KATEDRA INFORMATIKY
PŘÍRODOVĚDECKÁ FAKULTA
UNIVERZITA PALACKÉHO

ZÁKLADY TEORETICKÉ INFORMATIKY

PAVEL MARTINEK

VÝVOJ TOHOTO UČEBNÍHO TEXTU JE SPOLUFINANCOVÁN
EVROPSKÝM SOCIÁLNÍM FONDEM A STÁTNÍM ROZPOČTEM ČESKÉ REPUBLIKY

Olomouc 2006

Abstrakt

Tento text distančního vzdělávání si klade za cíl představit čtenáři základní disciplíny teoretické informatiky, konkrétně formální jazyky, automaty, vyčíslitelnost a složitost. Vzhledem k omezenému rozsahu textu jsou v něm zahrnuty pouze nejzákladnější informace z dotyčných disciplín. Pro pochopení látky je nezbytná znalost základů teorie množin, algebry a výrokové logiky včetně odpovídající matematické notace.

Cílová skupina

Text je určen posluchačům bakalářského studijního programu Aplikovaná informatika, provozovanému v kombinované formě na Přírodovědecké fakultě Univerzity Palackého v Olomouci.

Obsah

1 Formální jazyky a automaty

V tomto oddílu se čtenář dozví o velmi úzce souvisejících pojmech, a sice formálních jazycích a automatech.

Formální jazyk, jak napovídá samotný název, představuje formalizaci pojmu jazyk, který je běžně chápán jako prostředek komunikace mezi lidmi. Tato formalizace je natolik obecná, že zahrnuje rovněž popis programovacích jazyků, stadií růstu rostlin, apod.

Automat si lze představit jako mechanický či abstraktní stroj, který na základě vnějšího podnětu dokáže měnit svůj vnitřní stav. K tomu přistupují následující omezení. Druhů působících podnětů je pouze konečný počet a vnitřních stavů, v nichž se automat může nacházet, je také konečně mnoho.

Teorie automatů nachází působivé uplatnění např. při strojovém (tj. počítačovém) překladu kódu programovacího jazyka do strojového kódu.

Automaty slouží jako účinný nástroj pro práci s formálními jazyky.

1.1 Základní pojmy

Studijní cíle: Po prostudování kapitoly bude studující schopen formálně popsat jazyk, gramatiku a způsob, jakým gramatika generuje jazyk.

Klíčová slova: Abeceda, jazyk, gramatika.

Potřebný čas: 60 minut.

Při formalizaci pojmů jazyk či gramatika se pochopitelně vychází ze znalostí získaných studiem přirozených jazyků (Toto označení je užíváno pro jazyky jako jsou čeština, angličtina, arabština, apod.). V nejobecnější podobě lze každý jazyk ztotožnit s (případně nekonečnou) množinou vět, z nichž každá se skládá z konečně mnoha nižších jednotek. Souhrn všech těchto nižších jednotek bude tvořit tzv. abecedu, nad níž je daný jazyk definován. (Poznamenejme, že některé termíny známé z klasické lingvistiky se v teorii formálních jazyků váží k odlišným objektům.)

Definice 1.1. *Abecedou* nazveme libovolnou konečnou neprázdnou množinu. Její prvky nazýváme *symboly*.

Definice 1.2. *Řetězem (slovem)* v nad abecedou X rozumíme libovolnou konečnou posloupnost symbolů z X, tj. $v = a_1 \cdots a_n$, kde $a_1, \ldots, a_n \in X$. Číslo n nazýváme *délkou řetězu* v a píšeme $|v| = n$. Jestliže $a_i = a \in X$ pro $i \in \{1, 2, \ldots, n\}$, píšeme obvykle a^n místo $a_1 a_2 \cdots a_n$.

Pro řetězy zavádíme operaci zřetězení: Jestliže $u = a_1 \cdots a_m$ a $v = b_1 \cdots b_n$ jsou řetězy nad abecedou X, pak *zřetězením* u a v rozumíme řetěz $a_1 \cdots a_m b_1 \cdots b_n$; ozn. jej $u \cdot v$ nebo stručněji uv.

Jestliže u, v, w, x jsou řetězy takové, že $w = uxv$, pak x nazýváme *podřetězem* řetězu w.

Prázdným řetězem nazýváme řetěz délky 0; značíme jej ε.

Příklad 1.3. Nad abecedou $X = \{a, b\}$ můžeme sestavit například řetězy $u = abab$, $v = b^2a$, $w = \varepsilon$.

Všechny podřetězy řetězu u jsou: $\varepsilon, a, b, ab, ba, aba, bab, abab$.
Zřejmě je $|u| = 4$, $|v| = 3$ a $|w| = 0$.
Uveďme namátkou některá zřetězení: $uv = abab^3a$, $wu = abab$.
Platí: $|uv| = 7 = |u| + |v|$, $|wu| = 4 = |w| + |u|$.

Poznámka 1.4. Pro libovolnou abecedu X a řetězy u, v nad X zřejmě platí

- $|uv| = |u| + |v|$,
- $u\varepsilon = \varepsilon u = u$.

Definice 1.5. Množinu všech řetězů nad abecedou X značíme X^* a nazýváme *uzávěrem množiny* X. Jestliže z X^* odebereme prázdný řetěz, dostaneme *pozitivní uzávěr množiny* X; ozn. jej X^+.

Příklad 1.6. Uvažme abecedu $X = \{0, 1\}$. Pak je zřejmě
$X^+ = \{0, 1, 00, 01, 10, 11, 000, \ldots\}$,
$X^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$.

Definice 1.7. *Jazykem* L nad abecedou X nazýváme libovolnou podmnožinu množiny X^*. Říkáme, že jazyk L je
- *prázdný*, jestliže $L = \emptyset$,
- *konečný*, jestliže obsahuje konečně mnoho slov,
- *nekonečný*, jestliže obsahuje nekonečně mnoho slov.

Příklad 1.8. Následující množiny jsou jazyky nad abecedou $X = \{a, b\}$.

a) $L_1 = \{\varepsilon\}$,
b) $L_2 = \{a, bba, abbaa, baaab\}$,
c) $L_3 = \{a^p;\ p \text{ je prvočíslo}\}$,
d) $L_4 = \{w \in \{a, b\}^*;\ w \text{ obsahuje stejný počet } a \text{ jako } b\}$.

Příklad 1.9. Čeština zřejmě představuje jazyk nad abecedou tvořenou všemi českými slovními tvary (tj. Jana, Jany, Janě, píše, psala, jarní, jarního, jarnímu, ...) a interpunkčními znaménky – není těžké ověřit, že jde o abecedu. Tento jazyk obsahuje všechny správně utvořené české věty neboli jisté řetězy prvků právě popsané abecedy.

Abeceda a slovo jsou v teorii formálních jazyků jiné pojmy než v klasické jazykovědě.

V případě nekonečných jazyků je zásadní otázkou způsob jejich reprezentace. V předchozích příkladech byla příslušnost řetězu k nekonečnému jazyku vždy podmíněna splněním konkrétní podmínky (prvočíselnost, počty výskytů symbolů abecedy nebo gramatická správnost). Právě na příkladu češtiny je zřetelně vidět, že některé podmínky se ověřují obtížně – jistě si lze představit celou řadu vět, u nichž se budou dva fundovaní odborníci rozcházet v názoru, zda jde o správné české věty či nikoliv. Přestože do schémat formálních jazyků zapadají přirozené jazyky poněkud komplikovaně, dávají nám metody vyvinuté pro zkoumání přirozených jazyků konkrétní návod, jak nekonečné jazyky popsat pomocí jednoduchých pravidel: Sestavit příslušnou gramatiku![1]

Pro konečnou reprezentaci nekonečných jazyků se používají gramatiky a automaty.

Před vlastní definicí formální gramatiky nejdříve zavedeme pojem zřetězení množin.

Definice 1.10. Jsou-li A, B množiny řetězů nad abecedou Σ, pak *zřetězením* A a B nazýváme množinu

$$AB = \{w \in \Sigma^*;\ \exists u \in A, \exists v \in B :\ w = uv\}.$$

Průvodce studiem

Zřetězení množin (či jazyků) A a B obsahuje všechny možné řetězy vzniklé vždy zřetězením jednoho řetězu z A a jednoho řetězu z B v tomto pořadí.

[1] Později si ukážeme i systémově odlišný způsob popisu nekonečného jazyka – pomocí automatu.

Příklad 1.11. Uvažme množiny $A = \{\text{Eva, Jan, zajíc}\}$ a $B = \{\text{spí, běží}\}$. Potom $AB = \{\text{Evaspí, Janspí, zajícspí, Evaběží, Janběží, zajícběží}\}$.

Definice 1.12. *Gramatikou* nazýváme uspořádanou čtveřici $G = (N, \Sigma, P, S)$, kde

N je abeceda se symboly zvanými *neterminály*,
Σ je abeceda se symboly zvanými *terminály*, splňující podmínku $N \cap \Sigma = \emptyset$,
$P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ je konečná množina *pravidel*;
její prvky tvaru (α, β) píšeme obvykle jako $\alpha \to \beta$ (a čteme „α se přepisuje na β“),
$S \in N$ je *počáteční symbol* (*neterminál*).

Průvodce studiem

Pojem terminál bude označovat (v souladu s anglickou terminologií) koncový nebo závěrečný prvek, zatímco neterminál bude značit prvek, který koncový není a je zapotřebí s ním určitým způsobem pracovat ještě dál. Vše bude jasnější až po definici jazyka generovaného gramatikou. Můžeme předeslat, že tento jazyk bude obsahovat pouze řetězy terminálních symbolů; neterminály budou hrát roli pomocných symbolů (ovšem s nezastupitelnou rolí). Proto je také v definici gramatiky požadavek, aby množiny terminálů a neterminálů neobsahovaly žádný společný prvek.

Poněkud delší popis množiny pravidel P lze slovně vyjádřit tak, že v každém pravidle $\alpha \to \beta \in P$ jsou α i β řetězy, v nichž se mohou vyskytovat jak neterminály tak terminály. U řetězu α je navíc požadavek, že musí obsahovat alespoň jeden neterminální symbol. Odtud vyplývá, že například $\varepsilon \to S$ nemůže být pravidlo v žádné gramatice.

Časté způsoby označování terminálů, neterminálů a obecných řetězů

Při práci s gramatikami je často používána následující konvence:

- Neterminály jsou obvykle označovány velkými písmeny ze začátku latinské abecedy, tj. $A, B, C, \ldots$
- Terminály jsou obvykle označovány malými písmeny ze začátku latinské abecedy, tj. $a, b, c, \ldots$
- Řetězy nad abecedou $N \cup \Sigma$ jsou často označovány malými písmeny ze začátku řecké anebo konce latinské abecedy, tj. $\alpha, \beta, \gamma, z, y, x$, apod.
- Skupina pravidel
$$\begin{aligned} \alpha &\to \beta_1 \\ \alpha &\to \beta_2 \\ &\vdots \\ \alpha &\to \beta_n \end{aligned}$$
je často zapisována ve zkrácené podobě jako $\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$.

Gramatika bývá často zadána pouhým výčtem pravidel.

Poznámka 1.13. Na základě právě uvedené konvence bývá často popis gramatiky redukován na výčet pravidel, která obsahuje. Mlčky se předpokládá, že daná gramatika zahrnuje počáteční neterminál S, množina neterminálů dále obsahuje všechny symboly, jež jsou v množině pravidel označeny velkými písmeny latinské abecedy, podobně se získá množina terminálů.

Definice gramatiky by nám nebyla mnoho platná bez dalšího vysvětlení, jak se používá pro popis (přesněji pro generování) jazyka. Tomuto vysvětlení věnujeme zbytek kapitoly.

Definice 1.14. Jestliže (N, Σ, P, S) je gramatika, pak na množině $(N \cup \Sigma)^*$ definujeme binární relaci *přímého odvození* (*přímé derivace*; ozn. ji $\Rightarrow$) následovně. Pro libovolné $\alpha, \beta \in (N \cup \Sigma)^*$ klademe $\alpha \Rightarrow \beta$, právě když existují $u, v \in (N \cup \Sigma)^*$ a $y \to x \in P$ tak, že $\alpha = uyv$ a $\beta = uxv$. Říkáme, že řetěz β je přímo odvozen z řetězu α (anebo že z řetězu α se přímo derivuje řetěz β).

Průvodce studiem

Přímé odvození řetězu β z řetězu α znamená existenci pravidla $y \to x$ takového, že nahradíme-li některý podřetěz řetězu α rovný y řetězem x, dostaneme místo α řetěz β. Aniž bychom čtenáře zatěžovali formální definicí přepisovacího systému, můžeme konstatovat, že gramatika spolu se způsobem odvozování řetězů reprezentuje konkrétní případ takového přepisovacího systému.

Vzhledem k tomu, že se v textu opakovaně setkáme s běžným algebraickým pojmem uzávěr binární relace, raději si jej připomeňme definicí.

Připomenutí uzávěrů relací

Definice 1.15. Nechť r je binární relace na množině X.

1) *Reflexivním uzávěrem* r nazveme relaci r' splňující vlastnosti:
   a) Pro všechna $x \in X$ platí $(x, x) \in r'$,
   b) Pro všechna $x, y \in X,\ x \neq y,$ platí $(x, y) \in r',$ právě když $(x, y) \in r$.

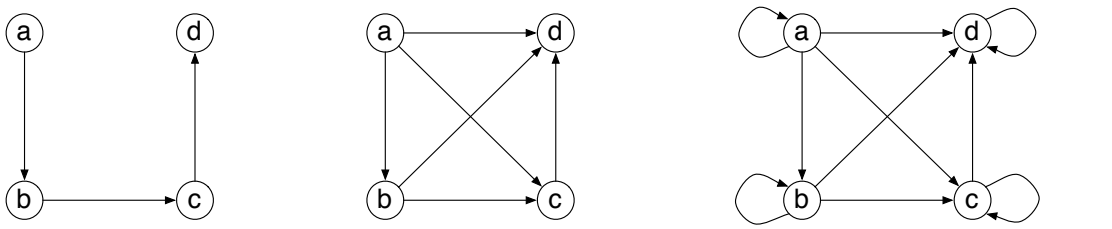
2) *Tranzitivním uzávěrem* r nazveme relaci r'' definovanou induktivně:
   a) Pro všechna $x, y \in X,\ (x, y) \in r,$ platí $(x, y) \in r''$,
   b) Pro všechna $x, y, z \in X,\ (x, y) \in r'',\ (y, z) \in r'',$ platí $(x, z) \in r''$.

3) *Reflexivním a tranzitivním uzávěrem* r nazveme relaci $\tilde{r}$ vzniklou sjednocením relací r' a r''.

Průvodce studiem

Bod 2) předešlé definice lze chápat tak, že relaci r'' můžeme postupně sestavovat zahrnutím všech prvků relace r a opakovaným „přidáváním“ prvků $(x, z) \in X \times X$, k nimž existují prvky $(x, y), (y, z)$ z doposud zkonstruované (chápej nehotové) relace (čili množiny) r''. V případě nekonečné množiny X bychom sice mohli dostat nekonečný proces, ale dá se na to nahlížet také obráceně: r'' obsahuje kromě prvků r pouze prvky, k nimž lze dojít konečně mnoha kroky právě popsaného procesu.

Příklad 1.16. Podívejme se na binární relaci $r = \{(a, b), (b, c), (c, d)\}$. Grafy na obr. 1 znázorňují postupně relaci r, tranzitivní uzávěr r a reflexivní a tranzitivní uzávěr r.



Obrázek 1: Ukázky uzávěrů relací

Definice 1.17. Reflexivní a tranzitivní uzávěr relace $\Rightarrow$ nazýváme *relací odvození* (*relací derivace*) a značíme $\Rightarrow^*$. Tranzitivní uzávěr relace $\Rightarrow$ označujeme $\Rightarrow^+$.

Příklad 1.18. V případě gramatiky obsahující pravidla $S \to aSS \mid ab$ zřejmě platí

$$S \Rightarrow aSS \Rightarrow aSab \Rightarrow aabab.$$

Lze tedy např. psát $S \Rightarrow^* S$,
$S \Rightarrow^* aSab$,
$aSS \Rightarrow^+ aabab$.

Nyní máme k dispozici všechny pojmy potřebné k popisu generování jazyka gramatikou. Do takového jazyka zahrneme všechny terminální řetězy odvoditelné z počátečního symbolu prostřednictvím pravidel dané gramatiky. Formálně tuto skutečnost popisuje následující definice.

Definice 1.19. *Jazykem generovaným gramatikou* $G = (N, \Sigma, P, S)$ nazýváme množinu

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

Příklad 1.20. Uvažme gramatiku $G = (N, \Sigma, P, S)$, kde

$N = \{S, C\}$,
$\Sigma = \{-, 0, 1, \ldots, 9\}$
a P obsahuje pouze pravidla $S \to -C \mid C$,
$C \to CC \mid 0 \mid 1 \mid \ldots \mid 9$.

Pak jsou v gramatice G možná například následující odvození:

$S \Rightarrow -C \Rightarrow -CC \Rightarrow -CCC \Rightarrow -3CC \Rightarrow -32C \Rightarrow -329$,
$S \Rightarrow C \Rightarrow^* CCC \Rightarrow^* 152$.

Není těžké ověřit, že jazyk generovaný gramatikou G zahrnuje množinu všech zápisů celých čísel v dekadické číselné soustavě (včetně „nadbytečných“ nul na začátku čísel, tj. např 0025).

Příklad 1.21. Mějme gramatiku G, jejíž množina pravidel P obsahuje pouze pravidla

$S \to \varepsilon \mid ASB$,
$AB \to BA$,
$A \to a,\ B \to b$.

(Zbývající složky popisu gramatiky G, tj. N, Σ, S, získáme na základě poznámky 1.13.)

Používáme-li opakovaně pravidla 1. skupiny, pak můžeme dospět buď k derivaci
$S \Rightarrow \varepsilon$
anebo k derivacím
$S \Rightarrow^* A^n S B^n$, $S \Rightarrow^* A^n B^n$, kde $n \in \mathbb{N}$.
Tj. počet neterminálů A je v odvozovaných řetězech stejný jako počet neterminálů B.

Pravidlo 2. skupiny pouze zaměňuje pořadí neterminálů A a B v derivovaných řetězech.

Pravidla 3. skupiny pak převádějí neterminály A, B na terminály a, b.

To, že např. pravidla 1. skupiny mohou být použita i po pravidlech 2. či 3. skupiny, zřejmě nic nemění na skutečnosti, že gramatika G generuje jazyk obsahující výhradně řetězy se stejnými počty symbolů a a symbolů b. Odtud

$$L(G) = \{w \in \{a, b\}^*;\ w \text{ obsahuje stejný počet } a \text{ jako } b\}.$$

Shrnutí

Abeceda znamená konečnou neprázdnou množinu.
Řetěz (slovo) nad abecedou X je konečná posloupnost symbolů z X.
Podřetěz řetězu sestávajícího z posloupnosti symbolů $(a_i)_{i=1}^n$ je tvořen libovolnou (i prázdnou) podposloupností.
Zřetězení řetězů $a_1 \cdots a_m$ a $b_1 \cdots b_n$ dává řetěz $a_1 \cdots a_m b_1 \cdots b_n$.
Prázdný řetěz ε je řetěz nulové délky, tj. neobsahuje žádný symbol.
Uzávěr množiny X obsahuje všechny řetězy nad X.
Pozitivní uzávěr množiny X obsahuje všechny řetězy kladných délek nad X.
Každá podmnožina uzávěru abecedy X je jazykem nad X.
Prázdný jazyk neobsahuje žádný řetěz a (ne)konečný jazyk obsahuje (ne)konečně mnoho řetězů.
Zřetězení množin A a B obsahuje všechna existující zřetězení uv, kde $u \in A$ a $v \in B$.
Gramatika je tvořena abecedou neterminálů (z nichž jeden je vyznačen jako počáteční – zpravidla jej označujeme symbolem S), abecedou terminálů a konečnou množinou přepisovacích pravidel. Přepisovací pravidlo má levou stranu tvořenu řetězem obsahujícím libovolný počet terminálů a libovolný kladný počet neterminálů. Pravá strana každého přepisovacího pravidla je tvořena řetězem obsahujícím libovolné počty terminálů a neterminálů.
Přímým odvozením řetězu β z řetězu α (pomocí pravidel dané gramatiky), označovaným $\alpha \Rightarrow \beta$, rozumíme to, že s pomocí některého pravidla přepíšeme nějaký podřetěz řetězu α a dostaneme β. Přepisem pomocí pravidla $y \to x$ máme na mysli náhradu y řetězem x.
Odvozením řetězu β z řetězu α (značeno $\alpha \Rightarrow^* \beta$) rozumíme to, že buď $\beta = \alpha$ anebo existuje $n > 1$ a posloupnost řetězů $(\gamma_i)_{i=1}^n$ taková, že $\gamma_1 = \alpha$, $\gamma_n = \beta$ a $\gamma_i \Rightarrow \gamma_{i+1}$ pro všechna $i \in \{1, \ldots, n-1\}$.

Alternativní popis odvození

Jazyk generovaný gramatikou sestává ze všech terminálních řetězů, jež lze odvodit z počátečního symbolu pomocí pravidel dané gramatiky.

Pojmy k zapamatování

- Abeceda,
- řetěz (prázdný, délka, zřetězení),
- podřetěz,
- jazyk nad abecedou (prázdný, konečný, nekonečný),
- zřetězení jazyků,
- binární relace (přímého odvození, odvození, reflexivní a tranzitivní uzávěr),
- gramatika,
- neterminál,
- terminál,
- počáteční symbol,
- jazyk generovaný gramatikou.

Kontrolní otázky

1. *Tvoří množina všech řetězů nul a jedniček abecedu?*
2. *Uvažme libovolnou abecedu X (tj. konečnou a neprázdnou množinu). Je pozitivní uzávěr X konečná nebo nekonečná množina?*
3. *Jaký je rozdíl mezi prázdným jazykem a jazykem $L = \{\varepsilon\}$?*
4. *Je zřetězení množin komutativní operace, tj. platí $AB = BA$ pro libovolné množiny A, B?*

Cvičení

1. Určete množiny A, B takové, že $AB = \{ba^2b, ba^3b, ba^4b\}$ a každý řetěz z A i z B má délku nejméně 2.
2. Najděte gramatiku, která generuje prázdný jazyk.
3. Najděte gramatiku, která generuje jazyk $L = \{\varepsilon\}$.
4. Najděte gramatiku, která generuje jazyk $L = \{b^2, b^3, b^2a, b^3a, ab^2, ab^3\}$.

Úkoly k textu

1. Již víte, že gramatika je obecně definována s ohledem na jazyk, který má generovat. Rozmyslete si, jaké důsledky pro definici gramatiky G coby uspořádané čtveřice (N, Σ, P, S) by mělo vynechání požadavku $w \in \Sigma^*$ obsaženého v definici jazyka generovaného gramatikou G.
2. Gramatiky jsou zaváděny pro popis jazyků. Objasněte důvod, proč je vyžadována konečnost jednotlivých složek gramatiky.
3. Zjistěte, jak by vypadaly generativní možnosti gramatiky, kdybychom v její definici místo požadavku $S \in N$ vyžadovali $S \in \Sigma$.
4. Změňte pravidla gramatiky z příkladu 1.20 tak, aby generovaný jazyk reprezentoval množinu všech celých čísel a přitom neobsahoval žádný řetěz s prvním symbolem rovným 0 a žádný řetěz s podřetězem -0.

Řešení

1. Řetěz ba^2b vznikl jistě zřetězením ba a ab, protože např. b a a^2b nesplňují podmínku délky nejméně 2. Odtud $ba \in A$, $ab \in B$. To spolu se vztahem $ba^3b \in AB$ vede k možnostem
   a) $ba^2 \in A$ a $a^2b \notin B$,
   b) $ba^2 \notin A$ a $a^2b \in B$,
   c) $ba^2 \in A$ a $a^2b \in B$.

   Vztah $ba^4b \in AB$ dává v případě
   a) výsledek $A = \{ba, ba^2, ba^3\}$ a $B = \{ab\}$,
   b) výsledek $A = \{ba\}$ a $B = \{ab, a^2b, a^3b\}$,
   c) výsledek $A = \{ba, ba^2\}$ a $B = \{ab, a^2b\}$.
2. Požadovaná gramatika nesmí mít žádnou možnost odvození terminálního řetězu z počátečního symbolu. Existuje nekonečně mnoho gramatik s touto vlastností. Buď ji splňují triviálně tím, že mají prázdnou množinu pravidel, anebo netriviálně – například
   a) P obsahuje pouze pravidlo $S \to aS$,
   b) P obsahuje pouze pravidla
      $S \to aA|A$,
      $A \to aS|S$,

   apod.

   Generativní síla gramatiky se nezmění, obohatíme-li ji o pravidla negenerující žádné terminální řetězy.
3. Stačí vzít z předchozího příkladu libovolnou gramatiku, která nemá na pravé straně žádného svého pravidla počáteční symbol S, a obohatit ji o pravidlo $S \to \varepsilon$.
4. I zde existuje nekonečně mnoho vyhovujících gramatik – např.
   a) $S \to b^2 \mid b^3 \mid b^2a \mid b^3a \mid ab^2 \mid ab^3 \mid AaA$,
      $A \to SAS$.
   b) $S \to A \mid Aa \mid aA$,
      $A \to b^2 \mid b^3$.

1.2 Chomského hierarchie gramatik

Studijní cíle: Po prostudování kapitoly bude studující umět klasifikovat formální jazyky a gramatiky.

Klíčová slova: Regulární jazyk, bezkontextový jazyk, kontextový jazyk, jazyk typu 0, regulární gramatika, bezkontextová gramatika, kontextová gramatika, gramatika typu 0.

Potřebný čas: 30 minut.

Nejznámější klasifikaci gramatik představuje Chomského[2] (čti čomského) hierarchie. V této klasifikaci jsou gramatiky odlišovány na základě omezení stanovených pro tvar pravidel. Na jednom konci této hierarchie budou omezení značně volná, na opačném konci pak omezení velmi svazující.

Průvodce studiem

Je pochopitelné, že gramatika s menším množstvím pravidel (podléhajícím přísnějším požadavkům) je schopna vygenerovat méně řetězů, a tedy také „chudší“ jazyk, než gramatika s větším množstvím pravidel. Později se však dozvíte, že ony „chudší“ jazyky lze zpracovávat pomocí jednodušších automatů než jazyky „bohatší“. Pokud nás tedy řešení nějakého problému (například problému sestavení překladače pro některý programovací jazyk) dovede k teorii formálních jazyků, naším prvořadým cílem je vybrat z klasifikace jazyků „nejbližší větší“ typ jazyka – větší proto, aby zahrnoval všechny možné případy řešeného problému, nejbližší proto, aby se k řešení dal použít co nejjednodušší typ automatu.

V následující definici i poznámce několikrát použijeme symbol $\circledast$ pro dovětek: „s eventuální výjimkou pravidla $S \to \varepsilon$, přičemž pak se S nesmí vyskytovat na pravé straně žádného pravidla gramatiky G“.

Definice 1.22 (Chomského hierarchie).

a) Výše definované gramatiky nazýváme *gramatikami typu 0*.

Nechť $G = (N, \Sigma, P, S)$ je gramatika.

b) Jestliže pro všechna pravidla $\alpha \to \beta \in P$ platí
$\alpha = \gamma_1 A \gamma_2$ a $\beta = \gamma_1 \delta \gamma_2$ $\circledast$, kde $\gamma_1, \gamma_2 \in (N \cup \Sigma)^*,\ A \in N,\ \delta \in (N \cup \Sigma)^+$,
pak G se nazývá *gramatikou typu 1* nebo *kontextovou gramatikou*.

c) Jestliže pro všechna pravidla $\alpha \to \beta \in P$ platí
$\alpha \in N$ a $\beta \in (N \cup \Sigma)^+$ $\circledast$,
pak G se nazývá *gramatikou typu 2* nebo *bezkontextovou gramatikou*.

d) Jestliže pro všechna pravidla $\alpha \to \beta \in P$ platí
$\alpha \in N$ a $\beta \in \Sigma \cup \Sigma \cdot N$ $\circledast$,
pak G se nazývá *gramatikou typu 3* nebo *regulární gramatikou*.

Průvodce studiem

Kontextová gramatika má svůj název podle způsobu přepisování odvozovaných řetězů – pokud se neterminál A vyskytuje obklopený řetězy γ_1 a γ_2 (čili v kontextu γ_1, γ_2), může být přepsán na řetěz δ (samozřejmě se zachováním okolního kontextu γ_1, γ_2). Všimněte si, že kontext může být tvořen i prázdnými řetězy ε, tj. kontextová gramatika může obsahovat pravidla umožňující přepis neterminálu bez ohledu na sousední symboly (či řetězy).

[2] Podle věhlasného amerického lingvisty Noama Chomského.

Bezkontextová gramatika obsahuje výhradně „bezkontextová“ pravidla, z nichž každé může být použito k přepisu neterminálu umístěného kdekoliv v přepisovaném řetězu.

Regulární gramatika obsahuje s výjimkou případného pravidla $S \to \varepsilon$ pouze pravidla, na jejichž pravých stranách jsou pouze řetězy délky 1 (1 terminál) anebo délky 2 (terminál následovaný neterminálem).

Poznámka 1.23. Kontextovou gramatiku je možné často najít s odlišnou definicí[3]: Každé pravidlo $\alpha \to \beta \in P$ musí splňovat podmínku $|\alpha| \leq |\beta|$ $\circledast$. Takto definovaná gramatika je občas přiléhavě nazývána *monotónní gramatikou*. Důkaz toho, že třídy jazyků generovaných kontextovými a monotónními gramatikami jsou stejné, lze nalézt například v [Chy82].

V literatuře je kontextová gramatika často definována pomocí monotónní gramatiky.

Věta 1.24. *Nechť $i \in \{1, 2, 3\}$. Každá gramatika typu i je také gramatikou typu $i - 1$.*

Důkaz. Tvrzení přímo vyplývá z definic příslušných gramatik. □

Příklad 1.25. Zjistěte, o kterou gramatiku se jedná:

a) G_1 obsahuje pouze pravidla $S \to bSb \,|\, a$,

b) G_2 obsahuje pouze pravidla
$$\begin{aligned} S &\to aA \,|\, \varepsilon, \\ A &\to aB \,|\, a, \\ B &\to aA, \end{aligned}$$

c) G_3 obsahuje pouze pravidla
$$\begin{aligned} S &\to Aa \,|\, bA, \\ Aa &\to bAa \,|\, ba, \\ bA &\to bAa \,|\, ba, \end{aligned}$$

d) G_4 obsahuje pouze pravidla
$$\begin{aligned} S &\to aS \,|\, b, \\ b &\to Sa. \end{aligned}$$

Řešení:

a) G_1 není regulární gramatikou, protože pravidlo $S \to bSb$ má na pravé straně řetěz délky 3. G_1 je gramatikou bezkontextovou, neboť u každého pravidla je levá strana tvořena právě jedním neterminálem a pravá strana řetězem kladné délky.

b) G_2 je regulární gramatikou, protože u každého pravidla je levá strana tvořena právě jedním neterminálem a pravá strana je (s výjimkou pravidla $S \to \varepsilon$; všimněte si, že S nefiguruje na pravé straně žádného pravidla) tvořena buď jedním terminálem anebo řetězem délky 2, kde je terminál následován neterminálem.

c) G_3 není ani regulární ani bezkontextovou gramatikou, neboť některá pravidla neobsahují na levých stranách pouze jeden neterminál. Jde o gramatiku kontextovou, která kromě bezkontextových pravidel $S \to Aa \,|\, bA$ obsahuje rovněž kontextová pravidla $Aa \to bAa \,|\, ba$ s kontextem tvořeným řetězy ε, a a kontextová pravidla $bA \to bAa \,|\, ba$ s kontextem tvořeným řetězy b, ε.

d) G_4 nesplňuje definici gramatiky, protože „pravidlo“ $b \to Sa$ nemá na levé straně žádný neterminál.

[3] V literatuře lze narazit na odlišné definice u více typů gramatik. Obvykle jde o víceméně drobnou modifikaci a nebývá obtížné dokázat, že třída generovaných jazyků je stejná, popř. se liší jenom nepřítomností prázdných řetězů v těchto jazycích, což představuje snadno odstranitelný rozdíl.

Definice 1.26. Nechť $i \in \{0, 1, 2, 3\}$. Říkáme, že *jazyk je typu* i, právě když existuje gramatika typu i, která jej generuje. Používáme také názvy:

kontextový jazyk (místo jazyka typu 1),
bezkontextový jazyk (místo jazyka typu 2),
regulární jazyk (místo jazyka typu 3).

Věta 1.27. *Nechť* $i \in \{1, 2, 3\}$. *Každý jazyk typu* i *je také jazykem typu* $i - 1$.

Důkaz. Jde o přímý důsledek věty 1.24. □

Poznámka 1.28. Předchozí věta nám nic neříká o odlišitelnosti jazyků typu i od jazyků typu $i - 1$. Dá se ovšem dokázat existence jazyka typu $i - 1$ (pro každé $i \in \{1, 2, 3\}$), který není jazykem typu i. Příslušné důkazy by ovšem neúměrně zvětšily rozsah tohoto textu, a proto odkážeme zájemce na podrobnější literaturu (např. [Sal73] nebo [Hop69]).

Shrnutí

Chomského hierarchie představuje klasickou klasifikaci gramatik i jazyků.
Gramatika typu 0 má na levé straně každého svého pravidla řetěz obsahující alespoň jeden neterminál, na pravé straně řetěz libovolné délky.
Kontextová gramatika má na levé straně každého svého pravidla neterminál „obklopený“ kontextem tvořeným řetězy libovolných délek, na pravé straně řetěz kladné délky „obklopený“ tímtéž kontextem.
Bezkontextová gramatika má levou stranu každého svého pravidla tvořenu jedním neterminálem, pravou stranu řetězem kladné délky.
Regulární gramatika má levou stranu každého svého pravidla tvořenu jedním neterminálem, pravou stranu jedním terminálem anebo řetězem délky 2 obsahujícím terminál a neterminál v tomto pořadí.
Kvůli tomu, aby zmiňované gramatiky mohly vygenerovat prázdný řetěz ε, přidáváme k definicím kontextové, bezkontextové a regulární gramatiky možnost výskytu pravidla $S \to \varepsilon$ s omezením, že v takovém případě gramatika nesmí mít počáteční symbol S na pravé straně žádného svého pravidla.
Každý jazyk typu 0 je generován gramatikou typu 0. Podobně je každý kontextový (bezkontextový, regulární) jazyk generován kontextovou (bezkontextovou, regulární) gramatikou.
Označíme-li symboly $\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2$ a $\mathcal{L}_3$ postupně třídu jazyků typu 0, typu 1 (tj. kontextových), typu 2 (tj. bezkontextových) a typu 3 (tj. regulárních), pak platí vztahy: $\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0$.

Pojmy k zapamatování

- Regulární jazyk,
- bezkontextový jazyk,
- kontextový jazyk,
- jazyk typu 0,
- regulární gramatika,
- bezkontextová gramatika,
- kontextová gramatika,
- gramatika typu 0.

Kontrolní otázky

1. *Je každý regulární jazyk kontextovým jazykem?*
2. *Existuje bezkontextový jazyk, který není regulární?*

3. *Co můžete prohlásit o jazyku generovaném gramatikou s pravidly* $S \to aSSb \mid b,$
$aS \to Sb,$
$bS \to SSS$?

Cvičení

1. Ověřte, že jazyk $L = \{\varepsilon\}$ je regulární.
2. Ověřte, že jazyk $L = \{b^n a b^n;\ n \in \mathbb{N}\}$ je bezkontextový.
3. Dokažte, že každý konečný jazyk je regulární.

Úkoly k textu

1. Promyslete podrobně důkaz věty 1.24 pro jednotlivá i.
2. Zjistěte, proč dovětek $\circledast$ není připojen k definici gramatiky typu 0.

Řešení

1. Jazyk je generován regulární gramatikou s jediným pravidlem: $S \to \varepsilon$.
2. Jazyk je generován například bezkontextovou gramatikou, která obsahuje pouze pravidla $S \to bSb \mid bab$.
3. Uvažme libovolnou abecedu Σ. Každý řetěz nad abecedou Σ lze vygenerovat pomocí konečně mnoha „regulárních“ pravidel:
   a) ε lze generovat pomocí $S \to \varepsilon$,
   b) $a \in \Sigma$ lze generovat pomocí $S \to a$,
   c) $a_1 \cdots a_n \in \Sigma^+$, kde $n > 1$, lze generovat pomocí
   $$\begin{aligned} S &\overset{ozn.}{=} A_0 \to a_1 A_1, \\ &\vdots \\ A_{n-2} &\to a_{n-1} A_{n-1}, \\ A_{n-1} &\to a_n, \end{aligned}$$
   kde $A_i \neq A_j$ pro všechna $i, j \in \{0, \ldots, n-1\}, i \neq j$.

   Každý konečný jazyk $L \subset \Sigma^*$ obsahuje konečně mnoho řetězů. Bez újmy na obecnosti lze jistě předpokládat, že neterminály vyskytující se v pravidlech popsaných výše jsou s výjimkou S pro libovolné dva různé řetězy z L pokaždé jiné. Množina P všech pravidel sestavených dle bodů a), b), c) pro všechny řetězy z L je tedy konečná. Gramatika $G = (N, \Sigma, P, S)$, kde N je množina všech neterminálů vyskytujících se v P (pokud $P = \emptyset$, klademe $N = \{S\}$), je podle definice regulární gramatikou. Zřejmě $L = L(G)$. Jazyk L je proto regulární.

1.3 Konečné automaty

Studijní cíle: Po prostudování kapitoly bude studující schopen formálně popsat konečný automat v deterministické i nedeterministické variantě a způsob, jakým přijímá jazyk. Dále bude umět dokázat, že třída deterministických i třída nedeterministických konečných automatů má stejnou výpočetní sílu. Rovněž by měl být schopen objasnit vztah mezi konečnými automaty a regulárními gramatikami.

Klíčová slova: Konečný automat, nedeterministický konečný automat.

Potřebný čas: 120 minut.

Průvodce studiem

Zatímco gramatiky představují generativní zařízení, tj. umožňují vygenerovat celý jazyk, opačný přístup, tzv. analytický, reprezentují automaty: automat „analyzuje“ řetěz předložený na jeho vstupu – v případě příslušnosti tohoto řetězu k jazyku automat signalizuje tuto skutečnost dohodnutým způsobem. Rozdílnost těchto přístupů vynikne zejména při řešení úlohy, zda konkrétní slovo w patří do zadaného jazyka L. Pokud je L zadán např. gramatikou G typu 0, která jej generuje, jediný obecně použitelný postup spočívá v postupném generování slov z jazyka $L(G)$ a jejich porovnávání s w. Pokud je však L určen automatem A, stačí dát slovo w automatu A na jeho vstup a čekat, jestli bude akceptováno.

V celém textu si postupně popíšeme tři různé typy automatů: konečné automaty, zásobníkové automaty a Turingovy stroje. Každý z nich představuje jistý výpočetní model, tj. abstraktní zařízení, které však může být s větší či menší úspěšností realizováno konkrétním výpočetním systémem. Základní rozdíl mezi jednotlivými typy spočívá v možnosti využívat nějaký druh paměťového zařízení, případně v jeho uspořádání a způsobu práce s uloženými daty.

Gramatiky generují jazyky, automaty „akceptují“ jazyky.

Nejjednodušším automatem je konečný automat, který není vybaven žádným speciálním paměťovým zařízením. Jde o stroj, který se může nacházet v některém z konečně mnoha stavů, na základě působícího podnětu (jednoho z konečně mnoha) buď přejde do stavu jiného anebo setrvá v aktuálním stavu, přičemž dvojici aktuální stav a působící podnět pokaždé odpovídá tentýž výsledný stav.

Konečný automat nemá žádnou paměťovou jednotku.

S právě popsanou charakteristikou se můžeme setkat v běžném životě na mnoha místech. Uvedené tvrzení budeme demonstrovat na velmi jednoduchém příkladu automaticky ovládaných dveří běžně umísťovaných při vstupu do obchodů, bank, apod. Tyto dveře bývají vybaveny fotobuňkou na vnitřní i vnější straně. Mohou se nacházet v jednom ze stavů OTEVŘENO, ZAVŘENO a působícími podněty jsou: pohyb signalizovaný aspoň jednou fotobuňkou a žádný pohyb. Jak již bylo uvedeno, na základě nového podnětu přechází systém z aktuálního stavu do stavu nového, nebo systém setrvá v aktuálním stavu. Tak např. pohyb před dveřmi, jež jsou ve stavu ZAVŘENO (OTEVŘENO) vede ke stavu OTEVŘENO (OTEVŘENO). Úplný výčet všech „přechodů“ dává tabulka 1, kde v průsečíku každého řádku s aktuálním stavem a sloupce s působícím podnětem najdete nový stav.

		Působící podnět	
		pohyb	žádný pohyb
Aktuální stav	OTEVŘENO	OTEVŘENO	ZAVŘENO
	ZAVŘENO	OTEVŘENO	ZAVŘENO

Tabulka 1: Výčet všech přechodů dveří ovládaných fotobuňkou

Poněkud komplikovanějšími zařízeními, jež lze simulovat pomocí konečného automatu, jsou například

- rychlovarná konvice
- autíčko s jednoduchým dálkovým ovládáním
- výtah
- nápojový automat

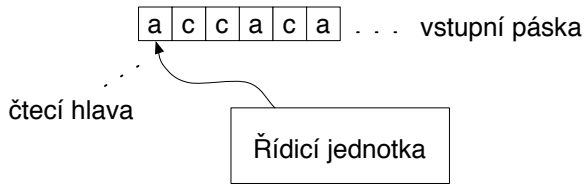
Při simulaci se však nemusíme omezovat pouze na technická zařízení. Se stejným úspěchem lze konečný automat použít např. i na jednoduché modely

Konečný automat může simulovat systémy studované jak v přírodních tak v humanitních vědách.

- chování rostlin
  - možné působící podněty: různé kombinace světla, tepla a vlhkosti,
  - možné stavy: růst, stagnace, uvadání, úhyn,
- chování skupin lidí
  - možné působící podněty: různé kombinace uspokojování životních a sociálních potřeb,
  - možné stavy: spokojenost, lhostejnost, nervozita, panika.

Pro konkrétní představu můžeme konečný automat chápat jako zařízení z obr. 2 skládající se

- z řídicí jednotky, která se může nacházet v některém z konečně mnoha stavů,
- ze vstupní pásky, na níž bude zapsán řetěz symbolů představujících posloupnost podnětů, jejichž působení (ve stejném pořadí jako jsou na pásce) bude konečný automat vystaven,
- ze čtecího zařízení, s jehož pomocí bude řídicí jednotka číst symbol z pásky, který je právě „na řadě“; vlastní čtení bude zprostředkovávat tzv. čtecí hlava.



Obrázek 2: Ilustrace konečného automatu

Jak bude probíhat činnost tohoto zařízení?

Neformální popis činnosti KA

Výchozí předpoklady:

1) Na vstupní pásce bude zapsáno tzv. vstupní slovo tvořené ze symbolů abecedy konečného automatu (nazývané vstupní abecedou). Páska (dělená na jednotlivá políčka) bude mít na každém políčku právě jeden symbol a délka pásky bude stejná jako délka vstupního slova, tj. páska je konečné délky a na jejím konci nejsou žádná nevyužitá políčka.
2) Čtecí hlava je umístěna nad prvním (tj. nejlevějším) políčkem pásky (a je tedy nachystána ke čtení 1. symbolu).
3) Konečný automat se nachází ve vyznačeném, takzvaném počátečním stavu.

Průběh výpočtu: Výpočet probíhá v jednotlivých krocích, přičemž v každém kroku konečný automat

a) přečte symbol pod čtecí hlavou a posune čtecí hlavu o jedno políčko doprava,
b) na základě přečteného vstupního symbolu a aktuálního stavu přejde do stavu nového, nebo setrvá v aktuálním stavu. Informace o tom, jak se má konkrétně zachovat, musí být součástí popisu každého konečného automatu; jinými slovy, součástí popisu každého konečného automatu bude tzv. přechodová funkce, která jednoznačně přiřazuje každé dvojici (stav, symbol vstupní abecedy) konkrétní stav.

Vyhodnocení výpočtu: Po přečtení vstupního slova zřejmě skončil konečný automat v nějakém stavu – pokud půjde o nějaký ze stavů, jež budeme nazývat koncovými, pak budeme tento výsledek chápat tak, že konečný automat příslušné vstupní slovo přijal (akceptoval). Pokud konečný automat skončí svou činnost mimo koncový stav, budeme to chápat jako zamítnutí vstupního slova.

Přejděme nyní k formální, a tedy exaktní definici konečného automatu.

Definice 1.29. *Konečným automatem* (ozn. *KA*) nazýváme uspořádanou pětici $(Q, \Sigma, \delta, q_0, F)$, kde

Q je konečná neprázdná množina s prvky zvanými *stavy*,
Σ je abeceda; říkáme jí *vstupní abeceda*,
$\delta : Q \times \Sigma \to Q$ je *přechodová funkce*,
$q_0 \in Q$ je *počáteční stav*,
$F \subseteq Q$ je *množina koncových stavů*.

Průvodce studiem

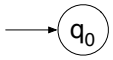
Zavádíme pojem koncového stavu, nikoliv konečného stavu.

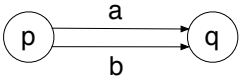
V předchozí definici byl zaveden pojem koncového stavu. Nezaměňujte jej prosím s „konečným“ stavem – ten jsme si nezavedli a ani tak neučiníme – jeho „doplňkem“ by byl zřejmě „nekonečný stav“, tj. stav, který nemá konečnou velikost(??) anebo který obsahuje nekonečně mnoho ... čeho vlastně??? Jinou interpretací konečného stavu by byl závěrečný stav, tj. stav, v němž se KA nachází po skončení své práce. Avšak právě u těchto stavů potřebujeme rozlišení mezi stavy „příznivými“ (v naší terminologii koncovými) a „nepříznivými“ (tj. nekoncovými).

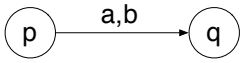
U přechodové funkce δ nepřehlédněte fakt, že jde o zobrazení celé množiny $Q \times \Sigma$ do Q, tj. funkce δ je definována pro každou dvojici stav a symbol.

Poznámka 1.30. KA bývá místo výčtu prvků Q, Σ, F a přechodů funkce δ často znázorňován pomocí takzvaného *stavového diagramu*, který je představován mírně modifikovaným orientovaným grafem, kde

Stavový diagram popisuje KA nejpřehledněji.

- každému stavu odpovídá právě jeden uzel – označený názvem tohoto stavu,
- mezi uzly označenými stavy p a q vede hrana označená symbolem a, právě když $\delta(p, a) = q$,
- každý uzel odpovídající koncovému stavu je znázorněn dvojitým kroužkem,
- do uzlu odpovídajícího počátečnímu stavu míří šipka bez jakéhokoliv označení, tj. 

,
- v případě $\delta(p, a) = q$ a $\delta(p, b) = q$ se místo dvojice hran 

obvykle kreslí hrana jediná: 

.

Ukázku stavového diagramu si můžete prohlédnout na obrázku 3.

Podobně jako jsme u gramatik formálně popisovali způsob, jak z počátečního symbolu generují výsledný řetěz, tak i u KA musíme být schopni formalizovat způsob práce.

Průvodce studiem

Leckdo by se mohl domnívat, že formální popis má jediný cíl: Zbytečně komplikovat studujícím život. Pochopitelně tomu tak není. Kdybyste chtěli dokázat nějaká tvrzení o systémech popsaných pouze neformálně (slovně), rychle byste zjistili, že Vám buď chybí dostatečná přesnost, nebo je důkaz neúměrně dlouhý a velmi nepřehledný. Máte-li o tom sebemenší pochybnosti, zkuste si sepsat např. důkaz věty 1.43 bez použití jakéhokoliv formalismu.

Výpočet KA dělíme na jednotlivé kroky výpočtu, jak bylo již popsáno v části předcházející poslední definici. Kroky výpočtu postupně mění situaci, v níž se KA nachází. Situace bude

jednoznačně charakterizována stavem KA a obsahem dosud nepřečtené vstupní pásky; místo situace budeme používat termín „konfigurace konečného automatu“. Výpočet tak lze popsat pomocí posloupnosti konfigurací, kde

Výpočet KA coby posloupnost konfigurací

- první konfigurace je charakterizována počátečním stavem KA a dosud nečteným vstupním slovem na vstupní pásce automatu, tj. čtecí hlava je nad 1. symbolem vstupního slova,
- poslední konfigurace je charakterizována „přečtením vstupního slova“, tj. dosud nečtená část vstupního slova je rovna prázdnému řetězu ε,
- přechod od každé konfigurace ke konfiguraci bezprostředně následující je zprostředkován krokem výpočtu respektujícím přechodovou funkci.

Formálně popsáno:

Definice 1.31. Mějme libovolný KA $A = (Q, \Sigma, \delta, q_0, F)$. Každou uspořádanou dvojici $(q, w) \in Q \times \Sigma^*$ nazveme *konfigurací konečného automatu* A.

Krok výpočtu A definujeme jako binární relaci $\vdash$ (na množině všech konfigurací) takto: Pro všechna $p, q \in Q, a \in \Sigma, x \in \Sigma^*$ je $(p, ax) \vdash (q, x)$, právě když $\delta(p, a) = q$.

Symbolem $\vdash^*$ budeme označovat reflexivní a tranzitivní uzávěr relace $\vdash$, symbolem $\vdash^+$ tranzitivní uzávěr relace $\vdash$.

Výpočet konečného automatu definujeme prostřednictvím relace $\vdash^*$.

Průvodce studiem

Pozorujete soulad mezi formální definicí kroku výpočtu a jeho neformálním popisem? Snad je zřetelné, že z konfigurace (p, ax), kdy je KA ve stavu p, jeho čtecí hlava čte symbol a a přechodová funkce přikazuje přechod do stavu q, skutečně KA přechází do stavu q a posouvá čtecí hlavu napravo za symbol a neboli přechází do konfigurace (q, x).

Definice 1.32. Řekneme, že KA $A = (Q, \Sigma, \delta, q_0, F)$ *přijímá slovo* $w \in \Sigma^*$, právě když $(q_0, w) \vdash^* (q_f, \varepsilon)$ a $q_f \in F$.

Průvodce studiem

Všimněte si, že slovo $w \in \Sigma^*$ je přijato konečným automatem A jedině tehdy, když se A po jeho přečtení nachází v některém ze svých koncových stavů.

Definice 1.33. *Jazyk* L *přijímaný (rozpoznávaný, akceptovaný) konečným automatem* $A = (Q, \Sigma, \delta, q_0, F)$ značíme $L(A)$ a definujeme následovně:

$$L(A) = \{w \in \Sigma^*;\ \text{KA } A \text{ přijímá } w\}.$$

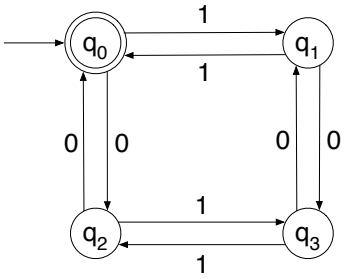
Příklad 1.34. Uvažme KA $A = (Q, \Sigma, \delta, q_0, F)$, kde

$Q = \{q_0, q_1, q_2, q_3\}$,
$\Sigma = \{0, 1\}$,
$F = \{q_0\}$

a přechodová funkce δ je popsána vztahy:

$\delta(q_0, 0) = q_2$ $\quad$ $\delta(q_2, 0) = q_0$
$\delta(q_0, 1) = q_1$ $\quad$ $\delta(q_2, 1) = q_3$
$\delta(q_1, 0) = q_3$ $\quad$ $\delta(q_3, 0) = q_1$
$\delta(q_1, 1) = q_0$ $\quad$ $\delta(q_3, 1) = q_2$

Stavový diagram automatu A je znázorněn na obrázku 3.



Ukázka stavového diagramu KA

Obrázek 3: Stavový diagram konečného automatu z příkladu č. 1.34

Podívejme se na výpočty A pro vstupní slova $w_1 = 0100$ a $w_2 = 1100$. Výpočet pro vstupní slovo w_1 probíhá prostřednictvím následujících výpočetních kroků:

$$(q_0, w_1) = (q_0, 0100) \vdash (q_2, 100) \vdash (q_3, 00) \vdash (q_1, 0) \vdash (q_3, \varepsilon).$$

Průvodce studiem

Krok $(q_0, 0100) \vdash (q_2, 100)$ je umožněn tím, že $\delta(q_0, 0) = q_2$. Následující krok odpovídá: $\delta(q_2, 1) = q_3$. Dále je použito $\delta(q_3, 0) = q_1$ a $\delta(q_1, 0) = q_3$.

Protože výpočet skončil mimo koncový stav ($q_3 \notin F$), slovo w_1 nepatří do jazyka přijímaného konečným automatem A. Výpočet A pro vstupní slovo w_2:

$$(q_0, w_2) = (q_0, 1100) \vdash (q_1, 100) \vdash (q_0, 00) \vdash (q_2, 0) \vdash (q_0, \varepsilon).$$

Tento výpočet skončil v koncovém stavu $q_0 \in F$, je tedy $w_2 \in L(A)$.

Promyslíme-li si s pomocí stavového diagramu z obrázku 3 možnosti výpočtů A, zjistíme, že pouze pro vstupní slova sudých délek (včetně nulové délky), která obsahují sudý počet jedniček, výpočty končí v koncovém stavu $q_0 \in F$. Odtud

$$L(A) = \{w \in \{0,1\}^*;\ w \text{ má sudou délku a obsahuje sudý počet jedniček}\}.$$

1.3.1 Nedeterministické konečné automaty

Determinismus versus nedeterminismus

V definici konečného automatu jsme požadovali tzv. *determinismus*. Šlo o požadavek, aby byl stavem a čteným symbolem jednoznačně určen stav, do nějž má automat přejít. Zajímavým a užitečným zobecněním, s nímž se setkáme i u dalších typů automatů, je *nedeterminismus*. U nedeterministického konečného automatu nepožadujeme, aby stavem a čteným symbolem byl určen jediný nový stav, ale připouštíme existenci celé (případně i prázdné) množiny stavů, z nichž musí automat v příslušném kroku výpočtu jeden vybrat.

Průvodce studiem

Na výpočet nedeterministického konečného automatu lze nahlížet několika způsoby. Jeden z nich spočívá v představě, že kdykoliv má stroj na výběr z několika stavů, rozdělí se na stejný počet identických strojů, z nichž každý přejde do jiného z nabízených stavů. Jakmile má kterákoli z těchto kopií při čtení nového symbolu podle přechodové funkce opět na výběr z několika stavů, dochází k novému „dělení“. Jestliže některé z kopií pro její stav a čtený nový symbol nabízí přechodová funkce prázdnou množinu stavů, pak tato kopie zanikne. Pokud se některá z kopií po přečtení vstupního slova ocitne v některém

z koncových stavů, budeme to chápat tak, že nedeterministický konečný automat toto slovo přijal. V rámci uvedené interpretace je patrné urychlení výpočtů nedeterministickým strojem vůči stroji deterministickému. Zmíněné urychlení má zřejmě exponenciální nárůst způsobený postupně narůstající paralelizací.

Jiná možná interpretace nahlíží na výpočet nedeterministického stroje, jako kdyby tento stroj věděl, kterou cestou se vydat, aby došel ke kýženému cíli – dospět po přečtení vstupního slova do některého koncového stavu.

Máte-li pochybnosti o užitečnosti takto abstraktního stroje, pak věžte, že nalézt k zadanému jazyku nedeterministický konečný automat, který tento jazyk přijímá, je většinou snazší, než nalézt odpovídající deterministický konečný automat. To, že každý nedeterministický konečný automat lze převést na deterministický konečný automat přijímající tentýž jazyk, si ukážeme později.

Definice 1.35. *Nedeterministickým konečným automatem* (zkráceně ozn. *NKA*) rozumíme uspořádanou pětici $(Q, \Sigma, \delta, q_0, F)$, kde

Q je konečná neprázdná množina stavů,
Σ je vstupní abeceda,
$\delta : Q \times \Sigma \to 2^Q$ je přechodová funkce,
$q_0 \in Q$ je počáteční stav,
$F \subseteq Q$ je množina koncových stavů.

Průvodce studiem

Připomeňme si, že 2^Q označuje potenční množinu množiny Q, tj. $2^Q = \{M;\ M \subseteq Q\}$.

Navazující definice se až na krok výpočtu formálně neliší od definic příslušných ke konečnému automatu.

Definice 1.36. Mějme libovolný NKA $A = (Q, \Sigma, \delta, q_0, F)$. Každou uspořádanou dvojici $(q, w) \in Q \times \Sigma^*$ nazveme *konfigurací nedeterministického konečného automatu* A.

Krok výpočtu A definujeme jako binární relaci $\vdash$ (na množině všech konfigurací) takto: Pro všechna $p, q \in Q, a \in \Sigma, x \in \Sigma^*$ je $(p, ax) \vdash (q, x)$, právě když $\delta(p, a)$ obsahuje q.

Symbolem $\vdash^*$ budeme označovat reflexivní a tranzitivní uzávěr relace $\vdash$, symbolem $\vdash^+$ tranzitivní uzávěr relace $\vdash$.

Výpočet nedeterministického konečného automatu definujeme prostřednictvím relace $\vdash^*$.

Průvodce studiem

Všimněte si, že v případě $\delta(p, a) = \{q_1, q_2, q_3\}$ jsou v binární relaci $\vdash$ následující dvojice konfigurací:

$(p, ax) \vdash (q_1, x)$,
$(p, ax) \vdash (q_2, x)$,
$(p, ax) \vdash (q_3, x)$.

Výpočet obsahující konfiguraci (p, ax) bude tedy zahrnovat tři „větve výpočtu“, z nichž každá pokračuje po konfiguraci (p, ax) jinou konfigurací z množiny $\{(q_1, x), (q_2, x), (q_3, x)\}$.

Definice 1.37. Řekneme, že NKA $A = (Q, \Sigma, \delta, q_0, F)$ *přijímá slovo* $w \in \Sigma^*$, právě když $(q_0, w) \vdash^* (q_f, \varepsilon)$ a $q_f \in F$.

Průvodce studiem

Volně řečeno, slovo je přijato nedeterministickým konečným automatem A, právě když pro ně existuje přijímající výpočet A.

Definice 1.38. *Jazyk L přijímaný (rozpoznávaný, akceptovaný) nedeterministickým konečným automatem $A = (Q, \Sigma, \delta, q_0, F)$* značíme $L(A)$ a definujeme následovně:

$$L(A) = \{w \in \Sigma^*;\ \text{NKA } A \text{ přijímá } w\}.$$

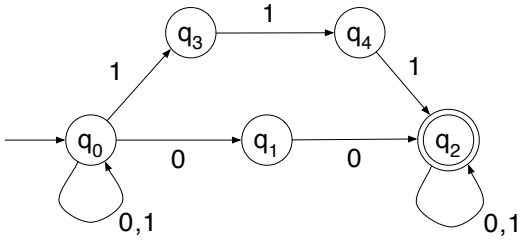
Průvodce studiem

Nenechte se zmást možností, že pro slovo z jazyka $L(A)$ může existovat „nepřijímající větev“ výpočtu. Jde-li skutečně o slovo z $L(A)$, pak pro ně nutně existuje i „přijímající větev“ výpočtu.

Příklad 1.39. Najděte NKA, který přijímá jazyk $L = \{w \in \{0,1\}^*;\ w$ obsahuje podřetěz 00 anebo podřetěz $111\}$.

Zadání zřejmě vyhovuje NKA, jehož stavový diagram je na obr. 4.

Ukázka stavového diagramu NKA



Obrázek 4: Stavový diagram nedeterministického konečného automatu z příkladu č. 1.39

Všimněte si nedeterminismu stroje ve stavu q_0 při čtení symbolu 0 nebo 1, konkrétně $\delta(q_0, 0) = \{q_0, q_1\}$ a $\delta(q_0, 1) = \{q_0, q_3\}$.

1.3.2 Vzájemný vztah nedeterministických a deterministických konečných automatů

Poznámka 1.40. Dosud jsme definovali KA a NKA, později budou následovat nedeterministický zásobníkový automat, deterministický a nedeterministický Turingův stroj; vše budeme označovat souhrnným názvem *stroj*. V tuto chvíli máme definován jazyk $L(A)$ pouze pro deterministický a nedeterministický konečný automat, pro ostatní stroje budou definice $L(A)$ do jisté míry obdobné.

Definice 1.41. O strojích A_1, A_2 řekneme, že jsou *ekvivalentní*, právě když $L(A_1) = L(A_2)$. *Ekvivalence strojů*

Definice 1.42. Dvě třídy strojů nazveme *výpočetně stejně silnými*, právě když ke každému stroji z první třídy existuje ekvivalentní stroj z třídy druhé a naopak (tj. ke každému stroji z druhé třídy existuje ekvivalentní stroj z třídy první). *Stejná výpočetní síla 2 tříd strojů*

Věta 1.43. *Ke každému NKA existuje ekvivalentní KA.*

Důkaz. Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je libovolný NKA. Sestrojíme KA A' takový, že $L(A') = L(A)$.

Průvodce studiem

Základní myšlenka příslušné konstrukce bude založena na sloučení všech stavů z množiny $\delta(p,a)$ do stavu jediného. Tím docílíme toho, že ať NKA A přejde ze stavu p při čtení a do kteréhokoliv z „přípustných“ stavů, zkonstruovaný KA A' bude tento krok výpočtu A simulovat (deterministickým!) přechodem do příslušného jediného stavu. Naznačené sloučení lze provést množinově, tj. u KA A' budou roli jednotlivých stavů hrát množiny obsahující stavy automatu A.

Stačí položit $A' = (Q', \Sigma, \delta', q_0', F')$, kde

Jeden stav z Q' odpovídá množině stavů z Q.

$Q' = 2^Q$ ... tzn. prvky Q' jsou tvořeny množinami prvků z Q,
$q_0' = \{q_0\}$,
$F' = \{K \in Q';\ K \cap F \neq \emptyset\}$,
δ' definujeme následovně: Pro všechna $K, \widetilde{K} \in Q'$ a všechna $a \in \Sigma$ klademe

$$\delta'(K,a) = \widetilde{K}, \text{ právě když } \bigcup_{q \in K} \delta(q,a) = \widetilde{K}.$$

Průvodce studiem

Všimněte si, že $\delta'(K,a)$ představuje podle definice přechodové funkce konečného automatu konkrétní jeden stav z množiny Q' čili dle definice Q' jde o množinu stavů z Q. Na druhou stranu je podle definice přechodové funkce nedeterministického konečného automatu $\delta(q,a)$ množinou stavů z Q. Definice δ' je tedy korektní.

Nyní ověříme, že $L(A') = L(A)$.

1) $\varepsilon \in L(A')$, právě když $q_0' \in F'$. To je ekvivalentní se vztahem $q_0 \in F$ čili $\varepsilon \in L(A)$.
2) Nechť $w \in \Sigma^+$, tj. $w = a_1 \cdots a_k$, kde $a_1, \ldots, a_k \in \Sigma$ a $k \in \mathbb{N}$. Označíme-li $K_0 = q_0'$, pak jsou ekvivalentní následující tvrzení, přičemž s výjimkou případů c) a d) (což bude rozvedeno níže) zmíněné ekvivalence plynou přímo z příslušných definic.
   a) $w \in L(A')$, tj. $a_1 \cdots a_k \in L(A')$.
   b) Existují stavy $K_1, \ldots, K_k \in Q'$ tak, že
   $$\begin{aligned} &\delta'(q_0', a_1) = K_1, \\ &\delta'(K_1, a_2) = K_2, \\ &\qquad\vdots \\ &\delta'(K_{k-1}, a_k) = K_k \text{ a } K_k \in F'. \end{aligned}$$
   c) Existují stavy $K_1, \ldots, K_k \in Q'$ tak, že
   $$\begin{aligned} &\bigcup_{q \in q_0'} \delta(q, a_1) = K_1 \text{ (tj. } \delta(q_0, a_1) = K_1), \\ &\bigcup_{q \in K_1} \delta(q, a_2) = K_2, \\ &\qquad\vdots \\ &\bigcup_{q \in K_{k-1}} \delta(q, a_k) = K_k \text{ a } K_k \in F'. \end{aligned}$$
   d) Existují stavy $q_1, \ldots, q_k \in Q$ tak, že
   $$\begin{aligned} &q_1 \in \delta(q_0, a_1), \\ &q_2 \in \delta(q_1, a_2), \\ &\quad\vdots \\ &q_k \in \delta(q_{k-1}, a_k) \text{ a } q_k \in F. \end{aligned}$$

e) Existují stavy $q_1, \ldots, q_k \in Q$ tak, že
$(q_0, w) = (q_0, a_1 \cdots a_k) \vdash (q_1, a_2 \cdots a_k) \vdash \ldots \vdash (q_k, \varepsilon)$ a $q_k \in F$.
f) $w \in L(A)$.

Nyní k ekvivalenci mezi c) a d). V případě $k = 1$ je ověření triviální. Uvažujme proto nadále $k > 1$.

- Nechť platí tvrzení d). Tvrzení c) pak bude splněno, položíme-li $K_1 = \delta(q_0, a_1),\ K_2 = \bigcup_{q \in K_1} \delta(q, a_2),\ \ldots,\ K_k = \bigcup_{q \in K_{k-1}} \delta(q, a_k)$. Potom

  $q_1 \in K_1$,
  $q_2 \in K_2$, neboť $q_2 \in \delta(q_1, a_2)$ a $\delta(q_1, a_2) \subseteq \bigcup_{q \in K_1} \delta(q, a_2) = K_2$
  na základě vztahu $q_1 \in K_1$,
  $\vdots$
  $q_k \in K_k$, neboť $q_k \in \delta(q_{k-1}, a_k)$ a $\delta(q_{k-1}, a_k) \subseteq \bigcup_{q \in K_{k-1}} \delta(q, a_k) = K_k$
  na základě vztahu $q_{k-1} \in K_{k-1}$.

  Protože je navíc $q_k \in F$, dostáváme $q_k \in K_k \cap F$, což znamená, že $K_k \cap F \neq \emptyset$, a tedy $K_k \in F'$.

- Nechť platí tvrzení c). Potom ze vztahu $K_k \in F'$ plyne podle definice množiny F', že K_k a F mají neprázdný průnik. Označme symbolem q_k libovolný prvek tohoto průniku. Pak $q_k \in F$ a $q_k \in K_k = \bigcup_{q \in K_{k-1}} \delta(q, a_k)$. Druhý vztah znamená existenci takového stavu (označme jej q_{k-1}), pro nějž platí $\delta(q_{k-1}, a_k) \ni q_k$ a $q_{k-1} \in K_{k-1}$. V případě $k > 2$ je tedy $q_{k-1} \in \bigcup_{q \in K_{k-2}} \delta(q, a_{k-1})$. Opakujeme-li uvedený postup, dostáváme posloupnost $q_k, q_{k-1}, \ldots, q_1$ vyhovující tvrzení d). □

Důsledek 1.44. *Třída nedeterministických konečných automatů má stejnou výpočetní sílu jako třída konečných automatů.*

Nedeterministické konečné automaty mají stejnou výpočetní sílu jako konečné automaty.

Důkaz. Ke každému NKA existuje ekvivalentní KA podle věty 1.43. To, že platí i obrácené tvrzení (tj. ke každému KA existuje ekvivalentní NKA), plyne z prosté úvahy, že každý KA může být chápán jako speciální případ NKA, a sice NKA s přechodovou funkcí, jejímž výsledkem je pro libovolnou dvojici (stav, symbol) jednoprvková množina. □

1.3.3 Vztah konečných automatů k regulárním jazykům

Průvodce studiem

Jak již bylo uvedeno v pasáži věnované Chomského hierarchii, jednodušší formální jazyky lze zpracovávat pomocí jednodušších automatů než jazyky složitější. Nejjednoduššími jazyky z Chomského hierarchie jsou regulární jazyky – jejich protipólem ve škále automatů budou právě konečné automaty. Díky důsledku 1.44 je dokonce nepodstatné, zda vzájemný vztah regulárních jazyků ke konečným automatům budeme zkoumat prostřednictvím KA nebo NKA.

Věta 1.45. *Ke každé regulární gramatice G existuje NKA M takový, že $L(G) = L(M)$.*

Důkaz. Nechť $G = (N, \Sigma, P, S)$ je libovolně zvolená regulární gramatika. Sestrojíme nyní NKA $M = (Q, \Sigma, \delta, q_0, F)$, který přijímá jazyk $L(G)$.

Průvodce studiem

Základní idea konstrukce bude spočívat ve ztotožnění neterminálů gramatiky G se stavy automatu M. Vygenerování terminálního symbolu v jednom odvozovacím kroku G bude u M odpovídat přečtení téhož (tentokrát vstupního) symbolu v jednom výpočetním kroku. V této souvislosti je zřejmé, že když G končí své odvození, není v odvozeném řetězu žádný neterminál; této situaci však musí u automatu M odpovídat nějaký koncový stav – obecně nemůže jít o žádný stav totožný s nějakým neterminálem, ale jako jediný bude zaveden nově.

Položíme

$Q = N \cup \{A\}$, kde $A \notin N$ (tj. A je nově dodaný symbol),
$q_0 = S$,
$F = \begin{cases} \{A, S\}, & \text{právě když } S \to \varepsilon \in P \\ \{A\} & \text{jinak.} \end{cases}$

Dále pro všechna $B \in Q, a \in \Sigma$

1) položíme $\delta(B, a) = \emptyset$ (připomeňte si, že definice přechodové funkce NKA připouští tuto možnost),
2) pro každé pravidlo $B \to aC \in P$ rozšíříme množinu $\delta(B, a)$ o stav C,
3) v případě $B \to a \in P$ rozšíříme množinu $\delta(B, a)$ o stav A.

Nyní ověříme, že $L(G) = L(M)$.

1) $\varepsilon \in L(G)$, právě když $S \to \varepsilon \in P$. To je podle definice množiny F ekvivalentní vztahu $S \in F$ čili $q_0 \in F$, což nastává, právě když $\varepsilon \in L(M)$.
2) Nechť $a \in \Sigma$. Potom je $a \in L(G)$, právě když $S \to a \in P$. To je podle definice množiny $\delta(S, a)$ ekvivalentní vztahu $A \in \delta(S, a)$ neboli $(q_0, a) = (S, a) \vdash (A, \varepsilon)$, což platí, právě když $a \in L(M)$.
3) Nechť $w \in \Sigma^+$ je délky alespoň 2, tj. $w = a_1 \cdots a_k$, kde $a_1, \ldots, a_k \in \Sigma$ a $k \geq 2$. Označme navíc $S = B_0$. Pak jsou na základě příslušných definic ekvivalentní následující tvrzení.
   a) $w \in L(G)$.
   b) $S \Rightarrow^* a_1 \cdots a_k$ (čili $B_0 \Rightarrow^* a_1 \cdots a_k$).
   c) V množině N existují neterminály $B_1, \ldots, B_{k-1}$ takové, že $S = B_0 \Rightarrow a_1 B_1 \Rightarrow \ldots \Rightarrow a_1 \cdots a_{k-1} B_{k-1} \Rightarrow a_1 \cdots a_k$.
   d) V množině P existují pravidla $B_0 \to a_1 B_1, \ldots, B_{k-2} \to a_{k-1} B_{k-1}, B_{k-1} \to a_k$.
   e) V množině Q existují stavy $B_1, \ldots, B_{k-1}$ takové, že $B_1 \in \delta(B_0, a_1), \ldots, B_{k-1} \in \delta(B_{k-2}, a_{k-1}), A \in \delta(B_{k-1}, a_k)$.
   f) U NKA M existuje posloupnost kroků výpočtu $(S, a_1 \cdots a_k) = (B_0, a_1 \cdots a_k) \vdash (B_1, a_2 \cdots a_k) \vdash \ldots \vdash (B_{k-1}, a_k) \vdash (A, \varepsilon)$.
   g) $w \in L(M)$. □

Při přechodu od KA k regulární gramatice budeme využívat následující pomocné tvrzení.

Lemma 1.46. *Nechť $G = (N, \Sigma, P, S)$ je gramatika obsahující pouze pravidla*
a) $A \to \beta$, kde $A \in N$ a $\beta \in \Sigma \cup \Sigma N$,
b) $S \to \varepsilon$.
Pak existuje regulární gramatika G' taková, že $L(G) = L(G')$.

Průvodce studiem

Všimněte si, že gramatika G by splňovala definici regulární gramatiky, pokud by nepřipouštěla možnost výskytu S na pravých stranách svých pravidel.

Důkaz. Z libovolné gramatiky $G = (N, \Sigma, P, S)$ splňující předpoklady lemmatu lze vytvořit požadovanou regulární gramatiku G' následovně.

1) K množině neterminálů přidáme nový počáteční neterminál S' a pro každé pravidlo $S \to \alpha \in P$ rozšíříme množinu pravidel o $S' \to \alpha$.

   Těmito úpravami budeme mít zajištěnu existenci pravidla $S' \to \varepsilon$ a navíc se nový počáteční symbol S' nebude vyskytovat na pravé straně žádného pravidla.

2) Z množiny pravidel odstraníme $S \to \varepsilon$ a naopak ke každému pravidlu $A \to aS$ přidáme pravidlo $A \to a$.

   Provedené úpravy zaručují splnění podmínek požadovaných po pravidlech regulární gramatiky.

Zbývá ověřit, že jazyky generované gramatikami G a G' jsou stejné. Toto ověření lze provést stejným způsobem, jaký je použit v důkazu věty 1.45. □

Věta 1.47. *Ke každému KA M existuje regulární gramatika G taková, že $L(M) = L(G)$.*

Důkaz. Nechť $M = (Q, \Sigma, \delta, q_0, F)$ je KA. V první fázi sestrojíme k M gramatiku $\widetilde{G}$, která bude generovat jazyk $L(M)$ a která nebude z požadavků kladených na regulární gramatiku splňovat pouze absenci počátečních neterminálů S na pravých stranách pravidel navzdory přítomnosti pravidla $S \to \varepsilon$. Převod na požadovanou regulární gramatiku nám zajistí lemma 1.46.

Průvodce studiem

Základní idea konstrukce $\widetilde{G}$ bude spočívat ve ztotožnění neterminálů gramatiky $\widetilde{G}$ se stavy automatu M. Přečtení vstupního symbolu v jednom kroku výpočtu automatu M bude u $\widetilde{G}$ odpovídat vygenerování téhož (tentokrát terminálního) symbolu v jednom odvozovacím kroku. Zakončení výpočtu M, indikovanému přechodem do některého z koncových stavů, musí pochopitelně odpovídat použití pravidla bez jakéhokoliv neterminálu na pravé straně – tím je znemožněno pokračování derivace, neboť v odvozovaném řetězu pak chybí neterminál, který by bylo možno dále přepisovat.

Položme $\widetilde{G} = (N, \Sigma, P, S)$, kde

$N = Q$,
$S = q_0$,
$P = \{B \to aC;\ a \in \Sigma,\ B, C \in Q :\ \delta(B, a) = C\} \cup$
$\{B \to a;\ a \in \Sigma,\ B \in Q :\ \delta(B, a) \in F\} \cup \{S \to \varepsilon;\ S \in F\}$.

Ověříme, že $L(G) = L(\widetilde{G})$:

1) $\varepsilon \in L(M)$, právě když $q_0 \in F$ neboli $S \in F$. To je podle definice množiny P ekvivalentní vztahu $S \to \varepsilon \in P$, což nastává (viz pravidla z P), právě když $\varepsilon \in L(\widetilde{G})$.

2) Nechť $a \in \Sigma$. Potom je $a \in L(M)$, právě když $(q_0, a) = (S, a) \vdash (X, \varepsilon)$ a $X \in F$ neboli $\delta(S, a) \in F$. To je podle definice množiny P ekvivalentní vztahu $S \to a \in P$, což platí, právě když $a \in L(\widetilde{G})$.

3) Nechť $w \in \Sigma^+$ je délky alespoň 2, tj. $w = a_1 \cdots a_k$, kde $a_1, \ldots, a_k \in \Sigma$ a $k \geq 2$. Označme navíc $S = B_0$. Pak jsou na základě příslušných definic ekvivalentní následující tvrzení.

   a) $w \in L(M)$.
   b) U konečného automatu M existuje výpočet $(q_0, a_1 \cdots a_k) = (B_0, a_1 \cdots a_k) \vdash (B_1, a_2 \cdots a_k) \vdash \ldots \vdash (B_k, \varepsilon)$ a $B_k \in F$.
   c) V množině Q existují stavy $B_1, \ldots, B_k$ takové, že $\delta(q_0, a_1) = \delta(B_0, a_1) = B_1, \ldots, \delta(B_{k-1}, a_k) = B_k$ a $B_k \in F$.
   d) V množině P existují pravidla $B_0 \to a_1 B_1, \ldots, B_{k-2} \to a_{k-1} B_{k-1}, B_{k-1} \to a_k$.

e) $S = B_0 \Rightarrow^* a_1 \cdots a_k$.
f) $w \in L(\widetilde{G})$.

Vzhledem k tomu, že gramatika $\widetilde{G}$ může mít počáteční symbol S $(= q_0)$ na pravých stranách svých pravidel a obsahuje pravidlo $S \to \varepsilon$, nemusí jít obecně o regulární gramatiku. Splňuje však předpoklady lemmatu 1.46. Existuje tedy regulární gramatika G generující stejný jazyk. Tímto jazykem je $L(M)$ a důkaz je hotov. □

Důsledek 1.48. *Třída regulárních jazyků je rovna třídě jazyků přijímaných konečnými automaty.*

Každý regulární jazyk lze popsat regulární gramatikou nebo konečným automatem.

Důkaz. Jde o přímý důsledek vět 1.45, 1.47 a důsledku 1.44. □

Shrnutí

KA je zařízení vybavené konečněstavovou řídicí jednotkou, čtecí hlavou a vstupní páskou konečné délky. Každý KA je jednoznačně určen

- množinou stavů (v nichž se může nacházet řídicí jednotka),
- vstupní abecedou Σ (slova nad Σ slouží jako jediné možné vstupy pro daný KA),
- přechodovou funkcí δ (která každé dvojici stav a symbol vstupní abecedy přiřazuje stav),
- jedním vyznačeným stavem (tzv. počátečním stavem – v něm bude řídicí jednotka před zahájením práce daného KA, ať je na vstupu jakékoli vstupní slovo),
- vyznačenou podmnožinou množiny stavů (tzv. množinou koncových stavů – pokud se KA po „přečtení“ vstupního slova bude nacházet v některém z koncových stavů, budeme říkat, že dané vstupní slovo je automatem přijato).

Výpočet každého KA je tvořen posloupností kroků výpočtu – v každém z nich KA přečte symbol pod čtecí hlavou, posune čtecí hlavu o jedno políčko doprava a přejde do stavu určeného přechodovou funkcí (která dvojici výchozí stav a přečtený symbol přiřazuje „nový“ stav).

V průběhu výpočtu se mění tzv. konfigurace KA, tj. situace, v níž se KA nachází a která je charakterizována aktuálním stavem a dosud nepřečtenou částí vstupního slova. Každý výpočet začíná v počáteční konfiguraci, která je určena počátečním stavem a vstupním slovem na pásce (se čtecí hlavou nad 1. symbolem tohoto slova). Výpočet končí v okamžiku, kdy byl proveden poslední možný krok výpočtu, tj. když byl přečten poslední symbol vstupního slova. Výsledná konfigurace je charakterizována dvojicí stav a prázdný řetěz. Pokud je stav KA ve výsledné konfiguraci jedním z koncových stavů, říkáme, že KA přijal své vstupní slovo. Množina všech slov, jež jsou přijímána, tvoří jazyk přijímaný konečným automatem.

NKA se od KA odlišuje v následujících ohledech: Přechodová funkce NKA zobrazuje každou dvojici stav a vstupní symbol na množinu „přípustných“ stavů. Krok výpočtu NKA pak znamená přechod z aktuálního stavu na základě čteného symbolu do některého z „přípustných“ stavů, přičemž množina „přípustných“ stavů je určena přechodovou funkcí. Výpočet NKA se tak v některých okamžicích může větvit do více variant. Pokud pro zadané vstupní slovo existuje alespoň jedna varianta, kdy skončí NKA svůj výpočet v některém z koncových stavů, říkáme, že NKA přijímá dané slovo. Množina všech slov nad vstupní abecedou, jež jsou přijímána, tvoří jazyk přijímaný nedeterministickým konečným automatem.

Třída všech nedeterministických konečných automatů má stejnou výpočetní sílu jako třída všech konečných automatů, tj. ke každému NKA existuje KA přijímající tentýž jazyk a naopak.

Třída všech jazyků přijímaných konečnými automaty je rovna třídě regulárních jazyků, tj. třídě všech jazyků generovatelných regulárními gramatikami.

Pojmy k zapamatování

- Konečný automat (KA),
- nedeterministický konečný automat (NKA),
- stav,
- vstupní abeceda,

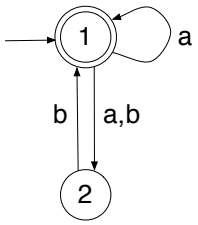
- přechodová funkce,
- počáteční stav,
- množina koncových stavů,
- konfigurace,
- krok výpočtu,
- výpočet,
- jazyk přijímaný KA a NKA,
- ekvivalence strojů,
- stejná výpočetní síla tříd strojů.

Kontrolní otázky

1. *Může být počáteční stav KA (NKA) obsažen v množině koncových stavů?*
2. *Jaký je rozdíl mezi stavem a konfigurací KA (NKA)?*
3. *Kolik různých jazyků může nejvýše přijímat jediný KA (NKA)?*

Cvičení

1. Najděte KA přijímající jazyk a) $L = \emptyset$,
   b) $L = \{\varepsilon\}$,
   c) $L = \{w \in \{1\}^+;\ |w| \text{ není dělitelná } 3\}$,
   d) $L = \Sigma^*$.
2. Najděte KA i NKA přijímající jazyk $L = \{w \in \{a,b\}^*;\ \exists v \in \{a,b\}^* :\ w = vbb\}$.
3. S využitím důkazu věty 1.43 sestavte KA přijímající tentýž jazyk jako NKA s následujícím stavovým diagramem:



Úkoly k textu

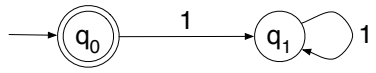
1. Zjistěte, zda je v souladu s definicí NKA to, že ve stavovém diagramu na obrázku 4 není např. ve stavu q_3 vyznačen žádný přechod pro symbol 0.

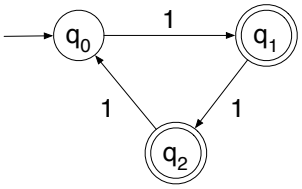
2. Dokončete důkaz lemmatu 1.46.

3. V některých knihách lze nalézt odlišnou definici KA:
   Není vyžadováno, aby byla přechodová funkce definována pro libovolnou dvojici stav a vstupní symbol. V případě, že takto definovaný KA A' při výpočtu nad nějakým vstupním slovem nemá v některém okamžiku definován krok výpočtu, je toto slovo zamítnuto, tj. nepatří do jazyka přijímaného KA A'.
   Dokažte, že výpočetní síla konečných automatů podle této i naší definice zůstává stejná.

Řešení

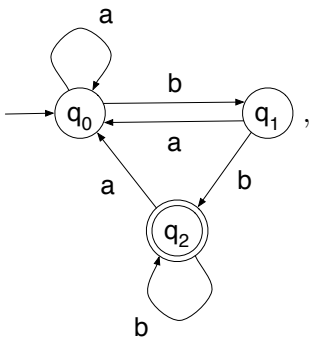
1. a) Zadání vyhovuje např. každý KA s prázdnou množinou koncových stavů.

   b) Např. KA se stavovým diagramem 

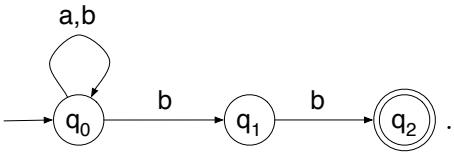
   c) Např. KA se stavovým diagramem 

   d) Zadání splňuje např. každý KA s množinou koncových stavů rovnou celé množině stavů.

2. Uvedený jazyk přijímá např.

   a) KA se stavovým diagramem 

   ,

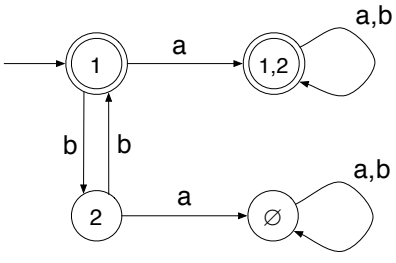
   b) NKA se stavovým diagramem 

   .

3. K NKA ze zadání bude ekvivalentní KA $A' = (Q', \Sigma, \delta', q_0', F')$, kde

   $Q' = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$,
   $\Sigma = \{a, b\}$,
   přechodová funkce δ' je definována:
   $\delta'(\emptyset, a) = \emptyset$,
   $\delta'(\emptyset, b) = \emptyset$,
   $\delta'(\{1\}, a) = \{1,2\}$,
   $\delta'(\{1\}, b) = \{2\}$,
   $\delta'(\{2\}, a) = \emptyset$,
   $\delta'(\{2\}, b) = \{1\}$,
   $\delta'(\{1,2\}, a) = \{1,2\}$ (neboť $\delta(1,a) \cup \delta(2,a) = \{1,2\}$),
   $\delta'(\{1,2\}, b) = \{1,2\}$ (neboť $\delta(1,b) \cup \delta(2,b) = \{1,2\}$),
   $q_0' = \{1\}$,
   $F' = \{\{1\}, \{1,2\}\}$.

   Uvedený KA A' přehledně popisuje stavový diagram:



1.4 Regulární výrazy

Studijní cíle: Po prostudování kapitoly bude studující umět definovat regulární výrazy i jim příslušné jazyky. Bude znát vztah mezi regulárními výrazy a regulárními jazyky. Dále získá rámcovou představu o vzájemných převodech mezi regulárními výrazy a konečnými automaty.

Klíčová slova: Regulární výraz.

Potřebný čas: 60 minut.

Dosud jsme si uvedli dva formalismy vedoucí k regulárním jazykům: regulární gramatiky a konečné automaty. Nyní si uvedeme formalismus třetí, a sice regulární výrazy. Jeho výraznou předností bude veliká názornost, co se týče tvaru slov popisovaného regulárního jazyka. Tato výhoda vede k častému používání regulárních výrazů zejména v počítačovém zpracovávání textů.

Definice 1.49. Nechť Σ je abeceda neobsahující symboly $\varepsilon, \emptyset, +, \cdot, ^*, (,)$. Řekneme, že R je *regulární výraz nad* Σ, jestliže je R rovno některému z následujících symbolů či výrazů:

- $\emptyset$,
- ε,
- a, kde $a \in \Sigma$,
- $(R_1 + R_2)$, kde R_1, R_2 jsou regulární výrazy,
- $(R_1 \cdot R_2)$, kde R_1, R_2 jsou regulární výrazy,
- $(R_1)^*$, kde R_1 je regulární výraz.

Průvodce studiem

Nepřikládejte v tuto chvíli žádný konkrétní význam symbolům použitým v definici regulárních výrazů. Tzn. regulární výraz ε je výraz tvořený symbolem ε, nikoliv tvořený prázdným řetězem. Podobně regulární výraz $\emptyset$. Rozlišujte tedy mezi jmény objektů a objekty vlastními. Regulární výrazy jsou tak posloupnosti symbolů. Zanedlouho se dozvíte, jak každou takovou posloupnost interpretovat čili jaký jazyk jí přiřadit.

Příklad 1.50. Nechť $\Sigma = \{a, b\}$. Uveďme několik případů regulárních výrazů nad abecedou Σ:

- $(a + b)^*$,
- $((\varepsilon + (b \cdot a))^* \cdot a)$.

Následující posloupnosti nejsou žádnými regulárními výrazy, neboť nemají požadovanou syntaxi:

- $+b$,
- $(a + b))$.

Konvence pro zjednodušení zápisu regulárních výrazů

Velké množství závorek v regulárních výrazech dle definice 1.49 způsobuje značnou nepřehlednost, a proto jsou běžně vynechávány stejně jako symbol $\cdot$. Toto vynechání však musí být doplněno o informaci, jakou prioritu máme přisuzovat symbolům $+, \cdot$ a *, jež budou odpovídat množinovým operacím sjednocení, zřetězení a uzávěru. Používaná priorita bude: * má přednost před $\cdot$ a $\cdot$ má přednost před $+$. Např. a konvenčně označuje (a),

$a^* + ab$ konvenčně označuje $((a)^* + (a \cdot b))$.

V případě dvou nebo více po sobě jdoucích symbolů $+$ (nebo $\cdot$) se závorky sdružují doleva. Tj. např.

$a + b + c$ zkracuje zápis $((a + b) + c)$.

Před definicí jazyka přiřazeného regulárnímu výrazu si zaveďme pojem uzávěru jazyka.

Definice 1.51. Nechť Σ je abeceda a L je jazyk nad Σ. *Uzávěr jazyka* značíme L^* a definujeme následovně:

$$L^* = \bigcup_{i=0}^{+\infty} L^i, \quad \text{kde} \quad L^0 \stackrel{def}{=} \{\varepsilon\},$$
$$L^i \stackrel{def}{=} L^{i-1} \cdot L \quad \text{pro všechna } i > 0.$$

Uzávěr jazyka L je roven sjednocení všech nezáporných celých mocnin jazyka L.

Průvodce studiem

$L^{i-1} \cdot L$ označuje zřetězení jazyků (množin) L^{i-1} a L zavedené v kapitole 1.1.

Příklad 1.52. Uvažme jazyk $L = \{a, b^2\}$. Potom

$L^0 = \{\varepsilon\}$,
$L^1 = L^0 \cdot L = \{\varepsilon\} \cdot \{a, b^2\} = \{a, b^2\}$,
$L^2 = L^1 \cdot L = \{a, b^2\} \cdot \{a, b^2\} = \{a^2, ab^2, b^2a, b^4\}$,
$L^3 = L^2 \cdot L = \{a^2, ab^2, b^2a, b^4\} \cdot \{a, b^2\} = \{a^3, a^2b^2, ab^2a, ab^4, b^2a^2, b^2ab^2, b^4a, b^6\}$,
atd.

Podle definice je $L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$ V našem případě tedy L^* obsahuje všechny řetězy nad abecedou $\{a, b\}$, jež mohly vzniknout zřetězením jakýchkoli počtů řetězů ε, a a b^2 v libovolném pořadí, a žádné jiné.

Definice 1.53. Nechť Σ je abeceda. Pak každý regulární výraz R nad abecedou Σ popisuje *jazyk* $L(R)$ nad Σ definovaný rekurzivně:

$$L(R) = \begin{cases} \emptyset, \text{ jestliže } R = \emptyset, \\ \{\varepsilon\}, \text{ jestliže } R = \varepsilon, \\ \{a\}, \text{ jestliže } R = a, \text{ kde } a \in \Sigma, \\ L(R_1) \cup L(R_2), \text{ jestliže } R = R_1 + R_2, \text{ kde } R_1, R_2 \text{ jsou regulární výrazy nad } \Sigma, \\ L(R_1) \cdot L(R_2), \text{ jestliže } R = R_1 \cdot R_2, \text{ kde } R_1, R_2 \text{ jsou regulární výrazy nad } \Sigma, \\ [L(R_1)]^*, \text{ jestliže } R = (R_1)^*, \text{ kde } R_1 \text{ je regulární výraz nad } \Sigma. \end{cases}$$

Příklad 1.54. Nechť $R = ab(a+b)^*$ je regulární výraz nad abecedou $\{a, b\}$. Pak je zřejmě

$$L(R) = L(R_1) \cdot L(R_2) \cdot L(R_3),$$

kde

$R_1 = a$, tj. $L(R_1) = \{a\}$,
$R_2 = b$, tj. $L(R_2) = \{b\}$,
$R_3 = (a+b)^*$, tj. $L(R_3) = (\{a\} \cup \{b\})^* = \{a, b\}^*$.

Odtud

$$L(R) = \{w \in \{a, b\}^*; \exists v \in \{a, b\}^* : w = abv\}.$$

(Tzn $L(R)$ neobsahuje žádný řetěz nad abecedou $\{a, b\}$, který nezačíná podřetězem ab.)

Příklad 1.55. Regulární výraz $(a^2 + b)c^*$ zřejmě popisuje jazyk zahrnující všechna slova začínající podřetězem a^2 nebo b následovaným libovolným (i nulovým) počtem c.

Průvodce studiem

Populárně řečeno, při zjišťování slov jazyka popsaného regulárním výrazem si stačí uvědomit, že symbol $+$ znamená výběr z poskytnutých možností a * znamená možnost libovolného (i nulového) počtu opakování.

Věta 1.56. *Třída regulárních jazyků je rovna třídě jazyků popsatelných regulárními výrazy.*

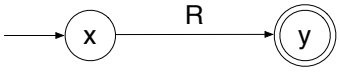
Regulární výrazy popisují třídu regulárních jazyků.

Důkaz. Přesný a úplný důkaz je zdlouhavý (viz např. [Sip97] nebo [Chy82]). Proto si uvedeme jen ideje, s jejichž pomocí jej lze provést. Popis rozdělíme do dvou částí. Vzhledem k větě 1.48, podle níž je třída regulárních jazyků rovna třídě jazyků přijímaných konečnými automaty, stačí umět převést každý regulární výraz na NKA reprezentující tentýž jazyk a umět převést každý KA na odpovídající regulární výraz.

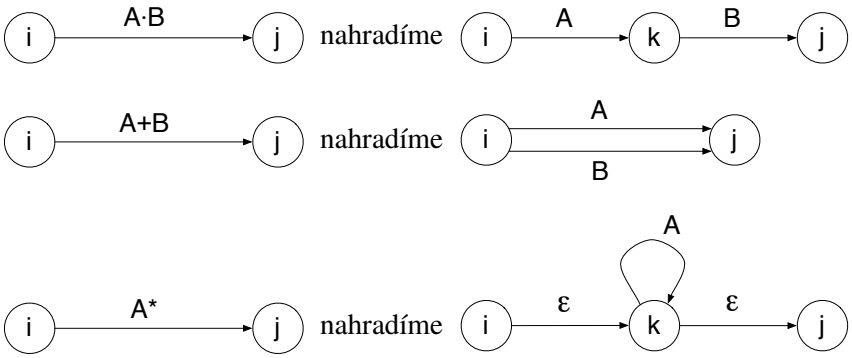
I. Převod regulárního výrazu R na odpovídající NKA:

NKA přijímající jazyk $L(R)$ bude popsán stavovým diagramem sestaveným posloupností následujících kroků.

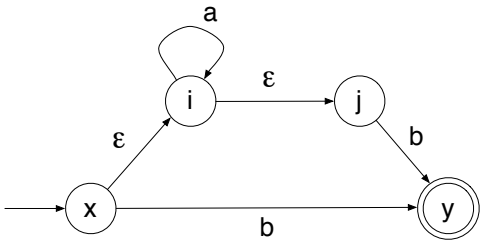
1) Sestavíme orientovaný graf se dvěma uzly označenými x a y, z uzlu x vede do uzlu y hrana označená výchozím regulárním výrazem R a uzel y je znázorněn dvojitým kroužkem. Navíc přidáme šipku mířící do uzlu x:



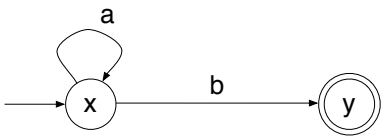
2) Postupně budeme rozkládat regulární výraz R na jednotlivé složky, z nichž je sestaven. Rozklad doprovázíme změnami grafu podle následujících pravidel.



3) Odstraníme hrany označené $\emptyset$ a upravíme výsledek tak, aby neobsahoval žádné hrany označené ε. Tj. např.

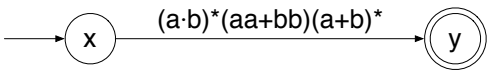


přepíšeme na

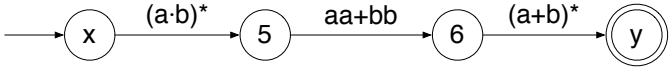


Demonstrace uvedeného postupu pro případ $R = (ab)^*(aa + bb)(a + b)^*$:

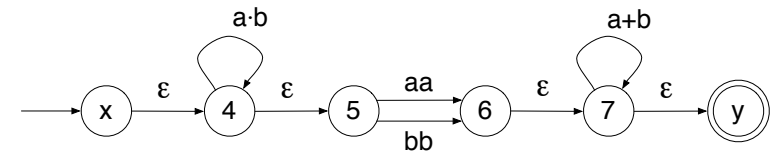
a)



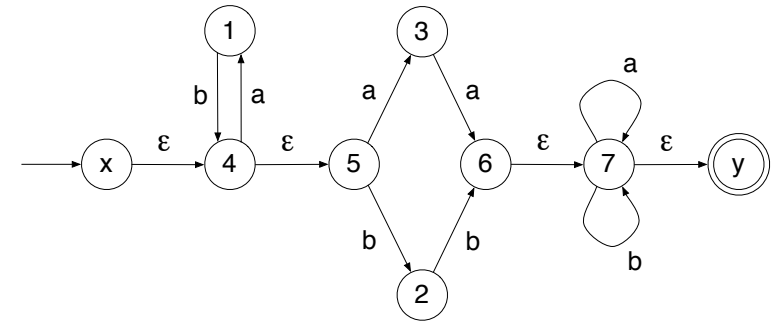
b)



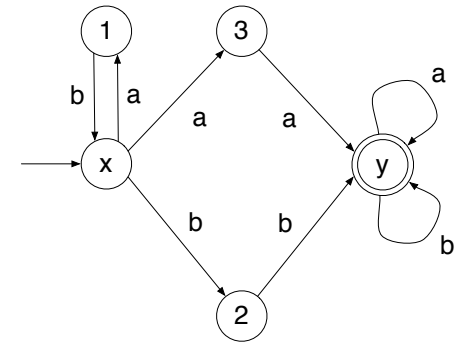
c)



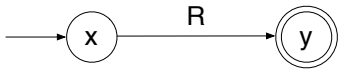
d)



e)



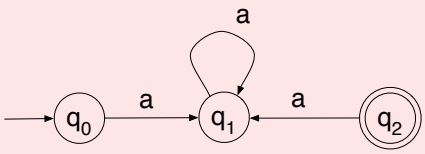
II. Převod KA A na odpovídající regulární výraz:

Postupně budeme upravovat stavový diagram konečného automatu A, dokud nezískáme mírně modifikovaný orientovaný graf tvaru 

, kde hrana mezi uzly x a y je označena regulárním výrazem R popisujícím právě jazyk $L(A)$.

Průvodce studiem

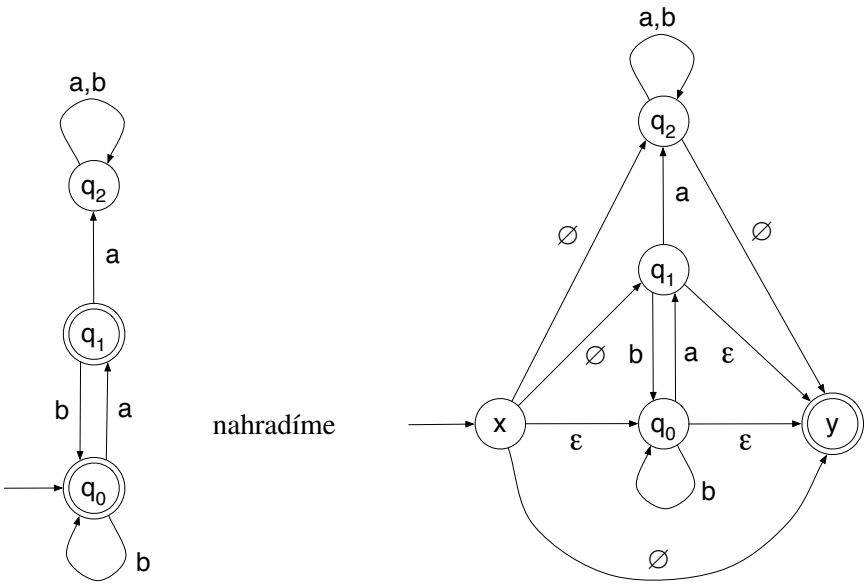
Zde se nabízí obrácený postup k části I. Ne vždy však budete mít zaručeno, že ze stavového diagramu KA dospějete k výchozí situaci části I. Viz např.



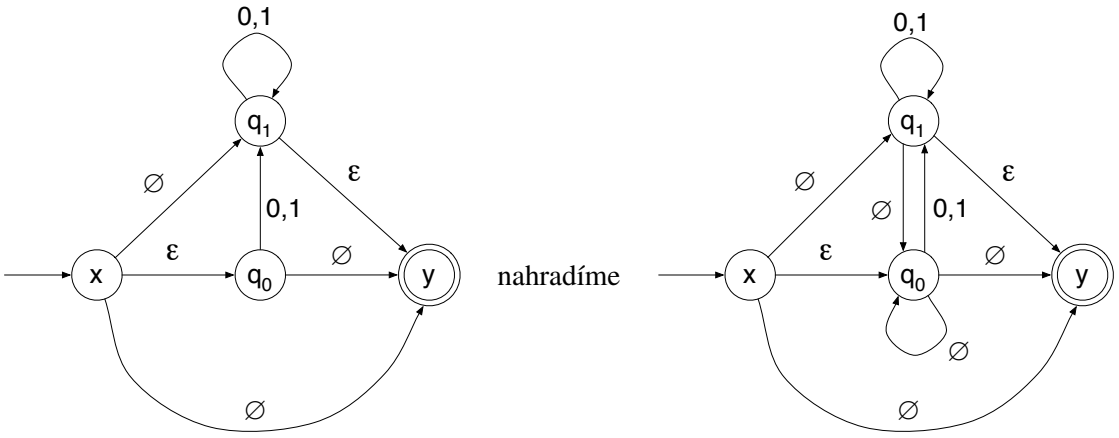
Zaručenou, i když delší metodu naleznete níže.

Použijeme posloupnost následujících úprav.

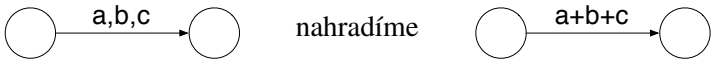
1) Ke stavovému diagramu A přidáme nový počáteční uzel x, z něj vedeme do původního počátečního uzlu q_0 hranu s označením ε a do všech ostatních uzlů vedeme hrany s označením $\emptyset$. Dále přidáme nový koncový uzel s označením y, do něj vedeme ze všech původních koncových uzlů hrany s označením ε a ze všech ostatních uzlů do něj vedeme hrany s označením $\emptyset$. Posléze přeznačíme diagram tak, aby byl x (y) jediným počátečním (koncovým) uzlem. Viz např.



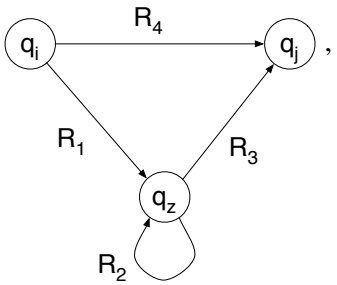
2) S výjimkou uzlů x, y doplníme případné chybějící hrany mezi každými dvěma uzly i případné chybějící smyčky. Nově dodané hrany označíme symbolem $\emptyset$. Např.



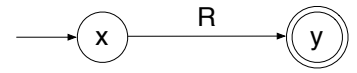
3) Označení každé hrany, které je tvaru $a_1, \ldots, a_k$, přepíšeme na regulární výraz $a_1 + \ldots + a_k$. Tj. např.



4) Postupně odstraňujeme všechny uzly s výjimkou x, y. Odstranění uzlu q_z provedeme následovně. Pro každou trojici uzlů (q_i, q_j, q_z), kde máme



změníme označení hrany mezi q_i a q_j na $R_4 + R_1(R_2)^* R_3$. Tuto úpravu provedeme i pro každou trojici uzlů (q_i, q_i, q_z), tj. změníme označení smyčky u uzlu q_i.

5) Závěrem máme výsledek tvaru 

, kde R je hledaný regulární výraz.

Průvodce studiem

Počítejte s tím, že R bude zbytečně komplikovaný. Téměř jistě bude obsahovat celou řadu nadbytečných symbolů ε a $\emptyset$. Určitě byste si však věděli rady s tím, jak zjednodušit podvýrazy tvaru $a \cdot \varepsilon$, $a \cdot \emptyset$, $a + \emptyset$ nebo $\emptyset^*$.

Tím je náš popis ideje důkazu u konce. □

Shrnutí

Uzávěrem jazyka L rozumíme sjednocení všech nezáporných mocnin L, přičemž L^0 označuje jazyk obsahující pouze prázdný řetěz, L^1 je L a L^n získáme zřetězením L^{n-1} a L.

Množina všech regulárních výrazů nad abecedou Σ je zavedena induktivně: Kromě výrazů $\emptyset, \varepsilon$ a všech symbolů ze Σ zahrnuje každý výraz tvaru $(R_1 + R_2)$, $(R_1 \cdot R_2)$ a $(R_1)^*$, kde R_1, R_2 jsou regulární výrazy.

Jazyk popsaný regulárním výrazem R je definován jako množina slov $L(R)$, k níž se dá dospět podobným induktivním způsobem jako k regulárnímu výrazu R, a sice: Jestliže je $R = \emptyset$ $(\varepsilon,\ a)$, pak je $L(R) = \emptyset$ $(\{\varepsilon\},\ \{a\})$. V případě $R = R_1^*$ ($R_1 + R_2$, resp. $R_1 \cdot R_2$) je $L(R)$ rovno uzávěru jazyka $L(R_1)$ (sjednocení, resp. zřetězení jazyků $L(R_1)$ a $L(R_2)$).

Platí, že třída všech jazyků popsatelných regulárními výrazy je rovna třídě regulárních jazyků. Regulární výrazy nám tak vedle regulárních gramatik a konečných automatů nabízejí další způsob reprezentace regulárních jazyků.

Pojmy k zapamatování

- Uzávěr jazyka,
- regulární výraz,
- jazyk popsaný regulárním výrazem.

Kontrolní otázky

1. *Je jazyk popsaný regulárním výrazem* *a)* $(a + ba)(\varepsilon + a)^*$
   b) $aba(ab + a)(\varepsilon^* + a)$
   c) $(\emptyset + a)^* b(a + \varepsilon)$
   konečný nebo nekonečný?
2. *Lze každý jazyk popsaný regulárním výrazem generovat pomocí bezkontextové gramatiky?*

Cvičení

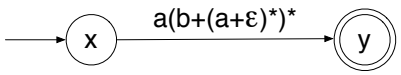
1. Jak vypadá jazyk popsaný regulárním výrazem $R_1 = (a + aa)(ba)^* ba$?
2. Najděte alespoň jedno slovo nepatřící do jazyka (nad abecedou $\{a, b\}$) popsaného regulárním výrazem $R_2 = a^* b^*$.
3. Převeďte regulární výraz $R_3 = a(b + (a + \varepsilon)^*)^*$ na NKA reprezentující tentýž jazyk.

Úkoly k textu

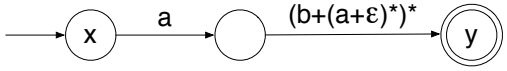
1. Ověřte, že regulární výrazy $b \cdot \varepsilon,\ b \cdot \emptyset,\ b + \emptyset$ a $\emptyset^*$ po řadě popisují tytéž jazyky jako regulární výrazy $b,\ \emptyset,\ b$ a ε.
2. Promyslete si detailně, zda totéž označení symbolem * u uzávěru abecedy a u uzávěru jazyka má nějakou souvislost.

Řešení

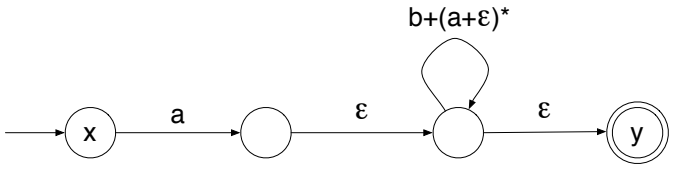
1. $L(R_1) = \{a(ba)^n;\ n \in \mathbb{N}\} \cup \{a^2(ba)^n;\ n \in \mathbb{N}\}$.
2. Do jazyka $L(R_2)$ nepatří žádné slovo, v němž se vyskytuje alespoň jedno a po symbolu b.
3. a)



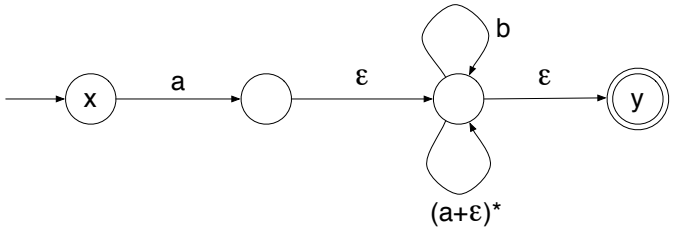
b)



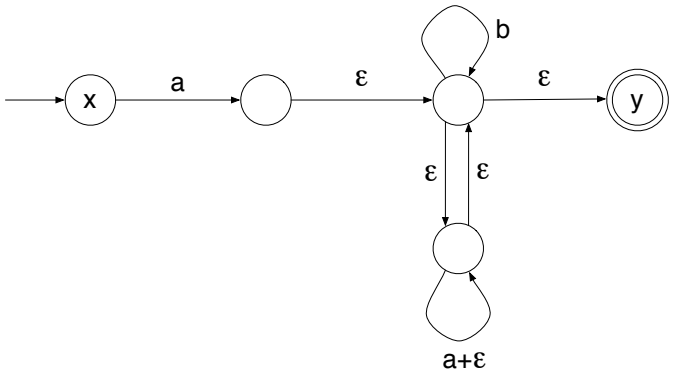
c)



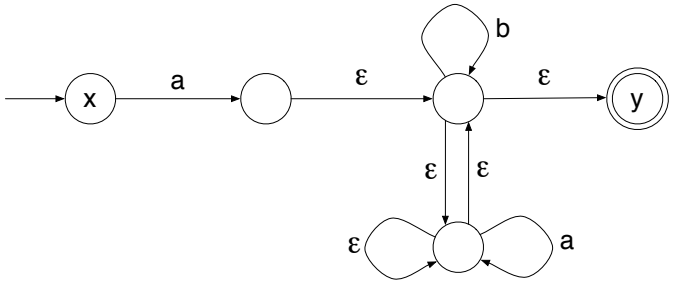
d)

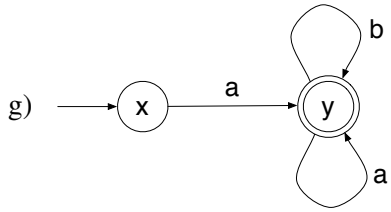


e)



f)





1.5 Zásobníkové automaty

Studijní cíle: Po prostudování kapitoly bude studující schopen formálně popsat zásobníkový automat a dva různé způsoby, jakými přijímá jazyk. Měl by být schopen objasnit vztah mezi zásobníkovými automaty a bezkontextovými gramatikami.

Klíčová slova: Zásobníkový automat, nedeterministický zásobníkový automat.

Potřebný čas: 120 minut.

ZA ∼ NKA rozšířený o zásobník

Dalším automatem, se kterým se seznámíme, je zásobníkový automat. Půjde o stroj, který si můžeme představit jako NKA rozšířený o paměť typu zásobník. Toto rozšíření povede k jistým odlišnostem vyplývajícím z nutnosti efektivně pracovat s uvedeným paměťovým zařízením.

Průvodce studiem

Pro úplnost si připomeňme, že zásobník představuje takový typ paměti, jenž umožňuje ukládat symboly z jisté abecedy; při vyjímání je však přístupný pouze posledně uložený symbol, nebyl-li mezitím ze zásobníku vyňat – v takovém případě je přístupný bezprostředně předcházející symbol, atd. Odtud vyplývá, že přístup k libovolnému symbolu v zásobníku získáme, odebereme-li ze zásobníku všechny později uložené symboly.

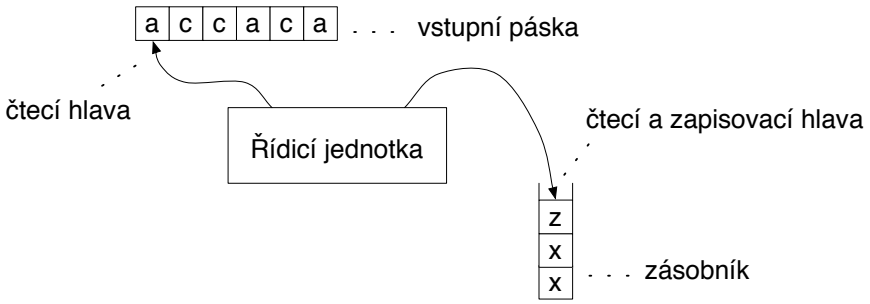
V souladu s běžně užívanou konvencí budeme nazývat poslední obsazenou paměťovou buňku (nebo chcete-li políčko) vrcholem zásobníku – vycházíme z představy svislého uspořádání paměťových buněk, přičemž je zaplňujeme odspodu nahoru.

Průvodce studiem

Pokud jste pozapomněli, z čeho sestává a jak pracuje NKA, raději si před studiem dalšího textu příslušné partie kapitoly 1.3 zopakujte.

Konkrétně lze tedy zásobníkový automat chápat jako zařízení z obr. 5 skládající se

- z řídicí jednotky, která se může nacházet v některém z konečně mnoha stavů,
- ze vstupní pásky stejné délky jako vstupní slovo, jež je na ní zapsáno,
- ze čtecího zařízení umožňujícího řídicí jednotce číst symbol z toho políčka vstupní pásky, nad nímž se nachází čtecí hlava,
- ze zásobníku představovaného potenciálně nekonečnou páskou (dělenou na jednotlivá políčka, přičemž na každém políčku může být nejvýše jeden symbol z takzvané zásobníkové abecedy),
- ze čtecího a zapisovacího zařízení umožňujícího řídicí jednotce číst i odstraňovat symbol z vrcholu zásobníku a zapisovat symboly „nad vrchol zásobníku“.



Obrázek 5: Ilustrace zásobníkového automatu

Průvodce studiem

Potenciálně nekonečnou pásku si můžete představit jako konečnou pásku, k níž lze kdykoli v případě potřeby přidat konečně mnoho dalších políček. Vzdálenou analogii by mohla představovat situace ukládání dat na CD – jakmile jedno zaplníte, položíte je na hromadu a do vypalovací mechaniky vložíte další CD.

Jak bude probíhat činnost výše uvedeného zařízení?

Neformální popis činnosti ZA

Výchozí předpoklady:

1) Na vstupní pásce bude zapsáno slovo tvořené ze symbolů vstupní abecedy zásobníkového automatu. Páska (dělená na jednotlivá políčka) bude mít na každém políčku právě jeden symbol a délka pásky bude stejná jako délka vstupního slova, tj. páska je konečné délky a na jejím konci nejsou žádná nevyužitá políčka.
2) Čtecí hlava je umístěna nad prvním (tj. nejlevějším) políčkem vstupní pásky (a je tedy nachystána ke čtení 1. symbolu).
3) V zásobníku je uložen jediný symbol – takzvaný počáteční symbol.
4) Čtecí a zapisovací hlava je umístěna nad vrcholem zásobníku.
5) Zásobníkový automat se nachází ve vyznačeném, takzvaném počátečním stavu.

Průběh výpočtu: Vycházíme z předpokladu, že řídicí jednotka má (díky čtecí i čtecí a zapisovací hlavě) vždy k dispozici informace o vstupním symbolu a vrcholu zásobníku. Výpočet probíhá nedeterministicky, přičemž v každém okamžiku má zásobníkový automat na výběr několik možností kroků výpočtu.

a) V první řadě má možnosti plynoucí ze znalosti vstupního symbolu, aktuálního stavu a symbolu z vrcholu zásobníku. Každá z těchto možností zahrnuje nový stav řídicí jednotky a (případně prázdný) řetěz zásobníkových symbolů, jimiž bude „přepsán“ vrchol zásobníku. V uvedeném případě automat přesune čtecí hlavu nad vstupní páskou o jedno políčko doprava, přejde do nového stavu, odstraní symbol z vrcholu zásobníku a do zásobníku zapíše příslušný řetěz, tj. je-li prázdný, nezapisuje nic, je-li délky 1, zapíše jej namísto odstraněného symbolu, a je-li délky alespoň 2, jeho 1. symbol zprava zapíše do zásobníku nejdříve, pak zapíše 2. symbol zprava atd. (tzn. po provedení zápisu se nejlevější symbol ocitne na vrcholu zásobníku).

Řetěz bývá do zásobníku ukládán „zprava doleva“.

b) V druhé řadě má možnosti obdobné bodu a), avšak lišící se tím, že není vyžadována žádná znalost vstupního symbolu. Tzn. automat pro svůj výpočetní krok potřebuje znát pouze aktuální stav a symbol z vrcholu zásobníku. Výsledkem je potom přechod do nového stavu a „přepsání“ vrcholu zásobníku – jinými slovy, automat „nepokračuje ve čtení vstupního slova“, pouze mění svůj stav a obsah zásobníku.

Informace o tom, jak se má zásobníkový automat konkrétně zachovat, budou součástí jeho popisu. Jmenovitě půjde o přechodovou funkci, která bude přiřazovat každé trojici (stav,

symbol vstupní abecedy nebo prázdný řetěz, symbol zásobníkové abecedy) množinu dvojic tvaru (stav, řetěz zásobníkových symbolů).

Vyhodnocení výpočtu: Uvedeme si dva způsoby, jak může zásobníkový automat přijímat vstupní slovo. Jeden způsob je naprosto analogický přijetí vstupního slova nedeterministickým konečným automatem, tj. je vyžadována existence „větve výpočtu“ končící „po přečtení vstupního slova“ v některém z vyznačených koncových stavů. Druhý způsob využívá přítomnosti paměťového zařízení a vyžaduje existenci „větve výpočtu“ končící „po přečtení vstupního slova“ vyprázdněním zásobníku. U každého zásobníkového automatu pochopitelně výslovně uvedeme, který z popsaných způsobů přijímání vstupních slov budeme používat.

Nyní přejdeme k formální definici zásobníkového automatu.

Definice 1.57. *Zásobníkovým automatem* (zkráceně ozn. *ZA*)[4] nazýváme uspořádanou sedmici $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde

Q je konečná neprázdná množina *stavů*,
Σ je *vstupní abeceda*,
Γ je *zásobníková abeceda*,
δ je *přechodová funkce*; zobrazuje $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ do množiny všech konečných podmnožin $Q \times \Gamma^*$,
$q_0 \in Q$ je *počáteční stav*,
$Z_0 \in \Gamma$ je *počáteční symbol*,
$F \subseteq Q$ je *množina koncových stavů*.

Průvodce studiem

V definici neklademe žádné omezující podmínky na vzájemný vztah vstupní a zásobníkové abecedy. To proto, aby bylo možno do zásobníku ukládat jak přečtené vstupní symboly tak pomocné symboly, jež se ve vstupní abecedě nevyskytují. Další komentář – týkající se přechodové funkce – odložíme až za následující definici.

Definice 1.58. Mějme libovolný ZA $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Každou uspořádanou trojici $(q, w, \alpha) \in Q \times \Sigma^* \times \Gamma^*$ nazveme *konfigurací zásobníkového automatu* A.

Krok výpočtu A definujeme jako binární relaci $\vdash$ (na množině všech konfigurací) takto: Pro všechna $p, q \in Q, a \in \Sigma \cup \{\varepsilon\}, x \in \Sigma^*, Z \in \Gamma, \alpha, \beta \in \Gamma^*$ je $(p, ax, Z\alpha) \vdash (q, x, \beta\alpha)$, právě když $\delta(p, a, Z)$ obsahuje (q, β).

Symbolem $\vdash^*$ budeme označovat reflexivní a tranzitivní uzávěr relace $\vdash$, symbolem $\vdash^+$ tranzitivní uzávěr relace $\vdash$.

Výpočet zásobníkového automatu definujeme prostřednictvím relace $\vdash^*$.

Průvodce studiem

Podobně, jako tomu bylo u KA a NKA, i v případě ZA konfigurace jednoznačně popisuje situaci, v níž se stroj nachází v průběhu výpočtu. Tuto situaci zřejmě určuje aktuální stav, dosud nepřečtená část vstupního slova a aktuální obsah zásobníku. Tj. konfiguraci (q, w, α) odpovídá situace, kdy se daný ZA nachází ve stavu q, dosud nepřečtená část vstupního slova je rovna w a v zásobníku je zapsán řetěz α, přičemž vrchol zásobníku obsahuje nejlevější symbol řetězu α.

V definici kroku výpočtu si všimněte požadavku na neprázdný obsah zásobníku. Jinak

[4] Zatímco KA bez jakéhokoliv přívlastku označoval deterministickou verzi uvedeného stroje, ZA bez dalšího přívlastku nám bude označovat verzi nedeterministickou – jednak v tomto textu deterministický zásobníkový automat zavádět nebudeme, jednak je v literatuře tato „asymetrie“ běžná.

řečeno, dojde-li v průběhu výpočtu k vyprázdnění zásobníku, není další výpočetní krok definován.

Co se týče výpočetních variant, lze spatřit dva různé důvody nedeterminismu u ZA. Oba vyplývají z definice přechodové funkce. Jednak má ZA na výběr, zda číst nebo nečíst vstupní symbol, jednak výsledek přechodové funkce může nabízet různá přípustná pokračování výpočtu.

Dvě různé příčiny nedeterminismu u ZA

Tak například pro $\delta(p, a, Z) = \{(q_1, a), (q_2, \varepsilon)\}$ a $\delta(p, \varepsilon, Z) = \{(q_3, Z), (q_4, aZ)\}$ máme v binární relaci $\vdash$ následující dvojice konfigurací:

$(p, a, Z) \vdash (q_1, \varepsilon, a)$,
$(p, a, Z) \vdash (q_2, \varepsilon, \varepsilon)$,
$(p, a, Z) \vdash (q_3, a, Z)$,
$(p, a, Z) \vdash (q_4, a, aZ)$.

Výpočet obsahující konfiguraci (p, a, Z) bude tedy zahrnovat čtyři „větve výpočtu“, z nichž každá pokračuje po konfiguraci (p, a, Z) jinou z výše uvedených konfigurací.

Jak bylo již dříve předesláno, pro ZA lze zavést dva různé způsoby přijetí vstupního slova, což obecně vede ke dvěma různým jazykům, jež přijímá tentýž ZA.

Definice 1.59. Nechť $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je ZA.

ZA může jazyk přijímat koncovým stavem nebo prázdným zásobníkem.

a) Množinu $L(A) = \{w \in \Sigma^*;\ (q_0, w, Z_0) \vdash^* (q_f, \varepsilon, \alpha)$ pro nějaké $q_f \in F,\ \alpha \in \Gamma^*\}$, nazveme *jazykem přijímaným (rozpoznávaným) zásobníkovým automatem A koncovým stavem*.

b) Množinu $N(A) = \{w \in \Sigma^*;\ (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$ pro nějaké $q \in Q\}$, nazveme *jazykem přijímaným (rozpoznávaným) zásobníkovým automatem A prázdným zásobníkem*.

Průvodce studiem

Jazyk přijímaný koncovým stavem zosobňuje stejný přístup, který jste mohli vidět u NKA, tj. do tohoto jazyka je zařazeno každé slovo, k němuž existuje výpočet končící „po přečtení daného slova“ v některém z koncových stavů. V případě ZA však nevnímejte vyjádření „přečtení slova“ jako okamžik, kdy byl přečten poslední symbol vstupního slova. ZA totiž může ještě provádět výpočetní kroky, jimiž pouze mění svůj stav a obsah zásobníku.

Analogicky, jazyk přijímaný prázdným zásobníkem obsahuje každé slovo, k němuž existuje výpočet, jenž „po přečtení daného slova“ končí konfigurací s prázdným zásobníkem. Poznámka z předchozího odstavce o možnosti měnit na závěr výpočtu pouze stav a obsah zásobníku platí pochopitelně i zde.

Příklad 1.60. Popišme si ZA, který bude přijímat jazyk $L = \{a^n b^n;\ n \geq 1\}$ koncovým stavem.

Nejjednodušší způsob činnosti požadovaného ZA bude následující. Dokud bude číst symboly a, setrvá v počátečním stavu q_0 a za každý přečtený symbol a uloží do zásobníku jeden symbol a. Tím bude vlastně zajištěno zkopírování všech symbolů a do zásobníku. Při přečtení prvního symbolu b přejde ZA do stavu q_1 a ze zásobníku odebere jeden symbol a. Ve stavu q_1 ZA setrvá, dokud bude číst symboly b – přitom bude odstraňovat symboly a ze zásobníku. Zůstane-li mu v zásobníku pouze symbol Z_0, přejde do koncového stavu, neboť právě dočetl řetěz tvaru $a^n b^n$.

Příslušnému popisu nepochybně odpovídá ZA $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde

$Q = \{q_0, q_1, q_2\}$,
$\Sigma = \{a, b\}$,
$\Gamma = \{Z_0, a\}$,
$F = \{q_2\}$

a přechodová funkce δ je definována následovně:

$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$,
$\delta(q_0, a, a) = \{(q_0, aa)\}$,
$\delta(q_0, b, a) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$,
$\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$,
ve všech ostatních případech je $\delta(x, y, z) = \emptyset$.

Pro větší názornost si uveďme příklad výpočetní větve potvrzující příslušnost slova a^2b^2 k jazyku L a příklad výpočetní větve pro slovo $aab \notin L$:

$(q_0, aabb, Z_0) \vdash (q_0, abb, aZ_0) \vdash (q_0, bb, aaZ_0) \vdash (q_1, b, aZ_0) \vdash (q_1, \varepsilon, Z_0) \vdash (q_2, \varepsilon, Z_0)$

$(q_0, aab, Z_0) \vdash (q_0, ab, aZ_0) \vdash (q_0, b, aaZ_0) \vdash (q_1, \varepsilon, aZ_0) \ldots$ a další krok výpočtu již není definován.

I když jsme zavedli dva různé způsoby, jak může ZA přijímat jazyk, následující věta potvrdí, že odpovídající třídy jazyků jsou stejné.

Třídy jazyků přijímaných zásobníkovými automaty koncovým stavem a prázdným zásobníkem jsou stejné.

Věta 1.61. *Jazyk $L \subseteq \Sigma^*$ je přijímaný nějakým ZA A_1 koncovým stavem, právě když je přijímaný nějakým ZA A_2 prázdným zásobníkem.*

Důkaz.

1) Nechť je jazyk L přijímán ZA $A_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ koncovým stavem. Sestrojíme ZA A_2, který bude přijímat L prázdným zásobníkem.

Průvodce studiem

Idea konstrukce bude celkem prostá. Stačí, aby automat A_2 pracoval stejně jako A_1 a aby, kdykoliv se A_1 ocitne v koncovém stavu, mohl (nedeterministicky) přejít do speciálního stavu, v němž vyprázdní svůj zásobník. Při vlastní realizaci této myšlenky však musíme vyřešit drobný zádrhel spočívající v eventualitě, že ZA A_1 může vyprázdnit svůj zásobník u slova nepatřícího do L – v takovém případě by ZA A_2 (pracující stejně jako A_1) ovšem dotyčné slovo přijal. Uvedený zádrhel řeší uložení speciálního symbolu do zásobníku stroje A_2 ještě před tím, než začne simulovat činnost stroje A_1. Při vyprázdnění zásobníku A_1 pak v zásobníku A_2 zůstane ještě onen speciální symbol.

Jistě lze bez újmy na obecnosti předpokládat, že množina stavů Q neobsahuje stavy q_0', q_ε a zásobníková abeceda Γ neobsahuje symbol Z_0'. Potom stačí položit
$A_2 = (Q \cup \{q_0', q_\varepsilon\}, \Sigma, \Gamma \cup \{Z_0'\}, \delta', q_0', Z_0', \emptyset)$,
kde přechodová funkce δ' je definována následovně:

U ZA přijímajícího prázdným zásobníkem nezřídka volíme prázdnou množinu koncových stavů.

Průvodce studiem

Nenechte se překvapit tím, že množina koncových stavů stroje A_2 je prázdná. Nenastává zde žádný rozpor s definicí, především však tato množina nehraje žádnou roli při přijímání slov prázdným zásobníkem.

a) $\delta'(q_0', \varepsilon, Z_0') = \{(q_0, Z_0 Z_0')\}$,
b) pro všechna $p, q \in Q,\ a \in \Sigma \cup \{\varepsilon\},\ Z \in \Gamma,\ \alpha \in \Gamma^*$ je
$(q, \alpha) \in \delta'(p, a, Z)$, právě když $(q, \alpha) \in \delta(p, a, Z)$,
c) pro všechna $q \in F,\ Z \in \Gamma \cup \{Z_0'\}$ je $(q_\varepsilon, \varepsilon) \in \delta'(q, \varepsilon, Z)$,
d) pro všechna $Z \in \Gamma \cup \{Z_0'\}$ je $\delta'(q_\varepsilon, \varepsilon, Z) = \{(q_\varepsilon, \varepsilon)\}$,
e) ve všech zbývajících případech je $\delta'(x, y, z) = \emptyset$.

Průvodce studiem

Bod a) zaručuje převedení A_2 do konfigurace stejné jako je počáteční konfigurace A_1 až na zásobník bohatší o symbol Z_0'.

Bod b) zajišťuje stejné výpočetní možnosti jako má A_1.

Bod c) poskytuje stroji A_2 možnost přejít do stavu q_ε, kdykoliv se A_1 nachází v některém ze svých koncových stavů.

Bod d) umožňuje stroji A_2 ve stavu q_ε vyprázdnit zásobník.

Nyní ověříme, že ZA A_2 přijímá prázdným zásobníkem právě jazyk $L = L(A_1)$.
Nechť $w \in \Sigma^*$. Potom jsou na základě definic jazyků přijímaných zásobníkovými automaty a definice stroje A_2 ekvivalentní následující tvrzení.

- $w \in L(A_1)$,
- existují $q \in F$, $n \in \mathbb{N}_0$ a $Z_1, \ldots, Z_n \in \Gamma$ takové, že u ZA A_1 je možný výpočet $(q_0, w, Z_0) \vdash^* (q, \varepsilon, Z_1 \cdots Z_n)$,
- existují $q \in F$, $n \in \mathbb{N}_0$ a $Z_1, \ldots, Z_n \in \Gamma$ takové, že u ZA A_2 je možný výpočet $(q_0', w, Z_0') \vdash (q_0, w, Z_0 Z_0') \vdash^* (q, \varepsilon, Z_1 \cdots Z_n Z_0') \vdash^* (q_\varepsilon, \varepsilon, \varepsilon)$,
- $w \in N(A_2)$.

2) Předpokládejme nyní, že jazyk L je přijímán ZA $A_2 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ prázdným zásobníkem. Sestrojíme ZA A_1, který bude přijímat L koncovým stavem.

Průvodce studiem

Idea konstrukce bude do jisté míry podobná části 1) tohoto důkazu. Stačí, aby hledaný automat A_1 vždy na začátku své práce vložil do zásobníku Z_0 nad svůj počáteční symbol a následně pracoval stejně jako A_2. V okamžiku, kdy A_2 vyprázdní svůj zásobník, zůstane stroji A_1 v zásobníku jeho počáteční symbol – jeho načtení převede stroj A_1 do koncového stavu.

Předpokládejme (bez újmy na obecnosti), že množina stavů Q neobsahuje stavy q_0', q_f a zásobníková abeceda Γ neobsahuje symbol Z_0'. Potom stačí položit
$A_1 = (Q \cup \{q_0', q_f\}, \Sigma, \Gamma \cup \{Z_0'\}, \delta', q_0', Z_0', \{q_f\})$,
kde přechodová funkce δ' je definována následovně:

a) $\delta'(q_0', \varepsilon, Z_0') = \{(q_0, Z_0 Z_0')\}$,
b) pro všechna $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $Z \in \Gamma$ je $\delta'(q, a, Z) = \delta(q, a, Z)$,
c) pro všechna $q \in Q$ je $\delta'(q, \varepsilon, Z_0') = \{(q_f, \varepsilon)\}$,
d) ve všech zbývajících případech je $\delta'(x, y, z) = \emptyset$.

Průvodce studiem

Bod a) zaručuje převedení A_1 do konfigurace stejné jako je počáteční konfigurace A_2 až na zásobník bohatší o symbol Z_0'.

Bod b) zajišťuje stejné výpočetní možnosti jako má A_2.

Bod c) dává stroji A_1 možnost přejít do koncového stavu q_f, kdykoliv A_2 vyprázdnil svůj zásobník.

K dokončení důkazu zbývá ověřit, že ZA A_1 přijímá koncovým stavem jazyk $L = N(A_2)$.
Nechť $w \in \Sigma^*$. Potom jsou na základě definic jazyků přijímaných zásobníkovými automaty a definice stroje A_1 ekvivalentní následující tvrzení.

- $w \in N(A_2)$,
- existuje $q \in Q$ tak, že u ZA A_2 je možný výpočet $(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$,
- existuje $q \in Q$ tak, že u ZA A_1 je možný výpočet $(q_0', w, Z_0') \vdash (q_0, w, Z_0 Z_0') \vdash^* (q, \varepsilon, Z_0') \vdash (q_f, \varepsilon, \varepsilon)$,
- $w \in L(A_1)$. □

Pojem zavedený v následující definici se uplatní v důkazu další věty.

Definice 1.62. Levým odvozením v bezkontextové gramatice rozumíme takovou derivaci $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n$, kde v každém přímém odvození $\alpha_{k-1} \Rightarrow \alpha_k$ (pro $k \in \{1, \ldots, n\}$) je použito pravidlo přepisující první neterminál v řetězu α_{k-1}.

Příklad 1.63. Uvažme bezkontextovou gramatiku obsahující pravidla

$S \to AB$,
$A \to aAB \mid a$,
$B \to b$.

Podle předchozí definice $S \Rightarrow AB \Rightarrow aABB \Rightarrow aaBB \Rightarrow aabB \Rightarrow aabb$ představuje levé odvození řetězu $aabb$.

Příklad odvození, které není levé: $S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$

Poznámka 1.64. Každé slovo z jazyka generovaného bezkontextovou gramatikou je v důsledku bezkontextového používání pravidel odvoditelné z počátečního symbolu pomocí levého odvození.

Průvodce studiem

Závěr kapitoly věnujeme objasnění vzájemného vztahu bezkontextových jazyků a zásobníkových automatů. Můžeme předeslat, že třída bezkontextových jazyků je rovna třídě jazyků přijímaných zásobníkovými automaty (přičemž podle věty 1.61 není podstatné, jakým způsobem zásobníkové automaty jazyky přijímají).

Věta 1.65. *Ke každé bezkontextové gramatice G existuje ZA M takový, že $L(G) = N(M)$.*

Důkaz. Nechť $G = (N, \Sigma, P, S)$ je libovolná bezkontextová gramatika.

Průvodce studiem

Důkaz bude založen na následující myšlence. V zásobníku automatu M budeme nedeterministicky simulovat levá odvození řetězů generovatelných gramatikou G – viz také poznámku 1.64. Pokud se kterýkoliv z těchto řetězů objeví na vstupu ZA M, pak bude existovat „větev výpočtu“ M, kdy se 1. symbol vstupního slova bude rovnat symbolu z vrcholu zásobníku (kde bylo nejdříve nasimulováno odpovídající levé odvození) a oba symboly mohou být „odstraněny“. Totéž může proběhnout se všemi následujícími symboly, a tak ZA M může s přečtením takového vstupního slova vyprázdnit svůj zásobník, což je přesně to, co potřebujeme.

Předveďme si uvedený přístup na levém odvození $S \Rightarrow ABC \Rightarrow xBC \Rightarrow xyC \Rightarrow xyz$. ZA M by mohl provést pro vstupní slovo xyz následující posloupnost kroků výpočtu: $(q_0, xyz, Z_0) \vdash (q_0, xyz, S) \vdash (q_0, xyz, ABC) \vdash (q_0, xyz, xBC) \vdash^* (q_0, yz, BC) \vdash (q_0, yz, yC) \vdash^* (q_0, z, C) \vdash (q_0, z, z) \vdash^* (q_0, \varepsilon, \varepsilon)$.

Položme $M = (\{q_0\}, \Sigma, N \cup \Sigma, \delta, q_0, S, \emptyset)$, kde přechodová funkce δ je definována následovně:

Průvodce studiem

Protože v zásobníku potřebujeme simulovat levá odvození gramatiky G, obsahuje zásobníková abeceda stroje M všechny terminální i neterminální symboly gramatiky G a roli počátečního symbolu (v zásobníku) plnohodnotně zastoupí počáteční neterminál S. Bezvýznamnost množiny koncových stavů pro ZA přijímající prázdným zásobníkem snad již není třeba komentovat.

a) Pro všechna $A \in N,\ \alpha \in (N \cup \Sigma)^*$ taková, že $A \to \alpha \in P$, je $(q_0, \alpha) \in \delta(q_0, \varepsilon, A)$.
b) Pro všechna $a \in \Sigma$ je $\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$.
c) Ve všech zbývajících případech je $\delta(x, y, z) = \emptyset$.

Průvodce studiem

Bod a) zaručuje možnost simulovat v zásobníku levá odvození gramatiky G.

Bod b) umožňuje stroji M odebírat z vrcholu zásobníku stejné symboly, jaké jsou aktuálně na vstupu.

Ověření, že ZA M přijímá prázdným zásobníkem jazyk $L(G)$, rozdělíme na dvě části.

I. Prokážeme, že každé slovo z jazyka $L(G)$ patří do jazyka $N(M)$.
Nechť $w \in \Sigma^*$. Matematickou indukcí nejdříve dokážeme tvrzení T:

Jestliže $A \in N$, pak pro každé nezáporné celé číslo m ze vztahu $A \Rightarrow^m w$ (tzn. w je odvoditelné z A v m derivačních krocích) plyne $(q_0, w, A) \vdash^* (q_0, \varepsilon, \varepsilon)$.

1) Nechť $A \Rightarrow w = w_1 \cdots w_k$, kde $k \in \mathbb{N}_0$ a $w_1, \ldots, w_k \in \Sigma$. Potom z definice přímého odvození plyne existence pravidla $A \to w_1 \cdots w_k$ v množině P, což vzhledem k bodu a) definice přechodové funkce automatu M znamená, že $(q_0, w_1 \cdots w_k) \in \delta(q_0, \varepsilon, A)$. Na základě definic kroku výpočtu ZA a přechodové funkce automatu M pak dostáváme:
$(q_0, w_1 \cdots w_k, A) \vdash (q_0, w_1 \cdots w_k, w_1 \cdots w_k) \vdash^* (q_0, \varepsilon, \varepsilon)$.
2) Nechť tvrzení T platí pro $m \in \{1, \ldots, n\}$, kde $n \in \mathbb{N}$. Ověříme jeho platnost pro $n + 1$:
Nechť $A \Rightarrow^{n+1} w$. Tuto derivaci lze vzhledem k poznámce 1.64 psát jako levé odvození $A \Rightarrow A_1 \cdots A_j \Rightarrow^{m_1} \alpha_1 A_2 \cdots A_j \Rightarrow^{m_2} \ldots \Rightarrow^{m_j} \alpha_1 \alpha_2 \cdots \alpha_j = w$, kde $m_i \leq n$ pro každý index $i \in \{1, \ldots, j\}$.

Průvodce studiem

V předchozím zápisu je formálně popsána skutečnost, že z neterminálu A lze v jednom derivačním kroku (přímo) odvodit řetěz $A_1 \cdots A_j$ a že každé A_i je buď rovno terminálu α_i anebo z něj lze nejvýše n derivačními kroky odvodit terminální řetěz α_i. Přitom zřetězení $\alpha_1 \cdots \alpha_j$ je rovno řetězu w.

Z definice přímého odvození plyne existence pravidla $A \to A_1 \cdots A_j$ v množině P, což vzhledem k bodu a) definice přechodové funkce automatu M znamená, že $(q_0, A_1 \cdots A_j) \in \delta(q_0, \varepsilon, A)$. S využitím definice kroku výpočtu ZA a indukčního předpokladu použitého na každé odvození $A_i \Rightarrow^{m_i} \alpha_i$ pak dostáváme:
$(q_0, w, A) = (q_0, \alpha_1 \cdots \alpha_j, A) \vdash (q_0, \alpha_1 \cdots \alpha_j, A_1 \cdots A_j) \vdash^*$
$(q_0, \alpha_2 \cdots \alpha_j, A_2 \cdots A_j) \vdash^* \ldots \vdash^* (q_0, \varepsilon, \varepsilon)$.

Tvrzení T pro $A = S$ dává požadovanou inkluzi $L(G) \subseteq N(M)$.

II. Ověření, že každé slovo z jazyka $N(M)$ patří do jazyka $L(G)$, plyne přímo z tvrzení T':

Jestliže $w \in \Sigma^*, A \in N, n \in \mathbb{N}$, pak ze vztahu $(q_0, w, A) \vdash^n (q_0, \varepsilon, \varepsilon)$ (tzn. z úvodní konfigurace je závěrečná konfigurace dosažitelná po n výpočetních krocích) plyne $A \Rightarrow^* w$.

Tvrzení T' lze snadno dokázat matematickou indukcí (obdobně jako tvrzení T v části I.).

□

Ukažme si v následujícím příkladu konstrukci i činnost zásobníkového automatu odpovídajícího konkrétní gramatice.

Příklad 1.66. Nechť bezkontextová gramatika G obsahuje pouze následující pravidla:

$S \to ASB \mid AB$,
$A \to ab$,
$B \to b$.

Ověření příslušnosti řetězu $(ab)^2b^2$ k jazyku $L(G)$ si demonstrujme za pomoci ZA M z věty 1.65. Podle konstrukce M popsané v důkazu uvedené věty je přechodová funkce stroje M definována následovně:

$\delta(q_0, \varepsilon, S) = \{(q_0, ASB), (q_0, AB)\}$,
$\delta(q_0, \varepsilon, A) = \{(q_0, ab)\}$,
$\delta(q_0, \varepsilon, B) = \{(q_0, b)\}$,
$\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$,
$\delta(q_0, b, b) = \{(q_0, \varepsilon)\}$
a ve všech zbývajících případech je $\delta(x, y, z) = \emptyset$.

Větev výpočtu ZA M přijímající vstupní slovo $(ab)^2b^2$ pak je

$$(q_0, (ab)^2b^2, S) \vdash (q_0, (ab)^2b^2, ASB) \vdash (q_0, (ab)^2b^2, abSB) \vdash^* (q_0, ab^3, SB) \vdash$$
$$\vdash (q_0, ab^3, ABB) \vdash (q_0, ab^3, abBB) \vdash^* (q_0, b^2, BB) \vdash^* (q_0, \varepsilon, \varepsilon).$$

Věta 1.67. *Ke každému ZA M existuje bezkontextová gramatika G taková, že $N(M) = L(G)$.*

Důkaz. Důkaz je poněkud zdlouhavý, a proto jej z úsporných důvodů vynecháme. Zájemce odkazujeme na [Chy82], [Hop69] nebo [Sip97]. □

Důsledek 1.68. *Třída bezkontextových jazyků je rovna třídě jazyků přijímaných zásobníkovými automaty.*

Každý bezkontextový jazyk lze popsat bezkontextovou gramatikou nebo zásobníkovým automatem.

Důkaz. Jde o přímý důsledek vět 1.65 a 1.67. □

Shrnutí

ZA je zařízení vybavené konečněstavovou řídicí jednotkou, vstupní páskou konečné délky (včetně čtecí hlavy) a pamětí typu zásobník (včetně čtecí a zapisovací hlavy). Každý ZA je nedeterministický stroj a je jednoznačně určen

- množinou stavů (v nichž se může nacházet řídicí jednotka),
- vstupní abecedou Σ (vstupní slova ZA jsou tvořeny symboly ze Σ),
- zásobníkovou abecedou Γ (do zásobníku jsou ukládány symboly z Γ),
- přechodovou funkcí δ, která každé trojici (stav, symbol vstupní abecedy nebo prázdný řetěz, symbol z vrcholu zásobníku) přiřazuje konečnou množinu uspořádaných dvojic typu (stav, řetěz zásobníkových symbolů),
- jedním vyznačeným stavem (tzv. počátečním stavem – v něm bude řídicí jednotka před zahájením práce daného ZA, ať je na vstupu jakékoli vstupní slovo),
- jedním vyznačeným zásobníkovým symbolem (tzv. počátečním symbolem – ten bude jako jediný uložený v zásobníku před zahájením práce daného ZA, ať je na vstupu jakékoli vstupní slovo),
- vyznačenou podmnožinou množiny stavů (tzv. množinou koncových stavů – ta bude hrát důležitou roli při jednom ze dvou možných způsobů přijímání vstupních slov).

Výpočet ZA probíhá obecně nedeterministicky. V každém kroku výpočtu může ZA z aktuálního stavu p na základě znalosti symbolu Z z vrcholu zásobníku

a) a čteného vstupního symbolu a přejít do nového stavu q, posunout čtecí hlavu za přečtený vstupní symbol a „přepsat vrchol zásobníku novým řetězem β“, přičemž dvojice (q, β) je obsažena ve výsledku přechodové funkce δ pro argumenty p, a, Z.
b) přejít do nového stavu q a „přepsat vrchol zásobníku novým řetězem β“, přičemž dvojice (q, β) je obsažena ve výsledku přechodové funkce δ pro argumenty p, ε, Z.

Z možnosti kroku výpočtu uvedené pod bodem b) vyplývá, že ZA může měnit svůj stav a obsah zásobníku, aniž by jakkoli zohledňoval vstupní symbol pod svou čtecí hlavou. Ve srovnání s NKA je tak nedeterminismus ZA posílen o „rozhodování“, zda v aktuálním výpočetním kroku číst či nečíst další symbol vstupního slova.

Díky nedeterminismu (vyplývajícímu z definice přechodové funkce) se výpočet ZA může větvit do více variant. Každý výpočet ovšem začíná v počáteční konfiguraci, která je určena počátečním stavem, celým vstupním slovem na pásce (se čtecí hlavou nad 1. symbolem tohoto slova) a zásobníkem obsahujícím pouze počáteční symbol. „Příznivá“ ukončení výpočtu jsou dvě:

1) Pokud pro zadané vstupní slovo existuje alespoň jedna výpočetní větev končící „po přečtení uvedeného slova“ v některém z koncových stavů, říkáme, že ZA přijímá dané slovo koncovým stavem. Množina všech slov nad vstupní abecedou, jež jsou přijímána tímto způsobem, tvoří jazyk přijímaný zásobníkovým automatem koncovým stavem.
2) Pokud pro zadané vstupní slovo existuje alespoň jedna výpočetní větev končící „po přečtení uvedeného slova“ vyprázdněním zásobníku, říkáme, že ZA přijímá dané slovo prázdným zásobníkem. Množina všech slov nad vstupní abecedou, jež jsou přijímána tímto způsobem, tvoří jazyk přijímaný zásobníkovým automatem prázdným zásobníkem.

Třída všech jazyků přijímaných zásobníkovými automaty koncovým stavem je rovna třídě všech jazyků přijímaných zásobníkovými automaty prázdným zásobníkem.

Třída všech jazyků přijímaných zásobníkovými automaty (koncovým stavem nebo prázdným zásobníkem) je rovna třídě bezkontextových jazyků, tj. třídě všech jazyků generovatelných bezkontextovými gramatikami.

Pojmy k zapamatování

- Zásobníkový automat (ZA),
- stav,
- vstupní abeceda,
- zásobníková abeceda,
- přechodová funkce,
- počáteční stav,
- počáteční symbol (v zásobníku),
- množina koncových stavů,
- konfigurace,
- krok výpočtu,
- výpočet,
- jazyk přijímaný ZA koncovým stavem,
- jazyk přijímaný ZA prázdným zásobníkem,
- levé odvození v bezkontextové gramatice.

Kontrolní otázky

1. *Může u ZA nastat situace, že při provedení kroku výpočtu nedojde k posunu čtecí hlavy nad vstupní páskou?*
2. *Může mít ZA přijímající koncovým stavem prázdnou množinu koncových stavů?*

Cvičení

1. Modifikujte řešení příkladu 1.60 alespoň dvěma různými způsoby tak, aby výsledné zásobníkové automaty přijímaly jazyk L prázdným zásobníkem.
2. Najděte ZA, který přijímá jazyk $L = \{w \in \{a, b\}^+$; první a poslední symbol řetězu w je stejný$\}$ koncovým stavem.

Úkoly k textu

1. Popište dvě různé situace, kdy ZA nemůže pokračovat ve svém výpočtu.
2. Dokončete část II. důkazu věty 1.65.

Řešení

1. Místo přechodu $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$ stačí např. definovat buď $\delta(q_1, \varepsilon, Z_0) = \{(q_2, \varepsilon)\}$ nebo $\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$.
2. Jazyk L bude přijímat koncovým stavem ZA M, který po načtení stejného symbolu, jako byl první symbol vstupního slova, přechází do koncového stavu a po načtení odlišného symbolu přechází do nekoncového stavu. Přitom 1. načtený symbol byl uložen do zásobníku a dále se obsah zásobníku nemění.
   Konkrétně $M = (\{q_0, q_1\}, \{a, b\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_1\})$, kde
   $\delta(q_0, a, Z_0) = \{(q_1, aZ_0)\}$ $\quad$ $\delta(q_0, b, Z_0) = \{(q_1, bZ_0)\}$
   $\delta(q_1, a, a) = \{(q_1, a)\}$ $\quad$ $\delta(q_1, b, b) = \{(q_1, b)\}$
   $\delta(q_1, b, a) = \{(q_0, a)\}$ $\quad$ $\delta(q_1, a, b) = \{(q_0, b)\}$
   $\delta(q_0, a, a) = \{(q_1, a)\}$ $\quad$ $\delta(q_0, b, b) = \{(q_1, b)\}$
   $\delta(q_0, b, a) = \{(q_0, a)\}$ $\quad$ $\delta(q_0, a, b) = \{(q_0, b)\}$
   Ve všech ostatních případech je $\delta(x, y, z) = \emptyset$.

2 Vyčíslitelnost

Vyčíslitelnost je oblast teoretické informatiky zabývající se zkoumáním, které problémy lze řešit pomocí algoritmů a které ne.

Průvodce studiem

V krásné (i „nekrásné“) literatuře se lze setkat s celou řadou povídek, jejichž význačnou postavou je obrýlený podivínský mladík, který veškerý čas věnuje matematice či fyzice, nejvíce však počítačům. Tento podivín má přes své mládí neuvěřitelně hluboké znalosti přírodních věd a je přesvědčen, že na svém počítači vyřeší každý problém.

Uvedená charakteristika bohužel vypovídá víc o autorovi než o literární postavě: Na jedné straně mladíkovy hluboké znalosti, na straně druhé elementární neznalost vyčíslitelnosti, která popisuje celou řadu problémů, jež na počítači vyřešit nelze.

2.1 Turingovy stroje

Studijní cíle: Po prostudování kapitoly bude studující schopen formálně popsat Turingův stroj v deterministické i nedeterministické variantě. Dále bude vědět, kdy Turingův stroj přijímá či rozhoduje jazyk a kdy nedeterministický Turingův stroj přijímá jazyk.

Klíčová slova: Turingův stroj.

Potřebný čas: 60 minut.

Pro vlastní studium algoritmicky řešitelných (či neřešitelných) problémů je velmi důležité zavést vhodný výpočetní prostředek – měl by být dostatečně obecný, aby umožňoval zpracovávat libovolný algoritmus, a na druhou stranu dostatečně jednoduchý, aby zkoumání jeho vlastností nebylo příliš komplikované. Uvedené podmínky bezezbytku splňuje Turingův stroj[5] (čti Ťůringův).

Průvodce studiem

Turingův stroj nám přirozeně rozšíří a uzavře sadu automatů, jimž byly věnovány předchozí partie tohoto textu:

- Konečné automaty lze použít k výpočtům, jež buď nevyžadují paměť žádnou, anebo vyžadují omezenou paměť, kterou u KA může suplovat konečně mnoho stavů jeho řídicí jednotky.
- Zásobníkové automaty vyhovují pro výpočty, jež sice vyžadují neomezenou paměť, avšak dostačuje paměť typu zásobník.
- Turingův stroj bude disponovat (podobně jako ZA) neomezenou pamětí, bude však mít možnost vracet se k dříve uloženým datům, aniž by přitom přišel o data uložená později.

3 role Turingova stroje

Turingův stroj nám splní tři různé role:

1) Turingův stroj bude (podobně jako KA a ZA) přijímat jazyk.
2) Turingův stroj se svou sadou instrukcí (představovanou přechodovou funkcí) bude reprezentovat algoritmus, a to v případě, že jeho činnost pro každé vstupní slovo skončí po konečně mnoha krocích (tj. „nezacyklí se“).

[5] Turingův stroj není jediným obecným výpočetním prostředkem navrženým v 1. polovině 20. století. Je však dokázána vzájemná převoditelnost Turingova stroje a ostatních formalismů. Protože Turingův stroj zaujal v knihách věnovaných vyčíslitelnosti dominantní postavení, nebudeme tuto „tradici“ narušovat.

3) Turingův stroj bude mít (se šikovně nadefinovanou přechodovou funkcí) schopnost pracovat jako obecný výpočetní prostředek, tj. v případě, že mu bude dodán na vstup libovolný algoritmus včetně vstupních dat, dokáže zpracovat vstupní data přesně stejným způsobem jako příslušný algoritmus.

Průvodce studiem

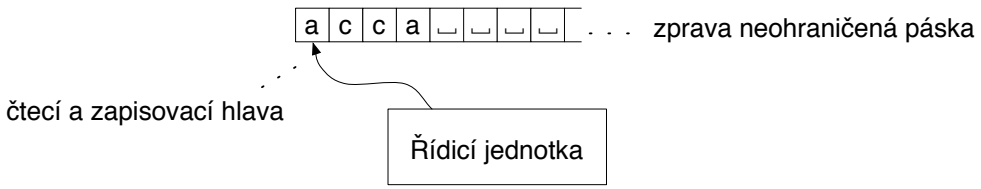
Jestliže vám některá z uvedených rolí není v tuto chvíli příliš jasná, nelamte si s tím hlavu a vraťte se k této pasáži po prostudování celé kapitoly 2.

Základní odlišností Turingova stroje od KA a ZA je využívání vstupní pásky coby paměťového média. Uvedená páska bude nekonečně dlouhá a kromě vstupního slova zapsaného před zahájením činnosti stroje na konečně mnoha políčcích budou všechna ostatní políčka pásky obsahovat tzv. prázdný znak. Čtecí a zapisovací hlava bude moci čtené symboly přepisovat jinými symboly a poté se posunout nad sousední políčko buď vlevo nebo vpravo – podle příkazu přechodové funkce. To znamená, že Turingův stroj během výpočtu vůbec nemusí dojít na konec vstupního slova. Proto (na rozdíl od KA a ZA) nebudeme pro přijetí vstupního slova Turingovým strojem požadovat „přečtení“ celého slova a omezíme se na požadavek přechodu stroje (v libovolném okamžiku) do koncového stavu označovaného jako přijímající stav.

Přijetí slova bez jeho „přečtení“

Pro větší názornost můžeme Turingův stroj chápat jako zařízení z obr. 6 skládající se

- z řídicí jednotky, která se může nacházet v některém z konečně mnoha stavů,
- ze vstupní pásky zleva ohraničené a zprava neohraničené (tj. sestávající z nekonečně mnoha políček),
- ze čtecího a zapisovacího zařízení umožňujícího řídicí jednotce číst i zapisovat symbol na políčko vstupní pásky, nad nímž se nachází čtecí a zapisovací hlava.



Obrázek 6: Ilustrace Turingova stroje

Jak bude probíhat činnost výše uvedeného zařízení?

Neformální popis činnosti TS

Výchozí předpoklady:

1) Na vstupní pásce bude zapsáno slovo tvořené ze symbolů vstupní abecedy Turingova stroje. Páska (dělená na jednotlivá políčka) bude mít na nejlevějším políčku 1. symbol vstupního slova, na následujícím políčku 2. symbol vstupního slova, atd. Všechna políčka následující políčko s posledním symbolem vstupního slova budou obsahovat speciální (tzv. prázdný) znak ␣ – půjde o symbol nevyskytující se ve vstupní abecedě. Všechny symboly, jež se mohou na pásce objevit, budou tvořit tzv. páskovou abecedu. Z dosavadního popisu tedy plyne, že pásková abeceda musí obsahovat přinejmenším prázdný znak a všechny symboly vstupní abecedy.
2) Čtecí a zapisovací hlava je umístěna nad prvním (tj. nejlevějším) políčkem pásky (a je tedy nachystána ke čtení 1. symbolu).
3) Turingův stroj se nachází ve vyznačeném, takzvaném počátečním stavu.

Průběh výpočtu: Výpočet probíhá v jednotlivých krocích, přičemž v každém kroku Turingův stroj

a) přečte symbol pod čtecí a zapisovací hlavou,

b) na základě přečteného vstupního symbolu a aktuálního stavu
   - přejde do stavu nového, nebo setrvá v aktuálním stavu,
   - přepíše přečtený symbol novým nebo tímtéž symbolem páskové abecedy,
   - posune čtecí a zapisovací hlavu o 1 políčko doleva nebo doprava.

   Informace o tom, jak se má Turingův stroj konkrétně zachovat, budou součástí jeho popisu. Konkrétně půjde o přechodovou funkci, která bude každé dvojici (stav, symbol páskové abecedy) přiřazovat trojici tvaru (stav, symbol páskové abecedy, posun čtecí a zapisovací hlavy).

Vyhodnocení výpočtu: Pokud se kdykoli v průběhu výpočtu ocitne Turingův stroj
- ve vyznačeném přijímajícím stavu, ukončí svou činnost a výsledek chápeme jako přijetí vstupního slova,
- ve vyznačeném zamítajícím stavu, ukončí svou činnost a výsledek chápeme jako zamítnutí vstupního slova.

Zbývající možností je, že se Turingův stroj nikdy nedostane ani do přijímajícího ani do zamítajícího stavu a jeho výpočet trvá donekonečna – tuto situaci budeme interpretovat jako cyklení.

Čtecí hlava místo čtecí a přepisovací hlavy

Před vlastním přechodem k formální definici Turingova stroje se domluvme na tom, že nadále místo přesného termínu čtecí a přepisovací hlava budeme uvádět zkráceně: čtecí hlava.

Definice 2.1. *Turingovým strojem* (zkráceně jej ozn. TS) nazýváme uspořádanou sedmici $(Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$, kde

Q je konečná neprázdná množina *stavů*,
Σ je *vstupní abeceda*,
Γ je *pásková abeceda*, $\Sigma \subset \Gamma$, $\Gamma - \Sigma$ obsahuje speciální (tzv. prázdný) symbol $\textvisiblespace$,
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, P\}$ je *přechodová funkce*,
$q_0 \in Q$ je *počáteční stav*,
$q_+ \in Q$ je *přijímající stav*,
$q_- \in Q$ je *zamítající stav*, přičemž $q_+ \neq q_-$.

Průvodce studiem

Právě uvedená definice Turingova stroje není jediná, s níž se můžete v knihách nebo na internetu setkat – odlišně může být definována přechodová funkce, „množina koncových stavů“, neomezenost pásky, apod. S jistou dávkou nadsázky by se mohlo říci: co autor, to jiná definice TS. Rozhodně to však neznamená, že by odlišná definice TS vedla k odlišným závěrům. Dá se totiž dokázat vzájemná převoditelnost těchto různých variant TS.

„Základních“ definic TS existuje celá řada.

Každopádně si však při čerpání informací z jakéhokoliv zdroje nejdříve pečlivě prostudujte nejen základní definici použitého TS, ale také způsob jeho činnosti.

Pro popis výpočtu TS budeme potřebovat následující pojem konfigurace.

Definice 2.2. Mějme libovolný TS $T = (Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$. Každou uspořádanou trojici $(q, \alpha, i) \in Q \times \Gamma^* \times \mathbb{N}$ nazveme *konfigurací Turingova stroje* T.

Konfigurace (q, β, i) a (q, β', i) takové, že $\beta' = \beta\textvisiblespace$, budeme považovat za stejné.

Průvodce studiem

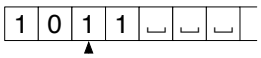
Podobně, jako tomu bylo u KA a ZA, i v případě TS konfigurace jednoznačně popisuje situaci, v níž se stroj nachází v průběhu výpočtu. Tuto situaci zřejmě určuje aktuální stav,

konečný úsek pásky (na němž bylo zapsáno vstupní slovo, případně kde dosud probíhal výpočet) následovaný pouze políčky obsahujícími prázdné znaky a umístění čtecí hlavy. Tj. konfiguraci $(q, \alpha, 8)$ odpovídá situace, kdy se daný TS nachází ve stavu q, na pásce je zapsán řetěz α následovaný samými prázdnými znaky a čtecí hlava je umístěna nad 8. políčkem pásky (počítáno zleva).

To, že konfigurací zamýšlíme jednoznačně popsat situaci, v níž se TS během svého výpočtu nachází, vysvětluje závěrečný dovětek v definici 2.2. Konfigurace (q, β, i) a $(q, \beta\text{␣}, i)$ se sice zdánlivě liší obsahem pásky, ale při důkladném prověření je vidět, že „v obou případech“ je na pásce řetěz β následován nekonečně mnoha prázdnými znaky.

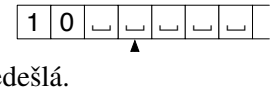
Příklad 2.3. Kvůli jednoznačnému pochopení předchozí symboliky si pojem konfigurace TS demonstrujme na následujících případech.

$(q_1, 1011, 3)$ je konfigurace, kdy se TS nachází ve stavu q_1 a obsah pásky včetně umístění čtecí hlavy lze znázornit obrázkem:



$(q_2, \text{␣␣␣}, 1)$ je konfigurace, kdy se TS nachází ve stavu q_2 a obsah pásky včetně umístění čtecí hlavy lze znázornit obrázkem:

$(q_3, 10\text{␣␣␣␣}, 4)$ je konfigurace, kdy se TS nachází ve stavu q_3 a obsah pásky včetně umístění čtecí hlavy lze znázornit obrázkem:



$(q_3, 10\text{␣␣}, 4)$ je konfigurace stejná jako ta předešlá.

Definice 2.4. *Krok výpočtu Turingova stroje T* definujeme jako binární relaci $\vdash$ (na množině všech konfigurací) takto:

Nechť $(q, a_1 \cdots a_n, i)$ je taková konfigurace T, kde $q \neq q_\pm$, $n \in \mathbb{N}$, $a_1, \ldots, a_n \in \Gamma$ a $i \in \{1, \ldots, n\}$.

a) Je-li $2 \leq i \leq n$ a $\delta(q, a_i) = (q', b, L)$, pak
$(q, a_1 \cdots a_n, i) \vdash (q', a_1 \cdots a_{i-1} b a_{i+1} \cdots a_n, i-1)$.

b) Je-li $\delta(q, a_1) = (q', b, L)$, pak $(q, a_1 \cdots a_n, 1) \vdash (q', b a_2 \cdots a_n, 1)$.

Má-li čtecí hlava „opustit“ pásku, setrvá nad 1. políčkem pásky.

c) Je-li $\delta(q, a_i) = (q', b, P)$, pak $(q, a_1 \cdots a_n, i) \vdash (q', a_1 \cdots a_{i-1} b a_{i+1} \cdots a_n, i+1)$.

Průvodce studiem

Prostudujete-li si definici kroku výpočtu podrobněji, zjistíte, že v případu a) výchozí konfigurace odpovídá situaci, kdy se TS nachází ve stavu q, čtecí hlava je nad i-tým políčkem pásky a čte páskový symbol a_i. Přechodová funkce pro takovou situaci „přikazuje“ přechod do stavu q', přepis a_i symbolem b a posun čtecí hlavy o 1 políčko doleva. A skutečně, konfigurace po provedeném kroku výpočtu odpovídá situaci, kdy se TS nachází ve stavu q', na pásce je místo a_i symbol b a čtecí hlava je nad $(i-1)$. políčkem pásky.

V případu b) je ve výchozí konfiguraci čtecí hlava nad 1. políčkem pásky a nemá tedy reálnou možnost posunu doleva. Definice „řeší“ tuto zapeklitou situaci „příkazem“, aby čtecí hlava jednoduše setrvala nad 1. políčkem pásky a aby ostatní změny byly provedeny v souladu s přechodovou funkcí.

V případu c) žádné problémy nenastávají, neboť u pravostranně neomezené pásky Turingova stroje má čtecí hlava kdykoliv možnost posunout se o 1 políčko doprava.

Pokud byste přemýšleli nad korektností zápisu řetězu $a_j \cdots a_k$, kde $j > k$, pak učiňme následující dohodu. Posloupnost $(a_i)_{i=j}^{k}$ (viz definice řetězu coby posloupnosti symbolů), kde $j > k$, budeme v souladu s konvencí interpretovat jako prázdnou posloupnost neboli prázdný řetěz.

Nakonec si všimněte předpokladu $q \neq q_\pm$ podmiňujícího provedení kroku výpočtu. Nejde o nic jiného, než že TS může provést krok výpočtu jedině tehdy, když se nenachází

ani v přijímajícím, ani v zamítajícím stavu. Jinak řečeno, přechod do přijímajícího nebo zamítajícího stavu znamená ukončení výpočtu Turingova stroje.

Definice 2.5. *Výpočet Turingova stroje* definujeme prostřednictvím relace $\vdash^*$ (tj. reflexivního a tranzitivního uzávěru relace $\vdash$). Každý výpočet pro vstupní slovo w bude začínat konfigurací $(q_0, w, 1)$ zvanou *počáteční*.

Definice 2.6. O každém TS $T = (Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$ řekneme, že

- *přijímá* vstupní slovo $w \in \Sigma^*$, právě když $(q_0, w, 1) \vdash^* (q_+, w', i)$, kde $w' \in \Gamma^*$ a $i \in \mathbb{N}$,
- *zamítá* vstupní slovo $w \in \Sigma^*$, právě když $(q_0, w, 1) \vdash^* (q_-, w', i)$, kde $w' \in \Gamma^*$ a $i \in \mathbb{N}$,
- pro vstupní slovo $w \in \Sigma^*$ *cyklí*, právě když w ani nepřijímá, ani nezamítá.

TS může cyklit.

Průvodce studiem

Podle uvedené definice je vstupní slovo přijato, pokud se při výpočtu nad tímto slovem TS ocitne v přijímajícím stavu – u výsledné konfigurace přitom vůbec nezáleží na aktuálním obsahu pásky a na umístění čtecí hlavy (tj. na rozdíl od KA a ZA nemusí být vstupní slovo „přečteno“). V případě rovnosti mezi počátečním a přijímajícím stavem (ověřte si v definici TS, že je to možné) je vstupní slovo dokonce přijato, aniž TS vykonal byť jen jediný krok výpočtu!

Zamítnutí vstupního slova je definováno podobně – jediným požadavkem je přechod TS po konečně mnoha krocích výpočtu do zamítajícího stavu.

Cyklení pro vstupní slovo zřejmě nastane tehdy, když TS neukončí po konečně mnoha výpočetních krocích svou činnost přechodem do přijímajícího nebo zamítajícího stavu.

Definice 2.7. *Jazyk L přijímaný Turingovým strojem* $T = (Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$ značíme $L(T)$ a definujeme následovně:

$$L(T) = \{w \in \Sigma^*;\ \text{TS } T \text{ přijímá } w\}.$$

TS přijímá (případně i rozhoduje) jazyk.

V případě, že TS T pro žádné vstupní slovo $w \in \Sigma^*$ necyklí, říkáme, že T *rozhoduje jazyk* $L(T)$.

Průvodce studiem

Uvedenou definici lze přeformulovat následovně:

Jazyk L je přijímán Turingovým strojem T, právě když TS T přijímá každé slovo z L a každé slovo neležící v L buď zamítá, anebo pro ně cyklí.

Jazyk L je rozhodován Turingovým strojem T, právě když TS T přijímá každé slovo z L a zamítá každé slovo neležící v L. (Tj. T pro žádné vstupní slovo necyklí.)

Všimněte si okamžitého důsledku definice: Pokud je jazyk L rozhodován nějakým Turingovým strojem, pak je tímto strojem také přijímán.

Příklad 2.8. Najdeme TS rozhodující jazyk $L = \{0^{2^n};\ n \geq 0\}$. (Tj. L zahrnuje výhradně řetězy délek 2^n sestávající ze samých nul.)[6]

Nejdříve si uveďme stručný slovní popis toho, jak by mohl hledaný TS pracovat.
TS T pro vstupní slovo $w \in \{0\}^*$:

1) Zamítá vstupní slovo w, pokud $w = \varepsilon$; v opačném případě přejde k bodu 2).
2) Projde zleva doprava pásku a „přeškrtne“ každý druhý symbol 0.
3) Pokud v části 2) obsahovala páska jediný symbol 0, přijme w. Jinak pokračuje bodem 4).

[6] Jazyk L není bezkontextový, neexistuje tedy ZA, který by jej přijímal. Pro důkaz tohoto tvrzení však nemáme o bezkontextových jazycích v našem textu dostatek poznatků.

4) Pokud v části 2) obsahovala páska více než jeden symbol 0 a celkový počet těchto symbolů byl $\begin{cases} \text{lichý, zamítne } w, \\ \text{sudý, vrátí čtecí hlavu na levý okraj pásky a přejde k bodu 2).} \end{cases}$

Průvodce studiem

Je zřejmé, že TS T při každém průchodu pásky v bodě 2) zmenší počet zapsaných nul na polovinu, tzn. obsahovala-li původně páska 2^k nul, po „škrtání“ jich bude obsahovat pouze 2^{k-1}. Opakuje-li se uvedená akce dostatečně mnohokrát, pak z každého vstupního slova tvaru 0^{2^n} zůstane na pásce jediná neškrtnutá nula a v bodě 3) je vstupní slovo přijato.

Jestliže vstupní slovo nebylo tvaru 0^{2^n}, musí časem dojít k tomu, že se na pásce objeví lichý počet nul, a slovo je pak v bodě 4) zamítnuto.

Jak máte rozumět termínu „škrtání“? Celkem jednoduše, TS má přece možnost přepsat čtený symbol jiným symbolem. Může tedy například přepsat symbol 0 páskovým symbolem X.

Před formálním popisem TS T si povšimněte následujícího technického zádrhelu: Bude-li se čtecí hlava vracet nad nejlevější políčko pásky, bez jeho označení symbolem odlišným od 0 a od X nemá TS možnost zjistit, že je čtecí hlava už nad tímto políčkem. Pokud by totiž na uvedeném políčku zůstala 0 (přeškrtnuta je vždy sudá 0), TS by se při návratu čtecí hlavy zřejmě zacyklil – již víte, že „příkaz“ posunout čtecí hlavu mimo levý okraj pásky vede k výpočetnímu kroku, při němž setrvá čtecí hlava nad nejlevějším políčkem; tento krok výpočtu by byl proveden znovu a znovu ... Snad tedy nebudete překvapeni tím, že 1. nula bude „označena“ přepsáním na symbol $\sqcup$ – prázdný znak v páskové abecedě již je, a nemusíme tedy používat další páskové symboly. Prázdný znak nám potom na začátku pásky bude signalizovat: „Zde je 1. nula vstupního slova!“

Formální popis výše uvedeného TS:

$T = (Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$, kde

$Q = \{q_0, q_1, q_2, q_3, q_4, q_+, q_-\}$,
$\Sigma = \{0\}$,
$\Gamma = \{0, X, \sqcup\}$
a přechodová funkce δ je pro větší názornost popsána na obrázku 7 grafem (analogickým stavovému diagramu KA), v němž označení hrany $(0, \sqcup, P)$ mezi uzly q_0, q_1 znamená, že $\delta(q_0, 0) = (q_1, \sqcup, P)$. Podobně u ostatních hran.

Průvodce studiem

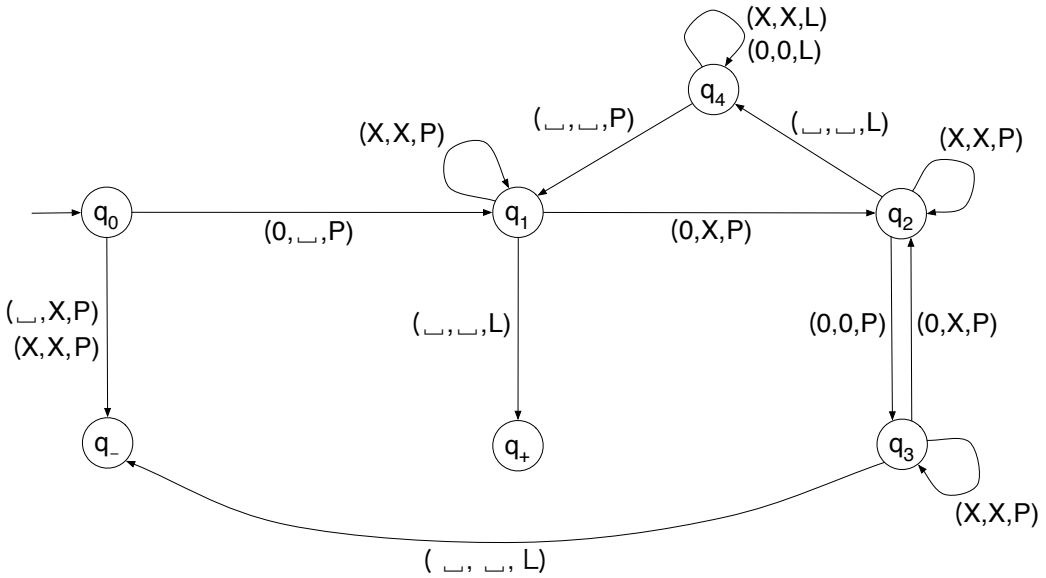
Neznepokojujte se úvahami na téma: Ve stavu q_0 přece TS T nemůže nikdy číst symbol X. Proč je tedy na uvedeném obrázku popsán přechod z takové situace do zamítajícího stavu?

Odpověď je prostá: Protože přechodovou funkci Turingova stroje máme definovánu pro každý stav a každý páskový symbol. Je tedy přirozené definovat pro takovou situaci přechod (například) do zamítajícího stavu.

Pro lepší pochopení činnosti TS T si uvedeme příklady výpočtů jednak pro slovo nepatřící do L jednak pro slovo z jazyka L. (Pro větší názornost u každé konfigurace podtrhneme symbol, nad nímž je čtecí hlava.)

1) $(q_0, \underline{0}00, 1) \vdash (q_1, \sqcup\underline{0}0, 2) \vdash (q_2, \sqcup X\underline{0}, 3) \vdash (q_3, \sqcup X0\underline{\sqcup}, 4) \vdash (q_-, \sqcup X\underline{0}, 3)$,

2) $(q_0, \underline{0}000, 1) \vdash (q_1, \sqcup\underline{0}00, 2) \vdash (q_2, \sqcup X\underline{0}0, 3) \vdash (q_3, \sqcup X0\underline{0}, 4) \vdash (q_2, \sqcup X0X\underline{\sqcup}, 5) \vdash$
$(q_4, \sqcup X0\underline{X}, 4) \vdash^* (q_4, \underline{\sqcup}X0X, 1) \vdash (q_1, \sqcup\underline{X}0X, 2) \vdash (q_1, \sqcup X\underline{0}X, 3) \vdash$
$(q_2, \sqcup XX\underline{X}, 4) \vdash (q_2, \sqcup XXX\underline{\sqcup}, 5) \vdash (q_4, \sqcup XX\underline{X}, 4) \vdash^* (q_4, \underline{\sqcup}XXX, 1) \vdash$
$(q_1, \sqcup\underline{X}XX, 2) \vdash^* (q_1, \sqcup XXX\underline{\sqcup}, 5) \vdash (q_+, \sqcup XX\underline{X}, 4)$.



Obrázek 7: Schéma TS k příkladu 2.8

2.1.1 Nedeterministické Turingovy stroje

Podobně, jako jsme definovali NKA nebo (nedeterministický) zásobníkový automat, zavedeme také nedeterministický TS. Půjde o takovou modifikaci TS, jehož přechodová funkce bude přiřazovat každé dvojici (stav, páskový symbol) množinu trojic tvaru (stav, páskový symbol, požadovaný posun čtecí hlavy).

Definice 2.9. *Nedeterministickým Turingovým strojem* (zkráceně ozn. NTS) nazýváme uspořádanou sedmici $(Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$, kde

Q je konečná neprázdná množina *stavů*,
Σ je *vstupní abeceda*,
Γ je *pásková abeceda*, $\Sigma \subset \Gamma$, $\Gamma - \Sigma$ obsahuje speciální (tzv. prázdný) symbol ␣,
$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,P\}}$ je *přechodová funkce*,
$q_0 \in Q$ je *počáteční stav*,
$q_+ \in Q$ je *přijímající stav*,
$q_- \in Q$ je *zamítající stav*, přičemž $q_+ \neq q_-$.

Průvodce studiem

Pojmy konfigurace, krok výpočtu a výpočet nedeterministického Turingova stroje zavádíme analogicky pojmům, které již znáte.

Definice 2.10. Mějme libovolný NTS $T = (Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$. Každou uspořádanou trojici $(q, \alpha, i) \in Q \times \Gamma^* \times \mathbb{N}$ nazveme *konfigurací nedeterministického Turingova stroje* T.

Krok výpočtu T definujeme za pomoci přechodové funkce δ jako binární relaci $\vdash$ (na množině všech konfigurací) analogicky kroku výpočtu TS.

Symbolem $\vdash^*$ budeme označovat reflexivní a tranzitivní uzávěr relace $\vdash$, symbolem $\vdash^+$ tranzitivní uzávěr relace $\vdash$.

Výpočet nedeterministického Turingova stroje definujeme prostřednictvím relace $\vdash^*$.

Definice 2.11. O každém NTS $T = (Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$ řekneme, že *přijímá* vstupní slovo $w \in \Sigma^*$, právě když $(q_0, w, 1) \vdash^* (q_+, w', i)$, kde $w' \in \Gamma^*$ a $i \in \mathbb{N}$.

Průvodce studiem

Podle uvedené definice je vstupní slovo w přijato, pokud pro w existuje „větev výpočtu“ NTS T končící konfigurací obsahující přijímající stav. Nezáleží tedy na tom, jestli pro NTS T a vstupní slovo w

- ostatní „větve výpočtu“ končí konfiguracemi obsahujícími zamítající stav,
- existuje „nikdy nekončící větev výpočtu“ (signalizující možnost zacyklení).

Co se týče zamítání vstupního slova, nebudeme je definovat! Ne že by definovat nešlo, ale v následujících partiích je vůbec nebudeme potřebovat.

Definice 2.12. *Jazyk L přijímaný nedeterministickým Turingovým strojem* $T = (Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$ značíme $L(T)$ a definujeme následovně:

$$L(T) = \{w \in \Sigma^*;\ \text{NTS } T \text{ přijímá } w\}.$$

Průvodce studiem

Uvedenou definici lze přeformulovat následovně:

Jazyk L je přijímán nedeterministickým Turingovým strojem T, právě když pro každé slovo z L existuje „přijímající větev výpočtu“ stroje T a pro žádné slovo neležící v L taková výpočetní větev T neexistuje.

Stejná výpočetní síla deterministických a nedeterministických Turingových strojů.

Poznámka 2.13. Poznamenejme, že třídy deterministických a nedeterministických Turingových strojů mají stejnou výpočetní sílu. Uvedené tvrzení bude přímým důsledkem věty 3.16.

Průvodce studiem

Možná si kladete otázku: Proč tedy zavádíme pojem NTS, když tím nezvýšíme výpočetní sílu deterministických Turingových strojů? Odpověď tkví v definici tříd složitostí popisovaných v kapitole 3 – jedna (velmi užitečná) třída složitosti je definována právě s pomocí NTS.

Shrnutí

TS je zařízení vybavené konečněstavovou řídicí jednotkou, páskou nekonečné délky a čtecí a zapisovací hlavou. Páska je ohraničená zleva a neohraničená zprava. Každý TS je jednoznačně určen

- množinou stavů (v nichž se může nacházet řídicí jednotka),
- vstupní abecedou Σ (vstupní slova TS jsou tvořena řetězy ze Σ^*),
- páskovou abecedou Γ (na každém políčku pásky může být zapsán výhradně symbol z Γ; pásková abeceda obsahuje všechny vstupní symboly a tzv. prázdný znak, může však zahrnovat ještě celou řadu dalších „pomocných“ symbolů),
- přechodovou funkcí δ, která každé dvojici (stav, symbol páskové abecedy) přiřazuje uspořádanou trojici typu (stav, páskový symbol, označení posunu čtecí a zapisovací hlavy),
- třemi vyznačenými stavy:
  a) počátečním stavem (v něm bude řídicí jednotka před zahájením práce daného TS, ať je na vstupu jakékoli vstupní slovo),
  b) přijímajícím stavem (ocitne-li se v něm řídicí jednotka kdykoli v průběhu činnosti, TS okamžitě zastaví svůj výpočet a přijímá vstupní slovo),
  c) zamítajícím stavem (ocitne-li se v něm řídicí jednotka kdykoli v průběhu činnosti, TS okamžitě zastaví svůj výpočet a zamítá vstupní slovo).

Výpočet každého TS je tvořen posloupností kroků výpočtu – v každém z nich TS přejde z aktuálního stavu do stavu nového, přepíše symbol na pásce pod čtecí a zapisovací hlavou novým symbolem a posune hlavu o jedno políčko doprava nebo doleva. (Pokud je hlava nad nejlevějším políčkem pásky, místo posunu doleva setrvá nad aktuálním políčkem.) Informace o tom, do jakého stavu má stroj přejít, jakým symbolem má být přepsán symbol pod čtecí a zapisovací hlavou a kam se má hlava posunout, jsou obsaženy ve výsledku přechodové funkce aplikované na dvojici (aktuální stav, čtený symbol).

V průběhu výpočtu se mění tzv. konfigurace TS, tj. situace, v níž se TS nachází a která je charakterizována aktuálním stavem, neprázdným obsahem pásky a umístěním čtecí a zapisovací hlavy. Každý výpočet začíná v počáteční konfiguraci, která je určena

- počátečním stavem,
- vstupním slovem na pásce (na každém políčku pásky je jeden symbol, pro zápis vstupního slova na pásku jsou využita nejlevější políčka pásky, na každém dalším políčku je pak zapsán prázdný symbol – napravo od vstupního slova je jich tedy nekonečně mnoho),
- umístěním čtecí a zapisovací hlavy nad 1. políčkem pásky.

Výpočet končí v okamžiku, kdy se TS ocitne

- v přijímajícím stavu – pak říkáme, že TS přijal své vstupní slovo,
- v zamítajícím stavu – pak říkáme, že TS zamítl své vstupní slovo.

Pokud se TS nedostane ani do přijímajícího, ani do zamítajícího stavu, říkáme, že cyklí. Množina všech slov, jež jsou přijímána, tvoří jazyk přijímaný Turingovým strojem. Jestliže TS pro žádné vstupní slovo necyklí, místo termínu jazyk přijímaný Turingovým strojem používáme termín jazyk rozhodovaný Turingovým strojem.

NTS se od TS odlišuje zejména tvarem přechodové funkce (k ní se pojí známým způsobem pojmy krok výpočtu a výpočet): Přechodová funkce NTS zobrazuje každou dvojici stav a páskový symbol na množinu uspořádaných trojic tvaru (stav, páskový symbol, označení posunu čtecí a zapisovací hlavy). V každém kroku výpočtu může NTS z aktuálního stavu p na základě znalosti čteného páskového symbolu a přejít do nového stavu q, přepsat čtený symbol páskovým symbolem b a posunout čtecí a zapisovací hlavu o 1 políčko doleva (resp. doprava) – to vše za podmínky, že uspořádaná trojice (q, b, L) (resp. (q, b, P)) je obsažena ve výsledku přechodové funkce δ pro argumenty p, a. V případě, že by se čtecí a zapisovací hlava při provádění kroku výpočtu měla posunout nalevo od nejlevějšího políčka pásky, zůstane hlava nad výchozím políčkem.

Výpočet NTS se (podobně jako u NKA či ZA) může větvit do více variant. Pokud pro zadané vstupní slovo existuje alespoň jedna varianta, kdy skončí NTS svůj výpočet v přijímajícím stavu, říkáme, že NTS přijímá dané slovo. Množina všech slov nad vstupní abecedou, jež jsou přijímána, tvoří jazyk přijímaný nedeterministickým Turingovým strojem.

Třída všech nedeterministických Turingových strojů má stejnou výpočetní sílu jako třída všech Turingových strojů.

Pojmy k zapamatování

- (Deterministický) Turingův stroj (TS),
- nedeterministický Turingův stroj (NTS),
- stav,
- vstupní abeceda,
- pásková abeceda,
- přechodová funkce,
- počáteční stav,
- přijímající stav,
- zamítající stav,
- konfigurace,
- krok výpočtu,

- výpočet,
- přijetí vstupního slova u TS i NTS,
- zamítnutí vstupního slova u TS,
- cyklení pro vstupní slovo u TS,
- jazyk přijímaný Turingovým strojem deterministickým i nedeterministickým,
- jazyk rozhodovaný Turingovým strojem.

Kontrolní otázky

1. *Může TS zamítnout vstupní slovo, aniž by vykonal aspoň jeden krok výpočtu?*
2. *Může se TS dostat do situace, kdy nedochází k posunu jeho čtecí a zapisovací hlavy a přitom cyklí?*
3. *Platí, že jazyk přijímaný Turingovým strojem obsahuje všechna slova nad vstupní abecedou, která nejsou tímto strojem zamítána?*
4. *Zavedli jsme pojem jazyka rozhodovaného nedeterministickým Turingovým strojem?*

Cvičení

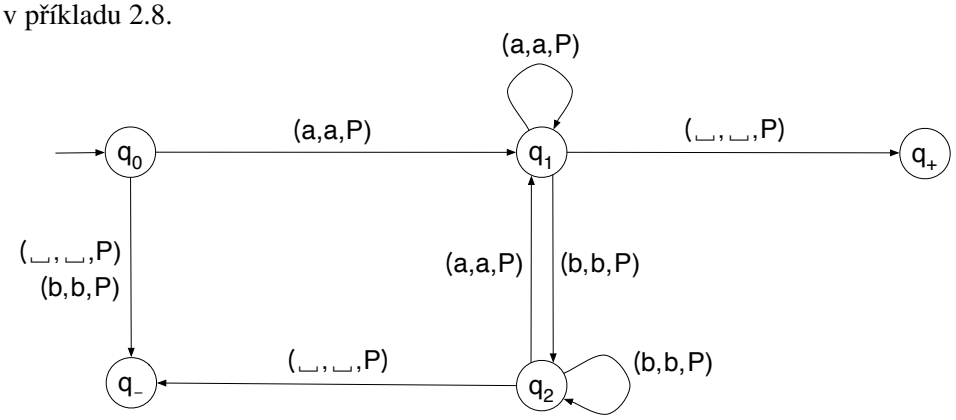
1. Najděte TS, který pro každé vstupní slovo cyklí.
2. Najděte TS, který rozhoduje jazyk $L = \{w \in \{a,b\}^+$; prvním i posledním symbolem řetězu w je $a\}$.

Úkoly k textu

1. Upravte páskovou abecedu a přechodovou funkci TS z příkladu 2.8 pro případ vstupní abecedy $\Sigma = \{0,1\}$.

Řešení

1. Turingových strojů, které cyklí pro všechna vstupní slova je nekonečně mnoho. Uvedeme dva jednoduché příklady:
   a) TS $T_1 = (\{q_0, q_+, q_-\}, \{a\}, \{a, \text{␣}\}, \delta, q_0, q_+, q_-)$,
   kde $\delta(q_0, a) = \delta(q_0, \text{␣}) = (q_0, \text{␣}, P)$.
   b) TS $T_2 = (\{q_0, q_+, q_-\}, \{a\}, \{a, \text{␣}\}, \delta, q_0, q_+, q_-)$,
   kde $\delta(q_0, a) = \delta(q_0, \text{␣}) = (q_0, a, L)$.
2. Zadaný jazyk je rozhodován například Turingovým strojem
   $T = (\{q_0, q_1, q_2, q_+, q_-\}, \{a, b\}, \{a, b, \text{␣}\}, \delta, q_0, q_+, q_-)$,
   kde přechodová funkce δ je schematizována níže, a to stejným způsobem, jaký byl použit v příkladu 2.8.



2.2 Jazyky a problémy

Studijní cíle: Po prostudování kapitoly bude studující znát, jaké jazyky jsou přiřazovány Turingovým strojům, jak lze exaktně definovat algoritmus a jak jsou definovány řešitelné a částečně řešitelné problémy. Dále bude schopen formulovat problém zastavení Turingova stroje a dokázat jeho neřešitelnost.

Klíčová slova: Rekurzivní jazyk, částečně rekurzivní jazyk, řešitelný problém, částečně řešitelný problém, problém zastavení Turingova stroje.

Potřebný čas: 150 minut.

Nejdříve si uvedeme terminologii používanou pro jazyky přiřazované k Turingovým strojům.

Definice 2.14. Nechť L je jazyk nad abecedou Σ.

a) Řekneme, že L je *rekurzivní jazyk*[7], právě když existuje TS, který rozhoduje L.
b) Řekneme, že L je *částečně rekurzivní jazyk*[8], právě když existuje TS, který přijímá L.

Průvodce studiem

Připomeňme si, co plyne z definice jazyka rozhodovaného (resp. přijímaného) Turingovým strojem:

Pro rekurzivní jazyk L tak musí existovat TS T, který přijímá každé slovo z L a zamítá každé slovo nepatřící do L; tzn. T nikdy necyklí.

Pro částečně rekurzivní jazyk L pak musí existovat TS T, který přijímá každé slovo z L a každé slovo nepatřící do L buď zamítá anebo pro něj cyklí.

Z definice 2.14 okamžitě plyne následující důsledek.

Důsledek 2.15. *Každý rekurzivní jazyk je také částečně rekurzivní.*

Poznámka 2.16. Lze dokázat, že třída částečně rekurzivních jazyků je rovna třídě jazyků typu 0. Příslušný důkaz[9] je však dlouhý, a proto jej vynecháme.

Jazyky typu 0 jsou přijímány Turingovými stroji.

2.2.1 Příklad jednoduchého rekurzivního jazyka

Příklad 2.17. Prokážeme, že jazyk $L = \{zz;\ z \in \{a, e\}^+\}$ je rekurzivní.

Podle definice rekurzivního jazyka musíme najít TS, který rozhoduje L. Příkladem takového Turingova stroje je TS T, jehož činnost lze rozdělit na dvě části:

I. T nalezne „střed“ vstupního slova, a to následujícím způsobem. Opakovaně označuje 1. dosud neoznačený vstupní symbol tečkou a poslední dosud neoznačený vstupní symbol dvěma tečkami. V okamžiku, kdy jsou všechny symboly vstupního slova označeny, je rozdělení slova na poloviny dokončeno. Pokud naposled označený symbol byl označen 1 tečkou, vstupní slovo má lichou délku a T je zamítne. V případě označení posledního symbolu 2 tečkami jde o slovo sudé délky a T přejde k porovnávání obou polovin.

Hledání „středu“ vstupního slova Turingovým strojem

[7] Terminologie, s níž se lze setkat v odborné literatuře není jednotná. Rekurzivní jazyky jsou také nazývány jazyky vyčíslitelnými, rozhodnutelnými nebo řešitelnými.

[8] Částečně rekurzivní jazyky bývají v odborné literatuře nazývány také jazyky rekurzivně spočetnými nebo rekurzivně vyčíslitelnými.

[9] Zájemce jej může nalézt např. v [Hop69] nebo v [Chy82].

Průvodce studiem

Aby bylo možné označovat symboly vstupní abecedy 1 či 2 tečkami, pásková abeceda musí pochopitelně obsahovat kromě každého vstupního symbolu x také „označené“ symboly $\dot{x}$ a $\ddot{x}$. Proces hledání „středu“ vstupního slova $aeeaee$ tak vede k následujícím obsahům pásky:

a) $aeeaee\,␣␣\ldots$
b) $\dot{a}eeaee\,␣␣\ldots$
c) $\dot{a}eeae\ddot{e}\,␣␣\ldots$
d) $\dot{a}\dot{e}eae\ddot{e}\,␣␣\ldots$
e) $\dot{a}\dot{e}ea\ddot{e}\ddot{e}\,␣␣\ldots$
f) $\dot{a}\dot{e}\dot{e}a\ddot{e}\ddot{e}\,␣␣\ldots$
g) $\dot{a}\dot{e}\dot{e}\ddot{a}\ddot{e}\ddot{e}\,␣␣\ldots$

II. T porovná 1. polovinu slova s 2. polovinou:
Postupně porovnává symboly na prvních pozicích, na druhých pozicích, atd. Pokud jsou všude na odpovídajících si pozicích tytéž symboly, TS T vstupní slovo přijme, při nesplnění uvedené podmínky vstupní slovo zamítne.

Porovnávání řetězů Turingovým strojem

Průvodce studiem

Aby TS T „věděl“, které symboly má právě porovnávat, úspěšně prověřené symboly pokaždé přeškrtne, tj. před porovnáváním symbolů na třetích pozicích bude na pásce například $X X \dot{e} X X \ddot{e}\,␣␣\ldots$

Následuje slovní popis uvedeného TS.

TS T pro vstupní slovo w:

1) Ověří, zda jde o slovo délky alespoň 2; pokud ne, zamítá w.
2) Označí nejlevější neoznačený symbol 1 tečkou. Jestliže je sousední symbol napravo označen 2 tečkami, zamítá w.
3) Označí nejpravější neprázdný neoznačený symbol 2 tečkami. Pokud je sousední symbol nalevo označen 1 tečkou, přejde k bodu 4); jinak přejde k bodu 2).
4) „Zapamatuje si“ nejlevější symbol označený 1 tečkou, „přeškrtne jej“ a přesune čtecí hlavu nad nejlevější symbol označený 2 tečkami – jde-li o jiný symbol, než je ten „zapamatovaný“, TS T zamítá w; v opačném případě symbol pod čtecí hlavou „přeškrtne“ a posune čtecí hlavu o 1 políčko doprava.
5) Je-li pod čtecí hlavou neprázdný symbol, přejde k bodu 4). Jinak TS T přijímá w.

Průvodce studiem

Jistě vás napadlo, že TS T nemusí pracovat „jednosměrně“, ale že kvůli urychlení své práce je pro něj výhodnější činnost „obousměrná“, tj. označování symbolů 1 a 2 tečkami může být prováděno i při návratu čtecí hlavy (od symbolu naposled označeného 2 tečkami) nad nejlevější neoznačený symbol. Pro vstupní slovo $aeeaee$ by pak označování probíhalo v následujícím pořadí:

a) $\dot{a}eeaee\,␣␣\ldots$
b) $\dot{a}eeae\ddot{e}\,␣␣\ldots$
c) $\dot{a}eea\ddot{e}\ddot{e}\,␣␣\ldots$
d) $\dot{a}\dot{e}ea\ddot{e}\ddot{e}\,␣␣\ldots$
e) $\dot{a}\dot{e}\dot{e}a\ddot{e}\ddot{e}\,␣␣\ldots$
f) $\dot{a}\dot{e}\dot{e}\ddot{a}\ddot{e}\ddot{e}\,␣␣\ldots$

Podobně lze urychlit i porovnávání symbolů v bodech 4) a 5).

Uvedená urychlení však vedou k trochu náročnějšímu slovnímu popisu TS T a také k mírně komplikovanější přechodové funkci stroje T. Hlavním důvodem, proč zde uvádíme TS T pracující „pouze jednosměrně“, je tedy jednodušší popis.

Formálně:

TS $T = (Q, \Sigma, \Gamma, \delta, q_0, q_+, q_-)$, kde

$Q = \{q_+, q_-, q_a, q_b, q_0, q_1, \ldots, q_{11}\}$,
$\Sigma = \{a, e\}$,
$\Gamma = \{a, e, \dot{a}, \dot{e}, \ddot{a}, \ddot{e}, \bar{a}, \bar{e}, X, \textvisiblespace\}$
a přechodová funkce δ je znázorněna na obrázku 8 tímtéž způsobem, jaký byl použit v příkladu 2.8.

(Pokud na obrázku z některého stavu nevede dál hrana pro některý páskový symbol, je tomu tak z důvodu větší přehlednosti – každá taková hrana by končila v zamítajícím stavu.)

Průvodce studiem

Všimněte si podobného obratu, jaký se vyskytl v příkladu 2.8: Místo označení symbolu na nejlevějším políčku pásky jednou tečkou je onen symbol označen pruhem. Odlišné označení pak umožní stroji T při návratu čtecí hlavy (nejenom) z konce vstupního slova rozpoznat 1. políčko pásky.

2.2.2 Konvence pro popis Turingových strojů

Příklad 2.17 ukazuje, jak narůstá počet stavů a komplikuje se formální popis i u TS zpracovávajícího celkem jednoduchou úlohu. Proto nadále formální popis u žádného TS již uvádět nebudeme a spokojíme se pouze se stručným slovním popisem. Detaily typu „jak TS označuje, porovnává či přesouvá symboly na pásce“ ponecháme laskavému čtenáři k samostatnému promýšlení – nikomu s alespoň minimální programátorskou zkušeností by to nemělo činit zvláštní potíže.

Nadále jen slovní popis TS

Protože se v tomto textu setkáme ještě s celou řadou Turingových strojů, domluvme se na používání následující symboliky. Pokud vstupní slovo Turingova stroje označíme symbolem

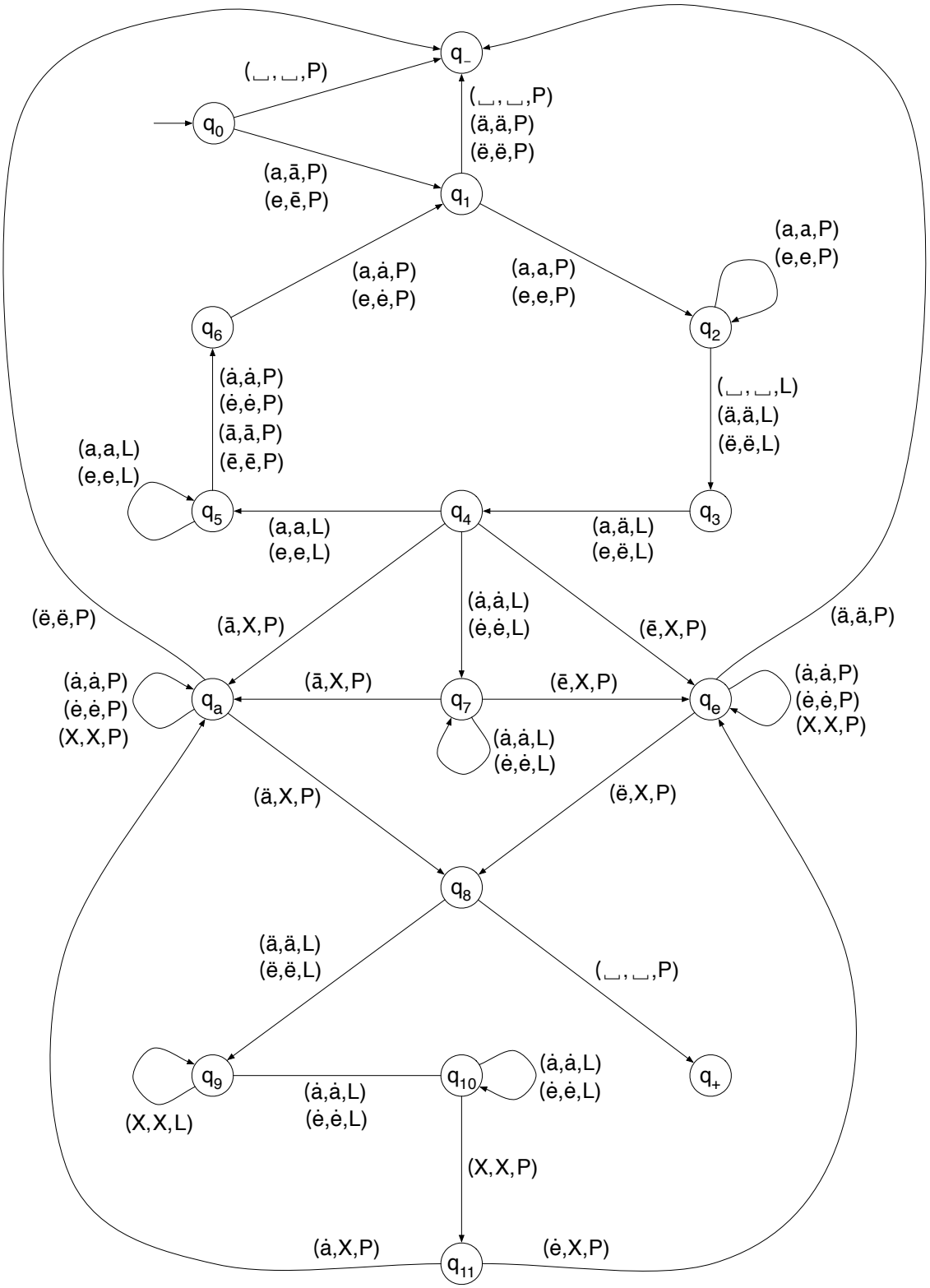
- w (x, s, apod.), budeme jej vnímat jen jako řetěz vstupních symbolů,
- $\langle A \rangle$, budeme jej chápat jako řetěz vstupních symbolů představující zakódování objektu A,
- $\langle A_1, \ldots, A_n \rangle$, budeme jej chápat jako řetěz vstupních symbolů představující zakódování objektů $A_1, \ldots, A_n$.

Dohoda pro označování vstupních slov TS

Průvodce studiem

Co si máte pod „zakódováním objektu A“ představit? Jednoduše řetěz symbolů, který dohodnutým způsobem jednoznačně popisuje objekt A.

Pokud je tedy A například maticí, může jít o řetěz obsahující jednotlivé prvky této matice, oddělené vždy nějakým speciálním znakem (např. &), přičemž nejdříve jsou uvedeny prvky 1. řádku, pak prvky 2. řádku, atd. Mezi posledním prvkem jednoho řádku a prvním prvkem řádku následujícího je přitom nějaký jiný speciální oddělovací znak (např. #).



Obrázek 8: Schéma TS k příkladu 2.17

Matice $\begin{pmatrix} a & b & a \\ c & a & a \end{pmatrix}$ by tak byla reprezentována řetězem $a \,\&\, b \,\&\, a \,\#\, c \,\&\, a \,\&\, a$.

Jestliže je objektů více, jejich jednotlivá zakódování mohou být od sebe oddělena opět nějakým speciálním symbolem. Ostatně, výše uvedenou matici můžeme vnímat jako soubor jejích řádků, přičemž každý je od ostatních oddělen symbolem $\#$.

TS nejdříve kontroluje správnost zakódování.

Dále se domluvme na zkrácení slovního popisu TS prostřednictvím sousloví „pro vstupní slovo $\langle A \rangle$“. Uvedené sousloví budeme používat v úvodu popisu a budeme jím zkracovat pokyn, aby daný TS nejdříve zkontroloval, zda $\langle A \rangle$ odpovídá správnému zakódování objektu A – pokud ne, TS zamítne $\langle A \rangle$.

Průvodce studiem

Představte si například, že popis zahájíme slovy: „TS T pro vstupní slovo $\langle A \rangle$, kde A je čtvercová matice ...“ Pak jsme stroji T již předepsali, že pro libovolné vstupní slovo má nejdříve zkontrolovat, jestli jde o korektní zakódování čtvercové matice, tj. zda je na každém řádku matice stejný počet prvků a zda tento počet odpovídá také počtu řádků. Není-li kontrola v pořádku, TS příslušné vstupní slovo ihned zamítne. V opačném případě může TS začít vykonávat další příkazy uvedené následovně (a nemusíme se tedy již starat o to, jestli je „zpracovávaná“ matice čtvercová.)

2.2.3 Churchova–Turingova teze

Dříve než si popíšeme problém, který nelze vyřešit pomocí žádného algoritmu, musíme si ujasnit, co přesně budeme chápat pod pojmem algoritmus.

Jde o pojem běžně užívaný a téměř každému srozumitelný – vždyť kdo by neznal ze základní školy algoritmy pro násobení 2 celých čísel či řešení 1 lineární rovnice o 1 neznámé nebo ze střední školy např. algoritmus pro nalezení řešení kvadratické rovnice. Algoritmus bývá charakterizován jako posloupnost elementárních kroků, která pro libovolná povolená vstupní data dává po konečně mnoha krocích požadovaný výsledek, přičemž

- každý elementární krok je mechanicky proveditelný (tj. např. pomocí nějakého stroje),
- pro stejná vstupní data je výsledek pokaždé tentýž.

Intuitivní pojetí algoritmu

Uvedená charakteristika je často nazývána *intuitivním pojetím algoritmu*. Zdá se být formulována přesně, tak proč intuitivní? Odpověď spočívá v nejasnosti termínu „elementární krok“.

Pokud jej vztáhneme ke konkrétnímu výpočetnímu (případně abstraktnímu) zařízení, dostaneme konečně exaktní definici algoritmu.

Vzhledem k předchozím kapitolám zřejmě nepřekvapí, že za konkrétní zařízení s přesně definovanými výpočetními kroky zvolíme právě TS. Je snadno ověřitelné, že TS, který necyklí pro žádné vstupní slovo, splňuje všechny podmínky požadované v intuitivní „definici“ algoritmu.

Průvodce studiem

Přemýšlíte-li o tom, kde hledat výstup Turingova stroje, který ukončil svůj výpočet přechodem do přijímajícího nebo zamítajícího stavu, podívejte se na jeho pásku – konečný výstup je představován neprázdným obsahem pásky.

Obrácené tvrzení, a sice, že ke každému intuitivně chápanému algoritmu lze nalézt odpovídající (nikdy necyklící) TS, obecně dokázat nelze. Nikdo na celém světě však nenalezl žádný protipříklad.

Průvodce studiem

Nemožnost obecně dokázat poslední tvrzení je důsledkem vágnosti pojmu elementární krok intuitivně chápaného algoritmu.

Dostáváme tak sice nedokazatelnou, ale všeobecně přijímanou tezi:

Churchova–Turingova[10] **teze** [čti Čerčova–Ťůringova]:
Intuitivně chápaný algoritmus = algoritmus realizovatelný na Turingově stroji. *Algoritmus a TS*

Průvodce studiem

Od této chvíle můžete veškeré výsledky vztahující se k necyklícím Turingovým strojům (a tedy také k rekurzivním jazykům) interpretovat jako výsledky vztahující se k algoritmům.

2.2.4 Rozhodovací problémy

Před vlastním studiem problémů bychom měli nejdříve popsat, co budeme v našem textu rozumět pod pojmem problém.

Problémem budeme chápat úlohu nalézt odpověď na otázku vztahující se ke konečně mnoha zadaným parametrům. Zadání problému tak musí zahrnovat

- obecný popis všech jeho parametrů,
- vlastní znění úlohy.

Případem problému pak budeme nazývat problém s konkrétními hodnotami jeho parametrů.[11]

Příklad 2.18. Demonstrujme výše uvedené pojmy na následujících příkladech.

1) *Problém ekvivalence bezkontextových gramatik* je dán
   a) svými parametry[12], jež tvoří dvě bezkontextové gramatiky,
   b) otázkou: „Jsou zadané bezkontextové gramatiky ekvivalentní?“

   Případem tohoto problému je např. úloha zjistit, zda jsou ekvivalentní bezkontextové gramatiky G_1 a G_2, kde

   $$\begin{aligned} G_1 \text{ obsahuje pouze pravidla} \quad & S \to aSSb \mid Ab, \\ & A \to aSb \mid a, \\ G_2 \text{ obsahuje pouze pravidla} \quad & S \to aAb \mid Ab, \\ & A \to aSb \mid SS \mid a. \end{aligned}$$

2) *Problém mohutnosti jazyka přijímaného Turingovým strojem* je dán
   a) jediným parametrem, a sice Turingovým strojem,
   b) otázkou: „Jaká je mohutnost jazyka přijímaného zadaným Turingovým strojem?“

   Případem tohoto problému je např. úloha zjistit mohutnost jazyka přijímaného Turingovým strojem popsaným v příkladu 2.8.

Pokud se máme zabývat otázkou existence algoritmu řešícího nějaký problém, bude výhodné zúžit výběr zkoumaných problémů jenom na ty, které mají co nejjednodušší možné výsledky. Jako optimální se jeví tzv. rozhodovací problémy.

[10] Uvedená teze byla nejdříve formulována A. Churchem s použitím jiného formalismu (konkrétně λ–kalkulu), A. Turing však záhy prokázal jeho (vzájemnou) převoditelnost na svůj formalismus.

[11] Místo případu problému se lze v literatuře setkat také s termínem instance problému. [Kuč89] používá místo dvojice problém a případ problému pojmy úloha a výskyt úlohy.

[12] Místo parametrů budeme často psávat „vstup“ problému.

Definice 2.19. Za *rozhodovací problém* považujeme problém, jehož každý případ má buď řešení ANO nebo řešení NE.

Příklad 2.20. Rozhodovacími problémy jsou například

- problém ekvivalence dvou zadaných gramatik,
- problém konečnosti jazyka přijímaného Turingovým strojem.

Rozhodovacím problémem naproti tomu není problém určený otázkou: Jaká je mohutnost jazyka přijímaného zadaným Turingovým strojem?

Průvodce studiem

Omezujeme-li se pouze na studium rozhodovacích problémů, neznamená to, že bychom rezignovali na všechny ostatní. Některé z nich je možné převádět na posloupnosti rozhodovacích problémů. Například úlohu zjistit, kolik různých reálných čísel tvoří řešení zadané kvadratické rovnice lze převést na následující posloupnost rozhodovacích problémů:

1) Problém existence reálného řešení ... $\begin{cases} \text{odpověď NE značí 0 reálných řešení,} \\ \text{při odpovědi ANO přejdeme k problému 2).} \end{cases}$
2) Problém počtu různých reálných řešení kvadratické rovnice (s nezáporným diskriminantem) ... $\begin{cases} \text{odpověď NE můžeme přiřadit 1 (dvojnásobnému) reálnému řešení,} \\ \text{odpověď ANO můžeme přiřadit 2 různým reálným řešením.} \end{cases}$

Definice 2.21. Řekneme, že rozhodovací problém je *řešitelný*[13], právě když existuje TS T, který pro každý případ daného problému zastaví v koncovém stavu

$\begin{cases} q_+, & \text{je-li řešením tohoto případu ANO,} \\ q_-, & \text{je-li řešením tohoto případu NE.} \end{cases}$

Neexistuje-li takový TS, říkáme, že daný problém je neřešitelný.

Průvodce studiem

Všímáte si, že byl v definici použit nikdy necyklící TS? Tj. rozhodovací problém nazýváme řešitelný, jestliže existuje algoritmus, který pro každý případ tohoto problému vydá jako svůj výstup správné řešení (ANO nebo NE).

Příklad 2.22. Je řešitelný rozhodovací problém přijetí vstupního slova konečným automatem? (Každý případ tohoto problému je zřejmě určen konkrétním KA a konkrétním vstupním slovem.)

Pro kladné zodpovězení položené otázky musíme být podle definice schopni nalézt odpovídající TS. (Ve zkratce řečeno, musíme najít TS, který řeší daný problém.) Požadavky kladené definicí splňuje následující TS.

TS T pro vstupní slovo $\langle B, w\rangle$, kde B je KA a w je řetěz nad jeho vstupní abecedou:

1) Simuluje činnost KA B pro vstupní slovo w.
2) Pokud B skončí po přečtení slova w $\begin{cases} \text{v koncovém stavu, TS } T \text{ přijímá } \langle B, w\rangle, \\ \text{mimo koncový stav, TS } T \text{ zamítá } \langle B, w\rangle. \end{cases}$

[13] Terminologie používaná v odborné literatuře opět není jednotná. Řešitelný problém bývá nazýván také problémem rozhodnutelným, rekurzivně řešitelným nebo algoritmicky řešitelným.

Průvodce studiem

1) Ve slovním popisu Turingova stroje je již použito sousloví „pro vstupní slovo $\langle B, w\rangle$, kde B je KA a w je ...“, které podle dřívější dohody interpretujeme jako pokyn, aby TS T nejdříve zkontroloval správnost zakódování u slova, které má na svém vstupu. Jak to může provést?

Představte si třeba KA $B = (Q, \Sigma, \delta, q_0, F)$, kde
$Q = \{q_1, \ldots, q_6\}$, $q_1 \overset{ozn.}{=} q_0$,
$\Sigma = \{a_1, \ldots, a_4\}$,
a $F = \{q_2, q_3\}$.

Například přechod $\delta(q_5, a_2) = q_4$ může být zakódován jako řetěz $0^5 10^2 10^4$.

Příklad zakódování KA

Celý KA B pak může být zakódován jako řetěz

$$\langle B\rangle = 10^6 110^4 11 \underbrace{0^5 10^2 10^4 11 \ldots}_{\text{zakódování } \delta} 1110^2 10^3 11.$$

(První podřetěz šesti nul znamená množinu šesti stavů, další podřetěz čtyř nul značí čtyři symboly ve vstupní abecedě, část odpovídající přechodové funkci δ obsahuje všechny přechody kódované výše popsaným způsobem a na závěr, za trojicí jedniček, podřetězy dvou a tří nul vyjadřují, že 2. i 3. stav patří do množiny koncových stavů.)

Vstupní slovo $w = a_2 a_2 a_3$ by mohlo být kupříkladu kódováno jako řetěz $\langle w\rangle = 0^2 10^2 10^3$.

Kontrola správnosti zakódování automatu B by spočívala v ověření požadované struktury řetězu $\langle B\rangle$ (tzn. jedna jednička následována kladným počtem nul, pak dvě jedničky, kladný počet nul, dvě jedničky, atd.) a dále v ověření, že v úseku popisujícím koncové stavy nikde počet nul nepřevyšuje počet nul udávajících celkový počet stavů v Q a že v úseku popisujícím přechodovou funkci je zanesen právě jeden přechod pro každou dvojici (stav, vstupní symbol). Podobně by proběhla kontrola správnosti zakódování vstupního slova w.

2) Jak si máte představit simulaci KA B Turingovým strojem T? Například následovně: TS T si za vstupní slovo na pásku zapíše označený počáteční stav automatu B a v zakódování slova w označí 1. symbol. Potom pro označený stav a označený symbol hledá v zakódování přechodové funkce automatu B odpovídající přechod. Jakmile jej najde, místo původního stavu označí přechodem určený výsledný stav, zruší označení symbolu ve vstupním slovu a označí následující symbol vstupního slova. Uvedenou činnost opakuje, dokud může označovat nové symboly vstupního slova. V okamžiku, kdy nemá k dispozici další symbol k označení (tj. vstupní slovo je přečteno), zkontroluje, jestli je poslední označený stav koncovým stavem automatu B. Na základě výsledku této kontroly pak TS T přejde buď do přijímajícího anebo do zamítajícího stavu.

Poznámka 2.23. Je zajímavé podívat se na jazyk, který je rozhodován Turingovým strojem T z příkladu 2.22. Jde o jazyk

$\{\langle B, w\rangle;\ B$ je KA, který přijímá vstupní slovo $w\}$,

který zahrnuje zakódování všech případů problému (zkoumaného v příkladu 2.22), jež mají řešení ANO.

Uvedený poznatek má obecnější platnost, jak sděluje následující věta.

Věta 2.24. *Problém $\mathcal{P}$ je řešitelný, právě když jazyk $\{\langle P\rangle;\ P$ je případ problému $\mathcal{P}$ s řešením ANO$\}$ je rekurzivní.*

Řešitelné problémy $\sim$ rekurzivní jazyky

Důkaz. Jde o přímý důsledek definic rekurzivního jazyka a řešitelného rozhodovacího problému. □

Poznámka 2.25. Věta 2.24 poukazuje na vzájemnou převoditelnost konkrétních rekurzivních jazyků a řešitelných problémů. Veškeré poznatky získané o rekurzivních jazycích tak můžeme aplikovat na řešitelné problémy.

Průvodce studiem

Kvůli velké důležitosti popisovaných vzájemných převodů (hojně využívaných v kapitole 3) si ještě jednou uvědomme, že jazyk „odpovídající“ řešitelnému problému vždy zahrnuje pouze zakódování těch případů daného problému, jež mají řešení ANO. Například jazyk „odpovídající“ problému prvočíselnosti přirozených čísel by zahrnoval zakódování všech prvočísel.

Pro úplnost uvedeme také definici částečně řešitelných problémů a jejich převoditelnost na částečně rekurzivní jazyky.

Definice 2.26. Řekneme, že rozhodovací problém je *částečně řešitelný*[14], právě když existuje TS T, který pro každý případ daného problému

$$\begin{cases} \text{zastaví v koncovém stavu } q_+, \text{ je-li řešením tohoto případu ANO,} \\ \text{zastaví v koncovém stavu } q_- \text{ anebo cyklí, je-li řešením tohoto případu NE.} \end{cases}$$

Průvodce studiem

Všimněte si, že TS použitý v definici nepředstavuje algoritmus, nýbrž proceduru, protože pro některá vstupní slova může cyklit.

Mimochodem, schopnost Turingových strojů začít cyklit dobře známe u jiných, reálnějších zařízení, jež většinou okupují naše pracovní stoly, a sice u stolních počítačů. Tento rys (stolních) počítačů, mnohými považovaný za základní, možná leckoho přesvědčí o tom, že TS skutečně představuje obecný výpočetní prostředek.

Věta 2.27. *Rozhodovací problém $\mathcal{P}$ je částečně řešitelný, právě když jazyk $\{\langle P\rangle;\ P$ je případ problému $\mathcal{P}$ s řešením ANO$\}$ je částečně rekurzivní.*

Částečná řešitelnost ∼ částečná rekurzivita

Důkaz. Jde o přímý důsledek definic částečně rekurzivního jazyka a částečně řešitelného rozhodovacího problému. □

2.2.5 Problém zastavení Turingova stroje

Definice 2.28. *Problémem zastavení Turingova stroje* nazýváme rozhodovací problém, v němž se má stanovit, zda zadaný TS pro dané vstupní slovo přejde po konečném počtu výpočetních kroků do přijímajícího nebo zamítajícího stavu.

Poznámka 2.29. Pokud by byl problém zastavení TS řešitelný, musel by existovat TS T' (neboli algoritmus), který pro každé vstupní slovo $\langle T, w\rangle$, kde T je TS a w jeho vstupní slovo,

$$\text{zastaví v koncovém stavu} \quad \begin{cases} q_+, \text{ jestliže } T \text{ přijímá anebo zamítá } w, \\ q_-, \text{ jestliže } T \text{ pro vstupní slovo } w \text{ cyklí.} \end{cases}$$

V poznámce 2.29 se objevuje požadavek existence Turingova stroje, který dokáže simulovat činnost libovolného TS T. Stroj s touto vlastností bývá nazýván *univerzálním Turingovým strojem*.

Univerzální TS

[14] V odborné literatuře bývá částečně řešitelný problém nazýván rovněž problémem částečně rozhodnutelným nebo částečně rekurzivně řešitelným.

Průvodce studiem

Připomenete-li si způsob, jakým TS simuloval činnost libovolného KA v komentáři k příkladu 2.22, není těžké představit si činnost univerzálního Turingova stroje.

Univerzální TS svými schopnostmi evidentně splňuje požadavky očekávané od obecného výpočetního prostředku, neboť dokáže zpracovávat libovolný algoritmus (představovaný Turingovým strojem) včetně příslušných vstupních dat. Tvoří tak teoretický předobraz reálných počítačů a nalezení jakýchkoli výpočetních omezení znamená stejná omezení pro reálné počítače.

Neřešitelnost problému zastavení TS

Věta 2.30. *Problém zastavení Turingova stroje není řešitelný.*

Průvodce studiem

Uvedené tvrzení není zase tak překvapivé. Vždyť jakým způsobem bychom měli zjistit, že TS (nebo počítačový program) cyklí a že není jenom ve velmi dlouhé fázi výpočtu, který bude po konečné době ukončen?

Důkaz. Důkaz povedeme sporem.

Předpokládejme, že jde o řešitelný problém. Podle definice (a také poznámky 2.29) pak existuje TS H, který pro vstupní slovo $\langle T, w\rangle$, kde T je TS a w jeho vstupní slovo

- přijímá $\langle T, w\rangle$, jestliže TS T přijímá anebo zamítá w,
- zamítá $\langle T, w\rangle$, jestliže TS T cyklí pro vstupní slovo w.

Uvažme TS D, který pro vstupní slovo $\langle T\rangle$, kde T je TS:

1) Simuluje činnost TS H pro vstupní slovo $\langle T, \langle T\rangle\rangle$.
2) Jestliže H $\begin{cases} \text{přijímá } \langle T, \langle T\rangle\rangle, \text{ TS } D \text{ začne cyklit,} \\ \text{zamítá } \langle T, \langle T\rangle\rangle, \text{ TS } D \text{ přijme } \langle T\rangle. \end{cases}$

Průvodce studiem

Má-li TS D simulovat činnost stroje H, musí mít k dispozici pro tuto simulaci vstupní slovo kódující 2 objekty: jednak TS, jednak vstupní slovo pro tento TS. Zakódování Turingova stroje má D na svém vstupu, použije jej tedy dvakrát: jednou coby TS, podruhé coby řetěz symbolů.

Na základě popisu TS D a TS H nám tedy vychází:

TS D $\begin{cases} \text{cyklí pro vstupní slovo } \langle T\rangle, \text{ právě když TS } T \text{ přijímá nebo zamítá vstupní slovo } \langle T\rangle, \\ \text{přijímá vstupní slovo } \langle T\rangle, \text{ právě když TS } T \text{ cyklí pro vstupní slovo } \langle T\rangle. \end{cases}$

TS D dostává na svůj vstup zakódování sebe samého.

Také D je TS. Přepíšeme-li tedy předchozí vztah pro případ $T = D$, dostaneme:

Průvodce studiem

Pokud vám připadá málo pochopitelná situace, že TS D dostane na svůj vstup zakódování sebe samého a pracuje s ním jako s jakýmkoli jiným vstupním slovem, představte si analogickou (a tentokrát již reálnou) situaci: Jistě lze napsat v programovacím jazyku Pascal počítačový program, který každý program zapsaný v Pascalu převede do strojového kódu. Spustíte-li jej sám na sebe (jde přece o program zapsaný v Pascalu), převede svůj vlastní kód do strojového kódu.

TS D $\begin{cases} \text{cyklí pro vstupní slovo } \langle D \rangle, \text{ právě když TS } D \text{ přijímá anebo zamítá vstupní slovo } \langle D \rangle, \\ \text{přijímá vstupní slovo } \langle D \rangle, \text{ právě když TS } D \text{ cyklí pro vstupní slovo } \langle D \rangle. \end{cases}$

To je ovšem evidentní spor. (Náš původní předpoklad o řešitelnosti problému zastavení TS se tedy ukázal být nesprávným.) □

Průvodce studiem

Nyní máme dokázánu existenci problému, který nelze řešit na žádném počítači. Problémů s touto vlastností pochopitelně existuje celá řada. Další však hledat nebudeme a spokojíme se s velmi jednoduše formulovaným a srozumitelným problémem zastavení TS.

Od této chvíle máte zaručeno, že každý pokus sestavit počítačový program, který by pro libovolný počítačový program (včetně vstupních dat) dokázal určit, že se nezacyklí, má stejnou naději na úspěch jako pokus sestrojit perpetuum mobile.

Shrnutí

Rekurzivní jazyk je jazyk, který je rozhodován nějakým TS. Částečně rekurzivní jazyk je jazyk, který je přijímán nějakým TS. Třída částečně rekurzivních jazyků je rovna třídě jazyků typu 0.

Aby kterýkoli TS mohl pracovat s nějakými objekty, musí je dostat na svůj vstup v podobě řetězů vstupních symbolů, jež dohodnutým způsobem jednoznačně kódují dané objekty. Úvodní činností TS v takovém případě bude vždy kontrola, zda předložený řetěz představuje správné zakódování očekávaných objektů.

Churchova–Turingova teze praví, že každý intuitivně chápaný algoritmus lze realizovat na TS a že každý algoritmus realizovatelný na TS odpovídá intuitivnímu chápání algoritmu. TS na základě této teze slouží jako exaktní model algoritmu.

Problémem rozumíme úlohu zjistit určitou vlastnost či podmínku platící pro jisté (obecné) objekty. Případem problému pak rozumíme úlohu týkající se konkrétně zadaných objektů.

Rozhodovací problém je problém, jehož každý případ má řešení typu ANO/NE.

Každý rozhodovací problém nazveme řešitelným, pokud existuje TS, který pro každý případ daného problému, jenž má řešení $\begin{cases} \text{ANO, zastaví ve stavu } q_+, \\ \text{NE, zastaví ve stavu } q_-. \end{cases}$

Každý rozhodovací problém nazveme částečně řešitelným, pokud existuje TS, který pro každý případ daného problému, jenž má řešení $\begin{cases} \text{ANO, zastaví ve stavu } q_+, \\ \text{NE, zastaví ve stavu } q_- \text{ anebo cyklí.} \end{cases}$

Každý (částečně) řešitelný rozhodovací problém lze převést na (částečně) rekurzivní jazyk.

Problém zastavení TS je typickým příkladem neřešitelného problému.

Pojmy k zapamatování

- Rekurzivní jazyk,
- částečně rekurzivní jazyk,
- kódování objektů,
- rozhodovací problém,
- řešitelný rozhodovací problém,
- částečně řešitelný rozhodovací problém,
- problém zastavení TS.

Kontrolní otázky

1. *Platí, že každý rekurzivní jazyk je jazykem typu 0?*
2. *Může se zacyklit TS, který představuje algoritmus?*

Cvičení

1. Zjistěte, jestli je problém zastavení TS částečně řešitelný.

Úkoly k textu

1. Napište posloupnosti konfigurací, jež postupně popisují výpočty TS z příkladu 2.17 pro vstupní slova $\varepsilon, ae, aaa, aeea, aeaaea$.
2. Upravte přechodovou funkci TS z příkladu 2.17 tak, aby při hledání „středu“ vstupního slova pracoval „obousměrně“ (dle komentáře předcházejícího formální popis T).

Řešení

1. K prokázání částečné řešitelnosti problému zastavení TS je zapotřebí nalézt TS, který přijímá jazyk $\{\langle T, w\rangle;\ T$ je TS, který zastaví pro vstupní slovo $w\}$. Takovým strojem je například TS R, který pro vstupní slovo $\langle T, w\rangle$, kde T je TS a w jeho vstupní slovo:
   1) Simuluje činnost TS T pro vstupní slovo w.
   2) Jestliže TS T přijímá nebo zamítá w, pak TS R přijme $\langle T, w\rangle$.

3 Složitost

Disciplína zvaná složitost se zabývá algoritmicky řešitelnými problémy, u nichž zkoumá výpočetní obtížnost jejich řešení.

3.1 Úvodní pojmy

Studijní cíle: Po prostudování kapitoly studující porozumí důvodům vedoucím k zjednodušujícím principům používaným při studiu časové složitosti. Dále bude rozumět řádovému porovnávání funkcí a bude vědět, jak se zavádí časová složitost deterministického i nedeterministického Turingova stroje.

Klíčová slova: O-notace, časová složitost.

Potřebný čas: 60 minut.

Časová a paměťová složitost

Poznamenejme, že složitost každého algoritmu může být studována buď z hlediska paměťové náročnosti nebo z hlediska časové náročnosti. Paměťovou náročností rozumíme požadavek na velikost paměti počítače (v případě TS na počet políček pásky), jež je zapotřebí k provedení výpočtu. Podobně časovou náročností rozumíme čas potřebný pro výpočet. Tento čas však nebudeme měřit v časových jednotkách, nýbrž počtem provedených elementárních kroků algoritmu.

Vzhledem k omezenému rozsahu našeho textu se nadále budeme věnovat výhradně časové složitosti.

Příklad 3.1. Prozkoumejme časovou náročnost algoritmu z příkladu 2.17, v němž TS T rozhodoval jazyk $L = \{zz;\ z \in \{a, e\}^+\}$. Tzn. podívejme se na počet výpočetních kroků provedených strojem T pro vstupní slovo w délky n. Předpokládejme nejdříve, že $w \in L$.

Průvodce studiem

Následující text (až do příštího „Průvodce studiem“) berte pouze informativně, tj. pokud jste například „v časové tísni“, neutrpíte vůbec žádnou újmu, když „neprověříte“ správnost níže uvedených propočtů.

V 1. fázi, kdy TS T hledá „střed“ vstupního slova, nejdříve svými prvními n kroky přesune čtecí hlavu za vstupní slovo, přičemž označí první symbol 1 tečkou (resp. pruhem) a poslední symbol 2 tečkami. Při návratu nad 1. symbol vykoná čtecí hlava dalších n kroků. Při označování dalšího páru symbolů vykoná čtecí hlava již jen $(n-1)+(n-2)$ kroků, neboť k jejímu obratu dojde nad n-tým políčkem pásky (a ne až za vstupním slovem) a při návratu se nevrací až nad výchozí políčko. Při označování následujícího páru pak vykoná $(n-3)+(n-4)$ kroků, atd. TS T tedy v 1. fázi vykonal celkem $n+n+(n-1)+\ldots+3+2 = n+\frac{(n-1)*(n+2)}{2} = \frac{n^2+3n-2}{2}$ výpočetních kroků.

Ve 2. fázi TS T porovnává 1. polovinu slova s 2. polovinou. Před vlastním porovnáváním však přesune čtecí hlavu $\frac{n}{2}-1$ kroky nad 1. symbol. Při porovnávání symbolů na stejných pozicích vždy vykoná:

1) $\frac{n}{2}$ kroků potřebných pro přesun nad stejnou pozici v 2. polovině slova.
2) 1 krok doprava pro kontrolu, jestli není porovnávání hotovo; v kladném případě provede ještě 1 krok, kterým přejde do přijímajícího stavu.
3) $\frac{n}{2}+1$ kroků pro návrat nad poslední porovnávaný symbol z 1. poloviny slova.
4) 1 krok doprava nad další porovnávaný symbol.

Počty kroků popsané pod body 1) – 4) vykoná TS T pro vstupní slovo patřící do jazyka L celkem $(\frac{n}{2}-1)$-krát, při porovnávání symbolů na posledních pozicích totiž provede pouze kroky popsané pod body 1) a 2), neboť bod 2) je zakončen úspěšnou kontrolou. Na 2. fázi tak připadá celkem $\frac{n}{2}-1+(\frac{n}{2}-1)*(\frac{n}{2}+1+\frac{n}{2}+1+1)+\frac{n}{2}+2=\frac{n^2+3n-4}{2}$ výpočetních kroků.

Pokud by vstupní slovo nepatřilo do jazyka L, buď by byl počet vykonaných kroků ve 2. fázi menší, nebo by se TS T ke 2. fázi vůbec nedostal. Z toho plyne, že počet kroků výpočtu stroje T (představujícího zde algoritmus) pro libovolné vstupní slovo délky n činí nejvýše $\frac{n^2+3n-2}{2}+\frac{n^2+3n-4}{2}=n^2+3n-3$.

Průvodce studiem

Připadá vám takové počítání dost úmorné? Nevěšte hlavu, tímto způsobem se počítání složitosti algoritmu standardně neprovádí. Jak vyplyne z dalších úvah, budeme ignorovat koeficienty a konstanty, které by se snad ve výrazu popisujícím takovou složitost měly objevit. Tj. budeme uvažovat pouze řádovou velikost – pro předchozí příklad bychom tedy uváděli výsledek „řádově“ n^2. Populárně řečeno, budeme „bezostyšně“ (a nad obvyklou míru) „zaokrouhlovat“. Přesvědčení o rozumnosti uvedeného přístupu snad ve vás příklad 3.1 posílil.

Charakteristika zkoumání složitosti algoritmů

V příkladu 3.1 lze vysledovat některé z následujících rysů, jež budeme používat pro popis složitosti algoritmů:

- Algoritmus budeme reprezentovat pomocí Turingova stroje. Hlavním důvodem je to, že jsme si jiný formální model algoritmu ani nezavedli. Je sice pravda, že s jiným modelem algoritmu bychom mohli získat odlišné výsledky, ne však natolik odlišné, abychom následně obdrželi jiné třídy složitostí než ke kterým v textu směřujeme (Konkrétně máme na mysli třídy P a NP.).
- Časovou složitost každého algoritmu budeme vztahovat k velikosti vstupních dat, tj. v případě TS k velikosti vstupních slov. Dopouštíme se tím jistého zjednodušení, protože tím zcela pomíjíme skutečnost, že stejná délka dvou vstupních slov (představujících např. zakódování dvou grafů, jednak jednoduchého s minimem hran, jednak složitějšího s méně vrcholy a více hranami) může vést k naprosto rozdílné časové náročnosti. U zakódování libovolných objektů budeme předpokládat, že je provedeno „rozumně“ (například zakódování čísla 7 v podobě binárního zápisu 111 považujeme za rozumné na rozdíl od unárního zápisu 1111111).
- Algoritmus budeme analyzovat z hlediska chování v nejhorším případě, tj. časovou složitost pro každé $n \in \mathbb{N}$ položíme rovnu délce nejdelšího možného výpočtu ze všech výpočtů TS pro vstupní slova délek n. Tím budeme mít zaručeno, že námi dosažené výsledky budou představovat horní hranici, nepřekročitelnou bez ohledu na to, jaké („sebebizarnější“) slovo bude na vstupu.
- U funkcí popisujících časovou složitost budeme uvažovat pouze jejich řádovou velikost, tj. například složitosti lišící se konstantním násobkem budeme považovat za stejné.

Průvodce studiem

Vidíte mohutnost zjednodušení plynoucího z posledního bodu? Jeho hlavním cílem je podstatně redukovat množství funkcí používaných k popisům složitostí algoritmů.

Jestliže máte pocit, že s uvedeným zjednodušováním nemůžeme získat žádné rozumné výsledky, budete překvapeni. (Jen výjimečně by totiž problémy z běžné praxe vedly k algoritmům se složitostí např. $10^{10} \cdot n^2$.)

V následující definici jednak přesně popíšeme, jak budeme funkce řádově porovnávat, jednak zavedeme příslušná značení [15]

Řádové porovnávání funkcí

Definice 3.2. Nechť f, g jsou dvě funkce, které přiřazují nezáporným celým číslům reálná čísla. Pak řekneme, že

- funkce f *roste řádově nejvýše* jako funkce g, a píšeme $f(n) = O(g(n))$, právě když existují čísla $K > 0$ a $n_0 \in \mathbb{N}$ taková, že pro každé přirozené číslo $n \geq n_0$ platí $f(n) \leq K \cdot g(n)$,
- funkce f *roste řádově aspoň* jako funkce g, a píšeme $f(n) = \Omega(g(n))$, právě když existují čísla $k > 0$ a $n_0 \in \mathbb{N}$ taková, že pro každé přirozené číslo $n \geq n_0$ platí $f(n) \geq k \cdot g(n)$,
- funkce f *roste řádově stejně*[16] jako funkce g, a píšeme $f(n) = \Theta(g(n))$, právě když $f(n) = O(g(n))$ a $f(n) = \Omega(g(n))$.

Průvodce studiem

Uvedená řádová srovnání především popisují chování uvedených funkcí pro dostatečně vysoká n (přesněji, pro všechna přirozená n od jistého n_0 počínaje), přitom však za účelem požadovaného výsledku „máme povoleno“ násobit funkci g libovolnou kladnou konstantou.

Příklad 3.3. Podívejme se na řádové srovnání funkcí $f(n) = 3n^3 - n^2 + 2n$ a $g(n) = n^3$. Jak už napovídá název „řádové srovnání“, zřejmě bychom měli porovnávat nejvyšší řády uvedených polynomických funkcí. Jak f tak g má nejvyšší řád roven 3, mělo by tedy jít o funkce řádově stejné. Ověřme správnost tohoto závěru za pomoci definice 3.2.

Máme-li prokázat, že funkce f roste řádově nejvýše jako g, musíme najít přirozená čísla K a n_0 taková, že pro každé přirozené číslo $n \geq n_0$ platí

$$3n^3 - n^2 + 2n \leq K \cdot n^3.$$

Úpravou uvedené nerovnice dostaneme

$$n^3(K - 3) + n^2 - 2n \geq 0.$$

Pro $K = 4$ postupně máme:

$$\begin{gathered} n^3 + n^2 - 2n \geq 0, \\ n(n^2 + n - 2) \geq 0, \\ n(n + 2)(n - 1) \geq 0. \end{gathered}$$

Jelikož je poslední nerovnice splněna pro $K = 4$ a pro všechna $n \geq n_0 = 1$, platí $f(n) = O(g(n))$.

Důkaz toho, že funkce f roste řádově aspoň jako g, se provede obdobně.

3.1.1 Složitost Turingova stroje a nedeterministického Turingova stroje

Složitost definujeme jen pro necyklící TS!

Chceme-li používat jako formální model algoritmu Turingův stroj, musíme definovat složitost nikdy necyklícího Turingova stroje.

Definice 3.4. Nechť T je TS, který pro každé vstupní slovo zastaví. *Časovou složitostí* T nazveme funkci f (přiřazující každému nezápornému celému číslu nezáporné celé číslo) takovou, že pro každé $n \in \mathbb{N}_0$ je

Definice složitosti TS pomocí chování v nejhorším případě

$f(n) = \max\{k \in \mathbb{N}_0;$ TS T pro vstupní slovo w délky n zastaví po k výpočetních krocích$\}$.

[15] Uváděné značení bývá běžně nazýváno „velkou O-notací“ nebo „asymptotickou notací“.

[16] Používají se také pojmy řádově ekvivalentní nebo asymptoticky ekvivalentní funkce.

Průvodce studiem

Všímáte si, že právě uvedená složitost algoritmu (představovaného nikdy necyklícím TS) je skutečně definována prostřednictvím „nejhoršího možného případu“, který se může vyskytnout mezi všemi vstupními daty o stejné velikosti n? Nejhorším možným případem tady rozumíme nejdelší možný výpočet.

Kvůli definici jedné velmi užitečné třídy složitosti (a sice NP) budeme potřebovat zavést rovněž pojem časové složitosti nedeterministického Turingova stroje.

Definice 3.5. Nechť T je NTS, který má pro každé vstupní slovo konečné všechny výpočetní větve. *Časovou složitostí* T nazveme funkci f (přiřazující každému nezápornému celému číslu nezáporné celé číslo) takovou, že pro každé $n \in \mathbb{N}_0$ je

$f(n) = \max\{k \in \mathbb{N}_0;$ maximální počet kroků výpočtu T pro vstupní slovo w, jež má délku n, sestává z k kroků$\}$.

Složitost definována jen pro NTS bez „cyklících větví“.

Průvodce studiem

Uvedená definice si zasluhuje trochu pozornosti při čtení. Obsahuje totiž dvě do sebe vnořené úrovně.

Nižší úroveň je představována číslem k, jež znamená nejdelší větev ze všech výpočetních větví stroje T pro konkrétní slovo w. Slovo w musí mít délku n, abychom je vůbec mohli zohledňovat. Povšimněte si zároveň, že po zmiňované nejdelší výpočetní větvi vůbec nepožadujeme, aby končila konfigurací s přijímajícím stavem.

A vyšší úroveň? Vzhledem k tomu, že nad m-prvkovou vstupní abecedou existuje právě m^n různých vstupních slov délky n, je $f(n)$ rovno maximu m^n-prvkové množiny zahrnující čísla k příslušná oněm slovům délky n.

Definice 3.6. Nechť T je TS (resp. NTS) s časovou složitostí f. O stroji T prohlásíme, že má *polynomickou časovou složitost*, právě když existuje polynom p takový, že $f(n) \leq p(n)$ pro všechna $n \in \mathbb{N}_0$.

Průvodce studiem

Jestliže tedy nějaký deterministický nebo nedeterministický Turingův stroj pracuje s polynomickou časovou složitostí, máte zaručenu existenci takového polynomu p, že libovolné slovo délky n je daným strojem „zpracováno“ v čase nejvýše $p(n)$.

Shrnutí

Rozlišujeme časovou a paměťovou složitost algoritmů. V případě časové složitosti nás zajímá počet elementárních kroků algoritmu potřebný k provedení výpočtu. Složitost bývá standardně popisována jako funkce závislá na velikosti vstupních dat algoritmu a u této funkce bývá uvažována pouze její řádová velikost.

Při řádovém porovnávání funkcí, zjednodušeně řečeno, nezohledňujeme koeficienty násobící porovnávané funkce a zjišťujeme vzájemný vztah jejich funkčních hodnot pro dostatečně vysoké argumenty.

Časovou složitost nikdy necyklícího TS (resp. NTS, jehož výpočetní větve jsou konečné pro libovolné vstupní slovo) T definujeme jako funkci f takovou, že T pro žádné vstupní slovo délky n nevykoná více než $f(n)$ kroků výpočtu (v případě NTS pak nemá žádná výpočetní větev délku větší než $f(n)$).

O TS (resp. NTS) říkáme, že pracuje s polynomickou časovou složitostí, pokud existuje polynom p takový, že libovolné slovo délky n je daným strojem „zpracováno“ v čase nejvýše $p(n)$.

Pojmy k zapamatování

- O-notace,
- časová složitost TS a NTS,
- polynomická časová složitost.

Kontrolní otázky

1. *Jaká je časová složitost TS, jehož počáteční stav je totožný s přijímajícím stavem?*

Cvičení

1. Zjistěte, zda jsou funkce $f(n) = 1$ a $g(n) = 5$ řádově stejné.

Úkoly k textu

1. Ověřte platnost vztahu $f(n) = \Omega(g(n))$ v příkladu 3.3.

Řešení

1. Jsou řádově stejné, neboť nerovnosti $k \cdot 5 \leq 1 \leq K \cdot 5$ platí pro $k = \frac{1}{5}, K = 1$ a pro všechna přirozená čísla $n \geq n_0 = 1$.

3.2 Třídy složitostí P a NP

Studijní cíle: Po prostudování kapitoly bude studující rozumět důvodům vedoucím k zavedení tříd složitostí P a NP. Bude umět tyto třídy definovat a bude umět popsat, co je známo o jejich vzájemném vztahu.

Klíčová slova: Třída P, třída NP.

Potřebný čas: 45 minut.

Existuje nepřeberné množství algoritmů s nejrůznějšími složitostmi. Požadavek kategorizace algoritmů do přijatelně malého počtu tříd složitostí se tedy jeví rozumně. Nejjednodušší možné dělení by zřejmě rozdělilo algoritmy na rychlé a pomalé. Jenže co si představit pod pojmem pomalý algoritmus? Je to ten, který nám na stolním počítači poběží 1 hodinu, 1 rok nebo snad 1000 let? Zřejmě je nutné nalézt nějakou dělící čáru, která by oddělila pomalé algoritmy od rychlých a přitom měla dlouhodobější platnost, protože některé algoritmy, jež nám z dnešního pohledu připadají pomalé, se mohou jevit s modernějšími a rychlejšími výpočetními systémy jako dostatečně rychlé. Zásadní krok vedoucí k nezávislosti na konkrétních reálných počítačích jsme již vykonali tím, že jsme si vybrali jako model pro zkoumání složitosti algoritmů Turingův stroj. Nicméně otázka, co považovat za pomalý a co za rychlý algoritmus, přetrvává. Podívejme se na ni tedy podrobněji prostřednictvím několika názorných tabulek.

Představme si počítač, u nějž trvá provedení 1 instrukce 1 nanosekundu. Je-li počet instrukcí potřebných k výpočtu algoritmu pro vstupní data o velikosti n popsán funkcí $f(n)$, pak tabulka 2 udává celkový výpočetní čas pro různá n a $f(n)$.

Na základě tabulky 2 čtenář nejspíš jako rozumné kritérium dostatečné rychlosti algoritmu uzná požadavek, aby časová složitost byla popsána nejvýše polynomickou funkcí.

$f(n)$	n						
	20	40	60	80	100	500	1 000
n	20 ns	40 ns	60 ns	80 ns	0,1 μs	0,5 μs	1 μs
n^2	0,4 μs	1,6 μs	3,6 μs	6,4 μs	10 μs	0,25 ms	1 ms
n^3	8 μs	6,4 μs	0,22 ms	0,5 ms	1 ms	0,125 s	1 s
n^5	3,2 ms	0,1 s	0,8 s	3,3 s	10 s	9 hodin	12 dní
2^n	1 ms	18 min	37 let	$14{\cdot}10^9$ let			
$n!$	77 let						

Tabulka 2: Porovnání délek výpočtů u algoritmů s různou časovou náročností

Průvodce studiem

Kritérium

prakticky použitelný algoritmus = algoritmus s nejvýše polynomickou časovou složitostí

pochopitelně nelze brát jako dogma. U funkcí

$f_1(n) = 2^{100} \cdot n$

$f_2(n) = 2^{n^{0,0001}}$ ($= 2^{10}$ pro $n = 10^{10^4}$)

byste asi dlouho nepřemýšleli, která z nich popisuje šikovnější algoritmus.

Je ovšem pravdou, že značná část známých algoritmů, jež pracují v polynomickém čase, má složitost popsánu polynomem se stupněm nejvýše 4.

Ještě přesvědčivější je tabulka 3, která znázorňuje vliv zrychlování výpočetních systémů na rozsah dat zpracovatelných pomocí algoritmů s polynomickými i horšími časovými složitostmi. Funkce f vždy udává příslušnou časovou složitost, na odpovídajících řádcích tabulky je pak velikost dat zpracovatelných algoritmem se složitostí f za předem daný časový limit.

$f(n)$	zrychlení výpočtu		
	1 x	100 x	1000 x
n	100	10 000	100 000
n^2	100	1 000	3 162
n^3	100	464	1 000
n^5	100	251	398
2^n	100	106	109
$n!$	100	100	101

Tabulka 3: Zvětšení rozsahu zpracovatelných dat při 100krát a 1000krát rychlejších počítačích

Průvodce studiem

V tabulce 3 je vidět, že už pro exponenciální algoritmy je typická existence mezní velikosti vstupních dat, nad níž je úloha prakticky neřešitelná i při zvýšení rychlosti počítače o několik řádů.

Nyní můžeme přistoupit k definici třídy zahrnující všechny problémy řešitelné v polynomickém čase.

Definice 3.7. Třídu všech jazyků přijímaných (deterministickými!) Turingovými stroji s polynomickými časovými složitostmi nazýváme *třídou* P. *Třída P*

Průvodce studiem

Do třídy P tedy zařadíme každý jazyk, který je přijímán nějakým TS s polynomickou časovou složitostí, a žádný jiný.

Pokud v tuto chvíli přemýšlíte nad tím, že TS přijímající nějaký jazyk může pro některá vstupní slova cyklit, připomeňte si definici TS majícího polynomickou časovou složitost – tuto vlastnost jsme definovali pouze pro nikdy necyklící TS (viz definice 3.4 a 3.6).

Poznámka 3.8. Poslední definici můžeme použít k definici *třídy problémů* P. Věta 2.24 totiž dává do souvislosti řešitelné problémy a rekurzivní jazyky. Jestliže tedy jazyk „příslušný“ k nějakému problému patří do třídy P, zařadíme onen problém do třídy problémů P.

Průvodce studiem

Potřebujete-li kratší popis třídy problémů P, jednoduše do ní zařaďte všechny problémy řešitelné (deterministickými) Turingovými stroji s polynomickými časovými složitostmi.

Vzhledem k tomu, že TS s polynomickou časovou složitostí nikdy necyklí, představuje vlastně jistý algoritmus pracující „v polynomickém čase“. Jestli vás napadlo, zda nelze třídu P definovat pomocí algoritmů s polynomickou časovou složitostí, pak vás jistě potěší kladná odpověď – jen je třeba mít zavedenu „rozumnou“ definici těchto algoritmů včetně polynomické časové složitosti.

K definici další, bohatší třídy jazyků nám velice dobře poslouží NTS.

Definice 3.9. Třídu všech jazyků přijímaných nedeterministickými Turingovými stroji s polynomickými časovými složitostmi nazýváme *třídou*[17] NP. *Třída NP*

Průvodce studiem

Do třídy NP tedy zařadíme každý jazyk, který je přijímán nějakým NTS s polynomickou časovou složitostí, a žádný jiný.

Poznámka 3.10. I v případě definice *třídy problémů* NP využijeme větu 2.24. Tj., jestliže jazyk „příslušný“ k nějakému problému patří do třídy NP, zařadíme onen problém do třídy problémů NP.

Průvodce studiem

Metoda „hrubé síly“ a třída NP

Typickým představitelem třídy NP je problém, pro jehož vyřešení je znám algoritmus pracující tzv. metodou „hrubé síly“, tj. algoritmus probírající všechny existující varianty vedoucí k možnému řešení (např. 2^n existujících podmnožin množiny s n prvky). Přitom však musí být splněna podmínka, že ověření správnosti každé z těchto variant vyžaduje nejvýše polynomický čas. Popisovanou situaci vám lépe osvětlí příklad po následující definici.

Definice 3.11. *Klikou v neorientovaném grafu* $G = (V, E)$ rozumíme takovou podmnožinu K množiny vrcholů V, že každé dva různé vrcholy z K jsou spojeny hranou (z E).

[17] Označení tříd P a NP je zkratkou slov polynomický a nedeterministicky polynomický. Vzhledem k tomu, že jde o třídy problémů definovaných pomocí časových složitostí, jsou pro ně užívány rovněž názvy PTIME a NPTIME.

Průvodce studiem

Jinak řečeno, klika je takový podgraf grafu G, který je úplný.

Příklad 3.12. Ověřte, že jazyk $L = \{\langle G, k\rangle;\ G$ je neorientovaný graf s klikou o k vrcholech$\}$ je z třídy NP.

Pro ověření příslušnosti k třídě NP musíme být schopni nalézt NTS s polynomickou časovou složitostí, který přijímá jazyk L.

Požadované podmínky splňuje NTS T, který pro vstupní slovo $\langle G, k\rangle$, kde G je neorientovaný graf a $k \in \mathbb{N}$:

1) Nedeterministicky vybere množinu C obsahující k vrcholů z grafu G.
2) Ověří, zda jsou v grafu G všechny vrcholy z C navzájem pospojovány hranami. Pokud ano, NTS T přijímá vstupní slovo $\langle G, k\rangle$; jinak T zamítá $\langle G, k\rangle$.

NTS T evidentně přijímá jazyk L. Zbývá nám pouze prokázat, že NTS T má polynomickou časovou složitost:

Provedení bodu 1) si lze představit následovně:

NTS T si v prvním kroku nedeterministicky vybere, zda označí nebo neoznačí 1. vrchol grafu G, ve druhém kroku nedeterministicky vybere, zda označí nebo neoznačí 2. vrchol, atd. V případě označení vrcholu pokaždé zvýší hodnotu pomocné proměnné o 1 a následně kontroluje, jestli uvedená proměnná nedosáhla hodnoty k – pokud ano, vybírání množiny C je u konce; v opačném případu pokračuje činnost stroje T dalším nedeterministickým výběrem (je-li ovšem ještě k dispozici nějaký dosud neuvažovaný vrchol; není-li, příslušná nedeterministická větev výpočtu končí bez výběru množiny C).

Popsaný postup vyžaduje projít úsek vstupního slova, představujícího zakódování množiny vrcholů grafu G – vše nejlépe vynikne, když si tento úsek představíme v podobě kódů jednotlivých vrcholů. Zmiňovaný úsek tvoří pouze část celého vstupního slova (délky n), a jeho délka je tedy kratší než n. Samotné označení k vrcholů by proto mělo zabrat méně kroků než je nějaký násobek čísla n, k tomu je ovšem nutno připočíst k-násobnou kontrolu, při níž čtecí hlava musí pokaždé absolvovat cestu za vstupní slovo nad část pásky s pomocnou proměnnou, upravit ji a vrátit se zpátky, tj. cestu délky zhruba $2n$. Celkem tedy vychází na kontrolu zhruba $k \cdot 2n$ dalších kroků čili řádově n^2, neboť $k = O(n)$. Dohromady tak dostáváme řádově n kroků (pro označování vrcholů) + n^2 kroků (pro kontrolu pomocné proměnné), tj. řádově n^2.

Bod 2) lze realizovat následovně:

Pro každý označený vrchol musí NTS provést kontrolu, zda je spojen hranou s ostatními označenými vrcholy. Přesněji, pro 1. vrchol musí provést $k-1$ kontrol, pro 2. vrchol $k-2$ kontrol, atd. až pro $(k-1)$. vrchol 1 kontrolu. Uvedené počty tvoří aritmetickou řadu se součtem $\frac{k(k-1)}{2}$. Provedení jedné kontroly pro prověřovaný vrchol přitom znamená cestu čtecí hlavy nad odpovídající úsek zakódované množiny hran a návrat zpět, tj. zhruba $2n$ kroků. Protože $k = O(n)$, dostáváme pro celkovou kontrolu (hran) řádově n^3 kroků.

Pro bod 1) tedy dostáváme řádově kvadratický počet a pro bod 2) kubický počet jistých kroků – ty ovšem nepředstavují konkrétní kroky výpočtu Turingova stroje T, neboť označení vrcholu zakódovaného řetězem délky 3 může být provedeno například označením každého symbolu použitého v zakódování (tj. alespoň třemi výpočetními kroky) nebo ještě názorněji: nalezení označeného vrcholu v množině hran znamená nalezení řetězu jistého tvaru. TS však může hledat pouze konkrétní páskový symbol, tedy např. 1. symbol hledaného řetězu. Najde-li jej, musí se čtecí hlava vracet na výchozí místo pásky pro informaci o 2. symbolu řetězu, s touto informací se pak zase vrací za nalezený 1. symbol kvůli prověření 2. symbolu, atd. Nalezení konkrétního

podřetězu délky k ve slově délky n tak znamená provedení nejméně $2kn$ výpočetních kroků TS.

Není však těžké ověřit, že každý krok popisovaný v bodech 1) a 2) lze provést pomocí nejvýše polynomického počtu kroků výpočtu Turingova stroje T. TS T má tedy skutečně polynomickou časovou složitost.

Uvedený příklad celkem dobře demonstruje dvě přirozené fáze obvyklého způsobu řešení problému z třídy NP:

1) nedeterministický výběr konkrétní varianty přicházející v úvahu jako možné řešení,
2) deterministické prověření, jestli má zvolená varianta řešení ANO.

Obě fáze musí být pochopitelně proveditelné v nejvýše polynomickém čase.

Třída NP a nejvýše polynomický čas pro ověření správnosti řešení

Poznámka 3.13. Pro ověření příslušnosti konkrétního problému k třídě NP dokonce stačí nalézt TS (či algoritmus), který pro libovolný případ daného problému a libovolný „návrh řešení“ dokáže v polynomickém čase prověřit správnost tohoto řešení.

Namísto nedeterministického algoritmu hledajícího konkrétní variantu řešení a následně prověřujícího správnost této varianty se tak dostáváme k deterministickému algoritmu ověřujícímu (v polynomickém čase) pouze správnost předložené varianty řešení.

To, že v obou případech dostaneme stejnou třídu problémů (a sice NP), je důsledkem následujících úvah.

1) Existuje-li k danému problému NTS, který v polynomickém čase nedeterministicky vybírá konkrétní variantu řešení, a pak deterministicky prověřuje její správnost, jistě k témuž problému a konkrétní variantě řešení existuje TS, který v polynomickém čase prověří správnost tohoto řešení.

2) Jestliže k danému problému $\mathcal{P}$ existuje TS T s polynomickou časovou složitostí $p(n)$, jenž pro každý případ problému $\mathcal{P}$ a pro každý návrh řešení ověří jeho správnost, pak pro potvrzení existence NTS T', který řeší v polynomickém čase tentýž problém včetně vyčerpávajícího výběru variant, je důležité zodpovědět otázku, jak T' dokáže v polynomickém čase projít všechny varianty každého případu problému $\mathcal{P}$. Zde je důležité zjištění, že relevantní popis žádného návrhu řešení, s nímž pracuje TS T v polynomickém čase(!), nemůže mít větší délku než $p(n)$. Počet všech existujících návrhů řešení zapsaných v podobě řetězů (nad k-prvkovou páskovou abecedou stroje T) je tak shora omezen číslem $k^{p(n)}$. NTS T' tak má za úkol při výběru varianty řešení projít nejvýše $k^{p(n)}$ řetězů délky nejvýše $p(n)$ – vzhledem k nedeterministickému výběru

   - 1. až k-tého znaku v 1. kroku,
   - 1. až k-tého znaku ve 2. kroku,
   - $\vdots$
   - 1. až k-tého znaku v $p(n)$-tém kroku

   máme zaručeno, že nejpozději v $p(n)$-tém výpočetním kroku má stroj T' každou variantu řešení k dispozici.

Věta 3.14. *$P \subseteq NP$.*

Důkaz. Jde o přímý důsledek definic 3.7 a 3.9, neboť každý (deterministický) TS lze chápat jako speciální případ nedeterministického Turingova stroje, a sice stroje, jehož přechodová funkce každé dvojici (stav, páskový symbol) přiřazuje množinu s jediným prvkem tvořeným trojicí (stav, páskový symbol, posun čtecí hlavy). □

Poznámka 3.15. Inkluze z věty 3.14 de facto připouští dvě možnosti vzájemného vztahu porovnávaných tříd:

1) P $\subset$ NP,
2) P $=$ NP.

Průvodce studiem

Zdůrazněme, že případ P $\subset$ NP značí eventualitu, že každý problém z třídy P je také z třídy NP a že existuje nějaký problém z třídy NP, který však neleží v P.

Snaha zpřesnit tvrzení věty 3.14 důkazem některého z bodů poznámky 3.15 je jen o pár let mladší než definice tříd P a NP. Dosud marná snaha o takové zpřesnění, tj. zda P $=$ NP nebo P $\neq$ NP, je známá pod pojmem *P–NP problém*. Důležitost rozřešení tohoto problému plyne z toho, že je známa celá skupina v praxi se běžně vyskytujících úloh, jež patří do třídy NP a k nimž jsou známy pouze algoritmy pracující v exponenciálním čase. O těchto úlohách je známo, že patří mezi nejsložitější úlohy třídy NP. (Jde o tzv. NP-úplné problémy, jimž bude věnována následující kapitola.) Pokud by bylo dokázáno, že P $\neq$ NP, měli bychom jistotu, že k žádné z takových úloh neexistuje algoritmus řešící ji v polynomickém čase.

P–NP problém

Průvodce studiem

P–NP problém patří mezi nejdůležitější nerozřešené problémy současné matematiky. Mnozí matematikové se dokonce domnívají, že jej v užívaných logických systémech rozřešit ani nelze.

Pokud by byla udělována Nobelova cena za matematiku, pak si buďte jisti, že by neminula toho, kdo P–NP problém rozřeší.

Přestože nemáme žádnou jistotu, všeobecně převládá přesvědčení, že P $\neq$ NP, a tedy P $\subset$ NP.

Již jsme se zmínili o problémech, jež patří do třídy NP a k nimž známe (deterministické) algoritmy s exponenciální časovou složitostí. To, že exponenciální časová složitost je pro každý problém z NP dosažitelnou hranicí, je přímým důsledkem následující věty.

Věta 3.16. *Nechť L je jazyk přijímaný nedeterministickým Turingovým strojem s časovou složitostí $p(n)$. Potom existuje TS s časovou složitostí $t(n) = 2^{O(p(n))}$, který přijímá jazyk L.*

Důkaz. Popíšeme hlavní ideu důkazu. Převedení této ideje do podoby konkrétního TS přenecháme čtenářům k rozmyšlení (i když naznačíme možné směry takové konkretizace).

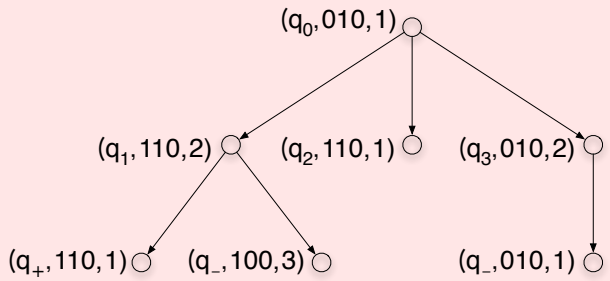
Výpočet každého NTS T pro dané vstupní slovo w lze názorně demonstrovat pomocí tzv. *stromu konfigurací* T_w, u nějž

Strom konfigurací NTS

1) každý vrchol je označen konfigurací NTS T,
2) kořen stromu je označen počáteční konfigurací NTS T pro slovo w (tj. konfigurací $(q_0, w, 1)$),
3) každý vrchol v je označen konfigurací (q, x, m) a všichni přímí následníci vrcholu v jsou označeni konfiguracemi $(q_1, x_1, m_1), \ldots, (q_j, x_j, m_j)$, právě když $(q, x, m) \vdash (q_i, x_i, m_i)$ pro všechna $i \in \{1, \ldots, j\}$.

Průvodce studiem

Níže je uveden příklad stromu konfigurací NTS T pro vstupní slovo 010.



Uvedený strom zohledňuje následující přechody přechodové funkce stroje T:

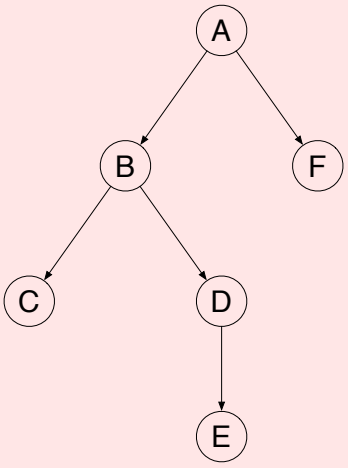
$\delta(q_0, 0) = \{(q_1, 1, P), (q_2, 1, L), (q_3, 0, P)\}$,
$\delta(q_1, 1) = \{(q_+, 1, L), (q_-, 0, P)\}$,
$\delta(q_2, 1) = \emptyset$,
$\delta(q_3, 1) = \{(q_-, 1, L)\}$.

Nepřehlédněte skutečnost, že v obecném případě může mít strom konfigurací i nekonečný počet vrcholů.

Pokud chceme simulovat činnost libovolného NTS T pro vstupní slovo w deterministickým Turingovým strojem T', pak stačí aby T' simuloval průchod stromem konfigurací T_w do šířky.

Průvodce studiem

Pro připomenutí, při prohledávání grafu (z výchozího vrcholu v_0) do šířky jsou nejdříve „navštíveny“ vrcholy dosažitelné z výchozího vrcholu cestou délky 1, pak délky 2, atd. Pořadí navštívených vrcholů v níže uvedeném grafu by tedy bylo: A, B, F, C, D, E.



Označíme-li

$k = \max_{q,x} \{\text{počet všech přechodů pro stav } q \text{ a páskový symbol } x \text{ v NTS } T\}$,

potom strom konfigurací T_w zahrnuje řádově nejvýše k^p výpočetních větví NTS T délky maximálně p, tj. větví délky $1, \ldots, p$.

Průvodce studiem

Řádovou hodnotu k^p získáte jako součet konečné geometrické řady $\sum_{i=1}^{p} k^i$, poněvadž v uvedeném stromu konfigurací je nejvýše k cest délky 1, k^2 cest délky 2, ..., k^p cest délky p. Konkrétně máme $\sum_{i=1}^{p} k^i = k \cdot \frac{k^p - 1}{k-1} = \frac{k}{k-1} \cdot (k^p - 1) < K \cdot k^p$, kde konstanta $K = \frac{k}{k-1}$.

Jestliže NTS T s časovou složitostí $p(n)$ přijímá vstupní slovo w délky n, pak podle definice časové složitosti NTS má strom konfigurací T_w hloubku maximálně $p \leq p(n)$ a TS T' simulující stroj T přijímá slovo w nejpozději v $K \cdot k^{p(n)} \cdot O(p(n))$ krocích výpočtu.

Průvodce studiem

Jak získáte uvedený počet kroků? $K \cdot k^{p(n)}$ udává maximální počet všech simulovaných cest délky nejvýše $p(n)$ v stromu konfigurací a $O(p(n))$ popisuje řádovou složitost simulace jedné výpočetní větve délky $p(n)$ strojem T'.

Pro časovou složitost $t(n)$ Turingova stroje T' tedy platí:
$t(n) = K \cdot k^{p(n)} \cdot O(p(n)) = (2^{\log_2 k})^{p(n)} \cdot K \cdot O(p(n)) = 2^{p(n) \cdot \log_2 k} \cdot O(p(n)) =$
$2^{O(p(n))} \cdot 2^{O(p(n))} = 2^{O(p(n))+O(p(n))} = 2^{O(p(n))}$. □

Shrnutí

Řešitelné problémy členíme do tříd složitostí, tj. problémy ze stejné třídy složitosti se vyznačují existencí algoritmů řešících je nejhůře s časovou (či paměťovou) složitostí příslušnou dané třídě. Ze známých a studovaných tříd složitostí jsme si popsali dvě následující třídy.

1) Třída P (někdy označovaná jako PTIME) obsahuje každý problém, který lze řešit Turingovým strojem (algoritmem) s polynomickou časovou složitostí.
2) Třída NP (někdy označovaná jako NPTIME) obsahuje každý problém, který lze řešit nedeterministickým Turingovým strojem (nedeterministickým algoritmem) s polynomickou časovou složitostí. Zde NTS řešící nějaký problém znamená stroj s konečnými výpočetními větvemi pro libovolné vstupní slovo, který přijímá všechna vstupní slova představující zakódování těch případů daného problému, jež mají řešení ANO.

Z definic tříd P a NP plyne, že P $\subseteq$ NP. Není však známo, zda P $=$ NP anebo P $\neq$ NP. Tato nerozřešená úloha bývá nazývána P–NP problémem.

Pojmy k zapamatování

- Třída P,
- třída NP,
- P–NP problém.

Kontrolní otázky

1. *Je dokázáno, že P $\subset$ NP?*
2. *Existuje ke každému problému z třídy NP algoritmus, který jej řeší nejhůře v exponenciálním čase?*

Cvičení

1. Dokažte, že problém generování neprázdného jazyka zadanou bezkontextovou gramatikou je z třídy P.

Úkoly k textu

1. Zdůvodněte každou rovnost ze závěru důkazu věty 3.16, tj.
$t(n) = K \cdot k^{p(n)} \cdot O(p(n)) = (2^{\log_2 k})^{p(n)} \cdot K \cdot O(p(n)) = 2^{p(n) \cdot \log_2 k} \cdot O(p(n)) =$
$2^{O(p(n))} \cdot 2^{O(p(n))} = 2^{O(p(n))+O(p(n))} = 2^{O(p(n))}$.

Řešení

1. K požadovanému důkazu stačí sestavit TS s polynomickou časovou složitostí, který rozhoduje jazyk $\{\langle G\rangle;\ G$ je bezkontextová gramatika generující neprázdný jazyk$\}$. Uvedený požadavek splňuje následující TS.

   TS T pro vstupní slovo $\langle G\rangle$, kde G je bezkontextová gramatika:
   1) Označí ε a každý terminál gramatiky G.
   2) Dokud označuje nové neterminály, provádí následující činnost.
      - Označí neterminál z levé strany každého pravidla, jehož pravá strana je tvořena řetězem označených symbolů.
   3) Jestliže je označen počáteční symbol gramatiky G, TS T přijímá vstupní slovo $\langle G\rangle$; jinak zamítá $\langle G\rangle$.

   Polynomickou časovou složitost stroje T lze prověřit stejným postupem jako v příkladu 3.12.

3.3 NP-úplné problémy

Studijní cíle: Po prostudování kapitoly bude studující umět definovat NP-úplné jazyky i NP-úplné problémy a bude znát několik jejich typických představitelů. Rovněž bude schopen popsat jejich význam pro řešení P–NP problému.

Klíčová slova: NP-úplné problémy, problém splnitelnosti booleovských formulí, problém kliky, problém obarvitelnosti grafu třemi barvami.

Potřebný čas: 60 minut.

Pro účely řešení P–NP problému by bylo užitečné popsat nejsložitější problémy z třídy NP. Pokud by takové problémy ležely všechny v třídě P, dalo by se očekávat, že v P budou i ostatní (méně komplikované) problémy z NP. V takovém případě by platila rovnost mezi třídami P a NP. Na druhou stranu, jestliže by byť jen jediný problém z NP neležel v třídě P, pochopitelně by P a NP představovaly rozdílné třídy složitostí.

To, že by příslušnost nejkomplikovanějších problémů z NP k třídě P měla znamenat stejnou vlastnost pro všechny problémy z NP, je ovšem nutno umět dokázat. Příslušný důkaz bude snadný, pokud šikovným způsobem definujeme převoditelnost jedněch problémů na problémy jiné.

Průvodce studiem

Pro lepší pochopení principu převoditelnosti problémů si představte například následující situaci. Toužíte navštívit během jednoho měsíce exotickou zemi, se kterou však náš stát nemá uzavřenou dohodu o bezvízovém styku. Víte, že vystavení turistického víza do této země trvá šest týdnů a vystavení „služebního víza“ určeného pro služební cesty trvá pouze týden. Na služební cestu vás přitom musí vyslat dostatečně důvěryhodný zaměstnavatel jako je např. redakce novin nebo časopisu, velký strojírenský podnik, velkoobchod (s exotickým ovocem), apod. Stojíte tedy před problémem získání víza a tento problém můžete „převést“ na problém vyslání na služební cestu. Původní problém tak můžete vyřešit, umíte-li vyřešit nově zformulovaný problém (např. uzavřením dohody s redakcí novin o napsání reportáže z navštívené země). Pokud navíc problém vyslání na služební cestu vyřešíte v potřebném čase (tří týdnů), dokážete vyřešit problém získání víza (do jednoho měsíce).

Všimněte si, že problém, na nějž se vám podařilo převést původní problém, vůbec nemusí být jednodušší – zkuste získat zaměstnání ve velkém podniku a dosáhnout vyslání na služební cestu do vysněné země, a to vše do tří týdnů.

Pokud hodláme popsat převod jednoho problému na jiný a tento převod by měl fungovat obecně, měli bychom jej popsat pomocí algoritmu (nebo formálně pomocí necyklícího Turingova stroje). Jestliže navíc pro řešitelnost obou problémů vyžadujeme nejvýše polynomický čas, neměl by mít příslušný algoritmus (čili TS) horší časovou složitost.

Algoritmus převodu jednoho problému na jiný

Formálně lze takový převod vyjádřit také pomocí jistého typu funkce, a sice funkce zobrazující řetězy nad určitou abecedou Σ na řetězy nad toutéž abecedou. Pokud každý případ studovaného problému zakódujeme do podoby konkrétního řetězu nad abecedou Σ, uvedená funkce takovému řetězu přiřadí jako výsledek jiný řetěz nad Σ, který bude představovat zakódování případu problému, na nějž je ten původní převáděn.

Uvedené ideje objasňují důvody pro zavedení následujících pojmů.

Definice 3.17. Nechť Σ je libovolná abeceda. Funkci $f : \Sigma^* \to \Sigma^*$ nazveme *vyčíslitelnou v polynomickém čase*, právě když existuje TS s polynomickou časovou složitostí, který pro každý řetěz $w \in \Sigma^*$ zastaví s tím, že na jeho pásce zůstane zapsán řetěz $f(w)$.

Vyčíslitelná funkce = funkce vyčíslovaná nějakým TS

Definice 3.18. Řekneme, že jazyk $L_1 \subseteq \Sigma^*$ je *převoditelný*[18] na jazyk $L_2 \subseteq \Sigma^*$ *v polynomickém čase*, právě když existuje funkce $f : \Sigma^* \to \Sigma^*$ vyčíslitelná v polynomickém čase taková, že pro každý řetěz $w \in \Sigma^*$ platí: $w \in L_1$, právě když $f(w) \in L_2$.

Píšeme pak $L_1 \lhd_p L_2$ (nebo $L_1 \leq_p L_2$). O funkci f říkáme, že *převádí jazyk L_1 na L_2 v polynomickém čase*.

Poznámka 3.19. Stejně jako v kapitole 3.2 můžeme poslední definici přeformulovat i pro definici převoditelnosti jednoho problému na jiný v polynomickém čase. Stačí opět použít větu 2.24 dávající do souvislosti řešitelné problémy a rekurzivní jazyky. Jestliže je tedy jazyk „příslušný“ k problému P_1 převoditelný na jazyk „příslušný“ k problému P_2 v polynomickém čase, říkáme, že *problém P_1 je převoditelný na problém P_2 v polynomickém čase*.

Vlastnosti problémů ∼ vlastnosti odpovídajících jazyků

Průvodce studiem

Jestliže si podrobně promyslíte, kdy je vlastně problém P_1 převoditelný na problém P_2, pak zjistíte, že

1) každý případ problému P_1 s řešením ANO musí být převeden na případ problému P_2 s řešením ANO,
2) každý případ problému P_1 s řešením NE musí být převeden na případ problému P_2 s řešením NE.

V našich definicích a tvrzeních se neustále opakuje u převoditelnosti dovětek „v polynomickém čase“. Je to nutné, neboť převoditelnost může být (a bývá) definována i v exponenciálním čase, logaritmické paměti, polynomické paměti, apod.

Existují různé typy převoditelností.

Lemma 3.20. *Převoditelnost $\lhd_p$ je tranzitivní, tj. ze vztahů $L_1 \lhd_p L_2$ a $L_2 \lhd_p L_3$ plyne $L_1 \lhd_p L_3$.*

Důkaz. Tvrzení lemmatu plyne přímo z definice převoditelnosti jednoho jazyka na druhý v polynomickém čase. □

Průvodce studiem

Tranzitivnost převoditelnosti problémů (nebo příslušných jazyků) si můžete snadno představit na příkladu z předminulého průvodce studiem. V něm jsme popisovali převoditelnost

[18] V literatuře je sice používanější termín redukovatelný, ale v češtině tento termín vzbuzuje nejčastěji představu, že se jedná o zjednodušení výchozího problému. Taková představa je ovšem v rozporu se skutečností, protože v souladu s definicí jsou například převody řešitelných problémů na neřešitelné, ale nikdy ne obráceně.

problému získání víza na problém vyslání na služební cestu. Problém „vyslání ...“ lze zřejmě dále převést na problém přesvědčit šéfredaktora konkrétních novin o svých literárních kvalitách. Jistě budete souhlasit s tím, že úspěšné přesvědčení šéfredaktora (během tří týdnů) povede k vyřešení problému získání víza.

Následující pomocné tvrzení budeme potřebovat v důkazu věty popisující řešení P–NP problému v případě splnění jisté podmínky týkající se nejsložitějších problémů z třídy NP.

Lemma 3.21. *Nechť* $L_1, L_2 \subseteq \Sigma^*$, $L_1 \lhd_p L_2$. *Jestliže* $L_2 \in P$, *pak* $L_1 \in P$.

Důkaz. Protože $L_2 \in P$, existuje TS T s polynomickou časovou složitostí, který rozhoduje jazyk L_2. Nechť funkce f převádí jazyk L_1 na L_2 v polynomickém čase. Sestavíme TS S s polynomickou časovou složitostí, který rozhoduje jazyk L_1.

TS S pro vstupní slovo w:

1) Spočítá $f(w)$ (simulací Turingova stroje vyčíslujícího v polynomickém čase funkci f).
2) Simuluje činnost Turingova stroje T pro vstupní slovo $f(w)$.
3) Jestliže simulovaný TS T $\begin{cases} \text{přijímá } f(w), \text{ pak TS } S \text{ přijme } w, \\ \text{zamítá } f(w), \text{ pak TS } S \text{ zamítne } w. \end{cases}$

Průvodce studiem

Kdyby vás náhodou trápila myšlenka, co učiní TS S v případě, že by TS T začal cyklit, pak si připomeňte úvodní zmínku o TS T, podle níž T rozhoduje jazyk L_2 (tj. T nikdy necyklí, každé slovo z L_2 přijme a každé slovo neležící v L_2 zamítne).

Polynomická časová složitost Turingova stroje S plyne z polynomických časových složitostí bodů 1) a 2). □

Definice 3.22. O jazyku L řekneme, že je *NP-úplný*, právě když jsou splněny následující podmínky.

1) $L \in$ NP.
2) Pro každý jazyk $L' \in$ NP platí: $L' \lhd_p L$.

Alternativně pro problémy:

Definice 3.23. O problému P řekneme, že je *NP-úplný*, právě když jsou splněny následující podmínky.

1) $P \in$ NP.
2) Pro každý problém $P' \in$ NP platí: $P' \lhd_p P$.

Průvodce studiem

NP neobsahuje složitější problémy než jsou NP-úplné.

Jak vyplývá z předchozího textu, každý NP-úplný problém se řadí mezi nejsložitější problémy z třídy NP. Tato vlastnost je důsledkem převoditelnosti libovolného problému z NP na NP-úplný problém v polynomickém čase. Tzn. máme-li být schopni na NP-úplný problém převést v polynomickém čase sebesložitější problém z NP, nemůže jít o jednoduchý problém.

V jistém smyslu tak každý NP-úplný problém P reprezentuje celou třídu NP. Nalezneme-li pro něj totiž konkrétní řešení, může být toto řešení použito spolu s algoritmem, převádějícím v polynomickém čase libovolný problém P' z NP na P, pro řešení problému P' – tato idea byla ostatně použita i v důkazu lemmatu 3.21.

Důkaz NP-úplnosti nějakého jazyka (či problému) může být proveden buď přímým prověřením obou podmínek požadovaných v definici anebo pomocí následující věty (uvedeme pouze variantu pro jazyky stejně jako u další věty – formulace variant pro problémy by čtenáři již neměly činit potíže).

Věta 3.24. *Nechť $L_1, L_2 \in NP$, $L_1 \lhd_p L_2$. Jestliže je L_1 NP-úplný jazyk, pak je L_2 také NP-úplný jazyk.*

Důkaz. Tvrzení věty plyne přímo z definice 3.22 a z tranzitivnosti převoditelnosti $\lhd_p$ (Viz lemma 3.20). □

Věta 3.25. *Jestliže existuje NP-úplný jazyk ležící v třídě P, pak platí rovnost P = NP.*

NP-úplnost a P–NP problém

Důkaz. Inkluze P $\subseteq$ NP je popsána již ve větě 3.14. Pro důkaz rovnosti nám stačí ověřit platnost inkluze NP $\subseteq$ P:

Nechť $L \in$ P je NP-úplný jazyk. Uvážíme-li libovolný jazyk $L' \in$ NP, pak $L' \lhd_p L$ podle definice NP-úplnosti jazyka L. Vzhledem k předpokladu $L \in$ P a lemmatu 3.21 tak máme $L' \in$ P, a tedy NP $\subseteq$ P. □

Až dosud byl hlavní důraz kladen na bližší vymezení vzájemného vztahu tříd P a NP. Nemělo by to však vést k představě, že NP-úplné problémy představují jen teoretický konstrukt, který nemá pro reálnou praxi žádný význam. Protože nikdo dosud nedokázal rovnost P = NP, není známa existence žádného algoritmu řešícího kterýkoliv NP-úplný problém v polynomickém čase. Odtud plyne, že každý programátor postavený před úkol nalézt dostatečně rychlý algoritmus řešící konkrétní NP-úplný problém

Programátor a NP-úplné problémy

a) má mizivou naději nalézt algoritmus řešící rychle všechny případy daného problému,
b) může snadno najít algoritmus řešící rychle některé případy daného problému a velmi pomalu (konkrétně v exponenciálním čase) případy ostatní.

Neuspokojují-li programátora výše nabízené varianty, má už jen následující možnosti:

1) pokusit se změnit zadání úkolu takovým způsobem, aby neobsahoval požadavky vedoucí k tak vysoké výpočetní náročnosti; jinými slovy modifikovat problém tak, aby nepatřil mezi NP-úplné, ale „pouze“ do třídy P,[19]
2) rezignovat na řešení zadaného úkolu.

3.3.1 Vybrané NP-úplné problémy

Jsou známy stovky NP-úplných problémů – my si uvedeme jen pár z nich. Náš krátký výčet zahájíme problémem splnitelnosti logických formulí, jehož NP-úplnost byla dokázána jako první.

Nejdříve si ovšem připomeňme několik nezbytných pojmů.

Definice 3.26. Množinu všech *logických formulí* definujeme rekurzivně:

- $0, 1, x_1, x_2, \ldots$ (kde $x_1, x_2, \ldots$ jsou logické proměnné, tj. proměnné, jež mohou nabývat hodnot 0 nebo 1) jsou logické formule,
- Jestliže E_1, E_2 jsou logické formule, pak také $\neg E_1, (E_1 \wedge E_2)$ a $(E_1 \vee E_2)$ jsou logické formule (symboly $\neg$, $\wedge$ a $\vee$ zde představují známé logické operace negace, konjunkce a disjunkce).

[19] K takovým úpravám je velice užitečná znalost konkrétních problémů: jednak těch, co prokazatelně patří do třídy P, jednak NP-úplných – v některých oblastech mezi nimi vede zdánlivě tenká hranice. Více informací lze nalézt například v [Gar79].

Poznámka 3.27. Vzhledem k asociativitě logických operací konjunkce a disjunkce vynecháváme některé páry závorek. Například místo $((E_1 \wedge E_2) \wedge E_3)$ píšeme $(E_1 \wedge E_2 \wedge E_3)$. Dále obvykle vynecháváme vnější páry závorek, tj. místo formule $\mathcal{F} = (E_1 \wedge E_2)$ píšeme $\mathcal{F} = E_1 \wedge E_2$.

Příklad 3.28. Uveďme dva příklady logických formulí

1) $\mathcal{F}_1 = (\neg x_1 \wedge x_2) \vee x_1$,
2) $\mathcal{F}_2 = (x_1 \vee x_2 \vee \neg x_1) \wedge (\neg x_3 \vee x_2)$.

Definice 3.29. Logickou formuli $\mathcal{F}$ nazveme *splnitelnou*, právě když existuje takové dosazení 0 a 1 za její proměnné, při němž je hodnota $\mathcal{F}$ rovna 1.

Příklad 3.30. $\mathcal{F}_1 = (x_1 \vee x_2) \wedge \neg x_1$ je podle definice splnitelná logická formule, neboť stačí dosadit $x_1 = 0$ a $x_2 = 1$.

Problém splnitelnosti logických formulí je zadán následovně:

Vstup: Logická formule.
Otázka: Je zadaná logická formule splnitelná?

Věta 3.31. (Cookova[20] věta) *Problém splnitelnosti logických formulí je NP-úplný.*

Průvodce studiem

Málokterý důkaz NP-úplnosti nějakého problému je krátký a jednoduchý. Nadále budou proto všechny důkazy vynechávány. Zajímají-li vás přesto, naleznete je v knihách [Sip97], [Gru97], [Gar79] a většinu také v [Kuč89].

Definice 3.32. Říkáme, že logická formule $\mathcal{F}$ je *v konjunktivním normálním tvaru* (zkráceně píšeme v *CNF*), právě když $\mathcal{F} = \bigwedge_{1 \leq i \leq m} \mathcal{F}_i$, kde každá podformule $\mathcal{F}_i$ je disjunkcí

- logických proměnných,
- negací logických proměnných.

Příklad 3.33. $\mathcal{F} = (x_1 \vee x_2 \vee \neg x_3) \wedge x_2 \wedge (\neg x_1 \vee \neg x_2)$ je podle definice logická formule v konjunktivním normálním tvaru.

Průvodce studiem

Nabízí se myšlenka, jestli nebude výpočetně snazší řešit problém splnitelnosti logických formulí v případě formulí v konjunktivním normálním tvaru – oproti obecně zadaným formulím mají jistý stupeň vnitřní uspořádanosti, který by možná mohl „vymanit náš problém z NP-úplného sevření“. Myšlenka je to sice hezká, ale následující věta ji vyvrací.

Věta 3.34. *Problém splnitelnosti logických formulí v konjunktivním normálním tvaru je NP-úplný.*

[20] S. A. Cook uvedenou větu dokázal v roce 1971.

Definice 3.35. Píšeme, že logická formule $\mathcal{F}$ je v *k-CNF* (kde $k \in \mathbb{N}$), právě když $\mathcal{F} = \bigwedge_{1 \leq i \leq m} \mathcal{F}_i$, kde každá podformule $\mathcal{F}_i$ je disjunkcí právě k logických proměnných nebo jejich negací.

Příklad 3.36. $\mathcal{F} = (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4)$ je podle definice logická formule v 3-CNF.

Průvodce studiem

Myšlenka (nastíněná v minulém průvodci studiem) o snadnějším řešení problému splnitelnosti logických formulí v k-CNF se vtírá mnohem neodbytněji – „stupeň vnitřní uspořádanosti" formulí v k-CNF je nepochybně ještě vyšší než u formulí v CNF.

V případě formulí v 1-CNF je výpočetní náročnost pouze polynomická, neboť u každé takové formule stačí prověřit, že se v ní žádná z jejích proměnných nevyskytuje bez negace i s negací, a tato kontrola zabere řádově nejvýše $j \cdot n$ výpočetních kroků, kde n je délka zakódování formule a j udává počet jejích proměnných.

Odpověď na otázku, zda je pro nějaké k problém splnitelnosti logických formulí v k-CNF NP-úplným problémem, dává následující věta.

Věta 3.37. *Problém splnitelnosti logických formulí v* 3*-CNF je NP-úplný.*

Poznámka 3.38. O problému splnitelnosti logických formulí v 2-CNF se dá dokázat, že je z třídy P. (Viz např. [Lew98])

Dále vybíráme NP-úplné problémy z oblasti teorie grafů.

Problém kliky je zadán následovně:

Vstup: Neorientovaný graf a číslo $k \in \mathbb{N}$.

Otázka: Obsahuje zadaný graf kliku o k vrcholech?

Věta 3.39. *Problém kliky je NP-úplný.*

Definice 3.40. O neorientovaném grafu řekneme, že je *obarvitelný k barvami* (kde $k \in \mathbb{N}$), právě když lze každému jeho vrcholu přiřadit jednu z k barev tak, aby žádná hrana grafu nespojovala vrcholy se stejnou barvou.

Problém obarvitelnosti grafu k barvami je zadán následovně:

Vstup: Neorientovaný graf a číslo $k \in \mathbb{N}$.

Otázka: Je zadaný graf obarvitelný k barvami?

Věta 3.41. *Problém obarvitelnosti grafu* 3 *barvami je NP-úplný.*

Náš kratičký výčet uzavřeme jedním z nejznámějších NP-úplných problémů.

Problém obchodního cestujícího je zadán následovně:

Vstup: Nezáporné celé číslo K, množina měst a vzdáleností mezi nimi.

Otázka: Může obchodní cestující projet všechna města tak, aby každé navštívil právě jednou, na závěr se vrátil do výchozího místa a přitom urazil vzdálenost menší nebo rovnu číslu K?

Průvodce studiem

V terminologii teorie grafů lze problém obchodního cestujícího formulovat následovně: Existuje v úplném neorientovaném grafu s ohodnocenými hranami hamiltonovská kružnice, u níž je součet ohodnocení jejích hran menší nebo roven předem dané hodnotě K?

Shrnutí

Funkcí vyčíslitelnou v polynomickém čase rozumíme každou funkci f splňující následující podmínky.

- f zobrazuje množinu všech řetězů nad nějakou abecedou Σ do sebe, tzn. pro každý řetěz $w \in \Sigma^*$ je $f(w)$ rovno nějakému řetězu nad Σ.
- Existuje TS s polynomickou časovou složitostí, jehož výpočet pro každé vstupní slovo $w \in \Sigma^*$ skončí buď v přijímajícím anebo v zamítajícím stavu, přičemž po skončení výpočtu je na pásce TS zapsáno slovo $f(w)$ (následované nekonečně mnoha prázdnými znaky).

Převoditelnost jazyka L_1 na jazyk L_2 v polynomickém čase definujeme pomocí funkce f vyčíslitelné v polynomickém čase, přičemž požadujeme, aby $\begin{cases} f(w) \in L_2 \text{ pro každé } w \in L_1, \\ f(w) \notin L_2 \text{ pro každé } w \notin L_1. \end{cases}$

NP-úplný jazyk musí patřit do třídy NP a každý jazyk z NP na něj musí být převoditelný v polynomickém čase.

Vlastnosti definované pro jazyky „přenášíme“ na problémy pomocí věty 2.24. Konkrétněji: Vlastnost X přiřkneme každému problému, jemuž „příslušný“ jazyk má vlastnost X.

K žádnému NP-úplnému problému není znám algoritmus řešící jej v polynomickém čase. Pokud by někdo dokázal příslušnost některého NP-úplného problému k třídě P, dokázal by rovnost P=NP.

Pojmy k zapamatování

- Funkce vyčíslitelná v polynomickém čase,
- převoditelnost jednoho jazyka (problému) na jiný v polynomickém čase,
- NP-úplný jazyk (problém).

Kontrolní otázky

1. *Je každý NP-úplný problém řešitelný?*
2. *Je každý NP-úplný problém převoditelný v polynomickém čase na libovolný jiný NP-úplný problém?*

Cvičení

1. Zjistěte, jaké důsledky pro řešení P-NP problému by měla existence NP-úplného problému, který by byl převoditelný v polynomickém čase na libovolný problém z třídy P.

Úkoly k textu

1. Dokažte detailně lemma 3.20.
2. Promyslete, proč je v předpokladech věty 3.24 požadavek, aby $L_2 \in$ NP.
3. Pokud byste se podívali do odborné literatury, častěji byste nalezli problém obchodního cestujícího ve variantě tázající se po minimální uražené vzdálenosti namísto porovnávání se zadanou hodnotou K. Rozmyslete si, jak lze uvedenou variantu převést na námi formulovaný problém obchodního cestujícího a naopak.

Řešení

1. V případě existence takového NP-úplného problému by platila rovnost P=NP. Zdůvodnění:

   Pokud $L' \in$ P a L je NP-úplný jazyk, který je převoditelný v polynomickém čase na L', pak podle lemmatu 3.21 leží jazyk L v třídě P. Tím je ovšem splněn předpoklad věty 3.25, podle níž v takovém případě nastává rovnost P=NP.

3.4 Využití výpočetně obtížných úloh

Studijní cíle: Po prostudování kapitoly bude studující umět zdůvodnit důležitost studia výpočetně obtížných úloh pro použití v praxi.

Klíčová slova: Šifrování, metoda RSA.

Potřebný čas: 5 minut.

Znalost úloh, které jsou výpočetně náročné, nepochybně ocení každý, kdo hledá a programuje řešení nejrůznějších úloh – v rámci možností se pak může snažit vyhnout zadáním, jež vedou ke známým, výpočetně obtížným úlohám; případně může obrátit pozornost k algoritmům, které nevedou pokaždé a bezvýhradně k požadovanému cíli (mezi ně řadíme stochastické, heuristické, genetické aj. algoritmy).

V právě uvedené oblasti je výskyt výpočetně obtížné úlohy vnímán obvykle jako nepříjemnost. Existuje nějaká oblast, kde je tomu naopak? Ano - jde o šifrování. Při utajené komunikaci je prvořadý požadavek bezpečnosti takové komunikace; jinými slovy požadavek „nemožnosti“ nebo alespoň enormní časové náročnosti odhalení klíče potřebného k dešifrování zašifrované zprávy. Jestliže systém tvorby klíče propojíme s některým ze známých obtížně řešitelných problémů, pak úkol dešifrovat odeslanou zprávu bez znalosti příslušného klíče odpovídá úloze nalézt konkrétní řešení souvisejícího problému. Víme-li, že v současnosti nikdo takový problém efektivně řešit neumí a uvedený problém je výpočetně náročný ve většině svých případů, pak jsme pro utajení zprávy učinili maximum.

Jen krátce k utajené komunikaci.

V zásadě lze utajení rozdělit na dvě části: jednak utajený přenos[21] (tj. zpráva by se neměla dostat nikomu nepovolanému do rukou), jednak utajení vlastního sdělení (tj. nikdo cizí by zprávu neměl umět přečíst). Optimální je pochopitelně spojení obou technik. Jsou ovšem situace, kdy je výhodné zveřejnit záměr vést utajenou komunikaci a komunikační cestu neskrývat. Za přesvědčivý příklad může například sloužit internetový obchod, který chce umožnit zákazníkům platit kreditní kartou. Bez utajení údajů své kreditní karty by však jistě žádný soudný zákazník tímto způsobem za zboží neplatil.

Průvodce studiem

Také vám připadá nepochopitelné, kolik lidí uvěří své šťastné hvězdě, jež nasměruje právě k nim závratně bohatého afrického mecenáše toužícího podělit se s nimi o své milióny? Poskytnutí svého bankovního účtu takovému filantropovi k dispozici je pak jen zanedbatelnou protislužbou, není-liž pravda?

Za typický příklad využití výpočetně obtížné úlohy v šifrování lze považovat metodu RSA používanou při šifrování s veřejně přístupným klíčem.

[21] Zajímavou současnou metodou utajeného přenosu je ukrývání zpráv do obsáhlých datových souborů, např. do obrazových souborů. Víc informací se mohou zájemci dozvědět např. na Internetu pod klíčovým slovem „steganography“.

Idea šifrování s veřejně přístupným klíčem je následující. Jeden účastník utajené komunikace (např. ústředí banky) vygeneruje dvojici klíčů A_1 a A_2, mezi nimiž je určitá matematická závislost. Klíč A_1 dostačuje k zašifrovávání posílaných zpráv, klíč A_2 je potřebný k rozšifrovávání přijatých zpráv. Klíč A_2 utají, A_1 naproti tomu rozešle ostatním účastníkům utajené komunikace (např. bankovním pobočkám) prostřednictvím veřejně přístupného média, jako je třeba pevná telefonní linka. S pomocí klíče A_1 tedy může každý zašifrovat svou zprávu, ale její rozšifrování lze provést pouze s využitím klíče A_2, jenž je známý výhradně iniciátoru celé komunikace.

Metoda RSA[22] je založena na využití součinu velkých prvočísel. Poznamenejme, že testování prvočíselnosti zadaného čísla je úloha zvládnutelná v rozumném čase, naproti tomu úloha nalézt rozklad zadaného přirozeného čísla na součin prvočinitelů je výpočetně velmi náročná (a to nebyla dokázána ani její příslušnost k NP–úplným úlohám). Úkol rozšifrovat zasílanou zprávu tak odpovídá úkolu nalézt přijatelně rychlý algoritmus pro rozklad na součin prvočinitelů.

[22] Více informací o této metodě lze nalézt např. v [Gru97], [Sip97], nebo na Internetu.

Literatura

[Gar79] Garey M. R., Johnson D. S.: *Computers and Intractability. A Guide to the Theory of NP-Completeness* W. H. Freeman and Company, San Francisco, 1979.

[Gru97] Gruska J.: *Foundations of Computing.* Intern. Thomson Computer Press, Boston, 1997.

[Hop69] Hopcroft J. E., Ullman J. D.: *Formal Languages and Their Relation to Automata.* Addison-Wesley, Reading (Massachusetts), 1969.

[Chy82] Chytil M.: *Teorie automatů a formálních jazyků.* Skriptum UJEP v Brně, SPN, Praha, 1982.

[Kou98] Koubková A., Pavelka J.: *Úvod do teoretické informatiky.* MATFYZPRESS, Praha, 1998.

[Kuč89] Kučera L.: *Kombinatorické algoritmy.* SNTL, Praha, 1989.

[Lew98] Lewis H. R., Papadimitriou C. H.: *Elements of the Theory of Computation.* Prentice Hall, Upper Saddle River (NJ), 1998.

[Man81] Manna Z.: *Matematická teorie programů.* Překlad z angličtiny. SNTL, Praha, 1981.

[Mol87] Molnár L'., Češka M., Melichar B.: *Gramatiky a jazyky.* ALFA, Bratislava, 1987.

[Nov88] Novotný M.: *S algebrou od jazyka ke gramatice a zpět.* Academia, Praha, 1988.

[Rei83] Reiterman J.: *Analýza algoritmů.* Skriptum FJFI ČVUT v Praze, Ediční středisko ČVUT, Praha, 1983.

[Sal73] Salomaa A.: *Formal Languages.* Academic Press, New York, 1973.

[Sip97] Sipser M.: *Introduction to the Theory of Computation.* PWS Publishing Company, Boston, 1997.

Rejstřík