Turingovy stroje

David Stanovský

Každý z vás, kdo někdy zkoušel programovat, jistě narazil na to, že některé problémy jsou snadné a některé těžké. A určitě měl každý z vás už někdy pocit, že zrovna ta úloha, co teď řešíte, je neřešitelná. Patrně to nebyla pravda a řešení se našlo, ale na snadě je otázka: lze skutečně na každý problém (který je možné zadat konečně mnoha slovy) najít algoritmus? Asi pro vás bude překvapením, že odpověď je záporná. Kupodivu, nalezení takového problému není nijak zvlášť těžké. A i když se jedná o problém trošku umělý, pomocí něj se ukáže neřešitelnost mnoha praktických úloh. Za všechny jmenujme pro nás asi nejpotěšitelnější výsledek, že matematiky nikdy nenahradí počítač: na dokazování matematických vět totiž nelze sestrojit algoritmus.

Největší potíží, která na nás čeká hned na začátku, je přesné vymezení pojmu *algoritmus*. Pokud chceme o něčem dokazovat, že to neexistuje, musíme to mít napřed přesně definované — jinak nám to každý šťoural rychle naboří a dopadneme stejně jako filosofové. Ale co to je algoritmicky řešitelný problém? To co jde naprogramovat v Pascalu? V C++? V Assembleru? To co umí spočítat počítač na našem stole? Nebo nějaký superpočítač s gigantickou pamětí? Tudy od pohledu cesta nevede, je třeba vymyslet nějakou abstrakci.

Zkusme nějak popsat vlastnosti algoritmu. Algoritmus je popsán nějakým konečným kódem, který využívá jakousi (potenciálně nekonečnou) vnější paměť. Algoritmu zadáme (konečný) vstup, načež začne běžet na něm závislý výpočet popsaný kódem. Pokud se výpočet někdy zastaví, sdělí nám nějaký (konečný) výstup.

Na základě těchto vlastností se definuje *Turingův stroj*. Stroj má řídící jednotku, která se může nacházet v jednom z konečně mnoha předem definovaných stavů, nekonečnou pásku a čtecí hlavu, která se umí po pásce pohybovat, umí z pásky číst a umí na pásku psát předem definovaná písmena. Dále má (konečnou) sadu instrukcí, které říkají, co má stroj dělat, nachází-li se v daném stavu a přečetl dané písmeno. Následuje formální definice stroje jakožto matematického pojmu.

Definice. *Turingův stroj T je pětice $(Q, F, q_0, A, \mathcal{I})$, kde Q je konečná množina stavů, $F \subseteq Q$ je podmnožina koncových stavů, $q_0 \in Q$ je počáteční stav, A je konečná abeceda obsahující speciální prázdný znak ε a $\mathcal{I}$ je sada instrukcí tvaru (q, a, M, a', q'), kde $q,\ q' \in Q$, $a,\ a' \in A$, $M \in \{L, P, N\}$, což interpretujeme jako „nachází-li se stroj ve stavu q a přečetl z pásky písmeno a, tak napíše na pásku písmeno a', přehodí se do stavu q' a posune čtecí hlavu o jedno pole ve směru M (tj. vlevo, vpravo nebo nikam)“. Výpočet Turingova stroje probíhá tak, že na začátku je stroj ve stavu q_0, na pásce je napsán vstup, který obsahuje jen konečně mnoho znaků různých od ε a někam je nastavena čtecí hlava. Poté stroj sleduje instrukce (mění stavy, čte, píše*

a pohybuje čtecí hlavou) až do té doby, než se ocitne v koncovém stavu $q \in F$. To, co se v té chvíli nachází na pásce, je považováno za výstup. Formálně zapisujeme $T(\texttt{vstup}) = \texttt{vystup}$. *To, že se stroj někdy zastaví, zapisujeme* $T(\texttt{vstup}) \downarrow$. *Pokud se stroj nikdy nezastaví, píšeme* $T(\texttt{vstup}) \uparrow$.

Mezi všemi matematiky i informatiky na světě panuje shoda (říká se tomu *Churchova hypotéza*), že problém je algoritmicky řešitelný právě tehdy, lze-li na jeho řešení sestrojit Turingův stroj. Nějaké argumenty pro tuto hypotézu předvedu na přednášce — je třeba si uvědomit, že pro jakýkoliv program v běžném programovacím jazyku mohu vymyslet Turingův stroj, který dělá totéž, a naopak, umím-li něco řešit Turingovým strojem, umím to také naprogramovat (třeba v Pascalu).

Příklad. Na přednášce si zkusíme sestrojit Turingovy stroje řešící mj. následující úlohy:
(1) Zdvoj zadané slovo.
(2) Otoč zadané slovo.
(3) Řekni mi, zda má zadané slovo sudou či lichou délku.
(4) Řekni mi, zda se zadané slovo skládá ze dvou stejných slov.
(5) Pokud zakódujeme přirozené číslo $n \geq 0$ v abecedě $\{\varepsilon, 1\}$ jako $n+1$ jedniček, sečti dvě zadaná čísla.

Druhou technickou obtíží, která nás čeká, je sestrojení tzv. *univerzálního Turingova stroje*. To je takový stroj U, který umí simulovat výpočet každého Turingova stroje T. Na vstupu je na pásku univerzálního stroje napsán kód stroje[2] T a požadovaný vstup v stroje T. Výsledek výpočtu univerzálního stroje je pak stejný, jako výsledek stroje T se vstupem v. Formálně můžeme vyslovit tuto větu.

Věta. *Existuje Turingův stroj U takový, že pro každý stroj T a pro každý jeho vstup v platí*

$$T(v)\uparrow \quad \Rightarrow \quad U(\text{kód}\,T, v)\uparrow \qquad \& \qquad T(v)\downarrow \quad \Rightarrow \quad U(\text{kód}\,T, v) = T(v).$$

Konstrukce univerzálního stroje není úplně snadná a na přednášce bude předveden návod, který vás (snad) přesvědčí, že to je možné. *Vypisuji odměnu půl kilogramu čokolády pro nejrychlejšího autora (nebo autorský kolektiv) funkčního univerzálního Turingova stroje.*

A nyní se dostáváme do finále. Mějme nádsledující problém (říká se mu *halting problem*). Chceme najít algoritmus, který pro každý Turingův stroj T a libovolný jeho vstup v zjistí, zda se T s tímto vstupem zastaví. Tj. hledáme stroj S takový, že

[2] Vzhledem k tomu, že Turingův stroj je popsán konečně mnoha znaky, není problém najít vhodný přepis instrukční sady na pásku.

$S(\text{kód}\,T, v)$ se vždy zastaví a

$$S(\text{kód}\,T, v) = \texttt{ano} \quad \Leftrightarrow \quad T(v)\downarrow \qquad \& \qquad S(\text{kód}\,T, v) = \texttt{ne} \quad \Leftrightarrow \quad T(v)\uparrow .$$

Předpokládejme, že existuje. Pak ovšem existuje také Turingův stroj A takový, že pro každý vstup v platí $A(v)\downarrow$ právě tehdy, když $U(v, v)\uparrow$ — stačí, když A spustí výpočet $S(\text{kód}\,U, (v, v))$ a v případě záporné odpovědi ihned skončí, kdežto v případě kladné odpovědi pokračuje nekonečnou smyčkou. Jenže potom platí

$$A(\text{kód}\,A)\downarrow \quad \Leftrightarrow \quad U(\text{kód}\,A, \text{kód}\,A)\uparrow \quad \Leftrightarrow \quad A(\text{kód}\,A)\uparrow$$

(první ekvivalence je definice A, druhá je definice U), čímž máme spor!

Přeloženo do řeči běžného programátora to tedy znamená, že nikdy nenaprogramujete algoritmus, který by pro libovolný zadaný zdrojový kód a libovolný vstup toho programu řekl, jestli výpočet skončí nebo ne. Problém se vám může zdát trochu umělý, ale pomocí něj se dokáže algritmická neřešitelnost celé řady praktických problémů. Například:

- (*Gödel*) Neexistuje algoritmus, kterému byste zadali populárně řečeno tvrzení z teorie čísel (formálně řečeno tvrzení v Peanově aritmetice), a on by vám odpověděl, zda je tvrzení pravdivé.
- (*Matijasević*) Neexistuje ani algoritmus, který by uměl řešit diofantické rovnice.
- (*problém slov*) Máme-li zadány nějaké rovnosti, pomocí kterých můžeme symbolicky upravovat výrazy, pak většinou neexistuje algoritmus, který by pro dva dané výrazy odpověděl, zda se rovnají. Slovíčko „většinou“ funguje tak, že pro některé soustavy rovností to funguje (např. pro samotnou rovnost $(ab)c = a(bc)$), ale pro naprostou většinu soustav algoritmus neexistuje.
- (*Rice*) Neexistuje algoritmus, který by pro libovolné dva Turingovy stroje (v řeči programátora: pro dva zdrojové kódy v nějakém jazyce) odpověděl, zda počítají totéž.

A ještě zajímavost na konec. Není těžké dokázat, že existuje Turingův stroj T, který při libovolném vstupu vypíše svůj kód, tj. $T(cokoliv) = \text{kód}\,T$. I tento problém se dá vyřešit, překvapivě snadno, v každém běžném programovacím jazyce. *Kdo napíše v Pascalu program, který nemá žádný vstup a na obrazovku vypíše svůj zdrojový kód (bez použití disku a podobných podvodů), bude pochválen před nastoupenou jednotkou (a možná i něco dostane).*