

Edellä esitetyt kielten A_{TM} ja $HALT_{TM}$ ratkeamattomuustodistukset ovat esimerkkejä **palautuksesta** (reduction).

Intuitiivisesti ongelman A palauttaminen ongelmaan B tarkoittaa, että

- Oletetaan, että meillä on proseduurin P_B , joka ratkaisee ongelman B .
- Muodostetaan proseduurin P_A , joka ratkaisee ongelman A **käyttäen aliruutiinina** proseduuria P_B .

Kun ongelma A on palautettu ongelmaan B , voidaan tehdä päätelmiä **kahteen** suuntaan:

1. Jos proseduurin P_B **todella on olemassa**, olemme saaneet toimivan proseduurin myös ongelmalle A .
2. Jos ongelma A tiedetään ratkeamattomaksi, niin proseduuria P_B **ei voi olla olemassa**, joten myös B on ratkeamaton.

Käytännön tietojenkäsittelyssä palautusten soveltaminen suuntaan 1 on keskeistä (aliohjelmakirjastot).

Ratkeamattomuustulosten todistukset ovat usein **epäsuoria** ja perustuvat suunnan 2 käyttöön.

Jos ongelma A voidaan palauttaa ongelmaan B , merkitsemme

$$A \leq B.$$

Käytämme merkintää tässä epämuodollisesti ilman tarkkaa matemaattista määritelmää. Tarkka määritelmä voidaan tehdä eri tavoilla, jolloin puhutaan esim. kuvauspalautuksesta $A \leq_m B$ [Sipser luku 5.3] tai Turing-palautuksesta $A \leq_T B$ [Sipser luku 6.3]. Intuitiivinen tulkinta on joka tapauksessa, että

ongelma A voidaan ratkaista ongelman B avulla,

joten jossain mielessä

B on ainakin yhtä vaikea kuin A .

Kun halutaan tietää, onko jokin ongelma X ratkeava, voidaan siis yrittää kahdensuuntaisia palautuksia:

- Jos $A \leq X$, missä A on ratkeamaton, niin X on ratkeamaton.
- Jos $X \leq B$, missä B on ratkeava, niin X on ratkeava.

Edellä lauseen 4.16 todistuksessa $X = A_{\text{TM}}$ ja $A = D$. Vastaavasti korollarissa 4.18 $X = \text{HALT}_{\text{TM}}$ ja $A = A_{\text{TM}}$. Sama idea toistuu jatkossa.

Edellä todettiin äärellisten automaattien ja yhteydettömien kielten tyhjyysoongelmat E_{DFA} ja E_{CFG} ratkeaviksi. Kuitenkin

Lause 4.19: [Sipser Thm. 5.2] Turingin koneiden tyhjyysoongelma

$$E_{\text{TM}} = \{ \langle M \rangle \mid M \text{ on Turingin kone ja } L(M) = \emptyset \}$$

on ratkeamaton.

Todistus: Tehdään palautus hyväksymisongelmasta A_{TM} . Osoitamme, että mistä tahansa Turingin koneesta M ja merkkijonosta w voidaan muodostaa Turingin kone M_1 , jolla on seuraava ominaisuus:

$$L(M_1) = \begin{cases} \{w\} & \text{jos } w \in L(M) \\ \emptyset & \text{muuten.} \end{cases}$$

Jos E_{TM} olisi ratkeava, kieli A_{TM} voitaisiin ratkaista algoritmilla, joka syötteellä $\langle M, w \rangle$ toimii seuraavasti:

1. Muodosta edellä mainitun lainen kone M_1 .
2. Jos $\langle M_1 \rangle \in E_{\text{TM}}$, niin **hylkää**. Muuten **hyväksy**.

Koska todellisuudessa A_{TM} **ei** ole ratkeava, myös E_{TM} on ratkeamaton.

Kone M_1 toimii syötteellä x seuraavasti:

1. Jos $x \neq w$, niin **hylkää**.
2. Muuten simuloi konetta M syötteellä x . Jos M hyväksyisi, niin **hyväksy**. Jos M hylkäisi, niin **hylkää**.

Nyt koneella M_1 on haluttu ominaisuus

$$L(M_1) = \begin{cases} \{w\} & \text{jos } w \in L(M) \\ \emptyset & \text{muuten.} \end{cases}$$

Huomaa, että konetta M_1 rakennettaessa ei tarvitse tietää, kumpi vaihtoehto pätee.

Jos nyt R olisi ratkaisija ongelmalle E_{TM} , niin ongelma A_{TM} voitaisiin ratkaista koneella, joka syötteellä $\langle M, w \rangle$ toimii seuraavasti:

1. Muodosta syötteestä $\langle M, w \rangle$ edellä kuvattu kone M_1 .
2. Simuloi konetta R syötteellä $\langle M_1 \rangle$. Jos R hyväksyisi, niin **hylkää**. Jos R hylkäisi, niin **hyväksy**.

Kone M_1 saadaan koneesta M lisäämällä suunnilleen $|x|$ tilaa, jotka tarkastavat syötteen x . Siis annetusta $\langle M, w \rangle$ osataan helposti muodostaa $\langle M_1 \rangle$. \square .

Kielen E_{TM} ratkeamattomuus on erikoistapaus [Ricen lauseesta](#).

Kieli A on [semanttinen ominaisuus](#), jos se voidaan esittää muodossa

$$A = \{ \langle M \rangle \mid L(M) \in \mathcal{S} \}$$

jollain [joukolla kieliä](#) $\mathcal{S} \subseteq \mathcal{P}(\Sigma^*)$. Esim. E_{TM} on semanttinen ominaisuus; siinä $\mathcal{S} = \{ \emptyset \}$ (joukko, jonka ainoa alkio on tyhjä kieli).

Siis jos A on semanttinen ominaisuus, kysymyksen ”päteekö $\langle M \rangle \in A$ ” vastaus riippuu vain koneen M hyväksymästä kielestä. Kääntäen kieli A **ei ole** semanttinen ominaisuus, jos joillain M_1 ja M_2 pätee $L(M_1) = L(M_2)$, mutta $\langle M_1 \rangle \in A$ ja $\langle M_2 \rangle \notin A$.

Semanttinen ominaisuus A on [triviaali](#), jos joko kaikilla M pätee $\langle M \rangle \in A$ tai kaikilla M pätee $\langle M \rangle \notin A$.

Lause 4.20 (Rice): [[Sipser Problem 5.28](#)] Jokainen ei-triviaali semanttinen ongelma on ratkeamaton.

Todistus sivuutetaan, mutta perustuu oleellisesti samaan ideaan kuin lauseen 4.19 todistus. \square

Lause 4.21: Turingin koneiden epätyhjyysongelma

$$\tilde{E}_{\text{TM}} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

on Turing-tunnistettava.

Todistus: Kieli \tilde{E}_{TM} voidaan tunnistaa epädeterministisellä Turingin koneella, joka syötteellä $\langle M \rangle$ toimii seuraavasti:

1. Valitse epädeterministisesti merkkijono w .
2. Simuloi konetta M syötteellä w . Jos M hyväksyisi, niin hyväksy. Jos M hylkäisi, niin hylkää.

□

Korollaari 4.22: Turingin koneiden tyhjyysongelma E_{TM} ei ole Turing-tunnistettava.

Todistus: Tyhjyysongelman komplementti voidaan esittää muodossa

$$\overline{E_{\text{TM}}} = \tilde{E}_{\text{TM}} \cup B,$$

missä B koostuu merkkijonoista, jotka eivät ole muotoa $\langle M \rangle$ millekään M . Emme ole tarkemmin määritelleet koodausta $\langle \cdot \rangle$, mutta minimivaatimus järkevälle koodaukselle on, että B on ratkeava. Siis $\overline{E_{\text{TM}}}$ on Turing-tunnistettava. Koska E_{TM} ei ole ratkeava, se ei ole edes Turing-tunnistettava (vrt. korollaari 4.17). □

Lause 4.23: [Sipser Thm. 5.4] Turingin koneiden ekvivalenssiongelma

$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ ja } M_2 \text{ ovat Turingin koneita ja } L(M_1) = L(M_2) \}$
on ratkeamaton

Todistus: Tehdään vastaoletus, että EQ_{TM} on ratkeava. Olkoon M_\emptyset jokin Turingin kone, jolla $L(M_\emptyset) = \emptyset$. Nyt E_{TM} voidaan ratkaista Turingin koneella, joka syötteellä $\langle M \rangle$ toimii seuraavasti:

1. Muodosta $w = \langle M, M_\emptyset \rangle$.
2. Jos $w \in EQ_{TM}$, niin hyväksy; muuten hylkää.

Mutta E_{TM} on ratkeamaton; ristiriita. \square

Itse asiassa pätee voimakkaammin

Lause 4.24: [Sipser Thm. 5.30] Kumpikaan kielistä EQ_{TM} ja $\overline{EQ_{TM}}$ ei ole Turing-tunnistettava.

Todistus: melko suoraviivainen, mutta sivuutetaan. \square

Todetaan lopuksi ilman todistusta, että esim. seuraavat yhteydettömiin kielioppeihin liittyvät ongelmat ovat ratkeamattomia:

- onko G moniselitteinen,
- päteekö $L(G_1) \cap L(G_2) = \emptyset$,
- päteekö $L(G) = \Sigma^*$ ja
- päteekö $L(G_1) = L(G_2)$.

Täsmällisemmin esim. viimeinen kohta tarkoittaa, että kieli

$$EQ_{CFG} = \{ \langle G_1, G_2 \rangle \mid G_1 \text{ ja } G_2 \text{ ovat CFG:itä ja } L(G_1) = L(G_2) \}$$

on ratkeamaton.

5. Laskettavuus, kieliopit, logiikka

Lopuksi tarkastelemme Turingin koneeseen (tai yleisemmin algoritmeihin) perustuvaa laskettavuuden käsitettä suhteessa kahteen muuhun tärkeään tiedonkäsittelyformalismiin:

Kieliopit: Turing-tunnistettavat kielet liitetään säännöllisten ja yhteydettömien kanssa [Chomskyn hierarkiaan](#).

Logiikka: Ratkeamattomuus on läheisessä yhteydessä formaalista logiikasta tuttuun [epätäydellisyyteen](#).

Tavoitteena on vetää yhteen kurssin teemoja ja asettaa niitä laajempaan yhteyteen. Teknisiin yksityiskohtiin ei juuri kiinnitetä huomiota (eikä niitä kysytä kokeessa).

Chomskyn hierarkia

Rajoittamaton kielioppi on nelikko $G = (V, \Sigma, R, S)$, missä

1. V on äärellinen muuttujien joukko,
2. Σ on päätesymbolien joukko, jolla $\Sigma \cap V = \emptyset$,
3. R on äärellinen joukko sääntöjä muotoa $u \rightarrow v$, missä $u \in (\Sigma \cup V)^+$ ja $v \in (\Sigma \cup V)^*$ ja
4. $S \in V$ on lähtösymboli.

Tämä laajentaa yhteydettömiä kielioppeja siten, että säännön $u \rightarrow v$ vasen puoli u saa olla mikä tahansa epätyhjä merkkijono, ei pelkästään yksi muuttujasymboli.

Kuten yhteydettömien kielioppien tapauksessa, merkitsemme $w \Rightarrow w'$, ja sanomme että w johtaa suoraan merkkijonon w' , jos voidaan kirjoittaa $w = xuy$ ja $w' = xvy$, missä $(u \rightarrow v) \in R$. Jos $w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_{n-1} \Rightarrow w_n$, sanomme että w_0 johtaa merkkijonon w_n , ja merkitsemme $w_0 \xRightarrow{*} w_n$.

Kieliopin tuottama kieli on $L(G) = \{ w \in \Sigma^* \mid S \xRightarrow{*} w \}$.

Esimerkki 5.1: Kieli

$$A = \{ a^i b^i c^i \mid i \in \mathbf{N} \}$$

tunnetusti ei ole yhteydetön. Se voidaan kuitenkin tuottaa rajoittamattomalla kieliopilla

$$\begin{aligned} S &\rightarrow aAbc \mid abc \mid \varepsilon \\ A &\rightarrow aAbC \mid abC \\ Cb &\rightarrow bC \\ Cc &\rightarrow cc, \end{aligned}$$

missä on käytetty samoja merkintäkonventioita kuin yhteydettömille kieliopille.

Esim. merkkijonolle aaabbbccc saadaan johto

$$\begin{aligned} S &\Rightarrow aAbc \Rightarrow aaAbCbc \Rightarrow aaabCbCbc \Rightarrow aaabCbbCc \\ &\Rightarrow aaabbCbCc \Rightarrow aaabbbCCc \Rightarrow aaabbbCcc \Rightarrow aaabbbccc. \end{aligned}$$

□

Kielioppien ja Turingin koneiden yhteys on seuraava:

Lause 5.2: Kieli on Turing-tunnistettava, jos ja vain jos jokin rajoittamaton kielioppi tuottaa sen.

Todistushahmotelma: Kun on annettu rajoittamaton kielioppi G , kieli $L(G)$ voidaan luetella seuraavasti:

1. Käy läpi kaikki yhden pituiset johdot $S \Rightarrow w_1$ ja tulosta kaikki päättemerkkijonot $w \in \Sigma^*$.
2. Käy läpi kaikki kahden pituiset johdot $S \Rightarrow w_1 \Rightarrow w_2$ ja tulosta kaikki päättemerkkijonot $w_2 \in \Sigma^*$.
3. Käy läpi kaikki kolmen pituiset johdot $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow w_3$ ja tulosta kaikki päättemerkkijonot $w_3 \in \Sigma^*$.
4.

Siis $L(G)$ on Turing-tunnistettava (lause 3.9).

Mielenkiintoisempi suunta on muodostaa annetulle Turingin koneelle $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ kielioppi $G = (V, \Sigma, R, S)$, jolla $L(G) = L(M)$.

Perusidea on tulkita Turingin koneen tilanteet

$$u_1u_1 \dots u_nqv_1v_2 \dots v_n, \quad u_i, v_i \in \Gamma, q \in Q$$

merkkijonoiksi. Tätä varten valitaan kieliopin muuttujiksi $V = Q \cup (\Gamma - \Sigma)$, jolloin tilanteet ovat suoraan aakkoston $V \cup \Sigma$ merkkijonoja.

Turingin koneen laskennan esittämiseksi liitetään kielioppiin sääntöjä seuraavasti:

- Jos $\delta(r, a) = (s, b, R)$, lisätään sääntö $ra \rightarrow bs$.
- Jos $\delta(r, a) = (s, b, L)$, lisätään sääntö $cra \rightarrow scb$ kaikilla $c \in \Gamma$.

Nyt Turingin koneen laskenta-askelta $uqv \vdash u'q'v'$ vastaa kieliopin suora johto $uqv \Rightarrow u'q'v'$.

Laskennan alun ja lopun vaatimat yksityiskohdat sivuutetaan (ks. esim. Sudkamp: Languages and Machines, tai Laskennan teoria luennot). \square

Yhteydettömien kielioppien ohella toinen tärkeä erikoistapaus on **yhteysherkkät** (context-sensitive) kieliopit. Tällaisessa kieliopissa kaikilla säännöillä $u \rightarrow v$ pitää olla $|u| \leq |v|$. Poikkeuksena sallitaan kuitenkin sääntö $S \rightarrow \varepsilon$ edellyttäen, että S ei esiinny minkään säännön oikealla puolella. Kieli A on yhteysherkkä, jos $A = L(G)$ jollain yhteysherkkällä G .

Esimerkki 5.3: Yhteydettömän kieliopin säännöissä $u \rightarrow v$ pätee aina $|u| = 1$. Poistamalla ε -säännöt (kuten Chomskyn normaalimuodon yhteydessä) yhteydetön kielioppi saadaan yhteysherkkään muotoon. Siis yhteydettömät kielet ovat yhteysherkkiä.

Toisaalta esimerkki 5.1 itse asiassa osoittaa kielen $\{ a^i b^i c^i \mid i \in \mathbf{N} \}$ yhteysherkkäksi. Siis kaikki yhteysherkkät kielet eivät ole yhteydettömiä. \square

Nimitys "yhteysherkkä" tulee siitä, että tällaisen kieliopin säännöt voidaan muuntaa muotoon $xAy \rightarrow xwy$, missä $A \in V$, $x, y \in (V \cup \Sigma)^*$ ja $w \in (V \cup \Sigma)^+$. Siis sääntöä $A \rightarrow w$ saadaan soveltaa vain "kontekstissa" $x _ y$.

Lause 5.4: Yhteysherkät kielet ovat ratkeavia.

Todistushahmotelma: Erikoistapausta $S \Rightarrow \varepsilon$ lukuunottamatta missä tahansa yhteysherkän kieliopin johdossa

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_n$$

pätee

$$1 \leq |w_1| \leq |w_2| \leq \dots \leq |w_n|.$$

Siis erityisesti jos kysytään jostakin merkkijonosta w , päteekö $S \xRightarrow{*} w$, niin riittää tarkastella johtoja, joiden välivaiheiden w_i pituudet $|w_i|$ ovat korkeintaan $|w|$. Koska tällaisia välivaiheita on äärellinen määrä, voidaan esim. käydä läpi kaikki niiden järjestykset ja katsoa, muodostuuko laillinen johto. \square

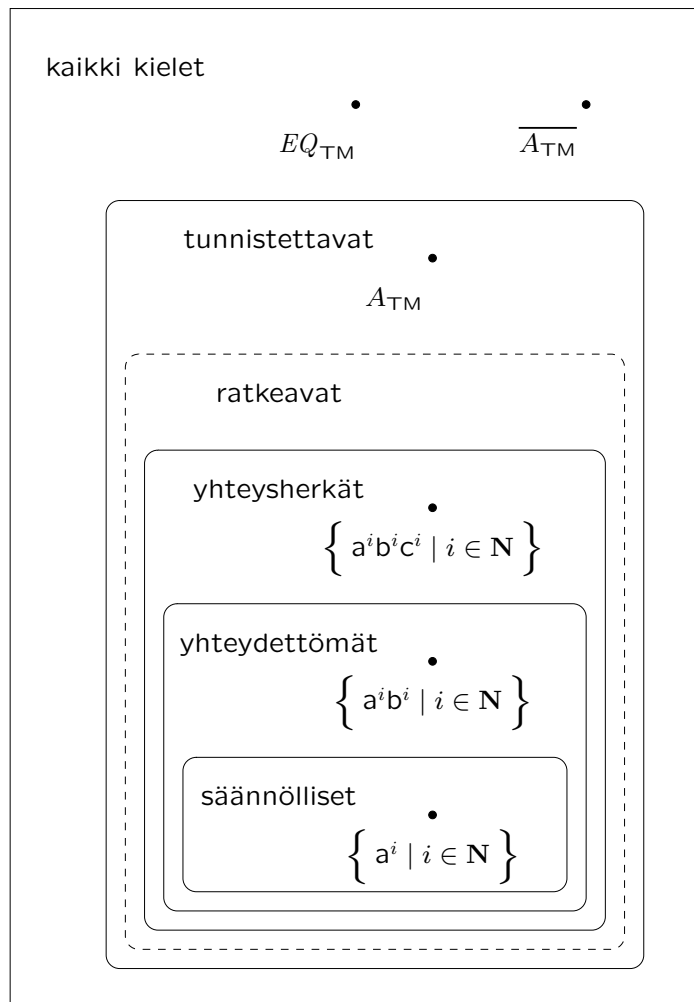
Tarkemmin voidaan osoittaa, että kieli on yhteysherkkä, jos ja vain jos se voidaan tunnistaa **lineaarisesti rajoitetulla automaatilla** (linear-bounded automaton, LBA). Tällainen automaatti on epädeterministinen Turingin kone, joka ei käytä nauhatilaa enempää kuin syötteen pituuden verran, ts. ei koskaan kirjoita mitään tyhjämärkin päälle.

Yleisiä, yhteysherkkiä, yhteydettömiä ja oikealle lineaarisia kielioppeja kutsutaan vastaavasti tyyppin 0, 1, 2 ja 3 kielioppeiksi. Saadaan seuraava Chomskyn hierarkia:

tyyppi	kieli	kielioppi	automaatti
0	tunnistettava	rajoittamaton	Turingin kone
1	yhteysherkkä	yhteysherkkä	lin. rajoitettu
2	yhteydetön	yhteydetön	pinoautom.
3	säännöllinen	oikealle lineaarinen	äärellinen autom.

(Oikealle lineaarista kielioppeista ks. harjoitus 6.) Ylempi taso sisältää myös kaikkien alempien tasojen kielet. Lisäksi tiedämme, että

- kaikki kielet eivät ole edes tunnistettavia,
- tasot ovat erillisiä (esim. on olemassa yhteysherkkiä ei-yhteydettömiä kieliä) ja
- yhteysherkkät \subset ratkeavat \subset tunnistettavat.



Chomskyn hierarkia ja eräiden kielten sijainti sen suhteen

Laskettavuus ja logiikka

Matematiikassa ja logiikassa **todistus** on keskeinen käsite. Yleisesti todistus on argumentti, jolla lukija vakuutetaan jonkin väitteen pätevyydestä.

Jotta todistukset todella olisivat vakuuttavia, niiden pitää perustua yhteisesti hyväksytyihin täsmällisiin sääntöihin.

Ääritapaus täsmällisyydestä on **formaali todistus**. Tällöin todistamisessa sallitut säännöt on kirjattu niin yksityiskohtaisesti, että annetun todistuksen oikeellisuus voidaan tarkistaa vaikka tietokoneella.

Täysin formaalit todistukset ovat yleensä hyvin hankalia. Käytännön matematiikassa tyydytään miltei aina vähäisempään täsmällisyyden asteeseen. Formaaliin todistamiseen liittyvät kysymykset ovat kuitenkin keskeisiä matematiikan perusteiden tarkastelussa.

Tarkastelemme seuraavassa logiikan formalisointia laskettavuuden näkökulmasta. **Varoitus:** yksityiskohdat sivuutetaan.

Tyypillinen tapa formalisoida päättelyä on määritellä **aksiomia** ja **päättelysääntöjä**.

Aksiomat ovat väittämiä, joita voidaan todistuksessa pitää annettuina. Tyypillinen esimerkki aksiomasta on

$$(\forall x)(\forall y)(\forall z)((x = y) \wedge (y = z)) \rightarrow (x = z),$$

joka esittää yhtäsuuruusrelaation transitiivisuuden.

Päättelysäännöt esitetään tyypillisesti muodossa

$$\frac{\psi_1, \dots, \psi_n}{\phi}.$$

Tämä sääntö tarkoittaa, että jos väittämät ψ_1, \dots, ψ_n on jo saatu todistetuksi, voidaan edelleen päätellä ϕ . Perusesimerkki päättelysäännöstä on **modus ponens**

$$\frac{\psi, \quad \psi \rightarrow \phi}{\phi}.$$

Väittämän ϕ todistus annetuista aksioomista ja päättelysäännöistä on jono ψ_1, \dots, ψ_n , missä

- $\psi_n = \phi$ ja
- kaikilla $1 \leq i \leq n$ pätee
 - ψ_i on aksiooma tai
 - on olemassa päättelysääntö

$$\frac{\theta_1, \dots, \theta_k}{\psi_i},$$

jolla $\{\theta_1, \dots, \theta_k\} \subseteq \{\psi_1, \dots, \psi_{i-1}\}$.

Oletetaan jatkossa, että aksioomien ja päättelysääntöjen joukot (sopivasti koodattuna) ovat ratkeavia. Tämä on edellä esitetyn motivaation valossa luonteva (mutta ei pakollinen) rajoitus.

Huomaa, että aksioomia ja päättelysääntöjä voi silti olla ääretön määrä.

Jos aksiomien ja päättelysääntöjen joukot ovat ratkeavia, myös kaikkien **todistusten** joukko

$$P = \{ \langle \psi_1, \dots, \psi_n \rangle \mid \psi_1, \dots, \psi_n \text{ on todistus kaavalle } \psi_n \}$$

on selvästi **ratkeava**.

Tästä seuraa edelleen, että **todistuvien väitteiden** joukko

$$T = \{ \langle \phi \rangle \mid \text{on olemassa } \psi_1, \dots, \psi_{n-1}, \text{ joilla } \langle \psi_1, \dots, \psi_{n-1}, \phi \rangle \in P \}$$

on **Turing-tunnistettava**. Kieli T voidaan luetella esim. seuraavasti:

1. Alusta $n := 1$.
2. Etsi leksikografisessa järjestyksessä ensimmäinen ϕ , jolla $\langle \psi_1, \dots, \psi_{n-1}, \phi \rangle \in P$.
3. Tulosta $\langle \phi \rangle$. Aseta $\psi_n := \phi$ ja $n := n + 1$. Palaa kohtaan 2.

Sen sijaan todistuvien väittämien joukko T ei välttämättä ole ratkeava. Lyhyenkin väittämän ϕ todistuksessa voi tarvita välivaiheina pitkiä väittämiä ψ_i . Tällöin ylläoleva luettelija ei tuota kieltä T leksikografisessa järjestyksessä.

Onko T todella ratkeava vai ei riippuu siitä, mitä nimenomaisia aksioomia ja päättelysääntöjä tarkastellaan.

Esim. luonnollisten lukujen Peanon aksioomien tapauksessa T ei ole ratkeava.

Toisaalta jos järjestelmä on **täydellinen** (complete) ja **ristiriidaton** (consistent), eli kaikilla ϕ pätee joko $\langle \phi \rangle \in T$ tai $\langle \neg\phi \rangle \in T$ mutta ei molemmat, niin T on ratkeava. Kysymys "pätee $\langle \phi \rangle \in T$ " voidaan näet ratkaista seuraavasti:

1. Simuloi edelläesitettyä kielen T luettelijaa, kunnes se tulostaisi $\langle \phi \rangle$ tai $\langle \neg\phi \rangle$.
2. Jos tulostettavana on $\langle \phi \rangle$, niin **hyväksy**.
Jos tulostettavana on $\langle \neg\phi \rangle$, niin **hylkää**.

(Asiat esitetään tarkemmin kurssilla Matemaattinen logiikka.)

Todetaan vielä seuraava kokonaislukujen aritmetiikkaan liittyvä keskeinen ratkeamattomuustulos:

Lause 5.5: Ei ole olemassa algoritmia, joka ratkaisee, onko annetulla kokonaislukukertoimisella polynomilla nollakohtia kokonaislukujen joukossa. Ts. kieli

$\{ \langle p \rangle \mid p(x_1, \dots, x_n) \text{ on kokonaislukukertoiminen polynomi} \\ \text{ja on olemassa } a_1, \dots, a_n \in \mathbf{Z} \text{ joilla } p(a_1, \dots, a_n) = 0 \}$

ei ole ratkeava.

Tästä seuraa yleisemmin, että ei ole olemassa menetelmää testata, onko jokin kokonaislukuja koskeva väittämä tosi. Huomaa, että tämä on eri asia kuin testata *toteutuvuutta* jossain formaalissa järjestelmässä (esim. Peanon aksioomat).