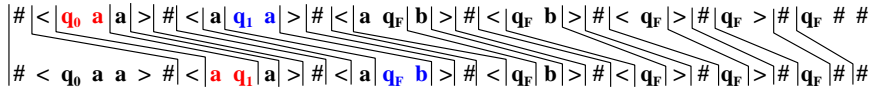
Věta 6.5.1 Postův korespondenční problém je nerozhodnutelný.

!

Důkaz. (Idea) Dá se ukázat, že nerozhodnutelnost PCP plyne z nerozhodnutelnosti tzv. *iniciálního PCP*, u kterého požadujeme, aby řešení začínalo vždy jedničkou. Nerozhodnutelnost iniciálního PCP se dá ukázat *redukcí z problému náležitosti pro TS*:

- Konfiguraci výpočtu TS lze zakódovat do řetězce: použitý obsah pásky ohraňujeme speciálními značkami (a jen ten si pamatujeme), řídicí stav poznáme speciálním symbolem před symbol pásky nacházející se aktuálně pod hlavou (kódujeme tak i pozici hlavy).
- Posloupnost konfigurací TS při přijetí řetězce budeme reprezentovat jako konkatenaci řetězců, která vzniká řešením PCP.
- Jedna z uvažovaných konkatenací bude celou dobu (až na poslední fázi) delší: na začátku bude obsahovat počáteční konfiguraci a pak bude vždy o krok napřed. V poslední fázi výpočtu konkatenace „zarovnáme“ (bude-li možný výpočet simulovaného TS ukončit přijetím).
- Výpočet TS budeme modelovat tak, že vždy jednu konkatenaci postupně prodloužíme o aktuální konfiguraci simulovaného TS a současně v druhé konkatenaci vygenerujeme novou konfiguraci TS.
- *Jednotlivé dvojice PCP budou modelovat následující kroky:*

- Vložení počáteční konfigurace simulovaného TS do jedné z konkatenačí, např. pravostranné ($\#, \#$ počáteční_konfigurace), $\# \notin \Gamma$ používáme jako oddělovač konfigurací.
 - Kopírování symbolů na pásce před a po aktuální pozici hlavy (z, z) pro každé $z \in \Gamma \cup \{\#, <, >\}$, kde $<, >$ ohraničují použitou část pásky.
 - Základní změna konfigurace: přepis $\delta(q_1, a) = (q_2, b)$: $(q_1 a, q_2 b)$, posuv doprava $\delta(q_1, a) = (q_2, R)$: $(q_1 a, a q_2)$, posuv doleva $\delta(q_1, b) = (q_2, L)$: $(a q_1 b, q_2 a b)$ pro každé $a \in \Gamma \cup \{<\}$. Navíc je zapotřebí ošetřit nájezd na $>$: čtení Δ , rozšiřování použité části pásky.
 - Pravidla pro „zarovnání“ obou konkatenačí při přijetí: na levé straně umožníme přidat symbol v okolí q_F , aniž bychom ho přidali na pravé straně.
- Simulace výpočtu TS, který načte a , posune hlavu doprava, přepíše a na b a zastaví, by pak na vstupu aa vypadala takto:



- Obecná *korektnost konstrukce* se dá dokázat indukcí k délce výpočtu.

□

6.5.3 Nerozhodnutelnost redukcí z PCP

Redukce z PCP či jeho doplňku se velmi často používají k důkazům nerozhodnutelnosti.

Jako příklad uvedeme *důkaz nerozhodnutelnosti problému prázdnosti jazyka dané kontextové gramatiky*:

- Použijeme *redukcí z komplementu PCP*. Redukce přiřadí seznamu $S = (\alpha_1, \beta_1), \dots, (\alpha_k, \beta_k)$, definujícímu instanci PCP, kontextovou gramatiku G takovou, že PCP založený na S nemá řešení právě tehdy, když $L(G) = \emptyset$.
- Uvažme jazyky L_α, L_β nad $\Sigma \cup \{\#, 1, \dots, k\}$ (předp. $\Sigma \cap \{\#, 1, \dots, k\} = \emptyset$):
 - $L_\alpha = \{\alpha_{i_1} \dots \alpha_{i_m} \# i_m \dots i_1 \mid 1 \leq i_j \leq k, j = 1, \dots, m, m \geq 1\}$,
 - $L_\beta = \{\beta_{i_1} \dots \beta_{i_m} \# i_m \dots i_1 \mid 1 \leq i_j \leq k, j = 1, \dots, m, m \geq 1\}$.
- Je zřejmé, že L_α, L_β jsou kontextové (dokonce deterministické bezkontextové), tudíž $L_\alpha \cap L_\beta$ je také kontextový (věta 8.10) a můžeme tedy efektivně sestavit gramatiku G , která tento jazyk generuje (např. konstrukcí přes LOA).
- $L_\alpha \cap L_\beta$ zřejmě obsahuje právě řetězce $u\#v$, kde v odpovídá inverzi řešení dané instance PCP.
- Hledaná redukce tedy přiřadí dané instanci PCP gramatiku G .

□

6.5.4 Souhrn některých vlastností jazyků

Uvedeme nyní souhrn některých důležitých vlastností probraných tříd jazyků; některé jsme dokázali, důkazy jiných lze nalézt v literatuře³ (u otázek nerozhod-

³Např. I. Černá, M. Křetínský, A. Kučera. Automaty a formální jazyky I. FI MU, 1999.

nutelnosti se často užívá redukce z PCP):⁴

	Reg	DCF	CF	CS	Rec	RE
$w \in L(G)?$	R	R	R	R	R	N
$L(G)$ prázdný? konečný?	R	R	R	N	N	N
$L(G) = \Sigma^*$?	R	R	N	N	N	N
$L(G) = R, R \in \mathcal{L}_3?$	R	R	N	N	N	N
$L(G_1) = L(G_2)?$	R	R	N	N	N	N
$L(G_1) \subseteq L(G_2)?$	R	N	N	N	N	N
$L(G_1) \in \mathcal{L}_3?$	A	R	N	N	N	N
$L(G_1) \cap L(G_2)$ je stejného typu?	A	N	N	A	A	A
$L(G_1) \cup L(G_2)$ je stejného typu?	A	N	A	A	A	A
Komplement $L(G)$ je stejného typu?	A	A	N	A	A	N
$L(G_1).L(G_2)$ je stejného typu?	A	N	A	A	A	A
$L(G)^*$ je stejného typu?	A	N	A	A	A	A
Je G víceznačná?	R	N	N	N	N	N

6.6 Riceova věta

Riceova věta je důležitým nástrojem vyčíslitelnostní klasifikace jazyků (založeným na redukci z HP a $co-HP$). Znovu se ukazuje, že **rozhodnutelnost je výjimkou, která potvrzuje pravidlo**.

6.6.1 Riceova věta – první část

Věta 6.6.1 *Každá netriviální vlastnost rekurzivně vyčíslitelných jazyků je nerozhodnutelná.*

Definice 6.6.1 *Budiž dána abeceda Σ . Vlastnost rekurzivně vyčíslitelných množin je zobrazení $P : \{ \text{rekurzivně vyčíslitelné podmnožiny množiny } \Sigma^* \} \rightarrow \{\perp, \top\}$, kde \top resp. \perp reprezentují pravdu resp. nepravdu.*

DEF

Příklad 6.6.1 *Vlastnost prázdnoty můžeme reprezentovat jako zobrazení*

$$P(A) = \begin{cases} \top, & \text{jestliže } A = \emptyset, \\ \perp, & \text{jestliže } A \neq \emptyset. \end{cases}$$

Zdůrazněme, že nyní mluvíme o vlastnostech rekurzivně vyčíslitelných množin, *nikoliv* o vlastnostech TS, které je přijímají. Následující vlastnosti tedy nejsou vlastnostmi r.v. množin:

- TS M má alespoň 2005 stavů.
- TS M zastaví na všech vstupech.

Definice 6.6.2 *Vlastnost rekurzivně vyčíslitelných množin je netriviální, pokud není vždy (pro všechny r.v. množiny) pravdivá ani vždy nepravdivá.*

⁴R = rozhodnutelný, N = nerozhodnutelný, A = vždy splněno

6.6.2 Důkaz 1. části Riceovy věty

Důkaz.

- Nechť P je netriviální vlastnost r.v. množin. Předpokládejme beze ztráty obecnosti, že $P(\emptyset) = \perp$, pro $P(\emptyset) = \top$ můžeme postupovat analogicky.
- Jelikož P je netriviální vlastnost, existuje r.v. množina A taková, že $P(A) = \top$. Nechť K je TS přijímající A .
- *Redukujeme HP na $\{\langle M \rangle \mid P(L(M)) = \top\}$.* Z $\langle M \rangle \# \langle w \rangle$ sestrojíme $\sigma(\langle M \rangle \# \langle w \rangle) = \langle M' \rangle$, kde M' je 2-páskový TS, který na vstupu x :
 1. Uloží x na 2. pásku.
 2. Zapiše na 1. pásku $w - w$ je „uložen“ v řízení M' .
 3. Odsimuluje na 1. pásce $M - M$ je rovněž „uložen“ v řízení M' .
 4. Pokud M zastaví na w , odsimuluje K na x a přijme, právě když K přijme.
- Dostáváme:
 - M zastaví na $w \Rightarrow L(M') = A \Rightarrow P(L(M')) = P(A) = \top$,
 - M cyklí na $w \Rightarrow L(M') = \emptyset \Rightarrow P(L(M')) = P(\emptyset) = \perp$,

Máme tedy skutečně redukci HP na $\{\langle M \rangle \mid P(L(M)) = \top\}$.

Protože HP není rekurzivní, není rekurzivní ani $P(L(M))$, a tudíž není rozhodnutelné, zda $L(M)$ splňuje vlastnost P .

□

6.6.3 Riceova věta – druhá část

Definice 6.6.3 *Vlastnost P r.v. množin nazveme monotónní, pokud pro každé dvě r.v. množiny A, B takové, že $A \subseteq B$, platí $P(A) \Rightarrow P(B)$.*

Příklad 6.6.2 *Mezi monotónní vlastnosti patří např.:*

- A je nekonečné.
- $A = \Sigma^*$.

Naopak mezi nemonotónní vlastnosti patří např.:

- A je konečné.
- $A = \emptyset$.

Věta 6.6.2 *Každá netriviální nemonotónní vlastnost rekurzivně vyčíslitelných jazyků není ani částečně rozhodnutelná.*

Důkaz. Redukcí z $co\text{-}HP$ – viz např. D. Kozen. Automata and Computability. □

DEF

6.7 Alternativy Turingova stroje

Mezi výpočetní mechanismy mající ekvivalentní výpočetní sílu jako TS patří např. *automaty s (jednou) frontou*:

- Uvažme stroj s konečným řízením, (neomezenou) FIFO frontou a přechody, které dovolují načíst ze začátku fronty a zapsat na konec fronty symboly z frontové abecedy Γ .
- Pomocí „rotace“ fronty je zřejmě možné simulovat pásku TS.

Ekvivalentní výpočetní sílu jako TS mají také *zásobníkové automaty se dvěma (a více) zásobníky*:

- Intuitivně: obsah pásky simulovaného TS máme v jednom zásobníku; chceme-li ho změnit (obecně nejen na vrcholu), přesuneme část do druhého zásobníku, abychom se dostali na potřebné místo, provedeme patřičnou změnu a vrátíme zpět odloženou část zásobníku.
- Poznámka: rovněž víme, že pomocí dvou zásobníků můžeme implementovat frontu.

Jiným výpočetním modelem s plnou Turingovskou silou jsou *automaty s čítači (pro dva a více čítačů) a operacemi $+1$, -1 a test na 0*:

- Zmíněné automaty mají konečné řízení a k čítačů, přičemž v každém kroku je možné tyto čítače nezávisle inkrementovat, dekrementovat a testovat na nulu (přechod je podmíněn tím, že jistý čítač obsahuje nulu).
- Pomocí čtyř čítačů je snadné simulovat dva zásobníky:
 - U ZA postačí mít $\Gamma = \{0, 1\}$: různé symboly můžeme kódovat určitým počtem 0 oddělených 1. Obsah zásobníku má pak charakter binárně zapsaného čísla. Vložení 0 odpovídá vynásobení 2, odebrání 0, vydělení 2. Podobně je tomu s vložím/odebráním 1.
 - Binární zásobník můžeme simulovat dvěma čítači: při násobení/dělení 2 odečítáme 1 (resp. 2) z jednoho čítače a přičítáme 2 (resp. 1) k druhému.
- Postačí ovšem i čítače dva:
 - Obsah čtyř čítačů i, j, k, l je možné zakódovat jako $2^i 3^j 5^k 7^l$.
 - Přičtení/odečtení je pak možné realizovat násobením/dělením 2, 3, 5, či 7.

Mezi další Turingovsky úplné výpočetní mechanismy pak patří např. λ -kalkul či *parciálně-rekurzivní funkce* (viz následující kapitola).

6.8 Vyčíslitelné funkce

6.8.1 Základy teorie rekurzivních funkcí

Budeme se snažit identifikovat takové funkce, které jsou „spočitatelné“, tj. vyčíslitelné v obecném smyslu (bez ohledu na konkrétní výpočetní systém). Abychom snížili extrémní velikost třídy těchto funkcí, která je dána také varietou definičních oborů a oborů hodnot, omezíme se, uvažující možnost kódování, na funkce tvaru:

$$f : \mathbb{N}^m \rightarrow \mathbb{N}^n$$

kde $\mathbb{N} = \{0, 1, 2, \dots\}$, $m, n \in \mathbb{N}$.

Konvence: n -tici $(x_1, x_2, \dots, x_n) \in \mathbb{N}^n$ budeme označovat jako \bar{x}

Klasifikace parciálních funkcí:

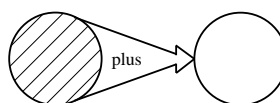
- *Totální funkce nad množinou X ⁵* – definičním oborem je celá X
- *Striktně parciální funkce nad množinou X* – alespoň pro jeden prvek $x \in X$ není definována funkční hodnota

DEF

Příklad 6.8.1 *Totální funkce plus*

$$plus : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$plus(x, y) = x + y$$

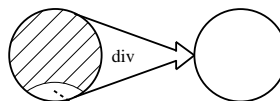
 $\mathbb{N} \times \mathbb{N}$

 \mathbb{N}

$$x + y$$

Příklad 6.8.2 *Striktně parciální funkce div*

$$div : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$$

$$div(x, y) = \text{celá část } x/y, \text{ je-li } y \neq 0$$

 $\mathbb{N} \times \mathbb{N}$

 \mathbb{N}

6.8.2 Počáteční funkce

Hierarchie vyčíslitelných funkcí je založena na dostatečně elementárních tzv. *počátečních funkcích*, které tvoří „stavební kameny“ vyšších funkcí.

Jsou to tyto funkce:

1. *Nulová funkce* (zero function): $\xi() = 0$
zobrazuje „prázdnou n -tici“ $\mapsto 0$
2. *Funkce následníka* (successor function): $\sigma : \mathbb{N} \rightarrow \mathbb{N}$
 $\sigma(x) = x + 1$
3. *Projekce* (projection): $\pi_k^n : \mathbb{N}^n \rightarrow \mathbb{N}$
Vybírá z n -tice k -tou složku, např.: $\pi_2^3(7, 6, 4) = 6$ a $\pi_1^2(5, 17) = 5$

DEF

Speciální případ: $\pi_0^n : \mathbb{N}^n \rightarrow \mathbb{N}^0$, tj. např. $\pi_0^3(1, 2, 3) = ()$

6.8.3 Primitivně rekurzivní funkce

Nyní definujme tři způsoby vytváření nových, složitějších funkcí. *Primitivně rekurzivní funkce* pak budou takto tvořeny z funkcí počátečních.

1. *Kombinace*:

Kombinací dvou funkcí $f : \mathbb{N}^k \rightarrow \mathbb{N}^m$ a $g : \mathbb{N}^k \rightarrow \mathbb{N}^n$ získáme funkci, pro kterou:

$$\begin{aligned} f \times g : \mathbb{N}^k &\rightarrow \mathbb{N}^{m+n} \\ f \times g(\bar{x}) &= (f(\bar{x}), g(\bar{x})), \bar{x} \in \mathbb{N}^k \end{aligned}$$

DEF

⁵V dalším bude množinou X obvykle \mathbb{N}^k .

Např.: $\pi_1^3 \times \pi_3^3(4, 12, 8) = (4, 8)$

2. *Kompozice*: Kompozice dvou funkcí $f : \mathbb{N}^k \rightarrow \mathbb{N}^m$ a $g : \mathbb{N}^m \rightarrow \mathbb{N}^n$ je funkce, pro kterou:

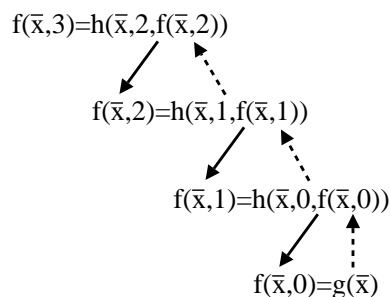
$$\begin{array}{l} g \circ f : \mathbb{N}^k \rightarrow \mathbb{N}^n \\ g \circ f(\bar{x}) = g(f(\bar{x})), \bar{x} \in \mathbb{N}^k \end{array}$$

Např.: $\sigma \circ \xi() = 1$
 $\sigma \circ \sigma \circ \xi() = 2$

3. *Primitivní rekurze* je technika, která umožňuje vytvořit funkci $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}^m$ na základě jiných dvou funkcí $g : \mathbb{N}^k \rightarrow \mathbb{N}^m$ a $h : \mathbb{N}^{k+m+1} \rightarrow \mathbb{N}^m$ rovnicemi:

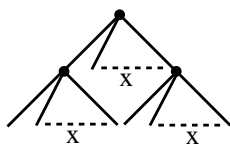
$$\begin{array}{l} f(\bar{x}, 0) = g(\bar{x}) \\ f(\bar{x}, y + 1) = h(\bar{x}, y, f(\bar{x}, y)), \bar{x} \in \mathbb{N}^k \end{array}$$

Ilustrace schématu vyčíslení (pro $y = 3$):



Příklad 6.8.3 Předpokládejme, že chceme definovat funkci $f : \mathbb{N}^2 \rightarrow \mathbb{N}$, jejíž hodnoty $f(x, y)$ udávají počet vrcholů x -árního stromu hloubky y , který je pravidelný (úplný):

$$x + y$$



Zřejmé:

1. $f(x, 0) = 1$

2. strom hloubky y má x^y listů. Zvětšením hloubky o jedničku na $y + 1$ musíme přidat $x^{y+1} = x^y \cdot x$ vrcholů

Takže funkci f můžeme definovat následujícím předpisem: $f(x, 0) = x^0$, $f(x, y + 1) = f(x, y) + x^y \cdot x$.

Např.: $f(3, 2) = f(3, 1) + 3^1 \cdot 3 = f(3, 0) + 3^0 \cdot 3 + 3^1 \cdot 3 = 1 + 3 + 9 = 13$

Příklad 6.8.4 Uvažujme funkci $plus : \mathbb{N}^2 \rightarrow \mathbb{N}$. Může být definována pomocí primitivní rekurze takto:

$$\begin{aligned} plus(x, 0) &= \pi_1^1(x) \\ plus(x, y + 1) &= \sigma \circ \pi_3^3(x, y, plus(x, y)) \end{aligned}$$

což vyjadřuje:

1. $x + 0 = x$
2. $x + (y + 1) = (x + y) + 1 = \sigma(x + y)$

Definice 6.8.1 Třída primitivních rekurzivních funkcí obsahuje všechny funkce, které mohou být z počátečních funkcí vytvořeny:

1. kombinací,
2. kompozicí a
3. primitivní rekurzí.

DEF

Věta 6.8.1 Každá primitivní rekurzivní funkce je totální funkcí.

Důkaz. Počáteční funkce jsou totální. Aplikací kombinace, kompozice a primitivní rekurze na totální funkce dostaneme totální funkce. \square

6.8.4 Příklady primitivně rekurzivních funkcí

Třída primitivně rekurzivních funkcí zahrnuje většinu funkcí typických v aplikacích počítačů. Ukážeme, jak je definovat z počátečních pomocí kombinace, kompozice a primitivní rekurze.

Konvence: Namísto funkcionálních zápisů typu $h \equiv plus \circ (\pi_1^3 \times \pi_3^3)$ budeme někdy používat zápis $h(x, y, z) = plus(x, z)$ nebo $h(x, y, z) = x + z$.

Konstantní funkce: Zavedeme funkci κ_m^n , která libovolné n -tici $\bar{x} \in \mathbb{N}^n$ přiřadí konstantní hodnotu $m \in \mathbb{N}$

$$\kappa_m^0 \equiv \underbrace{\sigma \circ \sigma \circ \dots \circ \sigma}_m \xi$$

Také pro $n > 0$ je κ_m^n funkce rekurzivně primitivní:

$$\begin{aligned} \kappa_m^n(\bar{x}, 0) &= \kappa_m^{n-1}(\bar{x}) \\ \kappa_m^n(\bar{x}, y + 1) &= \pi_{n+1}^{n+1}(\bar{x}, y, \kappa_m^n(\bar{x}, y)) \end{aligned}$$

Např.: $\kappa_3^2(1, 1) = \kappa_3^3(1, 0, \kappa_3^2(1, 0)) = \kappa_3^2(1, 0) = \kappa_3^1(1) = \kappa_3^1(0) = \kappa_3^0() = 3$
Kombinací funkcí κ_m^n dostáváme konstanty z \mathbb{N}^n , $n > 1$.

Např.: $\kappa_2^3 \times \kappa_5^3(x, y, z) = (2, 5)$

Funkce násobení:

$$\begin{aligned} mult(x, 0) &= \kappa_0^1(x) \\ mult(x, y + 1) &= plus(x, mult(x, y)) \end{aligned}$$

 $x + y$

Funkce umocňování: $\exp : \mathbb{N}^2 \rightarrow \mathbb{N}$ - analogicky - viz cvičení.

Funkce předchůdce:

$$\begin{aligned} \text{pred}(0) &= \xi() \\ \text{pred}(y+1) &= \pi_1^2(y, \text{pred}(y)) \end{aligned}$$

Poznámka: pred je totální funkcí: $\text{pred}(0) = 0$

Funkce monus:

$$\begin{aligned} \text{monus}(x, 0) &= \pi_1^1(x) \\ \text{monus}(x, y+1) &= \text{pred}(\text{monus}(x, y)) \end{aligned}$$

Význam: $\text{monus}(x, y) = \begin{cases} x - y & \text{je-li } x \geq y \\ 0 & \text{jinak} \end{cases}$

Notace: $\text{monus}(x, y) \equiv x \dot{-} y$

Funkce eq (equal): $\text{eq}(x, y) = \begin{cases} 1 & \text{je-li } x = y \\ 0 & \text{je-li } x \neq y \end{cases}$

Definice 6.8.2 $\text{eq}(x, y) = 1 \dot{-} ((y \dot{-} x) + (x \dot{-} y))$ nebo formálněji
 $\text{eq} \equiv \text{monus} \circ (\kappa_1^2 \times (\text{plus} \circ ((\text{monus} \circ (\pi_2^2 \times \pi_1^2)) \times \text{monus} \circ (\pi_1^2 \times \pi_2^2))))$

Příklad 6.8.5 $\text{eq}(5, 3) = 1 \dot{-} ((3 \dot{-} 5) + (5 \dot{-} 3)) = 1 \dot{-} (0 + 2) = 1 \dot{-} 2 = 0$

Funkce $\neg \text{eq}$: $\neg \text{eq} \equiv \text{monus} \circ (\kappa_1^2 \times \text{eq}) \quad (\equiv 1 \dot{-} \text{eq})$

Tabulární funkce:

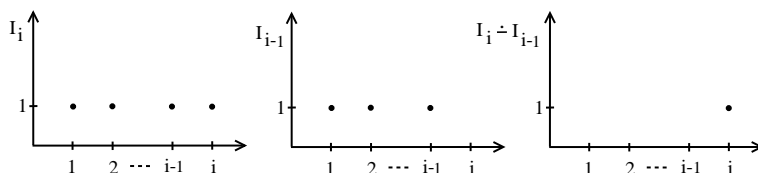
Uvažujme funkce typu:

$$f(x) = \begin{cases} 3 & \text{je-li } x = 0 \\ 5 & \text{je-li } x = 4 \\ 2 & \text{v ostatních případech} \end{cases}$$

které bývají zadávány tabulkou. Tyto funkce lze tvořit pomocí charakteristické funkce

$$\varphi_i(x) = \begin{cases} 1 & \text{je-li } x = i \\ 0 & \text{jinak} \end{cases}$$

která může být vyjádřena jako $\text{monus}(I_i, I_{i-1})$, kde $I_i(x) = \text{eq}(x \dot{-} i, 0)$



Tabulární funkce mohou být nyní tvořeny konečným součtem násobků konstant a funkcí φ_i a $\neg \varphi_i$.

Např.: nahore uvedená funkce $f(x)$ je vyjádřitelná ve tvaru:

$$f \equiv \text{mult}(3, \varphi_0) + \text{mult}(5, \varphi_4) + \text{mult}(2, \text{mult}(\neg \varphi_0, \neg \varphi_4))$$

DEF

 $x + y$

Funkce quo (quotient): $quo(x, y) = \begin{cases} \text{celá část podílu } x/y & \text{je-li } y \neq 0 \\ 0 & \text{je-li } y = 0 \end{cases}$

Tato funkce může být definována primitivní rekurzí:

$$\begin{aligned} quo(0, y) &= 0 \\ quo(x + 1, y) &= quo(x, y) + eq(x + 1, mult(quo(x, y), y) + y) \end{aligned}$$

6.8.5 Funkce mimo primitivně rekurzivní funkce

Existují funkce, které jsou vyčíslitelné a nejsou primitivně rekurzivními funkcemi. Jsou to všechny striktně parciální funkce (jako *div*), ale i totální funkce. Taková totální funkce byla prezentována W. Ackermannem (1928) a nazývá se *Ackermannova funkce*. Je dána rovnicemi:

$$\begin{aligned} A(0, y) &= y + 1 \\ A(x + 1, 0) &= A(x, 1) \\ A(x + 1, y + 1) &= A(x, A(x + 1, y)) \end{aligned}$$

Věta 6.8.2 *Existuje totální funkce z \mathbb{N} do \mathbb{N} , která není primitivně rekurzivní.*

Důkaz. Definice funkcí, které jsou primitivně rekurzivní budeme chápat jako řetězce a můžeme je uspořádat v lexikografickém pořadí s označením $f_1, f_2, \dots, f_n, \dots$.

Definujeme nyní funkci $f : \mathbb{N} \rightarrow \mathbb{N}$ tak, že $f(n) = f_n(n) + 1$ pro $\forall n \in \mathbb{N} \setminus \{0\}$. f je jasně totální a vyčíslitelná. f však není primitivně rekurzivní (kdyby byla, pak $f \equiv f_m$ pro nějaké $m \in \mathbb{N}$. Pak ale $f(m) = f_m(m)$ a ne $f_m(m) + 1$, jak vyžaduje definice funkce f).

□

Definice 6.8.3 *Třída totálních vyčíslitelných funkcí se nazývá μ -rekurzivní funkce.*

Dostáváme následující klasifikaci funkcí:

DEF



6.8.6 Parciálně rekurzivní funkce

K rozšíření třídy vyčíslitelných funkcí za totální vyčíslitelné funkce zavedeme techniku známou pod názvem *minimalizace*. Tato technika umožňuje vytvořit funkci $f : \mathbb{N}^n \rightarrow \mathbb{N}$ z jiné funkce $g : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ předpisem, v němž $f(\bar{x})$ je nejmenší y takové, že:

1. $g(\bar{x}, y) = 0$
2. $g(\bar{x}, z)$ je definována pro $\forall z < y, z \in \mathbb{N}$

Tuto konstrukci zapisujeme notací:

$$\boxed{f(\bar{x}) = \mu y[g(\bar{x}, y) = 0]}$$

Příklad 6.8.6 $f(x) = \mu y[\text{plus}(x, y) = 0]$ tj. $f(x) = \begin{cases} 0 & \text{pro } x = 0 \\ \text{nedef. jinak} \end{cases}$

$$\boxed{x + y}$$

Příklad 6.8.7 $\text{div}(x, y) = \mu t[(x + 1) \dot{-} (\text{mult}(t, y) + y) = 0]$

Příklad 6.8.8 $i(x) = \mu y[\text{monus}(x, y) = 0]$ tj. *identická funkce*

Funkce definovaná minimalizací je skutečně vyčíslitelná. Výpočet hodnoty $f(\bar{x})$ zahrnuje výpočet $g(\bar{x}, 0), g(\bar{x}, 1), \dots$ tak dlouho, pokud nedostaneme:

- $g(\bar{x}, y) = 0$ $(f(\bar{x}) = y)$,
- $g(\bar{x}, z)$ je nedefinována $(f(\bar{x}) \text{ je nedefinována})$.

Definice 6.8.4 Třída parciálně rekurzivních funkcí je *třída parciálních funkcí, které mohou být vytvořeny z počátečních funkcí aplikací*:

$$\boxed{\text{DEF}}$$

- *kombinace,*
- *kompozice,*
- *primitivní rekurze a*
- *minimalizace.*

6.9 Vztah vyčíslitelných funkcí a Turingových strojů

Turingův stroj můžeme „upravit“ tak, aby byl schopen počítat funkce. Navrhne jej tak, aby z počáteční konfigurace se vstupními parametry funkce na pásce zapsal výsledek výpočtu (požadovanou funkční hodnotu) na vstup, a jen tehdy normálně zastavil. Ukážeme, že rekurzivní funkce a Turingovy stroje vymezují pojem vyčíslitelnost funkcí stejně, t.j. funkce počítatelné Turingovými stroji jsou právě rekurzivní funkce

6.9.1 Turingovsky vyčíslitelné funkce

Definice 6.9.1 *Turingův stroj* $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ vyčísluje (počítá) *parciální funkci* $f : \Sigma^{*m} \rightarrow \Sigma_1^{*n}$, $\Sigma_1 \subseteq \Gamma$, $\Delta \notin \Sigma_1$, *jestliže pro každé* $(w_1, w_2, \dots, w_m) \in \Sigma^{*m}$ *a odpovídající počáteční konfiguraci* $\underline{\Delta}w_1\Delta w_2\Delta \dots \Delta w_m\Delta\Delta\Delta$ *stroj* M :

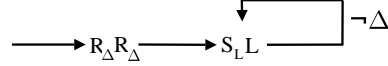
$$\boxed{\text{DEF}}$$

1. *v případě, že* $f(w_1, \dots, w_m)$ *je definována, pak* M *zastaví a páska obsahuje* $\underline{\Delta}v_1\Delta v_2\Delta \dots \Delta v_n\Delta\Delta\Delta$, *kde* $(v_1, v_2, \dots, v_n) = f(w_1, \dots, w_m)$
2. *v případě, že* $f(w_1, \dots, w_m)$ *není definována, M cykluje (nikdy nezastaví) nebo zastaví abnormálně.*

Parciální funkce, kterou může počítat nějaký Turingův stroj se nazývá *funkcí Turingovsky vyčíslitelnou*.

Příklad 6.9.1 *Turingův stroj, který počítá funkci* $f(w_1, w_2, w_3) = (w_1, w_3)$.

$$\boxed{x + y}$$



Příklad 6.9.2 *Nechť L je libovolný jazyk.*

$$\text{Funkce } f(w) = \begin{cases} |w| & \text{jestliže } w \in L \\ 0 & \text{jestliže } w \notin L \end{cases}$$

není Turingovsky vyčíslitelná.

Poznámka 6.9.1 *Definice výpočtu funkce Turingovým strojem nepředpokládala speciální pozici hlavy v koncové konfiguraci. Bez újmy na obecnosti můžeme předpokládat, že M končí v konfiguraci $\underline{\Delta}v_1\Delta \dots \Delta v_n\Delta\Delta\Delta$.*

6.9.2 Turingovská vyčíslitelnost parciálně rekurzivních funkcí

Budeme uvažovat Turingův stroj s abecedou $\Sigma = \{0, 1\}$ a parciální funkce tvaru $f : \mathbb{N}^m \rightarrow \mathbb{N}^n$, m -tice a n -tice budeme kódovat podle vzoru:

$$\Delta 11\Delta 10\Delta 100\Delta\Delta \quad \text{reprezentuje trojici} \quad (3, 2, 4)$$

Věta 6.9.1 *Každá parciálně rekurzivní funkce je Turingovsky vyčíslitelná.*

Důkaz.

1. Nejprve je třeba nalézt Turingovy stroje, které vyčíslují počáteční funkce ξ, σ, π .

(a) $\xi : \rightarrow R0L$ (b) (c) viz cvičení

2. Dále popíšeme konstrukci Turingových strojů pro aplikaci kombinace, kompozice, primitivní rekurze a minimalizace:

(a) Kombinace: Nechť Turingův stroj M_1 , resp. M_2 vyčísluje parciální funkci g_1 , resp. g_2 . Stroj M , který vyčísluje $g_1 \times g_2$ bude 3-páskový Turingův stroj, který začne duplikováním vstupu na 2. a 3. pásku. Pak M simuluje M_1 s využitím pásky 2 a M_2 s použitím pásky 3. Skončí-li M_1 i M_2 řádně, M vyčistí pásku 1 a okopíruje na ní obsah pásek 2 a 3 (oddělených blankem) a zastaví.

(b) Kompozice: Funkci $g_1 \circ g_2$ realizuje stroj $\rightarrow M_2M_1$.

(c) Primitivní rekurze: Uvažujme:

$$\begin{aligned} f(\bar{x}, 0) &= g(\bar{x}) \\ f(\bar{x}, y + 1) &= h(\bar{x}, y, f(\bar{x}, y)) \end{aligned}$$

kde g je parciální funkce, kterou vyčísluje stroj M_1 a h parciální funkce, kterou vyčísluje stroj M_2 . Funkci f vyčísluje Turingův stroj, který pracuje takto:

- i. Je-li poslední složka vstupu 0, pak vymaže tuto složku, vrátí hlavu na začátek a simuluje stroj M_1 .
- ii. Není-li poslední složkou vstupu 0, pak páska musí obsahovat sekvenci $\Delta\bar{x}\Delta y + 1\Delta\Delta\Delta$ pro nějaké $y + 1 > 0$. Potom:



- A. S využitím strojů pro kopírování a dekrement transformuj obsah pásky na sekvenci $\Delta\bar{x}\Delta y\Delta\bar{x}\Delta y - 1\Delta \dots \Delta\bar{x}\Delta 0\Delta\bar{x}\Delta\Delta$. Pak přesuň hlavu bezprostředně za 0 a simuluj stroj M_1 .
 - B. Nyní je obsahem pásky $\Delta\bar{x}\Delta y\Delta\bar{x}\Delta y - 1\Delta \dots \Delta\bar{x}\Delta 0\Delta g(\bar{x})\Delta\Delta$, což je ekvivalentní s $\Delta\bar{x}\Delta y\Delta\bar{x}\Delta y - 1\Delta \dots \Delta\bar{x}\Delta 0\Delta f(\bar{x}, 0)\Delta\Delta$. Přesuň hlavu před poslední \bar{x} a simuluj stroj M_2 . To vede k $\Delta\bar{x}\Delta y\Delta\bar{x}\Delta y - 1\Delta \dots \Delta\bar{x}\Delta 1\Delta h(\bar{x}, 0, f(\bar{x}, 0))\Delta\Delta$, což je ekvivalentní s $\Delta\bar{x}\Delta y\Delta\bar{x}\Delta y - 1\Delta \dots \Delta\bar{x}\Delta 1\Delta f(\bar{x}, 1)\Delta\Delta$.
 - C. Pokračuj v aplikaci stroje M_2 na zbytek pásky až je M_2 aplikován na $\Delta\bar{x}\Delta y\Delta f(\bar{x}, y)$ a páska je zredukována do tvaru $\Delta h(\bar{x}, y, f(\bar{x}, y))\Delta\Delta$, což je ekvivalentní žádanému výstupu $\Delta f(\bar{x}, y + 1)\Delta\Delta$.
- (d) Minimalizace: Uvažujme funkci $\mu y[g(\bar{x}, y) = 0]$, kde parciální funkci g vyčísluje stroj M_1 . Zkonstruuje 3-páskový stroj M , který pracuje takto:
- i. Zapiše 0 na pásku 2.
 - ii. Zkopíruje obsah pásky 1 a následně pásky 2 na pásku 3.
 - iii. Simuluje stroj M_1 na pásce 3.
 - iv. Jestliže páska 3 obsahuje 0, vymaže pásku 1, okopíruje pásku 2 na pásku 1 a zastaví. Jinak inkrementuje obsah na pásce 2, vymaže pásku 3 a pokračuje krokem (2).

□

6.9.3 Reprezentace Turingova stroje parciálně rekurzivními funkcemi

Abychom dokázali, že Turingova a Churchova teze jsou ekvivalentní, zbývá ukázat, že výpočetní síla Turingových strojů je omezena na možnost vyčíslovat parciálně rekurzivní funkce. K tomu účelu uvažujme Turingův stroj $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ a položme $b = |\Gamma|$. Nyní budeme obsah pásky interpretovat jako v opačném pořadí zapsané nezáporné celé číslo při základu b .

Například: Bude-li $\Gamma = \{x, y, \Delta\}$ a interpretujeme-li $x \approx 1, y \approx 2, \Delta \approx 0$, pak páska obsahující $\Delta y x \Delta \Delta y \Delta \Delta \dots$ je ekvivalentní s 02100200..., což po inverzi reprezentuje číslo ...00200120 při základu 3 a tedy 501.

S touto interpretací můžeme činnost každého Turingova stroje chápat jako výpočet funkce $f : \mathbb{N} \rightarrow \mathbb{N}$, která číslu odpovídajícímu počátečnímu obsahu pásky „přiřazuje“ číslo odpovídající obsahu pásky po zastavení stroje. Charakter funkce f specifikuje následující věta.

Věta 6.9.2 *Každý výpočetní proces prováděný Turingovým strojem je procesem vyčíslení nějaké parciálně rekurzivní funkce.*



Důkaz.

Budeme vycházet z právě zavedené interpretace činnosti Turingova stroje M , tj. pro každé $n \in \mathbb{N}$, $f(n)$ je definována obsahem pásky při zastavení stroje M . Dále provedeme zakódování stavů: $1 \approx q_0, 0 \approx q_F$, ostatní stavy čísla 2, 3, ..., $k-1$ za předpokladu $|Q| = k$. Tedy jak stavy, tak symboly mají přiřazené numerické hodnoty a můžeme definovat funkce, které sumarizují diagram přechodů stroje M' (M' je zakódovaný stroj M):

$$\begin{aligned}
mov(p, x) &= \begin{cases} 2 & \text{jestliže } \delta(p, x) = (*, R) \\ 1 & \text{jestliže } \delta(p, x) = (*, L) \\ 0 & \text{jinak} \end{cases} \\
sym(p, x) &= \begin{cases} y & \text{jestliže } \delta(p, x) = (*, y) \\ x & \text{jinak} \end{cases} \\
state(p, x) &= \begin{cases} q & \text{jestliže } \delta(p, x) = (q, *) \\ k & \text{jestliže } p = 0 \text{ nebo } \delta(p, x) \text{ není definována} \end{cases}
\end{aligned}$$

Funkce sym , mov a $state$ jsou tabulární funkce (totální) a jsou tedy primitivně rekurzivní.

Nyní uvažujme konfiguraci Turingova stroje M' ve tvaru trojice (w, p, n) , kde w je obsah pásky, p přítomný stav, n pozice hlavy ($n \geq 1$, nejlevější pozice je rovna 1).

Z informace obsažené v konfiguraci může být vypočítán symbol, který se nachází pod hlavou, primitivně rekurzivní funkcí $cursym$:

$$cursym(w, p, n) = quo(w, b^{n-1}) \dot{-} mult(b, quo(w, b^n))$$

Příklad 6.9.3 Nalezení n -té číslce řetězce $w = 1120121$ pro $n = 5$ a $b = 3$:

$$\begin{array}{ccc}
w=1120121 & \xrightarrow{\text{d\AA leno } b^{n-1}} & \begin{array}{c} 112 \\ 110 \end{array} \xrightarrow{\dot{-}} 2 \\
& & \uparrow \text{n\AA sobeno } b \\
w=1120121 & \xrightarrow{\text{d\AA leno } b^n} & 11
\end{array}
\quad
\begin{array}{l}
112 = quo(w, b^{n-1}) \\
110 = mult(b, quo(w, b^n))
\end{array}$$

Další funkce, které definujeme na množině konfigurací jsou:

$$\begin{aligned}
nexthead(w, p, n) &= n \dot{-} eq(mov(p, cursym(w, p, n)), 1) \\
&\quad + eq(mov(p, cursym(w, p, n)), 2)
\end{aligned}$$

určující příští pozici hlavy (0 indikuje abnormální posuv z 1. pozice vlevo)

$$nextstate(w, p, n) = state(p, cursym(w, p, n)) + mult(k, \neg nexthead(w, p, n))$$

(normálně je 2. sčítanec roven 0; při abnormálním posuvu tato funkce vyčíslí nelegální stav větší než $k - 1$) a konečně funkce

$$\begin{aligned}
nexttape(w, p, n) &= (w \dot{-} mult(b^{n-1}, cursym(w, p, n))) \\
&\quad + mult(b^{n-1}, sym(p, cursym(w, p, n)))
\end{aligned}$$

kteřá vyčísluje celé číslo reprezentující nový obsah pásky, po provedení přechodu z konfigurace (w, p, n) . Kombinací tří předchozích funkcí dostaneme funkci $step$, která modeluje 1. krok Turingova stroje, tj. přechod do nové konfigurace:

$$step = nexttape \times nextstate \times nexthead$$

Nyní definujme funkci $run : \mathbb{N}^4 \rightarrow \mathbb{N}^3$; $run(w, p, n, t)$ realizující t přechodů z konfigurace (w, p, n) . Opět uijeme primitivní rekurze:

$$\begin{aligned}
run(w, p, n, 0) &= (w, p, n) \\
run(w, p, n, t+1) &= step(run(w, p, n, t))
\end{aligned}$$

Výsledná funkce vyčíslovaná strojem M' (při vstupu w) je hodnota pásky po dosažení koncového stavu (stavu 0). Počet požadovaných kroků stroje M je dán funkcí *stoptime*

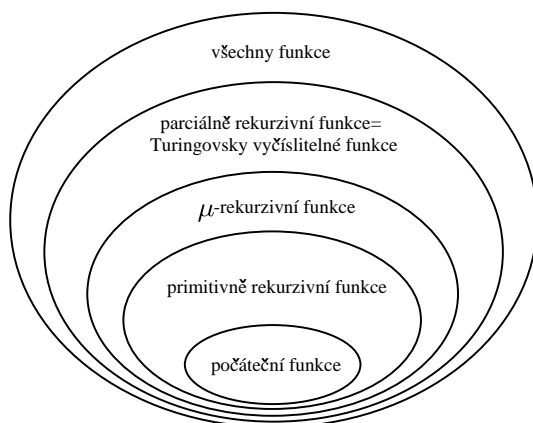
$$\text{stoptime}(w) = \mu t[\pi_2^3(\text{run}(w, 1, 1, t)) = 0]$$

Na závěr tedy, je-li $f : \mathbb{N} \rightarrow \mathbb{N}$ parciální funkce počítaná strojem M , pak

$$f(w) = \pi_1^3(\text{run}(w, 1, 1, \text{stoptime}(w)))$$

z její konstrukce plyne, že f je parciálně rekurzivní funkce. □

Shrňme v obrázku získané informace:



Existuje nespočetně mnoho formálních jazyků, které nelze popsat Turingovým strojem a leží tudíž mimo třídu rekurzivně vyčíslitelných jazyků. K těmto jazykům patří jazyky, které jsou striktně nerozhodnutelné jazyky a komplementy částečně rozhodnutelných jazyků. Mnohé z nich popisují problémy, které tudíž nelze algoritmicky řešit. Jejich obecnou, často využívanou reprezentací pro redukce je problém zastavení Turingova stroje a Postův korespondenční problém. Mezi řadou alternativních, Turingovým strojem ekvivalentních formálních systémů jsou důležitým prostředkem popisu a analýzy parciální rekurzivní funkce, které mají velmi úzký vztah k funkcionálnímu programování.

Σ

6.10 Cvičení

Příklad 6.10.1

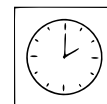
1. Definujte, kdy je problém rozhodnutelný, nerozhodnutelný a částečně rozhodnutelný.
2. Uveďte problémy, které jsou rozhodnutelné, částečně rozhodnutelné a problémy které nejsou ani částečně rozhodnutelné.
3. Definujte Postův korespondenční problém a rozhodněte, zda je rozhodnutelný nebo aspoň částečně rozhodnutelný. Dále uveďte příklad Postova systému, který má řešení resp. nemá řešení.

?

4. *Definujte pojem redukce a vysvětlete na příkladu jeho použití.*
5. *Uvedte 1. a 2. Riceova větu a jejich možné použití.*
6. *Uvedte příklady dalších výpočetních modelů, které mají ekvivalentní výpočtovou sílu jako Turingovy stroje.*
7. *Vysvětlete pojmy počáteční funkce, primitivně rekurzivní funkce a parciálně vyčíslitelná funkce.*
8. *Existuje primitivně rekurzivní funkce, která není totální. Své rozhodnutí zdůvodněte.*
9. *Uvedte příklady funkcí, které vznikly kombinací, kompozicí, primitivní rekurzí a minimalizací.*
10. *Uvedte vztah parciálně vyčíslitelných funkcí a Turingových strojů.*

Kapitola 7

Složitost



8:00



Cílem této kapitoly je pochopení aplikace Turingových strojů pro popis a klasifikaci časové a paměťové složitosti problémů a vymezení složitostních tříd těsně spjatých s otázkami časové a paměťové efektivity výpočtů a reálnosti realizovat řešení problémů na počítačích.

7.1 Základní pojmy složitosti

7.1.1 Složitost algoritmů

Základní teoretický přístup a volba Turingova stroje jako výpočetního modelu vychází z Church-Turingovy teze:

Každý algoritmus je implementovatelný jistým TS.

Zavedení TS nám umožňuje klasifikovat problémy (resp. funkce) do dvou tříd:

1. problémy, jež nejsou *algoritmicky ani částečně rozhodnutelné* (resp. funkce algoritmicky nevyčíslitelné)
2. problémy *algoritmicky alespoň částečně rozhodnutelné* (resp. funkce algoritmicky vyčíslitelné).

Nyní se budeme zabývat třídou algoritmicky (částečně) rozhodnutelných problémů (vyčíslitelných funkcí) v souvislosti s otázkou *složitosti* jejich rozhodování (vyčíslování).

Analýzu složitosti algoritmu budeme chápat jako analýzu složitosti výpočtů příslušného TS, jejímž cílem je *vyjádřit (kvantifikovat) požadované zdroje (čas, prostor) jako funkci závislé na délce vstupního řetězce*.

7.1.2 Různé případy při analýze složitosti

Nejdříve je třeba určit cenu jednoho konkrétního výpočtu konkrétního TS (paměť, čas výpočtu).

Mějme TS M . Složitost bude tedy funkcí $Compl_M : \mathbb{N} \rightarrow \mathbb{N}^1$. Je více možností její definice podle toho, který z možných výpočtů na vstupu příslušné délky (a jeho cenu) si vybereme jako určující:

1. analýza *složitosti nejhoršího případu*,
2. analýza *složitosti nejlepšího případu*,
3. analýza *složitosti průměrného případu*.

¹pro dané n vrací míru složitosti výpočtu M na řetězcích délky n

Průměrná složitost algoritmu je definována následovně: Jestliže algoritmus (TS) vede k m různým výpočtům (případům) s cenou c_1, c_2, \dots, c_m , jež nastávají s pravděpodobnostmi p_1, p_2, \dots, p_m , pak *průměrná složitost algoritmu* je dána jako $\sum_{i=1}^n p_i c_i$.

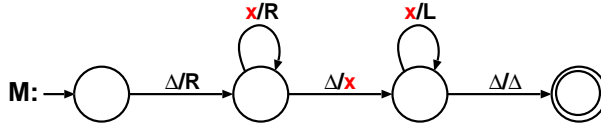
Obvykle (alespoň na teoretické úrovni) se věnuje *největší pozornost* složitosti nejhoršího případu.

7.1.3 Složitost výpočtů TS

Na cenu výpočtu se budeme dívat z hledisek času výpočtu a paměťové náročnosti.

- *Časová složitost* – počet kroků (přechodů) TS provedený od počátku do konce výpočtu.
- *Prostorová (paměťová) složitost* – počet „buněk“ pásky TS požadovaný pro daný výpočet.

Příklad 7.1.1 Uvažme následující TS M :



Pro vstup $w = \Delta \textcolor{red}{xxx} \Delta \Delta \dots$ je:

- časová složitost výpočtu M na w rovna 10,
- prostorová složitost výpočtu M na w rovna 5.

Lemma 7.1.1 Je-li časová složitost výpočtu prováděného TS rovna n , pak prostorová složitost tohoto výpočtu není větší než $n + 1$.

Důkaz. Tvrzení je jednoduchou implikací plynoucí z definice časové a prostorové složitosti. \square

Nyní budeme definovat funkce udávající složitost TS v závislosti na čase výpočtu (počtu kroků TS) a použitém prostoru (počtu použitých políček pásky). Bude se jednat o analýzu nejhoršího případu:

Definice 7.1.1 Řekneme, že k -páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v čase $T_M : \mathbb{N} \rightarrow \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. může přijmout) každé $w \in L$ v nanejvýš $T_M(|w|)$ krocích.

Definice 7.1.2 Řekneme, že k -páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v prostoru $S_M : \mathbb{N} \rightarrow \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. může přijmout) každé $w \in L$ při použití nanejvýš $S_M(|w|)$ buněk pásky.

Poznámka 7.1.1 Do prostorové složitosti nepočítáme buňky pásky, na nichž je zapsán vstup, pokud nebudou během výpočtu přepsány.

Zcela analogicky můžeme definovat *vyčíslování určité funkce daným TS* v určitém čase, resp. prostoru.

$x + y$

DEF

7.1.4 Složitost a cena atomických operací

Pro případ TS předpokládáme, že každý jeho výpočetní krok je stejně náročný. Používáme tedy takzvané *uniformní cenové kritérium*, kdy každé operaci přiřadíme *stejnou cenu*.

V jiném výpočetním prostředí, než jsou TS, tomu tak ale být nemusí. Používá se ale např. také tzv. *logaritmické cenové kritérium*, kdy operaci manipulující operand o velikosti i , $i > 0$, přiřadíme cenu $\lfloor \lg i \rfloor + 1$. Zohledňujeme to, že s rostoucí velikostí operandů roste cena operací – logaritmus odráží růst velikosti s ohledem na binární kódování (v n bitech zakódujeme 2^n hodnot).

Analýza složitosti za takových předpokladů není zcela přesná, důležité ale obvykle je to, aby se *zachovala informace o tom, jak rychle roste čas/prostor potřebný k výpočtu v závislosti na velikosti vstupu*.²

Význam srovnávání rychlosti růstu složitosti výpočtů si snad nejlépe ilustrujeme na příkladu:

Příklad 7.1.2 Srovnání polynomiální (přesněji kvadratické – n^2) a exponenciální (2^n) časové složitosti:

délka vstupu n	časová složitost $c_1 \cdot n^2$, $c_1 = 10^{-6}$	časová složitost $c_2 \cdot 2^n$, $c_2 = 9.766 \cdot 10^{-8}$
10	0.0001 s	0.0001 s
20	0.0004 s	0.1024 s
30	0.0009 s	1.75 min
40	0.0016 s	1.24 dne
50	0.0025 s	3.48 roku
60	0.0036 s	35.68 století
70	0.0049 s	3.65 mil. roků

$$x + y$$

7.1.5 Složitost výpočtů na TS a v jiných prostředích

Pro rozumná cenová kritéria se ukazuje, že výpočetní složitost je pro různé modely výpočtu blízké běžným počítačům (RAM, RASP stroje) *polynomiálně vázaná* se složitostí výpočtu na TS (viz dále) a tudíž *složitost výpočtu na TS není „příliš“ rozdílná oproti výpočtům na běžných počítačích*.

- *RAM stroje* mají paměť s náhodným přístupem (jedna buňka obsahuje libovolné přirozené číslo), instrukce typu LOAD, STORE, ADD, SUB, MULT, DIV (akumulátor a konstanta/přímá adresa/nepřímá adresa), vstup/výstup, nepodmíněný skok a podmíněný skok (test akumulátoru na nulu), HALT. U RAM stroje je program součástí řízení stroje (okamžitě dostupný), u *RASP* je uložen v paměti stejně jako operandy.
- Funkce $f_1(n), f_2(n) : \mathbb{N} \rightarrow \mathbb{N}$ jsou *polynomiálně vázané*, existují-li polynomy $p_1(x)$ a $p_2(x)$ takové, že pro všechny hodnoty n je $f_1(n) \leq p_1(f_2(n))$ a $f_2(n) \leq p_2(f_1(n))$.
- *Logaritmické cenové kritérium* se uplatní, uvažujeme-li možnost násobení dvou čísel. Jednoduchým cyklem typu $A_{i+1} = A_i * A_i$ ($A_0 = 2$) jsme schopni počítat 2^{2^n} , což TS s omezenou abecedou není schopen provést v polynomiálním čase (pro uložení/načtení potřebuje projít 2^n buněk při použití binárního kódování).

Ilustrujme si nyní určení složitosti v prostředí mimo TS:

Příklad 7.1.3 Uvažme následující implementaci porovnávání řetězců.

```
int str_cmp (int n, string a, string b) {
    int i;

    i = 0;
    while (i < n) {
        if (a[i] != b[i]) break;
        i++;
    }

    return (i == n);
}
```

$x + y$

- Pro určení složitosti aplikujme uniformní cenové kritérium např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- Složitost nejhoršího případu lze pak snadno určit jako $4n + 3$:
 - cyklus má 4 kroky, provede se n krát, tj. $4n$ kroků,
 - tělo funkce má 3 kroky (včetně testu ukončujícího cyklus).

Příklad 7.1.4 Uvažme následující implementaci řazení metodou insert-sort.

```
void insertsort(int n, int a[]) {
    int i, j, value;
    for (i=0; i < n; i++) {
        value = a[i];
        j = i - 1;
        while ((j >= 0) && (a[j] > value)) {
            a[j+1] = a[j];
            j = j - 1;
        }
        a[j+1] = value;
    }
}
```

$x + y$

- Pro určení složitosti aplikujme uniformní cenové kritérium např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- Složitost lze pak určit jako $2n^2 + 4n + 1$:
 - vnitřní cyklus má 4 kroky, provede se $0, 1, \dots, n - 1$ krát, tj. $4(0 + 1 + \dots + n - 1) = 4 \frac{n}{2}(0 + n - 1) = 2n^2 - 2n$ kroků,
 - vnější cyklus má mimo vnitřní cyklus 6 kroků (včetně testu ukončujícího vnitřní cyklus) a provede se n krát, tj. $6n$ kroků,
 - jeden krok připadá na ukončení vnějšího cyklu.

²Algoritmus A_1 , jehož nároky rostou pomaleji než u jiného algoritmu A_2 , nemusí být výhodnější než A_2 pro řešení malých instancí.

7.1.6 Asymptotická omezení složitosti

Přesná informace o složitosti algoritmu je pro nás většinou zbytečná, o skutečném čase (prostoru) výpočtu vypovídá málo:

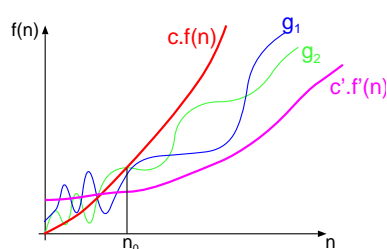
- různé aditivní a multiplikativní konstanty vzniknou velmi snadno „drobnými“ úpravami uvažovaných algoritmů,
- např. srovnávání dvou řetězců je možné donekonečna zrychlovat tak, že budeme porovnávat současně 2, 3, 4, ... za sebou jdoucích znaků,
- tyto úpravy ovšem nemají zásadní vliv na rychlost nárůstu časové složitosti (ve výše uvedeném případě nárůst zůstane kvadratický),
- navíc při analýze složitosti mimo prostředí TS se dopouštíme jisté nepřesnosti již zavedením různých cenových kritérií.

Zajímá nás tedy jen „důležitá“ část informace o složitosti (porovnání řetězců – čas roste lineárně, insert-sort – čas roste kvadraticky). Proto se často složitost popisuje pomocí tzv. *asymptotických odhadů složitosti*:

Definice 7.1.3 Necht \mathcal{F} je množina funkcí $f : \mathbb{N} \rightarrow \mathbb{N}$. Pro danou funkci $f \in \mathcal{F}$ definujeme množiny funkcí $O(f(n))$, $\Omega(f(n))$ a $\Theta(f(n))$ takto:

- Asymptotické horní omezení funkce $f(n)$ je množina $O(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq g(n) \leq c \cdot f(n)\}$.
- Asymptotické dolní omezení funkce $f(n)$ je množina $\Omega(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c \cdot f(n) \leq g(n)\}$.
- Asymptotické oboustranné omezení funkce $f(n)$ je množina $\Theta(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c_1, c_2 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)\}$.

DEF



Příklad 7.1.5 S využitím asymptotických odhadů složitosti můžeme říci, že složitost našeho srovnání řetězců patří do $O(n)$ a složitost insert-sort do $O(n^2)$.

Příklad 7.1.6

- Ukážeme, že platí $2 \cdot n \in O(n)$.
Dle definice zvolme $c = 3$ a $n_0 = 1$. Pak skutečně platí, že $\forall n \geq 1$ je $2 \cdot n \leq 3 \cdot n$
- Ukážeme, že platí $n^2 \notin O(n)$.
Důkaz provedeme sporem. Necht dle definice $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}$ takové, že $\forall n \geq n_0$ platí $n^2 \leq c \cdot n$. Pak také ale platí, že $\forall n \geq n_0$ je $n \leq c$, což je spor.

 $x + y$

Poznámka 7.1.2 Používá se i zobecnění O -notace, kdy se operátor O může vyskytovat na různých místech aritmetických výrazů. Například výraz $2^{O(f(n))}$ označuje množinu funkcí $2^{g(n)}$, kde $g(n) \in O(f(n))$. Všimněme si přitom, že $2^{O(f(n))} \neq O(2^{f(n)})$:

- $2^{O(f(n))}$ je množina všech funkcí, jejichž hodnota zůstává pro všechna n větší než nějaké n_0 pod $2^{a \cdot f(n) + b}$, neboli vlastně pod $c^{\frac{a \cdot f(n) + b}{d}}$, kde a, b, c, d jsou libovolné konstanty takové, že $c = 2^d$, tj. jedná se o exponenciály s libovolným základem.
- Naproti tomu $O(2^{f(n)})$ zahrnuje jen funkce, jejichž hodnota zůstává pro všechna n větší než nějaké n_0 pod $a \cdot 2^{f(n)} + b$ pro nějaké konstanty a, b – základem exponenciály je zde vždy 2.

Podobně třeba výraz $n^2 + O(n)$ označuje množinu polynomů $\{n^2 + g(n) \mid g(n) \in O(n)\}$. Lze užít i výrazy jako $O(f(n))^{O(g(n))} \cdot O(h(n))$, kde se O vyskytuje vícekrát. V uvedeném příkladu se jedná konkrétně o množinu funkcí $k(n)^{l(n)} \cdot m(n)$, kde $k(n) \in O(f(n))$, $l(n) \in O(g(n))$ a $m(n) \in O(h(n))$.

Příklad 7.1.7 Rozhodněte a zdůvodněte, zda platí:

- $n \cdot \log(n) \in O(n^2)$
- $n \cdot \log(n) \in O(n)$
- $3^n \in 2^{O(n)}$
- $6 \cdot n^3 + 50 \cdot n^2 + 6 \in O(n^3 - 8 \cdot n^2 - n - 5)$



7.2 Třídy složitosti

7.2.1 Složitost problémů

Přejdeme nyní od složitosti konkrétních algoritmů (Turingových strojů) ke složitosti problémů.

Třídy složitosti zavádíme jako prostředek ke kategorizaci (vytvoření hierarchie) problémů dle jejich složitosti, tedy dle toho, jak dalece efektivní algoritmy můžeme nahradit pro jejich rozhodování (u funkcí můžeme analogicky mluvit o složitosti jejich vyčíslování).

Podobně jako u určování typu jazyka se budeme snažit zařadit problém do co nejnížší třídy složitosti – tedy určit složitost problému jako složitost jeho nejlepšího možného řešení.

Poznámka 7.2.1

Omezením této snahy je to, že (jak uvidíme) existují problémy, jejichž řešení je možné do nekonečna významně zrychlovat.

Zařazení různých problémů do téže třídy složitosti může odhalit různé vnitřní podobnosti těchto problémů a může umožnit řešení jednoho převodem na druhý (a pragmatické využití např. různých již vyvinutých nástrojů).

7.2.2 Třídy složitosti

Definice 7.2.1 Mějme dány funkce $t, s : \mathbb{N} \rightarrow \mathbb{N}$ a necht T_M , resp. S_M , značí časovou, resp. prostorovou, složitost TS M . Definujeme následující časové a prostorové třídy složitosti deterministických a nedeterministických TS:

- $DTime[t(n)] = \{L \mid \exists k\text{-páskový DTS } M : L = L(M) \text{ a } T_M \in O(t(n))\}$.
- $NTime[t(n)] = \{L \mid \exists k\text{-páskový NTS } M : L = L(M) \text{ a } T_M \in O(t(n))\}$.
- $DSpace[s(n)] = \{L \mid \exists k\text{-páskový DTS } M : L = L(M) \text{ a } S_M \in O(s(n))\}$.
- $NSpace[s(n)] = \{L \mid \exists k\text{-páskový NTS } M : L = L(M) \text{ a } S_M \in O(s(n))\}$.

Definici tříd složitosti pak přímočaře zobecníme tak, aby mohly být založeny na množině funkcí, nejen na jedné konkrétní funkci.

Poznámka 7.2.2 Dále ukážeme, že použití více pásek přináší jen polynomiální zrychlení. Na druhou stranu ukážeme, že zatímco nedeterminismus nepřináší nic podstatného z hlediska vyčíslitelnosti, může přinášet mnoho z hlediska složitosti.

Poznámka 7.2.3 Výše uvedené definice tříd jsou třídami výpočetní složitosti rozhodovacích problémů. Studují se však také např. třídy tzv. optimalizačních problémů, u kterých se neptáme, zda daná instance je či není řešením, ale hledáme v určitém smyslu optimální řešení (např. se neptáme, zda daný graf má kliku určité velikosti, ale ptáme se, klika jaké největší velikosti v grafu existuje). Problémy pak mohou být také dále klasifikovány do tříd dle aproximovatelnosti jejich optimálního řešení, existují rovněž pravděpodobnostní třídy složitosti (pro řešení problémů pravděpodobnostními algoritmy) atd (viz např. [9]). Tyto třídy jsou však mimo rozsah tohoto textu.

7.2.3 Časově/prostorově konstruovatelné funkce

Třídy složitosti obvykle budujeme nad tzv. časově/prostorově konstruovatelnými funkcemi:

- Důvodem zavedení časově/prostorově konstruovatelných funkcí je dosáhnout intuitivní hierarchické struktury tříd složitosti – např. odlišení tříd $f(n)$ a $2^{f(n)}$, což, jak uvidíme, pro třídy založené na obecných rekurzivních funkcích nelze.

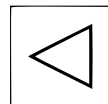
Definice 7.2.2 Funkci $t : \mathbb{N} \rightarrow \mathbb{N}$ nazveme časově konstruovatelnou (time constructible), jestliže existuje vícepáskový TS, jenž pro libovolný vstup w zastaví po přesně $t(|w|)$ krocích.

Definice 7.2.3 Funkci $s : \mathbb{N} \rightarrow \mathbb{N}$ nazveme prostorově konstruovatelnou (space constructible), jestliže existuje vícepáskový TS, jenž pro libovolný vstup w zastaví s využitím přesně $s(|w|)$ buněk pásky.

Příklad 7.2.1 Uveďme některé

- časově konstruovatelné funkce: $f(n) = n \log(n)$, $f(n) = n\sqrt{n}$,

DEF



DEF

x + y

- časově nekonstruovatelné funkce: $f(n) = c$, $f(n) = \log(n)$, $f(n) = n^3$,
- prostorově konstruovatelné funkce: $f(n) = \log(n)$, $f(n) = n^2$,
- prostorově nekonstruovatelné funkce: $f(n) = c$, $f(n) = \log \log n$.

Poznámka 7.2.4 Pokud je jazyk L nad Σ přijímán strojem v čase/prostoru omezeném časově/prostorově konstruovatelnou funkcí, pak je také přijímán strojem, který pro každé $w \in \Sigma^*$ vždy zastaví:

- U časového omezení $t(n)$ si stačí předem spočítat, jaký je potřebný počet kroků a zastavit po jeho vyčerpání.
- U prostorového omezení spočteme z $s(n)$, $|Q|$, $|\Delta|$ maximální počet konfigurací, které můžeme vidět a z toho také plyne maximální počet kroků, které můžeme udělat, aniž bychom cyklili.

7.2.4 Nejběžněji užívané třídy složitosti

Deterministický/nedeterministický polynomiální čas:

$$\mathbf{P} = \bigcup_{k=0}^{\infty} DTime(n^k) \qquad \mathbf{NP} = \bigcup_{k=0}^{\infty} NTime(n^k)$$

Deterministický/nedeterministický polynomiální prostor:

$$\mathbf{PSPACE} = \bigcup_{k=0}^{\infty} DSpace(n^k) \qquad \equiv \qquad \mathbf{NPSPACE} = \bigcup_{k=0}^{\infty} NSpace(n^k)$$

Poznámka 7.2.5 \mathbf{P} je zvlášť důležitou třídou. Vymezuje všechny prakticky dobře řešitelné problémy.

Poznámka 7.2.6 Problémy ze třídy \mathbf{PSPACE} se často ve skutečnosti neřeší v polynomiálním prostoru – zvyšují se prostorové nároky výměnou za alespoň časové (např. z $O(2^{n^2})$ na $O(2^n)$) snížení časových nároků.

7.2.5 Třídy pod a nad polynomiální složitostí

Deterministický/nedeterministický logaritmický prostor:

$$\mathbf{LOGSPACE} = \bigcup_{k=0}^{\infty} DSpace(k \lg n) \qquad \mathbf{NLOGSPACE} = \bigcup_{k=0}^{\infty} NSpace(k \lg n)$$

Deterministický/nedeterministický exponenciální čas:

$$\mathbf{EXP} = \bigcup_{k=0}^{\infty} DTime(2^{n^k}) \qquad \mathbf{NEXP} = \bigcup_{k=0}^{\infty} NTime(2^{n^k})$$

Deterministický/nedeterministický exponenciální prostor:

$$\mathbf{EXPSPACE} = \bigcup_{k=0}^{\infty} DSpace(2^{n^k}) \qquad \equiv \qquad \mathbf{NEXPSPACE} = \bigcup_{k=0}^{\infty} NSpace(2^{n^k})$$

³Funkce $f(n) = n$ není časově zkonstruovatelná, protože úvodní blank nepočítáme jako součást vstupu (někdy se v literatuře můžeme setkat i s jinými přístupy).

DEF

7.2.6 Třídy nad exponenciální složitostí

Det./nedet. k-exponenciální čas/prostor založený na věži exponenciál $2^{2^{\vdots^2}}$ o výšce k :

DEF

$$\begin{aligned} \mathbf{k-EXP} &= \bigcup_{l=0}^{\infty} DTime(2^{2^{\vdots^{2^l}}}) & \mathbf{k-NEXP} &= \bigcup_{l=0}^{\infty} NTime(2^{2^{\vdots^{2^l}}}) \\ \mathbf{k-EXPSPACE} &= \bigcup_{l=0}^{\infty} DSpace(2^{2^{\vdots^{2^l}}}) \equiv \mathbf{k-NEXPSPACE} = \bigcup_{l=0}^{\infty} NSpace(2^{2^{\vdots^{2^l}}}) \\ \mathbf{ELEMENTARY} &= \bigcup_{k=0}^{\infty} \mathbf{k-EXP} \end{aligned}$$

7.2.7 Vrchol hierarchie tříd složitosti

Na *vrcholu hierarchie tříd složitosti* se pak hovoří o obecných třídách jazyků (funkcí), se kterými jsme se již setkali:

- *třída primitivně-rekurzivních funkcí* **PR** (implementovatelných pomocí zanořených cyklů s pevným počtem opakování – `for i=... to ...`),
- *třída μ -rekurzivních funkcí* (rekurzivních jazyků) **R** (implementovatelných pomocí cyklů s předem neurčeným počtem opakování – `while ...`) a
- *třída rekurzivně vyčíslitelných funkcí* (rekurzivně vyčíslitelných jazyků) **RE**.

7.3 Vlastnosti tříd složitosti

Prozkoumáme nyní vztahy a vlastnosti zadaných složitostních tříd.

7.3.1 Vícepáskové stroje

Zavedení vícepáskového stroje nemělo vliv na vyčíslitelnost. Ani v teorii složitosti nepřinese mnoho nového. Ukážeme, že ke každému vícepáskovému stroji existuje ekvivalentní jednopáskový, jehož výpočty jsou maximálně polynomiálně časově náročnější.

Věta 7.3.1 Je-li jazyk L přijímán nějakým k -páskovým DTS M_k v čase $t(n)$, pak je také přijímán nějakým jednopáskovým DTS M_1 v čase $O(t(n)^2)$.

Důkaz. (idea) Ukážeme, jak může M_k simulovat M_1 v uvedeném čase:

- Na pásku stroje M_1 zapíšeme zřetězení obsahu pásek stroje M_k . Dále si stroj M_1 musí uchovávat informaci o aktuální pozici hlav stroje M_k (například symboly na pásce které jsou čteny hlavami stroje M_k budou podtrženy) a informaci o konci jednotlivých pásek stroje M_k (speciální oddělovače).

- V první fázi simulace výpočtu stroje M_k strojem M_1 si stroj M_1 upraví svoji pásku do požadovaného tvaru.
- K simulaci jednoho kroku stroje M_k musí stroj M_1 dvakrát přechít svoji pásku. V prvním průchodu shromáždí informace o symbolech, které byly čteny jednotlivými hlavami stroje M_k . V druhém průchodu patřičně upraví svoji pásku.
- Problém nastává, pokud při simulaci stroj M_k zapsal nějaký symbol na do teď prázdné políčko své pásky. V tom případě musí stroj M_1 posunout zbytek řetězce na pásce o jedno políčko doprava, aby uvolnil potřebné místo pro nový symbol.
- Stroj M_k nemůže mít na žádné pásce víc než $t(n)$ symbolů kde n je délka vstupního řetězce. Tudíž stroj M_1 nemůže mít víc než $k \cdot t(n)$ symbolů. Simulace jednoho kroku stroje M_k zabere $4 \cdot k \cdot (t(n))$ kroků (dva průchody páskou tam a zpět) plus nanejvýš $k \cdot (t(n))$ kroků (vytvoření prázdného políčka). Dohromady tedy počet kroků stroje M_1 bude v $O(k^2 \cdot t(n)^2)$. Jelikož k je konstanta nezávislá na vstupu, celková časová složitost stroje M_1 je v $O(t(n)^2)$.

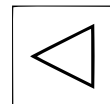
□

Příklad 7.3.1 Ukažte, že $L = \{w \in \Sigma^* \mid w \text{ je palindrom}\} \in P$.



7.3.2 Determinismus a nedeterminismus

Zatímco z hlediska vyčíslitelnosti nepřináší zavedení nedeterminismu nic nového, pro složitost je situace jiná. Ačkoli skutečné vztahy deterministických a nedeterministických tříd nebyly dosud dokázány, zdá se, že nedeterminismus výrazně snižuje zejména časové nároky výpočtu. Jeho sílu dobře ilustruje například následující pohled na výpočet nedeterministického stroje:



- Zatímco klasický deterministický stroj krok za krokem výsledek počítá, nedeterministický stroj je prostě může uhodnout, zapsat a po té jen ověřit jeho správnost.

Nedeterministický TS dokážeme simulovat strojem deterministickým, avšak jen za cenu exponenciálního nárůstu času:

Věta 7.3.2 Je-li jazyk L přijímán nějakým NTS M_n v čase $t(n)$, pak je také přijímán nějakým DTS M_d v čase $2^{O(t(n))}$.

Důkaz. (idea) Ukážeme, jak může M_d simulovat M_n v uvedeném čase:

- Očíslujeme si přechody M_n jako $1, 2, \dots, k$.
- M_d bude postupně simulovat veškeré možné posloupnosti přechodů M_n (obsah vstupní pásky si uloží na pomocnou pásku, aby ho mohl vždy obnovit; na jinou pásku si vygeneruje posloupnost přechodů z $\{1, 2, \dots, k\}^*$ a tu simuluje).

- Vzhledem k *možnosti nekonečných výpočtů* M_n nelze procházet jeho možné výpočty do hloubky – budeme-li je ale *procházet do šířky* (tj. nejdřív všechny řetězce z $\{1, 2, \dots, k\}^*$ délky 1, pak 2, pak 3, ...), určitě nalezneme nejkratší přijímající posloupnost přechodů pro M_n , existuje-li.
- Takto projdeme nanejvýš $O(k^{t(n)})$ cest, simulace každé z nich je v $O(t(n))$ a tudíž celkem využijeme nanejvýš čas $O(k^{t(n)})O(t(n)) = 2^{O(t(n))}$.

□

Příklad 7.3.2 Ukažte že $L = \{\phi \mid \phi \text{ je splnitelné formule v CNF}\} \in NP$

Vztah determinismu a nedeterminismu je jedním z nejslavnějších otevřených problémů informatiky. Zejména o problému ekvivalence polynomiálních časových tříd $P = NP$ bylo řečeno mnoho. Jeho řešení se však zatím zdá být za hranicemi dnešních možností. Komplikovanost problému naznačují například výsledky, které říkají, že pokud $P=NP$, nemůže tento vztah být dokázán simulací⁴ a naopak, pokud $P \neq NP$, nemůže být tento výsledek dokázán diagonalizací.

Zatímco se zdá, že nedeterminismus přináší značnou výhodu z hlediska časové složitosti výpočtů, v případě prostorové složitosti je situace jiná:

Věta 7.3.3 (Savitchův teorém) $NSpace[s(n)] \subseteq DSpace[s^2(n)]$ pro každou prostorově konstruovatelnou funkci $s(n) \geq \lg n$.

Důkaz. Uvažme NTS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ rozhodující $L(M)$ v prostoru $s(n)$:

- Existuje $k \in \mathbb{N}$ závislé jen na $|Q|$ a $|\Gamma|$ takové, že pro libovolný vstup w , M projde nanejvýš $k^{s(n)}$ konfigurací o délce max. $s(n)$.
- To implikuje, že M provede pro w nanejvýš $k^{s(n)} = 2^{s(n) \lg k}$ kroků.
- Pomocí DTS můžeme snadno implementovat proceduru $test(c, c', i)$, která otestuje, zda je možné v M dojít z konfigurace c do c' v 2^i krocích:

procedure $test(c, c', i)$

if $(i = 0)$ **then return** $((c = c') \vee (c \vdash_M c'))$

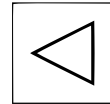
else for all configurations c'' such that $|c''| \leq s(n)$ **do**

if $(test(c, c'', i-1) \wedge test(c'', c', i-1))$ **then return true**

return false

- Všimněme si, že rekurzivním vyvoláváním $test$ vzniká strom o výšce i simulující posloupností svých listů posloupnost 2^i výpočetních kroků.
- Nyní k deterministické simulaci M postačí projít všechny akceptující konfigurace c_F takové, že $|c_F| \leq s(n)$, a ověřit, zda $test(c_0, c_F, \lceil s(n) \lg k \rceil)$, kde c_0 je počáteční konfigurace.
- Každé vyvolání $test$ zabere prostor $O(s(n))$, hloubka rekurze je $\lceil s(n) \lg k \rceil = O(s(n))$ a tedy celkově deterministicky simulujeme M v prostoru $O(s^2(n))$.
- Dodejme, že $s(n)$ může být zkonstruováno v prostoru $O(s(n))$ (jedná se o prostorově konstruovatelnou funkci) a tudíž neovlivňuje výše uvedené úvahy.

⁴ nemůže existovat polynomiálně časově ohraničený DTS simulující každý polynomiální NTS



□

Důsledkem Savitchova teorému jsou již dříve uvedené rovnosti:

- $\text{PSPACE} \equiv \text{NPSpace}$,
- $\text{k-EXPSPACE} \equiv \text{k-NEXPSPACE}$.

7.3.3 Prostor kontra čas

Intuitivně můžeme říci, že *zatímco prostor může růst relativně pomalu, čas může růst výrazně rychleji*, neboť můžeme opakovaně procházet týmiž buňkami pásky – opačně tomu být zřejmě nemůže (nemá smysl mít nevyužitý prostor).

Věta 7.3.4 $\text{NSpace}[t(n)] \subseteq \text{DTime}[O(1)^{t(n)}]$ pro každou časově konstruovatelnou funkci $t(n) \geq \lg n$.

Důkaz. Dá se použít do jisté míry podobná konstrukce jako u Savitchova teorému – blíže viz literatura. □

7.3.4 Uzavřenost vůči doplňku

Doplňkem třídy rozumíme třídu jazyků, které jsou doplňkem jazyků dané třídy. Tedy označíme-li doplněk třídy \mathcal{C} jako $\text{co-}\mathcal{C}$, pak $L \in \mathcal{C} \Leftrightarrow \bar{L} \in \text{co-}\mathcal{C}$. U rozhodování problémů toto znamená rozhodování komplementárního problému (prázdnot x neprázdnot apod.).

Prostorové třídy jsou obvykle uzavřeny vůči doplňku:

Věta 7.3.5 Jestliže $s(n) \geq \lg n$, pak $\text{NSpace}(s(n)) = \text{co-NSpace}(s(n))$.

Důkaz. Jedná se o teorém Immermana a Szelepcsenyiho – důkaz viz literatura. □

Pro *časové třídy* je situace jiná:

- Některé třídy jako **P** či **EXP** jsou uzavřeny vůči doplňku.
- U jiných významných tříd zůstává otázka uzavřenosti vůči doplňku otevřená. Proto má smysl hovořit např. i o třídách jako:
 - **co-NP** či
 - **co-NEXP**.

Věta 7.3.6 Třída **P** je uzavřena vůči doplňku.

Důkaz. (idea) Základem je to, že ukážeme, že jestliže jazyk L nad Σ může být přijat DTS M v polynomiálním čase, pak také existuje DTS M' , který L rozhoduje v polynomiálním čase, tj. $L = L(M')$ a existuje $k \in \mathbb{N}$ takové, že pro každé $w \in \Sigma^*$ M' zastaví v čase $O(|w|^k)$:

- M' na začátku výpočtu určí délku vstupu w a vypočte $p(|w|)$, kde $p(n)$ je polynom určující složitost přijímání strojem M . Na speciální dodatečnou pásku uloží $p(|w|)$ symbolů.

- Následně M' simuluje M , přičemž za každý krok umaže jeden symbol z dodatečné pásky. Pokud odebere z této pásky všechny symboly a M by mezitím nepřijal, M' abnormálně ukončí výpočet (odmítne).
- M' evidentně přijme všechny řetězce, které přijme M na to mu stačí $p(n)$ simulovaných kroků a nepřijme všechny řetězce, které by M nepřijal – pokud M nepřijme v $p(n)$ krocích, nepřijme vůbec. M' však vždy zastaví v $O(p(n))$ krocích.

□

7.3.5 Ostrost hierarchie tříd

Z dosavadního můžeme shrnout, že platí následující:

- $\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{P} \subseteq \text{NP}$
- $\text{NP} \subseteq \text{PSPACE} = \text{NPSpace} \subseteq \text{EXP} \subseteq \text{NEXP}$
- $\text{NEXP} \subseteq \text{EXPSPACE} = \text{NEXPSPACE} \subseteq \text{2-EXP} \subseteq \text{2-NEXP} \subseteq \dots$
- $\dots \subset \text{ELEMENTARY} \subset \text{PR} \subset \text{R} \subset \text{RE}^5$

Řada otázek ostrosti uvedených vztahů pak zůstává otevřená, nicméně z tzv. teorému hierarchie (nebudeme ho zde přesně uvádět, neboť je velmi technický – zájemci je naleznou v literatuře) plyne, že *exponenciální „mezery“ mezi třídami jsou „ostré“*:

- $\text{LOGSPACE}, \text{NLOGSPACE} \subset \text{PSPACE}$,
- $\text{P} \subset \text{EXP}$,
- $\text{NP} \subset \text{NEXP}$,
- $\text{PSPACE} \subset \text{NEXPSPACE}$,
- $\text{EXP} \subset \text{2-EXP}$, ...

7.3.6 Některé další zajímavé výsledky

Věta 7.3.7 (*Blumův teorém*) Pro každou totální vyčíslitelnou funkci $f : \mathbb{N} \rightarrow \mathbb{N}$ existuje problém, jehož každé řešení s nějakou složitostí $t(n)$ může být zlepšeno tak, že nové řešení má složitost $f(t(n))$ pro skoro každé $n \in \mathbb{N}$.

Důkaz. Viz literatura.

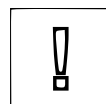
□

V důsledku Blumova teorému tedy existují problémy jejichž řešení se složitostí $t(n)$ je možné donekonečna zrychlovat na $\lg t(n)$, $\lg \lg t(n)$, $\lg \lg \lg t(n)$, ...

Nutnost pracovat s *časově konstruovatelnými funkcemi*, abychom se vyhnuli např. tomu, že $DTime[f(n)] = DTime[2^{f(n)}]$ pro nějaké $f(n)$, vyjadřuje následující teorém:

Věta 7.3.8 (*Gap Theorem*) Ke každé rekurzivní funkci $\phi(n) > n$ existuje rekurzivní funkce $f(n)$ taková, že $DTime[\phi(f(n))] = DTime[f(n)]$.

⁵ Bez důkazu jsme doplnili, že $\text{ELEMENTARY} \subset \text{PR}$.



Důkaz. Viz literatura – založeno na konstrukci funkce f takové, že žádný TS nezastaví pro vstup délky n počtem kroků mezi $f(n)$ a $\phi(f(n))$. \square

7.3.7 \mathcal{R} redukce, jazyky \mathcal{C} -těžké a \mathcal{C} -úplné

Až doposud jsme třídy používali jako horní omezení složitosti problémů. Všimněme si nyní *omezení dolního* – to zavedeme pomocí redukovatelnosti třídy problémů na daný problém.

\mathcal{R} redukce je (podobně jako v kapitole o vyčíslitelnosti) funkcí, která převádí problém na problém jiný. V teorii složitosti je ale potřeba, aby vyhovovala i jiným požadavkům, než je prostá vyčíslitelnost:

Definice 7.3.1 *Nechť \mathcal{R} je třída funkcí. Jazyk $L_1 \subseteq \Sigma_1^*$ je \mathcal{R} redukovatelný (přesněji \mathcal{R} many-to-one reducible) na jazyk $L_2 \subseteq \Sigma_2^*$, což zapisujeme $L_1 \leq_{\mathcal{R}}^m L_2$, jestliže existuje funkce f z \mathcal{R} taková, že $w \in L_1 \Leftrightarrow f(w) \in L_2$.*

Pokud se všechny problémy nějaké třídy \mathcal{R} redukují na jistý problém, pak je tento problém v oné třídě úplný vzhledem k \mathcal{R} redukci (alespoň tak těžký, jako kterýkoliv jiný problém této třídy):

Definice 7.3.2 *Nechť \mathcal{R} je třída funkcí a \mathcal{C} třída jazyků. Jazyk L_0 je \mathcal{C} -těžký (\mathcal{C} -hard) vzhledem k \mathcal{R} redukovatelnosti, jestliže $\forall L \in \mathcal{C} : L \leq_{\mathcal{R}}^m L_0$.*

Pokud problém těžký pro třídu vzhledem k \mathcal{R} redukci navíc do této třídy patří, pak je pro ni úplný (nejtěžší z této třídy):

Definice 7.3.3 *Nechť \mathcal{R} je třída funkcí a \mathcal{C} třída jazyků. Jazyk L_0 nazveme \mathcal{C} -úplný (\mathcal{C} -complete) vzhledem k \mathcal{R} redukovatelnosti, jestliže $L_0 \in \mathcal{C}$ a L_0 je \mathcal{C} -těžký (\mathcal{C} -hard) vzhledem k \mathcal{R} redukovatelnosti.*

Pro třídy $\mathcal{C}_1 \subseteq \mathcal{C}_2$ a jazyk L , jenž je \mathcal{C}_2 úplný vůči \mathcal{R} redukovatelnosti, platí, že buď \mathcal{C}_2 je celá \mathcal{R} redukovatelná na \mathcal{C}_1 nebo $L \in \mathcal{C}_2 \setminus \mathcal{C}_1$.

7.3.8 Nejběžnější typy \mathcal{R} redukce a úplnosti

Uvedme *nejběžněji používané typy úplnosti* – všimněme si, že je použita redukovatelnost dostatečně silná na to, aby bylo možné najít úplné problémy vůči ní a na druhou stranu nebyly příslušné třídy triviálně redukovány na jejich (možné) podtřídy:

- **NP, PSPACE, EXP** úplnost je definována vůči *polynomiální redukovatelnosti* (tj. redukovatelnosti pomocí DTS pracujících v polynomiálním čase),
- **P, NLOGSPACE** úplnost definujeme vůči *redukovatelnosti v deterministickém logaritmickém prostoru*,
- **NEXP** úplnost definujeme vůči *exponenciální redukovatelnosti* (tj. redukovatelnosti pomocí DTS pracujících v exponenciálním čase).

Zvláště důležitá je polynomiální redukce, protože podobně jako třída P vymezuje prakticky řešitelné problémy, polynomiální redukovatelnost odpovídá realizovatelné převoditelnosti problémů.

DEF

7.4 Příklady složitosti problémů

Příklady **LOGSPACE** problémů:

- existence cesty mezi dvěma uzly v neorientovaném grafu.

Příklady **NLOGSPACE**-úplných problémů:

- existence cesty mezi dvěma uzly v orientovaném grafu,
- 2-SAT (*SATisfiability*), tj. splnitelnost výrokových formulí tvaru konjunkce disjunkcí dvou literálů (literál je výroková proměnná nebo její negace), např. $(x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3)$.

Příklady **P**-úplných problémů:

- splnitelnost Hornových klauzulí $(p \wedge q \wedge \dots \wedge t) \Rightarrow u$, kde p, q, \dots jsou atomické formule predikátové logiky,
- náležitost řetězce do jazyka bezkontextové gramatiky,
- následnost uzlů při průchodu grafem do hloubky (pro dané řazení přímých následníků).

Příklady **NP**-úplných problémů:

- 3-SAT a obecný SAT – viz dále,
- řada grafových problémů, např.:
 - existence kliky dané velikosti,
 - existence Hamiltonovské kružnice v neorientovaném grafu,
 - existence orientované Hamiltonovské kružnice v orientovaném grafu,
 - barvitelnost: lze daný neorientovaný graf obarvit jistým počtem barev?,
 - uzlové pokrytí neorientovaného grafu množinou uzlů o určité velikosti (tj. množinou uzlů, se kterou souvisí všechny hrany),
 - ...
- problém obchodního cestujícího,
- *knapsack* – máme položky s cenou a hodnotou, maximalizujeme hodnotu tak, aby cena nepřekročila určitou mez.

Příklady **co-NP**-úplných problémů:

- ekvivalence regulárních výrazů bez iterace.

Příklady **PSPACE**-úplných problémů:

- ekvivalence regulárních výrazů,
- náležitost řetězce do jazyka kontextové gramatiky,
- model checking formulí lineární temporální logiky (LTL – výroková logika doplněná o temporální operátory *until, always, eventually, next-time*) s ohledem na velikost formule,
- nejlepší tah ve hře Sokoban.

$x + y$

Příklady **EXP-úplných problémů**:

- nejlepší tah v šachu (zobecněno na šachovnici $n \times n$),
- model checking procesů s neomezeným zásobníkem (rekurzí) vůči zafixované formuli logiky větvičího se času (CTL) – tj. EXP ve velikosti procesu.
- inkluze pro tzv. *visibly push-down* jazyky (operace push/pop, které provádí přijímající automat jsou součástí vstupního řetězce).

Příklady **EXPSPACE-úplných problémů**:

- ekvivalence regulárních výrazů doplněných o operaci kvadrát (tj. r^2).

k-EXP / k-EXPSPACE:

- rozhodování splnitelnosti formulí *Presburgerovy aritmetiky* – tj. celočíselné aritmetiky se sčítáním, porovnáváním (ne násobením – to vede na tzv. Peanovu aritmetiku, která je již nerozhodnutelná) a kvantifikací prvního řádu (např. $\forall x, y : x \leq x + y$) je problém, který je v **3-EXP (2-EXPSPACE-úplný)**.

Problémy mimo **ELEMENTARY**:

- ekvivalence regulárních výrazů doplněných o negaci,
- rozhodování splnitelnosti formulí logiky *WS1S* – celočíselná aritmetika s operací +1 a kvantifikací prvního a druhého řádu (tj. pro každou/existuje hodnota, resp. množina hodnot, taková, že ...),
- verifikace dosažitelnosti v tzv. *Lossy Channel Systems* – procesy komunikující přes neomezenou, ale ztrátovou frontu (cokoliv se může kdykoliv ztratit).

7.5 SAT-problém

SAT-problém (problém splnitelnosti booleovských formulí) byl prvním problémem, jehož NP-úplnost vzhledem k polynomiální redukci se podařilo dokázat. V mnoha případech jej lze s úspěchem použít k důkazu NP-těžkosti.

7.5.1 Polynomiální redukce

Definice 7.5.1 Polynomiální redukce jazyka L_1 nad abecedou Σ_1 na jazyk L_2 nad abecedou Σ_2 je funkce $f : \Sigma_1^* \rightarrow \Sigma_2^*$, pro kterou platí:

1. $\forall w \in \Sigma_1^* : w \in L_1 \Leftrightarrow f(w) \in L_2$

2. f je Turingovsky vyčíslitelná v polynomiálním čase

Existuje-li polynomiální redukce jazyka L_1 na L_2 , říkáme, že L_1 se redukuje na L_2 a píšeme $L_1 \leq_P^m L_2$.

Věta 7.5.1 Je-li $L_1 \leq_P^m L_2$ a L_2 je ve třídě P , pak L_1 je ve třídě P .

Důkaz. Nechť M_f je Turingův stroj, který provádí redukci f jazyka L_1 na L_2 a nechť $p(x)$ je jeho časová složitost. Pro libovolné $w \in L_1$ výpočet $f(w)$ vyžaduje nanejvýš $p(|w|)$ kroků a produkuje výstup maximální délky $p(|w|) + |w|$. Nechť

DEF

M_2 přijímá jazyk L_2 v polynomiálním čase daném polynomem $q(x)$. Uvažujme Turingův stroj, který vznikne kompozicí $\rightarrow M_f M_2$. Tento stroj přijímá jazyk L_1 tak, že pro každé $w \in L_1$ udělá stroj $\rightarrow M_f M_2$ maximálně $p(|w|) + q(p(|w|) + |w|)$ kroků, což je polynomem ve $|w|$ a tedy L_1 leží ve třídě P . \square

Příklad 7.5.1 Funkce $f : \{x, y\}^* \rightarrow \{x, y, z\}^*$ definována jako $f(v) = vzv$ je polynomiální redukcí jazyka $L_1 = \{w | w \text{ je palindrom nad } \{x, y\}\}$ na jazyk $L_2 = \{wzw^R | w \in \{x, y\}^*\}$.

Předchozí věta nám dává praktickou možnost jak ukázat, že určitý jazyk je ve třídě P . Navíc, přeformulujeme-li tuto větu takto: „Jestliže platí $L_1 \leq_P^p L_2$ a L_1 neleží v P , pak L_2 také neleží v P “, můžeme dokazovat, že určitý jazyk neleží v P .

$x + y$

7.5.2 Problém splnitelnosti - SAT problém

Nechť $V = \{v_1, v_2, \dots, v_n\}$ je konečná množina Booleovských proměnných (prvotních formulí výrokového počtu). *Literálem* nazveme každou proměnnou v_i nebo její negaci $\overline{v_i}$. *Klauzulí* nazveme výrokovou formuli obsahující pouze literály spojené výrokovou spojkou \vee (nebo).

Příklady klauzulí: $v_1 \vee \overline{v_2}$, $v_2 \vee v_3$, $\overline{v_1} \vee \overline{v_3} \vee v_2$.

SAT-problém lze formulovat takto: Je dána množina proměnných V a množina klauzulí nad V . Je tato množina klauzulí splnitelná?

Každý konkrétní SAT-problém můžeme zakódovat jediným řetězcem takto: Nechť $V = \{v_1, v_2, \dots, v_n\}$, každý literál v_i zakódujeme řetězcem délky m , který obsahuje samé 0 s výjimkou i -té pozice, která obsahuje symbol p , jde-li o literál v_i , nebo n , jde-li o literál $\overline{v_i}$. Klauzuli reprezentujeme seznamem zakódovaných literálů oddělených symbolem $/$. SAT-problém bude seznam klauzulí uzavřených v aritmetických závorkách.

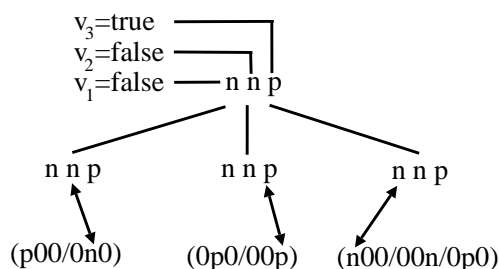
Příklad 7.5.2 SAT-problém obsahuje proměnné v_1, v_2, v_3 a klauzule $v_1 \vee \overline{v_2}$, $v_2 \vee v_3$, $\overline{v_1} \vee \overline{v_3} \vee v_2$ bude reprezentována řetězcem: $(p00/0n0)(0p0/00p)(n00/00n/0p0)$

$x + y$

Označme L_{SAT} jazyk obsahující řetězce tohoto typu, které reprezentují splnitelné množiny klauzulí. Řetězec $(p00/0n0)(0p0/00p)(n00/00n/0p0)$ je prvkem L_{SAT} ($v_1 = F, v_2 = F, v_3 = T$), na rozdíl od řetězce $(p00/0p0)(n00/0p0)(p00/0n0)$ ($n00/0n0$), který je kódem nesplnitelné množiny klauzulí $v_1 \vee v_2$, $\overline{v_1} \vee v_2$, $v_1 \vee \overline{v_2}$, $\overline{v_1} \vee \overline{v_2}$.

Přiřazení pravdivostních hodnot budeme reprezentovat řetězcem z $\{p, n\}^+$, kde p v i -té pozici představuje přiřazení $v_i \approx \text{true}$ a n v i -té pozici představuje přiřazení $v_i \approx \text{false}$.

Pak test, zda určité hodnocení je modelem množiny klauzulí (množina klauzulí je pro toto hodnocení splněna), je velmi jednoduchý a ilustruje ho obrázek:



Na uvedeném principu můžeme zkonstruovat nedeterministický Turingův stroj, který přijímá jazyk L_{SAT} v polynomiálním čase. Zvolíme 2-páskový Turingův stroj, který:

1. začíná kontrolou, zda vstup reprezentuje množinu klauzulí,
2. na 2. pásku vygeneruje řetězec z $\{n, p\}^m$ nedeterministickým způsobem,
3. posouvá hlavu na 1. pásce a testuje, zda pro dané ohodnocení (na 2. pásce) je množina klauzulí splnitelná.

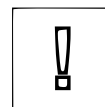
Tento proces může být snadno implementován s polynomiální složitostí přijetí v závislosti na délce vstupního řetězce a tedy $L_{SAT} \in NP$.

Věta 7.5.2 Cookův teorém: *Je-li L libovolný jazyk z NP , pak $L \leq_P^m L_{SAT}$.*

Důkaz. Protože $L \in NP$, existuje nedeterministický Turingův stroj M a polynom $p(x)$ tak, že pro každé $w \in L$ stroj M přijímá w v maximálně $p(|w|)$ krocích. Jádro důkazu tvoří konstrukce polynomiální redukce f z L na L_{SAT} : Pro každý řetězec $w \in L$ bude $f(w)$ množina klauzulí, které jsou splnitelné, právě když M přijímá w . \square

Věta 7.5.3 *SAT je NP-úplný vzhledem k polynomiální redukci.*

Důkaz. Přímou z předchozího. \square



7.5.3 NP-úplné jazyky

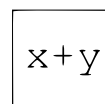
Po objevení Cookova teorému se ukázalo, že mnoho dalších NP jazyků má vlastnost podobnou jako L_{SAT} , t.j. jsou polynomiálními redukcemi ostatních NP jazyků.

Tyto jazyky se – jak již víme – nazývají *NP-úplné* (*NP-complete*) jazyky.

Kdyby se ukázalo, že libovolný z těchto jazyků je v P , pak by muselo platit $P = NP$; naopak důkaz, že některý z nich leží mimo P by znamenalo $P \subset NP$.

7.5.4 Význačné NP-úplné problémy

- *Satisfiability*: Je boolovský výraz splnitelný?
- *Clique*: Obsahuje neorientovaný graf kliku velikosti k ?
- *Vertex cover*: Má neorientovaný graf dominantní množinu mohutnosti k ?
- *Hamilton circuit*: Má neorientovaný graf Hamiltonovskou kružnici?
- *Colorability*: Má neorientovaný graf chromatické číslo k ?
- *Directed Hamilton circuit*: Má neorientovaný graf Hamiltonovský cyklus?
- *Set cover*: Je dána třída množin S_1, S_2, \dots, S_n . Existuje podtřída k množin $S_{i_1}, S_{i_2}, \dots, S_{i_k}$ taková, že $\bigcup_{j=1}^k S_{i_j} = \bigcup_{j=1}^n S_j$?
- *Exact cover*: Je dána třída množin S_1, S_2, \dots, S_n . Existuje množinové pokrytí (set cover) tvořené podtřídou po dvojicích disjunktních množin?



7.5.5 Použití redukcí k důkazu úplnosti

Příklad 7.5.3 Dokážeme, že problém $3SAT = \{\phi \mid \text{splnitelné formule } \phi \text{ v CNF obsahující v každé klauzuli nanejvýš 3 literály}\}$ je NP-úplný.

Musíme ukázat:

- a) $3SAT \in NP$
NTS si nedeterministicky zvolí přiřazení jednotlivým proměnným v $3SAT$ formulí a v polynomiálním čase ověří, zda jde o splňující přiřazení.
- b) $SAT \leq_p 3SAT$
Zkonstruujeme polynomiálně vyčíslitelnou funkci f , pro kterou platí:

$$A \in SAT \Leftrightarrow f(A) \in 3SAT$$

Funkce f pracuje následovně:

Nechť $K = (a_1, a_2, a_3, \dots, a_n)$ je klauzule v SAT která má víc než 3 literály. Klauzuli K nahradíme klauzulemi $K_1 = (a_1, a_2, b)$ a $K_2 = (\neg b, a_3, \dots, a_n)$, kde literál b se nevyskytuje v původní formuli. Je zřejmé, že platí:

$$K_1 \cup K_2 \text{ je splnitelná} \Leftrightarrow K \text{ je splnitelná}$$

a navíc klauzule K_2 má o jeden literál méně než původní klauzule K . Analogicky postupujeme, dokud formule obsahuje klauzule obsahující více než 3 literály. Je zřejmé, že funkce f je polynomiálně vyčíslitelná a navíc pro ni platí výše uvedený vztah.

Příklad 7.5.4 Dokážeme, že problém $KLIKA = \{(G, k) \mid G \text{ obsahuje úplný podgraf tvořený } k \text{ vrcholy}\}$ je NP-úplný. Musíme ukázat:

- a) $KLIKA \in NP$
NTS si nedeterministicky zvolí k vrcholů a v polynomiálním čase ověří zda tvoří úplný podgraf.
- b) $SAT \leq_p KLIKA$
Zkonstruujeme polynomiálně vyčíslitelnou funkci f , pro kterou platí:

$$A \in SAT \Leftrightarrow f(A) \in KLIKA$$

Funkce f vytvoří z formule ϕ v CNF graf G a číslo k takové, že formule ϕ je splnitelná, právě tehdy když graf G obsahuje úplný podgraf tvořený k vrcholy. Číslo k odpovídá počtu klauzulí ve formuli. Vrcholy grafu G odpovídají literálům vyskytujícím se ve formuli. Hrana v G mezi dvěma vrcholy, které odpovídají dvěma literálům, existuje právě tehdy, když tyto literály nejsou komplementární a nacházejí se ve dvou různých klauzulích.

\Leftarrow :

Předpokládejme, že formule ϕ je splnitelná a funkce $H : \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$ je přiřazení, které splňuje ϕ . Při tomto přiřazení existuje v každé klauzuli aspoň jeden literál, který má po tomto přiřazení hodnotu 1. Mezi každými dvěma vrcholy odpovídajícími těmto literálům existuje hrana, jelikož nejsou komplementární a nacházejí se v různých klauzulích. Tudíž vzniklý graf patří do $KLIKY$.

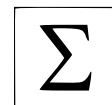
x + y

x + y

\Rightarrow :

Předpokládejme že vrcholy k_1, \dots, k_k jsou vrcholy grafu, které tvoří úplný podgraf. Jelikož mezi každými dvěma vrcholy je hrana, musí být odpovídající literály v navzájem různých klauzulích a zároveň nemohou být komplementární. Tudíž těmto literálům můžeme přiřadit hodnotu 1 a dostaneme splňující přiřazení pro ϕ .

Turingovy stroje jsou často používány také pro popis a klasifikaci časové a paměťové složitosti složitosti problémů (algoritmů). K nejdůležitějším složitostním třídám problémů (jazyků) patří třídy P, NP a NPC, avšak i ostatní třídy mají význam pro charakterizaci obtížnosti výpočtu. Reprezentativním problémem třídy NPC je SAT problém.



7.6 Cvičení

Příklad 7.6.1

1. Dokažte, že problém $\text{DoubleSat} = \{\phi \mid \text{formule } \phi \text{ v CNF která má dvě splnitelné přiřazení}\}$ je NP-úplný.
2. Dokažte, že problém rozhodnout, zda existuje obarvení neorientovaného grafu třemi barvami, tak aby každé dva sousední vrcholy měly různou barvu je NP-úplný.
3. Uvažme problém 2SAT. Rozhodněte, zda je NP úplný nebo leží v P.



Příklad 7.6.2

1. Definujte časovou a prostorovou složitost výpočtu Turingova stroje nad daným jazykem.
2. Proč se zavádí asymptotické omezení složitosti? Pro danou funkci f definujte množinu funkcí $O(f(n))$, $\Omega(f(n))$ a $\Theta(f(n))$.
3. Definujte následující složitostní třídy **P**, **NP**, **PSPACE**, **LOGSPACE**, **NLOGSPACE** a uveďte příklady problémů patřící do výše uvedených tříd.
4. Uveďte jaký vliv má použití vícepáskových TS a nedeterministických TS na složitost výpočtů.
5. Vysvětlete Savitchův teorém a jeho význam.
6. Uveďte hierarchii složitostních tříd a zamyslete se na ostrosti jednotlivých inkluzí.
7. Definujte obecně pojmy redukce, těžkost a úplnost. Dále definujte polynomiální redukci a vysvětlete na příkladu její použití.
8. Definujte pojem NP-úplnost a uveďte příklady problémů, které jsou NP-úplné.



Literatura

- [1] Aho, A. V, Ullman, J. D.: *The theory of parsing, translation and compiling*, EngelWood Cliffs, New Jersey, Prentice-Hall 1972.
- [2] Kolář, J.: *Algebra a grafy*, Skriptum ES ČVUT, Praha, 1982.
- [3] Kozen, D. C.: *A Completeness Theorem for KleeneAlgebras and the Algebra of Regular Events*, Technical Report TR 90-1123, 1990.
- [4] Kozen, D. C.: *Automata and Computability*, Springer-Verlag, New York, Inc, 1997. ISBN 0-387-94907-0
- [5] Rábová, Z., Češka, M.: *Základy systémového programování I.*, Učební texty vysokých škol. Ediční středisko VUT Brno, 1979.
- [6] Rábová, Z., Češka, M., Honzík, J. M., Hruška, T.: *Programovací techniky*, Učební texty vysokých škol. Ediční středisko VUT Brno, 1985.
- [7] Černá, I., Křetínský, M., Kučera, A.: *Automaty a formální jazyky I*, učební text FI MU, Brno, 1999.
- [8] Hopcroft, J.E., Motwani, R., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 2. vydání, 2000. ISBN 0-201-44124-1
- [9] Gruska, J.: *Foundations of Computing*, International Thomson Computer Press, 1997. ISBN 1-85032-243-0
- [10] Bovet, D.P., Crescenzi, P.: *Introduction to the Theory of Complexity*, Prentice Hall Europe, Pearson Education Limited, 1994. ISBN 0-13-915380-2