6.6 Konkreettisia ratkeamattomuustuloksia

Valitettavan monet tärkeät tietojenkäsittelyongelmat ovat ratkeamattomia. Ns. Ricen lauseen mukaan itse asiassa jokseenkin kaikki ohjelmien toimintaa, tai tarkemmin sanoen niiden laskemia syöte/tulos-kuvauksia koskevat kysymykset ovat ratkeamattomia.

6.6.1 Turingin koneiden pysähtymisongelma

Lause: Kieli

$$H = \{c_M w \mid M \text{ pysähtyy syötteellä } w\}$$

on rekursiivisesti lueteltava, mutta ei rekursiivinen.

Todistus. Todetaan ensin, että kieli H on rekursiivisesti lueteltava:

- Universaalikoneesta M_U on helppo muokata kone, joka syötteellä $c_M w$ simuloi koneen M laskentaa syötteellä w ja pysähtyy hyväksyvään lopputilaan, jos ja vain jos simuloitu laskenta ylipäätään pysähtyy.

Osoitetaan sitten, että kieli H ei ole rekursiivinen:

- Oletetaan, että olisi $H = L(M_H)$ jollakin totaalisella Turingin koneella M_H.
- Oletetaan lisäksi, että kone M_H pysähtyessään jättää nauhalle alkuperäisen syötteensä, mahdollisesti tyhjämerkeillä jatkettuna.
- Olkoon M_U edellä konstruoitu universaalikone.
- Kielelle U voitaisiin nyt muodostaa totaalinen tunnistaja yhdistämällä koneet M_H ja M_U kaavion esittämällä tavalla.
- Aiemman tuloksen mukaan tällaista totaalia kielen U tunnistajakonetta ei kuitenkaan voi olla olemassa! Saatu ristiriita osoittaa, että H ei voi olla rekursiivinen. □

⇒ Seuraus: Kieli

$$\tilde{H} = \{c_M w \mid M \text{ ei pysähdy syötteellä } x\}$$

ei ole rekursiivisesti lueteltava. □

6.6.2 Sama tulos Pascalilla

Turingin koneiden ja "tavallisten" ohjelmien vastaavuus:

Turingin kone -formalismi	$\sim$	ohjelmointikieli
Turingin kone	$\sim$	ohjelma
Turingin koneen koodi	$\sim$	konekieliesitys
Universaalikone	$\sim$	konekielen tulkki

- Koska mikä tahansa Turingin kone voidaan toteuttaa Pascal-ohjelmana ja kääntäen, siirtyvät Turingin koneita koskevat ratkeavuus- ja ratkeamattomuustulokset välittömästi koskemaan myös Pascal-ohjelmia.
- Esimerkiksi pysähtymisongelman Pascal-tulkinta on: "Ei ole olemassa totaalista Pascal-ohjelmaa, joka ratkaisisi, pysähtyykö annettu Pascal-ohjelma P annetulla syötteellä w".

Pysähtymisongelman ratkeamattomuustodistus suoraan Pascalilla:

Oletetaan, että voitaisiin kirjoittaa totaalinen Pascal-funktio

function $H(p, w : \mathit{text}) : \mathit{Boolean}$,

joka saa arvon **true**, jos merkkijonoparametrin p esittämä proseduuri pysähtyy syötteellä w, ja **false** muuten.

Kirjoitetaan tämän perusteella Pascal-proseduuri $\hat{H}$:

```
procedure Ĥ(p : text);
    function H(p, w : text) : Boolean;
    begin {Funktion H runko}
        ...
    end;
    begin {Pääohjelma}
        if H(p, p) then while true do;
    end.
```

Merkitään proseduurin $\hat{H}$ ohjelmatekstiä $\hat{h}$:lla ja tarkastellaan proseduurin $\hat{H}$ laskentaa omalla kuvauksellaan.

Saadaan ristiriita:

$$\begin{aligned} \hat{H}(\hat{h}) \text{ pysähtyy} &\Leftrightarrow H(\hat{h}, \hat{h}) = \textbf{false} \\ &\Leftrightarrow \hat{H}(\hat{h}) \text{ ei pysähdy.} \end{aligned}$$

Ristiriidasta seuraa, että oletettua totaalista pysähtymisentestausohjelmaa H ei voi olla olemassa.

6.6.3 Semanttisten ominaisuuksien ratkeamattomuus

- Turingin koneen M *semanttinen ominaisuus* $\sim$ mikä tahansa sellainen ominaisuus $\mathcal{S}$, joka riippuu vain koneen M tunnistamasta kielestä, ei sen syntaktisesta rakenteesta. (joskus puhutaan myös *funktionaalisista ominaisuuksista*, mikä viittaa koneen toimintaan)
- tarkkaan ottaen semanttinen ominaisuus $\mathcal{S}$ on mikä tahansa kokoelma rekursiivisesti lueteltavia aakkoston $\{0,1\}$ kieliä ja koneella M on ominaisuus $\mathcal{S}$, jos $L(M) \in \mathcal{S}$.
- Esimerkkejä semanttisista ominaisuuksista:
  - "M hyväksyy tyhjän syötejonon"
  - "M hyväksyy jonkin syötejonon"
  - "M hyväksyy äärettömän monta merkkijonoa"
  - "M:n tunnistama kieli on säännöllinen" jne.
- Jos kahdella Turingin koneella M_1 ja M_2 on $L(M_1) = L(M_2)$, niin koneilla M_1 ja M_2 on täsmälleen samat semanttiset ominaisuudet.
- Ominaisuus $\mathcal{S}$ on *ratkeava*, jos annetusta Turingin koneen koodista voidaan algoritmisesti päätellä, onko koneella kysytty semanttinen ominaisuus.
- ts. jos joukko

  $$\text{codes}(\mathcal{S}) = \{c \mid L(M_c) \in \mathcal{S}\}$$

  on rekursiivinen.
- *Triviaalit ominaisuudet* ovat ominaisuus $\mathcal{S}_\emptyset$, jota ei ole millään koneella ja ominaisuus $\mathcal{S}_{RE}$, joka on kaikilla koneilla.

Ricen lause: Kaikki Turingin koneiden epätriviaalit semanttiset ominaisuudet ovat ratkeamattomia.

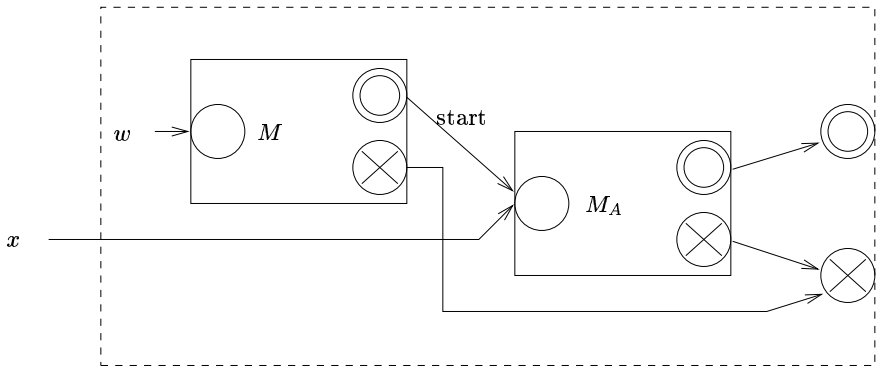
**Todistus.*

- Olkoon $\mathcal{S}$ mielivaltainen epätriviaali semanttinen ominaisuus.

- Voidaan olettaa, että $\emptyset \notin \mathcal{S}$: toisin sanoen, että tyhjän joukon tunnistavilla Turingin koneilla ei ole tarkasteltavaa ominaisuutta. (Tämä ei merkitse oleellista rajoitusta, sillä jos $\emptyset \in \mathcal{S}$, voidaan seuraavaa todistusta käyttäen osoittaa, että ominaisuus $\bar{\mathcal{S}} = \mathrm{RE} - \mathcal{S}$ on ratkeamaton, ja päätellä edelleen tästä, että myös ominaisuus $\mathcal{S}$ on ratkeamaton.)

- Koska $\mathcal{S}$ on epätriviaali, on olemassa jokin Turingin kone M_A, jolla on ominaisuus $\mathcal{S}$ — jolla siis $L(M_A) \neq \emptyset \in \mathcal{S}$.

- Olkoon M_{ENCODE} Turingin kone, joka muodostaa syötteenä annetusta, muotoa $c_M w$ olevasta merkkijonosta, seuraavanlaisen Turingin koneen M^w koodin (jos syöte ei ole vaadittua muotoa, M_{ENCODE} päätyy hylkäävään lopputilaan):



Kuva 6.6: Turingin koneen M^w rakenne (Ricen lause).

- Syötteellä x kone M^w toimii ensin kuten M syötteellä w. Jos M hyväksyy w:n, M^w toimii kuten kone M_A syötteellä x. Jos M hylkää w:n, myös M^w hylkää x:n (kuva 6.6).

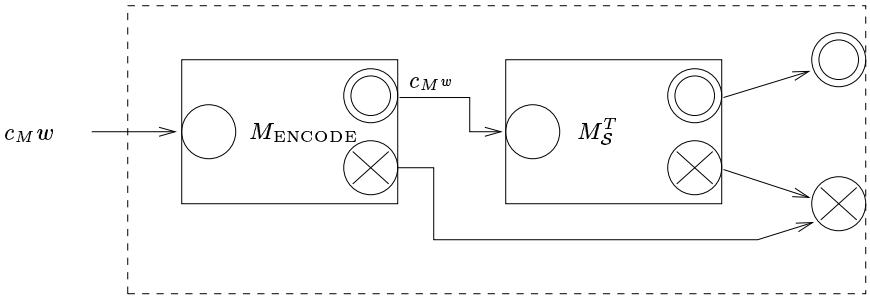
- Koneen M^w tunnistama kieli on siten:

$$L(M^w) = \begin{cases} L(M_A), & \text{jos } w \in L(M); \\ \emptyset, & \text{jos } w \notin L(M). \end{cases}$$

- Koska oletuksen mukaan $L(M_A) \in \mathcal{S}$ ja $\emptyset \notin \mathcal{S}$, on koneella M^w ominaisuus $\mathcal{S}$, jos ja vain jos $w \in L(M)$.

- Vastaväite: ominaisuus $\mathcal{S}$ on ratkeava, ts. kielellä codes($\mathcal{S}$) on totaalinen tunnistajakone $M_{\mathcal{S}}^T$.

- Tällöin saataisiin totaalinen tunnistajakone kielelle U yhdistämällä koneet M_{ENCODE} ja $M_{\mathcal{S}}^T$ kuvan mukaisesti. Selvästi kone M_U^T on totaalinen, jos $M_{\mathcal{S}}^T$



Kuva 6.7: Turingin koneen M_U^T rakenne (Ricen lause).

on, ja

$$\begin{aligned} & c_M w \in L(M_U^T) \\ \Leftrightarrow \quad & c_{M^w} \in L(M_{\mathcal{S}}^T) = \text{codes}(\mathcal{S}) \\ \Leftrightarrow \quad & L(M^w) \in \mathcal{S} \\ \Leftrightarrow \quad & w \in L(M). \end{aligned}$$

- Koska kieli U ei ole rekursiivinen, seuraa ristiriita, eli ominaisuus $\mathcal{S}$ ei voi olla ratkeava. □

6.6.4 Muita ratkeamattomia ongelmia

- **Predikaattikalkyylin ratkeamattomuus; Church/Turing 1936**

  Ei ole olemassa algoritmia, joka ratkaisisi, onko annettu ensimmäisen kertaluvun predikaattikalkyylin kaava ϕ validi ("loogisesti tosi", todistuva predikaattikalkyylin aksioomista). □

- **"Hilbertin 10. ongelma"; Matijasevitsh/Davis/Robinson/Putnam 1953–70**

  Ei ole olemassa algoritmia, joka ratkaisisi, onko annetulla kokonaislukukertoimisella polynomilla $P(x_1, \ldots, x_n)$ kokonaislukunollakohtia (so. jonoja $(m_1, \ldots, m_n) \in \mathbf{Z}^n$, joilla $P(m_1, \ldots, m_n) = 0$). Ongelma on ratkematon jo, kun muuttujien lkm $n = 15$ tai polynomin aste $\deg(P) = 4$. □

- Eräiden kielioppiongelmien ratkeavuus, kun annettuna on kieliopit G ja G' Chomskyn hierarkian tietyltä tasolta i ja merkkijono w. Taulukossa $R \sim$ "rat-

keava", $E \sim$ "ei ratkeava", $T \sim$ "aina totta".

	Taso i:			
Ongelma: onko	3	2	1	0
$w \in L(G)$?	R	R	R	E
$L(G) = \emptyset$?	R	R	E	E
$L(G) = \Sigma^*$?	R	E	E	E
$L(G) = L(G')$?	R	E	E	E
$L(G) \subseteq L(G')$?	R	E	E	E
$L(G) \cap L(G') = \emptyset$?	R	E	E	E
$L(G)$ säännollinen?	T	E	E	E
$\underline{L(G)} \cap L(G')$ tyyppiä i?	T	E	T	T
$\overline{L(G)}$ tyyppiä i?	T	E	T	E

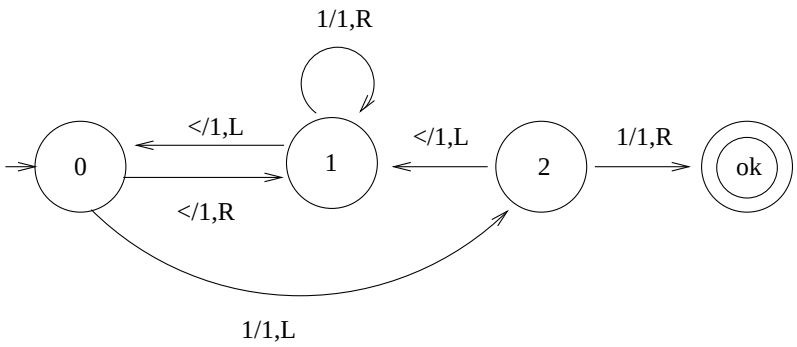
- Huom! Esim. jäsennysongelma $w \in L(G)$, kun kielioppi G on rajoittamaton, on siis ratkeamaton ongelma. Samoin mielivaltaisen kieliopin kuvaaman kielen säännöllisyys ja eräät sulkeumaominaisuudet.

6.6.5 *Joitain hauskoja ratkeamattomia ongelmia

Busy Beaver

Busy Beaver-ongelmassa tehtävänä on keksiä n-tilainen standardimallinen Turingin kone (jonka on siis joka askeleella siirryttävä joko oikealle tai vasemmalle – paikallaan ei saa pysyä), joka aloittaa toimintansa tyhjällä nauhalla ja on pysähtyessään kirjoittanut nauhalle mahdollisimman monta merkkiä. Aakkostona voidaan käyttää mitä tahansa unaariaakkostoa, esim. $\Sigma = \{a\}$, jolloin tehtävänä on kirjoittaa nauhalle mahdollisimman monta a-kirjainta. Vaihtoehtoisesti tehtävänä voi olla laatia mahdollisimman hidas Turingin kone, joka siis suorittaa mahdollisimman monta askelta ennen pysähtymistään. Kummassakin ongelmassa koneen täytyy kuitenkin pysähtyä lopulta. Tilojen lukumäärään n ei lasketa mukaan lopputiloja.

Esimerkiksi 3-tilainen Busy Beaver -kone voi kirjoittaa 6 merkkiä ja toimia 21 siirtymän ajan. Kuvassa kone, joka pysähtyy 13 siirtymän jälkeen ja kirjoittaa maksimaalisen 6 merkkiä:



Busy Beaver -funktio $f(n) = \{x | n$-tilainen kone kirjoittaa maksimissaan x merkkiä aloittaessaan tyhjältä nauhalta$\}$ kasvaa tavattoman nopeasti. Esim. 6-tilaisen koneen ennätys on 95,524,079 merkkiä ja kone pyörii parhaimmillaan 8,690,333,381,690,952 askeleen ajan. Funktiota ei kuitenkaan pysty laskemaan, joten tulokset on saatu kokeilemalla ja uusia ennätyksiä on odotettavissa.

Post correspondence problem (PCP)

PCP-ongelman voi mallintaa dominopelinä, jossa on käytettävissä äärellinen kokoelma erilaisia dominonappuloita. Kunkin nappulan ylä- ja alareunassa on merkkijono, esim. $[\frac{b}{ca}], [\frac{a}{ab}], [\frac{ca}{a}], [\frac{abc}{c}]$. Kustakin nappulasta voi "taikoa" niin monta kopiota kuin ikinä haluaa. Tehtävänä on asettaa nappulat sillä tavalla peräkkäin, että ala- ja yläreunaan muodostuu sama merkkijono. Esim. edellisen kokoelman nappuloilla eräs ratkaisu olisi $[\frac{a}{ab}][\frac{b}{ca}][\frac{ca}{a}][\frac{a}{ab}][\frac{abc}{c}]$.

Yleisesti ongelma on siis seuraava: Annettuna kokoelma $P = [\frac{t_1}{s_1}], [\frac{t_2}{s_2}], ..., [\frac{t_k}{s_k}]$, onko olemassa sellaista indeksijonoa $i_1, i_2, ..., i_n$, että $s_{i_1} s_{i_2} ... s_{i_n} = t_{i_1} t_{i_2} ... t_{i_n}$?

6.7 *Ongelmien palautukset

Joskus ongelma voidaan palauttaa (redusoida) joksikin toiseksi tunnetuksi ongelmaksi, jolloin voimme päätellä myös alkuperäisen ongelman ratkeavuuden/ratkeamattomuuden. Ehtona kuitenkin on, että ongelman palautus on itsessään ratkeava ongelma kaikilla syötteillä (ts. se voidaan suorittaa totaalisella Turingin konella).

Idea:

1. Meillä on tuntematon kieli A ja tunnettu kieli B
2. Palautetaan A B:ksi palautusfunktiolla f siten, että $x \in A \Leftrightarrow f(x) \in B$ kaikilla $x \in \Sigma^*$.

3. Nyt A on ratkeava, jos B on, A on osittain ratkeava, jos B on, ja A on ratkeamaton, jos B on, *kunhan* palautusfunktio f on itsessään ratkeava.

Määritelmä: Turingin koneen $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{yes}}, q_{\text{no}})$ *laskema osittaiskuvaus* (t. *-funktio*)

$$f_M : \Sigma^* \to \Gamma^*$$

määritellään:

$$f_M(x) = \begin{cases} u, & \text{jos } (q_0, \underline{x}) \vdash_M^* (q, u\underline{a}v) \text{ jollakin } q \in \{q_{\text{yes}}, q_{\text{no}}\}, av \in \Gamma^*; \\ \text{määrittelemätön, muuten.} \end{cases}$$

- Osittaisfunktio $f : \Sigma^* \to A$ on *osittaisrekursiivinen* jos se voidaan laskea jollakin Turingin koneella ja *(kokonais-)rekursiivinen*, jos se voidaan laskea jollakin totaalisella Turingin koneella.
- Ekvivalentisti voitaisiin määritellä, että osittaisrekursiivifunktio f on rekursiivinen, jos sen arvo $f(x)$ on määritelty kaikilla x.
- Tarvitsemme siis *rekursiivisen palautus funktion*

Määritelmä: Formaali kieli $A \subseteq \Sigma^*$ voidaan *palauttaa rekursiivisesti* kieleen $B \subseteq \Gamma^*$, merkitään

$$A \leq_m B,$$

jos on olemassa rekursiivinen funktio $f : \Sigma^* \to \Gamma^*$, jolla on ominaisuus:

$$x \in A \quad \Leftrightarrow \quad f(x) \in B, \qquad \text{kaikilla } x \in \Sigma^*.$$

- Huom! Palautusketju voi olla pitkäkin, sillä rekursiivinen palautusrelaatio on transitiivinen, ts. jos $A \leq_m B$ ja $B \leq_m C$, niin $A \leq_m C$.
- Voisimme siis palauttaa A:n B:ksi, B:n C:ksi, C:n D:ksi, jne. kunnes vastaan tulee tuttu kieli.
- Jos $A \leq_m B$ ja B on rekursiivisesti numeroituva, on A:kin rekursiivisesti numeroituva.
- Jos $A \leq_m B$ ja B on rekursiivinen, on A:kin rekursiivinen.

- Kaksi kieltä A ja B ovat keskenään isomorfiset, jos ne on määritelty samassa aakkostossa Σ ja rekursiivinen palautusfunktio on bijektio ts. voimme suorittaa palautuksen kumpaankin suuntaan.

- Formaalisti:

  Määritelmä: Kielet $A, B \subseteq \{0,1\}^*$ ovat *rekursiivisesti isomorfisia*, jos on olemassa rekursiivinen bijektio $f : \{0,1\}^* \to \{0,1\}^*$ (tällöin myös käänteisfunktio f^{-1} on välttämättä rekursiivinen), jolla

  $$x \in A \quad \Leftrightarrow \quad f(x) \in B, \qquad \text{kaikilla } x \in \Sigma^*.$$

- Rekursiivisesti numeroituvaa kieltä A sanotaan RE-täydelliseksi, jos mikä tahansa luokan RE kieli voidaan rekursiivisesti palauttaa A:han.

- Formaalisti:

  Kieli $A \subseteq \{0,1\}^*$ on *RE-täydellinen*, jos

  (i) $A \in$ RE ja
  (ii) $B \leq_m A$ kaikilla $B \in$ RE.

- esimerkiksi universaalikieli U on RE-täydellinen.

  Todistus. Tiedetään, että $U \in$ RE. Olkoon $B = L(M_B)$ mielivaltainen luokan RE kieli. Tällöin B voidaan palauttaa U:hun funktiolla $f(x) = c_{M_B}x$. Tämä funktio on selvästi rekursiivinen, ja sillä on ominaisuus

  $$x \in B = L(M_B) \quad \Leftrightarrow \quad f(x) = c_{M_B}x \in U. \quad \square$$

- Ricen lauseesta seuraa, että mm. kaikki ongelmat, joissa yritetään tehdä jotain päätelmiä Turingin koneiden tunnistamista kielistä niiden koodien perusteella ovat RE-täydellisiä.

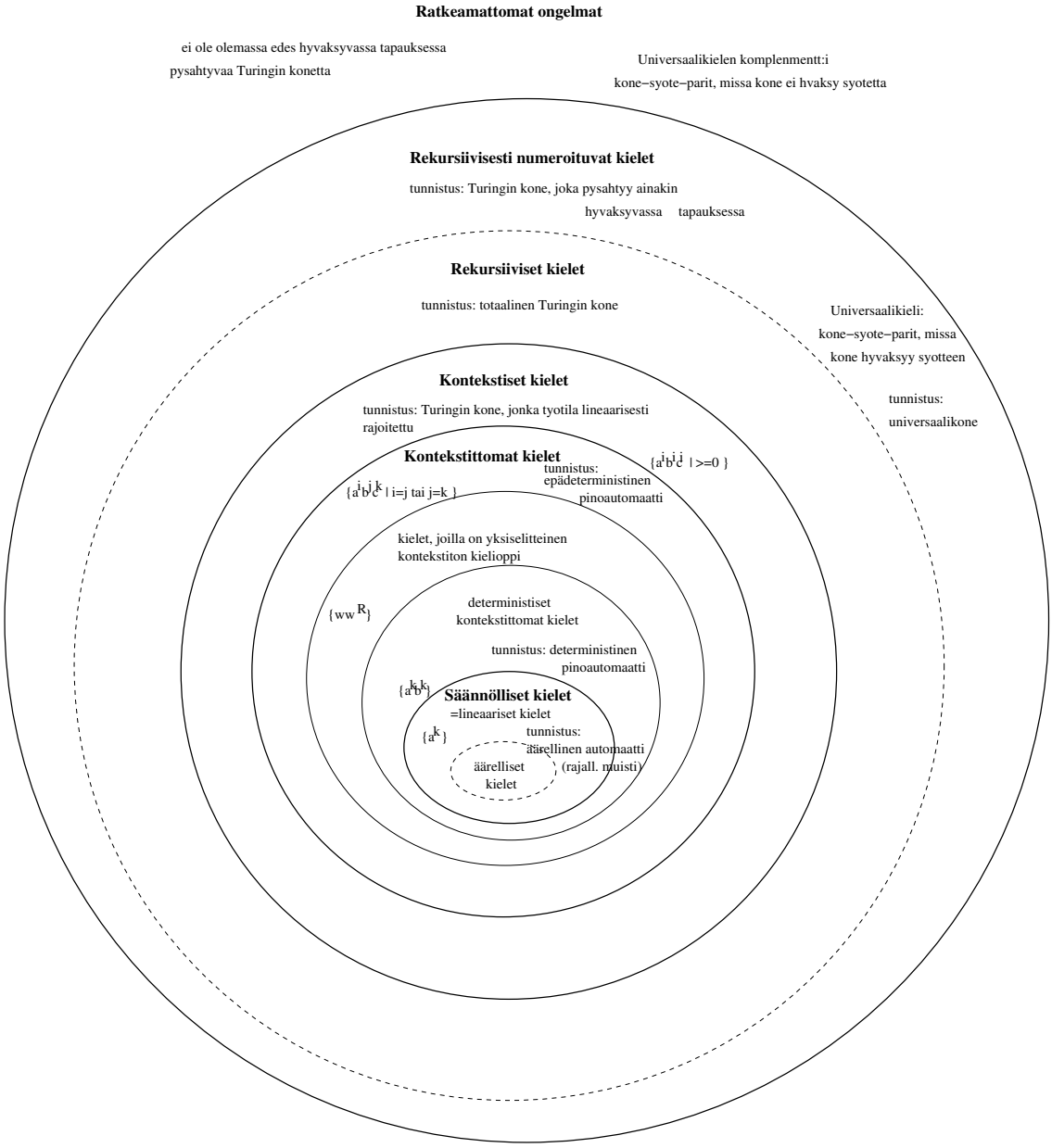
- Yleensäkin näyttää olevan niin, että kaikki "luonnolliset" rekursiivisesti lueteltavat, ei-rekursiiviset kielet ovat RE-täydellisiä.

- Teoreettisesti voidaan kuitenkin osoittaa, että [Post] Luokassa RE $\setminus$ REC on kieliä, jotka eivät ole RE-täydellisiä. $\square$

- Huom! Kaikki RE-täydelliset kielet ovat rekursiivisesti isomorfisia. (ts. ne voidaan palauttaa toisikseen)

6.8 Yhteenveto

viimeisellä luennolla...



6.9 *Laskennan vaativuudesta

Tietojenkäsittelijälle ei riitä tieto, onko ongelma ratkeava, vaan hän haluaa myös tietää, kuinka vaativa se on. Chomskyn kielihierarkian luokat antavat yleiskuvan siitä, kuinka vaikea ongelma on. Jos joku ongelma ratkeaa äärellisellä automaatilla ja toinen vaatii Turingin koneen, on ensimmäinen jossain mielessä helpompi ongelma. Mutta entä jos ensimmäinen ongelma vaatii äärellisen automaatin, jonka tilojen lukumäärä on eksponentiaalisesti suurempi kuin toisen ongelman ratkaisevan Turingin koneen? Enää ensimmäisen ongelman yksinkertaisuus ei olekaan niin itsestäänselvää.

Miten sitten voisimme verrata ongelmien vaativuutta objektiivisesti? Entä jos vertaisimme ratkaisevien ohjelmien koodeja tai pseudokoodialgoritmeja? Kumpikaan näistä ei kuitenkaan täysin paljasta ongelman todellista vaativuutta, vaikka käytännön ohjelmointitilanteissa niiden arviointi riittää hyvin. Ohjelma voi "piilottaa" yhteen askeleeseen hyvinkin vaihtelevan määrän operaatioita, eikä esim. assembler- ja Pascal-koodin askelien vertailu anna järkevää tulosta.

Onneksi kaikki ratkeavat ongelmat voidaan kuitenkin esittää Turingin koneina! Olipa kyseessä äärellisellä tai pinoautomaatilla ratkeava ongelma, voimme aina konstruoida myös Turingin koneen, joka ratkaisee ongelman, ja vertailla Turingin koneiden aikavaativuuksia. Tästä syystä Turingin koneet ovat hyvin tärkeitä, kun haluamme tietää ongelman todellisen aikavaativuuden, riipumatta siitä, millä kielellä tai laitteistolla se tullaan todellisuudessa toteuttamaan (tai jos haluamme tietää, kannattaako sitä ryhtyä lainkaan toteuttamaan).

Turingin koneen aika- ja tilavaativuus saadaan sen käyttämien askelten ja työnauhan solujen lukumääristä. Sanotaan, että kone M hyväksyy kielen L ajassa $T(n)$, jos kaikilla $x \in L$, missä $|x| = n$, M hyväksyy x:n $O(T(n))$:llä askeleella. Samoin kone M hyväksyy kielen L tilassa $S(n)$, jos kaikilla $x \in L$, $|x| = n$, M hyväksyy x:n käyttäen $O(S(n))$:ää työnauhan solua.

Huom! Epädeterministinen Turingin kone voi ratkaista ongelman pienemmin aika- ja tilavaatimuksin kuin deterministinen Turingin kone. Käytännössä epädeterministiset koneet täytyy ensin determinisoida, mutta teoreettisessa pohdiskelussa epädeterministisen Turingin koneiden vaativuusanalyysi voi olla hyvinkin valaisevaa. Esimerkiksi vaativuusluokan NP määritelmä perustuu tähän: luokan NP ongelmat kyetään ratkaisemaan polynomisessa ajassa epädeterministisellä Turingin koneella. Selvästi luokka P (ongelmat, jotka voidaan ratkaista polynomisessa ajassa deterministisellä Turingin koneella) sisältyy luokkaan NP ($P \subseteq NP$), mutta emme tiedä, päteekö sama toisinpäin, ts. onko $NP \subseteq NP$? Tämä $P = NP$-ongelma on ehkä tärkein tietojenkäsittelijöitä kiusaavista avoimista ongelmista.

Käytännössä merkittävimpiä luokan NP ongelmista ovat ns. *NP-täydelliset* ongelmat (*NP-complete problems*), jotka ovat eräänlaisia arkkityyppiongelmia luokassa NP. Mikä tahansa luokan NP ongelmista voidaan näet palauttaa polynomisessa ajassa NP-täydelliseksi ongelmaksi. Toisin sanoen, jos yksikin NP-täydellinen ongelma pystyttäisiin ratkaisemaan polynomisessa ajassa, niin kaikki muutkin luokan NP ongelmat ratkeaisivat polynomisessa ajassa. Tästä seuraa, että uusi ongelma voidaan osoittaa NP-täydelliseksi varsin helposti: riittää osoittaa, että se kuuluu luokkaan NP ja että jokin NP-täydelliseksi tunnettu ongelma voidaan palauttaa siihen. Tällöin myös kaikki muut luokan NP ongelmat seuraavat perässä. [5]

Nimitystä *NP-kova* ongelma (*NP-hard problem*) käytetään sellaisesta ongelmasta, joka myös "kattaa" kaikki luokan NP ongelmat, ts. mikä tahansa luokan NP ongelma voidaan muuntaa polynomisessa ajassa kyseiseksi ongelmaksi. Emme kuitenkaan tiedä, kuuluuko NP-kova itse luokkaan NP (se on siis niin "kova" ongelma, että kukaan ei ole keksinyt sille epädeterminististä, polynomisessa ajassa toimivaa Turingin konetta). Mikäli joku ongelma havaitaan NP-kovaksi, voi hyvin suurella todennäköisyydellä arvata sen olevan eksponentiaalista tai vieläkin pahempaa aikavaativuusluokkaa.

[5] Ainut ongelma on todistaa muulla tapaa, että se ensimmäinen ongelma on NP-täydellinen. Onneksi meillä on Cookin teoreema, joka todistaa ns. SAT-ongelman

$$SAT = \{F | F \text{ on toteutuva lausekalkyylin kaava}\}$$

NP-täydelliseksi, ja jatkossa voimme nojata tähän tulokseen.