Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

ŠTÁTNY PEDAGOGICKÝ ÚSTAV
NATIONAL INSTITUTE FOR EDUCATION

Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Kapitoly z informatiky 2

Predmet: Kapitoly z informatiky

Línia: Vlastný odborový kontext informatiky a informatickej výchovy

Operačný program VZDELÁVANIE

EURÓPSKA ÚNIA
Európsky sociálny fond

Európska únia
Európsky sociálny fond

Moderné vzdelávanie pre vedomostnú spoločnosť/Projekt je spolufinancovaný zo zdrojov ES

Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Kapitoly z informatiky 2

Garant predmetu:

Doc. RNDr. Gabriela Andrejková, CSc.,
ÚI PF UPJŠ, Košice
Gabriela.Andrejkova@upjs.sk

Autori textu:

Doc. RNDr. Gabriela Andrejková, CSc.,
UPJŠ Košice
Gabriela.Andrejkova@upjs.sk

Doc. RNDr. Stanislav Krajči, PhD., UPJŠ Košice
Stanislav.Krajci@upjs.sk

Identifikácia modulu

Aktivita projektu:	1.2 Vzdelávanie nekvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ
Línia aktivity:	Vlastný odborový kontext informatiky a informatickej výchovy
Predmet:	Kapitoly z informatiky

Zaradenie modulu

Modul *Kapitoly z informatiky 2* je zo série troch modulov *Kapitoly z informatiky*, ktoré majú za úlohu priniesť frekventantom základy informatiky ako vedy. Modul predpokladá absolvovanie všetkých programátorských modulov *Programovanie 1—9* a tiež modulov *Matematika pre učiteľov informatiky 1—3*, *Algoritmy a údajové štruktúry 1*, *2* a *Počítačové systémy 1—5*.

Modul má e-learningovú podporu

Abstrakt modulu

Predmet *Kapitoly z informatiky* má za úlohu frekventantom predstaviť rôzne oblasti informatiky ako vedy. Do tohto modulu sme zaradili témy o dvoch základných výpočtových modeloch — konečných automatoch a Turingových strojoch, a to preto, že sú to jednak oblasti, ktoré pomáhajú pochopiť veľa rôznych problémov v teoretickej, ale aj v aplikovanej informatike, a tiež preto, že učitelia potrebujú mať isté kompetencie v teoretických výpočtových modeloch, aby mohli správne nasmerovať talentovaných študentov a rozumieť, čomu sa študenti v informatike venujú v rámci mimoškolského vzdelávania.

Cieľ modulu

Cieľom modulu je uviesť učiteľov do problematiky teoretických výpočtových modelov – konečných automatov a Turingových strojov. Po absolvovaní modulu by mali byť schopní vytvoriť konečný automat a odhadnúť, či pre požadovaný formálny jazyk existuje konečný automat, resp. vysvetliť, prečo neexistuje. Tiež by mali byť schopní skonštruovať Turingove stroje na riešenie niektorých typov problémov a pochopiť problém zastavenia Turingovho stroja.

Úvod

Výpočtový model – konečný automat – je veľmi jednoduché zariadenie, charakterizované nasledujúcimi vlastnosťami:

1. má konečný počet stavov,
2. má konečnú pásku, na ktorej je umiestnený vstup, a hlava číta slovo zľava doprava (1-krát),
3. jeho inštrukcie pracujú na základe aktuálneho stavu a prečítaného symbolu z pásky, a to buď deterministicky – má jednoznačne určený jediný nasledujúci stav automatu, alebo nedeterministicky – má určenú množinu nasledujúcich stavov automatu (má viac možností).

Pri prechode od deterministických k nedeterministickým automatom urobíme zmenu v inštrukciách, povolíme prechod z aktuálneho stavu prečítaním jedného symbolu do viacerých stavov. Tým sa zjednoduší konštrukcia konečných automatov. Oba typy automatov akceptujú rovnakú triedu jazykov – triedu regulárnych jazykov.

Na základe uvedeného sa nám bude zdať prirodzené uvažovať o iných modifikáciách týchto zariadení. Vedci pracujúci v tejto oblasti skúmajú rôzne modifikácie konečných automatov [7]. Existuje mnoho vedeckých výsledkov charakterizujúcich výpočtové schopnosti modifikovaných modelov. Tomu, kto by sa chcel viac zaoberať touto problematikou, odporúčame začať študijnou literatúrou uvedenou v závere tohto modulu.

Jedna z modifikácií je však veľmi dôležitá, preto sa jej chceme venovať aj v rámci tohto modulu. Tento nový modifikovaný model povoľuje viac možností pri práci s páskou, a to hlavica sa môže pohybovať oboma smermi a nebude len čítacou, ale aj zapisovacou. Okrem toho na páske bude vždy toľko miesta, koľko treba (teda páska nie je vopred ohraničená).

Ak povolíme zapisovanie na pásku, tak je možné na páske zaznamenať aj nejaké výsledky, Hlava sa vie k nim dostať a použiť ich v ďalších inštrukciách. Teda intuitívne by sme mohli uvažovať o tom, že nový výpočtový model bude schopný riešiť ťažšie úlohy. Tento výpočtový model – Turingov stroj – je predmetom štúdia nasledujúcej kapitoly v tomto module. Je to silnejší model, v ktorom je možné naprogramovať každý algoritmus, a navyše je možné presne vypočítať, koľko inštrukcií a miest na páske bolo spotrebovaných. Teda časová a pamäťová zložitosť je dobre merateľná. Zároveň je možné rôzne programy z týchto hľadísk porovnávať.

Pre oba výpočtové modely uvedieme ich základné vlastnosti a sústredíme sa na problémy, ktorých riešenie je významne spojené práve s týmito modelmi.

Obsah

1. Konečné automaty

V bežnom slova zmysle pod slovom automat rozumieme samočinný stroj, ktorý po vhodení mincí/vložení peňazí/karty vydá tovar alebo vykoná jednoduché služby (napríklad predajný automat, hrací automat, fotografický automat, bankomat) [7].

Zaoberajme sa trochu mincovým automatom. Mohli by sme povedať, že automat reaguje na vstup, ktorý je predstavovaný mincami. Ak dostane prijateľnú postupnosť mincí — prijateľný vstup, tak vydá tovar — *výsledok* alebo oznámi, že skončil. Ak dostane nevhodnú postupnosť mincí, tovar nevydá (túto situáciu je potrebné vhodne riešiť). Pritom automat musí postupnosť mincí analyzovať pomocou svojich vnútorných stavov. Zaujímajú nás predovšetkým automaty, ktoré majú konečný počet týchto stavov. Vieme si predstaviť, že vstupom budú nejaké symbolické žetóny. Konštrukcia takého automatu si vyžaduje prípravu presných čiastočných aktivít (krokov), ktoré pri správnej postupnosti mincí povedú k výdaju tovaru — k výsledku.

Predmetom záujmu teórie konečných automatov [4] je každý systém, pre ktorý vieme určiť konečný počet jeho stavov, do ktorých sa môže dostať, a tiež konečný počet vonkajších podnetov spôsobujúcich zmeny stavov. Pritom je podstatné, aby stav systému a vonkajší podnet spolu určovali nasledujúci stav.

Od takýchto modelov automatov je možné prejsť k abstraktnejším modelom automatov, ktoré budú opísané formálnejšie tak, aby sa v nich dali presne sledovať kroky, ktoré vykonajú.

Automat v našom chápaní je teda veľmi jednoduchý výpočtový model, ktorý je väčšinou spájaný so spracovaním *postupností symbolov* na vstupe, ktoré budeme nazývať *slovami*, a teda môže byť použitý na rozpoznávanie slov, pri ktorých vydá výsledok (pripraví sa na jeho vydanie) a pri ktorých nie. Okrem toho automaty poskytujú veľmi jednoduchý spôsob na vysvetlenie toho, ako sa modelujú výpočty [3]. Z didaktického hľadiska sú modelom, pre ktorý vieme jednoducho a presne definovať základné pojmy, medzi ktoré patria *konfigurácia*, *krok výpočtu*, *výpočet*, *simulácia*, *determinizmus* a *nedeterminizmus*, pomocou ktorých je možné charakterizovať niektoré podtriedy algoritmov a dozvedieť sa niečo dôležité o ich vlastnostiach. Dôležité je tiež pochopiť, že existujú úlohy, ktoré nie je možné pomocou automatov riešiť.

Konečné automaty sú veľmi vhodným modelom pre modelovanie dôležitých typov softvéru a hardvéru, medzi ktoré patrí [2]:

1. Softvér na návrh a riadenie funkcií digitálnych obvodov.
2. „Lexikálny analyzátor“, ktorý je súčasťou kompilátorov.
3. Softvér na vyhľadávanie slov, fráz a iných vzorov v rozsiahlych textových dokumentoch.
4. Softvér na verifikáciu systémov všetkých typov s konečným počtom stavov, napríklad komunikačné protokoly.

Konečný automat je výpočtový model, preto nás zaujíma, **akým spôsobom realizuje výpočet, aké inštrukcie vie spracovať, akú má pamäť, aký je prístup do pamäti, ako spracováva vstupy a ako vydá (vyjadrí) výstup**.

Vstupom automatu bude postupnosť prvkov nejakej abecedy, preto definujme formálne prvky, ktoré sa týkajú vstupu.

Definícia 1.1 *K základným pojmom patria:*

- ***Abeceda*** *je ľubovoľná neprázdna konečná množina, ktorej prvky budeme nazývať* ***písmená****.*
- *Konečné postupnosti písmen budeme nazývať* ***slová****.*
- *Prázdnu postupnosť písmen budeme nazývať* ***prázdne slovo*** *a označovať* ε*.*
- *Ak* $a_1, \ldots, a_n$ *sú písmená, potom* ***dĺžkou slova*** $w = a_1 \ldots a_n$ *rozumieme číslo* $|w| = n$*. Dĺžka prázdneho slova je nula.*
- *Počet písmen* a *v slove* w *budeme označovať* $|w|_a$*.*
- *Ak* $u = a_1 \ldots a_n$ *a* $v = b_1 \ldots b_m$ *sú slová, potom* ***zreťazením*** *slov* u, v *rozumieme slovo* $u \cdot v = a_1 \ldots a_n b_1 \ldots b_m$*. Obyčajne budeme písať* uv *namiesto* $u \cdot v$*.*
- ***Jazykom*** *v abecede* Σ *rozumieme ľubovoľnú množinu slov v abecede* Σ *(t. j. ľubovoľnú podmnožinu množiny* Σ^* *s konečným alebo nekonečným počtom prvkov).*

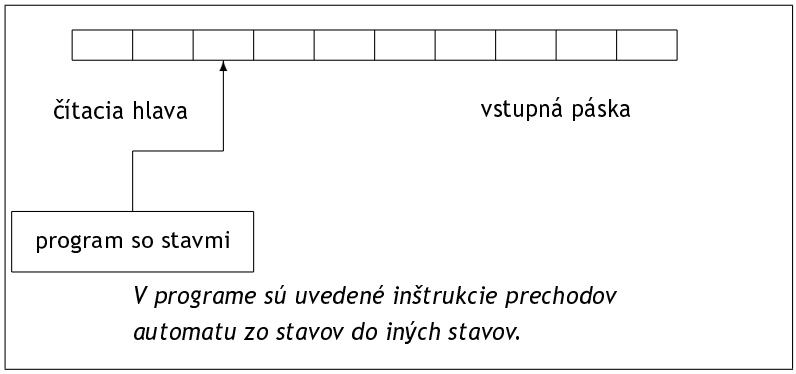
Problematika automatov a formálnych jazykov, o ktorých pojednáva kniha [4], patrí medzi najdlhšie študované a najpodrobnejšie spracované disciplíny tvoriace odbor matematická informatika.

Označenie. Množinu všetkých slov v abecede Σ označujeme Σ^*, množinu všetkých neprázdnych slov v abecede Σ označujeme Σ^+.

Príklad 1.1	Abecedy: $\Sigma_1 = \{0,1\}, \Sigma_2 = \{a,b,c,d\}, \Sigma_3 = \{0,1,2,3,4,5,6,7,8,9\}$. Slová: $000111, 0101$ v abecede Σ_1, $aaabbbcd$, $abcd$ v abecede Σ_2, 926 v abecede Σ_3. Dĺžka slova 000111 je 6, dĺžka $aaabbbcd$ je 8, atď. Počet písmen b v slove $aaabbbcd$ je 3, teda $\|aaabbbcd\|_b = 3$. Slovo $aaabbbcdabcd$ je zreťazením slov $aaabbbcd$, $abcd$. Nech $\Sigma = \{a,b,c\}$. Potom $\Sigma^* = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, \dots\}$. Jazykmi sú napríklad nasledujúce množiny $\emptyset, \{\varepsilon\}, \{a\}, \{abaa, ab, bbab\}$, množina všetkých slov v Σ^* s párnym počtom výskytov písmena c, $\{wcw \mid w \in \{a,b\}^*\}$, atď. Jazykom v abecede Σ_1 je napríklad množina všetkých slov s rovnakým počtom núl a jedničiek.

1.1. Reprezentácie konečných automatov

Konečný automat môže prepínať medzi svojimi stavmi (na základe vstupu), ktorých má konečný počet (aktuálny stav je pamätaný). Konečný automat si teda môžeme predstaviť ako zariadenie s konečnostavovou riadiacou jednotkou a vstupnou páskou (je tu umiestnený vstup), ako je to znázornené na obrázku 1.



Obr. 1: Štruktúra automatu.

Zariadenie má jednu čítaciu hlavu, ktorá sa môže posúvať po vstupnej páske len zľava doprava. Toto zariadenie pracuje v taktoch, pričom v každom takte (kroku), na základe stavu riadiacej jednotky a čítaného písmena zo vstupnej pásky, zmení stav riadiacej jednotky a pohne čítacou hlavou doprava na ďalšie písmenko vstupného slova.

1.1.1. Inštrukcie automatu

Akým spôsobom by sme mohli vyjadriť inštrukcie (príkazy) automatu? Sú to vlastne inštrukcie výberu vykonané na základe prečítaného písmena zo vstupu a stavu, v ktorom sa automat práve nachádza, pričom sa výber vykonáva väčšinou z viacerých možností. Pripomenieme si programovací jazyk Pascal, ktorý pre výber z viacerých možností ponúka príkaz case. Budeme tiež používať príkaz skoku, ktorý v jazyku Pascal tiež existuje, ale jeho používanie vo všeobecnosti sa neodporúča, pretože zneprehľadní program.

Teória konečných automatov vzišla [9] z práce W. S. McCullocha a V. Pittsa *A logical calculus of the ideas immanent in nervous activity* z r. 1943 o vzťahu algebry a neurónových sietí. Je zaujímavé, že von Neumann hodnotil ich výsledky ako „probably the most significant result obtained with axiomatic method up to now“, ktorý ukazuje, že „any functioning which can be defined at all logically, strictly and unambiguously in a finite number of words can also be realised by such a formal neural network“. Týmito slovami dnes obvykle charakterizujeme Turingov výsledok, ktorý von Neumann poznal. Znamená to, že ani takému géniovi, akým bol von Neumann, nebol vtedy dostatočne jasný rozdiel medzi tým, čo dnes nazývame konečným a nekonečným automatom.

Príkaz skoku

goto návestie

vykoná v programe skok na riadok s označením **návestie**.

Príkaz výberu **case** má tvar

návestie:	**case** vstup **of**
	{
	a_1 : príkaz skoku 1
	...
	a_n : príkaz skoku n
	}

Keďže program obsahuje len príkazy výberu a skoku, je nutné označiť návestím každý riadok, na ktorý sa skok vykoná. Kvôli jednoduchosti použijeme postupne návestia q_0, q_1, ..., q_n. Návestím q_0 označíme prvý, t. j. štartovací príkaz. V zápise programu štartovací príkaz označíme šípkou doprava. Ešte potrebujeme vyjadriť, kedy sa program skončí. Návestia príkazov ukončenia označíme šípkou doľava. Ak po prečítaní vstupu sa automat dostane na príkaz ukončenia, znamená to, že daný vstup automat akceptoval (rozpoznal), inak ho neakceptoval (dostal postupnosť, ktorá nevedie do príkazov ukončenia).

Príklad 1.2

Automat pracuje v abecede $\{a, b\}$, má program uvedený v tabuľke 1. Začne prvým príkazom, a ak má na vstupe prvé písmeno a, prejde na ďalšie písmeno na vstupe a na príkaz s návestím q_1. Ak má na vstupe prvé písmeno b, tak prejde na príkaz s návestím q_4 atď. Na návestí q_5 je prázdny príkaz a program sa tu skončí.

→	q_0 : **case** vstup **of** {	a : **goto**	q_1
		b: **goto**	q_4 }
	q_1 : **case** vstup **of** {	a : **goto**	q_2
		b: **goto**	q_4 }
	q_2 : **case** vstup **of** {	a : **goto**	q_3
		b : **goto**	q_4 }
	q_3 : **case** vstup **of** {	a : **goto**	q_1
		b : **goto**	q_4 }
	q_4 : **case** vstup **of** {	a : **goto**	q_5
		b : **goto**	q_4 }
←	q_5 :		

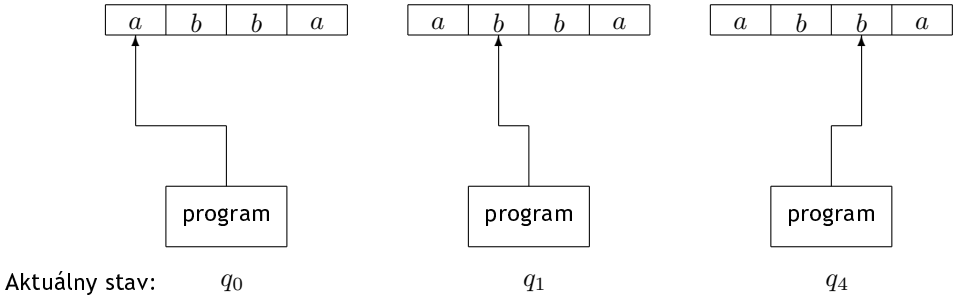
Tab. 1: Program pre automat v príklade 1.2.

Sledujme postup programu cez návestia pre dva rôzne vstupy:

- Pre vstup $abba$ dostávame postupnosť prechodov cez návestia q_0, q_1, q_4, q_4, q_5, program sa skončil v koncovom stave q_5; priebeh výpočtu v prvých troch krokoch je zaznamenaný na obrázku 2.
- Pre vstup $aaaa$ dostávame postupnosť prejdených návestí q_0, q_1, q_2, q_3, q_1, vstup je prečítaný a program sa neskončil v koncovom stave.

Pre niektoré slová na vstupe sa automat dostane do koncového stavu, pre niektoré nie, tiež niektoré vstupné slová na páske nedočíta do konca. **Zaujímavé sú pre nás**

tie slová, ktoré sú prečítané celé a pri ktorých automat skončil v koncovom stave. Môžeme vtedy povedať, že automat určuje množinu akceptovaných (rozpoznaných) slov – jazyk. Jazyk, chápaný v bežnom zmysle slova má tri aspekty – syntaktický, sémantický a pragmatický. Predmetom nášho záujmu je syntaktický aspekt – slová nás zaujímajú ako postupnosti písmen.



Obr. 2: Sledovanie výpočtu pre slovo $abba$ v prvých troch krokoch.

1.1.2. Reprezentácia automatu pomocou grafov

Tvorba takýchto programov nie je ťažká, avšak my poznáme názornejšie vyjadrenie automatu, a to pomocou grafov. Graf nám umožňuje názorne sledovať postup výpočtu pre dané vstupné slovo. Začneme nasledujúcou úlohou:

Úloha 1.1	Vytvorte program, ktorý bude používať príkazy **case** vo vyššie uvedenom tvare, a bude rozpoznávať jazyk L v abecede $\Sigma = \{a, b\}$, ktorý obsahuje všetky slová s nepárnym počtom písmen b.

Riešenie: Idea: Je nutné počítať počet prečítaných písmen b. V prípade ich nepárneho počtu slovo môže byť rozpoznané.

Program:

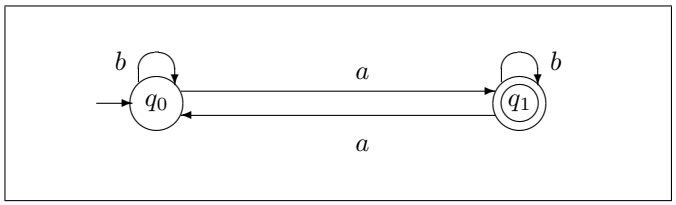
$\rightarrow$	q_0 : **case** vstup **of** {	a : **goto**	q_1
		b : **goto**	q_0 }
$\leftarrow$	q_1 : **case** vstup **of** {	a : **goto**	q_0
		b : **goto**	q_1 }

Tento program je možné previesť na graf uvedený na obrázku 3. Návestia (stavy automatu) zakreslíme ako vrcholy grafu, ich názvy zapíšeme do vnútra. Prípady príkazu **case** zakreslíme pomocou šípok. Počiatočné a koncové návestie označíme šípkou, resp. dvojitým krúžkom. Výpočet v automate je možné sledovať prechodom cez graf v smere šípok. Ak by sme chceli sledovať, ako sa bude automat správať na slove $bbaaab$, tak začneme vo vrchole s počiatočným návestím cez písmeno b prejdeme do vrcholu so stavom q_0 cez ďalšie b znovu do vrcholu so stavom q_0, cez písmeno a do vrcholu so stavom q_1 atď.

Kvôli zjednodušeniu vyjadrovania vrchol so stavom q_i budeme nazývať stav q_i.

1.1.3. Reprezentácia automatu pomocou prechodovej tabuľky

Skrátene by sme mohli celý program z príkladu 1.2 zapísať aj pomocou nasledujúcej úspornej prechodovej tabuľky 2, v ktorej je uvedený celý príkaz case v jednom riadku. Pod písmenom abecedy je uvedené návestie skoku.



Obr. 3: Automat z úlohy 1.1 vyjadrený pomocou grafu.

	Návestie	a	b
$\rightarrow$	q_0 :	q_1	q_4
	q_1 :	q_2	q_4
	q_2 :	q_3	q_4
	q_3 :	q_1	q_4
	q_4 :	q_5	q_4
$\leftarrow$	q_5 :		

Tab. 2: Prechodová tabuľka.

Máme už vyriešenú otázku vstupu, pamäte, inštrukcií, zostáva nám doriešiť ukončenie programu. Ukončenie programu je stanovené vykonaním skoku na vopred určené návestie. V predchádzajúcom príklade to je q_5. Ak automat skončí v takto označenom stave, je to v poriadku. Ak nie, je chyba buď v programe alebo na vstupe.

Dôležitým pozorovaním je tu to, že automat bude spracovávať slová v nejakej abecede a vie dať odpoveď pomocou koncového návestia, či slovo rozpoznal.

Úloha 1.2	Vytvorte program, ktorý bude používať príkazy **case** vo vyššie uvedenom tvare a bude rozpoznávať jazyk $L = \{01^i0; i > 0\}$.

Riešenie: Zrejme abeceda je $\Sigma = \{0, 1\}$. Slová patriace do jazyka majú taký tvar, že sa začínajú a končia jedným písmenom 0 a medzi nimi je ľubovoľný počet písmen 1, ale aspoň jedno.

Idea riešenia je nasledujúca: Ak je vo vstupnom slove na začiatku písmeno 0, tak je to dobre a program pokračuje na návestí q_1. Ak nasleduje 1, znovu je to dobre a program pokračuje na návestí q_2. Ak ďalej nasleduje niekoľko ďalších 1, znovu je to dobre a program pokračuje na q_2. Ak potom nasleduje 0, je to dobre a program prejde na návestie q_3. Ostatné prípady sú zlé a program sa končí v stave q_4, ktorý nie je koncový. Koncovým návestím je q_3. Ak sa program dostane na príkaz s návestím q_4 slovo nerozpozná.

Program:

$\rightarrow$	q_0 : **case** vstup **of** {	0 : **goto**	q_1
		1 : **goto**	q_4 }
	q_1 : **case** vstup **of** {	0 : **goto**	q_4
		1 : **goto**	q_2 }
	q_2 : **case** vstup **of** {	0 : **goto**	q_3
		1 : **goto**	q_2 }
$\leftarrow$	q_3 :		
	q_4 :		

Riešenie pomocou prechodovej tabuľky:

	Návestie	0	1
→	q_0 :	q_1	q_4
	q_1 :	q_4	q_2
	q_2 :	q_3	q_2
←	q_3 :		
	q_4 :		

Po vytvorení tohoto riešenia si kladieme otázku o jeho správnosti. Kedy by sme mohli považovať vytvorené riešenie za správne? Odpoveď znie:

Program je správny, ak rozpozná všetky slová požadovaného tvaru a žiadne iné.

Ukázať tieto dve požadované vlastnosti programu je niekedy celkom jednoduché, inokedy je potrebné použiť princíp matematickej indukcie alebo iné matematické prostriedky. My sa budeme snažiť uviesť rozumné, pochopiteľné zdôvodnenia a nebudeme robiť celkom formálne dôkazy.

Existuje viacero programov na vizualizáciu automatov. Jeden z nich nájdete na stránke: http://www.cs.usfca.edu/~jbovet/vas.html.

1.2. Formálny zápis automatu

Aby sme sa mohli zaoberať presnými výpočtami na automatoch detailnejšie, zavedieme formálnu definíciu konečného automatu.

Definícia 1.2 *Deterministický* ***konečný automat (KA)*** *nazývame päticu* $A = (K, \Sigma, \delta, q_0, F)$*, kde*

- K *je konečná množina* ***stavov*** *(v programe sú to návestia príkazov),*
- Σ *je* ***vstupná abeceda****,*
- $q_0 \in K$ *je* ***počiatočný stav****,*
- $F \subseteq K$ *je množina* ***koncových stavov****,*
- $\delta : K \times \Sigma \to K$ *je* ***prechodová funkcia****.*

$\delta(q, a) = p$ *určuje, že automat* A *prejde zo stavu* q *do stavu* p*, ak čítacia hlava prečítala písmeno* a*.*

Konfiguráciou konečného automatu A nazývame dvojicu (q, w), kde $q \in K$ a $w \in \Sigma^*$.

Vyjadrenie, že automat je v konfigurácii (q, w), znamená, že automat je v stave q a neprečítaná časť vstupu je w, pričom čítacia hlava je na prvom symbole slova w.

Konfigurácia (q_0, w) sa nazýva **počiatočná konfigurácia** automatu A na w.

Konfigurácia (q, ε) sa nazýva **koncová konfigurácia**, ak $q \in F$.

Definícia 1.3 ***Krok*** *konečného automatu* A *je určený prechodovou funkciou* δ *a vyjadruje prechod z nejakého stavu* q *do nejakého stavu* p *na základe vstupného znaku* a*. Budeme to zapisovať*

$$(q, aw) \vdash_A (p, w),$$

ak $\delta(q, a) = p$*, kde* $a \in \Sigma, w \in \Sigma^*$*.*

Ak je zrejme o aký automat ide, budeme písať $\vdash$ namiesto $\vdash_A$.

Príklad 1.3	Automat na obrázku Obr. 3 má formálne vyjadrenie $$A = (\{q_0, q_1\}, \{a, b\}, \delta, q_0, \{q_1\}),$$ pričom prechodová funkcia δ je určená daným grafom. Pre slovo bab je konfigurácia (q_0, bab) je počiatočná konfigurácia. Krok tohto automatu z počiatočnej konfigurácie (q_0, bab) je $(q_0, bab) \vdash (q_0, ab)$. Ďalší krok je $(q_0, ab) \vdash (q_1, b)$ a ďalší $(q_1, b) \vdash (q_1, \varepsilon)$. Koncová konfigurácia je (q_1, ε).

Definícia 1.4 *Výpočtom C automatu A nazývame postupnosť $C = C_1 C_2 \dots C_k$ konfigurácií takú, že susedné členy postupnosti sú kroky, t. j. $C_{i-1} \vdash_A C_i$ pre $0 < i \leq k$.*

C nazývame výpočet automatu A na slove w, ak platí $C_0 = (q_0, w)$ a $C_k \in Q \times \{\varepsilon\})$, teda celé vstupné slovo je prečítané.

C je akceptujúci výpočet, ak to je výpočet, a teda automat prečítal celé slovo a skončil v koncovom stave, t. j. $C_k \in F \times \{\varepsilon\}$.

C je odmietajúci výpočet, ak to je výpočet, a teda automat prečítal celé slovo a neskončil v koncovom stave, t. j. $C_k \in (Q - F) \times \{\varepsilon\}$.

Poznámka: Automat má pre každé slovo len akceptujúci, alebo len odmietajúci výpočet.

Definícia 1.5 *Jazyk akceptovaný konečným automatom A je množina slov, ktoré automat prečíta celé a skončí v koncovom stave.*

Príklad 1.4	Slovo $aababb$ patrí do $L(A)$ definovaného automatom v úlohe 2, lebo existuje akceptujúci výpočet: $(q_0, aababb) \vdash_A (q_1, ababb) \vdash_A (q_0, babb) \vdash_A (q_0, abb) \vdash_A (q_1, bb) \vdash_A (q_1, b) \vdash_A (q_1, \varepsilon)$. Slovo $bbabb$ nepatrí do $L(A)$ definovaného automatom v úlohe 1.1, lebo neexistuje akceptujúci výpočet. Uvedený výpočet je odmietajúci: $(q_0, bbabb) \vdash_A (q_0, babb) \vdash_A (q_0, abb) \vdash_A (q_1, bb) \vdash_A (q_0, b) \vdash_A (q_0, \varepsilon)$.

Definícia 1.6 *Trieda regulárnych jazykov je trieda všetkých jazykov, ktoré sú akceptované konečnými automatmi. Formálne to vieme vyjadriť*

$$\mathcal{L}_{\mathrm{RE}} = \{L(A); A \text{ je konečný automat}\}.$$

Ľubovoľný jazyk z $\mathcal{L}_{\mathrm{RE}}$ je regulárny.

Čo sme sa naučili:

1. Vyjadriť zariadenie - konečný automat, ktorý má konečný počet stavov a konečný vstup viacerými zápismi, zaviedli sme jeho formálnu definíciu a pojmy charakterizujúce výpočet na ňom.
2. Sledovať výpočet automatu a určiť, ktoré slová akceptuje a ktoré nie.
3. Definovať jazyk akceptovaný konečným automatom.

1.3. Cvičenie

C. 1.1	Navrhnite konečný automat pre každý z nasledujúcich regulárnych jazykov: (a) $\{w \in \{0,1,2\}^* \mid w = 002122x, x \in \{0,1,2\}^*\}$, (b) $\{w \in \{a,b,c\}^* \mid w = xabcabc, x \in \{a,b,c\}^*\}$, (c) $\{w \in \{a,b,c\}^* \mid w = xaabby, x,y \in \{a,b,c\}^*\}$, (d) $\{abbxbbby \mid x,y \in \{a,b\}^*\}$, (e) $\{w \in \{a,b\}^* \mid w = abbz$ pre $z \in \{a,b\}^* \wedge w = ubbbv$ pre $u,v \in \{a,b\}^*\}$, (f) $\{w \in \{a,b\}^* \mid \ \ \lvert w\rvert_a = 2 \wedge \lvert w\rvert_b \equiv 1 \pmod 3\}$. Postačuje uviesť grafovú reprezentáciu týchto automatov.

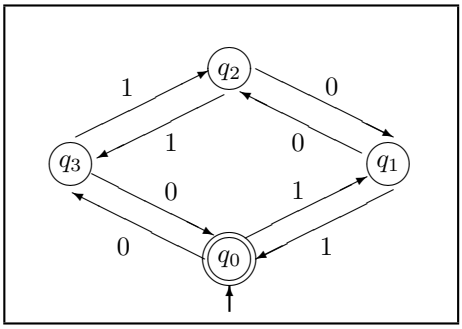
1.4. Čo vedia automaty kontrolovať

Pretože výpočty automatu krok po kroku predstavujú dlhé a málo prehľadné postupnosti, zavedieme ich zjednodušený zápis, ktorý vznikne združením po sebe idúcich krokov do jedného kroku. Budeme to zapisovať $\vdash_A^k$, kde k vyjadruje počet krokov, ktoré tu združujeme. $\vdash_A^*$ budeme používať na vyjadrenie existujúceho výpočtu, ak nevieme počet krokov. Analogicky, $\hat{\delta}(q,w) = p$ bude vyjadrovať, že automat je v stave p po prečítaní jedného alebo viac symbolov. $\hat{\delta}$ sa nazýva rozšírená prechodová funkcia.

Toto nám umožňuje inak vyjadriť jazyk akceptovaný konečným automatom, a to

$$L(A) = \{w \in \Sigma^* \mid (q_0, w) \vdash_A^* (p, \varepsilon), p \in F\}.$$

Príklad 1.5	Uvažujme konečnostavový automat vyjadrený v grafovom tvare na obrázku 4. Automat pracuje s abecedou $\Sigma = \{0,1\}$. Štartovacím aj koncovým stavom je stav q_0. Aké slová bude tento automat akceptovať? Budeme sa snažiť určiť jazyk akceptovaný týmto automatom.



Obr. 4: Konečnostavový automat pracujúci v abecede $\{0,1\}$.

Každá cesta zo štartovacieho stavu do štartovacieho stavu má párny počet hrán.

Sledujme výpočet na slove 010111

$$(q_0, 010111) \vdash_A (q_3, 10111) \vdash_A (q_2, 0111) \vdash_A (q_1, 111) \vdash_A (q_0, 11) \vdash_A (q_1, 1)$$

$$\vdash_A (q_0, \varepsilon)$$

alebo

$$(q_0, 010111) \vdash_A^* (q_0, \varepsilon)$$

Teda skúmané slovo patrí do jazyka rozpoznávaného týmto automatom. Všimnime si, že pri výpočte sme prechádzali stavom q_0, a teda aj slovo 0101 by bolo akceptované. Ďalším pozorovaním by sme prišli na to, že akceptovaným jazykom je jazyk tvorený všetkými slovami, ktoré obsahujú párny počet 0 a párny počet 1. Označme tento jazyk L, teda

$$L = \{w \in \{0,1\}^* | \quad |w|_0 \equiv 0 \,(\mathrm{mod}\, 2) \wedge |w|_1 \equiv 0 \,(\mathrm{mod}\, 2)\} \tag{1}$$

Označme

$$T[q_0] = \{w \in \{0,1\}^* | \hat{\delta}(q_0, w) = (q_0, \varepsilon)\}$$

jazyk akceptovaný týmto automatom ($[q_0]$ vyjadruje koncový stav automatu)

Chceli by sme dokázať, že platí

$$L = T[q_0], \tag{2}$$

t. j. L je jazyk akceptovaný automatom na obrázku 4. Toto je však potrebné doložiť formálnejším dôkazom.

Najprv urobíme nasledujúce úvahy:

Čo sa stane, ak zmeníme výstupný stav? Ak by výstupným stavom bol stav q_1, dostali by sme iný jazyk. Označme

$$T[q_1] = \{w \in \{0,1\}^* | \hat{\delta}(q_0, w) = (q_1, \varepsilon)\} \tag{3}$$

Úplne analogicky by sme mohli definovať $T[q_2]$ a $T[q_3]$:

$$T[q_2] = \{w \in \{0,1\}^* | \hat{\delta}(q_0, w) = (q_2, \varepsilon)\} \tag{4}$$

$$T[q_3] = \{w \in \{0,1\}^* | \hat{\delta}(q_0, w) = (q_3, \varepsilon)\} \tag{5}$$

Dostali sme po dvojiciach navzájom disjunktné množiny jazykov, ktorých zjednotením je množina $\Sigma^* = \{0,1\}^*$. Množiny $T[q_0], T[q_1], T[q_2],\ T[q_3]$ tvoria rozklad množiny $\{0,1\}^*$ na $|Q|$ (v našom prípade na 4 množiny (triedy).

V závislosti od koncového stavu dostávame rôzne triedy slov. Mohlo by sa stať, že automat má viac koncových stavov, a teda akceptovaný jazyk bude zjednotením akceptovaných množín, čo je možné napísať

$$L(A) = \bigcup_{p \in F} T[p]. \tag{6}$$

Teraz ukážeme, že platí (2), a teda platí (7):

$$T[q_0] = \{w \in \{0,1\}^* | \quad |w|_0 \equiv 0 \,(\mathrm{mod}\, 2) \wedge |w|_1 \equiv 0 \,(\mathrm{mod}\, 2)\} \tag{7}$$

Indukciou. Najprv ukážeme, že (7) platí pre všetky slová dĺžky najviac 2. Potom z predpokladu, že (7) platí pre slová dĺžky $n-1$ ukážeme jeho platnosť pre slová dĺžky n.

1. Analyzujme všetky slová dĺžky najviac 2.
   - $\hat{\delta}(q_0, \varepsilon) = q_0$, a teda $\varepsilon \in T[q_0]$
   - $\hat{\delta}(q_0, 0) = q_0$, a teda $0 \in T[q_0]$, $\hat{\delta}(q_0, 1) = q_0$, a teda $1 \in T[q_0]$
   - $\hat{\delta}(q_0, 00) = q_0$, a teda $00 \in T[q_0]$, $\hat{\delta}(q_0, 01) = q_0$, a teda $01 \in T[q_2]$
   - $\hat{\delta}(q_0, 10) = q_0$, a teda $10 \in T[q_2]$, $\hat{\delta}(q_0, 11) = q_0$, a teda $11 \in T[q_0]$

   Všetky slová dĺžky 2 patriace do $T[q_0]$ spĺňajú $|w|_0 \equiv 0 \,(\mathrm{mod}\, 2) \wedge |w|_1 \equiv 0 \,(\mathrm{mod}\, 2)$.
2. Predpokladajme, že (7) platí pre slová dĺžky $n-1$. Nech $|w| = n-1, w \in T[q_0]$ a $|w|_0 \equiv 0 \,(\mathrm{mod}\, 2) \wedge |w|_1 \equiv 0 \,(\mathrm{mod}\, 2)$.

- Ak vytvoríme slová $w0$ a $w1$, tak podmienka $|w|_0 \equiv 0 \pmod 2 \wedge |w|_1 \equiv 0 \pmod 2$ už nebude splnená ani pre jedno z nich, teda ani jedno z týchto slov nepatrí do $T[q_0]$ a zároveň $w0 \in T[q_3]$ a $w1 \in T[q_1]$.
- Nech $|w| = n-1, w \in T[q_2]$ a $|w|_0 \equiv 1 \pmod 2 \wedge |w|_1 \equiv 1 \pmod 2$. Ak vytvoríme slová $w0$ a $w1$, tak podmienka $|w|_0 \equiv 0 \pmod 2 \wedge |w|_1 \equiv 0 \pmod 2$ nebude splnená ani pre jedno z nich, teda ani jedno z týchto slov nepatrí do $T[q_0]$ a zároveň $w0 \in T[q_1]$ a $w1 \in T[q_3]$.
- Nech $|w| = n-1, w \in T[q_1]$ a $|w|_0 \equiv 0 \pmod 2 \wedge |w|_1 \equiv 1 \pmod 2$. Ak vytvoríme slová $w0$ a $w1$, tak podmienka $|w|_0 \equiv 0 \pmod 2 \wedge |w|_1 \equiv 0 \pmod 2$ bude splnená len pre $w1$, teda toto slovo patrí do $T[q_0]$ a zároveň $w0 \in T[q_2]$.
- Nech $|w| = n-1, w \in T[q_3]$ a $|w|_0 \equiv 1 \pmod 2 \wedge |w|_1 \equiv 0 \pmod 2$. Ak vytvoríme slová $w0$ a $w1$, tak podmienka $|w|_0 \equiv 0 \pmod 2 \wedge |w|_1 \equiv 0 \pmod 2$ bude splnená len pre $w0$, teda toto slovo patrí do $T[q_0]$ a zároveň $w1 \in T[q_2]$.

Sledujme výpočet na slove 01011 v prípade koncového stavu q_1:

$$(q_0, 01011) \vdash_A (q_3, 1011) \vdash_A (q_2, 011) \vdash_A (q_1, 11) \vdash_A (q_0, 1) \vdash_A (q_1, \varepsilon)$$

Automat dané slovo akceptuje. Ďalšími pozorovaniami by sme zistili, že akceptovaným jazykom je jazyk tvorený všetkými slovami, ktoré obsahujú párny počet 0 a nepárny počet 1. V tomto prípade to môžeme vyjadriť nasledovne:

$$L_1 = \{w \in \{0,1\}^* | \quad |w|_0 \equiv 0 \pmod 2 \wedge |w|_1 \equiv 1 \pmod 2)\} \tag{8}$$

Úplne analogickými pozorovaniami by sme dostali pre koncové stavy q_2 a q_3

$$L_2] = \{w \in \{0,1\}^* | \quad |w|_0 \equiv 1 \pmod 2 \wedge |w|_1 \equiv 1 \pmod 2)\} \tag{9}$$

$$L_3 = \{w \in \{0,1\}^* | \quad |w|_0 \equiv 1 \pmod 2 \wedge |w|_1 \equiv 0 \pmod 2)\} \tag{10}$$

Analogicky by sme vedeli dokázať, že platí

$$L_1 = T[q_1], L_2 = T[q_2], L_3 = T[q_3].$$

Čo sme sa naučili:

- Konečný automat rozkladá rozkladá jazyk Σ^* na konečný počet tried, Σ je abeceda, v ktorej automat pracuje.
- Automatom akceptovaný jazyk je zjednotením niektorých týchto tried, čo je určené koncovými stavmi automatu.

1.5. Pre niektoré jazyky konečné automaty neexistujú

Aby sme ukázali, že nejaký konkrétny jazyk nie je regulárny, postačí dokázať, že neexistuje automat, ktorý rozpoznáva tento jazyk. Obvykle dokázať neexistenciu riešenia daného problému v špecifickej triede algoritmov je veľmi ťažké (sú to najťažšie úlohy v teoretickej informatike). Pretože trieda konečných automatov je trieda veľmi silne ohraničených programov, dôkazy neexistencie typu „neexistuje žiadny konečný automat, ktorý akceptuje daný jazyk“ sú relatívne ľahké [3]. Tento fakt nám poslúži na prezentáciu metodológie dôkazov tohto typu. Najprv uvedieme jednoduchý príklad a potom sa budeme snažiť o zovšeobecnenie.

Dirichletov princíp, slabá forma: Ak máme $n+1$ objektov rozdelených do n skupín, tak v aspoň jednej skupine sú aspoň dva objekty.

Príklad 1.6	Jazyk $L = \{0^n 1^n | n \geq 1\}$ nie je akceptovateľný konečným automatom.

Prv než prejdeme k formálnejšiemu dôkazu, je dôležité si uvedomiť, že automat si pri prejdení do nejakého stavu (aktuálny stav) vôbec nepamätá, ako sa do tohto stavu

dostal. Podstatné je to, že je v danom stave a aký symbol hlava práve číta. Teda do daného aktuálneho stavu sa mohol dostať viacerými cestami v grafe a pritom hlava mohla prečítať slová rôznej dĺžky.

Dôkaz: Sporom. Predpokladajme, že jazyk L je rozpoznateľný konečným automatom A, ktorý má $k > 0$ stavov.

Uvažujme slovo $w = 0^n1^n$, kde $n > k$. Aj toto slovo je akceptované automatom A, a teda existuje akceptujúci výpočet $C = C_0C_1 \dots C_{2n-1}$, každé C_i, $0 \leq i \leq 2n-1$ je konfigurácia.

Keďže automat má len k stavov, v prvej polovici tejto postupnosti musia existovať dve konfigurácie, ktoré majú rovnaké stavy. Nech sú to konfigurácie C_i a C_j, $0 < i < j$, $j \leq n-1$. Keďže slovo má tvar 0^n1^n, poznáme tvar konfigurácií. Konfigurácia $C_i = (q, 0^{n-i}1^n)$ a konfigurácia $C_j = (q, 0^{n-j}1^n)$.

Uvažujme, čo sa stane, keď v akceptujúcom výpočte C vynecháme časť $C_i \dots C_{j-1}$:

Ak $\delta(q_0, u) = p = \delta(q_0, v)$, tak pre všetky x platí $\delta(q_0, ux) = p = \delta(q_0, vx)$.

Zrejme nová postupnosť C' bude akceptujúcim výpočtom pre slovo w', z ktorého vynecháme podslovo 0^{j-i}. Teda automat A bude akceptovať aj slovo $0^{n-(j-i)}1^n$. Podľa definície tohto jazyka ale $0^{n-(j-i)}1^n \notin L$. Dospeli sme teda k sporu s predpokladom, že L je rozpoznateľný konečným automatom.

Príklad 1.7	Jazyk tvorený slovami v abecede $\Sigma = \{), (\}$, pričom každé slovo je správne uzatvorkované. Napríklad, slová $(())()$, $(()())$, $((()()())())$ patria do daného jazyka. Tento jazyk nie je akceptovateľný konečným automatom. Keby bol, tak akceptuje aj podmnožinu slov $\{(^n)^n : n > 0\}$. Podľa predchádzajúceho príkladu vieme, že taký automat neexistuje.

Príklad 1.8	Jazyk $L = \{a^{n^2} \mid n \geq 0\}$ nie je akceptovateľný konečným automatom.

Dôkaz: Sporom. Ak by bol tento jazyk bol akceptovateľný konečným automatom, tak by taký automat existoval a mal by konečný počet stavov, povedzme $k > 0$.

Uvažujme akceptujúce výpočty pre slová a^{k^2}, a^{k^2+1}, . . . , a^{k^2+k}, ktorých je $k+1$.

$C^0 = C_0C_1 \dots C_{k^2}$

$C^1 = C_0C_1 \dots C_{k^2}C_{k^2+1}$

$C^2 = C_0C_1 \dots C_{k^2}C_{k^2+1}C_{k^2+2}$

$C^3 = C_0C_1 \dots C_{k^2}C_{k^2+1}C_{k^2+2}C_{k^2+3}$

. . .

$C^k = C_0C_1 \dots C_{k^2}C_{k^2+1} \dots C_{k^2+k}$

Keďže slov je $k+1$ a stavov je k, existujú dve slová i-te a^{k^2+i}, j-te a^{k^2+j} také, že $0 \leq i < j \leq k$, ktoré sú akceptované v rovnakom stave, t. j. platí

$$\hat{\delta}(q_0, a^{k^2+i}) = \hat{\delta}(q_0, a^{k^2+j}).$$

Tiež teda platí, že rozpoznanie slova $a^{k^2+i}a^{2k-j+1}$ a slova $a^{k^2+j}a^{2k-j+1}$ skončí v tom istom stave, pretože za obe slová sme pripojili rovnaké slovo.

Pritom ale slovo $a^{k^2+j}a^{2k-j+1} = a^{k^2+2k+1} = a^{(k+1)^2} \in L$, zatiaľ čo $k^2 < k^2 + i + 2k - j + 1 < (k+1)^2$, a preto $a^{k^2+i+2k-j+1} \notin L$. Teda znovu sme dospeli k sporu s predpokladom, že akceptujúci konečný automat existuje.

Štruktúra dôkazov v príkladoch 1.6 a 1.8 je rovnaká. Líši sa len voľbou reťazcov, ktoré slúžia ako východisko na odvodenie sporu. Nájdenie takéhoto reťazca sa spravidla

opiera o intuitívny pohľad, že skúmaný jazyk má vlastnosť, k dosiahnutiu ktorej je konečná pamäť automatu nedostatočná.

Ak sa nám podarí dokázať, že pre nejaký jazyk neexistuje deterministický konečný automat, ktorý ho akceptuje, tak tento jazyk určite nie je regulárny.

C. 1.2	Nech $L = \{w \in \{a,b\}^* \mid \|w\|_a = \|w\|_b\}$. Je možné skonštruovať konečný automat, ktorý bude akceptovať tento jazyk?

Čo sme sa naučili:

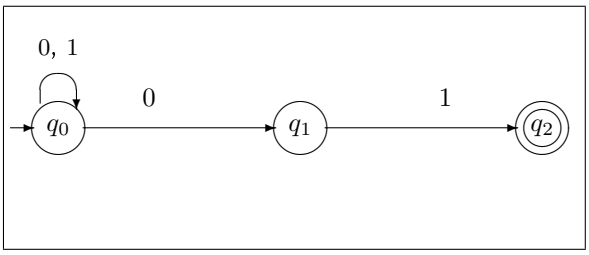
- Metódu na dokazovanie, že pre niektoré jazyky konečný automat neexistuje.

1.6. Neformálny pohľad na nedeterminizmus konečných automatov

Významnou vlastnosťou konečného automatu, tak ako sme ho definovali, bolo, že ku každému slovu w sme vedeli nájsť práve jeden výpočet na slove w. Podstata determinizmu je v tom, že v každom stave automatu je jednoznačne určený ďalší stav po prečítaní písmena pod hlavou. Ak povolíme prechod do viacerých stavov, už ďalší postup nie je jednoznačne určený, a automat sa stáva nedeterministickým. Nedeterministické automaty budú pracovať zložitejšie, ale pomocou nich je možné opísať akceptovaný jazyk „jednoduchším“ spôsobom.

Podobne ako deterministický konečný automat aj nedeterministický konečný automat má konečný počet stavov, konečnú abecedu, jeden štartovací stav a množinu akceptujúcich stavov. Tiež má prechodovú funkciu, ktorú budeme označovať δ. Rozdiel medzi deterministickým a nedeterministickým automatom je práve v prechodovej funkcii. Prechodová funkcia nedeterministického automatu dáva výsledok množinu stavov (môže byť prázdna, môže obsahovať jeden alebo viac stavov). Automat pri výpočte prechádza do jedného z týchto stavov. Vieme, že pri deterministickom automate to bol práve jeden stav.

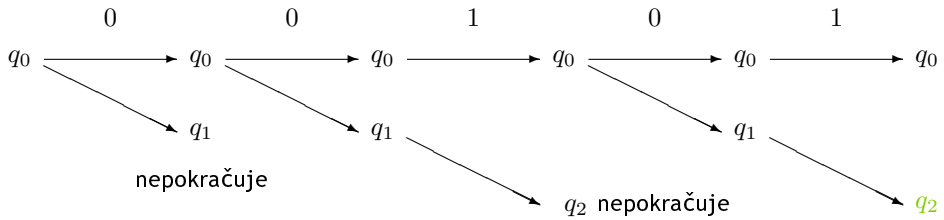
Príklad 1.9	Na obrázku 5 je uvedený nedeterministický automat. Stav q_0 je štartovací stav a v ňom môžeme predpokladať, že sa vždy začína koniec skúmaného slova, t. j. 01. Okrem toho stav q_0 je aj prechodový. Zo stavu q_0 je prechod do q_0 pri spracovaní aj 0 aj 1. Vidíme, že zo stavu q_0 pri spracovaní 0 automat môže prejsť do stavu q_0 alebo do stavu q_1. Uvidíme, že tento automat akceptuje všetky slová ukončené 01.



Obr. 5: Nederministický automat akceptujúci všetky slová v abecede $\{0,1\}$ ukončené 01.

Sledujme strom výpočtov automatu na slove 00101. Výpočty sa začínajú v štartovacom stave q_0.

- Na vstupe je 0, takže automat má dve možnosti prechodu, a to do stavu q_0 a do stavu q_1, sledujeme oba prechody. Strom výpočtov je znázornený na obrázku 6.
- Nasledujúcim vstupným symbolom je 0, postup zo stavu q_1 ďalej nie je možný. Zo stavu q_0 automat má znovu dve možnosti prechodu, a síce do stavu q_0 a do stavu q_1, sledujeme oba prechody.
- Ďalším vstupným symbolom je 1. Zo stavu q_1 je možný prechod do stavu q_2, ktorý je koncovým stavom, ale slovo nebolo celé prečítané a ďalší postup odtiaľ nie je možný, takže automat zatiaľ neakceptuje. Zo stavu q_0 automat má len jednu možnosť prechodu, a to do stavu q_0.
- Ďalším vstupným symbolom je 0. Zo stavu q_0 automat má znovu dve možnosti prechodu, a to do stavu q_0 a do stavu q_1, sleduje oba prechody.
- Posledným vstupným symbolom je 1. Zo stavu q_1 je možný prechod do stavu q_2, ktorý je koncovým stavom, slovo bolo prečítané celé, teda teraz automat slovo akceptuje. Ďalšia analýza nie je potrebná, pretože sme našli akceptujúci výpočet.



Obr. 6: Sledovanie stavov automatu z obrázku 5 počas spracovania slova 00101.

Podobne ako pri deterministických konečných automatoch aj pre nedeterministické je možné na ich zápis použiť prechodové tabuľky.

		0	1
$\rightarrow$	q_0 :	$\{q_0, q_1\}$	$\{q_0\}$
	q_1 :	$\emptyset$	$\{q_2\}$
$\leftarrow$	q_2 :	$\emptyset$	$\emptyset$

Tab. 3: Prechodová tabuľka k automatu, ktorý akceptuje všetky slová končiace sa na 01.

Definícia 1.7 *Nedeterministickým konečným automatom nazývame päticu $A = (K, \Sigma, \delta, q_0, F)$, kde K, Σ, q_0, F sú ako v definícii KA a $\delta : K \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^K$ je prechodová funkcia.*

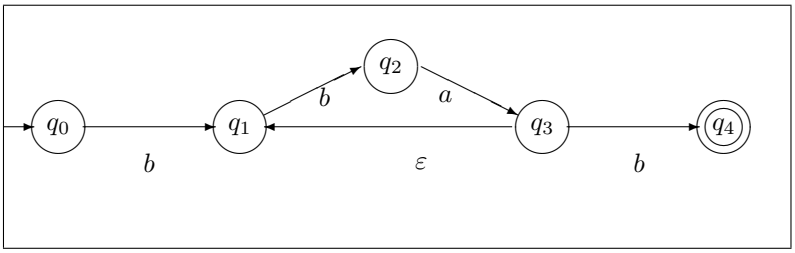
Pojem konfigurácie zostáva pre nedeterministický konečný automat rovnaký ako pre deterministický, ale líšiť sa bude v definícii kroku, pretože podľa definície je možný prechod do množiny stavov.

Definícia 1.8 *Krok nedeterministického konečného automatu A budeme označovať $\vdash_A$ a je definovaný na množine $Q \times \Sigma^*$ tak, že $(q, aw) \vdash_A (p, w)$, ak existuje $p \in \delta(q, a)$, kde $a \in \Sigma \cup \{\varepsilon\}$, $w \in \Sigma^*$.*

Ďalším rozdielom je, že prechodová funkcia u nedeterministického automatu je definovaná aj pre argument ε, a teda automat môže meniť stav bez toho, aby hýbal čítacou hlavou na vstupnom slove. Navyše, vzhľadom na to, že prázdna množina $\emptyset$

môže byť tiež hodnotou prechodovej funkcie, automat sa môže dostať do situácie, v ktorej nemôže vykonávať žiadnu činnosť.

Príklad 1.10	Na obrázku obrázku 7. je uvedený jednoduchý nedeterministický konečný automat, v ktorom sa stretávame s prechodom na ε, t. j. automat prejde na iný stav bez toho, aby posunul čítaciu hlavu. Tento automat má grafové vyjadrenie rovnaké ako deterministický. Tento automat akceptuje jazyk, ktorý začína a končí symbolom b a medzi nimi sa opakuje len dvojica ba, pričom je tam aspoň jedna. Napríklad slová $bababbb$, $aabb$ patria do jazyka akceptovaného týmto automatom.



Obr. 7: Nedeterministický automat s ε prechodom.

Definícia 1.9 *Jazyk rozpoznávaný nedeterministickým konečným automatom* A *je množina* $L(A) = \{w | (q_0, w) \vdash_A^* (q, \varepsilon), q \in F\}$.

V prípade deterministického konečného automatu A, k tomu, aby slovo w nepatrilo do L(A) stačilo, aby stav, v ktorom končil výpočet (jediný!) na w nebol koncový. V prípade nedeterministického automatu a buď nesmie existovať žiadny výpočet na w, alebo **všetky** výpočty na w musia končiť v stavoch rôznych od koncových.

Napriek tomu, že definícia 1.2 sa formálne zhoduje s definíciou 1.7, treba si uvedomiť podstatný rozdiel, spôsobovaný odlišnou definíciou **kroku** $\vdash$. Kým v definícii 1.2 sme mohli povedať, že $w \in L(A)$, ak sa výpočet automatu A na slove w končí v koncovom stave, v definícii 1.7 musíme hovoriť, že $w \in L(A)$, ak **existuje** výpočet automatu A na w končiaci sa v koncovom stave. Pri nedeterministickom automate sa totiž môže stať, že výpočet na w vôbec neexistuje, alebo naopak, že na w existuje niekoľko výpočtov, ale nie všetky musia končiť v koncovom stave.

Z definícií oboch automatov, determinisického a nedeterministického, vyplýva, že každý deterministický automat je zároveň nedeterministický. Opačne to neplatí, ale platí nasledujúce tvrdenie:

Tvrdenie:	Nech L je jazyk rozpoznávaný nejakým nederministickým konečným automatom. Potom existuje deterministický konečný automat, ktorý rozpoznáva jazyk L.

Objasníme situáciu na intuitívnej úrovni. Ukážeme na príklade, ako je možné k nedeterministickému automatu skonštruovať deterministický.

Ak sa chceme presvedčiť, či určitý nedeterministický konečný automat akceptuje dané slovo, nemusí náhodné hľadanie v tabuľke či v stavovom diagrame viesť k cieľu a treba postupovať nejakým systematickým spôsobom. Napríklad je možné urobiť analýzu tak, ako sme ju urobili v predchádzajúcom príklade na základe doposiaľ prečítaného úseku vstupného slova. K množinám stavov budeme vytvárať stavy, ktoré ich budú reprezentovať. Ku každému nedeterministickému konečnému automatu A teda takto môžme vytvoriť systém, ktorý je realizáciou istého konečného automatu rozpoznávajúceho $L(A)$.

Uvažujme nedeterministický konečný automat uvedený na obrázku 5, ktorý akceptuje slová v abecede $\{0,1\}$ ukončené 01. Chceme k tomuto automatu skonštruovať deterministický konečný automat.

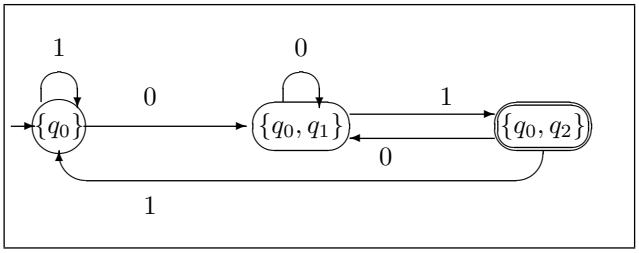
Keďže prechodová funkcia dáva výsledky, ktoré sú množinami, budeme analyzovať všetky možné podmnožiny množiny stavov.

Ak uvažujeme nejakú podmnožinu stavov a prechod z nich na nejaké písmeno, výsledkom bude tiež množina, ktorá je zjednotením množín daných prechodovou funkciou pre tieto stavy. Napríklad, ak uvažujeme množinu stavov $\{q_0, q_1\}$ automatu na obrázku 5, pre stav q_0 prechodová funkcia pri vstupe 0 dáva $\{q_0, q_1\}$ a pre stav q_1 je to množina $\{q_0\}$, takže zjednotením je $\{q_0, q_1\}$. Štartovacím stavom bude množina obsahujúca pôvodný štartovací stav a koncové stavy budú množiny obsahujúce koncový stav. Vytvorená prechodová tabuľka je tabuľka 4.

		0	1
	$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_1\}$	$\emptyset$	$\{q_2\}$
$\leftarrow$	$\{q_2\}$	$\emptyset$	$\emptyset$
	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\leftarrow$	$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\leftarrow$	$\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$\leftarrow$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Tab. 4: Prechodová tabuľka k deterministickému automatu, ktorý bol vytvorený k automatu na Obr. 5.

Nový automat by teda mal mať 8 stavov (čo je počet všetkých podmnožín danej množiny stavov), avšak ľahko zistíme, že do stavov $\emptyset, \{q_1\}, \{q_0, q_1, q_2\}, \{q_2\}, \{q_1, q_2\}$ sa nemožno dostať zo štartovacieho stavu, takže dostávame automat na obrázku 8.



Obr. 8: Deterministický automat akceptujúci všetky slová ukončené 01.

Čo sme sa naučili:

Nedeterminizmus umožní ľahšie konštruovať konečné automaty, ale trieda akceptovaných jazykov je rovnaká pre oba typy automatov.

1.7. Praktické využitie automatov

V tejto podkapitolke ukážeme, že tento teoretický výpočtový model je vynikajúcim modelom pre riešenie niektorých reálnych problémov, napríklad pri vyhľadávaní na Webe alebo pri získavaní informácií z textov. Budeme sa viac venovať druhému problému.

Formulácia problému	Je daná množina slov. Nájdite všetky dokumenty, ktoré obsahujú jedno (alebo všetky) z týchto slov.

Existuje viacero veľmi zaujímavých a prefíkaných algoritmov pre riešenie tohoto problému, ktoré zohľadňujú napríklad, či databáza dokumetov je usporiadaná, či sa často mení, a iné. Naše riešenie nebude zohľadňovať takmer nič a bude sformulované takto:

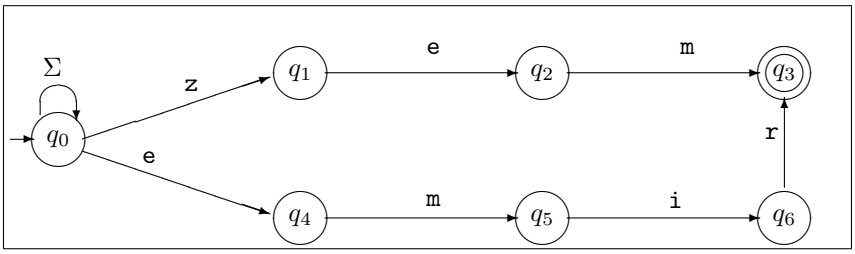
Budeme predpokladať, že máme danú množinu slov, ktoré budeme nazývať kľúčové slová, a chceme nájsť výskyt niektorého z týchto slov v danom veľmi dlhom textovom dokumente.

Samozrejme, že vieme vytvoriť jednoduchý intuitívny algoritmus taký, že pre každé kľúčové slovo prejdeme celý text a budeme postupne porovnávať symboly. Lenže tento algorimus si vyžaduje prečítanie dlhého textu toľkokrát, koľko je kľúčových slov.

Automaty nám pomôžu vytvoriť algoritmus, ktorý spracováva všetky slová naraz a dlhý dokument prečíta len jedenkrát. Z kľúčových slov vytvoríme nedeterministický konečný automat, prevedieme ho na deterministický a pre tento automat bude vstupom dlhý text, ktorý bude spracovaný len raz. Ak sa automat dostane do koncového stavu, znamená to, že niektoré kľúčové slovo našiel. Ukážeme to na malom príklade.

Príklad 1.11 *Predpokladajme, že v dlhom textovom dokumente chceme nájsť slová* `zem` *a* `emir`*. Vidíme, že slová sa prekrývajú, a toto je potrebné pri riešení zohľadniť.*

Vytvoríme nedeterministický automat uvedený na obrázku 9



Obr. 9: Nederministický automat akceptujúci slová ukončené kľúčovými slovami `zem` a `emir`.

Prechodová funkcia pre tento automat z obrázku 9 je uvedená v tabuľke 5.

		e	m	i	r	z	$\Sigma - \{\mathtt{e},\mathtt{z}\}$
$\rightarrow$	$\{q_0\}$	$\{q_0, q_4\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_1\}$	$\{q_2\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_2\}$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\leftarrow$	$\{q_3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_4\}$	$\emptyset$	$\{q_5\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_6\}$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_6\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$

Tab. 5: Prechodová tabuľka k nedeterministickému automatu na obrázku 9.

K tomto automatu vieme vytvoriť deterministický automat, ktorého prechodová tabuľka je tabuľka 6

Prechodom cez stĺpce tabuľky zistíme, že ďalšie množiny stavov, do ktorých by automat prešiel, už nevzniknú.

		e	m	i	r	z	$\Sigma - \{\mathtt{e},\mathtt{z}\}$
	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow$	$\{q_0\} = A$	$\{q_0, q_4\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_1\}$	$\{q_2\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_2\}$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\leftarrow$	$\{q_3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_4\}$	$\emptyset$	$\{q_5\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_6\}$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_6\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$
	$\{q_0, q_1\} = B$	$\{q_0, q_4, q_2\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_0, q_2\}$	$\{q_0, q_4\}$	$\{q_0, q_3\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\leftarrow$	$\{q_0, q_3\} = C$	$\{q_0, q_4\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_0, q_4\} = D$	$\{q_0, q_4\}$	$\{q_0, q_5\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_0, q_5\} = E$	$\{q_0, q_4\}$	$\{q_0\}$	$\{q_0, q_6\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_0, q_6\} = F$	$\{q_0, q_4\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_3\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_1, q_2\}$	$\{q_2\}$	$\{q_3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\leftarrow$	$\{q_1, q_3\}$	$\{q_2\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_1, q_4\}$	$\{q_2\}$	$\{q_5\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_1, q_5\}$	$\{q_2\}$	$\emptyset$	$\{q_6\}$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_1, q_6\}$	$\{q_2\}$	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$
$\leftarrow$	$\{q_2, q_3\}$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_2, q_4\}$	$\emptyset$	$\{q_3, q_5\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_2, q_5\}$	$\emptyset$	$\{q_3\}$	$\{q_6\}$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_2, q_6\}$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$
	$\{q_3, q_4\}$	$\emptyset$	$\{q_5\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_3, q_5\}$	$\emptyset$	$\emptyset$	$\{q_6\}$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_3, q_6\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$
	$\{q_4, q_5\}$	$\emptyset$	$\{q_5\}$	$\{q_6\}$	$\emptyset$	$\emptyset$	$\emptyset$
	$\{q_4, q_6\}$	$\emptyset$	$\{q_5\}$	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$
	$\{q_5, q_6\}$	$\emptyset$	$\emptyset$	$\{q_6\}$	$\{q_3\}$	$\emptyset$	$\emptyset$
	$\{q_0, q_2, q_4\} = G$	$\{q_0, q_4\}$	$\{q_0, q_3, q_5\}$	$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\leftarrow$	$\{q_0, q_3, q_5\} = H$	$\{q_0, q_4\}$	$\{q_0\}$	$\{q_0, q_6\}$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\ldots$	$\ldots$	$\ldots$				

Tab. 6: Prechodová tabuľka (neúplná) deterministického automatu vytvoreného k nedeterministickému automatu na Obr. 9. Červenou farbou sú označené stavy, ktoré budú vo výslednom deterministickom automate.

1.8. Cvičenia

C. 1.3	$L = \{w \in \{a,b\}^* \mid \exists u, v \in \{a,b\}^* \quad w = uav \wedge \lvert u\rvert \equiv 1 \pmod 2\}$ Inými slovami povedané, že v každom slove existuje symbol a na párnej pozícii od začiatku. Zostrojte konečný automat, ktorý rozpoznáva jazyk L.
C. 1.4	$L = \{w \in \{a,b\}^* \mid \exists u, v, y \in \{a,b\}^* \quad w = uavay \wedge \lvert u\rvert \equiv 1 \pmod 2 \wedge \lvert v\rvert \equiv 1 \pmod 2\}$ Inými slovami povedané, že v každom slove existujú aspoň dva symboly a na párnej pozícii od začiatku. Zostrojte konečný automat, ktorý rozpoznáva jazyk L.
C. 1.5	Zostrojte nedeterministický konečný automat pre jazyk: $L = \{w \in \{a,b\}^* \mid \exists v \in \{a,b\}^* w = vabba\}$. Vzniknutý akceptor preveďte na deterministický konečný automat.

Program, ktorý by sme pre vytvorený deterministický automat skonštruovali má tvar:

→	A : **case** vstup **of** {	`e` : **goto**	D
		`m` : **goto**	A
		`i` : **goto**	A
		`r` : **goto**	A
		`z` : **goto**	B
		$\Sigma - \{\mathtt{e},\mathtt{z}\}$: **goto**	A }
	B : **case** vstup **of** {	`e` : **goto**	G
		`m` : **goto**	A
		`i` : **goto**	A
		`r` : **goto**	A
		`z` : **goto**	B
		$\Sigma - \{\mathtt{e},\mathtt{z}\}$: **goto**	A }
←	C : **case** vstup **of** {	`e` : **goto**	D
		`m` : **goto**	A
		`i` : **goto**	A
		`r` : **goto**	A
		`z` : **goto**	B
		$\Sigma - \{\mathtt{e},\mathtt{z}\}$: **goto**	A }
	D : **case** vstup **of** {	`e` : **goto**	D
		`m` : **goto**	E
		`i` : **goto**	A
		`r` : **goto**	A
		`z` : **goto**	B
		$\Sigma - \{\mathtt{e},\mathtt{z}\}$: **goto**	A }
	E : **case** vstup **of** {	`e` : **goto**	D
		`m` : **goto**	A
		`i` : **goto**	F
		`r` : **goto**	A
		`z` : **goto**	B
		$\Sigma - \{\mathtt{e},\mathtt{z}\}$: **goto**	A }
	F : **case** vstup **of** {	`e` : **goto**	D
		`m` : **goto**	A
		`i` : **goto**	A
		`r` : **goto**	C
		`z` : **goto**	B
		$\Sigma - \{\mathtt{e},\mathtt{z}\}$: **goto**	A }
	G : **case** vstup **of** {	`e` : **goto**	D
		`m` : **goto**	H
		`i` : **goto**	A
		`r` : **goto**	A
		`z` : **goto**	B
		$\Sigma - \{\mathtt{e},\mathtt{z}\}$: **goto**	A }
←	H : **case** vstup **of** {	`e` : **goto**	D
		`m` : **goto**	A
		`i` : **goto**	F
		`r` : **goto**	A
		`z` : **goto**	B
		$\Sigma - \{\mathtt{e},\mathtt{z}\}$: **goto**	A }

Čo sme sa naučili:

- Konštruovať nedeterministické konečné automaty ku kľúčovým slovám.
- K nedeterministickému konečnému automatu skonštruovať deterministický a použiť ho v efektívnom algoritme na hľadanie kľúčových slov.

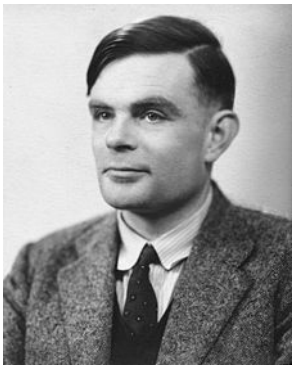
2. Turingove stroje

2.1. Načo Turingove stroje?

Je paradoxné, že kľúč k pochopeniu obmedzenosti informačných technológií poskytuje samotná veda o počítačoch (computer science), ba — netají sa tým. Dôležitou súčasťou teoretickej informatiky je totiž kapitola o Turingových strojoch. Tento výpočtový model má dve základné vlastnosti:

1. Tak ako každý iný počítačový program, i softvér Turingovho stroja je zložený z inštrukcií, v jeho prípade sú však všetky jediného typu.
2. Každý iný (doteraz známy) počítačový program možno bez straty informácie transformovať na program nejakého Turingovho stroja.

Kým druhá črta redukuje otázku, čo nedokáže počítač, na otázku, čo nedokáže Turingov stroj, prvá umožňuje omnoho jednoduchší výskum takejto otázky. Pomocou tohto výpočtového modelu tak môžeme nájsť konkrétne problémy, s ktorými si žiaden automat nikdy neporadí (azda najznámejší je problém zastavenia Turingovho stroja (tzv. halting problem)). Ich existencia dokazuje principiálnu obmedzenosť (nielen ideálnych) výpočtových prostriedkov, a tak slúži ako chrobák v učiteľovej hlave a povzbudzuje kritickosť jeho myslenia.



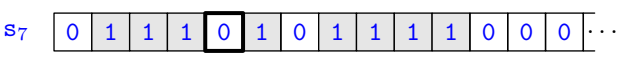
Alan Turing (1912—1954), britský matematik a logik
(http://en.wikipedia.org/wiki/Alan_Turing)

Preto považujeme za dôležité, aby sa každý učiteľ informatiky o Turingových strojoch dozvedel aspoň základné informácie, sú totiž nezastupiteľným článkom informatického vzdelania. Možná utilitaristická výhrada, že svojim žiakom nikdy o tejto problematike rozprávať nebude, neobstojí (mimochodom, možno ju vzniesť na veľkú väčšinu poznatkov získaných vysokoškolským štúdiom). Veď ak chce učiteľ (nielen informatiky) študentom poskytnúť sladké ovocie ozajstného poznania, musí mať sám hlboké korene.

2.2. Výpočty na Turingových strojoch

Hardvér Turingovho stroja má iba dve súčasti: „koľajnice“ — pevný, na jednu stranu nekonečný, rad (nerozoznateľných) štvorčekov, po ktorých sa (vždy po jednom štvorčeku) pohybuje jeden „vozík“, ktorý má šírku práve jedného štvorčeka. Každý štvorček pritom obsahuje vždy jeden z dvoch typov predmetov. Vo vozíku je (nevyčerpateľná) zásoba predmetov oboch typov, a tak (v prípade, že sa vozík nachádza v určitom stave) môžeme v tom štvorčeku, kde sa práve vozík nachádza, vymeniť predmet za iný.

Na konkrétnej podobe predmetov, pravdaže, nezáleží, budeme ich preto označovať iba symbolicky — informatikmi obľúbenými znakmi 0 a 1. Stavy vozíka tiež nebudeme popisovať slovami, ale číslami, presnejšie indexmi pevného symbolu s. Celú situáciu si teda môžeme znázorniť napríklad takýmto obrázkom (symbol s_7 vľavo je stav vozíka a ten je práve na sivom políčku):



Aby sme sa vyjadrovali odbornejšie, takúto situáciu budeme nazývať konfigurácia, koľajnice budeme nazývať páska a vozík hlava, pomenovanie stav si ponecháme.

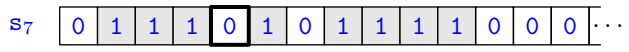
Dodajme, že nás budú zaujímať iba tie konfigurácie, v ktorých je počet jednotiek konečný, inými slovami, od istého miesta sú už vo všetkých ďalších štvorčekoch iba nuly.

Našou snahou bude, pravdaže, pohyb vozíka kontrolovať, a hlavne automatizovať. Na jeho ovládanie tak budeme používať isté inštrukcie. Je zaujímavé a dôležité, že všetky inštrukcie budú vyzerať v podstate rovnako, typickou ukážkou je napríklad $s_7 01 R s_8$. Každá z nich teda bude zložená z piatich písmen:

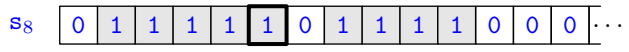
- Prvé dve budú potvrdzovať jej vhodnosť:
  - Prvý znak (v našom prípade s_7) sa musí zhodovať s aktuálnym stavom hlavy.
  - Druhý znak (u nás 0) musí byť rovnaký ako písmeno na štvorčeku s hlavou.

  V prípade splnenia oboch podmienok budeme hovoriť, že inštrukcia s aktuálnou konfiguráciou korešponduje. (Napríklad naša inštrukcia $s_7 01 R s_8$ korešponduje s konfiguráciou na obrázku.)
- Zvyšné tri písmená budú hovoriť, ako sa má konfigurácia postupne zmeniť:
  - V štvorčeku, na ktorom sa nachádza hlava, sa doterajší znak nahradí tretím znakom inštrukcie (v našom prípade 1). (O nahradzovaní môžeme hovoriť aj vtedy, keď sa druhý a tretí znak navzájom zhodujú, hoci sa v podstate nič nezmení.)
  - Štvrtý znak (ktorý môže byť len jedno z písmen L, N, alebo R) hovorí, ako sa má zmeniť poloha hlavy:
    - L: Hlava sa posunie o jedno políčko doľava.
    - N: Hlava ostáva na svojom pôvodnom mieste.
    - R: Hlava sa posunie o jedno políčko doprava (to je prípad našej inštrukcie).
  - Posledné písmeno inštrukcie (u nás s_8) hovorí, v akom stave bude hlava po vykonaní tejto inštrukcie.

Po aplikovaní našej inštrukcie $s_7 01 R s_8$ na vyššie uvedenú konfiguráciu



tak dostávame novú konfiguráciu:

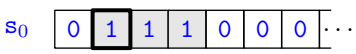


Ľahko vidieť, že pri každej konfigurácii vieme pripraviť inštrukciu, ktorú na ňu môžeme aplikovať — stačí z nej vyčítať stav a znak v sivom štvorčeku a doplniť ich jedným znakom 0 alebo 1, jedným posunom a jedným stavom. Po aplikácii takto vznikutej inštrukcie vznikne nová konfigurácia, tú však môžeme opäť zmeniť nejakou ďalšou inštrukciou, a takto celý proces podľa potreby opakujeme, až kým nevznikne konfigurácia, ktorú sme si predsavzali dosiahnuť na začiatku.

Naozaj však inštrukcie musíme pripravovať až „na poslednú chvíľu“, teda až vtedy, keď ich chceme aplikovať? Čo keby sme ich vytvorili ešte predtým? Z interaktívneho módu by sme tak prešli do programovacieho. Úlohou tu už nebude pripraviť jednu inštrukciu, ale rovno program, ktorý z nich bude zložený. Takýto program budeme nazývať Turingov stroj. Pre danú konfiguráciu sa potom z tohto systému vyberie tá inštrukcia, ktorá s danou konfiguráciou korešponduje.

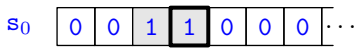
Uvedomme si, že pri takomto prístupe musíme vyhnúť situácii, keď s konfiguráciou korešponduje viac než jedna inštrukcia, v takom prípade by sme sa totiž nevedeli rozhodnúť, ktorú z nich použiť, a proces by sa tak stal nedeterministickým. Túto podmienku deterministickosti splníme ľahko — jednoducho budeme od Turingovho stroja požadovať, aby žiadne jeho dve inštrukcie nemali prvé dva znaky rovnaké.

Vezmime si napríklad Turingov stroj — systém inštrukcií $\{s_0 00 L s_3, s_0 10 R s_1, s_1 00 N s_2, s_1 11 R s_0\}$ (všimnime si, že spĺňa podmienku deterministickosti) a úvodná konfigurácia nech je

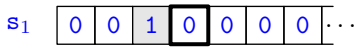


Sme v stave s_0 a aktuálny znak je 1. Korešpondujúca inštrukcia teda musí začínať dvojicou znakov $s_0 1$. Jediná taká (v našom stroji) je $s_0 10 R s_1$. Tá nám prikazuje prepísať znak v hlavou obsadenom políčku na 0, posunúť sa o jedno políčko doprava a zmeniť stav na s_1. Dostávame tak konfiguráciu

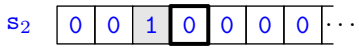
Pre túto konfiguráciu zasa hľadáme v našom Turingovom stroji inštrukciu, ktorá sa začína na $s_1 1$. Je to inštrukcia $s_1 11 R s_0$, po jej aplikovaní dostávame konfiguráciu



Opäť sme sa dostali do stavu s_0, pričom aktuálny znak na páske je 1, Jediná inštrukcia $s_0 10 R s_1$, ktorá s touto konfiguráciou korešponduje, už však bola použitá! Znamená to azda, že je už znova nepoužiteľná? Nie, pokojne ju môžeme, ba musíme aplikovať. Takže máme

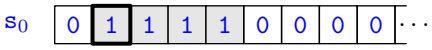


Teraz sme sa dostali do stavu s_1, avšak hlavou čítané písmeno je 0. Použijeme teda inštrukciu $s_1 00 N s_2$ a dostávame

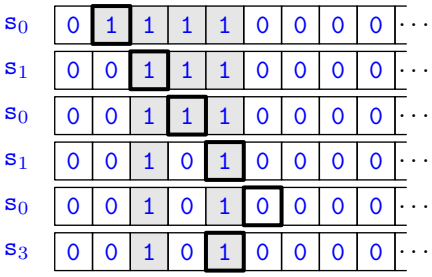


Dostali sme sa však zvláštnej situácie: Sme totiž v stave s_2, takže na túto konfiguráciu už nemôžeme aplikovať žiadnu inštrukciu z nášho zoznamu. A to znamená, že náš výpočet (teda postupnosť konfigurácií sledujúcich inštrukcie) sa končí. Výsledkom takéhoto výpočtu je potom počet jednotiek na páske, v našom prípade je to teda 1.

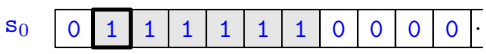
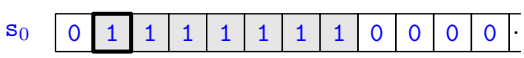
Skúsme spustiť ten istý Turingov stroj na inú počiatočnú konfiguráciu, a to



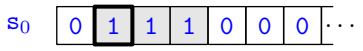
Po skúsenostiach z predchádzajúcej úlohy ľahko vidieť, že výpočet prebieha takto:



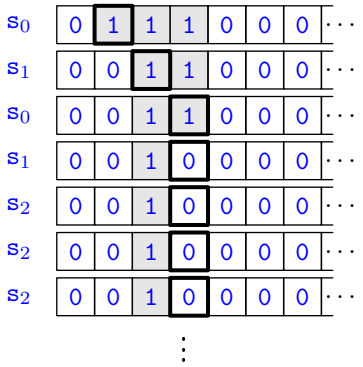
Tu sa výpočet končí, lebo (vzhľadom na stav s_3) už nemožno aplikovať žiadnu inštrukciu. Tentoraz je výsledok (čiže počet jednotiek na páske poslednej konfigurácie výpočtu) 2. Navyše si môžeme všimnúť, že stroj tu skočil svoj výpočet v stave s_3.

Úloha 2.1	Odkrokujte výpočet toho istého stroja na konfiguráciu  a na konfiguráciu  Ako závisia výsledok výpočtu a výsledný stav od vstupnej konfigurácie tohto typu?

Aby sme mali obraz o práci Turingovho stroja úplný, uveďme ešte jeden príklad: Aplikujme Turingov stroj $\{s_0 00 L s_3, s_0 10 R s_1, s_1 00 N s_2, s_1 11 R s_0, s_2 00 N s_2\}$ (t. j. k predchádzajúcemu stroju sme doplnili jednu novú inštrukciu) na konfiguráciu



Prvých pár krokov výpočtu je takýchto:



Vidíme, že posledné tri znázornené konfigurácie sú úplne rovnaké, a tak to zrejme bude pokračovať aj ďalej. To však znamená, že výpočet na tomto stroji z tejto konfigurácie sa nikdy neskončí.

Úloha 2.2	Nájdite úvodnú konfiguráciu, pre ktorú sa výpočet na predchádzajúcom Turingovom stroji skončí.
Úloha 2.3	Ako budú vyzerať predchádzajúce výpočty, ak v tomto Turingovom stroji namiesto inštrukcie $s_2 00 N s_2$ vezmeme inštrukciu $s_2 00 R s_2$?

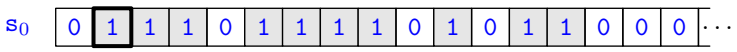
2.3. Turingovsky vypočítateľné funkcie

Už sme si povedali, aký má Turingov stroj výstup (buď počet jednotiek, alebo žiadny, ak sa nezastaví), treba ešte spomenúť, ako je to s jeho vstupom:

Turingov stroj vie, samozrejme, reagovať na každú vstupnú konfiguráciu. Aby však bol jeho výpočet rozumne interpretovateľný, naša vstupná konfigurácia mala pásku zloženú z tzv. blokov. Pod blokom kódujúcim číslo x budeme rozumieť súvislú konečnú postupnosť $x+1$ jednotiek, pred ktorými bude práve jedna nula, takže napríklad blok kódujúci číslo 3 bude vyzerať takto:

kým blok kódujúci číslo 0 bude

Páska kódujúca usporiadanú n-ticu $\langle x_1, x_2, \ldots, x_n \rangle$ (budeme ju nazývať bloková páska) potom bude združením blokov kódujúcich postupne čísla x_1, x_2, ..., x_n, pričom zvyšok pásky je vyplnený nulami. Pod štandardnou blokovou konfiguráciou potom budeme rozumieť konfiguráciu, ktorej páska je bloková, hlava je nastavená na prvej jednotke prvého bloku a stav je s_0. Takže štandardná bloková konfigurácia kódujúca štvoricu čísel $\langle 2, 3, 0, 1 \rangle$ bude vyzerať takto:



Keďže už takto vieme číselne interpretovať aj vstupy, aj výstup Turingovho stroja, môžeme tento stroj chápať ako prostriedok na počítanie funkcie, ktorej definičným oborom sú (možno nie niektoré) usporiadané n-tice prirodzených čísel (pri danom, pevnom počte n) a obor hodnôt je podmnožinou množiny prirodzených čísel:

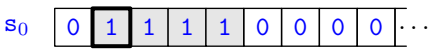
Budeme hovoriť, že Turingov stroj T počíta funkciu f s pevným kladným počtom vstupov n, ak pre každú usporiadanú n-ticu $\langle x_1, x_2, \ldots, x_n \rangle$ prirodzených čísel platí:

- Ak na stroji T existuje (zrejme jediný) konečný výpočet, ktorého úvodná konfigurácia je štandardná bloková a kóduje n-ticu $\langle x_1, x_2, \ldots, x_n \rangle$, tak hodnota $f(x_1, \ldots, x_n)$ je rovná počtu jednotiek na páske poslednej konfigurácie tohto výpočtu.
- Ak taký konečný výpočet neexistuje, tak hodnota $f(x_1, \ldots, x_n)$ nie je definovaná.

Pod turingovsky vypočítateľnou funkciou budeme rozumieť takú funkciu, pre ktorú existuje Turingov stroj, ktorý ju počíta.

Po príklad sa vráťme k Turingovmu stroju $\{\mathtt{s_0 00 L s_3}, \mathtt{s_0 10 R s_1}, \mathtt{s_1 00 N s_2}, \mathtt{s_1 11 R s_0}\}$, ktorý sme ukážkovo spúšťali na štandardných blokových konfiguráciách

a



V prvom prípade bol teda vstup $\langle 2 \rangle$ a výstup 1, v druhom bol vstup $\langle 3 \rangle$ a výstup 2. Riešením úlohy 2.1 sme sa dozvedeli, že pre vstup $\langle 2m \rangle$ (teda v prípade, že máme na páske $2m+1$ jednotiek), je výstupom číslo m, kým pre vstup $2m+1$ (teda v prípade, že máme na páske $2m+2$ jednotiek), je výstupom číslo $m+1$. Znamená to teda, že tento Turingov stroj počíta funkciu f s jedným argumentom, pre ktorú platí:

$$f(x_1) = \begin{cases} k, & \text{ak } x_1 = 2k, \\ k+1, & \text{ak } x_1 = 2k+1. \end{cases}$$

(Inými slovami, platí $f(x) = \lceil \frac{x}{2} \rceil$.)

Jemne upravený Turingov stroj $\{\mathtt{s_0 00 L s_3}, \mathtt{s_0 10 R s_1}, \mathtt{s_1 00 N s_2}, \mathtt{s_1 11 R s_0}, \mathtt{s_2 00 N s_2}\}$, ktorý bol spustený na konfiguráciu

zasa počítal funkciu g s jedným argumentom, pre ktorú platí:

$$g(x_1) \begin{cases} = k, & \text{ak } x_1 = 2k, \\ \text{nie je definované}, & \text{ak } x_1 = 2k+1. \end{cases}$$

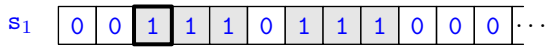
Všimnime si, že ďalšia malá úprava tohto stroja $\{\mathtt{s_0 00 L s_3}, \mathtt{s_0 10 R s_1}, \mathtt{s_1 00 N s_2}, \mathtt{s_1 11 R s_0}, \mathtt{s_2 00 N s_2}\}$, ktorá bola použitá v úlohe 2.3, počíta tú istú funkciu g. Vidíme teda, že jednu funkciu môže počítať i viacero rôznych Turingových strojov. Avšak naopak to neplatí — každý Turingov stroj počíta práve jednu funkciu s daným počtom argumentov.

Úloha 2.4	Akú funkciu s 2 argumentmi počíta Turingov stroj $\{\mathtt{s_0 00 L s_3}, \mathtt{s_0 10 R s_1}, \mathtt{s_1 00 N s_2}, \mathtt{s_1 10 R s_0}\}$?
Úloha 2.5	Akú funkciu s 2 argumentmi počíta Turingov stroj $\varnothing$?

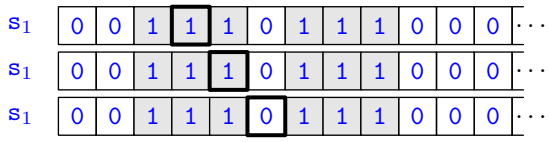
Doteraz sme hľadali funkcie, ktoré hľadajú dané Turingove stroje. Úloha je však veľmi často úplne opačná — k danej funkcii treba nájsť Turingov stroj, ktorý ju počíta.

Skúsme teda nájsť (čiže naprogramovať) Turingov stroj, ktorý počíta obvyklú funkciu Súčet. Tá má dva argumenty, ako vstup výsledného Turingovho stroja teda bude slúžiť ľubovoľná dvojica prirodzených čísel. Zároveň je táto funkcia totálna (teda jej definičný obor je celé $\mathbb{N} \times \mathbb{N}$), Turingov stroj teda musíme naprogramovať tak, aby zastal pre každý vstup. Vieme, že pri vstupe $\langle x_1, x_2 \rangle$ je na začiatku na páske $(x_1 + 1) + (x_2 + 1)$ jednotiek, čo je o dve viac, než naozaj treba. Úlohou teda bude dve z týchto jednotiek zmeniť na nuly. Ktoré to budú? Máme, pravdaže, viacero možností. Zdanlivo najjednoduchšia je vymazať prvé dve jednotky, vec však má háčik — čo ak bude prvý vstup 0? Druhá vymazaná jednotka bude musieť byť v takom prípade z druhého bloku, a navyše budeme musieť testovať, či tento prípad nastáva, alebo nie. Tomuto testovaniu sa vyhneme, ak budeme (spolu s prvou jednotkou prvého bloku, na ktorej je hlava na začiatku priamo nastavená) mazať prvú jednotku druhého bloku (čiže nahradzovať ju nulou) pre každý vstup. Vezmime si teda nejaký netriviálny vstup, napríklad $\langle 3, 2 \rangle$ (na konci má teda na páske zostať Súčet$(3, 2) = 3 + 2 = 5$ jednotiek):

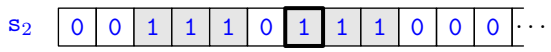
Najprv, samozrejme, vymažeme jednotku, na ktorej je nastavená hlava. Pritom sa posunieme na ďalšiu jednotku. Pokiaľ ide o stav, bolo by fatálnou chybou nezmeniť ho, príslušná inštrukcia $\mathtt{s_0 1 0 R s_0}$ by sa totiž nevykonala iba raz, ale nenávratne by vymazala celý prvý blok jednotiek. Ten však v celkovom výsledku potrebujeme, a nemôžeme ho teda v žiadnom prípade stratiť. Preto zmeníme stav na iný, povedzme na $\mathtt{s_1}$. Po vykonaní príslušnej inštrukcie $\mathtt{s_0 1 0 R s_1}$ sa tak dostávame do konfigurácie



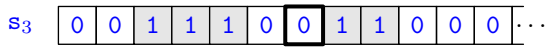
Teraz je naším cieľom prejsť celým zvyšným blokom jednotiek. V našom prípade sú tri, na prechod by preto stačilo použiť tri ďalšie inštrukcie. Uvedomme si, že náš program túto hodnotu x_1 vo všeobecnosti nepozná (vstupná konfigurácia je predsa zadávaná až neskôr), musí si preto poradiť s ľubovoľnou jej hodnotou. Našťastie existuje elegantné riešenie — stačí použiť inštrukciu $\mathtt{s_1 1 1 R s_1}$. Ako ľahko vidieť, po jej prvom vykonaní sme opäť v rovnakom stave $\mathtt{s_1}$ a hlava je opäť na znaku $\mathtt{1}$, preto sa táto inštrukcia použije vzápätí znova. Ľahko vidieť, že takto sa to bude opakovať v našom prípade trikrát (vo všeobecnosti x_1-krát):



Teraz sa už jednoducho posunieme z deliacej nuly na prvú jednotku druhého bloku, pričom musíme zmeniť stav, aby sme nechtiac neprešli cez celý druhý blok až na jeho koniec. Vykonáme teda inštrukciu $\mathtt{s_1 0 0 R s_2}$ a dostávame:



Napokon stačí vymazať jednotku, na ktorej je hlava nastavená, a keďže sa tým pádom získa správny počet jednotiek, zmeníme stav na doposiaľ nepoužitý, a to inštrukciou $\mathtt{s_2 1 0 N s_3}$:



Na páske je požadovaný počet 5 (vo všeobecnosti $x_1 + x_2$) jednotiek, na túto konfiguráciu už teda nechceme aplikovať žiadnu inštrukciu. Vyhovujúci Turingov stroj je preto $\{\mathtt{s_0 1 0 R s_0}, \mathtt{s_1 1 1 R s_1}, \mathtt{s_1 0 0 R s_2}, \mathtt{s_2 1 0 N s_3}\}$.

Úloha 2.6	Odkrokujte tento Turingov stroj pre dvojice $\langle 2,3\rangle$, $\langle 1,3\rangle$, $\langle 0,3\rangle$ a $\langle 0,0\rangle$.

Zostrojme ešte Turingov stroj, ktorý počíta funkciu Rozdiel. Tá má tiež dva argumenty, ako vstup výsledného Turingovho stroja teda bude slúžiť ľubovoľná dvojica prirodzených čísel. Na rozdiel od predchádzajúcej však táto funkcia totálna nie je – veď napr. hodnota $\texttt{Rozdiel}(1,2)$ nie je (na prirodzených číslach) definovaná. Znamená to, že pre niektoré vstupy (presnejšie práve tie $\langle x_1, x_2\rangle$, pre ktoré platí $x_1 < x_2$) sa výpočet na Turingovom stroji nesmie zastaviť. Myšlienka Turingovho stroja, ktorý budeme konštruovať, je založená na rovnosti (pre kladné x_1 a x_2)

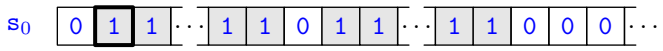
$$\texttt{Rozdiel}(x_1, x_2) = \texttt{Rozdiel}(x_1 - 1, x_2 - 1)$$

(v prípade, že nie je jedna z nich definovaná, ani druhá nie je definovaná). Budeme striedavo mazať jednotky (teda prepisovať ich nulami) z oboch blokov (pričom začneme druhým), a to až dovtedy, kým jeden z blokov nezanikne.

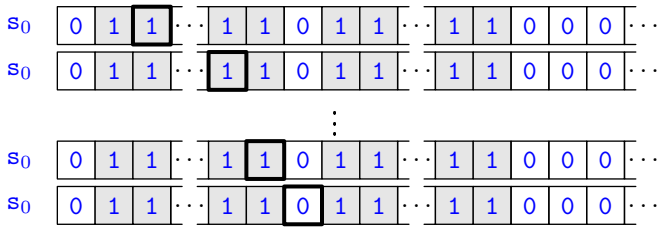
- V prípade, že skôr zanikne druhý blok, platí $x_1 \leq x_2$, a teda výsledkom počet jednotiek v (už redukovanom) prvom bloku, avšak zmenšený o 1 (keďže sme začali mazať z druhého bloku, z prvého sme zatiaľ vymazali o jednu jednotku menej).
- V prípade, že skôr zanikne prvý blok, platí $x_1 < x_2$, a teda výpočet na stroji sa nemá zastaviť.

Ešte sa musíme rozhodnúť, ktoré jednotky z oboch blokov budeme mazať. Všetky možnosti tu síce vedú k cieľu, ak však budeme mazať prvú jednotku z prvého bloku a poslednú z druhého, zachováme si výhodu existencie jedinej deliacej nuly.

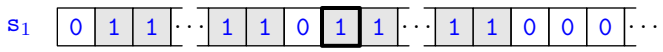
Majme teda konfiguráciu



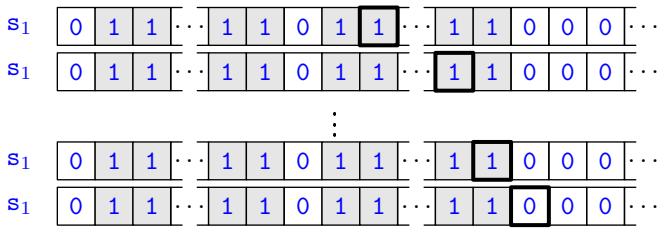
Najprv sa musíme presunúť na poslednú jednotku druhého bloku. Na prechod jednotkami prvého bloku stačí niekoľkokrát použitá inštrukcia $\texttt{s}_0\texttt{11Rs}_0$,



Cez deliacu nulu prejdeme pomocou inštrukcie $\texttt{s}_0\texttt{00Rs}_1$:



Druhý blok prejdeme pomocou inštrukcie $\texttt{s}_1\texttt{11Rs}_1$.



Jedným krokom späť (pomocou inštrukcie $s_1 00 L s_2$) sa dostaneme na požadovanú poslednú jednotku druhého bloku:

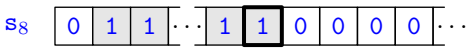
Vymažeme jednotku pod hlavou a posunieme sa doľava (pomocou inštrukcie $s_2 10 L s_3$):

Teraz musíme skontrolovať, či sme týmto vymazaním nespôsobili úplný zánik druhého bloku. Toto testovanie urobíme pomocou dvojice inštrukcií $s_3 00 L s_8$ a $s_3 11 L s_4$:

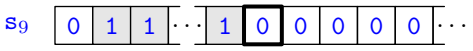
- Ak je znak hlavy 0, znamená to, že druhý blok sa minul a sme na deliacej nule:

  Ako sme už povedali, v takom prípade ešte treba vymazať jednu jednotku z prvého bloku. Vykonáme teda najprv inštrukciu $s_3 00 L s_8$ (stavy s_5, s_6 a s_7 ešte budeme potrebovať):

  a potom inštrukciu $s_8 10 N s_9$.

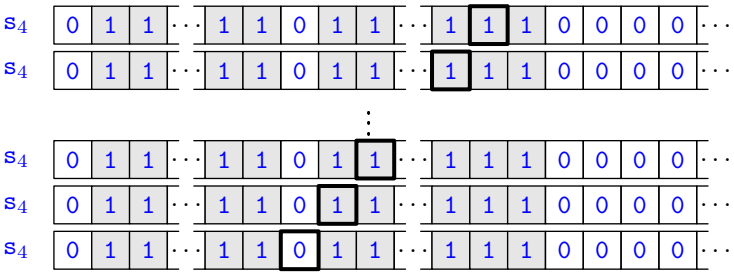
  

  Na páske tak ostalo práve $x_1 - x_2$ jednotiek, čo sme potrebovali.

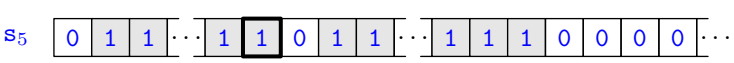
- Ak je znak hlavy 1, znamená to, že druhý blok sa ešte neminul a treba sa teda presunúť na prvú jednotku prvého bloku. Po vykonaní inštrukcie $s_3 11 L s_4$ tak dostávame:

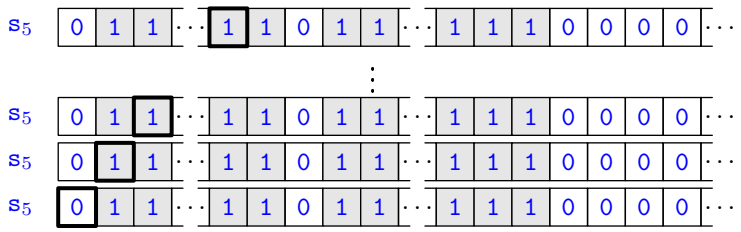
  Cez druhý blok prejdeme niekoľkonásobne opakovaným vykonaním inštrukcie $s_4 11 L s_4$:

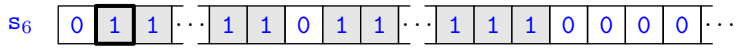
  Cez deliacu nulu jednorazovo inštrukciou $s_4 00 L s_5$:

  A cez prvý blok niekoľkonásobným použitím inštrukcie $s_5 11 L s_5$:



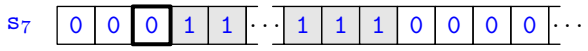
Inštrukciou $s_5 00 R s_6$ sa vrátime sa na prvú jednotku prvého bloku:



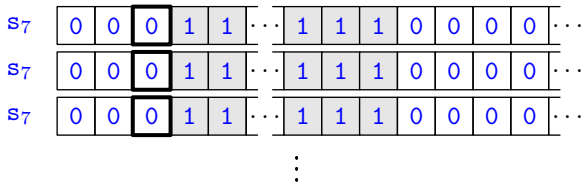
A vymažeme ju inštrukciou $s_6 1 0 R s_7$ (pozor, nie $s_6 1 0 R s_6$, to by sme stratili celý prvý blok!):

Teraz však musíme otestovať, či sme týmto vymazaním nespôsobili zánik (tentoraz prvého) bloku:

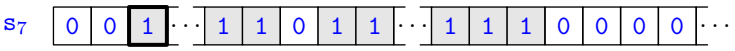
- Ak je na mieste hlavy nula, prvý blok jednotiek sa práve minul:

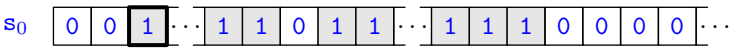
  a vtedy, ako vieme, sa výpočet nemá skončiť. Tu nám pomôžeme inštrukcia $s_7 00 N s_7$, ktorej opakované vykonávanie sa už nikdy neskončí:

- Ak je na mieste hlavy jednotka, prvý blok jednotiek sa ešte neskončil:

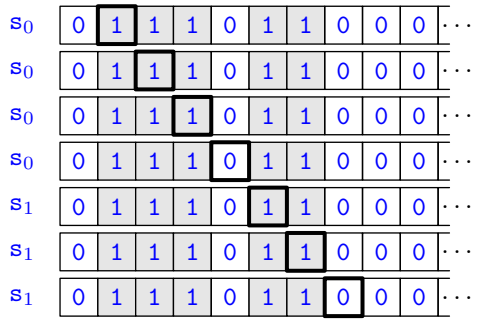
  V takom prípade sa však stačí prepnúť do stavu s_0 inštrukciou $s_7 11 N s_0$:

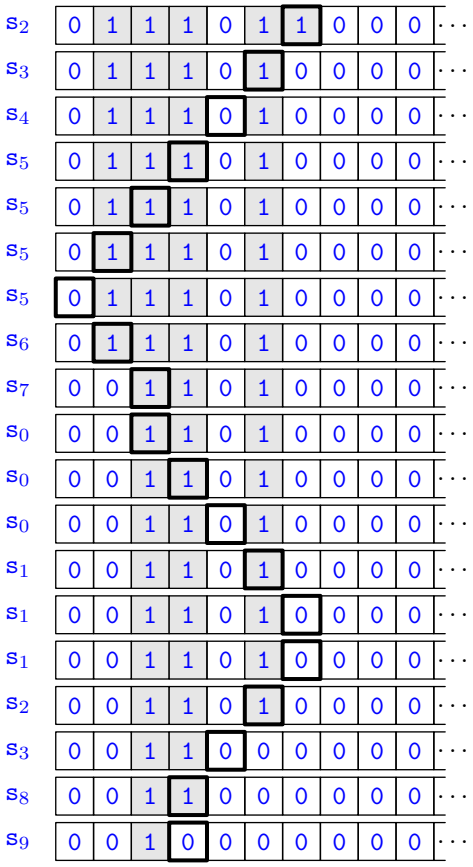
  

  Takto sme sa totiž dostali do situácie, v ktorej sme už raz boli, jediným rozdielom je, že v oboch blokoch ubudlo po jednej jednotke. A celý cirkus sa môže začať nanovo.

Výsledný Turingov stroj je teda $\{s_0 11 R s_0, s_0 00 R s_1, s_1 11 R s_1, s_1 00 L s_2, s_2 10 L s_3, s_3 00 L s_8, s_8 10 N s_9, s_3 11 L s_4, s_4 00 L s_5, s_5 11 L s_5, s_5 00 R s_6, s_6 10 R s_7, s_7 00 N s_7, s_7 11 N s_0\}$.

Odkrokujme ho pre vstup $\langle 2, 1 \rangle$:





Aj na tomto odkrokovaní sme si mohli všimnúť, že napriek svojej extrémnej jednoduchosti sú Turingove stroje schopné simulovať jednak vetvenie (v prípade predchádzajúceho stroja sa tak dialo v stavoch s_3 a s_7), jednak cyklus (keď sme sa zo stavu s_7 opäť dostali do stavu s_0), ktorý sa končil testom (šlo teda o cyklus neprávom zaznávaného typu „repeat-until“).

Úloha 2.7	Odkrokujte tento Turingov stroj pre vstupy $\langle 2,0\rangle$, $\langle 1,1\rangle$, $\langle 0,0\rangle$ a $\langle 1,2\rangle$.
Úloha 2.8	Vytvorte Turingove stroje počítajúce funkcie: • `Minimum` (vracia najmenšie z 2 čísel), • `Maximum` (vracia najväčšie z 2 čísel), • `Vzdialenosť` (vracia absolútnu hodnotu rozdielu 2 čísel).

2.4. Rekurzívne funkcie

Ukázali sme zopár príkladov turingovsky vypočítateľných funkcií. Vzniká však prirodzená otázka, či existuje aj opačný príklad — funkcia (samozrejme, definovaná len na nejakej podmnožine tíc prirodzených čísel s hodnotami tiež z $\mathbb{N}$), ktorá by turingovsky vypočítateľná nebola. Odpoveď poskytuje — pre niekoho azda prekvapivo — matematika, presnejšie teória vypočítateľnosti. Jedným z jej cieľov je definovanie množiny tzv. rekurzívnych funkcií, o ktorej sa neskôr ukáže, že je identická s množinou (v predchádzajúcej stati spomínaných) turingovsky vypočítateľných funkcií. Táto množina je definovaná induktívne, čiže v týchto dvoch etapách:

1. explicitne vyberieme niekoľko základných prvkov,
2. popíšeme niekoľko postupov, pomocou ktorých budeme môcť z jednoduchších prvkov vytvoriť zložitejšie.

V našom prípade množiny rekurzívnych funkcií budú hrať úlohu základných prvkov funkcie nasledujúcich troch typov:

- Funkcia Z (z anglického „zero“ – nula) bude mať definičný obor $\mathbb{N}$ a bude definovaná vzťahom
$$\mathrm{Z}(x) = 0.$$
Ide teda o nulovú funkciu, čiže napríklad $\mathrm{Z}(4) = 0$ alebo $\mathrm{Z}(123) = 0$.

- Funkcia S (z anglického „successor“ – nasledovník) bude mať tiež definičný obor $\mathbb{N}$ a bude pre ňu platiť
$$\mathrm{S}(x) = x + 1.$$
Takže napríklad $\mathrm{S}(4) = 5$ alebo $\mathrm{S}(123) = 124$.

- Kým prvé dva typy majú po jedinej funkcii, posledný ich bude obsahovať nekonečne veľa. Pre každé kladné prirodzené číslo n a každé i z množiny $\{1, 2, \ldots, n\}$ definujme funkciu P_i^n (z anglického „projection“ – projekcia) s definičným oborom $\mathbb{N}^n$ takto:
$$\mathrm{P}_i^n(x_1, x_2, \ldots, x_n) = x_i.$$
Konkrétne napríklad pre $n = 3$ a $i = 2$ teda máme projekciu P_2^3, pre ktorú platí $\mathrm{P}_2^3(x_1, x_2, x_3) = x_2$ (alebo rovnako dobre $\mathrm{P}_2^3(a, b, c) = b$), čiže napríklad $\mathrm{P}_2^3(34, 56, 78) = 56$ alebo $\mathrm{P}_2^3(4, 0, 3) = 0$. Analogicky trebárs $\mathrm{P}_4^4(4, 0, 3, 5) = 5$ alebo $\mathrm{P}_1^2(9, 4) = 9$. Za osobitnú zmienku stojí špeciálny prípad P_1^1, lebo $\mathrm{P}_1^1(x) = x$, a teda ide vlastne o identitu.

Všimnime si, že všetky tieto funkcie sú totálne, t. j. sú definované pre všetky prípustné vstupy – Z a S pre každé prirodzené číslo a P_i^n pre každú n-ticu prirodzených čísel.

Prvú etapu definície rekurzívnych funkcií sme tým uzavreli, poďme na druhú:

Najprv niekoľko príkladov:

- Všimnime si, čo sa stane, ak do funkcie S „dosadíme“ (t. j. „substituujeme“) funkciu Z (presnejšie – jej funkčnú hodnotu). Pre ľubovoľné prirodzené číslo x potom dostávame $\mathrm{S}(\mathrm{Z}(x)) = \mathrm{S}(0) = 1$. Vytvorili sme tak vlastne konštantnú funkciu C_1 vracajúcu pre každý vstup hodnotu 1.

- Do funkcie S však môžeme substituovať aj samu seba. Dostávame tak $\mathrm{S}(\mathrm{S}(x)) = \mathrm{S}(x + 1) = x + 2$, vytvorili sme teda funkciu, ktorá pre daný vstup vráti hodnotu o 2 väčšiu.

- Ak do S substituujeme trebárs P_2^3, dostávame $\mathrm{S}(\mathrm{P}_2^3(x_1, x_2, x_3)) = \mathrm{S}(x_2) = x_2 + 1$. Uvedomme si však, že výsledná funkcia má za vstup nie jediné prirodzené číslo, ale usporiadanú trojicu. Jej výstupom je potom hodnota druhej zložky tohto vstupu zväčšená o 1.

- Substituovať môžeme napríklad aj do P_2^3. Tá však má tri argumenty, substituovať teda musíme hodnoty troch funkcií (avšak v tom istom bode!). Ak to budú trebárs $\mathrm{Z}(x)$, $\mathrm{S}(x)$ a opäť $\mathrm{S}(x)$, dostávame $\mathrm{P}_2^3(\mathrm{Z}(x), \mathrm{S}(x), \mathrm{S}(x)) = \mathrm{P}_2^3(0, x+1, x+1) = x+1$. Výsledná funkcia (všimnime si, že to nie je nič iné ako S) má teda tiež jeden vstup.

Vo všeobecnosti budeme pod substitúciou rozumieť proces, do ktorého vstupuje jedna funkcia h, ktorej vstupy sú z množiny $\mathbb{N}^k$, a k ďalších funkcií g_1, g_2, …, g_k, ktorých vstupy sú z množiny $\mathbb{N}^n$. Výsledkom tohto procesu bude funkcia f, ktorej vstupy sú z množiny $\mathbb{N}^n$ (t. j. rovnako ako pri funkciách g_i) a pre ktorú platí vzťah

$$f(x_1, x_2, \ldots, x_n) = h(g_1(x_1, x_2, \ldots, x_n), g_2(x_1, x_2, \ldots, x_n), \ldots, g_k(x_1, x_2, \ldots, x_n))$$

(pričom ľavá strana je definovaná práve vtedy, keď je definovaná pravá strana). V takom prípade budeme hovoriť, že funkcia f vznikla substitúciou z funkcií h a g_1, g_2, …, g_k.

Napríklad v prvom vyššie uvedenom príklade hrá úlohu funkcie h (do ktorej substituujeme) funkcia S, takže počet jej argumentov k je 1. Úlohu funkcie g_1 (ktorej hodnotu do h substituujeme) hrá funkcia Z, takže počet jej argumentov n je 1. Úlohu výslednej funkcie f potom hrá spomínaná konštantná funkcia C_1 (ktorá má tiež jeden argument), lebo naozaj platí $\mathrm{C}_1(x_1) = \mathrm{S}(\mathrm{Z}(x_1))$ (obe strany sa totiž rovnajú 1). Môžeme teda povedať, že funkcia C_1 vznikla substitúciou z funkcií Z a S.

Úloha 2.9	Rozoberte podobne zvyšné tri príklady.

Pre každé prirodzené číslo m teraz definujme konštantnú funkciu C_m s jedným argumentom vzťahom

$$\mathrm{C}_m(x) = m$$

(takže C_0 je vlastne Z, C_1 sme už mali definované a napríklad $\mathrm{C}_7(x) = 7$). Potom platí

$$\mathrm{C}_2(x_1) = 2 = \mathrm{S}(1) = \mathrm{S}(\mathrm{C}_1(x_1)),$$

takže funkcie C_2, S a C_1 spĺňajú vzorec z definície substitúcie. To teda znamená, že funkcia C_2 vznikla substitúciou z funkcií S a C_1. Analogicky

$$\mathrm{C}_3(x_1) = 3 = \mathrm{S}(2) = \mathrm{S}(\mathrm{C}_2(x_1)),$$

takže funkcia C_3 vznikla substitúciou z funkcií S a C_2. Môžeme to zrejme úplne zovšeobecniť: Pre ľubovoľné prirodzené číslo m platí

$$\mathrm{C}_{m+1}(x_1) = m + 1 = \mathrm{S}(m) = \mathrm{S}(\mathrm{C}_m(x_1)),$$

funkcia C_{m+1} teda vznikla substitúciou z funkcií S a C_m.

Pokračujme ďalej: Pre každé kladné prirodzené číslo n a každé prirodzené číslo m definujme konštantnú funkciu C_m^n s n argumentmi vzťahom

$$\mathrm{C}_m^n(x_1, x_2, \ldots, x_n) = m$$

(napríklad $\mathrm{C}_9^3(a, b, c) = 9$). Špeciálne funkcia C_m^1 je teda práve funkcia C_m. Ľahko vidieť, že platí

$$\mathrm{C}_m^n(x_1, x_2, \ldots, x_n) = m = \mathrm{C}_m(x_1) = \mathrm{C}_m(\mathrm{P}_1^n(x_1, x_2, \ldots, x_n)),$$

takže funkcia C_m^n vznikla substitúciou z funkcií C_m a P_1^n.

Uvedomme si, že kým na začiatku sme mali len niekoľko základných funkcií, pomocou substitúcie vytvorili sadu nových (C_m, resp. C_m^n). Podobne by sme mohli pokračovať i ďalej, a dostali tak niekoľko ďalších typov funkcií, z ktorých si však drvivá väčšina vzhľadom na svoju nezaujímavosť nezaslúži ani len meno. Samotná substitúcia nám už totiž principiálne nič nové neprinesie, budeme musieť použiť zbraň väčšieho kalibru — postup, ktorý dal funkciám, s ktorými pracujeme, i meno.

Ide o proces zvaný rekurzia. Pre dané kladné prirodzené číslo n sú jeho vstupom dve funkcie: h, ktorej potenciálnymi vstupmi sú n-tice prirodzených čísel, a g, ktorej vstupmi sú $(n+2)$-tice prirodzených čísel. Výstupom rekuzie je funkcia, ktorej vstupy sú $(n+1)$-tice prirodzených čísel a pre ktorú platia (tentoraz dva) vzťahy:

1° $f(x_1, \ldots, x_n, 0) = h(x_1, \ldots, x_n)$.

2° $f(x_1, \ldots, x_n, y+1) = g(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))$.

(Opäť pritom platí, že ľavé strany sú definované práve vtedy, keď sú definované pravé.) V takom prípade budeme hovoriť, že funkcia f vznikla rekurziou z funkcií h a g.

Všimnime si oprávnenosť názvu rekurzia — na výpočet hodnoty funkcie f v bode (s poslednou zložkou) $y+1$ používame hodnotu tej istej funkcie f v bode (s poslednou zložkou) y (nemenné hodnoty x_1, …, x_n môžeme právom chápať ako parametre).

Rekuziu môžeme (ba musíme) použiť na vytvorenie funkcie Súčet (definovanej, samozrejme, vzťahom $\text{Súčet}(a,b) = a+b$). Ak teda má Súčet hrať úlohu f z definície rekurzie, ktoré funkcie budú hrať úlohy g a h? Uvedomme si, že v takom prípade pre počet argumentov funkcie f, t. j. Súčet, platí $n+1=2$, a teda $n=1$. Naše dva vzorce majú potom tvar:

1° $\text{Súčet}(x_1, 0) = h(x_1)$.

2° $\text{Súčet}(x_1, y+1) = g(x_1, y, \text{Súčet}(x_1, y))$.

Prvý vzťah teda hovorí, že

$$h(x_1) = \text{Súčet}(x_1, 0) = x_1 + 0 = x_1,$$

čiže funkcia h nie je nič iné ako identita P_1^1. Podľa druhého vzťahu má platiť

$$g(x_1, y, \text{Súčet}(x_1, y)) = \text{Súčet}(x_1, y+1) = x_1 + (y+1) = (x_1+y)+1 = \text{Súčet}(x_1, y)+1.$$

Stačí teda vziať funkciu g definovanú vzťahom $g(a,b,c) = c+1$, potom skutočne platí požadovaný vzťah $g(x_1, y, \text{Súčet}(x_1, y)) = \text{Súčet}(x_1, y)+1$, a tak funkcia Súčet vznikne rekurziou z funkcií P_1^1 a g. Máme tu však ešte jeden dlh – funkciu g sme zatiaľ nevyrobili. Hneď ho však splatíme:

$$g(a,b,c) = c+1 = \text{S}(c) = \text{S}(\text{P}_3^3(a,b,c)).$$

Funkcia g teda vznikla substitúciou zo (základných) funkcií S a P_3^3. Takto sme (hoci na viac krokov) vytvorili funkciu Súčet zo základných funkcií pomocou substitúcie a rekurzie.

Všimnime si ešte dočasný charakter označenia funkcie g, ktorý je (na rozdiel od trvalého označenia funkcií Z, S, P_i^n, či Súčet) vyjadrený kurzívou. Znamená to, že po ukončení príkladu sa takéto dočasné kurzívové označenie uvoľňuje a môže byť ďalej použité v inom zmysle.

Preto napríklad na označenie množiny prirodzených čísel zásadne používame označenie $\mathbb{N}$, a nie kurzívové N.

Ďalším príkladom funkcie, na konštrukciu ktorej je nevyhnutná rekurzia, je Súčin (pričom $\text{Súčin}(a,b) = ab$). V jeho prípade majú definičné vzorce rekurzie tvar:

1° $\text{Súčin}(x_1, 0) = h(x_1)$.

2° $\text{Súčin}(x_1, y+1) = g(x_1, y, \text{Súčin}(x_1, y))$.

Prvý vzťah hovorí, že $h(x_1) = \text{Súčin}(x_1, 0) = x_1 \cdot 0 = 0$, t. j. $h = \text{Z}$. Podľa druhého $g(x_1, y, \text{Súčin}(x_1, y)) = \text{Súčin}(x_1, y+1) = x_1(y+1) = x_1 y + x_1 = \text{Súčin}(x_1, y) + x_1$, takže stačí vziať g také, že $g(a,b,c) = c+a$. Pre takéto g platí

$$g(a,b,c) = c+a = \text{Súčet}(c,a) = \text{Súčet}(\text{P}_3^3(a,b,c), \text{P}_1^3(a,b,c)),$$

takže funkcia g vznikla substitúciou z (už skonštruovaných, resp. základných) funkcií Súčet a P_3^3, P_1^3. Funkcia Súčin potom vznikla rekurziou z funkcií Z a tejto g.

Úloha 2.10	Skonštruujte z už vytvorených funkcií pomocou substitúcie a rekurzie funkciu Mocnina, pre ktorú platí $\text{Mocnina}(a,b) = a^b$ (špeciálne $\text{Mocnina}(0,0) = 0^0 = 1$).

Aby sme sa mohli vyjadrovať jednoduchšie, budeme funkcie, ktoré vzniknú zo základných použitím konečného počtu substitúcií a/alebo rekurzií, nazývať primitívne rekurzívne. Vieme už teda, že medzi ne patria základné funkcie Z, S, P_i^n, ale i C_m^n, Súčet, Súčin, či Mocnina. Všimnime si, že každá primitívne rekurzívna funkcia je totálna. Nie je však nič prekvapujúce, lebo, ako sme už raz zdôraznili, všetky základné funkcie sú totálne, a, ako ľahko nahliadneme, ako substitúcia, tak rekurzia totalitu zachovávajú.

Vráťme sa ešte k rekurzii: Zatiaľ sme pri nej požadovali, aby bol počet n parametrov $x_1, \ldots, x_n$ kladný. Dá sa však ukázať, že na to, aby bola výsledná funkcia primitívne rekurzívna, táto podmienka nie je nutná. V prípade neprítomnosti týchto parametrov podmienka 1° úplne vypadne a druhá má tvar

$$f(y+1) = g(y, f(y)).$$

Takže ak je g primitívne rekurzívna, taká bude aj f.

Pomocou tejto vlastnosti ľahko nahliadneme primitívnu rekurzivitu niektorých dôležitých, hoci nenápadných, funkcií. Jednou z nich je funkcia KváziPredchodca, ktorá vznikne dodefinovaním netotálnej funkcie inverznej k funkcii S v bode 0 takto:

$$\text{KváziPredchodca}(a) = \begin{cases} 0, & \text{ak } a = 0, \\ a - 1, & \text{ak } a > 0. \end{cases}$$

Keďže pre každé prirodzené číslo y platí $y + 1 > 0$, dostávame

$$\text{KváziPredchodca}(y + 1) = y = \text{P}_1^2(y, \text{KváziPredchodca}(y)),$$

a pretože funkcia P_1^2 je primitívne rekurzívna, taká je aj funkcia KváziPredchodca.

Úloha 2.11	Ukážte podobne, že aj funkcie Signum a AntiSignum definované vzťahmi $$\text{Signum}(a) = \begin{cases} 0, & \text{ak } a = 0, \\ 1, & \text{ak } a > 0, \end{cases} \quad \text{AntiSignum}(a) = \begin{cases} 1, & \text{ak } a = 0, \\ 0, & \text{ak } a > 0, \end{cases}$$ sú primitívne rekurzívne.
Úloha 2.12	Ukážte, že funkcia KváziRozdiel, ktorá vznikne rozšírením netotálnej funkcie Rozdiel takto: $$\text{KváziRozdiel}(a, b) = \begin{cases} a - b, & \text{ak } a > b, \\ 0, & \text{ak } a \leq b, \end{cases}$$ je primitívne rekurzívna.
Návod	V druhom kroku rekurzie využite vzťah $\text{KváziRozdiel}(x_1, y + 1) = \text{KváziPredchodca}(\text{KváziRozdiel}(x_1, y))$.
Úloha 2.13	Ukážte, že logická (t. j. majúca hodnoty len 0 a 1) funkcia JeVäčšieAko definovaná vzťahom $$\text{JeVäčšieAko}(a, b) = \begin{cases} 1, & \text{ak } a > b, \\ 0, & \text{ak } a \leq b, \end{cases}$$ je primitívne rekurzívna.
Návod	Využite vzťah $\text{JeVäčšieAko}(a, b) = \text{Signum}(\text{KváziRozdiel}(a, b))$.
Úloha 2.14	Definujte analogicky logické funkcie • JeMenšieAko, • JeMenšieAleboRovnakéAko, • JeVäčšieAleboRovnakéAko, • JeRovnakéAko, • JeRôzneOd a ukážte, že sú primitívne rekurzívne.

Povedali sme už, že každá primitívne rekurzívna funkcia je totálna, ale i to, že napríklad funkcia Rozdiel totálna nie je (lebo trebárs hodnota $\text{Rozdiel}(1, 2)$ nie je definovaná). To teda znamená, že táto funkcia nemôže byť primitívne rekurzívna. Našťastie však máme poslednú, tretiu metódu, ktorá nám konštrukciu funkcie Rozdiel umožní. Tento postup sa nazýva minimalizácia. Pri danom kladnom prirodzenom čísle n je jeho vstupom tentoraz jediná funkcia g, ktorá má $n + 1$ argumentov, a výstupom je funkcia f s n argumentmi, pre ktorú platí

$$f(x_1, \ldots, x_n) = \min\{y : g(x_1, \ldots, x_n, y) = 0 \wedge (\forall i < y) g(x_1, \ldots, x_n, i) > 0\}$$

(ak je minimovaná množina prázdna, f nie je v tomto bode definovaná). Potom povieme, že funkcia f vznikla minimalizáciou z funkcie g.

Všimnime si, že ak je funkcia g totálna (a každá logická funkcia taká je), sa tento vzťah dokonca zrejme redukuje na jednoduchší tvar

$$f(x_1, \ldots, x_n) = \min\{y : g(x_1, \ldots, x_n, y) = 0\}.$$

Ak má hrať úlohu f funkcia `Rozdiel`, platí $n = 2$, hľadáme teda (najlepšie) logickú funkciu g, pre ktorú platí

$$\texttt{Rozdiel}(x_1, x_2) = \min\{y : g(x_1, x_2, y) = 0\}.$$

Vieme, že $\texttt{Rozdiel}(a, b) = c$ platí práve vtedy, keď $a = \texttt{Súčet}(b, c)$, t. j. keď platí $\texttt{JeRôzneOd}(a, \texttt{Súčet}(b, c)) = 0$. Stačí teda vziať pomocnú funkciu h takú, že

$$h(a, b, c) = \texttt{Súčet}(b, c) = \texttt{Súčet}(\texttt{P}_2^3(a, b, c), \texttt{P}_2^3(a, b, c)),$$

a funkciu g tak, že bude platiť

$$g(a, b, c) = \texttt{JeRôzneOd}(a, \texttt{Súčet}(b, c)) = \texttt{JeRôzneOd}(\texttt{P}_1^3(a, b, c), h(a, b, c)).$$

Z týchto vyjadrení vyplýva, že ako h, tak (následne) g sú primitívne rekurzívne, a navyše funkcia `Rozdiel` vznikla z funkcie g minimalizáciou.

Na tomto príklade si môžeme všimnúť, že minimalizácia nemusí zachovávať totalitu – vstupná funkcia g je totálna, avšak výsledná funkcia `Rozdiel` nie.

Teraz už môžeme definovať rekurzívne funkcie. Budú to práve tie, ktoré vzniknú zo základných pomocou konečného počtu substitúcií, rekurzií a minimalizácií.

2.5. Vzťah rekurzívnych a turingovsky vypočítateľných funkcií

Sústreďme sa teraz na niektoré pojmy z teórie (prirodzených) čísel. Začnime dôležitými pojmami deliteľnosti a prvočíselnosti, ktoré môžeme definovať pomocou výrokových funkcií:

- Deliteľnosť možno vyjadriť logickou funkciou `Delí` s dvoma argumentmi, z ktorých prvý bude deliteľom druhého:

  $\texttt{Delí}(a, b) = 1$ práve vtedy, keď $(\exists c) b = ac$.

  Takže napríklad $\texttt{Delí}(3, 6) = 1$ (príslušné c je zrejme 2), kým $\texttt{Delí}(3, 5) = 0$ (lebo neexistuje (prirodzené číslo) c, pre ktoré by platilo $3c = 5$).

- Prvočíselnosť bude vyjadrená logickou funkciou `JePrvočíslo` s jedným argumentom a použijeme v nej pred chvíľou definovanú funkciu `Delí`:

  $\texttt{JePrvočíslo}(b) = 1$ práve vtedy, keď

  $$b > 1 \wedge (\forall a)((a > 1 \wedge a < b) \rightarrow \texttt{Delí}(a, b) = 0).$$

  Takže napríklad $\texttt{JePrvočíslo}(3) = 1$, kým $\texttt{JePrvočíslo}(4) = 0$.

Ako je to s rekurzivitou týchto dvoch funkcií? Prvá výroková funkcia obsahuje násobenie, ktoré sme vyjadrili primitívne rekurzívnou funkciou `Súčin`, a rovnosť vyjadrenú tiež primitívne rekurzívnou logickou funkciou `JeRovnakéAko`. Podobne v druhej (po príslušnom prepise) používame primitívne rekurzívne logické funkcie `JeVäčšieAko` a `JeMenšieAko` a konštanty 1 a 0. Zatiaľ teda nič (primitívnu) rekurzivitu nekazí. Dá sa ukázať, že ju nepokazí ani použitie logických spojok (v našom prípade sú to $\wedge$ a $\rightarrow$). Problémom sú však kvantifikátory $(\exists c)$, resp. $(\forall a)$, ktoré rekurzivitu pokaziť môžu. Ak sa však podarí takúto kvantifikáciu rozumne zhora ohraničiť, (primitívna) rekurzivita

sa zachová. A to sa v oboch našich prípadoch podarí – ľahko vidieť, že prvú kvantifikáciu stačí zmeniť na $(\exists c \leq b)$ a druhú na $(\forall a \leq b)$. Takto sa nám podarilo ukázať, že ako `Delí`, tak (následne) `JePrvočíslo` sú primitívne rekurzívne funkcie.

Teórii čísel sa nevenujeme náhodou, umožňuje nám totiž efektívne kódovanie. To je založené na tzv. základnej vete aritmetiky, ktorá hovorí, že každé prirodzené číslo väčšie než 1 možno jednoznačným spôsobom rozložiť na súčin prvočísel. Takže napríklad

$$12936 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 7 \cdot 7 \cdot 11 = 2^3 \cdot 3^1 \cdot 5^0 \cdot 7^2 \cdot 11^1 = 2^3 \cdot 3^1 \cdot 5^0 \cdot 7^2 \cdot 11^1 \cdot 13^0 \cdot 17^0 \cdot 19^0 \cdot 23^0 \cdots$$

Každému číslu tak môžeme priradiť postupnosť exponentov jeho prvočíselného rozkladu. Uvedomme si, že od istého miesta musí obsahovať samé nuly, takže ju vlastne môžeme považovať aj za konečnú.

Tento prístup však môžeme aj otočiť: Každú konečnú postupnosť prirodzených čísel môžeme zakódovať do jediného prirodzeného čísla tak, že jej členy budeme považovať za exponenty jeho prvočíselného rozkladu (pravdaže, prvočísla v ňom sú usporiadané podľa veľkosti). Napríklad postupnosti $\langle 3, 1, 0, 2, 1 \rangle$ priradíme takýmto spôsobom kód $2^3 \cdot 3^1 \cdot 5^0 \cdot 7^2 \cdot 11^1 = 12936$. Samozrejme, z výsledného kódu vieme pomocou (ešte raz zdôraznime, že jednoznačného) prvočíselného rozkladu pôvodnú postupnosť spätne zrekonštruovať. Dôležité pritom je, že oba procesy – zakódovanie i rozkódovanie – sú primitívne rekurzívne.

Kurt Gödel (1906–1978), rakúsky matematik a logik
(zdroj: http://en.wikipedia.org/wiki/Kurt_G%C3%B6del)

Túto myšlienku môžeme s úspechom využiť za efektívne kódovanie jednotlivých pojmov z teórie Turingových strojov. Vezmime si napríklad inštrukciu. Pri prvej a piatej zložke sú podstatné vlastne iba indexy, čo sú prirodzené čísla, druhá a tretia sú priamo prirodzené čísla (hoci iba 0 alebo 1), štvrtú môžeme pri troche fantázie tiež chápať ako inak zapísané prirodzené číslo (napríklad `L` bude 0, `N` bude 1 a `R` môže byť 2). Inštrukciu teda môžeme chápať ako usporiadanú päticu prirodzených čísel (napríklad $\mathtt{s}_2\mathtt{10Ns}_3$ je vlastne $\langle 2, 1, 0, 1, 3 \rangle$), ktorú tak môžeme zakódovať vyššie uvedeným spôsobom (takže v našom prípade na číslo $2^2 \cdot 3^1 \cdot 5^0 \cdot 7^1 \cdot 11^3$). Takéto nahradenie nejakého pojmu sa nazýva gödelovské očíslovanie. Všimnime si, že nie každé číslo je kódom nejakej inštrukcie, musí zrejme spĺňať tieto podmienky:

- Musí byť kladné.
- V jeho prvočíselnom rozklade sa môže vyskytovať len prvých päť prvočísel.
- Exponenty pri prvočíslach 3 a 5 môžu byť len 0 alebo 1.
- Exponent pri prvočísle 7 môže byť len 0, 1, alebo 2.

Tieto (nutné, ale i postačujúce) podmienky možno zapísať do jedinej výrokovej funkcie, ktorá obsahuje len primitívne rekurzívne funkcie, logické spojky a ohraničené kvantifikátory, čo znamená, že byť kódom nejakej inštrukcie je tiež primitívne rekurzívna vlastnosť.

Úloha 2.15	Navrhnite gödelovské očíslovanie ďalších pojmov z teórie Turingových strojov: • Turingov stroj (nie ako množina, ale ako postupnosť inštrukcií), • páska, • konfigurácia, • konečný výpočet.

Pomocou týchto primitívne rekurzívnych funkcií (a navyše jedinej minimalizácie) vieme rekurzívne vyjadriť aj počet jednotiek na páske poslednej konfigurácie výpočtu na

danom Turingovom stroji začínajúceho sa danou štandardnou blokovou konfiguráciou. To však znamená, že každá turingovsky vypočítateľná funkcia je rekurzívna.

Platí aj opačné tvrdenie – že každá rekurzívna funkcia je turingovsky vypočítateľná – žiaľ, nedostatok priestoru nám znemožňuje predviesť čo i len základné črty tohto dôkazu. Musíme sa preto uspokojiť len s poznámkou, že jeho najrozsiahlejšia časť je založená na matematickej indukcii kopírujúcej definíciu množiny rekurzívnych funkcií.

Dospeli sme tak k dôležitému dôsledku zhrňujúcemu obe tvrdenia – množina turingovsky vypočítateľných funkcií a množina rekurzívnych funkcií sú totožné, a to i napriek rôznosti prostredí, v ktorých boli definované.

Keďže Turingov stroj môžeme chápať ako definíciu algoritmu, byť rekurzívny znamená práve byť algoritmický rozhodnuteľný.

2.6. Problém zastavenia Turingovho stroja

Majme dve funkcie, ZastavíSa a Výsledok, s dvoma parametrami, pričom platí:

- $\texttt{ZastavíSa}(x,t) = 1$, ak sa výpočet začínajúci sa štandardnou blokovou konfiguráciou s jedným blokom kódujúcim vstup x na Turingovom stroji s gödelovským číslom t zastaví. V takom prípade hodnota $\texttt{Výsledok}(x,t)$ vyjadruje počet jednotiek poslednej konfigurácie tohto výpočtu.
- V opačnom prípade $\texttt{ZastavíSa}(x,t) = 0$ a hodnota $\texttt{Výsledok}(x,t)$ nie je definovaná.

Z predchádzajúcej state vieme, že takto definovaná funkcia Výsledok je rekurzívna. Ukážeme sporom, že funkcia ZastavíSa rekurzívna nie je:

Predpokladajme teda, že ZastavíSa je rekurzívna. Definujme funkciu f vzťahom:

$$f(x) = \begin{cases} \texttt{Výsledok}(x,x)+1, & \text{ak } \texttt{ZastavíSa}(x,x) = 1, \\ 0, & \text{ak } \texttt{ZastavíSa}(x,x) = 0. \end{cases}$$

Funkcia f je zrejme totálna a tiež rekurzívna (keďže ZastavíSa je podľa predpokladu rekurzívna). To však znamená, že je aj turingovsky vypočítateľná (keďže tieto pojmy sú, ako už vieme, totožné). Existuje teda Turingov stroj, ktorý funkciu f počíta. Ak jeho kód označíme t_0, podľa state o Turingových strojoch a podľa definícií funkcií Výsledok a ZastavíSa platí

$$f(x) \begin{cases} = \texttt{Výsledok}(x,t_0), & \text{ak } \texttt{ZastavíSa}(x,t_0) = 1, \\ \text{nie je definovaná}, & \text{ak } \texttt{ZastavíSa}(x,t_0) = 0. \end{cases}$$

My však vieme, že funkcia f je totálna, pre každé x teda nastáva prvá možnosť, čiže $f(x) = \texttt{Výsledok}(x,t_0)$ a $\texttt{ZastavíSa}(x,t_0) = 1$. To teda znamená, že výpočet na našom stroji s číslom t_0 sa pre každý vstup x zastaví (a na páske ostane správny počet jednotiek).

Všimnime si teraz hodnotu $f(t_0)$ (nášmu stroju teda ako vstup podhodíme práve jeho číslo). Vieme, že platí $\texttt{ZastavíSa}(t_0,t_0) = 1$, a teda aj v prvom vzťahu nastáva prvá možnosť: $f(t_0) = \texttt{Výsledok}(t_0,t_0)+1$. Z predchádzajúceho odseku však vieme, že platí $f(t_0) = \texttt{Výsledok}(t_0,t_0)$, čo je spor.

To teda znamená, že funkcia ZastavíSa nie je rekurzívna; inými slovami – problém, či sa výpočet na danom Turingovom stroji pri danom vstupe zastaví, nie je algoritmický rozhodnuteľný.

Aktivita	Uvažujte nad dôsledkami tohto tvrdenia v súvislosti s možnosťami výpočtovej techniky.

Čo sme sa naučili v tomto module

V kapitole o konečných automatoch sme sa naučili:

1. konštruovať a formalizovať konečné automaty deterministické (aj nedeterministické) viacerými spôsobmi, vyjadriť jazyk akceptovaný konečným automatom,
2. ukázať, že pre niektoré jazyky neexistuje konečný automat (teda nie sú regulárne),
3. skonštruovať deterministický konečný automat k nedeterministickému.

V kapitole Turingove stroje sme:

1. sa naučili programovať neklasickým spôsobom,
2. porozumeli hlbším vzťahom medzi matematikou a informatikou,
3. pochopili, že nie všetky problémy sa dajú riešiť mechanicky.

Literatúra a použité zdroje

[1] J. E. Hopcroft, J. D. Ullman: *Formal Languages and Their Relation to Automata*. Reading (Mass.), Addison-Wesley, Publ. Comp. Inc., 1969. (Slovenský preklad: Formálne jazyky a automaty. Bratislava, Alfa, 1978.)

[2] J. E. Hopcroft, R. Motwani, J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.

[3] J. Hromkovič: *Theoretical Computer Science. Introduction to Automata, Computability, Complexity, Algorithmics, Randomization*, Springer, 2010.

[4] M. Chytil: *Automaty a gramatiky*. Matematický seminár SNTL, Praha, 1984.

[5] B. Rovan a kol.: *Matematická informatika II*. Skriptum MFF UK, Bratislava, 1982.

[6] V. Geffert, P. Mlynarčík: *Zbierka príkladov k prednáške Automaty a formálne jazyky*. PF UPJŠ, 2010.

[7] Wikipedia – *http://sk.wikipedia.org/wiki/Automat*, *http://en.wikipedia.org/wiki/Automata_theory*

[8] L. Bukovský: *Teória algoritmov*. Skriptum PF UPJŠ, Košice, 2004

[9] J. Gruska, I. M. Havel, J. Wiedermann, J. Zelený: *Počítačová revolúcia*. Zborník prednášok SOFSEM '83, Výskumné výpočtové stredisko, Bratislava, 1983.

Tento študijný materiál vznikol ako súčasť národného projektu Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika v rámci Aktivity „Vzdelávanie nekvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ“.

Autori © doc. RNDr. Gabriela Andrejková, CSc.
doc. RNDr. Stanislav Krajči, PhD.

Názov Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Podnázov Didaktika programovania pre stredné školy 1

Študijný materiál prešiel recenzným pokračovaním.

Recenzenti doc. RNDr. Dana Pardubská, CSc.
Mgr. Alexander Szabari, PhD.

Počet strán 40

Náklad 300 ks

Prvé vydanie, Bratislava 2011

Všetky práva vyhradené.

Toto dielo ani žiadnu jeho časť nemožno reprodukovať bez súhlasu majiteľa práv.

Vydal Štátny pedagogický ústav, Pluhová 8, 830 00 Bratislava, v súčinnosti s Univerzitou Pavla Jozefa Šafárika v Košiciach, Univerzitou Komenského v Bratislave, Univerzitou Konštantína Filozofa v Nitre, Univerzitou Mateja Bela v Banskej Bystrici a Žilinskou univerzitou v Žiline

Vytlačil BRATIA SABOVCI, s r.o., Zvolen

ISBN 978-80-8118-081-1