Myšlienka dôkazu nerozhodnuteľnosti problému zastavenia

spísal MišoF. (podľa textu Janosa Simona z UChicago)

Náš dôkaz nebude úplne formálny. Nebudeme hovoriť o konkrétnom výpočtovom modeli, ale všeobecne o algoritmoch. Naším cieľom bude dokázať nasledovné tvrdenie:

> Nech $F(P, x)$ je funkcia, ktorej vstupom je dvojica: program P a jeho vstup x. Pritom
>
> $$F(P, x) = \begin{cases} 1 \leftarrow \text{ak } P \text{ na } x \text{ zastane} \\ 0 \leftarrow \text{inak} \end{cases}$$
>
> Potom neexistuje žiaden algoritmus, počítajúci túto funkciu.

Pritom zanedbáme kopu detailov, ktoré by museli byť súčasťou formálneho dôkazu. Nebude nás veľmi trápiť, v akom jazyku je program P napísaný (je nám jedno, či je to Pascal alebo δ-fcia Turingovho stroja). Takisto sa nebudeme zaoberať rôznymi abecedami, všetko predsa vieme zakódovať do postupnosti 0 a 1. Navyše sa dohodneme, že ak P nie je korektný program, tak $\forall x; F(P, x) = 0$.

Dôkaz bude sporom. Predpokladajme, že existuje algoritmus **Rozhodni**, rátajúci funkciu F. (Teda pre všetky P a x je **Rozhodni**$(P, x) = F(P, x)$.) Zostrojme potom algoritmus **Popleť**, ktorý bude využívať **Rozhodni** ako procedúru a bude fungovať nasledovne: Ak **Rozhodni**$(P, x) = 1$, **Popleť**(P, x) sa zacyklí, inak skončí. Teda **Popleť** na vstupe (P, x) zastane práve vtedy, keď sa P na vstupe x zacyklí.

Základný trik dôkazu je podstrčiť programu na vstup jeho vlastný zdrojový kód. Vieme, že **Rozhodni**(P, P) nám povie, či P na svojom kóde zastane alebo nie. Čo potom robí algoritmus **Naopak**(P) = **Popleť**(P, P)? Ak P na svojom kóde zastane, **Popleť**(P, P) (čiže **Naopak**(P)) sa zacyklí. Ak sa P na svojom kóde zacyklí, **Popleť**(P, P) zastane.

Preto sa algoritmus **Naopak** na vstupe P správa (naozaj) naopak ako algoritmus P na vstupe P. Ale tu sa už blížime k hľadanému sporu. **Naopak** sa totiž nemôže správať naopak sám od seba!

Zistime teda, čo sa stane, keď spustíme **Naopak** na jeho vlastnom zdrojovom kóde. Ak **Naopak**(**Naopak**) zastane, znamená to, že **Naopak** sa na vstupe **Naopak** zacyklil, čo je spor. A ak **Naopak**(**Naopak**) nezastane, znamená to, že **Naopak** na vstupe **Naopak** zastal, čo je opäť spor.
Všetky kroky nášho dôkazu boli korektné a dospeli sme k sporu. Preto nutne pôvodný predpoklad nebol správny, čiže algoritmus **Rozhodni** neexistuje.