Prednáška 8 - Turingove stroje

Ing. Viliam Hromada, PhD.

C-510
Ústav informatiky a matematiky
FEI STU

viliam.hromada@stuba.sk

24.11.2020

Opakovanie

- Doteraz sme sa bavili o regulárnych jazykoch a bezkontextových jazykoch.
- Regulárne jazyky sú popísateľné konečnými automatmi a regulárnymi výrazmi.
- Bezkontextové sú popísateľné zásobníkovými automatmi a bezkontextovými gramatikami.
- Regulárne jazyky sú použiteľné pri analýze protokolov (modelovanie e-peňažného systému) alebo pri vyhľadávaní textových vzorov.
- Bezkontextové jazyky sú použiteľné pri popise syntaxe programovacích / značkovacích jazykov a pri ich parsovaní.
- Otázkou ale je, aké jazyky sú použiteľné pri popise problému **algoritmicky** riešiteľného na klasickom počítači.

Motivácia

- Zaujíma nás, aké typy problémov dokáže riešiť bežné výpočtové zariadenie (počítač).
- Znovu, ak máme nejaký rozhodovací problém, tak množina inštancií, pre ktorú je odpoveď "áno", tvorí nejaký jazyk.
- Zaujíma nás, čí dokáže počítač v konečnom čase rozhodnúť, či je pre danú inštanciu rozhodovacieho problému odpoveď áno alebo nie.
- T.j., či dokáže počítač v konečnom čase rozhodnúť, či reťazec popisujúci inštanciu problému patrí do nejakého jazyka alebo nie.

Motivácia

- Spomínali sme si napríklad problém, kde je potrebné zistiť, či 2 ľubovoľné bezkontextové gramatiky generujú presne ten istý jazyk, t.j. či $L(G_1) = L(G_2)$ pre 2 bezkontextové gramatiky G_1, G_2.
- Tento problém patrí práve medzi tzv. nerozhodnuteľné problémy, t.j. **neexistuje jeden algoritmus**, ktorý by dokázal pre **ľubovoľné bezkontextové gramatiky** v **konečnom čase** poskytnúť odpoveď áno / nie.
- Na štúdium toho, čo všetko dokáže výpočtové zariadenie (bežný počítač) potrebujeme jeho matematický model. Tým je tzv. **Turingov stroj**.

- Samozrejme, by niekto mohol skúsiť riešiť problém, či $L(G_1) = L(G_2)$ tak, že by hľadal taký reťazec, ktorý patrí do $L(G_1)$ a nepatrí do $L(G_2)$ alebo naopak.
- Existuje efektívny algoritmus, ktorý zistí, či $w \in L(G_1)$ alebo $w \in L(G_2)$.
- Teoreticky by sa dalo vygenerovať nejaké slovo $w \in L(G_1)$ a potom testovať, či $w \in L(G_2)$, alebo naopak.
- Ak sa také slovo nájde - máme odpoveď, že $L(G_1) \neq L(G_2)$.
- Čo ale, ak $L(G_1) = L(G_2)$? Taký program by predsa nikdy neskončil!
- Preto dôležitým kritériom toho, či existuje nejaký algoritmus je jeho **konečnosť**, t.j. či v konečnom čase skončí!

Počítače nie sú všemocné

- To, že musia existovať problémy, ktoré sa nedajú riešiť v konečnom čase na klasickom počítači vytvorením programu v nejakom jazyku, sa dá zdôvodniť jednoduchým myšlienkovým procesom.
- Nech uvažujeme nejaký programovací jazyk - napríklad C.
- Ak by sme uvažovali **všetky možné programy** napísané v jazyku C, tak týchto programov je nekonečne veľa. Platí však, že ich je **spočítateľné nekonečne veľa** - čo znamená, že ku každému programu vieme jednoznačne priradiť nejaké prirodzené číslo - napríklad by sme všetky možné programy abecedne zoradili a očíslovali ich poradie.
- Všetky platné C programy tak tvoria množinu, ktorej kardinalita je rovnaká, ako kardinalita prirodzených čísiel, t.j. $\aleph_0$.

- Rozhodovací problém je vlastne problém príslušnosti do jazyka. Jazyk je **každá podmnožina** množiny všetkých prípustných reťazcov, t.j. ak máme abecedu Σ, všetky reťazce sú Σ^*, tak každá podmnožina $L \subseteq \Sigma^*$ je nejaký jazyk, teda potenciálny rozhodovací problém.
- Ak by sme vedeli riešiť všetky rozhodovacie problémy, potom by pre každý potenciálny problém musel existovať C-kód, ktorý implementuje algoritmus, ktorý ho rieši.
- Problém je, že množina všetkých možných jazykov je **nespočítateľné nekonečná**, t.j. kardinality $2^{\aleph_0}$.
- To znamená, že **nemôže existovať** priradenie C-kódov ku každému problému, lebo problémov je "oveľa viac". Teda musia existovať jazyky, ktoré nie sú akceptovateľné C-programom, t.j. musia existovať nerozhodnuteľné problémy.

Začnime s niečím jednoduchým

- Ukážme si príklad jednoduchého problému, o ktorom vieme dokázať, že ho počítač nevie riešiť, t.j. nevie v **konečnom čase** poskytnúť jednoznačnú odpoveď.
- Náš problém je nasledovný: *Vieme pre zadaný program v jazyku C rozhodnúť, či prvá vec, ktorú vypíše, bude reťazec* `hello, world` *?*
- Zadaný je teda **ľubovoľný C program** a my musíme vedieť v **konečnom čase rozhodnúť**, **či k výpisu dôjde alebo nie**.

Je zrejmé, že pre jednoduchý program typu:

```
main()
{
    printf("hello, world\n");
}
```

je odpoveď zrejmá - **áno**, tento program ako prvú vec vypíše naozaj `hello, world`

Čo ale v prípade, že to chceme zistiť pre nasledovný kód, kde `exp(a,b)` je funkcia, ktorá počíta hodnotu a^b:

```
main()
{
    int n, total, x, y, z;
    scanf("%d",&n);
    total = 3;
    while(1) {
        for (x = 1; x <= total-2; x++)
            for (y = 1; y <= total-x-1; y++) {
                z = total - x - y;
                if (exp(x,n) + exp(y,n) == exp(z,n))
                    printf("hello, world\n");
                }
            total++;
        }
}
```

- Z našej analýzy kódu je zrejmé, že k výpisu `hello, world` dôjde vtedy, keď sa nájdu také celé čísla x, y, z, že pre zadané n bude platiť: $x^n + y^n = z^n$.
- Ak sa zadá $n = 2$, tak potom sú také čísla $x = 3, y = 4, z = 5$ a naozaj sa `hello, world` vypíše.
- Čo ak $n \geq 3$???
- Ak existujú x, y, z, tak program určite ako prvé vypíše `hello, world`
- Ak by také čísla neexistovali, tak program by sa zacyklil a `hello, world` by sa nikdy nevypísalo.
- Čiže otázka, či sa vypíše `hello, world` sa zredukuje na otázku, či existujú čísla x, y, z,pre nejaké n, že $x^n + y^n = z^n$.

- Odpoveď na túto otázku trvala matematikom 300 rokov - až v rokoch 1994/1995 sa podarilo britskému matematikovi Andrewovi Wilesovi dokázať tzv. Veľkú Fermatovu vetu, ktorá hovorí, že pre celé číslo $n > 2$ neexistujú celé čísla x, y, z také, že $x^n + y^n = z^n$, teda posledných cca 25 rokov vieme, že tento program pre $n > 2$ `hello, world` nikdy nevypíše a pre $n == 2$ ho vypíše.

- Čiže pre uvedený program **vieme povedať** pre všetky možné vstupy (v tomto prípade hodnota parametra *n*), či dôjde k výpisu `hello, world` alebo nie.
- Znovu sa však vráťme k otázke - existuje **jeden algoritmus**, ktorý by dokázal pre **akýkoľvek C program** a jeho **všetky prípustné vstupy** v **konečnom čase** povedať, či program vypíše `hello, world` ako svoj prvý výpis alebo nie?

- **Ak by taký algoritmus existoval**, potom by sa vďaka nemu dalo zodpovedať veľa doteraz nezodpovedaných otázok!
- Napríklad by sme urobili C program, v ktorom je výpis `hello, world` podmienený tým, že sa zároveň nájde odpoveď na nejaký doteraz neznámy problém.
- Ak by potom náš univerzálny algoritmus povedal, že daný program vypíše `hello, world`, znamenalo by to, že odpoveď na problém **musí existovať!**
- Naopak, ak by náš univerzálny algoritmus povedal, že daný program nikdy `hello, world` nevypíše, znamenalo by to, že odpoveď na problém **neexistuje!**

Iný príklad

Nech `goldbach(p1,p2,n)` je funkcia, ktorá vráti *True*, ak $n = p1 + p2$ a $p1, p2$ sú prvočísla:

```
main()
{
    int n = 4;
    int p1,p2, found;
    while(1) {
        found = 0;
        for (p1 = 2; p1 <= n/2; p1++){
            p2 = n - p1;
            if (goldbach(p1,p2,n)) {
                found = 1; break; }
        }
        if (found == 0)
            printf("hello, world\n");
        n+=2;
    }
}
```

- Uvedený program prechádza párne kladné čísla *n* a vypíše `hello, world` v prípade, že sa dané číslo *n* **nedá napísať** ako súčet 2 prvočísiel.
- Ak by existoval univerzálny algoritmus, ktorý by vedel povedať, či tento daný program vypíše `hello, world` alebo nie, tak v takom prípade:
  - Ak by povedal, že sa `hello, world` nikdy nevypíše, tak to znamená, že všetky párne kladné čísla sa dajú napísať ako súčet 2 prvočísiel
  - Ak by povedal, že sa `hello, world` vypíše, tak to znamená ,že musí **existovať** párne kladné číslo, ktoré nie je súčtom 2 prvočísiel
- Momentálne (22. 11. 2020) na otázku, či sa dá každé párne kladné číslo napísať ako súčet 2 prvočísiel **neexistuje v matematike odpoveď!** - ide o tzv. Goldbachovu hypotézu, ktorá tvrdí, že **každé párne kladné číslo** sa dá napísať ako súčet 2 prvočísiel, ale neexistuje jej dôkaz.
- A teda algoritmus, ktorý by vedel rozhodnúť, či sa vypíše `hello, world` by tým pádom Goldbachovu hypotézu nepriamo dokázal alebo vyvrátil!

- Vidíme teda, že dať odpoveď na otázku, či ľubovoľný C program vypíše `hello, world` ako svoj prvý výpis, nie je také jednoduché.
- Konkrétnejšie nás zaujíma, či existuje algoritmus *H*, ktorý by pre program *P* a jeho vstup *I* dokázal v konečnom čase rozhodnúť, či program *P* so vstupom *I* vypíše `hello, world`
- V podstate by výstupom programu *H* bola textová odpoveď `áno` alebo `nie` podľa toho, či program *P* so vstupom *I* vypíše / nevypíše `hello, world`
- Ak taký algoritmus *H* existuje, tento problém nazývame **rozhodnuteľný**.
- Ak taký algoritmus *H* neexistuje, tento problém nazývame **nerozhodnuteľný**.
- Ukážeme, že pre `hello, world` je tento problém **nerozhodnuteľný**, t.j. pre tento problém neexistuje algoritmus *H*.

Dôkaz

- Dôkaz toho, že **neexistuje algoritmus** *H*, ktorý by pre ľubovoľný program *P* a jeho vstup *I* dokázal v konečnom čase povedať, či program vypíše `hello, world`, urobíme ako dôkaz sporom.
- To znamená, že budeme **predpokladať**, že takýto algoritmus *H* **existuje**.
- Nech teda existuje nejaký algoritmus *H*, ktorý pre ľubovoľný program *P* a vstup *I* vráti výstup `yes` v prípade, ak program *P* pre vstup *I* vypíše `hello, world` a vráti výstup `no` v prípade, ak program *P* pre vstup *I* nevypíše `hello, world`.
- Keďže *H* je algoritmus, uvažujme, že *H* je aj jeho implementácia v jazyku *C* a vrátenie výstupu `yes` alebo `no` môžeme vnímať ako výpis na obrazovku.

- Nech sa teda algoritmus (program) *H* správa podľa obrázka:

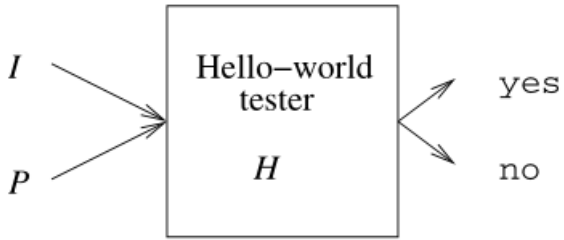


Figure: Správanie programu *H* [1]

- Ak existuje program H, tak potom určite existuje program H_1, ktorý vznikne tak, že v H zmeníme výpis `no` na výpis `hello, world`.
- To znamená, že ak program H_1 zistí, že analyzovaný program P so vstupom I vypíše na obrazovku `hello, world`, tak H_1 vypíše (vráti) `yes` a keď zistí, že P so vstupom I nevypíše na obrazovku `hello, world`, tak H_1 vypíše (vráti) `hello, world`
- Čiže H_1 sa správa rovnako ako H s tým rozdielom, že namiesto `no` vracia `hello, world`

- Nech sa teda program H_1 správa podľa obrázka:

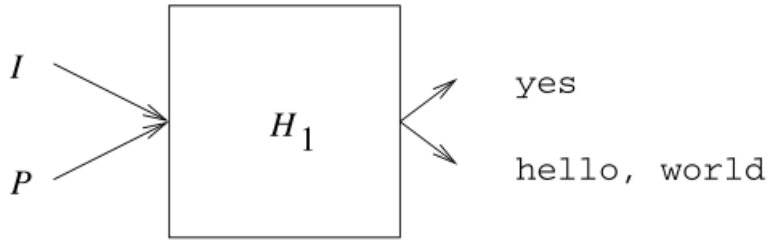


Figure: Správanie programu H_1 [1]

- Urobme ešte jednu modifikáciu.
- Ak existuje program H_1, tak potom určite existuje program H_2, ktorý vznikne tak, že v H_1 budeme uvažovať $P = I$, t.j. program H_2 bude simulovať správanie programov P, ktorých vstupom je znovu samotný program P, t.j. $I = P$ (to sa dá v praxi realizovať napríklad tak, že ak P je kód v jazyku C, tak I je textový reťazec reprezentujúci daný kód)
- T.j. H_2 bude simulovať správanie vstupného kódu P predstavujúceho nejaký program v jazyku C, ktorého vstupom je jeho vlastný kód P.
- Znovu platí, že H_2 vráti `yes` v prípade, že vstupný kód P spolu so vstupom P vypíše `hello, world` a H_2 vráti `hello, world` v prípade, že vstupný kód P spolu so vstupom P nevypíše `hello, world`

- Nech sa teda program H_2 správa podľa obrázka:

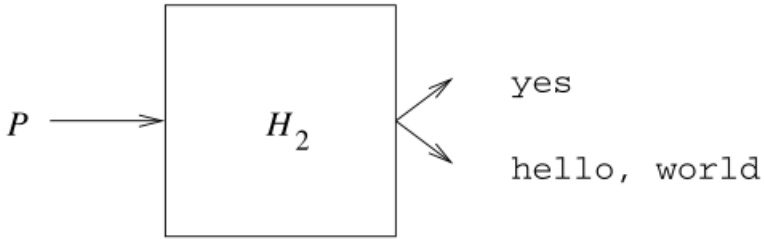


Figure: Správanie programu H_2 [1]

- Pre pripomenutie, všimnite si, že sme na začiatku predpokladali, že existuje program (algoritmus) H, ktorý dokáže **korektne pre všetky možné** programy v jazyku C P a ich vstupy I povedať, či vypíšu / nevypíšu na obrazovku `hello, world`
- Následne sme z programu H odvodili program H_1 (úpravou výpisu) a z neho program H_2 (úpravou vstupu)
- Všetky úpravy sú korektné, pretože spadajú do kategórie toho, čo vieme s algoritmom / programom v jazyku C urobiť.
- Pozrime sa, ako sa bude správať program H_2, **ak dostane na vstup sám seba**.

H_2 sa teda správa takto pre vstupný program P:

1. Ak P vypíše `hello, world`, tak H_2 vypíše `yes`
2. Ak P nevypíše `hello, world`, tak H_2 vypíše `hello, world`

Čo sa udeje, ak pomocou H_2 analyzujeme samotné H_2, t.j. ak $P = H_2$?

1. Predpokladajme, že $P = H_2$ vypíše `yes`. V takom prípade by H_2 v štvorčeku malo vrátiť (vypísať) `hello, world`, lebo jeho vstup nevypíše `hello, world`. Avšak to je v **spore s tým**, že predpokladáme, že H_2 vypíše `yes`!
2. Predpokladajme, že $P = H_2$ vypíše `hello, world`. V takom prípade by H_2 v štvorčeku malo vrátiť (vypísať) `yes`, lebo jeho vstup vypíše `hello, world`. Avšak to je v **spore s tým**, že predpokladáme, že H_2 vypíše `hello, world`!

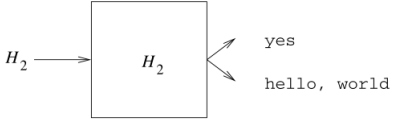


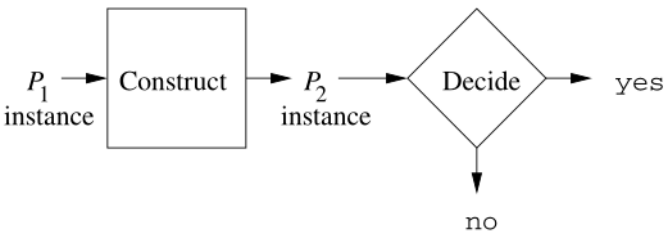
Figure: Správanie programu H_2 [1]

- Dospeli sme do situácie, že H_2 **nesprávne** určilo, či H_2 vypíše / nevypíše `hello, world`.
- Avšak H_2 bolo zostrojené **korektnými krokmi** z algoritmu (programu) H, ktorý by sa **nikdy nemal mýliť**, t.j. pre všetky vstupy **korektne povie**, či vypíšu `hello, world` alebo nie, preto ani H_2 by sa nikdy mýliť nemal!
- Tým pádom sme dospeli k **sporu** a keďže predpokladom bolo, že H existuje, tak záverom je, že H **existovať nemôže**!

- Ukázali sme teda, že **neexistuje algoritmus**, ktorý dokáže pre **akýkoľvek** vstupný program povedať, či vypíše `hello, world` alebo nie.
- Táto úloha - rozhodnúť, či ľubovoľný program vypíše / nevypíše `hello, world` je príkladom **nerozhodnuteľného problému**.

Redukcie problémov

- Existencia nerozhodnuteľných problémov nám vie poslúžiť na dôkaz toho, že aj iné problémy sú nerozhodnuteľné, pomocou tzv. redukcie.
- Majme problém P_1, o ktorom vieme, že je nerozhodnuteľný a problém P_2, o ktorom chceme dokázať, že je nerozhodnuteľný:
- Klasicky sa to robí pomocou nasledovnej schémy:



- Ak sa nám podarí ukázať, že platí implikácia: *Ak je problém* P_2 *rozhodnuteľný, potom je aj problém* P_1 *rozhodnuteľný*, tak zo znalosti toho, že P_1 je nerozhodnuteľný logicky vyplýva, že aj P_2 je nerozhodnuteľný.
- Pretože v opačnom prípade by nastala situácia, že P_1 je rozhodnuteľný a nerozhodnuteľný zároveň, čo nemôže byť.

- Štvorec "Construct" v obrázku predstavuje konštrukciou s cieľom:
  - Každú inštanciu w problému P_1 vie skonvertovať na inštanciu x problému P_2 s rovnakou odpoveďou na príslušný problém.
  - Ak w je inštancia problému P_1 s odpoveďou áno, tak bude skonvertovaná na inštanciu x problému P_2 s odpoveďou áno.
  - Ak w je inštancia problému P_1 s odpoveďou nie, tak bude skonvertovaná na inštanciu x problému P_2 s odpoveďou nie.
  - Následne, ak by sme vedeli zistiť, či x patrí do P_2, tak by sme z toho logicky usúdili, či w patrí do P_1 (teda ak by bol P_2 rozhodnuteľný, tak by bol aj P_1)
- To znamená, že ak by existoval algoritmus, ktorý by vedel rozhodovať o probléme P_2, tak na základe danej konverzie by musel existovať algoritmus, ktorý vie rozhodovať o probléme P_1. Ale keďže taký algoritmus pre P_1 **neexistuje**, tak nemôže existovať ani pre P_2.

Príklad

Uvažujme nasledovný rozhodovací problém: *Je daný program Q so vstupom y. Zavolá tento program niekedy počas svojej činnosti funkciu* `foo` *?*

- Ak program takú funkciu neobsahuje, tak je odpoveď určite **nie.**
- Ak takú funkciu obsahuje, tak potom je to problém zistiť.
- Pri dôkaze, či existuje algoritmus, ktorý by to vedel v konečnom čase zistiť, si pomôžeme tým, že vieme, že **neexistuje** algoritmus, ktorý by vedel v konečnom čase povedať, či nejaký program s nejakým vstupom ako prvý výpis vypíše `hello, world`.

- Nech problém P_1 je problém, či program ako prvý výstup vypíše `hello, world`
- Nech problém P_2 je problém, či program niekedy zavolá funkciu `foo`
- **Predpokladajme, že problém P_2 je rozhodnuteľný**, t.j. existuje algoritmus, ktorý vie v konečnom čase rozhodnúť, či program počas svojho behu zavolá funkciu `foo`.
- Našou úlohou je teraz nájsť redukciu problému P_1 na problém P_2.

1. Nech je daný nejaký program Q so vstupom y.
2. Vytvorme v programe Q novú prázdnu funkciu `foo`. Ak sa v ňom už nachádza funkcia s takýmto menom, tak pôvodnú funkciu premenujme, rovnako zmeníme všetky jej volania. Tento nový program nazvime Q_1.
3. V programe Q_1 budeme priebežne sledovať vypisované znaky. V prípade, že zistíme, že prvých 12 vypísaných znakov je `hello, world`, tak program zavolá novovytvorenú funkciu `foo`. Tento výsledný program nazvime R. Nech jeho vstup bude y, t.j. vstup pôvodného programu Q.

1. Čo by sa udialo, ak by program Q so vstupom y vypísal `hello, world` ako svoj prvý výpis? **Určite by program R zavolal funkciu `foo`.**
2. Čo by sa udialo, ak by program Q so vstupom y nevypísal `hello, world` ako svoj prvý výpis? **Určite by program R nezavolal funkciu `foo`.**
3. To znamená volanie `foo` v programe R je ekvivalentné výpisu `hello, world` ako prvého výpisu v programe Q.
4. Ak predpokladáme, že **existuje algoritmus**, ktorý vie v konečnom čase rozhodnúť, či R so vstupom y volá funkciu `foo`, tak potom by **ten istý algoritmus** zároveň detegoval, či program Q vypisuje `hello, world` ako svoj prvý výpis!
5. My však vieme, že taký algoritmus **nemôže existovať!**
6. Preto **nemôže existovať** ani algoritmus, ktorý v konečnom čase rozhodne, či program R so vstupom y volá funkciu `foo`.

- Doteraz sme sa bavili o nerozhodnuteľných problémoch najmä z pohľadu existencie nejakého programu v zdrojovom kóde.
- Avšak pre dôkazy o nerozhodnuteľnosti problémov potrebujeme nejaký teoretický matematický model počítača.
- Takýmto modelom je **Turingov stroj**.

Turingove stroje

- Turingov stroj je *v podstate* konečný automat, ktorý disponuje pamäťovým médiom - páskou nekonečnej dĺžky - z ktorej môže čítať a na ktorú môže zapisovať dáta.
- Táto páska pozostáva z buniek, v ktorých sa môžu nachádzať "dáta" formou symbolov.
- Na tejto páske sa počas svojej činnosti môže pohybovať doľava i doprava.
- Rovnako, ako KA, sa počas svojej činnosti nachádza v tzv. stavoch.
- Na začiatku svojej činnosti má na páske napísaný vstup (vstupný reťazec). Tento reťazec počas činnosti môže čítať, prípadne i prepisovať. Taktiež môže na pásku na hociktoré miesto zapisovať a čítať z nej.
- V prípade, že dospeje do niektorého zo stavov označeného ako akceptačný, tak vstup akceptuje, bez ohľadu na to, **čo sa na páske nachádza**. Taktiež bez ohľadu na to, či **prečítal celý vstup alebo nie**.

- Turingove stroje formálne zaviedol britský matematik Alan Turing v roku 1936 ako model výpočtového zariadenia.
- Zaujímavosťou je, že to bolo niekoľko rokov predtým, než boli vôbec zostrojené prvé elektrické, či dokonca elektromechanické počítače.
- Napriek svojmu pomerne primitívnemu dizajnu existuje predpoklad, že výpočtová sila Turingových strojov je ekvivalentná s výpočtovou silou počítačov, ako ich poznáme dnes.
- **Church-Turingova téza:** Ku každému algoritmu existuje ekvivalentný Turingov stroj.
- Táto téza nie je formálne dokázateľná a na jej vyvrátenie by musel existovať výpočtový stroj, ktorý vie riešiť problémy, ktoré Turingov stroj riešiť nevie - napríklad problém rozhodnutia v konečnom čase, či program vypíše `hello, world` ako svoj prvý výstup.

- Neformálne vieme povedať, že každý program, zapísaný v ľubovoľnom programovacom jazyku, sa dá preložiť do ekvivalentného programu vykonateľného na Turingovom stroji.
- V praxi hovoríme o tzv. Turingovsky-úplných programovacích jazykoch - sú to jazyky, ktoré majú rovnakú silu, ako Turingove stroje - napríklad C, C++, Java, Python, ...
- Na to, aby bol jazyk Turingovo-úplný, musí obsahovať (citát z Wikipedie):
  - cyklus while-do
  - neobmedzenú (teoreticky) rekurziu
  - podmienený skok

Turingov stroj

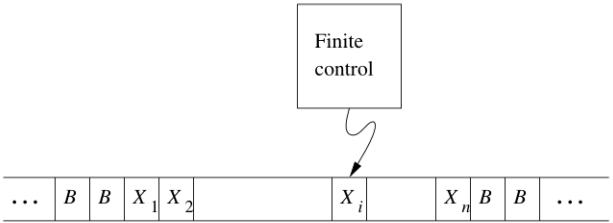


Figure: Schéma TS [1]

Finite control je riadiaca stavová jednotka, ktorá je v každom momente v nejakom stave. Zároveň sa vždy pomocou *čítaco/píšucej hlavy* díva na nejakú bunku nekonečnej pásky (na obrázku sa v bunke nachádza symbol X_i). Na páske sa okrem symbolov $X_1, X_2, ..., X_n$ nachádzajú aj prázdne bunky označené symbolom B (blank).

- Na začiatku sa na pásku zapíše *vstup*, ktorým je reťazec konečnej dĺžky nad *vstupnou abecedou*. Ostatné bunky pásky sú prázdne.
- Na páske sa teda môžu nachádzať symboly vstupnej abecedy. Ďalej tam môžu byť aj iné, tzv. páskové symboly a jeden špeciálny páskový symbol B, reprezentujúci prázdnu bunku. B nesmie byť zároveň aj vstupný symbol.
- Na začiatku je čítaco-píšuca hlava (ČPH) nastavená na prvom symbole vstupného reťazca.
- Zároveň je na začiatku riadiaca stavová jednotka v nejakom počiatočnom stave q_0.

- *Krok výpočtu TS* je funkciou aktuálneho stavu a symbolu čítaného z pásky. Na ich základe vykoná TS:
  1. Prechod do ďalšieho stavu (môže byť aj ten istý stav, ako aktuálny)
  2. Do čítanej bunky zapíše nejaký páskový symbol. Môže zapísať aj taký symbol, aký bol čítaný (t.j. de facto sa obsah bunky nezmení).
  3. Posunie ČPH o 1 pozíciu na páske doľava alebo doprava.

Formálne je Turingov stroj sedmica $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$:

1. Konečná množina stavov Q
2. Konečná abeceda vstupných symbolov Σ
3. Konečná množina páskových symbolov Γ
4. Počiatočný stav q_0, $q_0 \in Q$
5. Prázdny symbol B, $B \in \Gamma$, $B \notin \Sigma$, reprezentuje prázdne bunky na páske, t.j. na začiatku je všade na páske tam, kde nie je vstupné slovo
6. Množina akceptačných stavov F, $F \subseteq Q$.
7. Prechodová funkcia δ, definovaná na ďalšom slajde.

Prechodová funkcia δ má vstupné argumenty $\delta(q, X)$, kde:

- Aktuálny stav q, $q \in Q$
- Aktuálne čítaný páskový symbol X, $X \in \Gamma$

Nech je hodnota $\delta(q, X)$ definovaná ako trojica (p, Y, D). V tejto trojici:

- p je stav, do ktorého sa TS prepne, $p \in Q$
- Y je páskový symbol, ktorý sa zapíše na miesto, kde bol čítaný symbol X, $Y \in \Gamma$
- D je smer (vľavo = L, vpravo = R), kam sa posunie ČPH

Príklad

Nech je daný Turingov stroj: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, kde:

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, X, Y, B\}$
- $F = \{q_4\}$

a prechodová funkcia δ je daná:

Stav \ Symbol	0	1	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	$(q_1, 0, R)$	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Ako uvedený TS spracuje vstup 0011? Pre sledovanie činnosti potrebujeme vždy sledovať

- Aktuálny stav
- Aktuálny obsah pásky a aktuálne čítaný symbol

Pre reťazec 0011 na začiatku platí:

- Aktuálny stav je počiatočný stav q_0
- Na páske je zapísaný vstup, t.j 0011 a aktuálne je nastavená ČPH (čítaco-píšuca hlava) na prvom symbole, t.j. na prvej nule. Ostatné symboly pásky sú prázdne (čo "označíme" tak, že tam bude zobrazený symbol B)
- Pozíciu ČPH vzhľadom na pásku označíme šípkou

Stav: q_0

Pozícia ČPH			↓					
Páska	...	B	0	0	1	1	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_0, 0) = (q_1, X, R)$, t.j. TS prejde zo stavu q_0 čítaním symbolu 0 z pásky do stavu q_1, na pásku zapíše symbol X a posunie hlavu o jednu pozíciu vpravo.

Stav: q_1

Pozícia ČPH				↓				
Páska	...	B	X	0	1	1	B	...

Stav: q_1

Pozícia ČPH				↓				
Páska	...	B	X	0	1	1	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_1, 0) = (q_1, 0, R)$, t.j. TS prejde zo stavu q_1 čítaním symbolu 0 z pásky do stavu q_1, na pásku zapíše symbol 0 a posunie hlavu o jednu pozíciu vpravo.

Stav: q_1

Pozícia ČPH					↓			
Páska	...	B	X	0	1	1	B	...

Stav: q_1

Pozícia ČPH					↓			
Páska	...	B	X	0	1	1	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_1, 1) = (q_2, Y, L)$, t.j. TS prejde zo stavu q_1 čítaním symbolu 1 z pásky do stavu q_2, na pásku zapíše symbol Y a posunie hlavu o jednu pozíciu vľavo.

Stav: q_2

Pozícia ČPH				↓				
Páska	...	B	X	0	Y	1	B	...

Stav: q_2

Pozícia ČPH				↓				
Páska	...	B	X	0	Y	1	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_2, 0) = (q_2, 0, L)$, t.j. TS prejde zo stavu q_2 čítaním symbolu 0 z pásky do stavu q_2, na pásku zapíše symbol 0 a posunie hlavu o jednu pozíciu vľavo.

Stav: q_2

Pozícia ČPH			↓					
Páska	...	B	X	0	Y	1	B	...

Stav: q_2

Pozícia ČPH			↓					
Páska	...	B	X	0	Y	1	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_2, X) = (q_0, X, R)$, t.j. TS prejde zo stavu q_2 čítaním symbolu X z pásky do stavu q_0, na pásku zapíše symbol X a posunie hlavu o jednu pozíciu vpravo.

Stav: q_0

Pozícia ČPH				↓				
Páska	...	B	X	0	Y	1	B	...

Stav: q_0

Pozícia ČPH				↓				
Páska	...	B	X	0	Y	1	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_0, 0) = (q_1, X, R)$, t.j. TS prejde zo stavu q_0 čítaním symbolu 0 z pásky do stavu q_1, na pásku zapíše symbol X a posunie hlavu o jednu pozíciu vpravo.

Stav: q_1

Pozícia ČPH					↓			
Páska	...	B	X	X	Y	1	B	...

Stav: q_1

Pozícia ČPH					↓			
Páska	...	B	X	X	Y	1	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_1, Y) = (q_1, Y, R)$, t.j. TS prejde zo stavu q_1 čítaním symbolu Y z pásky do stavu q_1, na pásku zapíše symbol Y a posunie hlavu o jednu pozíciu vpravo.

Stav: q_1

Pozícia ČPH						↓		
Páska	...	B	X	X	Y	1	B	...

Stav: q_1

Pozícia ČPH						↓		
Páska	...	B	X	X	Y	1	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_1, 1) = (q_2, Y, L)$, t.j. TS prejde zo stavu q_1 čítaním symbolu 1 z pásky do stavu q_2, na pásku zapíše symbol Y a posunie hlavu o jednu pozíciu vľavo.

Stav: q_2

Pozícia ČPH					↓			
Páska	...	B	X	X	Y	Y	B	...

Stav: q_2

Pozícia ČPH					↓			
Páska	...	B	X	X	Y	Y	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_2, Y) = (q_2, Y, L)$, t.j. TS prejde zo stavu q_2 čítaním symbolu Y z pásky do stavu q_2, na pásku zapíše symbol Y a posunie hlavu o jednu pozíciu vľavo.

Stav: q_2

Pozícia ČPH				↓				
Páska	...	B	X	X	Y	Y	B	...

Stav: q_2

Pozícia ČPH				↓				
Páska	...	B	X	X	Y	Y	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_2, X) = (q_0, X, R)$, t.j. TS prejde zo stavu q_2 čítaním symbolu X z pásky do stavu q_0, na pásku zapíše symbol X a posunie hlavu o jednu pozíciu vpravo.

Stav: q_0

Pozícia ČPH					↓			
Páska	...	B	X	X	Y	Y	B	...

Stav: q_0

Pozícia ČPH					↓			
Páska	...	B	X	X	Y	Y	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_0, Y) = (q_3, Y, R)$, t.j. TS prejde zo stavu q_0 čítaním symbolu Y z pásky do stavu q_3, na pásku zapíše symbol Y a posunie hlavu o jednu pozíciu vpravo.

Stav: q_3

Pozícia ČPH						↓		
Páska	...	B	X	X	Y	Y	B	...

Stav: q_3

Pozícia ČPH						↓		
Páska	...	B	X	X	Y	Y	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_3, Y) = (q_3, Y, R)$, t.j. TS prejde zo stavu q_3 čítaním symbolu Y z pásky do stavu q_3, na pásku zapíše symbol Y a posunie hlavu o jednu pozíciu vpravo.

Stav: q_3

Pozícia ČPH							↓	
Páska	...	B	X	X	Y	Y	B	...

Stav: q_3

Pozícia ČPH						↓		
Páska	...	B	X	X	Y	Y	B	...

Podľa prechodovej funkcie je pre situáciu $\delta(q_3, B) = (q_4, B, R)$, t.j. TS prejde zo stavu q_3 čítaním symbolu B z pásky (t.j. číta prázdny symbol, resp. na páske sa v bunke nenachádza žiaden symbol) do stavu q_4, na pásku zapíše symbol B a posunie hlavu o jednu pozíciu vpravo.

Stav: q_4

Pozícia ČPH								↓	
Páska	...	B	X	X	Y	Y	B	B	...

Stav: q_4

Pozícia ČPH								↓	
Páska	...	B	X	X	Y	Y	B	B	...

- Keďže TS sa ocitol v stave q_4, ktorý je akceptačný, tak v danom momente končí svoju činnosť a akceptuje vstupný reťazec 0011.
- Navyše, pre stav q_4 a páskový symbol B nie je definovaný žiaden prechod, takže v tejto situácii by sa TS zastavil (zasekol).

Keby nás zaujímalo, ako by sa TS správal pre vstup 001, tak sa dá ukázať, že z počiatočnej situácie:

Stav: q_0

Pozícia ČPH			↓				
Páska	...	B	0	0	1	B	...

by sme sa vedeli dostať do situácie:

Stav: q_1

Pozícia ČPH						↓	
Páska	...	B	X	X	Y	B	...

Keďže neexistuje prechod $\delta(q_1, B)$, tak v tejto situácii by sa TS zastavil. Avšak, keďže q_1 nie je akceptačný stav, tak vstup 001 by TS neakceptoval.

Keby nás zaujímalo, ako by sa TS správal pre vstup ε, tak sa dá ukázať, že z počiatočnej situácie:
Stav: q_0

Pozícia ČPH			↓				
Páska	...	B	B	B	B	B	...

by sme sa nevedeli ďalej pohnúť, lebo $\delta(q_0, B)$ nie je definované. A keďže q_0 nie je akceptačný stav, tak tento TS neakceptuje vstup ε.
Všimnite si ,že keďže sme chceli spracovať prázdny reťazec, tak na páske bol už na začiatku napísaný prázdny reťazec - t.j. všetky symboly pásky boli prázdne. V takom prípade môže hlava ukazovať na ľubovoľný prázdny symbol.

Konfigurácia TS

- Pre popis činnosti TS používame podobne ako pri zásobníkových automatoch tzv. konfigurácie.
- Každá konfigurácia popisuje v akom **stave** sa nachádza TS, aký je **aktuálny obsah** pásky a aký je **aktuálny vstupný symbol**.
- Hoci páska je nekonečné dlhá, tak keďže počet vykonaných krokov je vždy konečný, aj počet nie-prázdnych symbolov na páske je konečný.
- Preto v konfigurácii uvádzame zväčša len neprázdne symboly, t.j. relevantný obsah pásky.
- Prázdne symboly pásky B uvádzame len, ak je to potrebné, pretože sa na ne díva ČPH, prípadne sa nachádzajú medzi blokmi neprázdnych symbolov.

Konfigurácia TS

- Na zachytenie pozície ČPH a aktuálneho stavu popisujeme konfiguráciu tak, že akoby aktuálny stav vložíme do obsahu pásky, pričom aktuálny stav zapíšeme **vľavo** pred aktuálne čítaný vstupný symbol.
- Preto je vhodné, aby sa mená stavov zároveň nevyskytovali ako páskové symboly.
- To znamená, že reťazec $X_1 X_2 ... X_{i-1} q X_i X_{i+1} ... X_n$ reprezentuje nasledovnú konfiguráciu

Stav: q

Pozícia ČPH							$\downarrow$					
Páska	...	B	X_1	X_2	...	X_{i-1}	X_i	X_{i+1}	...	X_n	B	...

- Krok výpočtu TS M zapíšeme ako reláciu medzi konfiguráciami označenú $\vdash$
- Nula-alebo-viac krokov výpočtu medzi konfiguráciami zapíšeme $\vdash^*$
- Predpokladajme, že $\delta(q, X_i) = (p, Y, L)$, potom platí:

$$X_1 X_2 ... X_{i-1} q X_i X_{i+1} ... X_n \vdash X_1 X_2 ... p X_{i-1} Y X_{i+1} ... X_n$$

t.j. prechod z konfigurácie
Stav: q

Pozícia ČPH							$\downarrow$					
Páska	...	B	X_1	X_2	...	X_{i-1}	X_i	X_{i+1}	...	X_n	B	...

do konfigurácie
Stav: p

Pozícia ČPH						$\downarrow$						
Páska	...	B	X_1	X_2	...	X_{i-1}	Y	X_{i+1}	...	X_n	B	...

Špeciálne, nech $\delta(q, X_i) = (p, Y, L)$, t.j. hlava sa hýbe vľavo a:

- Ak $i = 1$:

$$qX_1X_2...X_n \vdash pBYX_2...X_n$$

- Ak $i = n$, $Y = B$

$$X_1X_2...X_{n-1}qX_n \vdash X_1X_2...X_{n-2}pX_{n-1}$$

Keď $\delta(q, X_i) = (p, Y, R)$, t.j. hlava sa hýbe vpravo, tak vo všeobecnosti:

$$X_1X_2...X_{i-1}qX_iX_{i+1}...X_n \vdash X_1X_2...X_{i-1}YpX_{i+1}...X_n$$

Znovu, existujú 2 špeciálne prípady:

- Ak $i = n$:

$$X_1X_2...X_{n-1}qX_n \vdash X_1X_2...X_{n-1}YpB$$

- Ak $i = 1$, $Y = B$

$$qX_1X_2...X_n \vdash pX_2...X_n$$

Príklad

Nech je daný Turingov stroj: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, kde:

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, X, Y, B\}$
- $F = \{q_4\}$

a prechodová funkcia δ je daná:

Stav \ Symbol	0	1	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	$(q_1, 0, R)$	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Spracovanie vstupu 0011 zapíšeme pomocou konfigurácií nasledovne:

$$q_0 0011 \vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y 1 \vdash q_2 X 0 Y 1 \vdash X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash$$

$$\vdash X X q_2 Y Y \vdash X q_2 X Y Y \vdash X X q_0 Y Y \vdash X X Y q_3 Y \vdash X X Y Y q_3 B \vdash X X Y Y B q_4 B$$

T.j. v skrátenom tvare vieme výpočet zapísať:

$$q_0 0011 \vdash^* X X Y Y B q_4 B$$

Spracovanie vstupu 001 zapíšeme pomocou konfigurácií nasledovne:

$$q_0001 \vdash Xq_101 \vdash X0q_11 \vdash Xq_20Y \vdash q_2X0Y \vdash Xq_00Y \vdash XXq_1Y \vdash XXYq_1B$$

T.j. v skrátenom tvare vieme výpočet zapísať:

$$q_0001 \vdash^* XXYq_1B$$

Podobne spracovanie vstupu ε by na začiatku (a hneď na konci, keďže nie je možný žiaden prechod) predstavovala konfigurácia:

$$q_0B$$

t.j. je možný výpočet na nula krokov:

$$q_0B \vdash^* q_0B$$

Rozoberme si, čo vykonáva uvedený TS:

- Na začiatku je na páske nejaký reťazec z núl a jednotiek (a vo zvyšných bunkách sú prázdne symboly B) a automat v stave q_0.
- Ak číta v stave q_0 jednotku - zasekne sa v neakceptačnom stave q_0.
- Ak číta v stave q_0 nulu - prepíše ju na X, prepne sa do stavu q_1 a pokračuje vpravo.
- V stave q_1 - "preskakuje" nuly, preskakuje Y a hľadá jednotku.
- Ak v stave q_1 číta jednotku - prepíše ju na Y, prepne sa do stavu q_2 a vyberie sa po páske doľava.
- V stave q_2 sa posúva po páske vľavo, pričom preskakuje symboly Y a nuly. Ak narazí na X, tak sa prepne do stavu q_0 a znovu sa vyberie doprava.

- T.j. v stave q_0 nájde prvú nulu - prepíše ju na X a hľadá prvú jednotku vpravo od nej. Tú prepíše na Y a vráti sa vľavo na poslednú nulu prepísanú na X. T.j. po prvom takomto cykle sa páska

$$0 \mid 0 \mid ... \mid 0 \mid 1 \mid 1 \mid ... \mid 1$$

zmení na

$$X \mid 0 \mid ... \mid 0 \mid Y \mid 1 \mid ... \mid 1$$

- T.j. akoby sa ku každej nule na páske hľadá jednotka vpravo od nej
- **Ak je vstup v tvare** 0^n1^n, tak určite nastane situácia, že po spárovaní poslednej nuly a poslednej jednotky sa pri posune vľavo v stave q_2 narazí na také X, od ktorého vpravo už žiadna ďalšia nula neexistuje.
- V takom prípade sa prepne do stavu q_3 a prechodom cez všetky Y prejde na prvý prázdny symbol B za vstupom. Ten následne umožní prechod do akceptačného stavu.
- Preto uvedený TS skončí v stave q_4 len pre jazyk $L = \{0^n1^n \mid n \geq 1\}$.

Jazyk akceptovaný TS

Definícia

*Nech $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ je Turingov stroj. Nech $L(M)$ je množina všetkých reťazcov $w \in \Sigma^*$ takých, že $q_0 w \vdash^* \alpha p \beta$, kde $p \in F$ a α, β sú nejaké reťazce páskových symbolov. Potom množinu $L(M)$ nazývame **jazykom akceptovaným Turingovym strojom** M.*

Jazyky, pre ktoré existuje Turingov stroj, ktorý ich akceptuje, sa nazývajú **rekurzívne vyčísliteľné jazyky**, alebo RE jazyky (Recursively Enumerable).

Grafické znázornenie TS

- Podobne ako KA alebo ZA, aj TS vieme znázorniť pomocou prechodového diagramu.
- Uzly (vrcholy) v prechodovom diagrame budú stavy, počiatočný stav je označený vstupujúcou šípkou, akceptačné stavy sú označené 2 kružnicami
- Prechod $\delta(q, X) = (p, Y, L)$ je v diagrame označený orientovanou hranou zo stavu q do stavu p, pričom hrana je označená $X/Y \leftarrow$, t.j. je v nej ľavou šípkou označený posun hlavy doľava.
- Prechod $\delta(q, X) = (p, Y, R)$ je v diagrame označený orientovanou hranou zo stavu q do stavu p, pričom hrana je označená $X/Y \rightarrow$, t.j. je v nej pravou šípkou označený posun hlavy doprava.

Príklad

TS zo slajdov 65, resp. 43 by bol prechodovým diagramom znázornený ako:

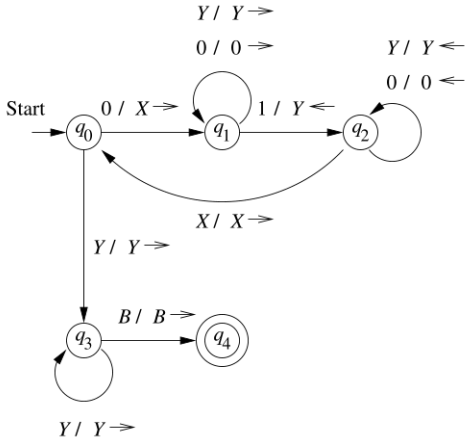


Figure: TS pre jazyk $\{a^n b^n, n \geq 1\}$, [1]

(Halting) Zastavenie a TS

- V definícii jazyka TS sme uviedli, že ho tvoria tie reťazce, počas spracovania ktorých vie dôjsť TS do akceptačného stavu.
- T.j. TS by neakceptoval tie reťazce, pre ktoré sa nikdy nedostane do akceptačného stavu - či už z dôvodu toho, že sa zasekne, alebo z dôvodu toho, že sa zacyklí.

(Halting) Zastavenie a TS

- Často sa uvažuje iná definícia "akceptácie" Turingovým strojom - akceptácia zastavením.
- Hovoríme, že TS sa zastaví, ak sa ocitne v stave q, na páske číta symbol X a $\delta(q, X)$ nedefinuje žiaden prechod.
- Ak by sme chceli upraviť TS, ktorý má akceptačné stavy, aby sa zároveň zastavil, vieme ho upraviť, aby pre akceptačný stav q neboli definované prechody pre žiadne symboly X.

(Halting) Zastavenie a TS

- Žiaľ, nie vždy vieme docieliť to, aby TS zastavil pre reťazce, ktoré akceptovať **nemá**.
- Preto sa v praxi TS ešte delia na také TS, ktoré sa zastavia **vždy** pre ľubovoľné reťazce, bez ohľadu na to, či ich akceptujú alebo nie.
- A na TS, ktoré sa zastavia len pre reťazce, ktoré akceptujú a pre reťazce, ktoré neakceptujú, sa alebo zaseknú v neakceptačnom stave, alebo sa počas ich spracovania **zacyklia**, t.j. TS teoreticky nikdy neskončí svoju činnosť.

- Tie TS, ktoré sa **vždy** zastavia bez ohľadu na vstup a to, či ho akceptujú alebo nie, predstavujú "dobrý" model algoritmu.
- A ak takýto TS existuje pre riešenie nejakého problému, t.j. pre ľubovoľný vstup TS určite v konečnom čase skončí a dokáže povedať, či vstup akceptuje alebo nie, tak tento TS je zároveň modelom algoritmu pre daný problém a daný problém nazývame **rozhodnuteľný** (decidable).

Použitá literatúra

1 Hopcroft, Motwani, Ullman - Introduction to Automata Theory, Languages and Computations, 3rd Ed.