1 Přednáška 20. září

Obsah: *Opakování: formalizace pojmů výpočetní proces a (rozhodovací) algoritmický problém, Turingův stroj jako model výpočtu, jeho varianty, jeho výpočet, rozpoznatelné (rekurzivní) a rozhodnutelné (částečně rekurznivní) jazyky, univerzální turingův stroj a simulace, existence a příklady nerozpoznatelných a nerozhodnutelných problémů. Turingovská a many-to-one redukce.*

Literatura: [1], kapitoly 3, 4, 5, 6

Intuitivně chápaný výpočetní proces formalizujeme pomocí Turingova stroje (TS). Neformálně je TS stavovým strojem, který může zapisovat a číst symboly z políček jednostranně nekonečné pásky. Formálně je dán:

- konečnou množinou stavů Q;
- vstupní abecedou Σ, která neobsahuje symbol $\sqcup$ pro prázdné políčko pásky;
- páskovou abecedou Γ, pro kterou $\Sigma \subseteq \Gamma$, $\sqcup \in \Gamma$;
- přechodovou funkcí $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$.
- výběrem speciálních stavů z Q: počáteční stav q_o, přijímací stav q_{accept} a zamítací stav q_{reject}.

Situace, ve které se TS nachází, je dána aktuálním stavem, obsahem pásky a aktuálně čteným políčkem. Vše můžeme zachytit např. pomocí řetězce wqw', kde $w \in \Gamma^*$ je obsah pásky od začátku až po poslední políčko před aktuálně čteným, $q \in Q$ je aktuální stav, a $w' \in \Gamma^*$ je obsah pásky začínající aktuálně čteným políčkem, přičemž na konci vynecháme nekonečnou posloupnost symbolů $\sqcup$.

Z wqw' provedeme jeden krok tak, že se podíváme na $\delta(q, w'[1])$, a podle výsledku nastavíme nový aktuální stav, zapíšeme znak na aktuální políčko pásky, a za aktuální políčko pásky vybereme to, které je vlevo nebo vpravo. (na prvním políčku se neposunujeme vlevo, ale aktuální políčko ponecháme stejné). Pomocí $c_1 \vdash c_2$ značíme, že TS přejde z konfigurace c_1 do c_2, $\vdash^+$ značí tranzitivní uzávěr a $\vdash^*$ tranzitivní a reflexivní uzávěr relace $\vdash$.

Pro $x \in \Sigma^*$ je počáteční konfigurace TS q_0x. Řekneme, že TS přijímá x, pokud $q_0x \vdash^* wq_{\text{accept}}w'$; TS zamítá x, pokud $q_0x \vdash^* wq_{\text{reject}}w'$ (pro nějaké řetězce w, w'). Další možností je, že TS pro x cyklí, tj. nikdy nedojde do přijímacího nebo zamítacího stavu. Předpokládáme, že když TS přijímá nebo zamítá, jeho výpočet dále nepokračuje: říkáme tedy, že zastaví. Množinu všech řetězců přijímaných TS T budeme značit $L(T)$.

Výpočet TS pro x lze vizualizovat jako posloupnost konfigurací.

Rozhodnovací algoritmický problém formalizujeme pomocí jazyka: jako množinu instancí, pro které je odpověď ANO, zakódovaných jako řetězce nad Σ.

TS T *rozpoznává* jazyk L, pokud všechny $x \in L$ TS T přijímá a $x \in L$ zamítá nebo pro ně cyklí. TS T *rozhoduje* jazyk L, pokud všechny $x \in L$ TS T přijímá a všechny $x \notin L$ zamítá.

Varianty TS . Existuje mnoho variant TS (například více pásek, oboustraně nekonečná páska atd.). Pro nás je důležitá varianta *nedeterministického TS* (NTS). V něm má přechodová funkce tvar $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,P\}}$. NTS může mít tedy na výběr více konfigurací, do kterým může v jednom kroku přejít. Výpočet NTS tak lze přirozeně zachytit jako strom, kde uzly jsou konfigurace a pomomky konfigurace c jsou právě ty konfigurace, do kterých může NTS přejít z c pomocí jednoho kroku. Pro $x \in \Sigma$ je kořenem takového stromu q_0x. NTS přijímá x, pokud ve stromu existuje větev, která končí konfigurací obsahující q_{accept}.

Věta 1. *Ke každému NTS* T *existuje TS, který přijímá stejný jazyk.*

Proof. Lze sestavit deterministický TS, který dělá průchod do šířky stromem konfigurací NTS T. Pokud narazí na přijímací konfiguraci, pak přijímá. Pokud NTS pro všechny vstupy ve všech větvích výpočtu zastaví, sestavený TS potom rozhoduje $\mathsf{L}(\mathsf{T})$. Detaily: [1], kapitola 3. □

Můžeme tedy rozšířit definice rozpoznávání a rozhodování jazyků i na NTS.

Existence nerozhodnutelných a nerozpoznatelných jazyků

TS je konečný objekt (všechny jeho složky jsou konečné). TS T lze zakódovat pomocí řetězce $\langle\mathsf{T}\rangle$ nad vhodně zvolenou abecedou. Lze navíc dokázat, že existuje TS U, který přijímá $(\langle\mathsf{T}\rangle, w)$, právě když T přijímá w, zamítá $(\langle\mathsf{T}\rangle, w)$, právě když T zamítá w, a zacyklí se, když se zacyklí T.[1] TS U tedy simuluje výpočet TS T. Navíc můžeme předpokládat, že libovolný TS může používat U jako svoji součást a simulovat tak libovolný TS (tedy volat simulaci jako podprogram): například i upravit popis TS ze vstupu nebo vytvořit nový TS, a ty poté simulovat. Existuje zakódování TS takové, že existuje TS D, který přijímá $\mathtt{x}$, právě když $\mathtt{x}$ je zakódováním nějakého TS. Zbývá otázka, co provede U výše (nebo libovolný jiný TS očekávající na vstup zakódování TS), pokud dostane na vstupu řetězec $\mathtt{y}$, který není zakódováním nějakého TS. V takovém případě budeme předpokládat, že $\mathtt{y}$ kóduje TS, který zamítá všechny řetězce. Každý řetězec tak tedy kóduje nějaký TS. Technické detaily lze nalézt v [1].

Množina všech TS je spočetná (každý TS lze zakódovat do konečného řetězce), kdežto množina všech jazyků je nespočetná (lze dokázat Cantorovou diagonalizační metodou). Existují tedy nerozpoznatelné jazyky. Ve zbytku přednášky se podíváme na konkrétní příklady.

Uvážíme jazyk $\mathrm{ATM} = \{(\langle\mathsf{M}\rangle, w) \mid \mathsf{M} \text{ přijímá } w\}$. Hned vidíme, že ATM je rozpoznatelný. TS, který jej rozpoznává, pro vstup $(\langle\mathsf{M}\rangle, w)$ simuluje TS M pro vstup w, a přijímá, pokud M přijímá. Problém ATM bývá někdy někdy označován jako *halt problém*.

Věta 2. ATM *není rozhodnutelný.*

Proof. Sporem. Předpokládejme, že existuje TS H, který rozhoduje ATM. Sestavíme TS D, který pro vstup $\mathtt{x}$

1. Simuluje činnost H pro $(\mathtt{x}, \mathtt{x})$.
2. Pokud H přijme, pak D zamítne. Pokud H zamítne, pak D přijme.

Protože H rozhoduje ATM, vždycky zastaví a D také vždycky zastaví. Nyní se podívejme, co se stane při výpočtu D pro vstup $\langle\mathsf{D}\rangle$. Pokud D přijímá $\langle\mathsf{D}\rangle$, pak H zamítá $(\langle\mathsf{D}\rangle, \langle\mathsf{D}\rangle)$, což ovšem znamená, že D zamítá $\langle\mathsf{D}\rangle$. Podobně dojdeme ke sporu pokud začneme s tím, že D zamítá $\langle\mathsf{D}\rangle$. □

Věta 3. *Pokud jsou jazyky* L *a* $\overline{\mathsf{L}}$ *rozpoznatelné, jsou i rozhodnutelné.*

Proof. Zjevně, je-li jazyk rozhodnutelný, je rozhodnutelný i jeho doplněk. Stačí tedy ukázat, že je rozhodnutelný L. Z předpokladů věty máme, že existují TS T_1 a T_2 tak, že $\mathsf{L} = \mathsf{L}(\mathsf{T}_1)$ a $\overline{\mathsf{L}} = \mathsf{L}(\mathsf{T}_2)$. Sestavíme TS D, který pro vstup $\mathtt{x}$

1. simuluje krok výpočtu T_1 (pro vstup $\mathtt{x}$),
2. simuluje krok výpočtu T_2 (pro vstup $\mathtt{x}$),
3. pokud v předchozím T_1 přijímá, pak D přijímá, pokud v předchozím T_2 přijímá, pak D zamítá. Jinak pokračuje krokem 1.

TS D paralelně simuluje výpočty obou T_1 a T_2 pro $\mathtt{x}$. Pokud $\mathtt{x} \in \mathsf{L}$, pak se jednou přijímá T_1 (a ne T_2), jinak jednou přijímá T_2 (a ne T_1). TS D tedy skutečně rozhoduje L. □

[1] pomocí $(\mathtt{x}, \mathtt{y}, \dots)$ značíme zakódování n-tice řetězců pomocí jiného řetězce.

Důsledkem předchozích dvou vět je, že $\overline{\text{ATM}}$ není rozpoznatelný.

Další příklady nerozhodnutelných problémů.

Věta 4. *Jazyk* $\text{EMPTY} = \{\langle M \rangle \mid L(M) = \emptyset\}$ *není rozhodnutelný.*

Proof. Sporem. Předpokládejme, že existuje TS R rozhodující EMPTY. Potom existuje TS S, který pro vstup $(\langle M \rangle, w)$:

1. Zkonstruuje TS D, který pro vstup x
   - pokud $x \neq w$, zamítne
   - jinak simuluje M pro w a pokud M přijme, pak D také přijme.
2. Simuluje R pro vstup $\langle D \rangle$. Pokud R přijme, pak S také přijme. Pokud R zamítne, tak S také zamítne.

Klíčové pozorování je, že pokud TS M přijímá w, pak $L(D) = \{w\}$, jinak $L(D) = \emptyset$. TS S tak rozhoduje ATM, což je spor. □

Věta 5. *Jazyk* $\text{EQUAL} = \{(\langle M_1 \rangle, \langle M_2 \rangle) \mid L(M_1) = L(M_2)\}$ *není rozhodnutelný.*

Proof. Sporem. Předpokládejme, že existuje TS R, který rozhoduje EQUAL. Potom TS S, který pro vstup $\langle M \rangle$

1. Simuluje R pro dvojici (M, N), kde N je triviální TS, který v prvním kroku zamítá.
2. Pokud R zamítá, pak S zamítá. Pokud R přijímá, pak S přijímá.

Očividně TS S rozhoduje EMPTY a to je spor. □

Koncept redukce

Oracle pro jazyk L je externí zařízení, které je schopné v jednom kroku rozhodnout, jestli $x \in L$. *Oracle TS* je TS, který má k dispozici oracle a může jej používat (např. někam na pásku zapíše x a přechodem do speciálního stavu položí oracle dotaz, odpověď se TS dozví z toho do jakého stavu ho oracle přesune). Pomocí značení M^B dáváme explicitně najevo, že M je oracle TS s přístupem k oracle pro jazyk B.

Řekneme, že jazyk A je Turingovsky redukovatelný na jazyk B, značeno $A \leq_T B$ pokud existuje oracle TS M^B, který rozhoduje A.

Věta 6. *Pokud* $A \leq_T B$ *a* B *je rozhodnutelný, je i* A *rozhodnutelný.*

Proof. Volání oracle v oracle TS existujícímu z definice redukce nahradíme simulací TS rozhodujícího B. Dostaneme tak TS rozhodující A. □

Důsledkem předchozí věty je, že pokud $A \leq_T B$ a A je nerozhodnutelný, je i B nerozhodnutelný. Všimněme si, že v důkazech vět 4 a 5 jsme sestavily oracle TS pro ATM a EMPTY.

Funkce $f : \Sigma^* \to \Sigma^*$ je vyčíslitelná, pokud existuje TS M, který pro každé w zastaví a na pásce zůstane $f(w)$.

Jazyk A je many-to-one redukovatelný na jazyk B (značeno $A \leq_m B$), pokud existuje vyčíslitelná funkce f tak, že pro každé $w \in \Sigma^*$ platí: $w \in A$ právě když $f(w) \in B$.

Rychle vidíme, že $A \leq_m B$ implikuje $A \leq_T B$ (pro vstup w oracle TS spočítá $f(w)$ a použije výsledek jako dotaz oracle pro B, a na základě odpovědi hned přijímá nebo zamítá.) Pro many-to-one redukce tak platí věta 6. Navíc ovšem máme

Věta 7. *Pokud* $A \leq_m B$ *a* B *je rozpoznatelný, pak i* A *je rozpoznatelný.*

Pro obecnou Turingovskou redukci toto neplatí. Triviálně totiž máme $\overline{\text{ATM}} \leq_T \text{ATM}$, ale přitom ATM je rozpoznatelný a $\overline{\text{ATM}}$ není.

2 Přednáška 27. září

Obsah: *enumerace, věta o rekurzi, Riceova věta.*

Literatura: [1], kapitoly 3, 6; [2] kapitola 8.4

Enumerace

Enumerátor E je TS obohacený o schopnost tisknout řetězce nad Σ (tisknout může například pomocí přechodu do speciálního stavu). *Enumerátor* při výpočtu nemusí zastavit, ani neuvažujeme jeho vstup. Zajímá nás pouze množina řetězců, které vytiskne, kterou značíme $L(E)$.

Enumerátor může vytisknout řetězec $x \in L$ vícekrát, ovšem každý řetězec $x \in L$ vytiskne po konečném počtu kroků. Zjevně existuje enumerátor pro jazyk Σ^* (může řetězce tisknou vzestupně podle délky, a v rámci jedné délky lexikograficky).

Věta 8. *Jazyk* L *je rozpoznatelný, právě když existuje enumerátor* E *tak, že* $L = L(E)$.

Proof. $\Rightarrow$: Nechť M rozpoznávající L. Sestavíme enumerátor E, který pro $i = 1, 2, 3, ...$:

1. Simuluje prvních i kroků výpočtu M pro vstup x_i. Tady je x_i řetězec, který jako i-tý v pořadí vytiskne enumerátor pro Σ^*. E tento enumerátor simuluje, aby získal $x_1, x_2, \ldots, x_i$
2. Pokud v předchozím bodu M přijme x_i pro některé i, pak E vytiskne x_i.

Vidíme, že pro $x \in L$ enumerátor E jednou simuluje M pro dostatečný počet kroků, aby M přešel do přijímací konfigurace a E tedy vytiskne x. Na druhou stranu, pokud $x \notin L$, enumerátor E řetězec x nikdy nevytiskne.

$\Leftarrow$: Předpokládejme, že E enumeruje L. Potom TS M, který pro vstup x:

1. simuluje E a čte řetězce, které tiskne.
2. pokud se řetězec vytištěný E rovná x, pak M přijímá.

Protože E každý řetězec z L tiskne po konečném počtu kroků, TS M přijímá právě jazyk L. □

Věta o rekurzi

Snadno ověříme následující tvrzení o existenci TS. Pro každé $x \in \Sigma^*$ existuje TS PRINT[x], který pro vstup w

1. smaže w z pásky,
2. zapíše na pásku x, aktuální políčko nastaví na první políčko na pásce.

Existuje i varianta, PRINTSKIP[x], která slovo w nesmaže a x zapíše až za něj, přičemž aktuálním políčkem nastaví první znak x.

Pro každé $x \in \Sigma$ tak existuje i TS MACRO, který pro vstup x zastaví a na pásce zůstane $\langle$PRINT[x]$\rangle$. Existuje i varianta MACROSKIP, která takto vytvoří $\langle$PRINTSKIP[x]$\rangle$.

Pro TS A a B označíme pomocí $A \to B$ Turingův stroj, který pro vstup x

1. provádí A pro vstup x
2. v momentě, kdy A zastaví, pokračuje s výpočtem B, přičemž je z kroku 1 ponechán obsah pásky i aktuální políčko.

Uvažme TS B, který pro vstup $\langle M \rangle$:

1. spustí MACRO pro vstup $\langle M \rangle$, někde na pásce tak má zapsáno $\langle \text{PRINT}[\langle M \rangle] \rangle$.
2. Sestaví a na pásku zapíše popis Turingova stroje $\text{PRINT}[\langle M \rangle] \to \langle M \rangle$. Popis je na začátku pásky, nic jiného tam není.

Když se nyní podíváme na Turingův stroj $\text{PRINT}[\langle B \rangle] \to B$, zjistíme, že na když zastaví (pro libovolný vstup), zůstane na pásce jeho vlastní popis. Vidíme tak příklad TS, který umí vytisknout svoje vlastní zakódování a toto zakódování tak musí znát. Pokud předchozí myšlenku drobně upravíme tak, abychom mohli zpracovat i vstup, dostaneme následující větu, která říká, že předpokládat, že TS znají svůj popis můžeme obecně.

Věta 9 (O rekurzi)**.** *Nechť* T *je TS vyčíslující funkci* $t : \Sigma^* \times \Sigma^* \to \Sigma^*$*. Potom existuje TS* R *vyčíslující funkci* $r : \Sigma^* \to \Sigma^*$ *tak, že pro všechna* $w \in \Sigma^*$ *máme* $r(w) = t(w, \langle R \rangle)$

Proof. (Důkaz je velmi podobný konstrukci nad větou)

Uvažme TS B, který pro vstup $w\langle M \rangle$

1. spustí MACROSKIP pro $\langle M \rangle$ a někde na pásce tak má $\langle \text{PRINTSKIP}[\langle M \rangle] \rangle$.
2. vytvoří popis stroje $\text{PRINTSKIP}[\langle M \rangle] \to \langle M \rangle$ tak, aby když B zastaví, byl obsah pásky $w\langle \text{PRINTSKIP}[\langle M \rangle] \to \langle M \rangle \rangle$ a aktuálním políčkem bylo první políčko pásky.

Cílený TS R je potom $\text{PRINTSKIP}[\langle B \to T \rangle] \to (B \to T)$. Pro vstup w tohoto stroje je v momentě, kdy spouštíme T na pásce $w\langle R \rangle$, což je přesně to, co jsme chtěli dokázat.

□

Řekneme, že TS M je minimální, pokud neexistuje TS N s kratším zakódováním, který je M ekvivalentní (oba přijímají a zamítají stejné řetězce).

Věta 10. *Jazyk* $\text{MINTM} = \{\langle M \rangle \mid M \text{ je minimální}\}$ *není rozpoznatelný.*

Proof. Sporem. Předpokládejme, že MINTM rozpoznatelný je a že existuje enumerátor E tak, že $L(E) = \text{MINTM}$. Potom existuje TS C, který pro vstup x

1. zjistí délku svého zakódování (viz věta o rekurzi),
2. simuluje E, dokud nevytiskne $\langle D \rangle$, které je delší než $\langle C \rangle$,
3. Simuluje D pro x (a chová se stejně jako D).

Protože MINTM je nekonečná množina, existuje TS s delším zakódováním než C. Zjevně, TS C je ekvivalentní D. Současně má ale $\langle D \rangle$ delší zakódování než C, a tedy $\langle D \rangle \notin \text{MINTM}$. To je ale spor. □

Riceova věta

Uvažme zobrazení $P : \Sigma^* \to \{0, 1\}$. Řekneme, že P je *vlastnost rozpoznatelných jazyků*, pokud pro všechny dvojice TS M_1, M_2 máme, že $L(M_1) = L(M_2)$ implikuje $P(\langle M_1 \rangle) = P(\langle M_2 \rangle)$. Vlastnost P je *netriviální*, pokud existují x_1, x_2 tak, že $P(x_1) \neq P(x_2)$.

Pro TS rozpoznávající stejný jazyk je vlastnost P stejná, můžeme ji tedy chápat jako vlastnost přijímaných jazyků, nikoliv TS. Například

- Mějme $P(x) = 1$ právě když x kóduje TS, který přijímá konečný jazyk. Potom je P netriviální vlastnost rozpoznatelných jazyků.
- Mějme $P(x) = 1$ právě když x kóduje TS s méně než 100 stavy. Potom P není vlastnost rozpoznatelných jazyků.

Věta 11. *Nechť* P *je netriviální vlastnost rozpoznatelných jazyků. Potom je jazyk*

$$L_P = \{w \mid P(w) = 1\}$$

nerozhodnutelný.

Proof. Předpokládejme, že existuje netriviální vlastnost rozpoznatelných jazyků P taková, že $P(x) = 0$ pro každé x, které kóduje TS přijímající prázdný jazyk. Z netriviality P pak plyne existence rozpoznatelného jazyka A takového, že pro každý TS M, pro který $L(M) = A$, máme $P(\langle M \rangle) = 1$. Protože A je rozpoznatelný, existuje TS K tak, že $L(K) = A$.

Sestavíme TS R, který pro vstup $(\langle M \rangle, w)$ sestaví zakódování TS M' takového, že

$$L(M') = \begin{cases} A & \text{pokud } M \text{ přijímá } w, \\ \emptyset & \text{jinak.} \end{cases}$$

Popis fungování R vynecháme, bude jasný z popisu fungování M'. Tento TS pro vstup x

1. simuluje M pro w,
2. pokud M zamítá, tak M' také zamítá,
3. simuluje K pro x (a chová se stejně jako K).

Vidíme, že pokud se M zacyklí nebo zamítá x, pak M' nepřijímá žádný řetězec, pokud M přijímá w, tak M' přijímá $L(K) = A$.

Nyní ukážeme ATM $\leq_T L_P$. Požadovaný oracle TS pro vstup $(\langle M \rangle, w)$:

1. Simuluje R pro $(\langle M \rangle, w)$ a získá tak $\langle M' \rangle$.
2. Zavolá oracle pro L_P s dotazem $\langle M' \rangle$.
3. Pokud oracle přijímá, pak také přijímá. Jinak zamítá.

Korektnost redukce plyne z předchozí diskuze. □

3 Přednáška 4. října

Obsah: *Vztah rekurzivních a částečně rekurzivních jazyků k jazykům Chomského hierarchie.*

Literatura: [2] kapitola 9.

Gramatika G je dána

- množinou neterminálů N,
- množinou terminálů Σ,
- množinou pravidel $P = \{\alpha \to \beta \mid \alpha, \beta \in (N \cup \Sigma)^*, \alpha \neq \epsilon\}$,
- počátečním neterminálem $S \in N$.

(*Pozn. ke značení: tradičně značíme neterminály velkými písmeny, terminály malými písmeny; více pravidel se stejnou pravou stranou píšeme na jeden řádek a oddělujeme pomocí svislítka.*)

Řetězec z $(N \cup \Sigma)^*$ nazýváme větná forma. Odvozování v gramatice formalizujeme pomocí binární relace $\vdash$ mezi větnými formami: $\gamma\alpha\omega \vdash \gamma\beta\omega$ pokud existuje pravidlo $\alpha \to \beta \in P$; pomocí $\vdash^+, \vdash^*$ značíme tranzitivní a reflexivně tranzitivní uzávěry relace $\vdash$. Jazyk generovaný gramatikou G je $L(G) = \{w \in \Sigma^* \mid S \vdash^* w\}$.

TS a gramatika typu 0

Všechny gramatiky jsou typu 0 (nemáme žádné omezení na tvar pravidel).

Věta 12. *Pokud je G gramatika typu 0, je jazyk $L(G)$ rozpoznatelný.*

Proof. Ukážeme, že existuje TS M tak, že $L(M) = L(G)$. To uděláme tak, že sestavíme nedeterministický TS R tak, že $L(R) = L(G)$. Tento NTS funguje pro vstup w následovně:
(*předpokládejme, že má dvě pásky, na první pásce má zapsáno w a na druhou pásku si zapíše startovací symbol S.*)

1. Nedeterministicky vybere pozici ve větné formě zapsané na druhé pásce.
2. Nedeterministicky vybere pravidlo z gramatiky.
3. Zkontroluje, zda jde vybrané pravidlo aplikovat na vybraném místě druhé pásky. Pokud ano, pak toto pravidlo aplikuje.
4. Ověří, jestli se větná forma na druhé pásce rovná řetězci na první pásce, pokud ano, tak přijímá. Pokud ne, tak přejde ke kroku 1.

Vidíme, že pokud R v kroku 3 úspěšně aplikuje pravidlo, provádí vlastně odvození podle $\vdash$. Pokud tedy $w \in L(G)$, pak existuje $S \vdash^* w$ a tedy existuje výpočetní větev stroje R, která končí přijetím a máme tak $w \in L(R)$. Současně žádná výpočetní větev, která neodpovídá odvození w z S, nepřijímá. □

Věta 13. *Pokud je L rozpoznatelný, pak existuje gramatika G tak, že $L(G) = L$.*

Proof. Jazyk L je rozpoznatelný, existuje tak TS, který jej rozpoznává.
Idea: sestavíme gramatiku G tak, že odvozování simuluje kroky TS. Přitom větná forma zhruba odpovídá konfiguraci TS. Musíme přitom vyřešit technické problémy, např. jak vygenerovat počáteční konfiguraci, z přijímací konfiguraci musíme odvodit přijímaný řetězec, musíme nějak reprezentovat páskovou abecedu atd.

Jako terminály bereme vstupní abecedu TS (značíme Σ). Neterminály využijeme k reprezentaci konfigurace a k pamatování si vstupního řetězce. Máme tak:

$$N = (\Sigma \cup \{\epsilon\}) \times \Gamma \cup \{A_1, A_2, A_3\} \cup Q.$$

V první komponentě složených neterminálů si budeme pamatovat symboly vstupního řetězce, v druhé komponentě obsah políčka na pásce TS. Neterminál z Q slouží k pamatování si aktuálního stavu. Ostatní symboly jsou pomocné.

Pravidla k vygenerování počáteční konfigurace:

$$\begin{aligned}
A_1 &\to q_0 A_2 && q_0 \text{ je počáteční stav} \\
A_2 &\to [a, a] A_2 && \text{pro každé } a \in \Sigma \\
A_2 &\to A_3 \\
A_3 &\to [\epsilon, \sqcup] A_3 \\
A_3 &\to \epsilon
\end{aligned}$$

Tato pravidla jsou schopna vygenerovat počáteční konfiguraci TS s kusem pásky, který je dostatečný pro všechny konfigurace v přijímacím výpočtu. (*Pomocí prvních dvou pravidel připravíme vstupní řetězec, potom pomocí čtvrtého pravidla nafoukneme konfiguraci podle potřeby.*) Pro vstup $w = w_1 w_2 \dots w_n$ tak vygenerují větnou formu

$$q_0[w_1, w_1][w_2, w_2] \dots [w_n, w_n][\epsilon, \sqcup] \dots [\epsilon, \sqcup],$$

kde symbol $[\epsilon, \sqcup]$ je zopakován v dostatečném počtu.

Pravidla simulující výpočet TS:

$$\begin{aligned}
q[a, X] &\to [a, Y] q' && \text{pro } \delta(q, X) = (q', Y, \text{right}) \\
[b, Z] q [a, X] &\to q' [b, Z] [a, Y] && \text{pro } \delta(q, X) = (q', Y, \text{left})
\end{aligned}$$

(*Intuitivně vidíme, že předchozí pravidla simulují výpočet TS; předpokládáme zde, že TS se nepokusí posunout se doleva na prvním políčku pásky, to už víme ze semináře, že můžeme. Formálně bychom udělali důkaz indukcí.*)

Pravidla pro smazání neterminálních symbolů:

$$\begin{aligned}
[a, X] q &\to q a q && \text{pro } a \in \Sigma \cup \{\epsilon\}, X \in \Gamma, q \in Q \\
q [a, X] &\to q a q && \text{pro } a \in \Sigma \cup \{\epsilon\}, X \in \Gamma q \in Q \\
q &\to \epsilon && \text{pro } q = q_{accept}
\end{aligned}$$

Složené neterminální symboly nahradíme pomocí symbolu aktuálního stavu. Symboly aktuálního stavu umíme posledním pravidlem odstranit, pokud je to q_{accept}. Větnou formu skládající se z neterminálů tak dostaneme, pouze pokud TS takový řetězec přijímal.

□

Lineárně omezený automat a kontextově závislá gramatika

Lineárně omezený automat (LBA) je nedeterministický TS, ve kterém platí

- Pásková abeceda obsahuje speciální znaky $\triangleright$ a $\triangleleft$ pro označení začátku a konce pásky, $\triangleright, \triangleleft \notin \Sigma$.
- TS nemůže $\triangleright, \triangleleft$ přepsat jiným znakem, je-li na obsahuje-li akt. políčku $\triangleright$, TS se nemůže posunou doleva. Je-li na aktuálním políčku $\triangleleft$, TS se nemůže posunout doprava.

Jazyk LBA A je definován jako $L(A) = \{x \in \Sigma^* \mid \triangleright x \triangleleft \vdash^* wq_{\text{accept}}w'$ pro nějaké řetězce $w, w'\}$.

Gramatika je *kontextově závislá* (nebo nezkracující), pokud její pravidla jsou ve tvaru $\alpha \to \beta$, kde $|\alpha| \leq |\beta|$.

Věta 14. *Pro každou kontextově závislou gramatiku* G *existuje LBA* A *tak, že* $L(G) = L(A)$.

Proof. Analogicky důkazu Věty 12 chceme, aby LBA simulovalo odvozování z počátečního symbolu gramatiky. Musíme si dát pozor na dvě věci:

- Nemáme k dispozice dvě pásky, to vyřešíme pomocí dobře zvolené páskové abecedy.
- Máme k dispozici pouze n políček pásky (kde n je velikost vstupu). To ovšem nevadí: G je nezkracující, žádná věta vedoucí k odvození řetězce délky n nemůže být delší než n. Pokud v některé větvi výpočtu zjistíme, že bychom potřebovali více místa, pak LBA zamítne.

Do páskové abecedy mimo Σ a $\sqcup$ budou patřit i symboly z množiny $\Sigma \times (N \cup \sqcup)$. ($N$ je množina neterminálů v gramatice G.) Představíme si, že ve dvojici $[a, K]$ je a na první pásce a K je na druhé pásce. (*Intuitivně vidíme, že práci na dvou páskách lze tedy simulovat pomocí práce s „dvoupáskovými" symboly páskové abecedy.*)

Na začátku LBA pro vstup $w = w_1 w_2 \dots w_n$ upraví pásku tak, aby obsahovala

$$\triangleright[w_1, S][w_2, \sqcup] \dots [w_n, \sqcup]\triangleleft$$

LBA ted během procesu odvozování:

1. Nedeterministicky vybere pozici ve větné formě zapsané na druhé pásce.
2. Nedeterministicky vybere pravidlo z gramatiky.
3. Zkontroluje, zda jde vybrané pravidlo aplikovat na vybraném místě druhé pásky. Pokud ano, pak toto pravidlo aplikuje. (*Tady může být nutné část větné formy na druhé pásce posunout doprava. Pokud bychom zjistili, že by se tak konec větné formy dostal za konec pásky, zamítáme.*)
4. Ověří, jestli se větná forma na druhé pásce rovná řetězci na první pásce, pokud ano, tak přijímá. Pokud ne, tak přejde ke kroku 1.

□

Věta 15. *Pro každé LBA* A *existuje kontextově závislá gramatika* B *tak, že* $L(A) - \{\epsilon\} = L(G)$.

Proof. Důkaz je analogický důkazu Věty 13 (o existenci gramatiky typu 0 pro rozpoznatelné jazyky). Musíme vyřešit několik technických problémů, např. musíme zapracovat symboly pro začátek a konec pásky a symbol pro aktuální stav do složených symbolů, protože cílová gramatika musí být nezkracující.

Technické detaily jsou ponechány pro samostudium z literatury. □

Rozhodnutelné vs kontextově závislé jazyky

Věta 16. *Pro každou kontextově závislou gramatiku* G *platí, že* $L(G)$ *je rozhodnutelný.*

Proof. Rozhodující TS sestaví pro vstup x délky n následující graf:

- Vrcholy jsou větné formy gramatiky G o délce maximálně n.
- Z větné formy α vede hrana do větné formy β právě když $\alpha \vdash \beta$.
- TS ověří v sestaveném grafu existenci cesty z vrcholu S do vrcholu x. (Problém existence cesty v grafu je rozhodnutelný, viz např. algoritmus pomocí výpočtu tranzitivního uzávěru).

□

Věta 17. *Existuje rozhodnutelný jazyk* L *tak, že pro každou kontextově závislou gramatiku* G *platí* $L \neq L(G)$.

Proof. Každá kontextově závislá gramatika G je konečný objekt, existuje její zakódování $\langle G \rangle$ tak, že pro každý řetězec x můžeme pomocí TS rozhodnout, jesli kóduje gramatiku. Existuje tak enumerátor bezkontextové gramatiky enumerující.

Protože bezkontextových gramatik je spočetně mnoho, můžeme uvažovat jejich posloupnost $G_1, G_2 \ldots$. Uvážíme všechny řetězce $x_1, x_2, \ldots$. Definujeme jazyk $L = \{x_i \notin L(G_i)\}$. Pro všechna $i = 1, 2, \ldots$ tak máme $L \neq L(G_i)$ a L tak není generován kontextově závislou gramatikou.

Existuje TS, který rozhoduje L. Pro vstup x

1. najde index i tak, že $x = x_i$ (pro Σ^* totiž existuje enumerátor),
2. vygeneruje $\langle G_i \rangle$ (opět pomocí enumerátoru),
3. ověří, jestli G_i generuje x_i a podle toho přijímá/zamítá. (*Ověření jestli* $x_i \in L(G_i)$*: TS na základě* $\langle G \rangle$ *a* $|x_i|$ *sestaví graf větných forem jako v důkazu předchozí věty a poté ověří dosažitelnost* x_i *z počátečního neterminálu.*)

□

References

[1] Michael Sipser. *Introduction to the theory of computation* (international edition).

[2] John E. Hopcroft, Jeffrey D. Ullman. *Introduction to automata theory, languages and computation* (first edition, 1979).