

Prednáška 11 - Redukcie problémov, Riceova veta

Ing. Viliam Hromada, PhD.

C-510

Ústav informatiky a matematiky
FEI STU

`viliam.hromada@stuba.sk`

8.12.2020

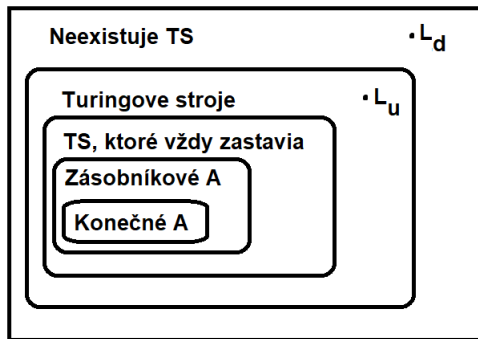


- Na minulej prednáške sme sa bavili o možnostiach rozšírenia Turingovych strojov - t.j. možnosť pridať viacero pásov alebo pripustiť nedeterministické správanie TS, pričom TS s obomi rozšíreniami vedia robiť len to, čo deterministický TS s jednou páskou.
- Taktiež sme sa bavili o tom, že rekurzívne vyčísliteľné jazyky (RE jazyky) - t.j. tie, ktoré je možné akceptovať Turingovými strojmi, obsahujú dôležitú podmnožinu
 - Jazyky, pre ktoré existuje Turingov stroj, ktorý **vždy zastaví** - tzv. **rekurzívne jazyky**
 - Tie jazyky, pre ktoré existuje len taký Turingov stroj, ktorý **zastaví len** v prípade, že **vstup akceptuje** - predstavujú tie jazyky, ktoré sú RE ale nie sú rekurzívne.

- 

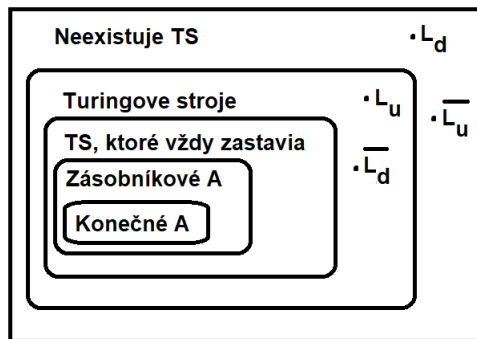
- Príklad jazyka, ktorý nie je RE - t.j. **neexistuje TS**, ktorý by vedel akceptovať práve daný jazyk,
$$L_d = \{w \mid w \text{ kóduje taký TS } M, \text{ ktorý neakceptuje sám seba, } w \notin L(M)\}$$
- Na minulej prednáške sme dokázali, že taký TS nemôže existovať.





- Rekurzívne jazyky - tie, pre ktoré existujú **algoritmy, ktoré ich riešia** - t.j. vieme napísať počítačový program, ktorý **vždy skončí**, t.j. vždy korektne vráti odpoveď ÁNO/NIE.
- RE jazyky, ktoré nie sú rekurzívne - vieme napísať počítačový program, ktorý **skončí, ak správna odpoveď je ÁNO** - ak je odpoveďou NIE, existuje možnosť, že program NIKDY neskončí.
- jazyky, ktoré nie sú RE - **neexistuje program**, ktorý by problém vedel riešiť.





Redukcia problému P_1 na P_2

- Vieme teda, že jazyk L_d nie je rekurzívne vyčísliteľný (teda problém L_d nie je rozhodnuteľný a zároveň neexistuje TS, ktorý by aspoň vedel zastaviť v prípade, že odpoveď je ÁNO).
- A jazyk L_u je rekurzívne vyčísliteľný, ale nie je rekurzívny (teda problém L_u nie je rozhodnuteľný, avšak existuje TS, ktorý vie zastaviť aspoň v prípade, že odpoveď je ÁNO).
- Za ich pomoci vieme o ďalších iných jazykoch dokázať, že tiež alebo nie sú RE, alebo sú RE ale nie sú rekurzívne.
- Urobíme to pomocou tzv. **redukcie**.



Idea redukcie P_1 na P_2

- Pre $\gcd(a, b)$ máme efektívny algoritmus: Euklidov algoritmus.
- Navyše vieme, že v matematike platí: $\text{lcm}(a, b) = \frac{ab}{\gcd(a, b)}$
- Teda namiesto priameho počítania $\text{lcm}(a, b)$ vypočítame $\gcd(a, b)$ a vydelíme $\frac{ab}{\gcd(a, b)}$
- Výsledok potom predstavuje hľadanú hodnotu $\text{lcm}(a, b)$
- Čiže namiesto riešenia problému P_1 sme vyriešili problém P_2 a jeho riešenie použili na riešenie problému P_1 .



Redukcia problému P_1 na P_2

- Uvažujme, že máme rozhodovací problém P_1 a rozhodovací problém P_2 .
- Ak **existuje algoritmus**, ktorý dokáže konvertovať **každú inštanciu** problému P_1 s odpoveďou **ÁNO** na **nejakú inštanciu** problému P_2 s odpoveďou **ÁNO**, a **každú inštanciu** problému P_1 s odpoveďou **NIE** na **nejakú inštanciu** problému P_2 s odpoveďou **NIE**, potom hovoríme, že sa problém P_1 **redukuje** na problém P_2 .



- To znamená, že ukážeme, že problém P_2 je **aspoň tak ťažký**, ako problém P_1 .
- Z toho logicky vyplýva, že **ak by sme vedeli riešiť** P_2 , potom by sme **museli vedieť riešiť** aj P_1 .
- A naopak, ak vieme, že P_1 je **nerozhodnuteľný problém**, potom aj P_2 musí byť **taktiež nerozhodnuteľný problém**.
- Respektíve, ak vieme, že P_1 nie je v množine RE jazykov (teda preň neexistuje TS), potom aj P_2 musí byť mimo množinu RE jazykov (teda ani preň nemôže existovať TS).



Príklad

Uvažujme napríklad nasledovné 2 problémy:

- P_1 : Vypíše program Q so vstupom y ako prvých 12 znakov na obrazovku `hello, world?`
- P_2 : Zavolá program R so vstupom z počas svojho behu funkciu `foo`?

Len pripomínam, že sme si ukázali pred 2 týždňami, že problém P_1 je **nerozhodnuteľný**, t.j. príslušný jazyk nie je **rekurzívny**.



1. Ak program `Q` obsahuje funkciu s menom `foo`, tak zmeň jej meno a jej volania na `ine`.
2. Pridaj do programu novú prázdnu funkciu `foo`.
3. Pridaj do programu sledovanie vypísaných znakov. Ak zistíš, že prvých 12 znakov na výpise je `hello, world`, zavolaj novo pridanú funkciu `foo`.

Tento nový program, ktorý sme vytvorili z programu Q označme ako R . Nech vstup programu R bude rovnaký, ako vstup Q , t.j. $z = y$.



- Je zrejmé, že v programe R dôjde k volaniu funkcie `foo` **práve vtedy**, keď v programe Q dôjde k tomu, že sa na výpise ako prvých 12 znakov zjaví `hello, world`.
- Teda **každý program**, ktorý ako prvý výpis vypíše `hello, world` sa dá previesť na **nejaký** program, ktorý volá `foo`, a zároveň **každý program**, ktorý nevypíše ako prvý výpis `hello, world` sa dá previesť na **nejaký** program, ktorý nikdy nevolá `foo`.
- Našli sme teda **algoritmus**, ktorý každú inštanciu `hello, world` problému P_1 prevedie na inštanciu `foo` problému P_2 s **rovnakou odpoveďou**.
- Teda sme našli **redukciu** problému P_1 na problém P_2 .



Predpokladajme, že by sme vedeli **rozhodnúť** problém P_2 - teda pre ľubovoľný program by sme vedeli s **istotou rozhodnúť**, či dôjde k volaniu funkcie `foo` alebo nie, teda mali by sme **algoritmus**, ktorý rieši (rozhoduje) P_2

- Ak by sme daný rozhodovací algoritmus aplikovali na tie programy, ktoré vznikli z inštancií `hello, world` problému postupom zo slajdu č. 15, potom by sme vedeli rozhodnúť **či dochádza / nedochádza** k výpisu `hello, world!`
- Teda, ak by sme vedeli algoritmicky rozhodnúť problém P_2 , **vedeli by sme s istotou** rozhodnúť problém P_1 .



- O probléme P_1 však vieme, že sa **nedá algoritmicky rozhodnúť**, či dochádza k ako prvému výpisu k výpisu `hello, world!`
- A teda sa logicky nemôže dať ani rozhodnúť, či program volá funkciu `foo`, teda problém P_2 je rovnako **algoritmicky nerozhodnuteľný**!



Redukcia P_1 na P_2

Na obrázku je znázornené, ako dochádza ku algoritmickej konverzii inštancií problému P_1 na inštancie problému P_2 s rovnakou odpoveďou:

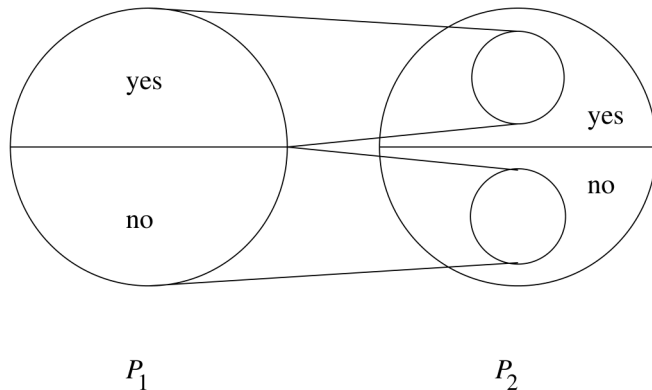


Figure: Redukcia P_1 na P_2 , [1]

Len opakujem - a z obrázka je to rovnako zrejmé, že pri redukcii inštancií problému P_1 na inštalácie problému P_2 s rovnakou odpoveďou dochádza k tomu, že:

- **Každú inštaláciu** P_1 s odpoveďou ÁNO konvertujeme na **nejakú inštaláciu** problému P_2 s odpoveďou ÁNO
- **Každú inštaláciu** P_1 s odpoveďou NIE konvertujeme na **nejakú inštaláciu** problému P_2 s odpoveďou NIE

Všimnite si v obrázku, že tie inštalácie P_2 s odpoveďou (ÁNO/NIE), na ktoré sme konvertovali inštalácie problému P_1 môžu tvoriť len **podmnožinu** všetkých inštancií P_2 . To je dôvod, pre ktorý hovoríme, že P_2 je aspoň tak ťažké ako P_1 .



Totíto:

1. Inštancie P_1 sú všetky možné zdrojové kódy - časť z nich, ktoré vypisuje `hello, world` ako prvý výpis má odpoveď ÁNO, ostatné majú odpoveď NIE.
2. Inštancie P_2 sú všetky možné zdrojové kódy - časť z nich, ktorá volá `f○○` má odpoveď ÁNO, ostatné majú odpoveď NIE.
3. Inštancie P_2 , ktoré vznikli redukciou z P_1 sú **len podmnožina** všetkých zdrojových kódov v P_2 - to je dané napríklad tým, že samotná funkcia `f○○`, ktorú sme tam ručne vyrobili, **nič nerobí, teda jej kód je prázdny**.



- Samotná redukcia inštancií P_1 na inštancie P_2 musí byť vykonateľná pomocou nejakého **algoritmu**, t.j. redukciu inštancií P_1 na P_2 si vieme predstaviť ako Turingov stroj, ktorému na pásku zapíšeme inštanciu P_1 a tento TS **vždy zastaví** s tým, že na páske zostane inštancia P_2 .
- V našom príklade to znamenalo upraviť zdrojový kód programu Q na iný zdrojový kód programu R - táto úprava sa dá vždy urobiť v konečnom čase, teda naozaj šlo o algoritmus (TS, ktorý vždy zastaví).

Dôležitá veta

Na základe uvedeného platí nasledovná veta:

Veta

Nech existuje redukcia problému P_1 na problém P_2 . Potom

- 1. Ak problém P_1 je nerozhodnuteľný, potom aj problém P_2 je nerozhodnuteľný.*
- 2. Ak problém P_1 nie je rekurzívne vyčísliteľný, potom aj problém P_2 nie je rekurzívne vyčísliteľný.*



Dôkaz

Nech existuje redukcia problému P_1 na P_2 . To znamená, že existuje nasledovné:

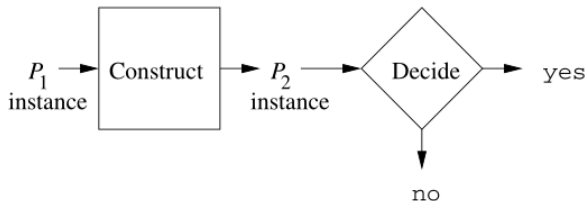


Figure: Redukcia P_1 na P_2 [1]

Štvorec "construct" predstavuje spomínaný algoritmus, ktorý každú inštanciu P_1 prevedie na nejakú inštanciu P_2 s rovnakou odpoveďou.

Predpokladajme, že P_1 je **nerozhodnuteľný**.

1. Vezmeme inštanciu w problému P_1 a prevedieme ju na inštanciu x problému P_2 .
2. Ak vieme rozhodovať problém P_2 , tak aj je odpoveď pre inštanciu x ÁNO, potom aj pre problém w musí byť odpoveď ÁNO (lebo redukcia zachováva odpovede).
3. Ak je odpoveď pre x NIE, potom aj pre w musí byť odpoveď NIE.
4. To znamená, že rozhodovač pre problém P_2 vieme použiť v kombinácii s redukciou z P_1 na P_2 pre rozhodovanie problému P_1 - **čo je v spore s tým, že predpokladáme, že P_1 je nerozhodnuteľný!**

Preto ak P_1 je nerozhodnuteľný a existuje redukcia z P_1 na P_2 , potom aj P_2 **musí byť nerozhodnuteľný!**



Predpokladajme, že P_1 nie je RE, t.j. **neexistuje TS**, ktorý by vedel pre inštancie w problému P_1 s odpoveďou ÁNO túto odpoveď vrátiť v konečnom čase.

1. Vezmeme inštanciu w problému P_1 a prevedieme ju na inštanciu x problému P_2 .
2. Ak by bol problém P_2 rekurzívne vyčísliteľný, tak existuje TS, ktorý by pre inštancie x problému P_2 vrátil odpoveď ÁNO v konečnom čase pre tie x , ktoré majú odpoveď ÁNO.
3. To však znamená, že ak w prevedieme na x a pre x vieme zistiť odpoveď v konečnom čase, aj pre w vieme zistiť odpoveď ÁNO v konečnom čase.
4. Teda TS pre problém P_2 s redukciami z P_1 na P_2 nám dokopy tvoria TS pre problém P_1 , čo je v **spore s tým, že predpokladáme, že TS pre P_1 neexistuje**.

Preto ak P_1 nie je rekurzívne vyčísliteľný a existuje redukcia z P_1 na P_2 , potom ani P_2 **nemôže byť RE!**



2 dôležité jazyky L_e , L_{ne}

- Minulý týždeň sme si ukázali, že **každý binárny reťazec** w vieme konvertovať na nejaký Turingov stroj M_i .
- Zároveň vieme, že každý Turingov stroj môže akceptovať nejaké reťazce.
- Položme si otázku, či $L(M_i) = \emptyset$. To znamená zaujíma nás, či je jazyk akceptovaný Turingovým strojom prázdny - t.j. situácia, že TS **neakceptuje nič** - to môže nastať alebo tým, že TS nemá žiadne akceptačné stavy, alebo tým, že sú nedosiahnuteľné pre ľubovoľný vstup.
- Doplnkovou otázkou je, či $L(M_i) \neq \emptyset$, teda či TS akceptuje aspoň nejaký (stačí jeden) reťazec.



2 dôležité jazyky L_e, L_{ne}

- Množinu **všetkých binárných reťazcov**, ktoré kódujú také Turingove stroje, ktoré **nič neakceptujú**, označíme ako L_e .

$$L_e = \{M \mid L(M) = \emptyset\}$$

- Množinu **všetkých binárných reťazcov**, ktoré kódujú také Turingove stroje, ktoré **akceptujú aspoň jeden reťazec**, označíme ako L_{ne} .

$$L_{ne} = \{M \mid L(M) \neq \emptyset\}$$

- V oboch prípadoch ide o jazyk tvorený binárnymi reťazcami, t.j. nad abecedou $A = \{0, 1\}$.



L_{ne} je RE jazyk

Najprv si dokážeme nasledovnú vetu:

Veta

L_{ne} je rekurzívne vyčísliteľný jazyk.

To znamená, že existuje **jeden** Turingov stroj akceptujúci jazyk L_{ne} , ktorý dokáže pre tie reťazce, ktoré predstavujú Turingove stroje, ktoré akceptujú aspoň jeden reťazec, v konečnom čase vrátiť odpoveď ÁNO.



Dôkaz

Ak chceme dokázať, že nejaký jazyk je RE, musíme nájsť Turingov stroj M , ktorý ho akceptuje.

1. Hľadáme Turingov stroj M , ktorému na pásku napíšeme binárny reťazec.
2. Tento TS následne zistí, či tento binárny reťazec predstavuje kód takého Turingovho stroja M_i , ktorý dokáže akceptovať nejaký reťazec.
3. Náš hľadaný Turingov stroj M bude fungovať ako nedeterministický Turingov stroj.
4. Nedeterministicky si "vygeneruje" nejaký reťazec w a otestuje, či M_i akceptuje w - to vie urobiť napríklad pomocou Univerzálneho TS U .
5. Ak M_i akceptuje reťazec w , tak UTS U vráti pre dvojicu (M_i, w) odpoveď ÁNO.
6. A teda M zistí, že jeho vstup predstavuje taký TS M_i , ktorý akceptuje aspoň 1 reťazec.



Uvedený TS M :

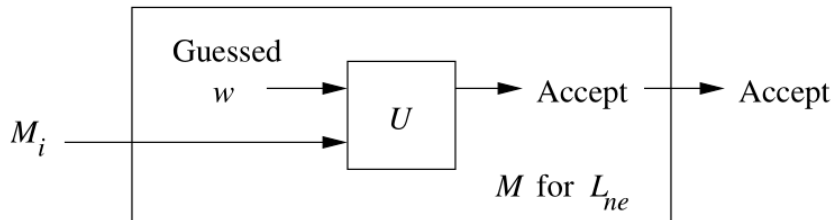


Figure: TS M pre akceptáciu L_{ne} [1]

Dôkaz

- Ak M_i naozaj predstavuje taký TS, ktorý akceptuje aspoň 1 reťazec, tak hoci sa náš TS M správa nedeterministicky, bude existovať výpočet, v ktorom vygeneruje reťazec w práve taký, že ho M_i bude akceptovať a teda náš TS M vráti správnu odpoveď ÁNO.
- V prípade, že by M_i neakceptovalo nič, tak:
 - Ak sa M_i zastaví pre všetky vstupy, aj naša simulácia v M sa zastaví vždy.
 - Ak sa M_i môže pre nejaké vstupy zacykliť, aj naša simulácia v M sa môže zacykliť.
- Preto v prípade, že M_i neakceptuje nič, existuje šanca, že náš TS M nevráti žiadnu odpoveď.
- Jazyk L_{ne} je teda určite RE, lebo existuje aspoň taký TS, ktorý vráti vždy korektnú odpoveď ÁNO v konečnom čase.



L_{ne} nie je rekurzívny jazyk

- Dokázali sme teda, že L_{ne} je RE jazyk, pretože sme našli taký TS, ktorý pre všetky kódy Turingových strojov, ktoré patria do L_{ne} , vráti odpoveď ÁNO.
- Vieme však nájsť aj taký TS, ktorý by vždy v konečnom čase vrátil aj korektnú odpoveď NIE?
- Inými slovami - je jazyk L_{ne} rekurzívny a teda rozhodnuteľný?
- Dokážeme, že jazyk L_{ne} **nie je rekurzívny**, t.j. taký Turingov stroj **neexistuje** a jazyk L_{ne} je **nerozhodnuteľný**.
- Dokážeme to práve pomocou **redukcie z nerozhodnuteľného problému** L_U .



Redukcia L_U na L_{ne}

Idea redukcie:

- Vezmime dvojicu (M, w) .
- Uvažujme taký Turingov stroj M' , ktorý vezme dvojicu (M, w) a spustí jej simuláciu pomocou Univerzálneho TS. Ak simulácia zistí, že $w \in L(M)$, tak potom Turingov stroj M' akceptuje svoj vlastný vstup (nech je ním čokoľvek).
- To znamená, že ak $w \in L(M)$, potom $L(M')$ obsahuje nekonečne veľa reťazcov, teda $L(M') \neq \emptyset$.
- A ak $w \notin L(M)$, potom $L(M') = \emptyset$.



L_{ne} nie je rekurzívny jazyk

Platí veta:

Veta

Jazyk L_{ne} nie je rekurzívny.

Dôkaz

Aby sme dokázali, že existuje redukcia L_u na L_{ne} , musíme ukázať, že sa dá algoritmicky:

- Skonvertovať inštancia L_u , t.j. pár (M, w) na inštanciu L_{ne} , t.j. M' , pričom ak $(M, w) \in L_u$, tak potom $M' \in L_{ne}$ a ak $(M, w) \notin L_u$, tak potom $M' \notin L_{ne}$.
- Teda $L(M') \neq \emptyset$ vtedy a len vtedy, ak $w \in L(M)$.



Takýto algoritmus redukcie by prebiehal nasledovne: Uvažuj Turingov stroj M' ktorý vykonáva nasledovné:

1. Ignoruje svoj vlastný vstup x a miesto toho ho na páske prepíše reťazcom, ktorý popisuje nejaký pár (M, w) .
2. Následne zavolá v rámci svojej činnosti Univerzálny TS U , na dvojici (M, w) - t.j. **simuluje** činnosť TS M na vstupe w .
3. Ak U akceptuje (M, w) , potom M' vráti ÁNO.
4. Teda ak U neakceptuje (M, w) , potom ani M' neakceptuje.



Je zrejmé, že pre uvedený Turingov stroj M' platí:

- Ak pracuje s párom (M, w) takým, že $w \in L(M)$, tak potom **vždy** skončí svoju činnosť s odpoveďou ÁNO. Teda M' určite akceptuje nejaké (dokonca všetky) reťazce a teda pre $(M, w) \in L_U$ máme $M' \in L_{ne}$.
- Ak pracuje s párom (M, w) takým, že $w \notin L(M)$, tak potom sa **alebo** nezastaví, alebo zastaví, ale vstup neakceptuje (podľa toho, čo vykoná U). V oboch prípadoch však nikdy nevráti ÁNO a teda neakceptuje žiaden vstup a pre $(M, w) \notin L_U$ máme $M' \notin L_{ne}$.
- Z uvedeného vyplýva, že sme našli redukciu, ktorá pre **každú inštanciu** (M, w) problému L_U vráti inštanciu M' problému L_{ne} s rovnakou odpoveďou.
- A teda, keďže vieme, že L_U je **nerozhodnuteľný problém**, podľa vety zo slajdu č. 23 vieme, že aj L_{ne} je tým pádom **nerozhodnuteľný problém**.



A čo jazyk L_e ???

Dokázali sme teda 2 dôležité výsledky pre jazyk $L_{ne} = \{M \mid L(M) \neq \emptyset\}$:

- Jazyk L_{ne} je rekurzívne vyčísliteľný. (slajd č. 29)
- Jazyk L_{ne} nie je rekurzívny (slajd č. 36).

Keďže jazyk $L_e = \{M \mid L(M) = \emptyset\}$ je **doplnok jazyka** L_{ne} , tak podľa informácií z minulej prednášky, resp. zo slajdu 7, musí platiť, že L_e **nie je rekurzívne vyčísliteľný jazyk**.



Všetky možné jazyky $\cdot L_d$

Rekurzívne vyčísliteľné $\cdot L_u$

Rekurzívne $\cdot \overline{L_u}$

Bezkontextové $\cdot \overline{L_d}$

Regulárne $\cdot L_e$

$\cdot \overline{L_{ne}}$

The diagram illustrates the hierarchy of Turing machines and their complexity classes. It consists of four nested rectangles, each representing a set of Turing machines, with associated complexity classes listed to the right:

- Outermost rectangle:** Neexistuje TS (Does not exist TS). Complexity classes: $\cdot L_d$.
- Second rectangle:** Turingove stroje (Turing machines). Complexity classes: $\cdot L_u$ and $\cdot \overline{L_u}$.
- Third rectangle:** TS, ktoré vždy zastavia (TS that always halt). Complexity classes: $\cdot \overline{L_d}$ and $\cdot L_e$.
- Innermost rectangle:** Zásobníkové A (Stack A). Complexity class: $\cdot L_{ne}$.

Below the innermost rectangle, the text "Konečné A" (Finite A) is written.

Pre pripomenutie

Otázku, či je nejaký jazyk $L = \emptyset$, t.j. prázdny, sme už riešili pri regulárnych a bezkontextových jazykoch:

- Pre regulárne jazyky je tento problém rozhodnuteľný algoritmom, kde vezmeme DKA A , kde $L = L(A)$ a zistíme, či je nejaký akceptačný stav dosiahnuteľný z počiatočného.
- Pre bezkontextové jazyky je tento problém rozhodnuteľný algoritmom, kde vezmeme bezkontextovú gramatiku G , kde $L = L(G)$ a zistíme, či je počiatočný neterminál S generujúci symbol.

A teraz sme si ukázali, že otázka, či je rekurzívne vyčísliteľný jazyk prázdny, t.j. či $L = \emptyset$ nielen, že **nie je rozhodnuteľná**, ale dokonca ani **neexistuje TS**, ktorý by dokázal vrátiť odpoveď ÁNO v konečnom čase.



Dôležité!

Uvedomte si rozdiel medzi 2 jazykmi L_{ne} a L_e :

- Pre L_{ne} existuje aspoň taký postup, ktorý vie vrátiť odpoveď ÁNO v konečnom čase.
- Pre L_e dokonca neexistuje postup, ktorý by vedel vrátiť odpoveď ÁNO v konečnom čase.

Ako by ste (programátorsky) testovali, či nejaký jazyk (Turingov stroj) akceptuje nejaké reťazce?

- Náhodne budem generovať reťazce a skúšať, či ich akceptuje - ak mám šťastie a trafiť taký, ktorý tam patrí, tak viem s istotou, že jazyk patrí do L_{ne} - dá sa ukázať, že existuje inteligentný postup (tzv. interleaving), ako skúšať náhodné reťazce tak, že sa nezacyklím v jednom reťazci.
- Na to, aby som zistil, že jazyk je prázdny, by som však musel vedieť (nejako???) ukázať, že taký reťazec neexistuje!!! - čo sa určite **nedá robiť tak**, že budem skúšať všetky vstupy a každý by TS neakceptoval - keďže ich je nekonečne veľa.



Riceova veta

- Fakt, že oba jazyky L_e, L_{ne} sú nerozhodnuteľné sa dá dokázať aj pomocou tzv. Riceovej vety.
- Riceova veta hovorí, že všetky netriviálne vlastnosti rekurzívne vyčísliteľných jazykov sú nerozhodnuteľné.
- To znamená, že nie je možné pomocou Turingovho stroja, ktorý vždy zastaví, rozhodnúť či nejaký rekurzívne vyčísliteľný jazyk spĺňa nejakú netriviálnu vlastnosť.
- Ekvivalentne, nie je možné pomocou Turingovho stroja, ktorý vždy zastaví, rozpoznávať také **binárne reťazce**, ktoré kódujú Turingove stroje, ktorých akceptované jazyky spĺňajú nejakú netriviálnu vlastnosť.



Riceova veta

- Napríklad, podľa Riceovej vety nie je možné rozhodnúť, či je rekurzívne vyčísliteľný jazyk súčasne aj bezkontextovým jazykom.
- Teda ekvivalentne, či **nejaký Turingov stroj**, akceptuje práve taký jazyk, ktorý je súčasne aj bezkontextovým jazykom.

Čo je to "vlastnosť" RE jazyka

- Uvažujme množinu všetkých rekurzívne vyčísliteľných jazykov.
- Teda množinu všetkých takých jazykov, ku ktorým sa dá zostrojiť Turingov stroj.
- Ľubovoľná podmnožina týchto jazykov sa nazýva *vlastnosť* RE jazykov.
- T.j. napríklad *vlastnosť RE jazykov* *taká*, že jazyk je zároveň aj bezkontextový, je v tomto zmysle naozaj *vlastnosťou* - a teda by sme uvažovali množinu $L_{CFL} = \{L \mid L \text{ je bezkontextový jazyk} \}$. Ekvivalentne $L_{CFL} = \{L \mid \text{existuje ZA } P, L = L(P)\}$.
- A podľa Riceovej vety neexistuje algoritmus, ktorý by vedel v konečnom čase rozhodnúť, či je zadaný rekurzívne vyčísliteľný jazyk súčasne aj bezkontextový alebo nie.



Čo je to "vlastnosť" RE jazyka

- Rovnako aj to, či RE jazyk obsahuje aspoň nejaký reťazec alebo nie, je taktiež *vlastnosť* RE jazykov.
- T.j. jazyky L_e , L_{ne} predstavujú tiež "vlastnosti" RE jazykov.

Čo je to netriviálna vlastnosť

- Ak je daná taká vlastnosť, že ju spĺňajú **alebo všetky rekurzívne vyčísliteľné jazyky**, alebo **žaden RE jazyk**, potom takúto vlastnosť nazývame **triviálnou vlastnosťou**.
- V opačnom prípade túto vlastnosť nazývame **netriviálnou** - t.j. v takom prípade existuje minimálne jeden RE jazyk, ktorý túto vlastnosť spĺňa a minimálne jeden jazyk, ktorý túto vlastnosť nespĺňa.



Vlastnosti RE jazykov a TS

- Keďže typický RE jazyk je nekonečný, to, či spĺňa nejakú vlastnosť nevieme posúdiť tak, že by sme prehľadali všetky jeho reťazce.
- Pre posúdenie toho, či jazyk spĺňa nejakú vlastnosť potrebujeme jeho **konečnú reprezentáciu**.
- **Konečnou reprezentáciou** RE jazyka L je práve Turingov stroj M , ktorý ho akceptuje $L = L(M)$.
- Zároveň vieme, že každý Turingov stroj M_i vieme popísať **konečným reťazcom** w_i , ktorý ho kóduje.
- Preto, ak zisťujeme, či RE jazyky spĺňajú nejakú vlastnosť, vieme to previesť na otázku, či túto vlastnosť spĺňajú jazyky $L(M_i)$ príslušných Turingových strojov.



Vlastnosti RE jazykov a TS

- Ak $\mathcal{P}$ je vlastnosť RE jazykov, tak $L_{\mathcal{P}}$ je množina kódov Turingových strojov M_i takých, že $L(M_i)$ je jazyk s vlastnosťou $\mathcal{P}$.
- Preto, ak uvažujeme rozhodnuteľnosť vlastnosti $\mathcal{P}$, tak sa bavíme o rozhodnuteľnosti jazyka $L_{\mathcal{P}}$.



Riceova veta

Pre rekurzívne vyčísliteľné jazyky platí Riceova veta

Veta

Každá netriviálna vlastnosť rekurzívne vyčísliteľných jazykov je nerozhodnuteľná.



Príklad

Uvažujme teda napríklad všetky rekurzívne vyčísliteľné jazyky nad abecedou $\{0, 1\}$ a vlastnosť:

- $\mathcal{P} : L$ je bezkontextový jazyk
- Čiže $L_{\mathcal{P}} = \{M \mid M \text{ je TS akceptujúci bezkontextový jazyk}\}$
- Ide o netriviálnu vlastnosť, pretože určite existuje RE jazyk, ktorý je bezkontextový - napríklad $\{0^n 1^n \mid n \geq 1\}$ a zároveň existuje RE jazyk, ktorý nie je bezkontextový - napríklad $\{ww \mid w \in \{0, 1\}^*\}$.
- Podľa Riceovej vety je teda to, či je daný RE jazyk zároveň aj bezkontextový, nerozhodnuteľný problém.
- Znamená to teda, že neexistuje jeden Turingov stroj, ktorý by dokázal pre ľubovoľný RE jazyk (ľubovoľný TS) povedať, či zároveň existuje / neexistuje zásobníkový automat (bezkontextová gramatika), ktorý akceptuje ten istý jazyk.



Dôkaz Riceovej vety

Chceme dokázať, že každá netriviálna vlastnosť $\mathcal{P}$ RE jazykov je nerozhodnuteľná.

Nech teda $\mathcal{P}$ je nejaká netriviálna vlastnosť RE jazykov. Navyše predpokladajme, že prázdny jazyk $L = \emptyset$ vlastnosť **nesplňa**.

- Keďže vlastnosť je netriviálna a nespĺňa ju ani prázdny jazyk, musí existovať nejaký neprázdny jazyk $L \in \mathcal{P}$, ktorý ju splňa.
- Nech M_L je Turingov stroj tohto jazyka L , teda $M_L \in L_{\mathcal{P}}$, $L(M_L) = L$.
- Ukážeme, že v tomto prípade je určite L_u redukovateľný na $L_{\mathcal{P}}$, čím dokážeme, že $L_{\mathcal{P}}$ je nerozhodnuteľný.



Dôkaz Riceovej vety

Redukcia L_U na $L_{\mathcal{P}}$, $\emptyset \notin \mathcal{P}$:

- Musíme ukázať, že každý pár (M, w) , kde $w \in L(M)$ sa dá previesť na nejaký Turingov stroj M' , ktorého jazyk $L(M')$ spĺňa vlastnosť $\mathcal{P}$.
- Zároveň, že každý pár (M, w) , kde $w \in L(M)$ sa dá previesť na nejaký Turingov stroj M' , ktorého jazyk $L(M')$ nespĺňa vlastnosť $\mathcal{P}$.



Uvažujme nasledovnú konštrukciu:

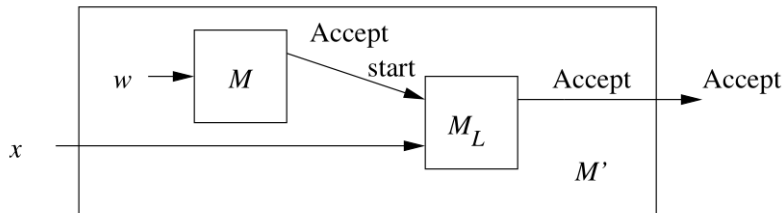


Figure: Redukcia (M, w) na TS M' , [1]

Vysvetlenie:

- TS M' najprv simuluje činnosť M pre vstup w . Ak M akceptuje w , tak potom spracuje svoj vlastný vstup x pomocou TS M_L , teda M' zároveň akceptuje všetky tie vstupy, ktoré akceptuje M_L .
- Ak $w \in L(M)$, potom M' akceptuje jazyk L , ktorý spĺňa $\mathcal{P}$, lebo $L(M') = L(M_L) = L \in \mathcal{P}$
- V prípade, že M neakceptuje w , potom TS M' ignoruje všetky svoje vstupy a žiaden neakceptuje, teda akceptuje prázdny jazyk.
- Ak $w \notin L(M)$, potom M' neakceptuje žiaden reťazec - teda akceptuje jazyk $\emptyset$, o ktorom predpokladáme, že $\mathcal{P}$ nespĺňa, lebo $L(M_L) = \emptyset \notin \mathcal{P}$.



Zostrojenie Turingovho stroja M' z dvojice (M, w) je realizovateľné algoritmicky, pričom Turingov stroj M' vznikne nasledovne:

1. Simuluje činnosť M na vstupe w , napríklad pomocou Univerzálneho Turingovho stroja. Všimnite si, že (M, w) **nie sú** vstup M' , ale predstavujú napríklad "konštanty" do neho zabudované.
2. V prípade, že M neakceptuje w , tak M' svoj vstup ignoruje a nikdy nevráti ÁNO, teda nič neakceptuje.
3. V prípade, že M akceptuje w , tak M' simuluje činnosť M_L na svojom reálnom vstupe x . Teda, ak M_L akceptuje x , ak M' akceptuje x a teda $L(M_L) = L(M')$.
 - Ak $(M, w) \in L_u$, tak $M' \in L_{\mathcal{P}}$.
 - Ak $(M, w) \notin L_u$, tak $M' \notin L_{\mathcal{P}}$.

Teda ide o redukciu L_u na $L_{\mathcal{P}}$. A keďže L_u nie je rozhodnuteľný problém, ani $L_{\mathcal{P}}$ nie je rozhodnuteľný problém.



Dôkaz

Dokázali sme teda, že ak je netriviálna vlastnosť $\mathcal{P}$ taká, že ju nespĺňa prázdny jazyk, určite je nerozhodnuteľná. Čo však v prípade, že ju prázdny jazyk spĺňa?

Nech teda $\mathcal{P}$ je nejaká netriviálna vlastnosť RE jazykov a predpokladajme, že prázdny jazyk $L = \emptyset$ vlastnosť **spĺňa**.

- Ak $\mathcal{P}$ je netriviálna vlastnosť, ktorú zároveň spĺňa aj prázdny jazyk, potom jej **doplnková vlastnosť** $\overline{\mathcal{P}}$ je tiež netriviálna vlastnosť, ktorú tvorí množina RE jazykov, ktoré vlastnosť $\mathcal{P}$ nespĺňajú!
- Teraz by sme, rovnako ako v predošlom prípade, kde prázdny jazyk nespĺňal $\mathcal{P}$ vedeli dokázať, že L_u je redukovateľný aj na $L_{\overline{\mathcal{P}}}$.
- Teda vieme dokázať, že vlastnosť $\overline{\mathcal{P}}$ je **nerozhodnuteľná**.



Dôkaz

- Ak $\overline{\mathcal{P}}$ je netriviálna nerozhodnuteľná vlastnosť, tak potom jej príslušný jazyk, $L_{\overline{\mathcal{P}}}$, ktorý tvoria tie Turingove stroje, ktoré akceptujú RE jazyky, ktoré vlastnosť $\mathcal{P}$ nespĺňajú, určite **nie je rekurzívny**.
- A keďže $L_{\overline{\mathcal{P}}}$ nie je rekurzívny, ani jeho doplnok, jazyk $L_{\mathcal{P}}$ **nemôže byť rekurzívny**.
- Teda ani vlastnosť $\mathcal{P}$ taká, ktorú v tomto prípade spĺňa ak prázdny jazyk, **nemôže byť rozhodnuteľná**.



Dôkaz

Podarilo sa nám teda dokázať, že ak máme **ľubovoľnú netriviálnu vlastnosť** RE jazykov, tak určite **nie je rozhodnuteľná**.

Teda sme dokázali Riceovu vetu.

Význam Riceovej vety

- Uvedomte si, že Riceova veta hovorí o tom, že netriviálna vlastnosť RE jazykov je nerozhodnuteľná - teda že **neexistuje algoritmus**, resp. **Turingov stroj, ktorý vždy skončí**, ktorý by daný jazyk vedel rozhodnúť.
- Avšak nedokáže určiť, či je daný problém rekurzívne vyčísliteľný - t.j. že síce existuje Turingov stroj, ale len taký, že skončí v konečnom čase, ak odpoveď je áno - alebo či daný problém nie je vôbec rekurzívne vyčísliteľný - t.j. že Turingov stroj neexistuje.



Význam Riceovej vety

- Riceova veta hovorí, že **všetky problémy**, ktoré sa týkajú **vlastností jazykov** Turingovych strojov, sú **nerozhodnuteľné**.
- Medzi takéto problémy patrí napríklad:
 - Je jazyk akceptovaný Turingovým strojom prázdny?
 - Je jazyk akceptovaný Turingovým strojom konečný?
 - Je jazyk akceptovaný TS regulárny?
 - Je jazyk akceptovaný TS bezkontextový?
- Avšak Riceova veta **nehovorí, že každý problém o Turingovych strojoch je nerozhodnuteľný**.
- Napríklad problémy o **štruktúre TS** rozhodnuteľné byť môžu:



Význam Riceovej vety

- Rozhodnuteľné problémy o Turingových strojoch:
 - Má Turingov stroj 5 stavov? - vieme rozhodnúť, stačí sa pozrieť, koľko stavov sa nachádza v prechodoch.
 - Má Turingov stroj 5 prechodov? - vieme rozhodnúť, stačí sa pozrieť, koľko prechodov má prechodová funkcia
 - Existuje taký vstup Turingovho stroja, že naň Turingov stroj vykoná aspoň 5 krokov výpočtu? - vieme rozhodnúť tak, že simulujeme konečný počet situácií, kde na páske sú všetky kombinácie 5 symbolov vstupu alebo menej a sledujeme všetky možné správania po dobu 5 krokov výpočtu.

Dôsledky Riceovej vety pre počítačové programy

Prečo je Riceova veta dôležitá z praktického hľadiska?

- Každý Turingov stroj si vieme predstaviť ako nejaký počítačový program.
- Spustenie Turingovho stroja s nejakým vstupom si vieme predstaviť ako spustenie programu s nejakým vstupom.
- Vlastnosti Turingovho stroja z hľadiska počtu stavov, počtu prechodov sa rovnajú **syntaktickým vlastnostiam** - ekvivalentne napríklad koľko premenných má program, koľko vetvení sa fyzicky nachádza v zdrojovom kóde, a podobne.
- Teda, **ako Turingov stroj / počítačový program vyzerá**

Dôsledky Riceovej vety pre počítačové programy

Prečo je Riceova veta dôležitá z praktického hľadiska?

- Tvrdenia o **jazyku, ktorý TS akceptuje** sa rovnajú **sémantickým vlastnostiam**, t.j. **čo Turingov stroj vykonáva**
- Ekvivalentne - čo konkrétne vykonáva počítačový program? Aký výpočet realizuje? Ako očakávame, že sa bude program správať pre jednotlivé vstupy z hľadiska výstupu?

Dôsledky Riceovej vety pre počítačové programy

- Riceova veta hovorí, že akékoľvek tvrdenie o **správaní** Turingových strojov z hľadiska toho, čo vykonávajú, je nerozhodnuteľné.
- Analogicky teda akákoľvek otázka o správaní počítačových programov je nerozhodnuteľná!
- Napríklad otázka: Vypíše počítačový program ako prvý výpis na obrazovku `hello, world` je nerozhodnuteľná.
- Alebo otázka: Zavolá počítačový program počas svojej činnosti funkciu `f()` je nerozhodnuteľná.
- Teda **neexistuje jeden univerzálny počítačový program**, ktorý by dokázal spracúvať rôzne programy a v konečnom čase **rozhodnúť otázky o ich správaní!**



That's all folks!

To je pre tento semester všetko, ďakujem Vám všetkým za prejavenú priazeň a
želám veľa šťastia v skúškovom období.



Použitá literatúra

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