

Számításelmélet

Tamás Herendi

Számításelmélet

Tamás Herendi

Publication date 2014.

Table of Contents

1. Előszó	1
2. Formális nyelvek	2
3. Függvények növekedési rendje	8
4. A Turing-gép	14
1. A Turing-gép definíciója	15
2. Felismerő Turing-gépek	17
3. Turing-gépek megadása	18
3.1. Táblázatos reprezentáció:	18
3.2. Gráfprezentáció:	24
3.3. Turing-gép megadása felsorolással:	24
3.4. Példák	25
3.5. Feladatok	25
4. Church-Turing tézis	26
5. Turing-gépek összefűzése	27
6. Többszalagos Turing-gépek, szimuláció	33
5. Kiszmáthatoság-elmélet	39
1. Univerzális Turing-gép és az univerzális nyelv	39
2. A diagonális nyelv	41
3. Nyelvek rekurzivitása	41
4. Megállási probléma	56
6. Nemdeterminisztikus Turing-gépek	59
1. Nemdeterminisztikus Turing-gépek definíciója	59
2. Nemdeterminisztikus Turing-gépek szimulációja	63
7. Bonyolultságfogalmak	66
1. Idő-, tár- és programbonyolultság	66
2. Bonyolultsági osztályok	69
3. Tár-idő tételek	72
8. Az NP nyelvosztály	74
1. A Tanú-tétel	74
2. <i>NP</i> -teljesség	76
Irodalomjegyzék	80

List of Figures

3.1. 1. ábra	8
3.2. 2. ábra	9
3.3. 3. ábra	9
4.1. Turing gép modell	15
4.2. Paritásellenőrző bit hozzáfűzése a bemenő szóhoz	18
4.3. Kezdő konfiguráció: K_0	19
4.4. K_1	19
4.5. K_2	19
4.6. K_3	19
4.7. K_4	19
4.8. K_5	19
4.9. K_6	19
4.10. K_7	20
4.11. K_8	20
4.12. K_9	20
4.13. K_{10}	20
4.14. K_{11}	20
4.15. K_{12}	20
4.16. K_{13}	20
4.17. A Turing-gép számítása a 011001 bemeneten	20
4.18. Paritásellenőrző bit tesztelése	21
4.19. Kezdő konfiguráció: K_0	21
4.20. K_1	22
4.21. K_2	22
4.22. K_3	22
4.23. K_4	22
4.24. K_5	22
4.25. K_6	22
4.26. K_7	22
4.27. K_8	22
4.28. K_9	22
4.29. K_{10}	23
4.30. K_{11}	23
4.31. K_{12}	23
4.32. K_{13}	23
4.33. K_{14}	23
4.34. K_{15}	23
4.35. A Turing-gép számítása a 0110010 bemeneten	23
4.36. T_1 és T_2 kompozíciója	28
4.37. T_1 , T_2 és T_3 feltételes kompozíciója	30
4.38. T_1 iterációja	32
4.39. Többszalagos Turing-gép	33

4.40. Egy kétszalagos Turing-gép szimulációja 37

List of Examples

3.1. 1. példa	8
3.2. 2. példa	8
3.3. 3. példa	9

Chapter 1. Előszó

Jelen jegyzet a Debreceni Egyetemen tartott Számításelmélet kurzushoz, annak anyagára épülve készült. A kurzusnak - és ennek megfelelően a jegyzetnek is - alapvető célja, hogy a hallgatók megfelelő háttérképzést jussanak a kiszámíthatóság- és bonyolultságelméleti alapfogalmakból.

A 2. fejezet az általános algoritmikus feladat és a Turing-gép megfogalmazásához szükséges formális nyelvekkel kapcsolatos fogalmakat ismerteti.

A 3. fejezetben a bonyolultságelméletben alkalmazandó, függvények növekedési rendjével kapcsolatos definíciók és tulajdonságok találhatók.

A 4. fejezet a Turing-gép definícióját és a hozzá kapcsolódó alapvető ismereteket tárgyalja. Itt kerül ismertetésre a Turing-gépek számításának fogalma, a Church-Turing tézis, a Turing-gépek összefűzésének elmélete, a több szalagos Turing-gépek definíciója és a hozzá kapcsolódó szimulációs tételek.

Az 5. fejezetben a kiszámíthatóságelmélettel találkozhatunk, ahol a rekurzivitást, rekurzív felsorolhatóságot és ezek kapcsolatát vizsgáljuk. A fejezet végén az eldönthetőségi problémákról esik szó.

A 6. fejezet a nemdeterminisztikus Turing-gépekkel kapcsolatos definíciókkal és alapvető tulajdonságokkal, valamint a modell Turing-ekvivalenciával foglalkozik.

A 7. fejezetben az idő-, tár- és programbonyolultságról, a velük definiálható bonyolultsági osztályokról és kapcsolatukról esik szó.

A 8. fejezet az NP nyelvosztályhoz köthető eredményeket tárgyalja, többek között a tanú tételel és az NP -teljességgel kapcsolatos legfontosabb ismereteket.

Chapter 2. Formális nyelvek

Ahogy általában a programokat, úgy az algoritmusokat is jellemzően nem egy konkrét feladat, hanem egy feladatcsoport megoldására alkotjuk. Az algoritmus bemenetét, kimenetét illetve más kapcsolódó objektumokat (pl. magát az algoritmust is) valamilyen általános módon szeretnénk megadni, kezelní. Ehhez szükségünk lesz néhány alapvető fogalomra és eredményre a formális nyelvek elméletéből.

Először következzen néhány függvényekkel kapcsolatos fogalom.

2.1. Definíció

Legyen A és B két tetszőleges nem üres halmaz.

A két halmaz **Descartes-szorzatán** az $A \times B = \{(a, b) | a \in A \text{ és } b \in B\}$ párokból álló halmazt értjük. Ha $B = A$, akkor az $A \times A = A^2$ jelölést is használhatjuk.

Legyen $R \subseteq A \times B$. Az R halmazt az $A \times B$ -n értelmezett **relációnak** nevezik.

Ha valamelyen $a \in A$ és $b \in B$ esetén $(a, b) \in R$, akkor azt mondjuk, hogy a és b R **szerinti relációban van**.

Legyen $R \subseteq A \times B$ egy reláció. Ha $\forall a \in A$ és $\forall b_1, b_2 \in B$ esetén $(a, b_1), (a, b_2) \in R$ csak akkor lehet, ha $b_1 = b_2$, akkor R -et **(parciális) függvénynek** nevezzük és $R: A \rightarrow B$ -vel jelöljük, illetve ha $(a, b) \in R$, akkor a hagyományos $R(a) = b$ jelölést használjuk.

Ha $\forall a \in A \exists b \in B$ amelyikre $R(a) = b$, akkor **totális függvénynek** nevezzük.

Legyen $R: A \rightarrow B$ egy totális függvény. Ha $\forall b \in B$ esetén $\exists a \in A$ amelyikre $R(a) = b$, akkor R -et **szürjektív leképezésnek** vagy ráképezésnek nevezzük.

Amennyiben $\forall a_1, a_2 \in A$ esetén $R(a_1) = R(a_2)$ -ból következik, hogy $a_1 = a_2$, akkor R -et **injectív leképezésnek** vagy egy-egy értelmű leképezésnek nevezzük.

Ha R szürjektív és injektív is, akkor **bijektívnek** nevezzük.

2.2. Megjegyzés

Az $R \subseteq A \times B$ reláció esetén a hagyományos jelölés aRb , ahogy például $a \leq$ és \subseteq relációknál megszoktuk.

Ha $f: A_1 \rightarrow B_1$ és $g: A_2 \rightarrow B_2$ két függvény, akkor értelmezhetjük az uniójukat, $f \cup g$ mint relációk unióját. Ez azonban nem lesz feltétlenül újabb függvény. Amennyiben feltesszük, hogy $A_1 \cap A_2 = \emptyset$, akkor már $f \cup g$ is függvény, amire $(f \cup g): (A_1 \cup A_2) \rightarrow (B_1 \cup B_2)$.

Mivel a későbbiek során azt szeretnénk igazolni, hogy a felépített algoritmusfogalmunk matematikai szempontból precízen megadható, ezért szükség van az ábécé, szó és nyelv pontos, a szemléletestől eltérő, de minden részletre kiterjedő definíciójára. Erre azért is van különösen nagy szükség, hogy az üres jelnek nevezett szimbólum helyét megtaláljuk a rendszerben.

2.3. Definíció

Általánosítás: a V^* véges nem üres halmazt **ábécének** nevezzük.

elemeit **betűknek** (időnként jeleknek) nevezzük $V^* \setminus \{*\} \neq \emptyset$

Amennyiben $* \in V$, akkor $*$ -re megköveteljük, hogy legyen.
(A továbbiakban $*$ -t **üres jelnek** nevezzük.)

2.4. Definíció

Legyen W egy nem üres halmaz és $\sigma_a : W \rightarrow W$ egy leképezés $\forall a \in V$ esetén.

Ha W -re teljesülnek a következők:

1. $\exists \lambda \in W$;
2. $\forall a \in V \setminus \{*\}$ betűre igaz, hogy $\nexists w \in W$ amire $\sigma_a(w) = \lambda$;
3. $\forall a, b \in V \setminus \{*\}$ és $\forall w, v \in W$ esetén $\sigma_a(w) = \sigma_b(v)$, akkor és csak akkor, ha $w = v$ és $a = b$; (lehet, hogy elegendő az "akkor")
4. $\forall w \in W \setminus \{\lambda\}$ esetén, $\exists a \in V \setminus \{*\}$ és $v \in W$, amelyikre $\sigma_a(v) = w$; (Ez lehet, hogy 5-ból következik.)
5. (Teljes indukció.) Legyen $A(\cdot)$ egy állítás W elemein. Ha $A(\lambda)$ igaz és $\exists a \in V \setminus \{*\}$ esetén $A(w)$ -ból következik $A(\sigma_a(w))$, akkor $\forall w \in W$ esetén igaz $A(w)$;
6. Ha $* \in V$, akkor $\forall w \in W \sigma_*(w) = w$, akkor W -t a V ábécé fölötti **véges szavak halmazának** és λ -t az **üres szónak** nevezzük.

A hagyományoknak megfelelően használni fogjuk a $V^* = W$ és a $V^+ = V^* \setminus \{\lambda\}$ jelölést.

2.5. Megjegyzés

1. Legyen $a \in V \setminus \{*\}$. A $\Sigma_a(w)$ leképezést egyszerűen úgy értelmezhetjük, hogy $\Sigma_a(w)$ -t w -ból egy a betű hozzáírásával kapjuk.
2. A V és V^* halmazok között megadható egy $\varphi : V \rightarrow V^*$ természetes beágyazás, ahol $\varphi(a) = \Sigma_a(\lambda)$. Ennek megfelelően a V^+ halmaz elemei tekinthetők a $V \setminus \{*\}$ halmaz elemeiből álló véges hosszúságú sorozatoknak.
3. Az előző definíció 6. pontja alapján $\varphi(*) = \lambda$, azaz $*$ és λ különbözők, de a két halmazban egymásnak megfeleltethetők.
4. A definíció 6. pontja alapján a 3. pontját kiterjeszthetjük az $a = *$ és $b = *$ esetekre is.
5. Az 5. pont lényegében azt jelenti, hogy minden szó előállítható az üres szóból betűk hozzáadogatásával. Ilyen módon, amennyiben $w = \Sigma_{a_n}(\Sigma_{a_{n-1}}(\dots(\Sigma_{a_1}(\lambda))\dots))$, ahol $a_i \in V \setminus \{*\}$, akkor a szót jelölhetjük a szokásos módon $w = a_1 \dots a_{n-1} a_n$ alakkal is.
6. A 3. pont alapján az előbbi előállítás egyértelmű.
7. Az előző pontot kiterjeszthetjük az üres jelre is, $a_1 \dots a_k * a_{k+1} \dots a_n = a_1 \dots a_k a_{k+1} \dots a_n$ tulajdonsággal.

2.6. Definíció

Legyen $l : V^* \rightarrow \mathbb{N}$ függvény a következő tulajdonságokkal:
1. $l(\lambda) = 0$;

2. $l(\Sigma_a(w)) = l(w) + 1$, $\forall a \in V \setminus \{\star\}$ és $\forall w \in V^*$ esetén.

Ekkor az $l(w)$ értéket a **w szó hosszának** nevezzük.

2.7. Tétel

$\forall w \in V^*$ esetén $l(w)$ értéke egyértelműen definiált.

Bizonyítás

Egy w szó hossza a definíció alapján azt jelenti, hogy hányszor kellett σ_a leképezést alkalmazni az üres szóra, hogy megkapjuk w -t. Mivel a szó előállítása során a valódi bővítő lépések darabszáma és sorrendje rögzített, w hosszára egyértelmű értéket kapunk. ✓

2.8. Megjegyzés

Belátható, hogy amennyiben egy szót véges jelsorozatnak tekintünk, a hossza pontosan a benne szereplő jelek számával egyezik meg.

2.9. Definíció

Legyen \cdot egy kétváltozós művelet V^* -on, azaz: $\cdot : V^* \times V^* \rightarrow V^*$, a következő tulajdonságokkal:

1. $\forall w \in V^*$ legyen $w \cdot \lambda = w$;
2. $\forall a \in V \setminus \{\star\}$ és $\forall w, v \in W$ esetén $w \cdot \Sigma_a(v) = \Sigma_a(w \cdot v)$.

Ekkor \cdot -t az **összefűzés** (más néven **konkatenáció**) műveletének nevezzük.

A hagyományoknak megfelelően a $w \cdot v$ jelölés helyett gyakran az egyszerűbb wv írásmódot választjuk.

2.10. Tétel

A konkatenáció műveletére igaz, hogy $\forall w_1, w_2 \in V^*$ esetén, ha $w_1 = a_1 \dots a_n$ és $w_2 = b_1 \dots b_m$, akkor $w_1 \cdot w_2 = a_1 \dots a_n b_1 \dots b_m$.

Bizonyítás

A bizonyítást m -re vonatkozó teljes indukcióval végezzük.

a.) A definíció alapján, ha $w_2 = \lambda$, azaz $l(w_2) = 0$, akkor az állítás igaz.

b.) Tegyük fel, hogy egy adott m_0 esetén minden $w_1, w_2 \in V^*$ szóra, ha $l(w_2) = m_0$, akkor az állítás igaz.

Legyen $w'_2 \in V^*$ egy $m_0 + 1$ hosszú szó. Ha w'_2 utolsó betűje a , akkor $w'_2 = \sigma_a(w_2)$ valamilyen w_2 -re, ahol $l(w_2) = m_0$.

Az indukciós feltevés szerint $w_1 \cdot w_2 = a_1 \dots a_n b_1 \dots b_{m_0}$. Felhasználva az összefűzés definícióját,

$$w_1 \cdot w'_2 = w_1 \cdot \sigma_a(w_2) = \sigma_a(w_1 \cdot w_2) = a_1 \dots a_n b_1 \dots b_{m_0} b$$

ami azt jelenti, hogy az állítás w'_2 -re is igaz.

a.)-ból és b.)-ból teljes indukcióval következik az állítás. ✓

2.11. Tétel

V^*

a konkatenációval, mint kétváltozós művelettel egységelemes félcsoportot alkot.

Bizonyítás

A bizonyítást teljes indukcióval végezzük.

1. A konkatenáció asszociatív:

a.) Definíció szerint $(u \cdot v) \lambda = u \cdot v = u \cdot (v \cdot \lambda)$.

b.) Tegyük fel, hogy egy adott m_0 esetén minden $u, v, w \in V^*$ szóra, ha $l(w) = m_0$, akkor igaz az asszociativitás. Legyen $w' \in V^*$ olyan, hogy $l(w') = m_0 + 1$. Ekkor $\exists a \in V$ és $w \in V^*$ amelyikre $l(w) = m_0$ és $w' = \sigma_a(w)$. Az indukciós feltevés alapján minden $u, v \in V^*$ szóra igaz, hogy $(u \cdot v) \cdot w = u \cdot (v \cdot w)$. Ez alapján

$$\begin{aligned} (u \cdot v) \cdot w' &= (u \cdot v) \cdot \sigma_a(w) \\ &= \sigma_a((u \cdot v) \cdot w) \\ &= \sigma_a(u \cdot (v \cdot w)) \\ &= (u \cdot \sigma_a(v \cdot w)) \\ &= (u \cdot (v \cdot \sigma_a(w))) \\ &= (u \cdot (v \cdot w')) \end{aligned},$$

ami azt jelenti, hogy w' -re is igaz az állítás.

a.)-ból és b.)-ból teljes indukcióval következik az asszociativitás.

2. λ egységelem. Definíció szerint $v\lambda = v$ minden $v \in V^*$ esetén.

Azt, hogy $\lambda \cdot v = v$, teljes indukcióval bizonyíthatjuk.

a.) Definíció szerint $\lambda \cdot \lambda = \lambda$.

b.) Tegyük fel, hogy minden $v \in V^*$ esetén, ha $l(v) = m_0$, akkor $\lambda \cdot v = v$. Legyen $v' \in V^*$ egy $m_0 + 1$ hosszúságú szó. Ekkor $\exists a \in V$ és $v \in V^*$, amire $l(v) = m_0$ és $v' = \sigma_a(v)$. Az indukciós feltevés alapján

$$\lambda \cdot v' = \lambda \cdot \sigma_a(v) = \sigma_a(\lambda \cdot v) = \sigma_a(v) = v',$$

ami azt jelenti, hogy az állítás $m_0 + 1$ -re is igaz.

a.)-ból és b.)-ból teljes indukcióval következik $\lambda \cdot v = v$.

Ezzel a tételek beláttuk. ✓

2.12. Tétel

A konkatenáció műveletére érvényes az egyszerűsítési szabály, azaz $\forall w_1, w_2, v \in V^*$ esetén $w_1 \cdot v = w_2 \cdot v$ -ból következik $w_1 = w_2$.

Hasonlóan $v \cdot w_1 = v \cdot w_2$ -ból következik $w_1 = w_2$

Bizonyítás

A bizonyítást az előzőekhez hasonlóan teljes indukcióval végezhetjük. Az első esetben v -re, a második esetben pedig $-$ -re és $-$ -re egyidejűleg. ✓

Szavak szerkezetének leírására hasznosak a következő definíciók.

2.13. Definíció

Legyen $u, v, w \in V^*$ olyan, hogy $w = u \cdot v$.
Ekkor u -t a w egy **kezdőszeletének**, v -t pedig egy **zároszeletének** nevezünk.

2.14. Megjegyzés

Egy szó saját magának triviális kezdő- és zárószelete.
Az üres szó minden szónak triviális kezdő- és zárószelete.

A véges szavak halmazán megadhatunk $\varphi: V^* \rightarrow V^*$ alakú transzformációkat. Ezek rendelkezhetnek bizonos jó tulajdonságokkal. Néhány a legfontosabbak közül a következőképpen adható meg.

2.15. Definíció

Egy $\varphi: V^* \rightarrow V^*$ alakú transzformációt **hossztartónak** nevezünk,
ha $\forall v \in V^*$ esetén $l(v) = l(\varphi(v))$.

2.16. Megjegyzés

Hossztartó leképezés például a tükrözés, betűpermutáció, ciklikus permutáció, stb.

2.17. Definíció

Egy $\varphi: V^* \rightarrow V^*$ alakú transzformációt **kezdőszelettartónak** nevezünk,
ha $\forall u, v \in V^*$ esetén $\exists w \in V^*$ amelyikre $\varphi(u \cdot v) = \varphi(u) \cdot w$.

2.18. Megjegyzés

Az előző példák közül a tükrözés és ciklikus permutáció nem, viszont a betűpermutáció kezdőszelettartó.
Kezdőszelettartó, viszont nem hossztartó például a következő transzformáció: $\varphi(w) = w \cdot w$. (Ismétlés.)

A szavak fogalmát felhasználva az algoritmusok szempontjából fontos objektum meghatározásához érkeztünk.

2.19. Definíció

Legyen V egy véges ábécé. Az $L \subseteq V^*$ halmazt egy V fölötti (**formális**) **nyelvnek** nevezünk.

Nyelveken értelmezhetjük a hagyományos halmazműveleteket, pl. ha $L_1, L_2 \subseteq V^*$ két nyelv, akkor az $L_1 \cup L_2$ és $L_1 \cap L_2$ is az. Az $\bar{L} = V^* \setminus L$ nyelvet az L komplementerének nevezünk.

Ezekben kívül kiemelt szerepük a következő nyelvműveletek.

2.20. Definíció

Legyen $L_1, L_2 \subseteq V^*$ két nyelv. Az $L = \{u \cdot v \mid u \in L_1 \text{ és } v \in L_2\}$ nyelvet a **két nyelv összefűzésével nyert nyelvnek** nevezünk és $L = L_1 \cdot L_2$ -vel jelöljük.

2.21. Megjegyzés

Ha az L nyelvet saját magával fűzzük össze, használhatjuk a következő egyszerűsített jelölést: $\overline{L}^n = \overline{L} \cdot \overline{L}^{n-1}$ $n \geq 0$
és , ha .

2.22. Definíció

Legyen $L \subseteq V^*$ egy nyelv. Az $\overline{L}^* = \bigcup_{n=0}^{\infty} \overline{L}^n$ nyelvet az **Literáltjának**, vagy más néven a **konkatenációra vett lezártjának** nevezzük.

2.23. Megjegyzés

A lezárt elnevezés abból a tényből ered, hogy \overline{L}^* pontosan az a legsűkebb nyelv, amelyiknek részhalmaza L és nem vezet ki belőle az összefűzés művelete.

2.24. Definíció

Legyen \mathcal{H} nyelveknek egy osztálya.
Ekkor $co\mathcal{H} = \{L \mid \overline{L} \in \mathcal{H}\}$. Itt \overline{L} az L nyelv komplementerét jelöli.

2.25. Megjegyzés

$co\mathcal{H}$ nem a \mathcal{H} komplementerét jelöli. Olyannyira nem, hogy $\mathcal{H} \cap co\mathcal{H}$ nem feltétlenül üres. Ha pl. \mathcal{H} az összes nyelv osztálya, akkor $\mathcal{H} = co\mathcal{H}$.

2.26. Tétel

Legyen \mathcal{H} és \mathcal{G} két ugyanazon ábécé fölötti nyelvek osztálya.
Ekkor

1. $co\mathcal{H} \cup co\mathcal{G} = co(\mathcal{H} \cap \mathcal{G})$;
2. $co\mathcal{H} \cap co\mathcal{G} = co(\mathcal{H} \cup \mathcal{G})$;
3. $co(co\mathcal{H}) = \mathcal{H}$;
4. $\mathcal{H} \subseteq \mathcal{G}$ akkor és csak akkor, ha $co\mathcal{H} \subseteq co\mathcal{G}$.

Bizonyítás

Szokásos halmazelméleti egyenlőségbizonyításokkal dolgozhatunk.

1. Felírhatjuk a következő ekvivalencialáncot:

$$L \in co\mathcal{H} \cup co\mathcal{G} \equiv L \in co\mathcal{H} \text{ vagy } L \in co\mathcal{G} \equiv \overline{L} \in \mathcal{H} \text{ vagy } \overline{L} \in \mathcal{G} \equiv \overline{L} \in \mathcal{H} \cup \mathcal{G} \equiv L \in co(\mathcal{H} \cup \mathcal{G})$$

2. Az előzőhez teljesen hasonlóan felírhatjuk a következő ekvivalencialáncot:

$$L \in co\mathcal{H} \cap co\mathcal{G} \equiv L \in co\mathcal{H} \text{ és } L \in co\mathcal{G} \equiv \overline{L} \in \mathcal{H} \text{ és } \overline{L} \in \mathcal{G} \equiv \overline{L} \in \mathcal{H} \cap \mathcal{G} \equiv L \in co(\mathcal{H} \cap \mathcal{G})$$

3. Ebben az esetben a következő írhatjuk:

$$L \in co(co\mathcal{H}) \equiv \overline{L} \in co\mathcal{H} \equiv \overline{\overline{L}} \in \mathcal{H} \equiv L \in \mathcal{H}$$

4. Definíció szerint $L \in \mathcal{H}$ akkor és csak akkor, ha $\overline{L} \in co\mathcal{H}$ és $L \in \mathcal{G}$ akkor és csak akkor, ha $\overline{L} \in co\mathcal{G}$. Definíció szerint $\mathcal{H} \subseteq \mathcal{G}$ akkor és csak akkor, ha $\forall L \in \mathcal{H}$ -ból következik $L \in \mathcal{G}$. Az előzőek alapján ez pontosan akkor áll fenn, ha $\forall \overline{L} \in \mathcal{H}$ -ból következik $\overline{L} \in \mathcal{G}$. ✓

Chapter 3. Függvények növekedési rendje

A későbbi fejezetekben, bonyolultságelméleti vizsgálatoknál, szükségünk lesz arra, hogy függvények növekedési rendjét (aszimptotikus, azaz a végtelenhez tartó viselkedését) egységes formában ki tudjuk fejezni. Ehhez lényegében egy olyan formulát fogunk használni, amelyik segítségével egy függvényből egyszerűen le tudjuk választani a legfontosabb, legmeredekebben növő összetevőjét.

3.1. Definíció

Legyen $f : \mathbb{N} \rightarrow \mathbb{R}^+$ egy monoton növekvő függvény. Az $\mathcal{O}(f) = \{g \mid \exists c > 0, N > 0, g(n) < c \cdot f(n), \text{ ha } n > N\}$ halmazt az f **függvény növekedési rendjébe tartozó függvények osztályának** nevezzük.
Ha $g \in \mathcal{O}(f)$, azt mondjuk, hogy " g egy nagy ordó f " függvény.

3.2. Megjegyzés

A növekedési rend egyértelművé tételehez a legtisztább, ha f -ről feltesszük, hogy monoton növekvő, de ez nem feltétlenül szükséges. Ez a feltételezés sok vizsgálatot feleslegesen elbonyolítana, az érthetőséget rontaná.

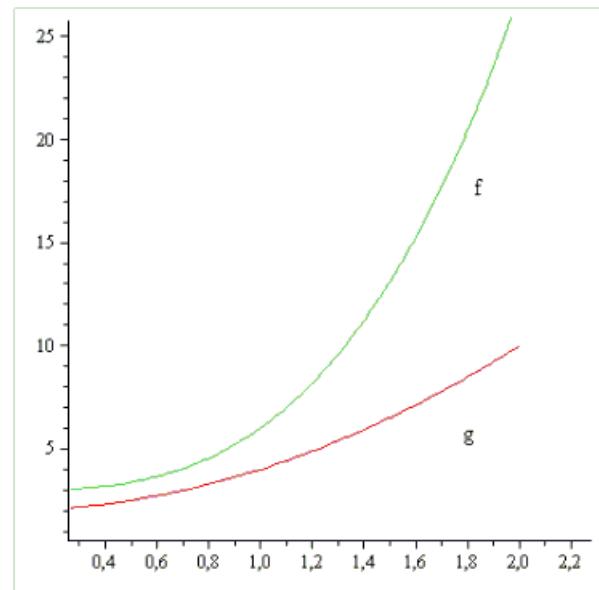
A $g \in \mathcal{O}(f)$ helyett gyakran a hagyományos $g = \mathcal{O}(f)$ jelölést használják.

A definíció alapján legnyilvánvalóbb példa, ha egy függvény felülről korlátoz egy másikat. Erre azonban nem feltétlenül van szükség. Az alábbiakban néhány szemléletes példán keresztül megvizsgáljuk, hogy első megközelítésben mit is fejez ki a növekedési rend.

Example 3.1. 1. példa

Ha feltételezzük, hogy $g(n) < f(n) \forall n \in \mathbb{N}$, akkor a definíció feltételei $c = 1$, $N = 1$ választással teljesülnek.

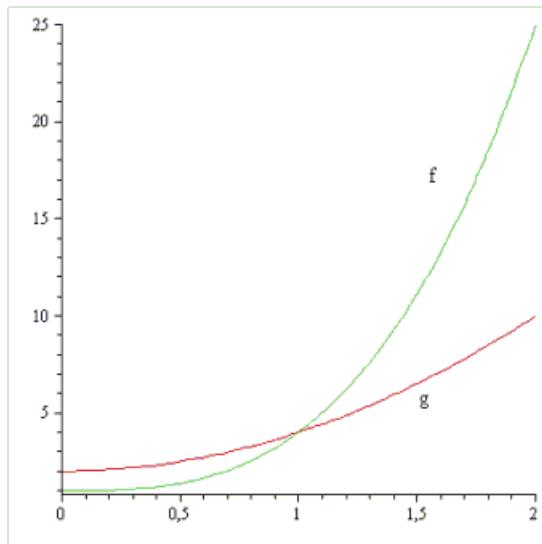
Figure 3.1. 1. ábra



Example 3.2. 2. példa

Ha feltételezzük, hogy $g(n) < f(n)$, ha $n > N$, kis n -ek esetén nem törődünk a függvények viszonyával.

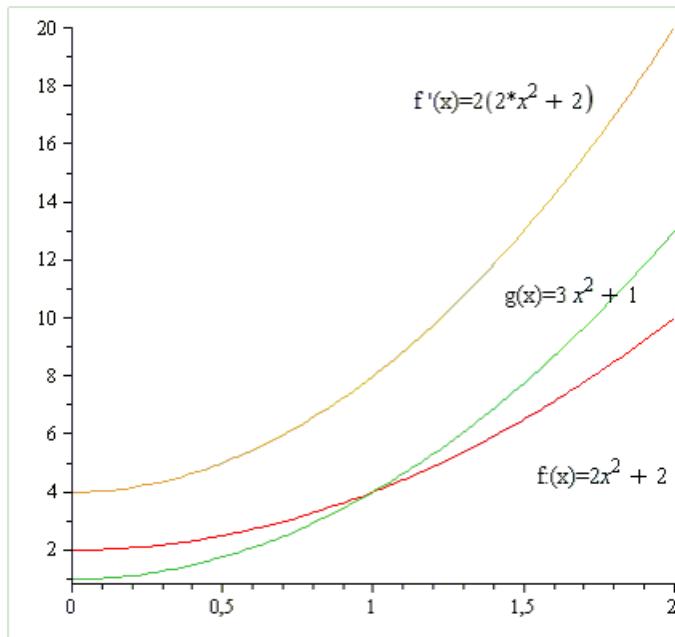
Figure 3.2. 2. ábra



Example 3.3. 3. példa

Itt $g(n) > f(n) \forall n \in \mathbb{N}$, viszont f egy c konstanssal megszorozva már nem kisebb mint g .

Figure 3.3. 3. ábra



3.3. Tulajdonságok

1. $f(n) \in O(f(n))$ (reflexivitás)
2. $n^k \in O(n^{k+1})$ minden $k > 0$ esetén
3. Legyen $f(n) \in O(g(n))$ és $g(n) \in O(h(n))$. Ekkor $f(n) \in O(h(n))$ (tranzitivitás)

Bizonyítás

1. Legyen $c = 2$ és $N = 0$. Mivel $f(n) > 0$ minden $n \in \mathbb{N}$ esetén, így $f(n) < 2f(n)$ minden $n > 0$ esetén.

2. Legyen $c = 1$ és $N = 1$. Ekkor $n^k < 1 \cdot n^{k+1}$, azaz $1 < n$, minden $n > 1$ esetén. ✓

3. Legyen c_1 és N_1 olyan, hogy $f(n) < c_1 g(n)$, ha $n > N_1$ és c_2 és N_2 olyan, hogy $g(n) < c_2 h(n)$, ha $n > N_2$. Ekkor $c = c_1 c_2$ és $N = \max(N_1, N_2)$ választással azt kapjuk, hogy $f(n) < c_1 g(n) < c_1(c_2 h(n)) = c \cdot h(n)$, ha $n > N_1$ és $n > N_2$, azaz $n > N$.

3.4. Megjegyzés

1. A tranzitív tulajdonság kifejezi azt az elvárásunkat, hogy egy f függvény gyorsabban nő, mint g , akkor minden olyan függvénytől is gyorsabban nő, amelyiktől g gyorsabb.
2. A növekedési rend definíciója alapján nem csak monoton függvényekre értelmezhető, de abban az esetben a vizsgálataink szempontjából nem értékes a jelentése.
3. A növekedési rendekre sem a szimmetria, sem az antiszimmetria nem teljesül. A szimmetriára ($g \in O(f) \Rightarrow f \in O(g)$), az $f = n$ és $g = n^2$ függvénypár, míg az antiszimmetriára ($g \in O(f), f \in O(g) \Rightarrow f = g$) az $f = n$ és $g = 2n$ függvénypár jelent ellenpéldát.

3.5. További tulajdonságok

4. Legyen $g_1 \in O(f_1)$ és $g_2 \in O(f_2)$. Ekkor $g_1 + g_2 \in O(f_1 + f_2) \Rightarrow f \in O(g)$.
5. Legyen $g_1 \in O(f_1)$ és $g_2 \in O(f_2)$. Ekkor $g_1 g_2 \in O(f_1 f_2) \Rightarrow f \in O(g)$.
6. Legyen f monoton növekvő, amire $f(n) > 1 \forall n \in \mathbb{N}$ és $g \in O(f)$. Ekkor $\log(g) \in O(\log(f))$.

Bizonyítás

3. Tegyük fel, hogy c_1 és c_2 -re illetve N_1 , N_2 -re teljesül, hogy $g_1(n) < c_1 f_1(n) \forall n > N_1$ és $g_2(n) < c_2 f_2(n) \forall n > N_2$. Legyen $c = \max(c_1, c_2)$ és $N = \max(N_1, N_2)$. Ekkor $g_1(n) + g_2(n) < c_1 f_1(n) + c_2 f_2(n) < c(f_1(n) + f_2(n)) \forall n > N$.

4. Hasonlóan az előzőhez, tegyük fel, hogy c_1 és c_2 -re illetve N_1 , N_2 -re teljesül, hogy $g_1(n) < c_1 f_1(n) \forall n > N_1$ és $g_2(n) < c_2 f_2(n) \forall n > N_2$. Legyen $c = c_1 c_2$ és $N = \max(N_1, N_2)$. Ekkor $g_1(n) g_2(n) < c_1 f_1(n) c_2 f_2(n) < c f_1(n) f_2(n) \forall n > N$.

5. Tegyük fel, hogy c -re illetve N -re teljesül, hogy $g(n) < c \cdot f(n) \forall n > N$. Az általánosság megszorítása nélkül feltehetjük, hogy $c > 1$. Mivel a $\log(\cdot)$ függvény szigorúan monoton növekvő, ezért $\log(g(n)) < \log(c \cdot f(n)) = \log(c) + \log(f(n)) \forall n > N$. Mivel $c \cdot f(1) > 1$, ezért $\log(c) + \log(f(1)) > 0$.

Legyen $c' = 1 + \frac{\log(c)}{\log(f(1))}$. f monotonitása miatt $f(1) < f(n) \forall n > 1$, és

$$\begin{aligned} \log(c) + \log(f(n)) &= \left(\frac{\log(c)}{\log(f(n))} + 1 \right) \log(f(n)) \\ &< \left(\frac{\log(c)}{\log(f(1))} + 1 \right) \log(f(n)) \\ &= c' \log(f(n)) \quad \forall n \in \mathbb{N} \quad \checkmark \end{aligned}$$

3.6. Következmények

1. Legyen $k_1, k_2 > 0$! Ekkor $n^{k_1} \in O(n^{k_2})$ akkor és csak akkor, ha $k_1 \leq k_2$.
2. Legyen $k_1, k_2 > 0$! Ekkor $k_1^n \in O(k_2^n)$ akkor és csak akkor, ha $k_1 \leq c \cdot k_2$.
3. Legyen $k > 0$, $k > 0$! Ekkor $n^k \in O(2^n)$ és $2^n \notin O(n^k)$.
4. Legyen $k > 0$! Ekkor $\log(n) \in O(n^k)$.

Bizonyítás

Az 1., 2., 3. és 4. következmények egyszerű megfontolással származtathatók az 4., 5. és 6. tulajdonságokból.

A következőkben definiálunk egy érdekes függvény sorozatot illetve hozzájuk kapcsolódóan egy rendkívül gyorsan növő függvényt.

3.7. Definíció

Legyen $f_i(x, y) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, $\forall i \in \mathbb{N}$ egy függvény sorozat, melyekre teljesülnek a következő tulajdonságok:

1. $f_0(x, y) = x + 1$
2. $f_i(x, 1) = f_{i-1}(x, 1)$
3. $f_i(x, y) = f_{i-1}(f_i(x, y - 1), 1)$

A függvény sorozat első néhány eleme a jól ismert aritmetikai műveleteket adja vissza.

1. f_0 a rákövetkező,
2. f_1 az összeadás, azaz $f_1(x, y) = x + y$
3. f_2 a szorzás, azaz $f_2(x, y) = x \cdot y$
4. f_3 a hatványozás, azaz $f_3(x, y) = x^y$

művelete.

A definíció lényegében a szokásos meghatározásoknak a kiterjesztése:

1. az $x + y$ összeadás értelmezése, hogy x -et y -szor növeljük 1-el;
2. a szorzás értelmezése, hogy x -et y -szor adjuk össze önmagával;
3. a hatványozásnál x -et y -szor szorozzuk össze önmagával; stb.

Ilyen módon egyre gyorsabban növő függvényeket kapunk. A függvény sorozat felhasználva tudunk készíteni egy olyan függvényt, amelyik a sorozat minden elemétől jobban nő: $F(x) = f_x(x, x)$. Szokás közvetlenül, a függvény sorozat felhasználása nélkül is definiálni egy ehhez kapcsolódó, rendkívül gyorsan növő függvényt.

3.9. Definíció

Ackermann-függvény:

Legyen $f: \mathbb{N} \rightarrow \mathbb{R}^+$ függvény adott a következő rekurzív definícióval $\forall y \in \mathbb{N}$

1. $A(x, 0) = A(x-1, 1) \quad \forall x \in \mathbb{N} \setminus \{0\}$
2. $A(x, y) = A(x-1, A(x, y-1)) \quad \forall x, y \in \mathbb{N} \setminus \{0\}$
3. A

Az f függvényt **Ackermann-függvénynek** nevezzük.

Az $F(x)$ nagyjából az $A(x, x)$ -nek megfelelő értéket vesz fel (valamivel kisebb). F első néhány függvényértéke:

$$F(0) = f_0(0, 0) = 1$$

$$F(1) = f_1(1, 1) = 1+1 = 2$$

$$F(2) = f_2(2, 2) = 2 \cdot 2 = 4$$

$$F(3) = f_3(3, 3) = 3^3 = 27$$

$$F(4) = f_4(4, 4) = 4^{4^{4^4}} = 4^{4^{4^{256}}} \approx 4^{4^{1024^k}} \approx 4^{4^{10^{150}}} \approx 4^{4^k}, \text{ ahol } k \text{ egy több mint } 150 \text{ jegyű szám.}$$

Világos, hogy ez az érték messze meghaladja a az elközelhető, vagy akár a lejegyezhető értéket. (Jóval nagyobb, mint a világegyetem összes atomjának száma.) És a növekedése egyre nagyobb ütemű.

A növekedési rendek pontosabb kifejezésére más definíciókat is szokás használni. Ezek közül néhány fontosabbat megadunk a következő oldalon, megvizsgálva egy-két alapvető tulajdonságukat.

3.10. Definíció

Legyen $f: \mathbb{N} \rightarrow \mathbb{R}^+$ egy függvény.

$\Theta(f) = \{g | \exists c_1, c_2 > 0, N > 0, c_1 f(n) < g(n) < c_2 f(n), \text{ ha } n > N\}$

halmazt az f függvény **pontos növekedési rendjébe** tartozó függvények osztályának nevezzük.

3.11. Tulajdonságok

1. Legyen $g, f: \mathbb{N} \rightarrow \mathbb{R}^+$ két függvény. Ekkor $g \in \Theta(f)$ akkor és csak akkor, ha $f \in \Theta(g)$.

2. Legyen $g, f: \mathbb{N} \rightarrow \mathbb{R}^+$ két függvény. Ekkor $g \in \Theta(f)$ akkor és csak akkor, ha $g \in O(f)$ és $f \in O(g)$.

Bizonyítás

1. Legyen $c_1, c_2 > 0$ és $N > 0$ olyan, hogy $c_1 f(n) < g(n) < c_2 f(n)$, ha $n > N$, valamint $c_1' = \frac{1}{c_2}$ és $c_2' = \frac{1}{c_1}$. Ekkor az első egyenlőtlenség alapján $c_1 f(n) < g(n) < c_2 f(n) \Rightarrow \frac{f(n)}{c_1} < g(n) = c_2' g(n)$, ha $n > N$ illetve a második egyenlőtlenség alapján $c_2 f(n) > g(n) \Rightarrow \frac{f(n)}{c_2} > g(n) = c_1' g(n)$, ha $n > N$. Ezzel tulajdonképpen az állítást igazoltuk.

2. A $g \in O(f)$ és $f \in O(g)$ relációk definíciója alapján $\exists c_1 > 0$ és $N_1 > 0$ valamint $\exists c_2 > 0$ és $N_2 > 0$, amelyekre $g(n) < c_1 f(n)$, ha $n > N_1$ és $f(n) < c_2 g(n)$, ha $n > N_2$. Legyen $N = \max(N_1, N_2)$, $c_1' = \frac{1}{c_2}$ és $c_2' = c_1$. Az előző két egyenlőtlenség alapján azt kapjuk, hogy $c_1' f(n) < g(n) < c_2' f(n)$, ha $n > N$. ✓

3.12. Definíció

Legyen $f: \mathbb{N} \rightarrow \mathbb{R}^+$ egy függvény. Az $\circ(f) = \{g | \forall c > 0, \exists N > 0, g(n) < cf(n), \text{ ha } n > N\}$ halmazt az f függvénytől **kisebb növekedési rendű** függvények osztályának nevezzük.
Ha $g \in \circ(f)$, azt mondjuk, hogy " g egy kis ordó f " függvény.

3.14. Tulajdonságok

1. Legyen $g, f: \mathbb{N} \rightarrow \mathbb{R}^+$ két függvény. Ekkor $g(n) \in \circ(f(n))$ akkor és $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$ csak akkor, ha $\circ(g) \subseteq \circ(f)$.
2. Legyen $f: \mathbb{N} \rightarrow \mathbb{R}^+$ egy függvény. Ekkor $\circ(f(n)) = \Theta(f(n)) \cup \circ(f)$.
3. Legyen $f: \mathbb{N} \rightarrow \mathbb{R}^+$ egy függvény. Ekkor $\Theta(f(n)) \cap \circ(f(n)) = \emptyset$.

Bizonyítás

1. Legyen $c > 0$ és $N_c > 0$ olyan, hogy $g(n) < c \cdot f(n)$, ha $n > N_c$. Ekkor $0 < \frac{g(n)}{f(n)} < c$, ha $n > N_c$, azaz $\frac{g(n)}{f(n)} < c$ akármilyen kis $c > 0$ esetén megadható egy N_c -től nagyobb n -ek esetén. Ez a határérték definíciója alapján pontosan azt jelenti, hogy $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$. Visszafelé, $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$ azt jelenti, hogy $0 < \frac{g(n)}{f(n)} < c$ $\forall c > 0$ esetén $\exists N > 0$, amelyikre $\frac{g(n)}{f(n)} < c$, ha $n > N$. Ez alapján $f(n)$ -nel beszorozva pontosan a keresett állítást kapjuk.

2. A definíciók alapján világos, hogy $\circ(f(n)) \subseteq \circ(f(n))$ és $\Theta(f(n)) \subseteq \circ(f(n))$. Legyen $g(n) \in \circ(f(n)) \setminus \Theta(f(n))$. A definíciók alapján ekkor $\exists c_2 > 0$ és $N_2 > 0$, amelyekre $g(n) < c_2 f(n)$, ha $n > N_2$, és $\forall c_1 > 0$ esetén $\exists N_{c_1} > 0$ úgy, hogy $c_1 f(n) \geq g(n)$ ha $n > N_{c_1}$. Legyen $N'_1 = N_{c_1+1}$. Erre az N'_1 -re igaz, hogy $c_1 f(n) > (c_1+1) f(n) \geq g(n)$ ha $n > N'_{c_1+1}$, ami pontosan a bizonyítandó állítás.

3. Indirekt módon, ha feltételezzük, hogy $\Theta(f(n)) \cap \circ(f(n)) \neq \emptyset$, akkor $\exists g(n) \in \Theta(f(n)) \cap \circ(f(n))$. Erre a g -re igaz, hogy $\exists c_1 > 0$ és $\exists N_1 > 0$, amelyekre $c_1 f(n) < g(n)$ ha $n > N_1$, továbbá $\forall c > 0$, $\exists N_c > 0$ úgy, hogy $g(n) < c f(n)$, ha $n > N$. Legyen $N' = \max(N_1, N_{c_1})$. Erre az N' -re igaz, hogy $c_1 f(n) < g(n)$ és $g(n) < c_1 f(n)$, ha $n > N'$, ami ellentmondás. ✓

Chapter 4. A Turing-gép

Az algoritmus fogalmáról az informatikával szorosabb kapcsolatot ápolóknak van valamelyen elképzelése. A legtöbb esetben meg tudjuk mondani, hogy az éppen vizsgált dolog algoritmus-e vagy sem. Legalábbis a "hétköznapi" esetekben. Sokan asszociálnak az algoritmusok kapcsán számítógép-programokra, nem teljesen alaptalanul. Nem egyértelmű azonban, hogy például egy orvosi vizsgálat, vagy akár a járás tekinthető-e algoritmusnak.

Minél inkább próbáljuk pontosítani a fogalmat, annál inkább ütközünk korlátokba. Az algoritmusokkal kapcsolatban a következő elvárásaink vannak:

Az algoritmusnak jól meghatározott feladatot illetve feladattípust kell megoldania.

Az algoritmus elkülöníthető lépésekkel álljon, ezen lépések száma (típusa) véges legyen. Szeretnénk a feladatra adott megoldást véges időben megkapni.

Az algoritmus minden egyes lépésének pontosan definiáltnak kell lennie.

Az algoritmus számára szükség lehet bizonyos bemenő adatokra, amivel a megoldandó feladatosztály egy speciális feladatát jelöljük ki. Az adatok mennyisége csak véges lehet, annak minden értelmében.

Az algoritmus válaszoljon a bemenetre. A válasznak megfelelően értelmezhetőnek és természetesen végesnek kell lennie.

Látható, hogy ha megfelelően általánosan akarjuk meghatározni az algoritmus fogalmát, akkor több bizonytalanságot kell hagynunk benne. A legproblémásabb rész a "lépés" fogalma. Mit jelent az, hogy "pontosan definiált"? Ha megadunk egy igazán pontos definíciót, márás leszűkitjük a lehetőségeinket.

Az algoritmus mibenlétének pontos meghatározásával a múlt század első felében többen is próbálkoztak. Ennek eredményeképpen több, egymástól különböző fogalom is megszületett, mint például a λ függvények, rekurzív függvények, Markov-gép és a Turing-gép. Mivel ezek pontosan definiált modellek, az az érzésünk lehet, hogy nem tekinthetők teljes mértékben alkalmasnak az algoritmus fogalmának helyettesítésre.

Belátható, hogy az előbb felsorolt modellek egymással ekvivalensek, ami azt jelenti, hogy minden feladat, ami az egyik modellben megoldható, megoldható a másikban is.

Az algoritmus fogalmának helyettesítésére azonban mind a mai napig nem találtak a felsoroltaknál teljesebb modellt. A jegyzetben a legnagyobb figyelmet a Turing-gép modelljének fogjuk szentelni, mivel az egyik leginkább letisztult, világosan érthető és sok szempontból kifejező fogalomnak bizonyult. Igaz ez annak ellenére is, hogy a ma legszéleskörűbben elterjedt számítógép-architektúrák modellezésére nem kifejezetten alkalmas.

Turing-gépek segítségével könnyen ki tudjuk fejezni a kiszámíthatóság, azaz az algoritmussal való megoldhatóság fogalmát, és meg tudunk határozni egy precíz mértéket az algoritmusok bonyolultságának valamint a feladatok nehézségének leírására is.

A Turing-gép elnevezés a modell kitalálójára, Alan Turingra (1912-1954) utal. A tényleges definícióinak számtalan azonos értelmű változata létezik. Ezek közül mi az egyik leginkább letisztult, megfelelően kifejező változatot használjuk.

Ahhoz, hogy egyáltalan megpróbálhassuk matematikai eszközökkel leírni az algoritmusokat, szükiéri kell a megoldásra váró feladatok körét. Ki kell zárnunk a lehetőségek közül többek között a fizikai objektumokon vérehajtott mechanikai műveleteket. Ez nem jelenti azt, hogy az ilyen jellegű problémákat nem tekintjük algoritmikusan megoldhatónak, csak annyi, hogy a fizikai algoritmus helyett annak egy matematikai modelljét tudjuk csak kezelní, és egy megfelelő interfész segítségével fizikai műveletekké alakítani. Ez lényegében a gyakorlatban minden esetben így működik, hiszen önmagukban a vérehajtott algoritmusaink, illetve a nekik megfelelő programjaink eredményeit nem észlelhetnénk.

Feltételezzük tehát, hogy a feladat és annak paraméterei, bemenő adatai valamelyen véges reprezentációval leírhatóak. Ennek megfelelően a továbbiakban az algoritmusaink bemenetét (a feladat leírását) egy rögzített véges ábécé feletti szóként adjuk meg, és hasonló formában várjuk a választ is. Egy algoritmust tekinthetünk

tehát úgy, mint egy leképezést, amely szavakhoz szavakat rendel. Világos, hogy meg tudunk adni olyan algoritmusokat, amelyek bizonyos bemenetekre "nem reagálnak", azaz nem adnak kimenetet (ilyenkor parciális függvényt valósítanak meg). Elképzelhetők olyan speciális algoritmusok is, amelyek a bemenő szavak végtelen számoságára ellenére csak véges sok lehetséges választ adhatnak. Erre példa a klasszikus elfogadó (felismerő) algoritmus, amely a bemenet értékétől függően igennel vagy nemmel válaszol (elfogadja, illetve elutasítja a bemenetet).

4.1. Definíció

Legyen V egy ábécé, $L \subseteq V^*$ egy nyelv, $w \in V^*$ és $\phi: L \rightarrow V^*$ egy transzformáció.

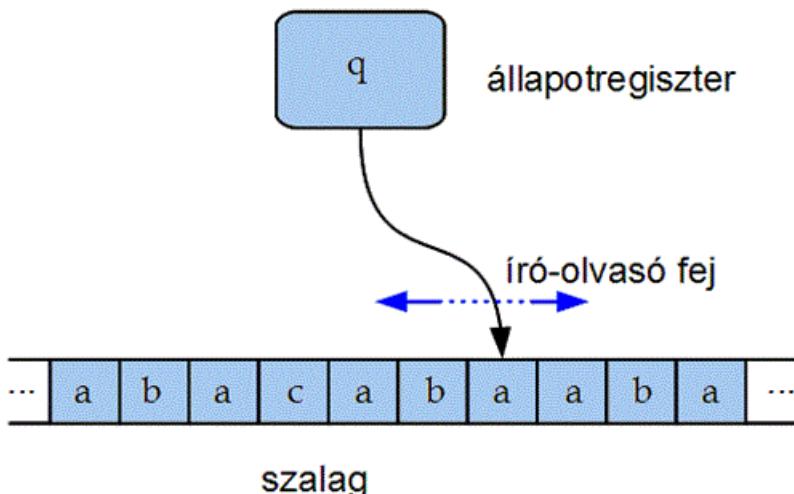
Algoritmikus feladatnak (vagy egyszerűen csak **feladatnak**) nevezzük a következőket:

1. Határozzuk meg, hogy fennáll-e a $w \in L$ reláció!
2. Határozzuk meg $\phi(w)$ értékét!

Az 1. típusú feladatot **döntési (felismerési) feladatnak**, míg a 2. típusút **transzformációs feladatnak** nevezzük.

A későbbiekben látni fogjuk, hogy már az előbb említett, egyszerűnek látszó döntési feladat sem könnyű. Bebizonyítjuk, hogy vannak olyan problémák, amelyekről nem tudjuk eldönthetni még azt sem, hogy egyáltalán megoldhatóak-e.

Figure 4.1. Turing gép modell



1. A Turing-gép definíciója

4.2. Definíció

A $T = (Q, \Sigma, s, \delta, H)$ ötöst **Turing-gépnek** nevezzük, ha

Q : véges, nem üres halmaz; **állapotok halmaza** Σ véges, legalább 2 elemű halmaz, $* \in \Sigma$;

szalag ábécé $s \in Q$; **kezdő állapot** $H \subseteq Q$, nem üres; **végállapotok halmaza**

$\delta: (Q \setminus H) \times \Sigma \rightarrow Q \times (\Sigma \cup \{ \leftarrow, \Rightarrow \})$; **átmenetfüggvény**

A fenti formális definíciót megfelelő értelmezéssel kell ellátnunk. A szemünk előtt a következő "fizikai" modell fog lebegni:

A Turing-gép három fő részből áll:

1. egy minden két irányban végtelen szalag, cellákra osztva, a cellák tartalma a Σ ábécé elemei közül kerül ki;

a szalagon legfeljebb véges sok cellában van Σ -beli elem és ezek között nem lehet * (összefüggően helyezkednek el az értékes jelek);

2. egy regiszter, \mathcal{Q} -beli értéket tartalmaz, ez határozza meg a Turing-gép pillanatnyi működését;

3. egy író-olvasó fej, ami mindenkor egy konkrét cellára mutat; ez kapcsolja össze a szalagot a regiszterrel.

A Turing-gép egy lépése során beolvassa a szalagról az író-olvasó fej alatt levő jelet, a beolvastott jel értékétől, és a regiszterben tárolt állapottól függően a δ által meghatározott módon lép:

- visszaír egy jelet az író-olvasó fej alatti cellába és megváltoztatja az állapotát vagy

- a szomszédos cellák valamelyikére mozdítja az író-olvasó fejet és megváltoztatja az állapotát.

A Turing-gép alternatív meghatározásában általában egyidejűleg történik a szalagra írás és fejmozgatás, a szétválasztás segítségével azonban bizonyos definíciókat lényegesen egyszerűbben és letisztultabban adhatunk meg.

A szem előtt tartott értelmezésnek megfelelően szükség van matematikailag is pontos, jól meghatározott "működésre". Ezt írjuk le a következő definíciókkal.

4.3. Definíció

A T Turing-gép egy **konfigurációja** $K : K \in Q \times ((\Sigma^* \times \Sigma \times \Sigma^*) \setminus (\Sigma^* \times \{*\} \times \Sigma^*))$
($=$ regiszterállapot + szalagtartalom, az író-olvasó fej helyének jelölésével)

4.4. Megjegyzés

Egy T Turing-gépnek csak akkor létezik $K = (q, w_1, *, w_2)$ alakú konfigurációja,
ha $w_1 = \lambda$ vagy $w_2 = \lambda$.

4.5. Definíció

Azt mondjuk, hogy a T Turing-gép. **egy lépéssben** (vagy közvetlenül) **átmegy** K -ból a K' konfigurációba (jelekben $K \vdash K'$), ha
 $K = (q, w_1, a, w_2)$ $K' = (q', w'_1, a', w'_2)$
és a következők közül pontosan egy teljesül:
1) $w'_1 = w_1$ $w'_2 = w_2$ $\delta(q, a) = (q', a')$, ahol $a, a' \in \Sigma$, $q \in Q \setminus H$ és $q' \in Q$.
--- felülírási üzemmód
2) $w'_1 = w_1 \cdot a$ $w'_2 = a' \cdot w'_2$ $\delta(q, a) = (q', \Rightarrow)$, ahol $\delta(q, a) = (q', \Rightarrow)$, $q \in Q \setminus H$ és $q' \in Q$.
--- jobbra lépési üzemmód
3) $w_1 = w'_1$ $a' \cdot w'_2 = a \cdot w_2$ $\delta(q, a) = (q', \Leftarrow)$, ahol $a \in \Sigma$, $q \in Q$ és $q' \in Q$.
--- balra lépési üzemmód

4.6. Megjegyzés

- Mivel $\delta(q, a)$ minden $q \in Q$ és a $a \in \Sigma$ esetén egyértelműen definiált, ezért minden szabályos konfiguráció esetén vagy létezik egyértelműen K' konfiguráció, amelyikbe közvetlenül átmegy, vagy egyáltalán nem létezik ilyen.
- A Turing-gép konfigurációjának definíciója, illetve a definíció utáni megjegyzés alapján, ha $K = (q, w_1, a, w_2)$ és $\delta(q, a) = (q', *)$, akkor a Turing-gép csak tud továbblépni, ha $w_1 = \lambda$ vagy $w_2 = \lambda$.
- A * jel tulajdonságai alapján, ha $\delta(q, *) = (q', \Rightarrow)$ és $K = (q, w_1, *, \lambda)$, a $K \vdash K'$ átmenetben $K' = (q', w_1, *, \lambda)$.

Hasonlóképpen, ha $\delta(q, *) = (q', \leftarrow)$ és $K = (q, \lambda, *, w_2)$, akkor a $K \vdash K'$ átmenetben $K' = (q', \lambda, *, w_2)$.

4.7. Definíció

A T Turing-gép egy **számítása** konfigurációk egy $K_0, K_1, \dots, K_n, \dots$ sorozata ($i = 0, 1, \dots$) amelyekre

1. $K_0 = (s, \lambda, w[1], w[2 \dots n])$, ahol $n = l(w)$;
2. $K_i \vdash K_{i+1}$, ha létezik K_i -ból közvetlenül elérhető konfiguráció (és ez az egyértelműség miatt K_{i+1});
3. $K_i = K_{i+1}$, ha nem létezik K_i -ból közvetlenül elérhető konfiguráció.

A w szót a T **bemenetének** nevezzük.

Ha $K_n = (h, w_1, \alpha, w_2)$ olyan, hogy $h \in H$, akkor azt mondjuk, hogy a számítás véges, a Turing-gép a h végállapotban megáll. Ekkor a $w' = w_1 \cdot \alpha \cdot w_2$ szót T **kimenetének** nevezzük.

Jelölése: $w' = T(w)$.

Ha egy $w \in \Sigma^*$ esetén a T Turing-gépnek végtelen számítása van, akkor nem értelmezünk kimenetet. Jelekben: $T(w) = \emptyset$. (Nem összetévesztendő a $T(w) = \lambda$ kimenettel.)

4.8. Megjegyzés

Egy T Turing-gépnek kétféleképpen lehet végtelen számítása egy w bemeneten:

1. $\forall i$ -re létezik K_i -ból közvetlenül elérhető konfiguráció;
2. $\exists i$, amelyikre K_i -ból nincs közvetlenül elérhető konfiguráció és K_i -hez tartozó állapot nem végállapot. (A Turing-gép "befagy" a K_i konfigurációban.)

2. Felismerő Turing-gépek

4.9. Definíció

Legyen $T = (Q, \Sigma, s, \delta, H)$ egy Turing-gép, $H = H^+ \cup H^-$ és $H^+ \cap H^- = \emptyset$.

Ekkor T -t elfogadó Turing-gépnek nevezzük, a H^+ -beli állapotokat pedig elfogadó állapotoknak.

4.10. Definíció

Legyen T egy elfogadó Turing-gép, H^+ a T elfogadó állapotaiból halmaza. Azt mondjuk, hogy T elfogadja a $w \in \Sigma^*$ szót, hogyha T számítása véges a w bemeneten, és megálláskor H^+ -beli állapotban van.

4.11. Definíció

Legyen T egy elfogadó Turing-gép.

Az $L(T) = \{w \mid w \in \Sigma^* \text{ és } T \text{ elfogadja } w\}$ nyelvet a T által felismert nyelvnek nevezzük.

4.12. Lemma

Legyen T egy elfogadó Turing-gép és $L = L(T)$. Ekkor $\exists T'$ átalakító Turing-gép, amelyikre $T'(w) = 1$ pontosan akkor, ha $w \in L$.

Bizonyítás

Az előbbi lemma alapján megállapíthatjuk, hogy az elfogadó Turing-gépek lényegében nem jelentenek újdonságot az átalakító Turing-gépekhez képest, azonosnak tekinthetők az általuk felismert nyelv karakterisztikus függvényét megvalósító Turing-géppel.

3. Turing-gépek megadása

Egy Turing-gép megadása a definícióban szereplő ötös leírását jelenti. $(Q, \Sigma, q_0, H, \delta)$ esetében ez nem jelent különösebb kihívást, viszont az átmenetfüggvény meghatározása kicsit több erőfeszítést igényel. Mivel azonban véges halmazon értelmezett, véges halmazba képező függvényről van szó, szerencsére több egyszerű lehetőségünk is adódik rá.

A különböző reprezentációk használatát két Turing-gép különböző megadásával szemléltetjük.

Az első Turing-gép által megvalósított feladat a $\{0, 1\}$ ábécé fölötti szavakon végrehajtott paritásellenőrző bites kódolás. Ez egy tipikus transzformációs feladat. Kezdőszövelettartó, viszont nem hossztartó átalakítás. A leképezés lényege, hogy a bemenő szóban megszámoljuk az 1 jeleket és a szót kiegészítjük egy új jellel úgy, hogy a keletkezett szóban az 1-esek száma páros legyen. Ez a transzformáció az egyik legegyszerűbb, de az egyik leghatékonyabb és széles körben alkalmazott hibafelismerő kódolás.

Pl. Paritásbit hozzáfűzése:

Be: 01001011010 Ki: 010010110101

Be: 1101101010110 Ki: 11011010101100

A tényleges alkalmazásoknál általában feltételezik, hogy az egyes kódszavak azonos hosszúságúak, de a példában nem élünk ezzel a megszorítással.

A második Turing-gép segítségével azt ellenőrizzük, hogy a bemenő szó megfelel-e a paritási feltételnek, azaz páros sok 1 található benne. Ha igen, akkor elfogadjuk, ha nem, akkor sérültnek tekintjük.

3.1. Táblázatos reprezentáció:

A táblázatos reprezentációban az átmenetfüggvényt többféleképpen is megadhatjuk. Egyik lehetőség, hogy a táblázat oszlopait állapotokhoz, sorait bemenetekhez (az író-olvasó fej által beolvasható értékekhez) rendeljük, a cellák tartalma pedig a függvényértékeket tartalmazza a megfelelő argumentumok esetén.

A táblázat természetesen hiányos is lehet, amennyiben az átmenetfüggvény nem mindenütt definiált.

Pl. Paritásbit hozzáfűzése

A szakasz elején megadott feladat megoldására alkalmas Turing-gép a következő:

$$T = (Q, \Sigma, q_0, \delta, H)$$

ahol $Q = \{q_0, q_1, q_2, q_3, q_s\}$, $\Sigma = \{0, 1\}$, $H = \{q_s\}$ és δ :

Figure 4.2. Paritásellenőrző bit hozzáfűzése a bemenő szóhoz

δ	Q	q_0	q_1	q_2	q_3	q_s
0		$q_1, 0$	q_0, \Rightarrow	$q_3, 1$	q_2, \Rightarrow	
1		$q_3, 1$	q_0, \Rightarrow	$q_1, 1$	q_2, \Rightarrow	
$*$		$q_s, 0$		$q_s, 1$		

A Turing-gép q_s állapota végállapot, a q_0 kezdő állapot és egyben annak jelzésére szolgál, hogy az eddig megvizsgált jelsorozatban páros sok 1 található. A q_2 állapot jelentése, hogy az eddigi vizsgálatok során páratlan sok 1 található. A q_1 és q_3 állapotok az író-olvasó fej továbbítására és a paritásérték megjegyzésére szolgálnak.

Egy konkrét bemeneten a Turing-gép a következő számítást végez el:

Be: 011001

Figure 4.3. Kezdő konfiguráció: K_0

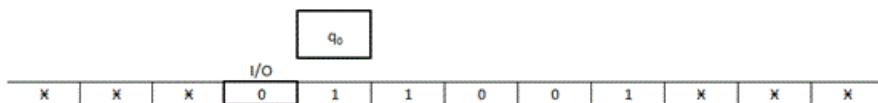


Figure 4.4. K_1

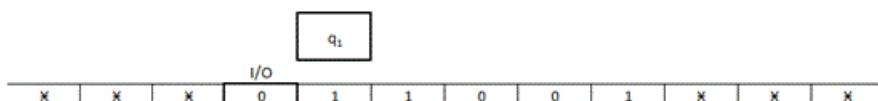


Figure 4.5. K_2

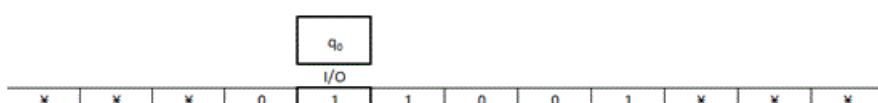


Figure 4.6. K_3

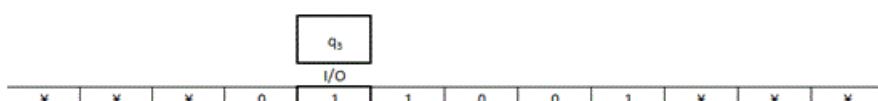


Figure 4.7. K_4

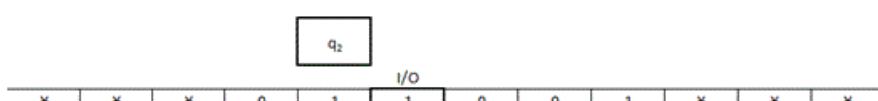


Figure 4.8. K_5

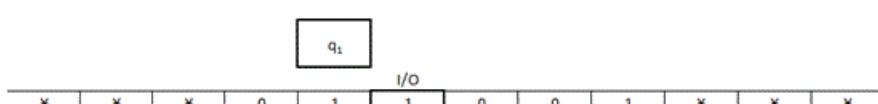


Figure 4.9. K_6

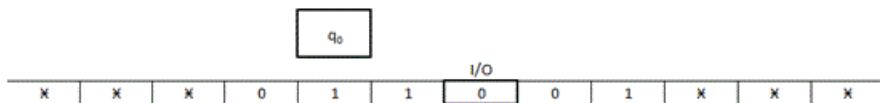


Figure 4.10. K_7

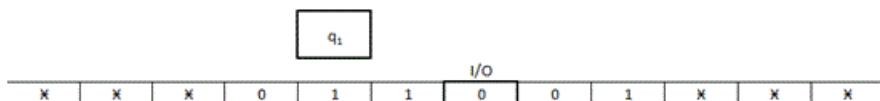


Figure 4.11. K_8

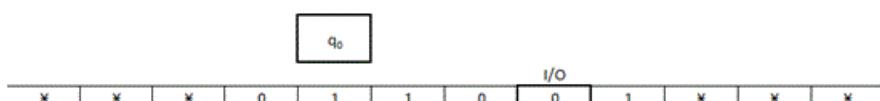


Figure 4.12. K_9

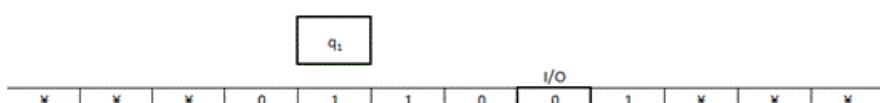


Figure 4.13. K_{10}

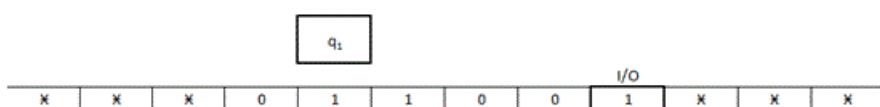


Figure 4.14. K_{11}

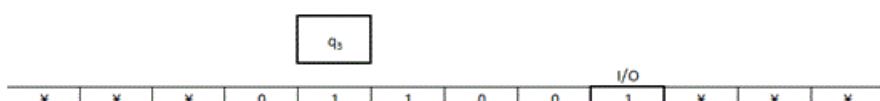


Figure 4.15. K_{12}

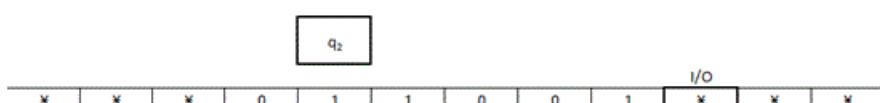
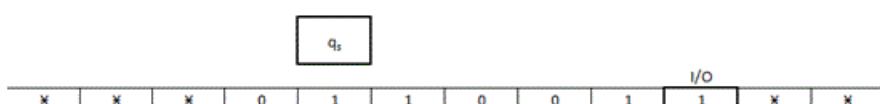


Figure 4.16. K_{13}



Látható, hogy a Turing-gép számításának leírása így meglehetősen kényelmetlen, bár könnyen feldolgozható. Sokkal egyszerűbb az eredeti definíciók megfelelően a konfigurációk leírásával megadni.

Figure 4.17. A Turing-gép számítása a 011001 bemeneten

K_0	=	$(q_0, \lambda, 0, 110010)$
K_1	=	$(q_1, \lambda, 0, 110010)$
K_2	=	$(q_0, 0, 1, 10010)$
K_3	=	$(q_3, 0, 1, 10010)$
K_4	=	$(q_2, 01, 1, 0010)$
K_5	=	$(q_1, 01, 1, 0010)$
K_6	=	$(q_0, 011, 0, 010)$
K_7	=	$(q_1, 011, 0, 010)$
K_8	=	$(q_0, 0110, 0, 10)$
K_9	=	$(q_1, 0110, 0, 10)$
K_{10}	=	$(q_0, 01100, 1, 0)$
K_{11}	=	$(q_3, 01100, 1, 0)$
K_{12}	=	$(q_2, 011001, 0, \lambda)$
K_{13}	=	$(q_3, 011001, 0, \lambda)$
K_{14}	=	$(q_2, 0110010, *, \lambda)$
K_{15}	=	$(q_s, 0110010, *, \lambda)$

Pl. Paritásbit ellenőrzése

$$T = (\mathcal{Q}, \Sigma, q_0, \delta, H)$$

ahol $\mathcal{Q} = \{q_0, q_1, q_2, q_3, q_y, q_n\}$, $\Sigma = \{0, 1\}$, $H = \{q_y, q_n\}$ és δ :

Figure 4.18. Paritásellenőrző bit tesztelése

Σ	q	q_0	q_1	q_2	q_3	q_y	q_n
0		$q_1, 0$	q_0, \Rightarrow	$q_3, 1$	q_2, \Rightarrow		
1		$q_3, 1$	q_0, \Rightarrow	$q_1, 1$	q_2, \Rightarrow		
*		$q_y, *$		$q_n, *$			

A Turing-gép q_y állapota elfogadó-, míg q_n elutasító állapot, azaz $H^+ = \{q_y\}$ és $H^- = \{q_n\}$ a q_0 kezdő állapot és egyben annak jelzésére szolgál, hogy az eddig megvizsgált jelsorozatban páros sok 1 található. A q_2 állapot jelentése, hogy az eddigi vizsgálatok során páratlan sok 1 található. A q_1 és q_3 állapotok az író-olvasó fej továbbítására és a paritásérték megjegyzésére szolgálnak.

Egy konkrét bemeneten a Turing-gép a következő számítást végzi el:

Be: 0110010

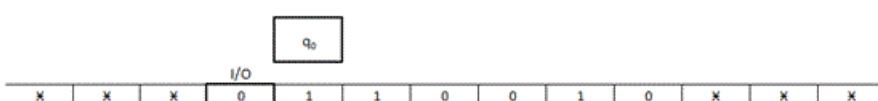
Figure 4.19. Kezdő konfiguráció: K_0 

Figure 4.20. K_1



Figure 4.21. K_2

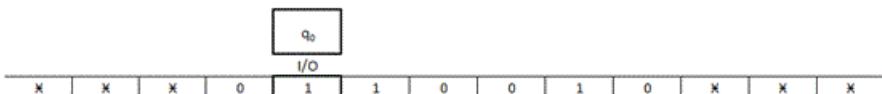


Figure 4.22. K_3



Figure 4.23. K_4

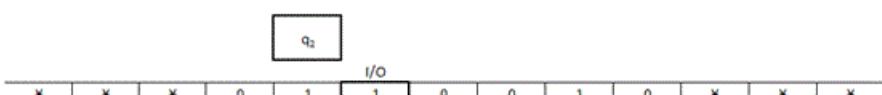


Figure 4.24. K_5



Figure 4.25. K_6

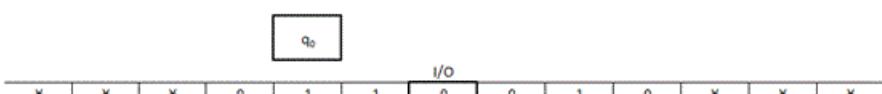


Figure 4.26. K_7

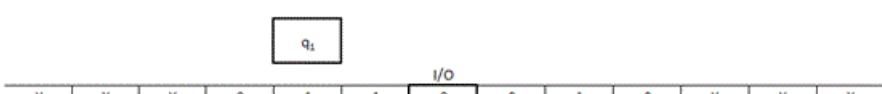


Figure 4.27. K_8



Figure 4.28. K_9

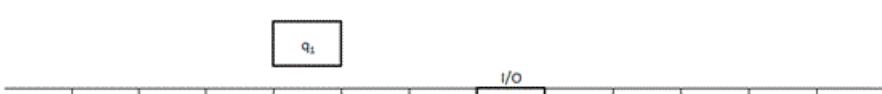


Figure 4.29. K_{10}

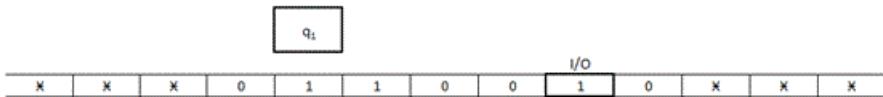


Figure 4.30. K_{11}

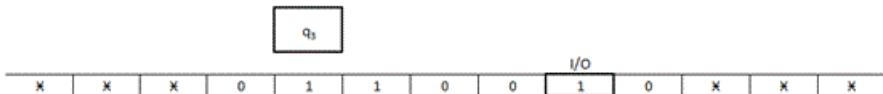


Figure 4.31. K_{12}

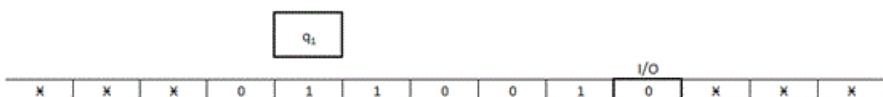


Figure 4.32. K_{13}



Figure 4.33. K_{14}

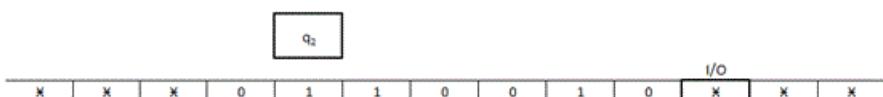


Figure 4.34. K_{15}

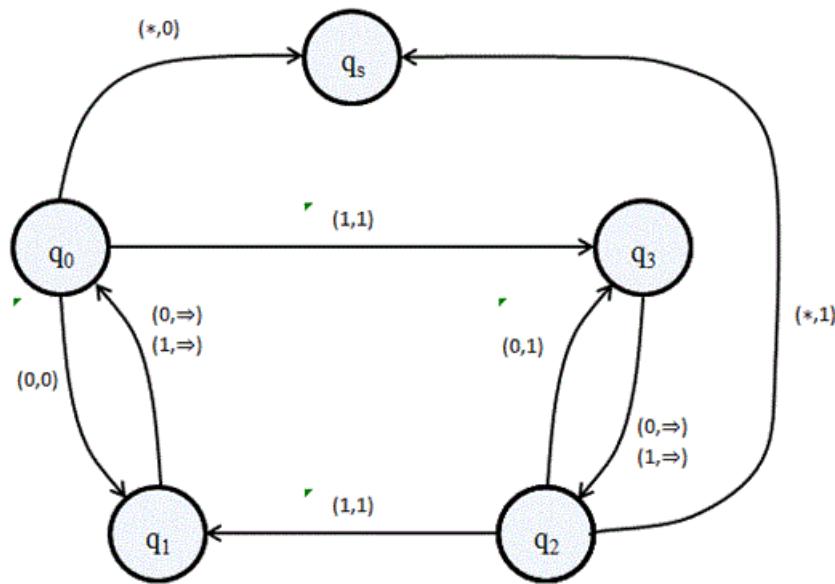


A Turing-gép számításának leírása a konfigurációi segítségével:

Figure 4.35. A Turing-gép számítása a 0110010 bemeneten

K_0	$= (q_0 , \lambda , 0 , 110010)$
K_1	$= (q_1 , \lambda , 0 , 110010)$
K_2	$= (q_0 , 0 , 1 , 10010)$
K_3	$= (q_3 , 0 , 1 , 10010)$
K_4	$= (q_2 , 01 , 1 , 0010)$
K_5	$= (q_1 , 01 , 1 , 0010)$
K_6	$= (q_0 , 011 , 0 , 010)$
K_7	$= (q_1 , 011 , 0 , 010)$
K_8	$= (q_0 , 0110 , 0 , 10)$
K_9	$= (q_1 , 0110 , 0 , 10)$
K_{10}	$= (q_0 , 01100 , 1 , 0)$
K_{11}	$= (q_3 , 01100 , 1 , 0)$
K_{12}	$= (q_2 , 011001 , 0 , \lambda)$
K_{13}	$= (q_3 , 011001 , 0 , \lambda)$
K_{14}	$= (q_2 , 0110010 , * , \lambda)$
K_{15}	$= (q_s , 0110010 , * , \lambda)$

3.2. Gráfprezentáció:



3.3. Turing-gép megadása felsorolással:

Az átmenetfüggvény megadható a definiált argumentumokon felvett értékek felsorolásával is.

A paritásbit generáló Turing-gép átmenetfüggvénye:

$$\delta =$$

$(q_0, 0) \rightarrow (q_1, 0)$ $(q_0, 1) \rightarrow (q_3, 0)$ $(q_0, *) \rightarrow (q_1, 0)$ $(q_1, 0) \rightarrow (q_0, \Rightarrow)$ $(q_1, 1) \rightarrow (q_0, \Rightarrow)$ $(q_2, 0) \rightarrow (q_3, 0)$ $(q_2, 1) \rightarrow (q_1, 0)$ $(q_2, *) \rightarrow (q_1, 1)$ $(q_3, 0) \rightarrow (q_2, \Rightarrow)$ $(q_3, 1) \rightarrow (q_2, \Rightarrow)$

A paritásbit ellenőrző Turing-gép átmenetfüggvénye:

 $\delta =$ $(q_0, 0) \rightarrow (q_1, 0)$ $(q_0, 1) \rightarrow (q_3, 0)$ $(q_0, *) \rightarrow (q_y, *)$ $(q_1, 0) \rightarrow (q_0, \Rightarrow)$ $(q_1, 1) \rightarrow (q_0, \Rightarrow)$ $(q_2, 0) \rightarrow (q_3, 0)$ $(q_2, 1) \rightarrow (q_1, 0)$ $(q_2, *) \rightarrow (q_x, *)$ $(q_{3,0}) \rightarrow (q_2, \Rightarrow)$ $(q_{3,1}) \rightarrow (q_2, \Rightarrow)$

3.4. Példák

1. A bemenő szót tükröző Turing-gép: $w \rightarrow w^T$.
2. A bemenő szót megduplázó Turing-gép: $w \rightarrow ww$.
3. A bemenő szót, mint bináris számot eggyel megnövelő Turing-gép: $w \rightarrow w'$, ahol $w' = \overline{w} + 1$.

3.5. Feladatok

1. Adjunk meg egy Turing-gépet, amelyik a 01001 szót írja a szalagra!

2. Adjunk meg egy Turing-gépet, amelyik a $w = w_1 w_2 \dots w_n$ szót írja a szalagra!
3. Adjunk meg egy Turing-gépet, amelyik a bemenő szót invertálja, azaz minden 1-t 0-ra, minden 0-t 1-re cserél!
4. Adjunk meg egy Turing-gépet, amelyik a bemenő szóban páronként megcseréli a jeleket!

Pl.

$01001 \rightarrow 10001$

$011011 \rightarrow 100111$

$010101 \rightarrow 101010$

5. Adjunk meg egy Turing-gépet, amelyik a bemenő szóról eldönti, hogy melyik jelből található több! Ha 0-ból, akkor q_u , ha 1-ből, akkor q_v , illetve ha egyenlőek, akkor q_w - állapotban áll meg.

4. Church-Turing tézis

Ahogy azt a fejezet elején levő bevezetőben leírtuk, egy algoritmusra úgy tekintünk, mint egy függvényre, ami egy bemenő szóhoz (feladat) kiszámít egy kimenő szót (megoldás).

Az előző szakasz definíciói alapján a Turing-gépek is hasonló feladatot oldanak meg.

Felvetődik a kérdés, hogy melyek azok a feladatok, amelyek Turing-géppel megoldhatóak. Erre próbál választ adni a következő állítás.

4.13. Church-Turing tézis

Legyen A egy algoritmus, ami szavakat szavakká alakít, azaz $A: \Sigma^* \rightarrow \Sigma^*$.
Ekkor létezik egy $T: \Sigma^* \rightarrow \Sigma^*$ Turing-gép, amelyikre $A(w) = T(w) \forall w \in \Sigma^*$ esetén.

4.14. Megjegyzés

1. Amennyiben egy adott w bemenet esetén A nem áll meg, $A(w)$ nem definiált. Ekkor feltételezzük, hogy T sem áll meg az adott szón.
2. A Church-Turing tézis mind a transzformációs, mind a döntési feladatra értelmezhető. Ez utóbbi esetben az algoritmus kimenete értelemszerűen nem egy szó, hanem egy döntés. (pl. igen-nem)

A Church-Turing tézis érdekkessége, hogy a benne szereplő egyik fogalomnak (algoritmus) nincs matematikai definíciója. Ennek következménye, hogy az állításnak nem létezhet matematikai bizonyítása. Igazságérteke viszont van. Mit kezdhetünk akkor vele? Szokatlan módon, igazolni ugyan elvileg is lehetetlen, amennyiben azonban hamis az állítás, azt lehet bizonyítani. Ehhez elegendő találni egy olyan feladatot, amit valamelyen algoritmusnak elfogadható módszerrel meg tudunk oldani, Turing-géppel viszont bizonyíthatóan nem.

A tézis első megfogalmazása az 1930-as évekre tehető. Azóta cáfolni nem sikerült, annak ellenére, hogy jó párán próbálkoztak vele. Tipikus módszer erre az, hogy kifejlesztenek egy újabb algoritmusmodellt, majd megpróbálják belátni, hogy van olyan feladat, amit ennek segítségével meg lehet oldani, Turing-géppel viszont nem. A mai napig azonban minden modellről kiderült, hogy nem erősebb a Turing-gép modellnél. Ha a megoldható feladatok osztálya az adott modell esetén ugyanaz, mint a Turing-gépekkel megoldható feladatok osztálya, a kérdéses algoritmusleírást Turing-ekvivalensnek nevezik. A későbbiek során mi is megismerünk néhány újabb modellt (alapvetően a Turing-gép általánosításairól lesz szó), és be is látjuk Turing-ekvivalenciájukat. Nem véletlen az sem, hogy számtalan Turing-gép definíció létezik. Ekvivalenciájuk miatt elhanyagolható a köztük levő különbség, ezért több-kevesebb szabadságot engedhetünk meg, megtartva az eredeti ötletet: minden (vagy csak egy) irányban végtelen (vagy végtelenül nyújtható) szalag, véges állapotú regiszter.

A legismertebb Turing-ekvivalens algoritmusmodellek:

- λ kalkulus
- RAM gép
- Markov algoritmus
- Rekurzív függvények
- Sejtautomaták
- Neurális háló modell
- Genetikus algoritmus

5. Turing-gépek összefűzése

Egy konkrét feladat megoldása Turing-gép segítségével a legtöbb esetben nem nyilvánvaló. Megadható azonban néhány művelet a Turing-gépeken, amelyek segítségével Turing-gépek felhasználásával új Turing-gépeket hozhatunk létre. Ezen műveletek hatásának megismérésével egyszerűbbé válik a modell használata, a bizonyítások megértése. Segítségével a jóval hétköznapibb eljáráselvű programozáshoz hasonló megoldásokat és gondolkodásmenetet követhetjük, megtartva a matematikai precizitás lehetőségét.

4.15. Definíció (Turing-gépek összefűzése [kompozíciója])

Legyen $T_1 = (\mathcal{Q}_1, \Sigma_1, s_1, \delta_1, H_1)$ és $T_2 = (\mathcal{Q}_2, \Sigma_2, s_2, \delta_2, H_2)$ két Turing-gép.

Tegyük fel, hogy $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$.

A $T = T_1 \cdot T_2$ (vagy $T = T_1 \rightarrow T_2$) Turing-gépet a következőképpen definiáljuk:

$T = (\mathcal{Q}, \Sigma, s, \delta, H)$, ahol $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \{s_t\}$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $s = s_1$,

$H = H_2$ és $\delta = \delta_1 \cup \delta_2 \cup \delta'$.

Itt

$\delta' : (H_1 \cup \{s_t\}) \times (\Sigma \cup \{\star\}) \rightarrow (s_t, s_2) \times (\Sigma \cup \{\star\})$

ahol

$\delta'(h, a) = (s_t, a)$, minden $h \in H_1$ és $a \in \Sigma$ esetén, valamint

$\delta'(s_t, a) = (s_t, \Rightarrow)$ minden $a \in \Sigma$ esetén és $\delta'(s_t, \star) = (s_2, \Rightarrow)$.

4.16. Megjegyzés

A δ' függvények uniója itt a 2. fejezetben értelmezett módon relációk uniójaként tekinthető.

4.17. Tétel

Az előbbi definícióban adott T Turing-gépre $\forall w \in \Sigma^* T(w) = T_2(T_1(w))$,

azaz a T Turing-gép lényegében a T_1 és T_2 Turing-gépek egymás utáni "futtatásával" kapható meg.

Bizonyítás

Legyen $w \in \Sigma^*$ egy tetszőleges l hosszúságú szó, T_1 számítása a w bemeneten $K_0^{(1)}, K_1^{(1)}, K_2^{(1)}, \dots$,

T számítása a w bemeneten K_0, K_1, K_2, \dots . Ha T_1 megáll a w bemeneten, legyen

$w' = T_1(w)$ és T_2 számítása a w' bemeneten $K_0^{(2)}, K_1^{(2)}, K_2^{(2)}, \dots$.

A definíció alapján, mivel $s = s_1$, így $K_0 = (s, \lambda, w[1], w[2 \dots l]) = (s_1, \lambda, w[1], w[2 \dots l]) = K_0^{(1)}$.

Ugyancsak a definíció szerint, ha $q \in Q_1$, akkor $\delta(q, a) = \delta_1(q, a)$, azaz ameddig $q \notin H_1$, addig

$$K_n = K_n^{(1)}.$$

Ez alapján, ha T_1 nem áll meg w -n, akkor az T sem áll meg.

Ha $K_v^{(1)} = (h, w_1, a, w_2)$, ahol $h \in H_1$, akkor a T_1 Turing-gép megáll. Ebben az esetben

$w' = w_1 \cdot a \cdot w_2$, azaz a szalag tartalma w' . T definíciója alapján ilyenkor továbbszámol, a δ' -nek

megfelelően, azaz először átmegy a $K = (s_t, w_1, a, w_2) \rightarrow K = (s_t, w_1, a, w_2)$ konfigurációba, majd a következő lépések során

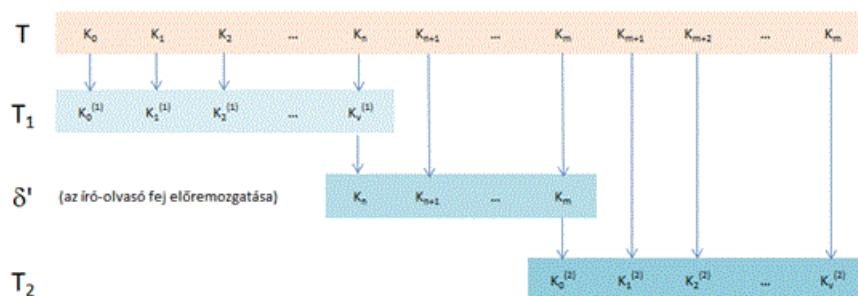
az író olvasó fejet a szalag elejére viszi. Ha azt eléri, átmegy a $K_m = (s_2, \lambda, w'[1], w'[2..l])$ konfigurációba

- itt $k = l(w')$ -, ami viszont megegyezik a $K_0^{(2)}$ konfigurációval. Mivel $\delta(q, a) = \delta_2(q, a)$, ha $q \in Q_2$,

ezért innentől kezdve $K_{m+1} = K_m^{(2)}$, azaz ha T_2 nem áll meg w' -n, akkor T sem fog megállni, egyébként

pedig $T(w) = T_2(w) = T_2(T_1(w))$.

Figure 4.36. T_1 és T_2 kompozíciója



✓

Felismerő Turing-gépek segítségével a feltételes végrehajtás (feltételes eljáráshívás) struktúráját is modellezni tudjuk.

4.18. Definíció (feltételes elágazás)

Legyen $T_1 = (Q_1, \Sigma_1, s_1, \delta_1, H_1)$, $T_2 = (Q_2, \Sigma_2, s_2, \delta_2, H_2)$ és $T_3 = (Q_3, \Sigma_3, s_3, \delta_3, H_3)$ három Turing-gép.

Tegyük fel, hogy $Q_1 \cap Q_2 = \emptyset$, $Q_1 \cap Q_3 = \emptyset$, $Q_2 \cap Q_3 = \emptyset$,

$$H_1 = H_{11} \cup H_{12}, \quad H_{11} \cap H_{12} = \emptyset$$

A $T = T_1?(H_{11}:T_2)|(H_{12}:T_3)$ Turing-gépet a következőképpen definiáljuk:

$T = (Q, \Sigma, \delta, H)$, ahol $Q = Q_1 \cup Q_2 \cup Q_3 \cup (s_{t_1}, s_{t_2})$,

$$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3, \quad s = s_1,$$

$$H = H_2 \cup H_3 \quad \text{és} \quad \delta = \delta_1 \cup \delta_2 \cup \delta_3 \cup \delta'$$

$$\text{Itt } \delta' : (H_1 \cup \{s_{t_1}, s_{t_2}\}) \times (\Sigma \cup \{\#\}) \rightarrow \{s_{t_1}, s_{t_2}, s_2, s_3\} \times (\Sigma \cup \{\#\})$$

, ahol

$$\delta'(h, a) = (s_{t_1}, a), \quad \text{ minden } h \in H_{11} \quad \text{és} \quad a \in \Sigma \quad \text{esetén,}$$

$\delta'(h, a) = (s_t, a)$, minden $h \in H_{12}$ és $a \in \Sigma$ esetén,
 $\delta'(s_t, a) = (s_t, \Leftarrow)$ minden $a \in \Sigma$ és $i = 1, 2$ esetén valamint
 $\delta'(s_t, *) = (s_{i+1}, \Rightarrow)$, ahol $i = 1, 2$.

4.19. Megjegyzés

Többszörös elágazás is definiálható, ha H_1 -et nem két, hanem több részre particionáljuk, és mindegyiknek megfeleltetünk egy új Turing-gépet.

4.20. Tétel

Legyen $w \in \Sigma^*$ és $T_1(w) = w'$ és a T_1 válaszát jelöljük
 V_1 -gyel, ha a w bemeneten H_{11} -beli állapotban, illetve
 V_2 -vel, ha a w bemeneten H_{12} -beli állapotban áll meg.
Az előbbi definícióban adott T Turing-gépre
 $T(w) = T_2(w)$, ha T_1 válasza a w bemeneten V_1 , illetve
 $T(w) = T_1(w)$, ha T_1 válasza a w bemeneten V_2 .

Bizonyítás

Legyen $w \in \Sigma^*$ egy tetszőleges l hosszúságú szó, T_1 számítása a w bemeneten $K_0^{(1)}, K_1^{(1)}, K_2^{(1)}, \dots$,

T számítása a w bemeneten K_0, K_1, K_2, \dots . Ha T_1 megáll a w bemeneten, legyen

$w' = T_1(w)$ és T_2 számítása a w' bemeneten $K_0^{(2)}, K_1^{(2)}, K_2^{(2)}, \dots$,

míg T_3 számítása a w' bemeneten $K_0^{(3)}, K_1^{(3)}, K_2^{(3)}, \dots$.

A definíció alapján, mivel $s = s_1$, így $K_0 = (s, \lambda, w[1], w[2 \dots l]) = (s_1, \lambda, w[1], w[2 \dots l]) = K_0^{(1)}$.

Ugyancsak a definíció szerint, ha $q \in Q_1$, akkor $\delta(q, a) = \delta_1(q, a)$, azaz ameddig $q \notin H_1$, addig

$K_n = K_n^{(1)}$.

Ez alapján, ha T_1 nem áll meg w -n, akkor az T sem áll meg.

Ha $K_y^{(1)} = (h, w_1, a, w_2)$, ahol $h \in H_1$, akkor a T_1 Turing-gép megáll. Ebben az esetben

$w' = w_1 \cdot a \cdot w_2$, azaz a szalag tartalma w' . T definíciója alapján ilyenkor továbbszámol, a δ' -nek megfelelően.

a.) Ha $h \in H_{11}$, akkor először átmegy a $K = (s_t, w_1, a, w_2)$ konfigurációba, majd a következő lépések során

az író olvasó fejet a szalag elejére viszi. Ha azt eléri, átmegy a $K_m = (s_2, \lambda, w[1], w[2 \dots k])$ konfigurációba

- itt $k = l(w)$ -, ami viszont megegyezik a $K_0^{(2)}$ konfigurációval. Mivel $\delta(q, a) = \delta_2(q, a)$, ha $q \in Q_2$,

ezért innentől kezdve $K_{m+n} = K_n^{(2)}$, azaz ha T_2 nem áll meg w' -n, akkor T sem fog megállni, egyébként

pedig $T(w) = T_2(w)$

b.) Ha $h \in H_{1^n}$, akkor először átmegy a $K = (s_{t_1}, w_1, a, w_2)$ konfigurációba, majd a következő lépések során

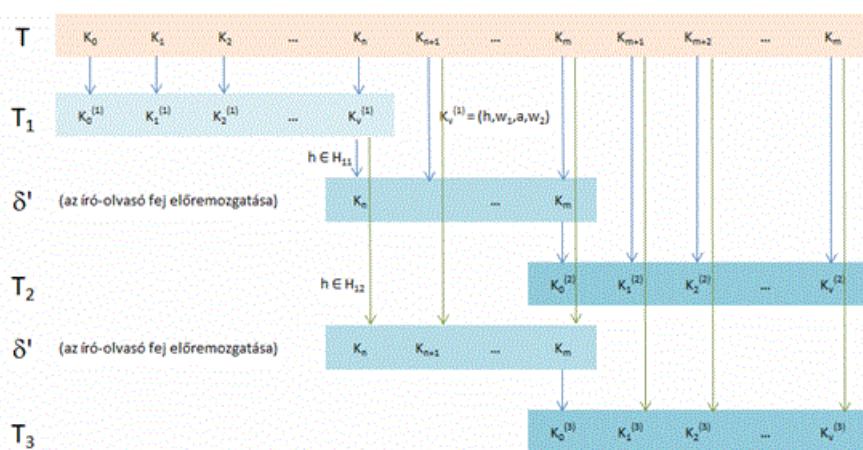
az író olvasó fejet a szalag elejére viszi. Ha azt eléri, átmegy a $K_m = (s_3, \lambda, w'[1], w'[2\dots k])$ konfigurációba

- itt $k = l(w')$ -, ami viszont megegyezik a $K_0^{(3)}$ konfigurációval. Mivel $\delta(q, a) = \delta_3(q, a)$, ha $q \in Q_3$

ezért innentől kezdve $K_{m+n} = K_n^{(3)}$, azaz ha T_2 nem áll meg w' -n, akkor T sem fog megállni, egyébként

pedig $T(w) = T_3(w)$.

Figure 4.37. T_1 , T_2 és T_3 feltételes kompozíciója



✓

Amennyiben egy Turing-gépet előre megadott számban többször egymás után szeretnénk végrehajtani, egy kicsit pontosítani kell a definíciót, mivel az egyszerű kompozíciónál feltettük, hogy az összefűzendő Turing-gépek állapothalmazai diszjunktak.

4.21. Definíció (rögzített ismétlésű ciklus)

Legyen $T = (Q, \Sigma, s, \delta, H)$ egy Turing-gép. A $T' = (Q', \Sigma, s', \delta', H')$ Turing-gépet definiáljuk úgy, hogy a T megfelelő komponenseit helyettesítjük egy megfelelő, $'$ -vel ellátott komponenssel. (Az átjelöljük az összetevőit.) Ezzel a jelöléssel legyen $T^2 = T \cdot T = T \cdot T'$. Általában:

Legyen $T^1 = T$ és

$T^n = T^{n-1} \cdot T$, ha $n > 1$.

4.22. Megjegyzés

Az egyszerű összefűzés definíciója utáni tétel alapján $T^2(w) = T(T(w))$. Iteratív módon alkalmazva $T^n(w) = T(T(\dots T(w)\dots))$, ahol az egymásba ágyazás mélysége n . (T^n jelölése pontosan megegyezik a függvénykompozícionál hagyományosan elfogadottal.)

A T^n -nel jelölt Turing-gép a T Turing-gép n -szer egymás után történő végrehajtásaként értelmezhető.

A programozásnál megszokott utasítások közül már csak a feltételes (végtelen) ciklus konstrukciója hiányzik.

4.23. Definíció (iteráció)

Legyen $T_1 = (\mathcal{Q}_1, \Sigma_1, s_1, \delta_1, H_1)$ és $H_1 = H_{11} \cup H_{12}$, $H_{11} \cap H_{12} = \emptyset$.

$T = (T_1 ? H_{11} : * | H_{12} : Stop)^*$ Turing-gépet a következőképpen definiáljuk:

$T = (\mathcal{Q}, \Sigma, s, \delta, H)$, ahol $\mathcal{Q} = \mathcal{Q}_1 \cup \{s_t\}$, $\Sigma = \Sigma_1$, $s = s_1$, $H = H_{11}$ és $\delta = \delta_1 \cup \delta'$.

Itt

$\delta' : (H_{11} \cup \{s_t\}) \times (\Sigma \cup \{\ast\}) \rightarrow \{s_1, s_t\} \times (\Sigma \cup \{\ast\})$

, ahol

$\delta'(h, a) = (s_t, a)$, minden $h \in H_{11}$, $a \in \Sigma$ esetén,

$\delta'(s_t, a) = (s_t, \leftarrow)$, $\forall a \in \Sigma$ és $\delta'(s_t, \ast) = (s_1, \ast)$.

4.24. Megjegyzés

Ha $H_{11} = H_1$, akkor T semmilyen bemeneten nem áll meg (= végtelen ciklus).

4.25. Tétel

$T = (T_1 ? H_{11} : * | H_{12} : Stop)^* = (T_1 ? H_{11} : (T_1 ? H_{11} : * | H_{12} : Stop)^* | H_{12} : Stop)^*$,

azaz T tekinthető a T_1 Turing-gép iteratív végrehajtásának.

Bizonyítás

Legyen $w \in \Sigma^*$ egy tetszőleges l hosszúságú szó, T_1 számítása a w bemeneten $K_0^{(1)}, K_1^{(1)}, K_2^{(1)}, \dots$,

T számítása a w bemeneten K_0, K_1, K_2, \dots . Ha T_1 megáll a w bemeneten, legyen

$w' = T_1(w)$ és T_2 számítása a w' bemeneten $K_0^{(2)}, K_1^{(2)}, K_2^{(2)}, \dots$

A definíció alapján, mivel $s = s_1$, így $K_0 = (s, \lambda, w_1, w_{2..l}) = (s_1, \lambda, w_1, w_{2..l}) = K_0^{(1)}$.

Ugyancsak a definíció szerint, ha $q \in \mathcal{Q}_1$, akkor $\delta(q, a) = \delta_1(q, a)$, azaz ameddig $q \notin H_1$, addig

$$K_n = K_n^{(1)}$$

Ez alapján, ha T_1 nem áll meg w -n, akkor az T sem áll meg.

Ha $K_v^{(1)} = (h, w_1, a, w_2)$, ahol $h \in H_1$, akkor a T_1 Turing-gép megáll. Ebben az esetben

$$w' = w_1 \cdot a \cdot w_2, \text{ azaz a szalag tartalma } w'.$$

T definíciója alapján, ha $h \in H_{11}$, akkor továbbszámol, a δ' -nek

megfelelően, ha $h \in H_{12}$, akkor megáll.

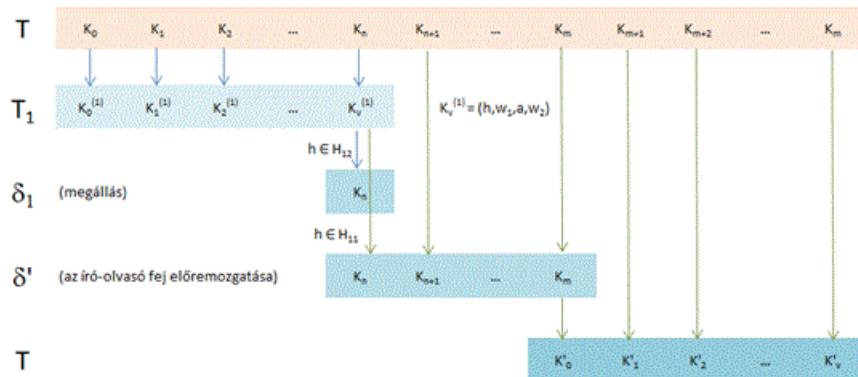
Ha tehát $h \in H_{11}$, akkor először átmegy a $K = (s_t, w_1, a, w_2)$ konfigurációba, majd a következő lépések során

az író olvasó fejet a szalag elejére viszi. Ha azt eléri, átmegy a $K_m = (s_1, \lambda, w'[1], w'[2..k])$ konfigurációba

- itt $k = l(w')$ -, ami viszont T_1 kezdőkonfigurációja a w' bemenőszóval.

Innen től kezdve $K_{m+1} = K'_1$, azaz pontosan úgy viselkedik, mintha -t a w' bemeneten indítottuk volna..

Figure 4.38. T_1 iterációja



✓

4.26. Megjegyzés

A különböző típusú összefűzések definíciójából elhagyhatjuk az író-olvasó fej előremozgatását. Ebben az esetben a következő Turing-gép végrehajtása a szalag azon pozíciójából folytatódik, ahol az előző abbahagyta a számolást. A legtöbb esetben ez a változat könnyebben használható, így viszont a függvények kompozíciójaként való értelmezés nem lenne helytálló.

Ezen összefűzések segítségével, ahogy a definíciók előtt is említettük, egyszerűbb Turing-gépekből komplikáltabb és komplikáltabb gépeket építhetünk, úgy tekintve az építőelemekre, mintha eljárások lennének.

Ilyen egyszerűbb gépek (a teljesség igénye nélkül) a következők lehetnek:

- 1 cellát jobbra vagy balra lép
- 1 cellát megvizsgál, a válasz "igen" vagy "nem"
- a szalag első vagy utolsó értékes jelére állítja az író-olvasó fejet
- egy rögzített jelet a szalagra ír
- 1 cella tartalmát eggyel balra (jobbra) másolja
- ...

Összetett Turing-gépek

- a bemenő szó végére vagy elejére ír egy jelet
- a bemenő szó végéről vagy elejéről töröl egy jelet
- a bemenő szót megduplázza
- a bemenő szót tükrözi
- a bemenő szóról eldönti, hogy páros hosszságú-e

- két bemenő szó hosszúságát összehasonlítja
- számrendszerök között átvált
- két szót összead

Példa:

Legyen a lehetséges bemenő szavak halmaza $\{0,1\}^*$ és legyenek adottak a következő egyszerű Turing-gépek:

1. $T_{\#?}$: megvizsgálja az író-olvasó fej alatti jelet; ha az $\#$, akkor "igen", ellenkező esetben "nem" választ ad.
2. $T_{*?}$: megvizsgálja az író-olvasó fej alatti jelet; ha az $*$, akkor "igen", ellenkező esetben "nem" választ ad.
3. $T_1?$: megvizsgálja az író-olvasó fej alatti jelet; ha az 1 , akkor "igen", ellenkező esetben "nem" választ ad.
4. T_{\leftarrow} : egy cellát balra lép
5. T_{\rightarrow} : egy cellát jobbra lép
6. T_0 : 0 -t ír a szalagra
7. T_1 : 1 -et ír a szalagra
8. $T_{\#}$: $\#$ -et ír a szalagra

Konstruáljuk meg belőlük az összefűzési műveletek segítségével azt a Turing-gépet, amelyik a bemenő w szóból elkészíti a ww szót!

Először készítsük el azt a Turing-gépet, amelyik a szalagon levő szó (jobboldali) végére áll:

$$T_{\Rightarrow} = T_{\#?} : ((igen \rightarrow Stop) | (nem \rightarrow T_{\rightarrow}))^*$$

Hasonlóan adható meg a szó elejére álló is:

$$T_{\Leftarrow} = T_{\#?} : ((igen \rightarrow Stop) | (nem \rightarrow T_{\leftarrow}))^*$$

Az a Turing-gép, amelyik jobbra haladva megkeresi az első $\#$ jelet, a következőképpen adható meg:

$$T_{\Rightarrow\#} = T_{\#?} : ((igen \rightarrow Stop) | (nem \rightarrow T_{\rightarrow}))^*$$

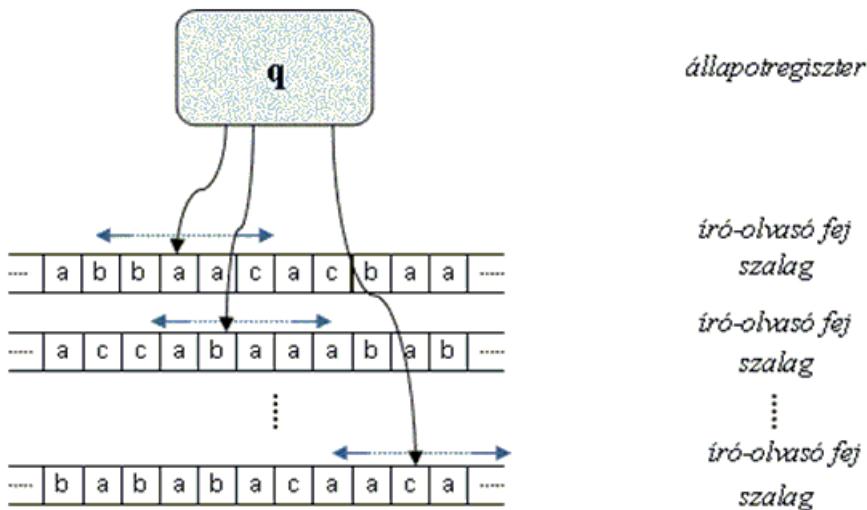
Az a Turing-gép, amelyik balra haladva megkeresi az első $\#$ jelet, a következőképpen adható meg:

$$T_{\Leftarrow\#} = T_{\#?} : ((igen \leftarrow Stop) | (nem \rightarrow T_{\rightarrow}))^*$$

6. Többszalagos Turing-gépek, szimuláció

Az eddig megismert Turing-gép modell általánosításaként bevezetjük a többszalagos Turing-gépek fogalmát. Itt a Turing-gép állapotregisztere egynél több szalaggal is kapcsolatban állhat, azokhoz külön író-olvasó fejjel rendelkezik, melyeket egymástól függetlenül mozgathat.

Figure 4.39. Többszalagos Turing-gép



4.27. Definíció (többszalagos Turing-gép)

Legyen $k \geq 1$, egész szám. A $T = (k, Q, \Sigma, s, \delta, H)$ k -os k -szalagos Turing-gépnek nevezzük, ha

- Q véges, nem üres halmaz; állapotok halmaza
- Σ véges, nem üres halmaz, $* \notin \Sigma$; szalag ábécé
- $s \in Q$, kezdő állapot
- $H \subseteq Q$, nem üres; végállapotok halmaza
- $\delta: Q \times ((\Sigma \cup \{*\}) \setminus H)^k \rightarrow Q \times (\Sigma \cup \{*\} \cup \{\leftarrow, \rightarrow\})^k$, átmenetfüggvény

Hasonlóan az egyszerű (egy szalagos) Turing-géphez, definiálhatjuk a konfiguráció, konfigurációátmenet és számítás fogalmát.

4.28. Definíció

Legyen $T = (k, Q, \Sigma, s, \delta, H)$ egy k -szalagos Turing-gép és $\Gamma = (\Sigma^* \times \Sigma \times \Sigma^*) \setminus (\Sigma^* \times \{*\} \times \Sigma^*)$.

$Q \times \Gamma^k$ -t a T lehetséges konfigurációi halmazának, elemeit T lehetséges konfigurációknak nevezzük.

Ekkor Γ a T egy szalagjának lehetséges leírásait tartalmazó halmaz.

4.29. Definíció

Legyen $T = (k, Q, \Sigma, s, \delta, H)$ egy k -szalagos Turing-gép és $K, K' \in Q \times \Gamma^k$ két konfigurációja. Tegyük fel, hogy

$$K = (q, \gamma_1, \dots, \gamma_k) \quad K' = (q', \gamma'_1, \dots, \gamma'_k)$$

ahol $q, q' \in Q$, $\gamma_1, \dots, \gamma_k, \gamma'_1, \dots, \gamma'_k \in \Gamma$,

$$\gamma_i = (w_{i1}, a_i, w_{ik}),$$

$$\gamma'_i = (w'_{i1}, a'_i, w'_{ik}) \quad \forall i \in \{1, \dots, k\}$$

és

$$\delta(q, a_1, \dots, a_k) = (q', b_1, \dots, b_k, m_1, \dots, m_k), \quad \text{ahol } q \notin H.$$

Azt mondjuk, hogy T egy lépésben (vagy közvetlenül) átmegy K -ból a K' konfigurációba (jelekben $K \vdash K'$), ha a minden $i \in \{1, \dots, k\}$ esetén a következők közül pontosan egy igaz:

- 1) $w'_{ii} = w_{ii}$, $w'_{ki} = w_{ki}$ és
 $a'_i = b_i$, ahol $b, a_i \in \Sigma$.

```

--- felülírási üzemmód
2)  $w'_i = w_i \cdot a$  ,
 $w_{i_1} = a'_{i_1} \cdot w'_{i_2}$  és
 $b_i = \Rightarrow$ 
--- jobbra lépési üzemmód
3)  $w_i = w'_{i_1} \cdot a'_{i_1}$  ,
 $w'_{i_2} = a_i \cdot w_{i_2}$  és
 $b_i = \Leftarrow$ 
--- balra lépési üzemmód

```

4.30. Megjegyzés

Többszalagos Turing-gép esetén egy lépés során szalagonként függetlenül - bár az átmenet függvény által közösen meghatározott módon - végzünk műveleteket.

4.31. Definíció

Legyen T egy k -szalagos Turing-gép.

T egy **számítása** az w hosszúságú w bemeneten konfigurációk egy $K_0, K_1, \dots, K_m, \dots$ sorozata amelyekre

1. $K_0 = (s, \gamma_1, \dots, \gamma_k)$, ahol $\gamma_1 = (\lambda, w[1], w[2 \dots n])$ és
 $\gamma_i = (\lambda, *, \lambda)$, ha $i = 2, \dots, k$;
2. $K_m \vdash K_{m+1}$, ha létezik K_m -ból közvetlenül elérhető konfiguráció (és ez az egyértelműség miatt K_{m+1});
3. $K_m = K_{m+1}$, ha nem létezik K_m -ból közvetlenül elérhető konfiguráció.

Ha $K_m = (h, \gamma_1, \dots, \gamma_k)$ olyan, hogy $h \in H$, akkor azt mondjuk, hogy a számítás véges, a Turing-gép a h végállapotban megáll. Ha ekkor $\gamma_1 = (w_1, a, w_2)$, akkor a $w = w_1 \cdot a \cdot w_2$ szót T **kimenetének** nevezzük.

Jelölése: $w' = T(w)$.

Ha egy $w \in \Sigma^*$ esetén a T Turing-gépnek végtelen számítása van, akkor nem értelmezünk kimenetet. Jelekben: $T(w) = \emptyset$. (Nem összetévesztendő a $T(w) = \lambda$ kimenettel.)

4.32. Megjegyzés

A többszalagos Turing-gép definíciójában feltételeztük, hogy a bemenetet az első szalagon adjuk meg és a kimenetet ugyanoda írjuk vissza. Ez pusztán megállapodás kérdése. Természetesen egyedi esetekben más - esetleg több - szalag is tartalmazhatja a bemenetet vagy a kimenetet.

Világos, hogy a frissen definiált többszalagos Turing-gép modell legalább olyan erős, mint az eredeti, egyszerű egyszalagos, hiszen minden egyszalagos Turing-gép kiegészíthető tetszőleges számú szalaggal, amelyeket menet közben nem használunk. Működés szempontjából teljesen azonosnak tekinthetők az egy- és belőle készített többszalagos Turing-gépek.

Ahhoz, hogy az egyes modellek erősségeit megvizsgáljuk, először is szükségünk lesz egy pontos definícióra, ami elég kifejezően leírja a modellek viszonyát. Ennek segítségével az új és régi modellt össze tudjuk hasonlítani.

4.33. Definíció

Legyen $T_1 \subseteq T_2$ et Turing-gépet. Azt mondjuk, hogy T_2 szimulálja T_1 -et, ha és esetén.

4.34. Megjegyzés

Az előző definíció alapján T_1 tetszőleges bemenete lehetséges bemenete T_2 -nek is.

Továbbá, $T_1(w) = T_2(w)$ azt jelenti, hogy ha T_1 megáll a w bemeneten és kimenetként egy w' szót ad, akkor T_2 is megáll és szintén w' -t ad válaszul. Ha T_1 nem áll meg a w bemeneten, akkor T_2 sem.

Elképzelhető, hogy T_2 sokkal több minden ki tud számolni, mint T_1 , más bemeneteket is kaphat, de azt mindenki által meg tudja csinálni, amit T_1 .

4.35. Tétel

Legyen $k \geq 2$, T egy k -szalagos Turing-gép. Ekkor létezik T -t szimuláló 1-szalagos T' Turing-gép.

Bizonyítás

A precíz bizonyítás meglehetősen sok technikai részlet kidolgozását igényelné, ezért csak a vázlatát ismertetjük.

Konstruktív módon látjuk be az állítás igazságát, ami azt jelenti, hogy nem egy általánosan létező Turing-gépről fogunk igazolni valamit, hanem megadjuk a konkrét felépítését is.

A szimuláció alapötlete az, hogy lépésről-lépéstre végrehajtjuk az eredeti Turing-gép adott inputon elvégzett számítását. Ehhez a szimuláló Turing-gép folyamatosan tárolja a szimulált Turing-gép aktuális konfigurációját. Általánosan megfogalmazható, hogy adattárolásra kétféle lehetőség adódik: az állapotot keresztül vagy a szalagon. Mivel az állapotok segítségével csak véges mennyiségi adat tárolható, a konfigurációk száma pedig végtelen, természetes módon adódik, hogy a szalagon tároljuk a szükséges információt. Mivel a szimuláló Turing-gépnek csak egyetlen szalagja van, így azon az egyetlen szalagon kényetlen tárolni a szimulált Turing-gép teljes szalagtartalmát. Ennek megoldására több lehetőség is adódik. Az egyik az lehet, hogy a szimulált T Turing-gép szalagjain levő szavakat a szimuláló T' Turing-gép szalagján egymás után, valamilyen határolójelekkel elválasztva tároljuk. Ekkor az jelent külön problémát, hogy a szavakhoz való hozzáírás vagy azokból való törlés esetén a többi szót folyamatosan mozgatni kell, hogy ne veszítünk el egyetlen értékes jelet sem, illetve, hogy az ábrázolt szavak ne távolodjanak el egymástól. Egy másik megközelítés - amit végül is követni fogunk a bizonyítás során -, hogy az egyetlen szalagot "sávokra" bontjuk úgy, mintha továbbra is meglennének a korábbi szalagok, csak az író-olvasó fejeik mozgása egymáshoz rögzített lenne. Ekkor (akkorcsak az előbbi modellnél) szükséges az eredeti Turing-gép író-olvasó fejének pozícióját tárolni. Ez utóbbi megoldást követve a szalag-abc jelei az eredeti szalag-abc jeleiből képzett k komponensű vektorokként képzelhetők el, kiegészítve az író-olvasó fejek helyének jelölésére szolgáló k komponensű $0,1$ vektorral.

Lássuk az előbb leírtaknak megfelelő T' definícióját.

Legyen $T' = (Q', \Sigma', s', \delta', H')$ a T -t szimuláló Turing-gép, ahol

$$Q' = Q \times (\Sigma \cup \{-1, 0, 1\})^k \times Q_{op}$$

$$\Sigma' = \Sigma^k \times \{0, 1\}^k \cup \{\#\}$$

$$s = (s, a_1, \dots, a_k, 0, \dots, 0, q_{start})$$

$$H' = \{(h, a_1, \dots, a_k, q_{stop}) \mid h \in H, a_i \in \Sigma \cup \{-1, 0, 1\} \forall i \in \{1, \dots, k\}\}$$

és

δ' az átmenet függvény, amit az alábbiakban írunk le.

A bizonyítás elején leírtaknak megfelelően T' szalagján fogjuk tárolni az eredeti Turing-gép szalagjairól származó összes információt. Eközben T' állapota tárolja többek között T' aktuális állapotát.

A T' egy $K' = (q', w_{11}, a_1, w_{12}, a_2, w_{21}, a_2, w_{22}, \dots, w_{k_1}, a_k, w_{k_2})$ konfigurációjának kiindulásképpen a T következő konfigurációját fogjuk megfeleltetni: $K = (\bar{q}, \lambda, \bar{a}, \bar{w})$.

K egyes összetevőit a következőképpen adhatjuk meg:

$$\bar{q} = (q', *, \dots, *, q_{\text{scan}}),$$

$$\bar{a} = (*, \dots, *, a_1, \dots, a_k), \text{ ahol } a_i = 1 \text{ ha } w_{k_i} = \lambda \text{ és } a_k = *, \text{ illetve } 0 \text{ egyébként,}$$

valamint

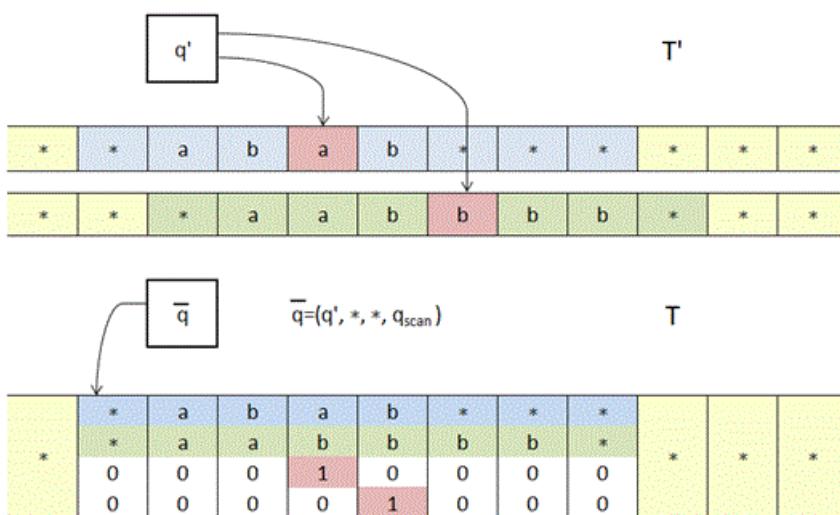
$\bar{w} = \bar{w}_1 \dots \bar{w}_l$, ahol $\bar{w}_i = \max_{i=1 \dots k} \{l(w_{k_i} \cdot a_i \cdot w_{k_i})\} + 1$ (azaz \bar{w} hossza eggyel hosszabb, mint a T' szalajain levő szavak leghosszabbikának hosszával),

$\bar{w}_i = (w_{1_i}, \dots, w_{k_i}, a_{j_i}, \dots, a_{j_i})$ minden $j \in \{1, \dots, l\}$ esetén, ahol $w_i = w_{k_i} \cdot a_i \cdot w_{k_i} \forall 1 \leq i \leq k$ (ha $l(w_i) < j$ akkor legyen $w_i = *$) és

$a_{j_i} = 1$, ha $l(w_{k_i} \cdot a_i) = j$ illetve 0 egyébként (azaz az író-olvasó fej az i . szalagon levő szó j . betűjén áll), minden $1 \leq i \leq k$ és $1 \leq j \leq l$ esetén.

Egy konkrét 2-szalagos Turing-gép esetén ez a következőképpen néz ki:

Figure 4.40. Egy kétszalagos Turing-gép szimulációja



A T Turing-gép egy ilyen kiindulási konfigurációból a következő számítási fázisokat hajta végre:

1. fázis: Beolvasás.

Végig lépked a szalag értékes celláin, és ahol az író-olvasó fej pozíciójában 1-t talál, a megfelelő szimbólumot beírja az állapotának megfelelő pozíójába.

Az előbbi példában a negyedik lépésig $\bar{q}_1 = (q', \alpha, *, 0, 0, q_{scn})$ értéket vesz fel. Az ötödik lépés után pedig megint változik a $\bar{q}_1 = (q'', b_1, \dots, b_k, q_{write,0})$ értékre.

Ha a szalag utolsó értékes celláját is megvizsgálta átlép a következő fázisba.

Tovább vizsgálva a példát, T' állapota már nem változik ebben a fázisban, mivel nincs több író-olvasó fej a szalagon.

2. fázis: $T' \$ T \$$ lépések véghajtása.

Ebben a fázisban összesen egy dolgot csinál: ha $\bar{q} = (q', \alpha_1, \dots, \alpha_k, q_{scn})$, akkor átmegy a $\bar{q}_1 = (q'', b_1, \dots, b_k, q_{write,0})$ állapotba, ahol $\delta'(q', \alpha_1, \dots, \alpha_k) = (q'', b_1, \dots, b_k)$

Itt lényegében kiszámolja és eltárolja az állapotában, hogy a szimulált Turing-gép milyen állapotváltozáson megy keresztül és milyen módon változtatja meg a szalagtartalmakat, vagy az író-olvasó fejek helyét.

Ezután átlép a harmadik fázisba.

3. fázis: Visszairás.

A megváltozott állapot tartalma alapján átállítja a szalagtartalmat. Visszafelé lépkedve megkeresi az író-olvasó fejek helyét jelölő 1-eket, és ha talál egyet, megnézi, hogy milyen műveletet kell véghajtania. Ha cserélni kell az eredeti Turing-gép szalagján levő szimbólumot, akkor a megfelelő a_i -t kicseréli b_i -re, ha mozgatni kell az író-olvasó fejet, akkor törli az 1 értéket a cella megfelelő komponenséből és a szomszédos cella megfelelő komponensébe írja át (jobb- vagy baloldali szomszéd, a b_i értékének megfelelően).

Ezek a lépések az összetettségük miatt nem hajthatók végre egyetlen állapot segítségével, ezért menet közben \bar{q} utolsó komponensét értelemszerűen változtatni kell. ($q_{write,1}, q_{write,2}, \dots, q_{write,m}$ -mel jelölhetjük a fázison belüli különböző műveleteket),

Ha elérte a szalag elejét, átlép a negyedik fázisba, amit $q_{nom,1} \$$ komponenssel jelölhetünk.

4. fázis: Kiigazítás.

Ez a fázis a legkomplikáltabb. Amennyiben T' valamelyik szalagjának elején hozzáír vagy töröl egy szimbólumot, akkor a T' szalagján elcsúsznak egymáshoz képest a szalagkezdetek. Ezt normalizálni kell, hogy ugyanolyan típusú konfigurációból indulhasson a következő lépés szimulálása, mint az előzőben.

Amennyiben a kiigazítással végzett, kétféleképpen folytathatja a számítást.

Ha a $\bar{q} = (q'', b_1, \dots, b_k, q_{nom,*})$ állapot $q'' \$ q'' \$$ komponense végállapot, akkor átmegy a $\bar{q}_1 = (q'', *, \dots, *, q_{swp})$ állapotba, ha q'' nem végállapot, akkor pedig a $\bar{q}_1 = (q'', *, \dots, *, q_{scn})$ állapotba. Ez utóbbi esetben a T' Turing-gép az 1. fázisnak megfelelő kiindulási konfigurációba kerül, és folytatja a számítást (az 1. fázis lépéseivel).

Amennyiben a T' Turing-gépet a T' kezdőkonfigurációjának megfelelő kiindulási konfigurációból indítjuk látható, hogy T' a 4.-1. fázisámenet során pontosan a T' számításának megfelelő konfigurációkon halad végig. Megállni pontosan akkor fog, ha az eredeti Turing-gép is megállt volna, a szalagtartalma pedig (megfelelő értelmezés mellett) pontosan a T' szalagtartalmának felel meg. Azaz T' valóban szimulálja T' -t. ✓

Feladat:

1. Írunk Turing-gépet az Ackermann-függvény kiszámítására!

Chapter 5. Kiszámíthatóság-elmélet

A kiszámíthatóságelmélet azzal a problémakörrel foglalkozik, hogy milyen módon lehetne a lehető legfontosabban megfogalmazni és körülírni az algoritmussal megoldható feladatokat. Megoldható-e egyáltalán minden feladat valamelyen algoritmussal? Van-e különbség megoldások között? Lehetséges-e, hogy egy feladatot csak részben lehet megoldani? Hogyan lehet ezt pontosítani, és mi a jelentése?

A vizsgálatokhoz szükségünk lesz néhány fogalomra. Az egyik ilyen az univerzális Turing-gép, a másik - pontosabban másik kettő - pedig a rekurzív - illetve rekurzív felsorolható - nyelvek fogalma.

1. Univerzális Turing-gép és az univerzális nyelv

A minden napokban használatos számítógépeket univerzálisnak nevezük azon tulajdonságuk miatt, hogy nem csak egy konkrét feladat megoldására használhatók, hanem bárhol korlátok között tetszőleges célra. Ez a képességük azon múlik, hogy alapvető működésük kiegészítéseként programokat rendelhetünk hozzájuk, futtathatunk rajtuk. A Turing-géppel kapcsolatos vizsgálataink egyik kiindulópontja az volt, hogy ez a modell a Church-Turing tézis szerint minden algoritmus leírására alkalmas. Ennek megfelelően felvetődik a kérdés, hogy ha minden leírhatunk vele, tekinthető az univerzális számítógéppel analógnak? Ennek vizsgálatához először is szükségünk van a tulajdonság pontos definíciójára.

5.1. Definíció

Legyen $k > 0$ egész. Egy T Turing-gépet univerzálisnak nevezünk a k -szalagos Turing-gépekre nézve, ha minden T^k -szalagos Turing-géphez létezik $P_T \in \Sigma^*$ szó, amire $T(P_T \# w) = T^k(w) \forall w \in \Sigma^*$ esetén. Itt $\# \notin \Sigma$ egy megfelelő elválasztójel.

Az előző definícióban leírt univerzális Turing-géptől tehát azt várjuk el, hogy az adott szalagszámhoz tartozó összes Turing-gépet legyen képes szimulálni egy megfelelő segédszó segítségével. A számítógépekkel való analógiát követve ez a P_T segédszó tekinthető a T Turing-gép programjának. Az egyszerűség kedvéért a későbbiekben ezt így is fogjuk nevezni.

A definícióhoz kapcsolódóan azonnal felvetődik a kérdés, hogy létezik-e egyáltalán ilyen Turing-gép valamelyen k -ra. A választ a következő téTEL adja meg.

5.2. Tétel

Minden $k > 0$ egész esetén létezik a k -szalagos Turing-gépekre nézve univerzális Turing-gép.

Bizonyítás

A bizonyítás konstruktív.

Rögzítsük k értékét. Hasonlóan a szimulációs téTEL bizonyításához, ha a bizonyítás minden lépését precízen szeretnénk végrehajtani, nagyon sok apró technikai részletre kellene odafigyelnünk, ezért újfent csak a bizonyítás vázlatát nézzük meg.

A konstrukció során egy olyan Turing-gépet hozunk létre, amelyik képes tárolni a szimulálandó Turing-gép programját és annak aktuális konfigurációját is. Ezt olyan módon fogjuk megoldani, hogy az univerzális Turing-gépünköt $k+2$ szalagosnak választjuk. Az első k szalagot úgy használjuk, mint ha azok a szimulált Turing-gép szalagjai lennének. A $k+1$. szalagon fogjuk tárolni a programot, a $k+2$ -en pedig a szimulált Turing-gép aktuális állapotát.

Az egyszerűség kedvéért tételezzük fel, hogy a szimulált Turing-gép szalag-abc-je is rögzített. Ez nem jelent különösebb megszorítást, hiszen jól ismert, hogy minden abc leképezhető a $\{0,1\}$ abc-re úgy hogy a különböző betűknek különböző azonos hosszúságú $0,1$ jelsorozatok felelnek meg. Ezáltal a leképezés egy egyértelműen dekódolható kódolást valósít meg, amivel a feladatok (nyelvek) a két abc között átalakíthatók.

Az univerzális Turing-gépet jelöljük $T = (k+2, Q, \Sigma, s, \delta, H)$ -val.

Ahhoz, hogy a működését megértsük, először is a szimulálandó Turing-gép programját kell meghatároznunk.

A program lényegében a Turing-gép megadását, egészen pontosan átmenetfüggvényének felsorolásos leírását jelenti. Amilyen komponensei vannak, azokat egy egységesített nyelven leírjuk.

Legyen a szimulálandó Turing-gép $T' = (k, Q', \Sigma, s', \delta', H')$,

Q' elemszáma M és feleltekük meg Q' elemeinek a $0, \dots, M-1$ számokat úgy, hogy s' -höz a 0 -t rendeljük. A számokat megfelelő módon ábrázolhatjuk a Σ abc-ben, viszont a bizonyítás során továbbra is a $q\$q\$$ szimbólumokat fogjuk használni az állapotok jelölésére.

Ekkor a T' programjának szerkezete a következő:

$P = h_1 \# h_2 \# \dots \# h_r \# \# \text{cmd}_1 \# \text{cmd}_2 \# \dots \# \text{cmd}_N$, ahol $H' = \{h_1, h_2, \dots, h_r\}$ és cmd_i a δ' átmenetfüggvény egy adott argumentumát és a felvett értékét tartalmazza. Egy ilyen egységet T' egy utasításának nevezünk. Pontosítva a leírást $\text{cmd}_i = q \# a_1 \# a_2 \# \dots \# a_k \# q' \# b_1 \# b_2 \# \dots \# b_k$ alakú, ha $\delta(q, a_1, a_2, \dots, a_k) = (q', b_1, b_2, \dots, b_k)$. A programban szükségszerűen az összes lehetséges argumentum esetére definiálnunk kell az utasításokat.

T kezdő konfigurációjaként beállítjuk a T' szalagtartalmát az első k szalagra, majd a $k+2$. szalagra 0 -t (azaz az s -nek megfelelő állapotot) írunk.

T működését a következő fázisokra bonthatjuk:

1. fázis: Megállási feltétel ellenőrzése.

Megvizsgáljuk, hogy szerepel-e a $k+2$. szalag tartalma a felsorolt végállapotok között. Ha igen, végállapotba állítjuk a T Turing-gépet. Ha nem, folytatjuk a 2. fázis végrehajtásával.

2. fázis: Állapotkeresés.

Ebben a fázisban, megkeressük a programszalagon a következő utasítást, amelyiknek argumentum részében szerepel a $k+2$. szalagon tárolt állapot.

Mivel a Turing-gép definíciójánál feltételeztük, hogy minden konfigurációból tovább tudunk lépni, ha nem végállapotban van, ezért minden fogunk ilyen utasítást találni.

3. fázis: Bemenet keresés.

Miután azonosítottuk, hogy a megfelelő állapothoz tartozó utasításnál járunk, leellenőrizzük, hogy az $1, \dots, k$ szalagokon az író-olvasó fej alatti szimbólumok megegyeznek-e az utasítás argumentumában szereplő megfelelő szimbólumokkal. Ha megegyeznek továbblépünk a 4. fázisba, ha nem, visszalépünk a 2-hoz.

4. fázis: Végrehajtás.

A megtalált végrehajtandó utasítás értékrészéből a benne szereplő q' állapotot a q helyére másoljuk a $k+2$. szalagon, majd az értékrész fennmaradó részének megfelelően az első k szalagon a szükséges módosításokat (szimbólumok cseréje illetve író-olvasó fejek mozgatása) végrehajtjuk.

5. fázis: Következő lépés előkészítése.

A $k+1$. és $k+2$. szalagon az író-olvasó fejet előre mozgatjuk, majd folytatjuk a végrehajtást az 1. fázissal.

Látható, hogy az 1. fázis első konfigurációi kölcsönösen egyértelműen megfelelhetők a szimulálandó T' Turing-gép számításának konfigurációinak. Ennek következtében a T Turing-gép pontosan a T' számításának megfelelően változtatja az első k és $k+2$. szalag tartalmát. Megállni pontosan akkor áll meg, amikor a T' is megállna, a kimenete pedig ekkor megegyezik T' kimenetével.

Azaz tényleg a téTELben megfogalmazottaknak megfelelő Turing-gépet konstruáltunk. ✓

5.3. Megjegyzés

Az előző fejezetben leírtak alapján minden k -szalagos Turing-gép szimulálható 1-szalagossal.

Az összes 1-szalagos Turing-gép szimulálható az 1-szalagos Turing-gépekhez tartozó univerzális Turing-géppel, amely szintén szimulálható egy 1-szalagossal.

Ezek után megállapíthatjuk, hogy van olyan 1 szalagos Turing-gép, amelyik minden Turing-gépet tud szimulálni. Ha nem említjük külön, univerzális Turing-gépnek ezt a Turing-gépet fogjuk nevezni, és T_u -val fogjuk jelölni.

5.4. Definíció

A T_u univerzális Turing-gép által felismert nyelvet univerzális nyelvnek fogjuk nevezni, és U -val jelöljük. Formálisan:

$$U = L(T_u)$$

Az univerzális nyelv a definíciója szerint azon szavakból áll, amelyeket az univerzális Turing-gép elfogad. Az univerzális Turing-gép viszont azon szavakat fogadja el, amelyek $P \# w$ alakúak, ahol P egy Turing-gép programja, w pedig egy olyan bemenet, amelyet a P -hez tartozó Turing-gép elfogad. Ezek alapján, visszatérve az eredeti kiindulási vizsgálatainkhoz, úgy tekinthetjük, hogy U az összes algoritmussal megoldható feladatot tartalmazza az összes megoldásával együtt. Érezhető, hogy egy meglehetősen összetett nyelvet definiáltunk univerzális nyelvként. Ráadásul a nyelv abból a szempontból extremális, hogy minden megoldható feladat-megoldás párt tartalmaz, így nem bővíthető tovább. Nem véletlen, hogy a későbbiekben alaposabban megvizsgálva, alapvető fontosságú eredményekhez jutunk általa.

2. A diagonális nyelv

A későbbiekben szükségünk lesz egy különleges nyelvre. Ez az úgynevezett diagonális nyelv.

5.5. Definíció

Legyen D azon szavak nyelve, amelyek Turing-gép programok, és a hozzájuk tartozó Turing-gép nem fogadja el őket. Formálisan:

$$D = \{w \mid \exists T \text{ Turing-gép}, w = P_T \text{ és } w \notin L(T)\}$$

Az így megadott nyelvet diagonális nyelvnek nevezzük. A diagonális nyelv definíciója meglehetősen különleges tulajdonságokat sugall. Itt a nyelv egy elemét egyszerre tekintjük feladatnak és algoritmusnak is. Ráadásul azzal a nyakatekert tulajdonsággal, hogy saját magát mint feladatot az algoritmus nem tudja megoldani.

3. Nyelvek rekurzivitása

5.6. Definíció

Az L nyelvet rekurzívnak nevezzük, ha $\exists T$ Turing-gép, amelyik minden bemeneten megáll és $L = L(T)$.

5.7. Definíció

Az L nyelvet rekurzív felsorolhatónak nevezzük, ha $\exists T$ Turing-gép, amelyikre $L = L(T)$.

A két definíció között látszólag apró különbség található: a második esetben nem követeljük meg a Turing-géptől, hogy minden esetben megálljon.

Valóban jelentős ez a különbség, vagy csak egyszerűen egy másik meghatározást adtunk ugyanarra a fogalomra? Annyi minden esetben világos, hogy ha egy nyelv megfelel az első definícióknak, azaz rekurzív, akkor természetes módon megfelel a másodiknak is, azaz rekurzív felsorolható.

5.8. Jelölés

$$\begin{aligned} R &= \{L \mid L \text{ rekurzív}\} \\ Rf &= \{L \mid L \text{ rekurzív felsorolható}\} \end{aligned}$$

Az előző észrevételt az előbb bevezetett jelöléssel a következőképpen írhatjuk le.

5.9. Megjegyzés

$$R \subseteq Rf$$

A felvetődött kérdést pedig így.

5.10. Kérdés

$$R = Rf ? \quad (\text{Más formában: } R \subseteq Rf, \text{ azaz valódi részhalmaza?})$$

A rekurzív nyelvek néhány tulajdonsága.

5.11. Tulajdonság

- Legyen $L_1, L_2 \in R$. Ekkor
1. $L_1 \cap L_2 \in R$
 2. $L_1 \cup L_2 \in R$
 3. $\bar{L}_1 \in R$ (másképpen: $R = coR$)
 4. $L_1 \cdot L_2 \in R$
 5. $L_1^* \in R$

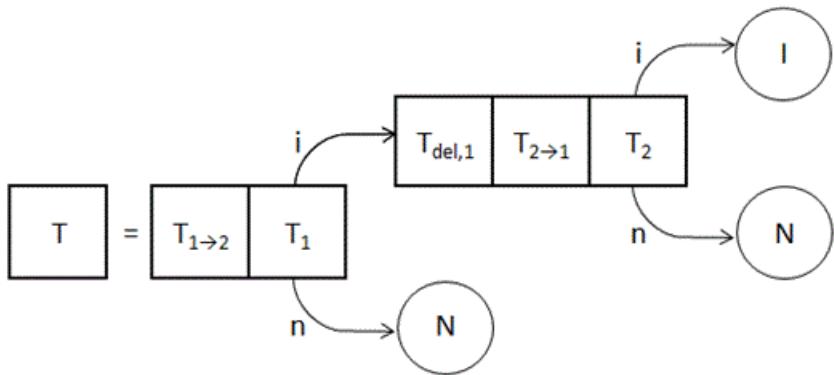
Bizonyítás

Definíció szerint ekkor léteznek T_1, T_2 Turing-gépek, amelyek minden bemeneten megállnak és $L_1 = L(T_1)$ illetve $L_2 = L(T_2)$. Az egyszerűség kedvéért tételezzük fel, hogy mind a kettő rendelkezik a megfelelő számú szalaggal, de csak az első szalagtól használja számításra.

A bizonyítás során a következő Turing-gépeket fogjuk felhasználni.

- a $T_{x \rightarrow y}$ Turing-gépek olyanok, amelyek a x . szalag tartalmát az y . szalagra másolják (ha van y -on értékes szimbólum, akkor az ott található szó végéhez csatolva)
- a $T_{del,x}$ Turing-gépek olyanok, amelyek az x . szalag tartalmát letörlik
- a $T_{y \neq t,x,y}$ olyan, amelyik az y . szalagon található szó első jelét az x . végére másolja, miközben letörli azt y -ről. Ha y üres (még a másolás előtt), akkor nem állapotban, egyébként igen állapotban áll meg.
- a $T_{x?x}$ megvizsgálja, hogy az x . szalag tartalma üres-e. Ha üres, igen állapotban, egyébként nem állapotban megáll.

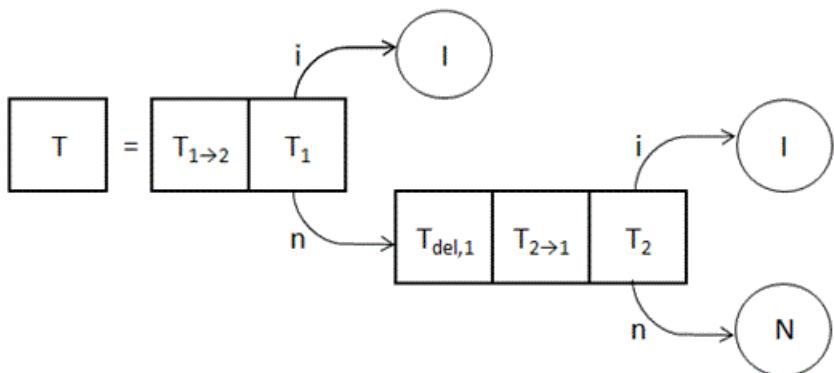
1. Legyen T a következő Turing-gép:



T akkor fogadja el a bemenetet, ha I állapotban áll meg és akkor utasítja el, ha N -ben. T_1 és T_2 hasonlóan az i irányban elfogadja, n irányban elutasítja a bemenetét. Mivel mind a két Turing-gép (T_1 és T_2) a T eredeti bemenetét kapja bemenetként, ezért T a w szónakor áll meg I állapotban, ha T_1 és T_2 is i állapotban áll meg w -n. Azaz T pontosan akkor fogadja el w -t, ha T_1 és T_2 is elfogadja. Megint másnéven megfogalmazva:

$w \in L(T)$ akkor és csak akkor, ha $w \in L(T_1)$ és $w \in L(T_2)$, vagyis $L(T) = L_1 \cap L_2$. Mivel T minden bemeneten megáll, ezért $L(T) \in R$.

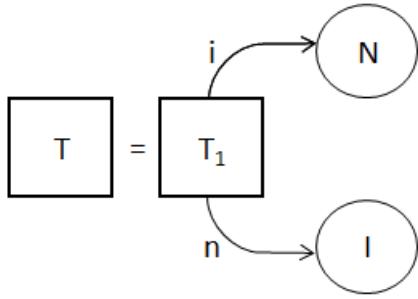
2. Legyen T a következő Turing-gép:



Az 1. pont bizonyításához hasonlóan T akkor fogadja el a bemenetet, ha I állapotban áll meg és akkor utasítja el, ha N -ben. T_1 és T_2 továbbra is az i irányban elfogadja, n irányban elutasítja a bemenetét. Mivel mind a két Turing-gép (T_1 és T_2) a T eredeti bemenetét kapja bemenetként, ezért T a w szónakor áll meg I állapotban, ha T_1 vagy T_2 i állapotban áll meg w -n. Azaz T pontosan akkor fogadja el w -t, ha T_1 vagy T_2 elfogadja. Megint másnéven megfogalmazva:

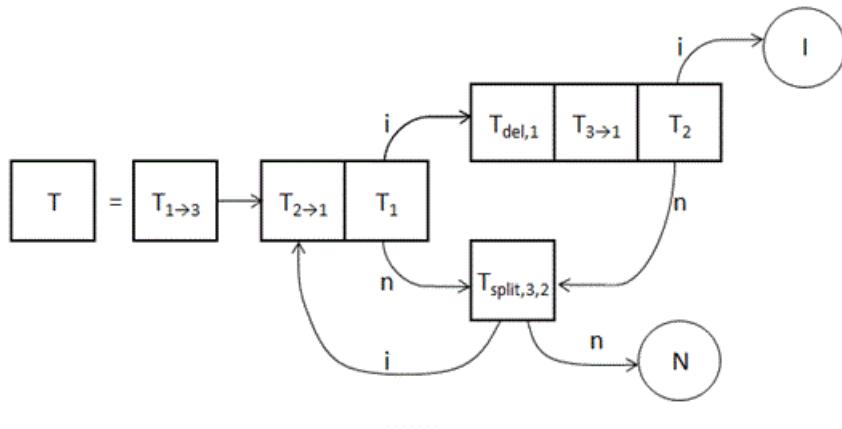
$w \in L(T)$ akkor és csak akkor, ha $w \in L(T_1)$ vagy $w \in L(T_2)$, vagyis $L(T) = L_1 \cup L_2$. Mivel T minden bemeneten megáll, ezért $L(T) \in R$.

3. Legyen T a következő Turing-gép:



T pontosan akkor fogadja el a bemenetét, amikor T_1 elutasítja, vagyis $L(T) = \bar{L}_1$. Mivel T minden bemeneten megáll, ezért $\bar{L}_1 \in R$.

4. Legyen T a következő Turing-gép:



A Turing-gép az alábbiak szerint működik:

1. Átmásolja a bemenő w szót a 3 . szalagra.
2. Leteszeli, hogy a 2 . szalagon levő szó benne van-e L_1 -ben, illetve a 3 . szalagon levő szó L_2 -ben. Ha mindkettőre igen a válasz, elfogadja a bemenetet és megáll. Ha valamelyik nemleges, folytatja a számolást.
3. Megvizsgálja, hogy üres-e a 3 . szalag. Ha igen ($T_{\text{empt},3,2} = \#$), akkor nemleges válasszal megáll. Ha nem üres, folytatja a számolást.
4. A 3 . szalag első szimbólumát átmásolja a 2 . szalagon levő szó végére és folytatja a számolást a 2. lépéssel.

T tulajdonképpen a bemenő szót felbontja az összes lehetséges módon két szó szorzatára (~ összefűzésére) és leellenőrzi, hogy az első rész benne van-e L_1 -ben, a második pedig L_2 -ben. Az első sikeres felbontásnál megáll és elfogadja a bemenetet. Ha nem talál ilyet, nemleges válasszal megáll.

A leírtak alapján $L(T) = L_1 \cdot L_2$. Mivel T minden bemeneten megáll, $L_1 \cdot L_2 \in R$.

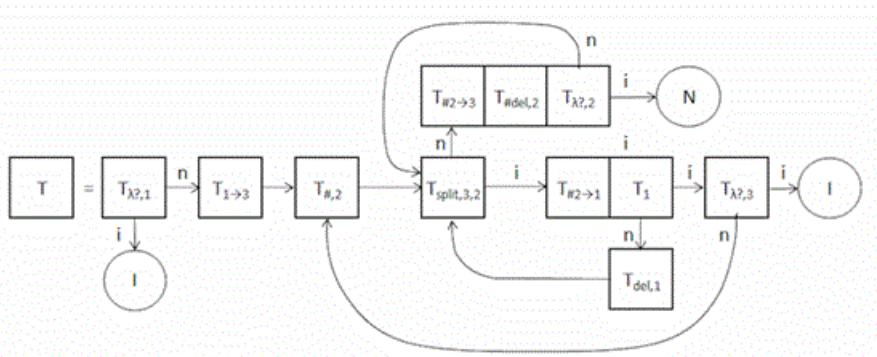
5. Az állítás igazolásához néhány további Turing-gépre lesz szükségünk.

$T_{\#,x}$ elhelyez az x . szalag végén egy $\#$ jelet. Ha a szalag üres volt, akkor csak a $\#$ jel lesz rajta.

$T_{\# \rightarrow \lambda^2}$ az x . szalag végétől indulva az első talált λ jelig átmásolja a ~~az adaltsimbólumokat~~ az y . szalag elejére. Ha üres volt, csak a másolt szó lesz rajta. Ha például tartalma λ , tartalma pedig λ , akkor λ -ra az λ szó kerül.

$T_{\# \leftarrow \lambda^2, x}$ az x . szalag végétől visszatéről az első megtalált $\#$ jelig. (A $\#$ szimbólumot is törli.) Ha például x tartalma $\#10110\#111\#1010$, akkor működése után x tartalma $\#10110\#111$ lesz.

Legyen T a következő Turing-gép:



A Turing-gép működése:

1. Ha λ a bemenet, elfogadja és megáll, egyébként a 3 . szalagra másolja.
2. Egy $\#$ jelet (elválasztó jel) helyez el a 2 . szalagon.
3. A 3 . szalag egyre hosszabb kezdőszeletét másolja a 2 . szalagra, egészen addig, amíg L_1 -be nem tartozik.
4. Ha talált egy kezdőszeletet, amelyik L_1 -be tartozik, folytatja a 2. lépéssel, azaz a következő szeletét vizsgálja meg, hogy L_1 -be tartozik-e. Ha nem talált megfelelő kezdőszeletet, az 5. lépéssel folytatja.
5. Visszamásolja a 2 . szalag végéről a 3 . szalag elejére az utoljára kiválasztott szeletet, majd az azt megelőző szelet növelésével folytatja a 3. lépéssel. Ha visszatörleszt után nem marad semmi a 2 . szalagon, akkor nem lehetett megfelelően felbontani a bemenő szót L_1 -beli szavak összefűzésére. Ekkor megáll és elutasítja a bemenetet.

T alaposabb vizsgálatával belátható, hogy pontosan az L_1^* nyelvet ismeri fel. Mivel minden bemeneten megáll, ezért $L_1^* \in R$.

✓

Hasonló tulajdonságok megfogalmazhatók rekurzív felsorolható nyelvekre is.

5.12. Tétel

Legyen $L_1, L_2 \in R_f$. Ekkor

1. $L_1 \cap L_2 \in R_f$
2. $L_1 \cup L_2 \in R_f$
3. $L_1 \cdot L_2 \in R_f$
4. $L_1^* \in R_f$

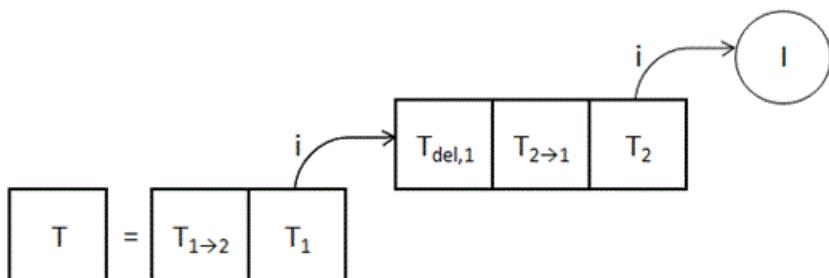
Bizonyítás

Hasonlóan az előző bizonyításhoz megállapíthatjuk, hogy definíció szerint léteznek T_1, T_2 Turing-gépek, amelyek nem feltétlenül állnak meg minden bemeneten és illetve . Az egyszerűség kedvéért továbbra is feltételezzük, hogy mind a kettő rendelkezik a megfelelő számú szalaggal, de csak az első szalagját használja számításra.

A bizonyítás során a következő Turing-gépeket fogjuk felhasználni.

- a $T_{x \rightarrow y}$ Turing-gépek olyanok, amelyek a x . szalag tartalmát az y . szalagra másolják (ha van y -on értékes szimbólum, akkor az ott található szó végéhez csatolva)
- a $T_{del,x}$ Turing-gépek olyanok, amelyek az x . szalag tartalmát letörlik
- a $T_{\#st,x,y}$ olyan, amelyik az y . szalagon található szó első jelét az x . végére másolja, miközben letörli azt y -ról. Ha y üres (még a másolás előtt), akkor nem állapotban, egyébként igen állapotban áll meg.
- a $T_{\lambda?x}$ megvizsgálja, hogy az x . szalag tartalma üres-e. Ha üres, igen állapotban, egyébként nem állapotban megáll.

1. Legyen T a következő Turing-gép:



Hasonlóan a rekurzív esethez, T akkor fogadja el a bemenetet, ha I állapotban áll meg, azaz T_1 és T_2 is elfogadja. Mivel mind a két Turing-gép (T_1 és T_2) a T eredeti bemenetét kapja bemenetként, ezért T a w szónakor áll meg I állapotban, ha T_1 és T_2 is i állapotban áll meg w -n. Azaz T pontosan akkor fogadja el w -t, ha T_1 és T_2 is elfogadja. Megint másképp megfogalmazva:

$w \in L(T)$ akkor és csak akkor, ha $w \in L(T_1)$ és $w \in L(T_2)$, vagyis $L(T) = L_1 \cap L_2$. Így $L(T) = L_1 \cap L_2 \in R_f$. Mivel megállást nem tudunk bizonyítani, így a $L(T) = L_1 \cap L_2 \in R$ állítás sem igazolható.

2. Ez az eset már nem olyan egyszerű, mint a metszetre vonatkozó bizonyításnál, hiszen az állítás rekurzív megfelelőjénél a Turing-gépeket a nem elfogadó ágon kapcsoltuk össze. Mivel azonban csak rekurzív felsorolhatóságot tudunk L_1 -ről és L_2 -ről, így ez a út nem járható.

A probléma úgy oldható fel, hogy alkalmazzuk a Turing-gépek párhuzamos összefűzését. Ennek definíciója a következő (egyszalagos Turing-gépekre adjuk meg, de teljesen hasonlóan lehet többszalagosokra is definiálni):

Legyen $T_1 = (Q_1, \Sigma_1, s_1, \delta_1, H_1)$ és $T_2 = (Q_2, \Sigma_2, s_2, \delta_2, H_2)$ két Turing-gép.

A $T = (2, Q, \Sigma, s, \delta, H)$ Turing-gépet a $T_1\$T_2$ párhuzamos összefűzésének nevezzük, ha

$$Q = Q_1 \times Q_2,$$

$$\Sigma = \Sigma_1 \cup \Sigma_2,$$

$$s = (s_1, s_2),$$

$\delta = \delta_1 \times \delta_2$, azaz $\delta((q_1, q_2), a_1, a_2) = ((q'_1, q'_2), b_1, b_2)$ pontosan akkor, ha $\delta_1(q_1, a_1) = (q'_1, b_1)$ és $\delta(q_2, a_2) = (q'_2, b_2)$, valamint

$H = (H_1 \times Q_2) \cup (Q_1 \times H_2)$, azaz T akkor áll meg, ha legalább az egyik komponense megáll.

Az így definiált 2-szalagos T Turing-gép az első szalagján pontosan azokat a műveleteket végzi, mint amit a T_1 , a második szalagján pedig pontosan azokat, mint amit a T_2 végzett volna.

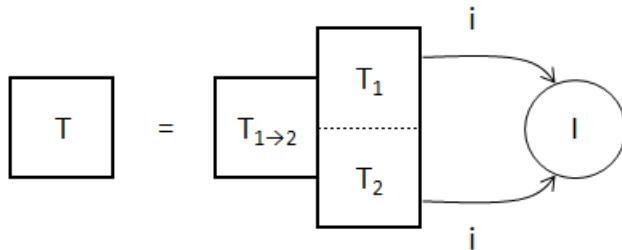
Azt, hogy T mikor fogadja el a bemenetet, a feladattól függően változtathatók. Esetünkben célszerű a következő definíció:

$$H^+ = (H_1^+ \times Q_2) \cup (Q_1 \times H_2^+), \text{ míg}$$

$$H^- = H \setminus H^+.$$

Vagyis T akkor fogadja el a bemenetet, ha legalább az egyik Turing-gép elfogadja.

A párhuzamos összefűzést a $T = T_1 \parallel T_2$ jelöléssel írhatjuk le.

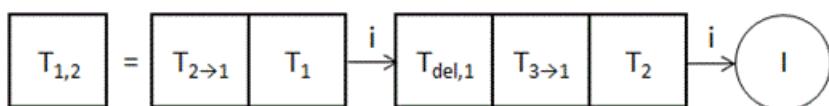


Az így megadott T Turing-gép pontosan akkor fogja elfogadni a bemenetet, ha T_1 és T_2 közül legalább az egyik elfogadja, vagyis $L(T) = L_1 \cup L_2$.

Mivel T nem feltétlenül áll meg minden bemeneten, ezért csak $L(T) \in Rf$ írható.

3. A bizonyításhoz felhasználjuk az előző tétel 4. pontjának bizonyításánál konstruált Turing-gép részleteit, és még néhány újabb Turing-gépet. Ezek a következők:

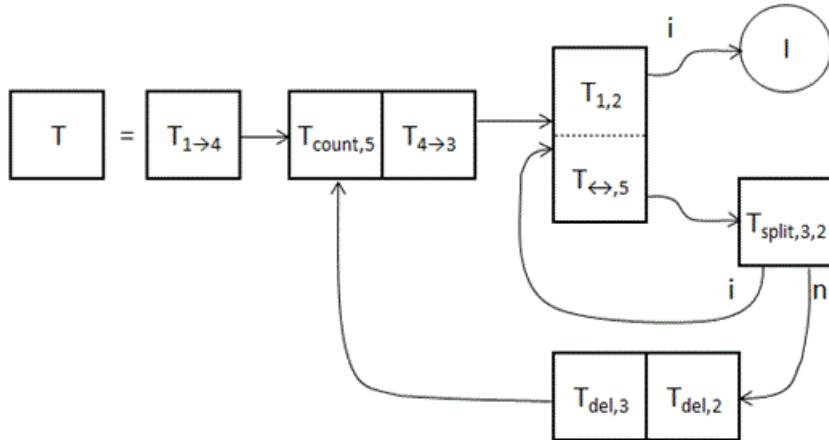
- $T_{1,2}$ egy olyan Turing-gép, amelyik leellenőrzi, hogy a 2. szalagján L_1 -beli, a 3. szalagján pedig L_2 -beli szó áll-e:



- $T_{\text{count},x}$ Turing-gép csak annyit csinál, hogy az x . szalagján található szó végére egy 1-t ír,

- $T_{\leftrightarrow,x}$ végig megy az x . szalagon található szón, és mikor eléri a szó végét, visszafordul és a szó elejére áll.

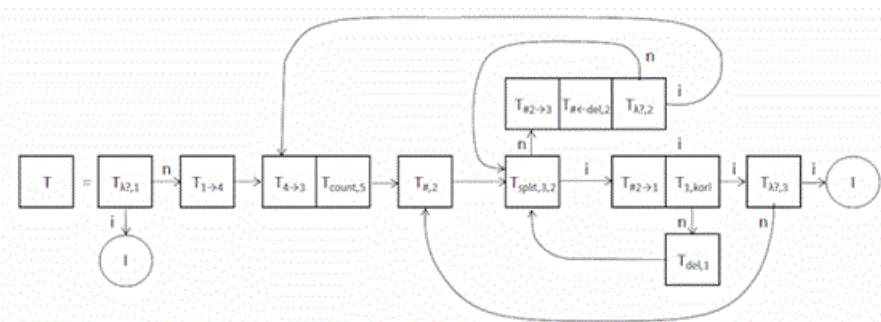
Ezekkel a definíciókkal legyen T a következő:



Ha alaposabban megfigyeljük, ez a Turing-gép annyiban tér el az előző tételeknél, hogy konstruálttól, hogy beépítettünk egy számlálót, ami nem engedi a $T_{1,2}$ Turing-gépet tetszőleges ideig futni, hanem csak addig, míg az 5. szalagon levő szón oda-vissza végigmegy. Ha közben azt találja, hogy a 2. szalagon L_1 -beli, a 3. szalagon pedig L_2 -beli szó áll, akkor elfogadó állapotban megáll. Ha a rendelkezésre álló idő alatt nem jutott pozitív eredményre, akkor tovább lép, és megpróbálja a következő felbontásra ugyanezt. Ha az összes felbontáson végighaladt, és még mindig nincs pozitív válasz, akkor kiüríti a 2. és 3. szalagot, eggyel megnöveli az 5. szalagon levő szó hosszát, és újrakezdi a vizsgálatot. Ezzel a beépített "időkorlátozó" Turing-géppel elérhetjük azt, hogy minden próbát csak korlátos ideig végezhet, így nem fordulhat elő, hogy az egyik vizsgálat során végelyen ciklusba keveredik, ami által nem tud átlépni egy másik vizsgálatba, ahol pozitív választ kaphatna. Egy teljes vizsgálati ciklus lefuttatása után viszont az, hogy nem kaptunk pozitív választ, még nem jelenti azt, hogy a bemenet nem eleme a vizsgált nyelvnek. Előfordulhat ugyanis, hogy a számolást az időkorlát miatt hamarabb abbahagyjuk, mint ahogy a pozitív választ megkaphatnánk. Azért, hogy ez a hiba ne léphessen fel, újraindítjuk a teljes ciklust, de hosszabb időkorláttal. Mindezt addig ismétljük, míg egyszer elfogadó állapotban meg nem áll a Turing-gép. Ha viszont a bemenő semelyik felbontása nem olyan, hogy az első fele L_1 -beli, a második fele pedig L_2 -beli, akkor természetesen a ciklus sohasem fog megállni.

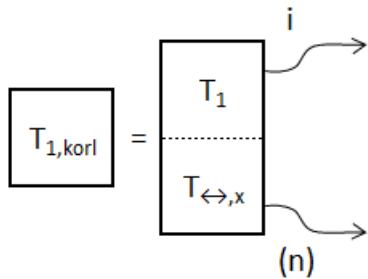
Látható, hogy a T Turing-gép által felismert nyelv pontosan a keresett $L_1 \cdot L_2$ nyelv, vagyis definíció szerint $L_1 \cdot L_2 \in R_f^*$.

4. Az állítás bizonyítása a 3. pontéhoz hasonlóan származtatható az előző tételeknél bizonyítására konstruált Turing-gépből. A módosítás a következőképpen néz ki:



A 3. pont bizonyításához hasonlóan, itt is bevezethetjük ugyanazt a korlátozó Turing-gépet. Most közvetlenül a T_1 Turing-géppel fűzzük össze párhuzamosan, aminek eredményeképpen a $T_{1,korl}$ Turing-gépet kapjuk. A nem

állapotban megállás most azt jelenti, hogy nem a T_1 elfogadó állapota állította meg a számítást, hanem a korlátozó Turing-gép.



Természetesen most sem mondhatjuk, hogy ha nem találtunk (korlátos idő alatt) megoldást, akkor nincs is, ezért a nem elfogadó állapotban való megállás helyett visszatérünk a Turing-gép elején levő részhez, megnöveljük a korlátot és újrakezdjük a számolást. Ezzel ugyancsak a \bar{L}_1 nyelvet felismerő Turing-gépet állítottuk elő, vagyis $\bar{L}_1 \in R_f$.

✓

5.13. Megjegyzés

A bizonyításban használt párhuzamos összefűzés, és az időben korlátozott végrehajtás általánosan is használható, sok esetben lehet hasznát venni.

5.14. Következmény

Legyen $L_1, L_2 \in coR_f$. Ekkor

1. $L_1 \cap L_2 \in coR_f$
2. $L_1 \cup L_2 \in coR_f$

Bizonyítás

1. Definíció szerint $L_1, L_2 \in coR_f$ pontosan azt jelenti, hogy $\bar{L}_1, \bar{L}_2 \in R_f$. Az előző tétel 2. pontja szerint ekkor $\overline{L_1 \cap L_2} = \bar{L}_1 \cup \bar{L}_2 \in R_f$, vagyis $L_1 \cap L_2 \in coR_f$.

2. Az előző ponthoz hasonlóan, $L_1, L_2 \in coR_f$ pontosan azt jelenti, hogy $\bar{L}_1, \bar{L}_2 \in R_f$, és az előző tétel 1. pontja szerint ekkor $\overline{L_1 \cup L_2} = \bar{L}_1 \cap \bar{L}_2 \in R_f$, vagyis $L_1 \cup L_2 \in coR_f$. ✓

5.15. Tétel

Legyen $L \in \mathcal{L}$.

Ekkor $L \in R$ akkor és csak akkor, ha $L \in R_f$ és $\bar{L} \in R_f$.

Másképpen: $R = R_f \cap coR_f$.

Bizonyítás

A téTEL két állításból tevődik össze:

1. Ha $L \in R$ akkor $L \in R_f$ és $L \in coR_f$.
2. Ha $L \in R_f$ és $L \in coR_f$, akkor $L \in R$.

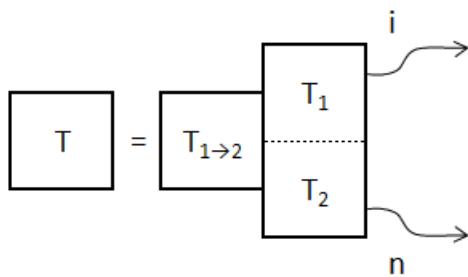
Nézzük először az egyszerűbbnek tűnő esetet:

1. A definíciók után megjegyzés alapján, ha $L \in R$ akkor $L \in R^f$. A rekurzív nyelvek tulajdonságairól szóló tétel alapján, ha $L \in R$, akkor $L^f \in R^f$, azaz $L^f = L$. Ezzel az állítást beláttuk.

A nehezebb (bár nem túl sokkal) állítás igazolása a következőképpen történhet:

2. Definíció szerint, ha $L \in R^f$ és $L \in coR^f$, akkor léteznek T_1 és T_2 Turing-gépek, amelyekre $L = L(T_1)$ és $L = L(T_2)$.

Legyenek a $T_{x \rightarrow y}$ Turing-gépek olyanok, amelyek a x . szalag tartalmát az y . szalagra másolják, és legyen a T Turing-gép a következő párhuzamos összefűzéssel definiált gép:



Mivel egy adott w bemenőszóra vagy az igaz, hogy eleme L -nek, vagy az, hogy nem, ezért a T Turing-gép minden bemenő szón megáll. Ha $w \in L$, akkor a T_1 Turing-gép, ha $w \notin L$, akkor a T_2 Turing-gép megállása miatt. A T által felismert nyelv $L = L(T)$, mivel pontosan azokat a szavakat fogja elfogadni, amelyeket amúgy T_1 elfogadott. Definíció szerint ez viszont azt jelenti, hogy $L \in R$. ✓

A téTEL egy nem túl meglepő dolgot állít, aminek az értelmezése visszatükröződik a bizonyításban is. Azt mondja ki ugyanis, hogy ha van egy algoritmusunk, amivel meg tudjuk állapítani, ha egy szó benne van egy nyelvben, és van egy másik, amivel meg tudjuk mondani, ha nincs benne a nyelvben, akkor van egy olyan egységes algoritmus is, amivel eldönthetjük, hogy benne van-e a nyelvben vagy nincs. Természetesen tűnik a gondolat, hogy a két algoritmusból gyűrünk egy újat, ahogy végül is a bizonyításban is történik.

Elérkeztünk oda, hogy a rekurzív és rekurzív felsorolható nyelvek osztályai között már látjuk az alapvető összefüggéseket, de még mindig nem tudjuk, hogy meg kell-e egyáltalán különböztetnünk őket. A következő két téTEL pontosan erre vonatkozó eredményt állapít meg, felhasználva a diagonális és az univerzális nyelvet.

5.16. Tétel

$$D \notin R^f$$

Bizonyítás

A bizonyítás indirekt. Tegyük fel, hogy $D \in R^f$. Definíció szerint ekkor létezik egy T Turing-gép, amelyikre $D = L(T)$. Legyen ennek a Turing-gépnek a programja P . Feltehetjük a kérdést, hogy vajon $P \in D$ vagy sem. Megpróbáljuk igazolni valamelyiket.

1. Tegyük fel, hogy $P \in D$. Ekkor

a. $D = L(T)$ miatt T elfogadja P -t, valamint

b. D definíciója szerint, ha $P \in D$ akkor olyan Turing-gép programja amely nem fogadja el saját magát mint bemenetet. Vagyis T nem fogadja el P -t.

Így viszont a. és b. ellentmond egymásnak, vagyis nem lehet $P \in D$.

2. Most tegyük fel, hogy $P \notin D$. Ekkor

a. $D = L(T)$ miatt T nem fogadja el P -t, valamint

b. D definíciója szerint, ha $P \notin D$ akkor vagy P nem egy Turing-gép programja, vagy olyan Turing-gép programja amely elfogadja saját magát mint bemenetet. Jelen esetben tudjuk, hogy P a T programja, tehát mindenképpen a második feltételnek tesz eleget, vagyis T elfogadja P -t.

Így viszont a. és b. megint csak ellentmond egymásnak, vagyis nem lehet, hogy $P \notin D$.

Mivel azonban $P \in D$ és $P \notin D$ közül pontosan az egyiknek teljesülni kell, ellentmondásra jutottunk. Ez az indirekt feltételezésünkönkből származik, vagyis $D \in R$ nem lehet igaz. ✓

5.17. Tétel

$$U \in Rf \setminus R$$

Bizonyítás

A téTEL két állításként fogalmazható meg:

1. $U \in Rf$ és

2. $U \notin R$.

Az egyszerűbbik állítás bizonyításával kezdjük:

1. Mivel $U = L(T_u)$, ez definíció szerint azt jelenti, hogy $U \in Rf$.

A második állítás bizonyítása indirekt módon történik:

2. Tegyük fel az állítás ellenetét, vagyis hogy $U \in R$. Ez definíció szerint azt jelenti, hogy létezik egy minden bemeneten megálló T Turing-gép, amelyikre $U = L(T)$.

Ennek segítségével előállítunk egy másik Turing-gépet, amihez a következő egyszerűbb Turing-gépeket használjuk fel:

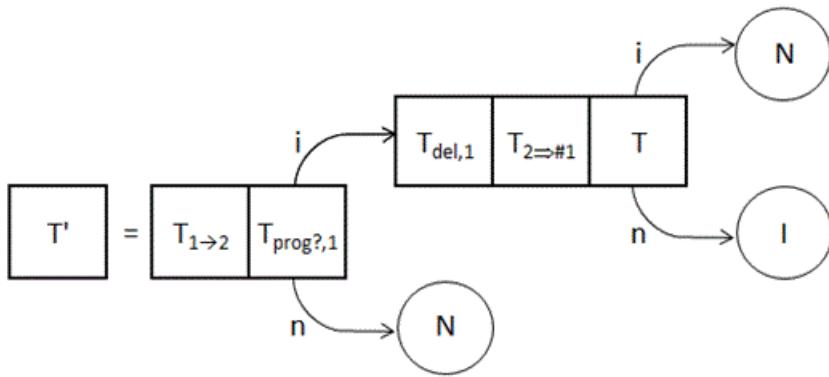
- $T_{prog?x}$ eldönti, hogy a bemenetére írt szó egy Turing-gép programja-e. Az univerzális Turing-gép létezéséről szóló téTEL bizonyítása során láthattuk, hogy egy Turing-gép program meglehetősen pontosan megfogalmazható formai feltételeknek felel meg, ezért nem nehéz átgondolni, hogy konstruálható egy olyan Turing-gép, amelyik minden bemeneten megáll és pontosan azokat a szavakat fogadja el, amelyek Turing-gép programok.

- $T_{x=\#y}$, ami az x . szalagon levő w szót kétszer az y . szalagra másolja, elválasztva öket egy $\#$ jellel, azaz w -hez $w\#\bar{w}$ -t rendel.

- $T_{\bar{x}\rightarrow y}$, az x . szalag tartalmát az y szalagra másolja.

- $T_{del,x}$ letörli az x . szalag tartalmát.

Definiáljuk most a T' Turing-gépet a következő módon:



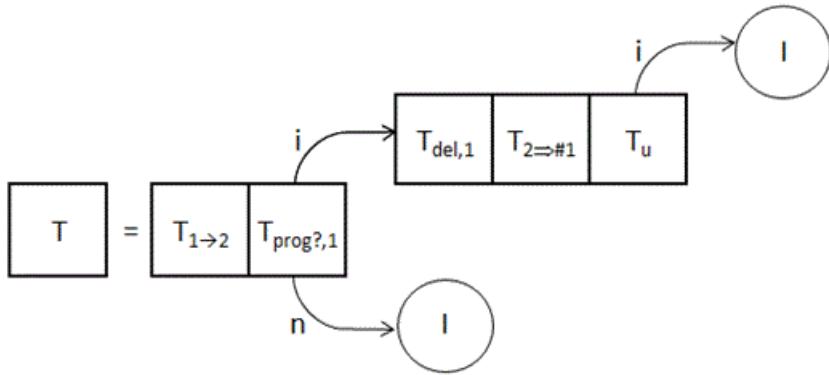
Világos, hogy T' minden bemeneten megáll, és pontosan azokat a w szavakat fogadja el, amelyek Turing-gép programok ($T_{\text{prog?}1} = i$) és a hozzájuk tartozó Turing-gép nem fogadja el őket ($w \# w \in \overline{L(T)} = \bar{U}$). Ez viszont pontosan a diagonális nyelvbe tartozó szavakat jelenti, amiből az következne, hogy $D \in R$. Az előző tételeben beláttuk, hogy $D \notin R_f$. Mivel $R \subseteq R_f$, a két állítás ellen mond egymásnak. Az ellentmondás az indirekt feltételezésünk következménye, vagyis $U \in R$ nem igaz. ✓

5.18. Következmény

1. $D \in coR_f$ és
2. $U \notin coR_f$.

Bizonyítás

1. Az előző tétel bizonyításában szereplő Turing-gépek felhasználásával definiáljuk a következő Turing-gépet:



T pontosan azokat a szavakat fogadja el, amelyek vagy nem Turing-gép programok, vagy ha azok, a hozzájuk tartozó Turing-gép elfogadja a saját programját. Másnéven $L(T) = \bar{D}$. Ez azt jelenti, hogy $\bar{D} \in R_f$, vagyis $D \in coR_f$.

2. Az előző tétel alapján $U \in R_f$. Ha $U \in coR_f$ lenne, akkor egy korábbi térel szerint $U \in R$ is teljesülne. Ezt viszont beláttuk, hogy nem igaz, tehát $U \notin coR_f$. ✓

Egyelőre tehát annyit láttunk be, hogy létezik rekurzív felsorolható, de nem rekurzív, valamint olyan nyelv, melynek komplementere rekurzív felsorolható, de nem rekurzív. (Igazából, ha \bar{L} megfelel az elsőként említett tulajdonságnak, akkor \bar{L} a másodiknak.) Ezzel viszont még igazoltuk azt, hogy van olyan nyelv, amelyik nem rekurzív felsorolható és a komplementere sem az. Erről szól a következő téTEL.

5.19. TéTEL

Létezik L nyelv, amelyikre $L \notin Rf \cup coRf$.

Bizonyítás

Minden Turing-géphez hozzárendelhetünk egy véges szót, a programját. Ezek szerint a Turing-gépek száma legfeljebb annyi lehet, mint a véges szavak száma, vagyis megszámlálhatóan végtelen. A rekurzív felsorolható nyelvek száma nem lehet több, mint a felismerő Turing-gépek száma, vagyis megszámlálhatóan végtelen.

Mivel egy adott abc fölötti szavak száma, azaz Σ^* számosága megszámlálhatóan végtelen, ezért a hozzá tartozó hatványhalmaz $2^{\Sigma^*} = \{L \mid L \subseteq \Sigma^*\}$ kontinuum számoságú. Mivel a kontinuum szigorúan nagyobb számoság mint a megszámlálható, ezért létezik olyan nyelv, amelyik nem rekurzív. ✓

A rekurzív felsorolható nyelvek nem véletlenül kapták ezt az elnevezést. Igaz ugyanis rájuk, hogy elemeik rekurzív módon (azaz Turing-géppel) felsorolhatók.

5.20. Tétel

Legyen $L \in \mathcal{L}$. Ekkor $L \in Rf$ akkor és csak akkor, ha $\exists T$ minden bemeneten megálló Turing-gép, amelyikre $L = \text{Range}(T)$.

Bizonyítás

Ha $L = \emptyset$, az állításban szereplő minden feltétel triviálisan teljesül, ezért az általánosság megszorítása nélkül feltételezhetjük, hogy L nem üres.

Legyen σ egy abc. Σ^* elemeit felsorolhatjuk hosszlexikografikus módon. Ha például $\Sigma = \{0,1\}$, akkor a felsorolás $(w_i)_{i=0}^\infty = [\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, 0001, \dots]$. A felsorolásban természetesen minden szó szerepel, és minden szóhoz egyszerű megállapítani, hogy mi a hosszlexikografikus rákövetkezője. (Lényegében az 1-el növelés kicsit módosított algoritmus.)

A téTEL KÉT ÁLLÍTÁSBÓL TEVŐDIK ÖSSZE:

1. Ha $L \in Rf$, akkor $\exists T$ minden bemeneten megálló Turing-gép, amelyikre $L = \text{Range}(T)$.
 2. Ha $\exists T$ minden bemeneten megálló Turing-gép, amelyikre $L = \text{Range}(T)$, akkor $L \in Rf$.
1. A bizonyítást konstruktív módon végezzük.

Feltéttük, hogy L nem üres, így van legalább egy eleme. Jelöljük w -vel L egy tetszőleges, rögzített elemét. Mivel $L \in Rf$, így a definíciónak megfelelően létezik T' Turing-gép, amelyikre $L = \text{Range}(T')$.

A következő Turing-gépeket fogjuk használni T előállításához:

- a $T_{x \rightarrow y}$ Turing-gép az x . szalag tartalmát az y . szalagra másolja (ha van y -on értékes szimbólum, akkor az ott található szó végéhez csatolva)

- a $T_{del,x}$ Turing-gép az x . szalag tartalmát letörli

- $T_{\leftrightarrow,x}$ végig megy az x . szalagon található szón, és mikor eléri a szó végét, visszafordul és a szó elejére áll.

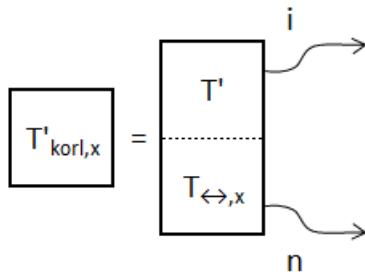
- $T_{w,x}$ az x . szalagra írja a w szót.

- $T_{\# \rightarrow y}$ az x . szalag végétől indulva az első talált $\#$ jelig átmásolja a talált szimbólumokat az y . szalag elejére. Ha y üres volt, csak a másolt szó lesz rajta.

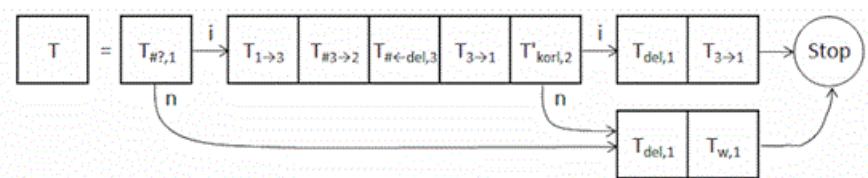
- $T_{\#\leftarrow del,x}$ az x . szalag végétől visszatéről az első megtalált $\#$ jelig. (A $\#$ szimbólumot is törli.)

- $T_{\#x}$ meggázolja, hogy az x . szalag hány darab x szimbólumot tartalmaz. Ha 1 darabot, i válasszal, egyébként válasszal áll meg.

A korábban tárgyaltakhoz hasonlóan párhuzamos összefűzéssel definiáljuk a T' időben korlátozott változatát:



Legyen T a következő Turing-gép:



Mivel T minden komponense megáll minden bemeneten, ezért T is megáll minden bemeneten.

T láthatóan nem felismerő Turing-gép. Mi lesz a kimenete a különböző bemeneteken? Ha a bemenet nem $w_1 \# w_2$ alakú, akkor mindenkorban a w szó lesz a kimenete. Ha a bemenet $w_1 \# w_2$ alakú, akkor kettébontja. A 3. és 1. szalagra w_1 -et, a 2. szalagra w_2 -t írja. Ezután a w_2 hossza által meghatározott ideig elvégzi a T' lépéseit w_1 -en. Ha a rendelkezésre álló idő alatt elfogadja w_1 -t, ha ennyi idő alatt nem tudta elfogadni w -t ír a kimenő szalagra.

Világos, hogy a kimenetek között csak L elemei lehetnek, hiszen mindenkorban vagy w , vagy egy T' által elfogadott szó kerül megállás előtt az első szalagra.

Továbbá, ha $w_1 \in L$, akkor T' elfogadja w_1 -t. Legyen az elfogadó számításának hossza τ és $w_2 = 1^x$ (azaz τ darab 1 sorozata). Mivel w_2 elég hosszú ahhoz, hogy w_1 -en $T'_{korl,1}$ elfogadó állapotban álljon meg, ezért $T(w_1 \# w_2) = w_1$. Ez azt jelenti, hogy minden L -beli szó képpé válik (és az előbbi bevezetésben leírtak szerint csak az L -beliek), amivel az állítást beláttuk.

2. Az előzőhez hasonlóan most is konstruktív bizonyítást adunk.

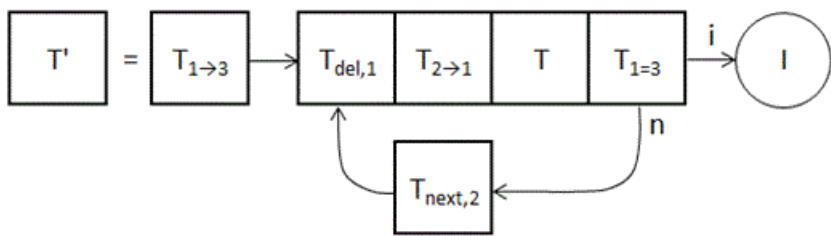
Legyen T egy olyan minden bemeneten megálló Turing-gép, amelyikre $L = \text{Range}(T)$.

A konstrukcióhoz az előbbi Turing-gépeken kívül még a következőkre lesz szükségünk:

- $T_{xent,x}$ az x . szalagon levő szót felülírja a hosszlexikografikus rákövetkezőjével.

- $T_{x=y}$ összehasonlíta az x . és y . szalag tartalmát. Ha megegyeznek i , egyébként n válasszal megáll.

Definiáljuk a keresett felismerő Turing-gépet az alábbi módon:



T' működésének magyarázata a következő:

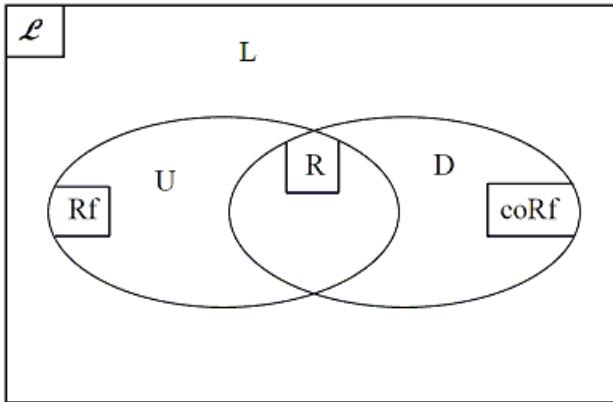
Az egyes szalagok tartalma:

1. Ezen számol, ide kerül a részeredmény
2. T aktuális bemenetét tárolja. Menet közben lexikografikusan növekvő sorrendben változik.
3. A bemenő szó másolata.

Legyen w az aktuális bemenő szó. T' működése során a lexikografikus sorrendben változó összes w' jelsorozathoz meghatározza $T(w')$ értékét. Ha van olyan w' , amire $T(w') = w$, akkor igen válasszal megáll, egyébként folytatja a w' növelését és újraellenőrzését.

Mivel minden $w \in L$ szóhoz létezik w_1 , amelyikre $T(w_1) = w$, ezért T' pontosan a $w \in L$ szavakat fogja elfogadni, azaz $L = L(T')$. Definíció szerint ez azt jelenti, hogy $L \in Rf$. ✓

A fejezet eredményei alapján a következő diagrammot rajzolhatjuk fel:



Az ábrán L egyike az előbb igazolt, nem pontosan definiált nyelveknek, amelyikre $L \notin Rf \cup coRf$.

Feladatok:

Legyen L_1 és L_2 két nem üres nyelv.

1. Igazoljuk, hogy ha $L_1 \in R$ és $L_1 \cdot L_2 \in R$ akkor $L_2 \in R$.

Hasonlóan, ha $L_2 \in R$ és $L_1 \cdot L_2 \in R$ akkor $L_1 \in R$.

2. Igazoljuk, hogy ha $L_1 \in Rf$ és $L_1 \cdot L_2 \in Rf$ akkor $L_2 \in Rf$.

Hasonlóan, ha $L_2 \in Rf$ és $L_1 \cdot L_2 \in Rf$ akkor $L_1 \in Rf$.

3. Igazoljuk, hogy ha $L_1 \in Rf$ és $L_2 \in coRf$ akkor $L_1 \cdot L_2 \notin Rf \cup coRf$.

4. Megállási probléma

Az előzőekben láttuk, hogy a feladatok túlnyomó többsége algoritmikusan nem megoldható, és általában annak eldöntése, hogy egy feladat algoritmikusan megoldható-e egyáltalán nem nyilvánvaló. A kérdéskört tovább vizsgálva az a természetes gondolat vetődik fel az emberben, hogy ha ennyire általános feladat nem is, de talán az megoldható, hogy egy konkrét T Turing-gépről és egy $w \in \Sigma^*$ szóról megállapítsuk, hogy a T elfogadja-e w -t vagy sem. Másnéven: el tudjuk dönteni, hogy egy adott algoritmus megold-e egy konkrét problémát vagy sem. Ennek vizsgálatához először is pontosan meg kell fogalmaznunk, a megoldandó feladatot.

5.21. Megállási probléma

Döntsük el a \tilde{T}_T programjával adott T Turing-gépről és $w \in \Sigma^*$ szóról, hogy T megáll-e a w bemeneten vagy sem.

A megállási problémának két alapvető megközelítése létezik: egy-egy konkrét esetre, vagy teljesen általánosan szeretnénk erre választ adni.

Az első esetben egyszerűbb a dolgunk, hiszen egyedi módszereket alkalmazhatunk minden alkalommal, speciálisan az adott problémához igazítva.

A második eset sokkal bonyolultabbnak tűnik, hiszen olyan bizonyítási eljárást kell találnunk, amelyik minden Turing-gépre és bemenetre működik. Mivel a bizonyítás lényegében logikai lépések sorozata, így lényegében egy egységes módszert - algoritmust - kell találnunk, ami eldönti az argumentumairól, hogy megfelelnek vagy sem. Ennek alapján a probléma általános megoldhatóságáról a következő állítást fogalmazhatjuk meg és bizonyíthatjuk be.

5.22. Tétel

Nem létezik olyan Turing-gép, amelyik minden \tilde{T}_T programjával adott T Turing-gépről és $w \in \Sigma^*$ szóról eldönti, hogy T megáll-e a w bemeneten vagy sem.

Bizonyítás

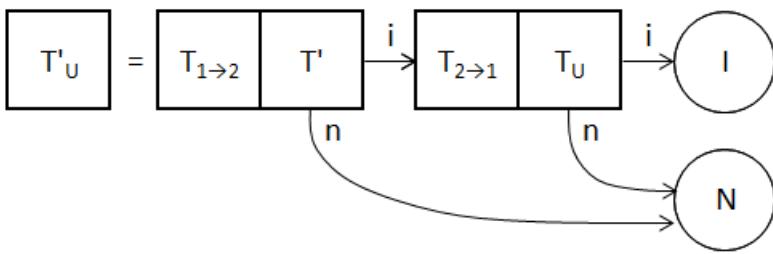
A tételet indirekt módon igazoljuk.

Tegyük fel, hogy létezik T' Turing-gép, amelyik minden \tilde{T}_T programjával adott T Turing-gépről és $w \in \Sigma^*$ szóról eldönti, hogy T megáll-e a w bemeneten vagy sem. Ha T megáll a w bemeneten i , egyébként n válasszal tér vissza. Az egyszerűség kedvéért azt is feltételezzük, hogy T' bemenetét $\tilde{T}_T \# w$ alakban adjuk meg.

A korábbi jelöléseknek megfelelően legyen

- T_u az univerzális Turing-gép,
- $T_{x \rightarrow y}$ egy Turing-gép, ami az x . szalag tartalmát az y . szalagra írja.

Definiáljuk T'_v -t a következőképpen:



T'_v pontosan azokat a w szavakat fogadja el, amelyeket T_v , azaz $L(T'_v) = U$, hiszen azokat a szavakat, amelyeken meg sem áll természetesen nem fogadja el T_v . Mivel T' már csak azokat a w -ket engedi át bemenetként T_v számára, amelyekről kiderítette, hogy meg fog állni rajta, ezért a hátralevő számítás mindenkorban befejeződik, azaz T'_v minden bemenő w szón megáll. Ez azt jelentené, hogy $U \in R$, amiről már beláttuk, hogy nem igaz. Az ellentmondás oka a hibás (indirekt) feltételezésünk. ✓

Az eldönthetőségi problémának megfogalmazható egy egyszerűbb változata is.

5.23. Megállási probléma 2

Döntsük el a $\overset{P}{T}$ programjával adott T Turing-gépről, hogy megáll-e az üres bemeneten vagy sem.

Bár lényegesen egyszerűbbnek tűnik, erre a feladatra is hasonló állítást lehet bizonyítani.

5.24. Tétel

Nem létezik olyan Turing-gép, amelyik minden $\overset{P}{T}$ programjával adott T Turing-gépről eldönti, hogy T megáll-e a λ bemeneten vagy sem.

Bizonyítás

A tételet most is indirekt módon igazoljuk.

Tegyük fel, hogy létezik T' Turing-gép, amelyik minden $\overset{P}{T}$ programjával adott T Turing-gépről eldönti, hogy T megáll-e a λ bemeneten vagy sem.

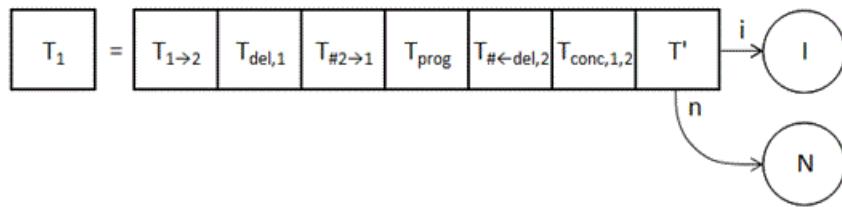
Legyen T_w egy olyan Turing-gép, amelyik a bemenő szalagjára felírja a $w \$$ szót. Nagyon egyszerű ilyen Turing-gépet létrehozni, hiszen csak $n = 2 \cdot l(w)$ darab állapot kell hozzá, például q_1, \dots, q_n , az átmenetfüggvénye pedig úgy definiálható, hogy $\delta(q_i, *) = (q_{i+1}, b_i)$ ahol $b_i = w \left[\frac{(i+1)}{2} \right]$, ha i páratlan és $b_i = \Rightarrow$, ha i páros.

Legyen $T_{\#x}$ egy olyan Turing-gép, ami a bemenő w szóhoz elkészíti T_w programját. A konstrukció során a további szokásos Turing-gépeket fogjuk használni:

- $T_{x \rightarrow y}$ egy Turing-gép, ami az x . szalag tartalmát az y . szalagra írja
- $T_{\#x \rightarrow y}$ az x . szalag végétől indulva az első talált $\#$ jelig átmásolja a talált szimbólumokat az y . szalagra
- $T_{\#\leftarrow del,x}$ az x . szalag végétől visszatöröl az első megtalált $\#$ jelig. (A $\#$ szimbólumot is törli.)

- $T_{conc,x,y}$ ha az x . és y . szalagon a T_x és T_y Turing-gép programok találhatók, előállítja az x . szalagra a $T_x \cdot T_y$ Turing-gép programját. Ez szintén nem bonyolult feladat, hiszen lényegében csak össze kell fűzni a két programot, illetve az első végállapotaiból a második kezdőállapotába való átmenetet kell definiálni.

Ezek után definiáljuk \tilde{T}_1 -t a következőképpen:



Világos, hogy \tilde{T}_1 minden bemeneten megáll.

Ha bemenetként egy $\tilde{T}_r^{\#w}$ alakú szót adunk meg, akkor \tilde{T}_1 a következőket csinálja:

1. átmásolja $\tilde{T}_r^{\#w}$ -t a 2. szalagra,
2. a 2. szalag # utáni részét, azaz w -t átmásolja az 1. szalagra
3. az első szalagon w -hez elkészíti a \tilde{T}_w Turing-gép programját.
4. a 2. szalagon eltávolítja a # utáni részt, azaz csak \tilde{T}_r marad
5. az 1. és 2. szalagon levő Turing-gép programokból elkészíti az 1. szalagra az összefűzött programot.
6. megvizsgálja, hogy az 1. szalagon levő Turing-gép megáll-e az üres bemeneten.

Az 5. lépésben létrehozott Turing-gép működése olyan, hogy ha bemenetként az üres szót kapja, akkor először felírja a szalajára a w szót, majd úgy működik, mint az eredeti T . Vagyis pontosan akkor áll meg az üres bemeneten, ha T megáll a w bemeneten.

Ez azt jelenti, hogy \tilde{T}_1 pontosan azokat a $\tilde{T}_r^{\#w}$ szavakat fogadja el, amelyekre T megáll w -n. Ekkor viszont \tilde{T}_1 megoldaná az általános megállási problémát, amiről az előző téTELben bizonyítottuk, hogy lehetetlen. Az ellentmondás oka az indirekt feltételezésünk.

✓

Chapter 6. Nemdeterminisztikus Turing-gépek

Az eddigiek során megismerkedtünk a Turing-gép alapvető modelljeivel. Ebben a fejezetben a modellek egy általánosításával a nemdeterminisztikus üzemelés bevezetésével foglalkozunk.

1. Nemdeterminisztikus Turing-gépek definíciója

Az egyszerűség kedvéért csak az \mathcal{A} szalagos nemdeterminisztikus Turing-gép definícióját adjuk meg, de természetesen a determinisztikus Turing-gépekhez hasonló módon a többszalagos modell is precízen leírható. A későbbiekben látni fogjuk, hogy nem feltétlenül szükséges, de amennyiben egyértelműen nemdeterminisztikus Turing-géppel dolgozunk, a jobb megkülönböztethetőség érdekében \mathcal{T} helyett $\hat{\mathcal{T}}$ jelölést fogunk használni.

6.1. Definíció

A $\hat{\mathcal{T}} = (\mathcal{Q}, \Sigma, s, \hat{\delta}, H)$ ötöst
nemdeterminisztikus Turing-gépnek nevezzük, ha
 \mathcal{Q} véges, nem üres halmaz; állapotok halmaza Σ véges, legalább 2 elemű halmaz, $* \in \Sigma$;
szalag ábécé $s \in \mathcal{Q}$; kezdő állapot $H \subseteq \mathcal{Q}$, nem üres; végállapotok halmaza
 $\hat{\delta}: (\mathcal{Q} \setminus H) \times \Sigma \rightarrow 2^{Q(\Sigma \cup \{*, =\})}$; átmenetfüggvény

Az előbbi definícióban használt jelölés a 2^A az A halmaz hatványhalmazát jelöli. A hatványhalmaz az adott halmaz részhalmazaiból álló halmaz:

$2^A = \{B | B \subseteq A\}$. A definíció értelmében a $\hat{\delta}$ függvény az argumentumaihoz nem egy konkrét értéket, hanem értékek egy halmazát rendeli.

Az eddigi Turing-gépeket, megkülönböztetésül az új nemdeterminisztikus modelltől, determinisztikus Turing-gépeknek fogjuk nevezni.

Ahhoz, hogy értelmezni tudjuk az új modellt, hasonló fogalmakat kell bevezetnünk, mint a determinisztikus Turing-gépek esetén.

A determinisztikus modellhez teljesen hasonló módon definiálhatjuk a nemdeterminisztikus Turing-gépek konfigurációját.

6.2. Definíció

A $\hat{\mathcal{T}}$ nemdeterminisztikus Turing-gép egy konfigurációja $K: K \in \mathcal{Q} \times ((\Sigma^* \times \Sigma \times \Sigma^*) \setminus (\Sigma^+ \times \{*\} \times \Sigma^+))$.

6.3. Megjegyzés

Itt is igaz a determinisztikus Turing-gépeknél megfigyelt szabály, miszerint egy $\hat{\mathcal{T}}$ nemdeterminisztikus Turing-gépnek csak akkor létezik $K = (q, w_1, *, w_2)$ alakú konfigurációja, ha $w_1 = \lambda$ vagy $w_2 = \lambda$. Észrevehetjük még, hogy a nemdeterminisztikus Turing-gépek konfigurációt semmi sem különbözteti meg a determinisztikus Turing-gépekétől.

Az igazán fontos és érdekes különbség a determinisztikus és nemdeterminisztikus Turing-gépek között a működésükben található.

6.4. Definíció

Azt mondjuk, hogy a \hat{T} nemdeterminisztikus Turing-gép **egy lépésben** (vagy következőként) átmehet -ból a konfigurációba (jelöléssel $(q, w_1, a, w_2) \xrightarrow{h} (q', w'_1, a', w'_2)$)

és a következők közül pontosan egy teljesül:

- 1) $a, a' \in \Sigma, q \in Q \setminus H, q' \in Q$
ahol , és .
 $w'_1 = w_1 \cdot a \cdot w_2 (q, \Rightarrow) \in \hat{\delta}(q, a)$, $a \in \Sigma, q \in Q \setminus H, q' \in Q$
- 2) , ahol , és .
 $w'_1 = w_1 \cdot a \cdot w_2 (q, \Leftarrow) \in \hat{\delta}(q, a)$, $a \in \Sigma, q \in Q \setminus H, q' \in Q$
- 3) , ahol , és .
--- balra lépési üzemmód

A definíció alapján azt mondhatjuk tehát, hogy a nemdeterminisztikus Turing-gépek nem átmennek egy konkrét, hanem átmehetnek egy lehetséges konfigurációba. Hogy még pontosabban megértsük a definíciók közötti eltérést nézzük meg hogyan változik a számítás fogalma.

6.5. Definíció

A \hat{T} nemdeterminisztikus Turing-gép egy **lehetséges számítása** konfigurációk egy $K_0, K_1, \dots, K_n, \dots$ sorozata ($i = 0, 1, \dots$) amelyekre

1. $K_0 = (s, \lambda, w_1, w_2, \dots)$, ahol $n = l(w)$;
2. $K_i \vdash K_{i+1}$, ha létezik K_i -ból közvetlenül elérhető konfiguráció
3. $K_i = K_{i+1}$, ha nem létezik K_i -ból közvetlenül elérhető konfiguráció.

A w szót a T **bemenetének** nevezzük.

A K_n konfigurációról azt mondjuk, hogy elérhető a K_0 konfigurációból, ha a Turing-gépnek van $K_0, K_1, \dots, K_n, \dots$ számítása.

Ha $K_n = (h, w_1, a, w_2)$ elérhető K_0 -ból, és $h \in H$, akkor azt mondjuk, hogy a számítás K_n -hez tartozó ága véges, a Turing-gép ezen az ágon a h végállapotban megáll. Ekkor a $w' = w_1 \cdot a \cdot w_2$ szót \hat{T} **egy kimenetének** nevezzük.

A nemdeterminisztikus Turing-gépek kimenete a következőképpen megadott halmaz:

$$\hat{T}(w) = \{w' \mid \text{ahol } \exists K = (h, w_1, a, w_2), K_0\text{-ból elérhető konfiguráció, és } w' = w_1 \cdot a \cdot w_2\}$$

A helyes értelmezéshez hozzá tartozik, hogy a lehetséges konfigurációváltások között nincs kiemelt, nincs valószínűsége az egyes lépéseknek (mint sztochasztikus automaták esetén), hanem úgy tekintendő, mintha a lehetséges irányok mindenekben folytatódna a számítás. (A Turing-gép az egyes konfigurációk után továbblépésnél "osztódik".)

Példák

1. Legyen $\hat{T} = (Q, \Sigma, s, \hat{\delta}, H)$, ahol $Q = \{s, h\}$, $H = \{h\}$, $\Sigma = \{0, 1\}$ és

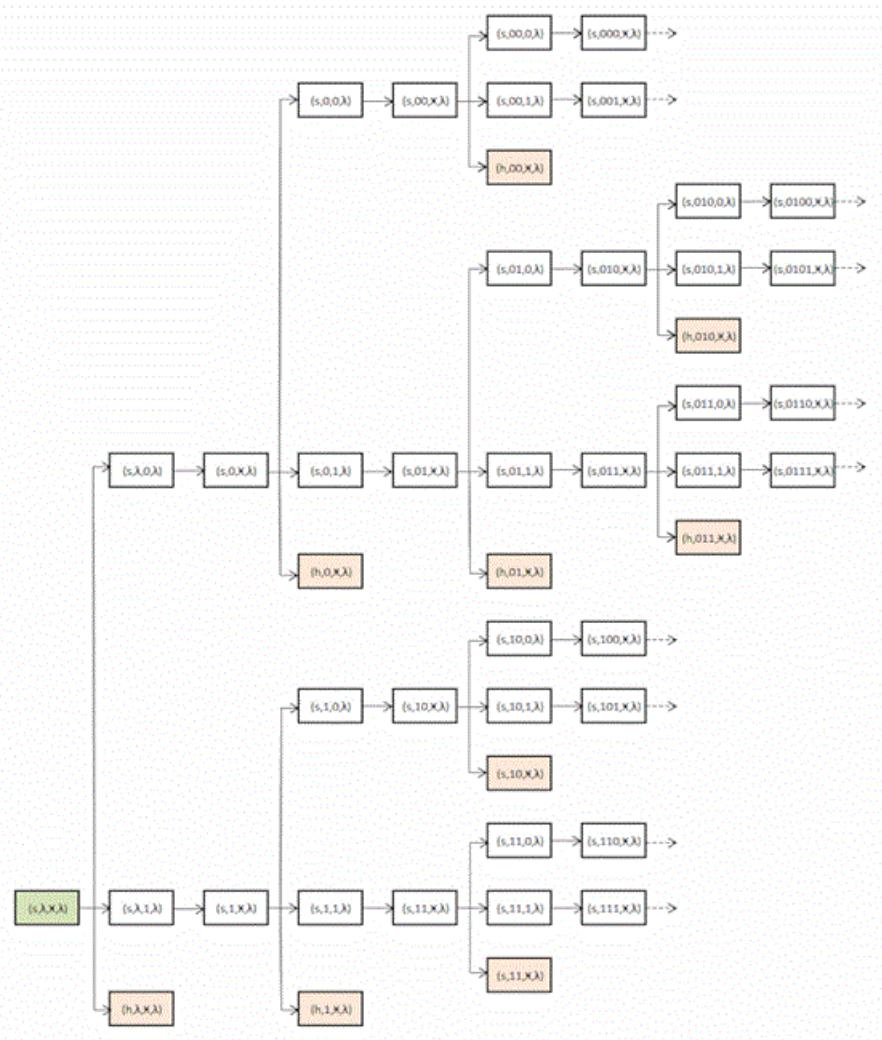
$\hat{\delta}$:

$$(s, *) \rightarrow ((s, 0), (s, 1), (h, *))$$

$(s, 0) \rightarrow ((s, \Rightarrow))$
 $(s, 1) \rightarrow ((s, \Rightarrow))$

A Turing-gép kimenete az üres szón: $T(\lambda) = \Sigma^*$.

A Turing-gép lehetséges számításai egy végtelen fastruktúrával írhatók le:



Az ábrán zöld színnel a kezdőkonfigurációt, pirossal a megálló konfigurációkat jelöltük.

2. Legyen $\hat{T} = (\mathcal{Q}, \Sigma, s, \hat{\delta}, H)$, ahol $\mathcal{Q} = \{s, q_1, h\}$, $H = \{h\}$, $\Sigma = \{0, 1\}$ és

$\hat{\delta}$:

 $(s, *) \rightarrow ((q_1, 0), (q_1, 1))$
 $(q_1, *) \rightarrow ((q_1, 0), (q_1, 1), (h, *))$
 $(q_1, 0) \rightarrow ((q_1, \Rightarrow))$
 $(q_{1,1}) \rightarrow ((q_1, \Rightarrow))$

A Turing-gép kimenete az üres szón: $T(\lambda) = \Sigma^*$.

3. Legyen $\hat{T} = (\mathcal{Q}, \Sigma, s, \hat{\delta}, H)$, ahol $\mathcal{Q} = \{s, q_1, h\}$, $H = \{h\}$, $\Sigma = \{0, 1\}$ és

$\hat{\delta}$:

$(s, *) \rightarrow ((q_2, 0), (q_1, 1))$

$(q_1, *) \rightarrow ((q_1, 0), (q_1, 1), (h, *))$

$(q_{10}) \rightarrow ((q_1, \Rightarrow))$

$(q_{11}) \rightarrow ((q_1, \Rightarrow))$

$(q_2, *) \rightarrow ((h, *))$

A Turing-gép kimenete az üres szón: $T(\lambda) = \mathbb{N}$.

Hasonlóan a determinisztikus Turing-gépekhez, itt is definiálhatjuk a felismerő Turing-gép fogalmát. A nemdeterminisztikus Turing-gépek működéséből adódóan azonban ez kicsit másképp értelmezhető mint a determinisztikus esetben.

6.6. Definíció

Legyen $\hat{T} = (\mathcal{Q}, \Sigma, s, \hat{\delta}, H)$ egy nemdeterminisztikus Turing-gép,
 $H = H^+ \cup H^-$ és $H^+ \cap H^- = \emptyset$.

Ekkor \hat{T} nemdeterminisztikus elfogadó Turing-gépnek nevezzük,
a H^+ -beli állapotokat pedig elfogadó állapotoknak.

6.7. Definíció

Legyen \hat{T} egy nemdeterminisztikus elfogadó Turing-gép,
 H^+ a \hat{T} elfogadó állapotainak halmaza.
Azt mondjuk, hogy \hat{T} elfogadja a $w \in \Sigma^*$ szót, hogy ha van \hat{T} -nek
olyan véges számítása a w bemeneten, amelyik végén megálláskor
 H^+ -beli állapotban van.

6.8. Definíció

Legyen \hat{T} egy nemdeterminisztikus elfogadó Turing-gép.
Az $L(\hat{T}) = \{w \mid w \in \Sigma^* \text{ és } \hat{T} \text{ elfogadja } w-i\}$ nyelvet a \hat{T} által felismert
nyelvnek nevezzük.

Azt mondhatjuk tehát, hogy egy nemdeterminisztikus Turing-gép elfogad egy w szót, ha van olyan számítása, amely elfogadja. Mindeközben azonban lehet olyan számítása is, amely elutasítja, ez nem befolyásolja azt, hogy elfogadja-e a Turing-gép. Egy szót akkor nem fogad el egy Turing-gép, ha nincs olyan számítása amelyik elfogadja. Látható, hogy ez általános esetben lényegesen bonyolultabb kérdés, mint determinisztikus esetben, hiszen itt akár végtelen sok számítás is indulhat egy kezdőkonfigurációból.

A determinisztikus Turing-gépek összefűzéséhez hasonlóan nemdeterminisztikus Turing-gépek esetén is lehetőségünk van egyszerűbb gépekből bonyolultabbak felépítésére.

6.9. Definíció (nemdeterminisztikus Turing-gépek összefűzése)

Legyen $\hat{T}_1 = (\mathcal{Q}_1, \Sigma_1, s_1, \hat{\delta}_1, H_1)$ és $\hat{T}_2 = (\mathcal{Q}_2, \Sigma_2, s_2, \hat{\delta}_2, H_2)$
 két nemdeterminisztikus Turing-gép.
 $\mathcal{Q}_1 \cap \mathcal{Q}_2 = \emptyset$

Tegyük fel, hogy $\hat{T} = \hat{T}_1 \rightarrow \hat{T}_2$.

A \hat{T} (vagy definiáljuk) Turing-gépet a következőképpen

$$\hat{T} = (\hat{\mathcal{Q}}, \hat{\Sigma}, \hat{s}, \hat{\delta}, \hat{H})$$

$$\hat{\mathcal{Q}} = \mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \{s_t\}^{\text{ahol}} \hat{\Sigma} = \Sigma_1 \cup \Sigma_2 \quad s = s_1$$

$$H = H_2 \quad \hat{\delta} = \hat{\delta}_1 \cup \hat{\delta}_2 \cup \delta' \quad \hat{H} = H_2$$

és

$$\hat{\delta}': (H_1 \cup \{s_t\}) \times (\Sigma \cup \{\star\}) \rightarrow 2^{(H_2 \cup \{s_t\}) \times (\Sigma \cup \{\star\})}$$

$$\hat{\delta}(h, a) = ((s_t, a)) \quad h \in H_1$$

$$\delta'(s_t, a) = ((s_t, \leftarrow)) \quad \text{ minden } \delta'(s_t, \leftarrow) = ((s_2, \leftarrow)) \quad \text{vagyamint esetén és } \delta'(s_t, \leftarrow) = ((s_2, \rightarrow))$$

Észrevehetjük, hogy a definícióban az egyik Turing-gépről a másikra való átugrás ugyanúgy egyértelmű mint a determinisztikus Turing-gépeknél, hiszen az ugrást megvalósító állapotokon az átmenetfüggvény értéke egyelemű halmaz.

2. Nemdeterminisztikus Turing-gépek szimulációja

Természetes kérdésként merül fel az emberben, hogy milyen kapcsolat van a determinisztikus és nemdeterminisztikus Turing-gépek között. Mennyivel másabb az új definíció és mit jelent ez algoritmikus hatékonyság szempontjából.

Hasonlóan a determinisztikus Turing-gépeknél leírtakhoz, nemdeterminisztikus Turing-gépekre is definiálhatjuk a szimuláció fogalmát.

6.10. Definíció

Legyen \hat{T}_1, \hat{T}_2 két nemdeterminisztikus Turing-gép. Azt mondjuk, hogy \hat{T}_2 szimulálja \hat{T}_1 -et, ha $Be(\hat{T}_1) \subseteq Be(\hat{T}_2)$ és $\forall w \in Be(\hat{T}_1)$ esetén $\hat{T}_1(w) = \hat{T}_2(w)$.

A szimuláció ilyen megfogalmazása azonban csak a nemdeterminisztikus Turing-gépek közötti összefüggések leírására alkalmas. Egy kis módosítással kiterjeszhetjük determinisztikus Turing-gépekre is.

6.11. Definíció

Legyen T_1 egy determinisztikus és \hat{T}_2 egy nemdeterminisztikus Turing-gép.
 Azt mondjuk, hogy \hat{T}_2 szimulálja T_1 -et, ha $Be(T_1) \subseteq Be(\hat{T}_2)$ és $\forall w \in Be(T_1)$ esetén $(T_1(w)) = \hat{T}_2(w)$.

Első megfigyelésként kimondhatjuk a következő tételeket.

6.12. Tétel

Minden T_1 determinisztikus Turing-géphez létezik \hat{T}_2 nemdeterminisztikus Turing-gép, amelyik szimulálja T_1 -t.

Bizonyítás

Tegyük fel, hogy a szimulálandó Turing-gép $\hat{T}_1 = (\mathcal{Q}_1, \Sigma, s_1, \delta_1, H_1)$. Ez alapján legyen $\hat{T}_2 = (\mathcal{Q}_2, \Sigma, s_2, \hat{\delta}_2, H_2)$, ahol

$$\mathcal{Q}_2 = \mathcal{Q}_1$$

$$s_2 = s_1$$

$$H_2 = H_1$$

és az átmenetfüggvény olyan, amire minden $q \in \mathcal{Q}$ és $a \in \Sigma$ esetén teljesül az

$$\hat{\delta}_2(q, a) = \{\delta_1(q, a)\}$$

egyenlőség.

A definíciója következtében \hat{T}_2 minden konfigurációjából legfeljebb egy másik konfigurációba léphet tovább.

Világos, hogy \hat{T}_2 szimulálja \hat{T}_1 -et, mivel \hat{T}_2 -nek tetszőleges w bemeneten legfeljebb egy lehetséges számítása van és az pontosan megegyezik \hat{T}_1 számításával, így a kimenetük is megfelel egymásnak. (Ha \hat{T}_2 -nek van kimenete, az mindenkorábban egy elemű.) ✓

6.13. Megjegyzés

Az előző téTEL bIZONYÍTÁSA szerint minden determinisztikus Turing-gépnek egyértelműen megfeleltethetünk egy vele lényegében teljesen megegyező nemdeterminisztikus Turing-gépet. Ezen megfeleltetés alapján a determinisztikus Turing-gépeket beágyazhatjuk a nemdeterminisztikus Turing-gépek halmazába. A beágyazás lehetősége miatt így egy determinisztikus Turing-gépet tekinthetünk nemdeterminisztikusnak is.

A téTEL alapján kimondhatjuk, hogy minden feladat, ami megoldható determinisztikus Turing-géppel, megoldható nemdeterminisztikus Turing-géppel is. Ez azt jelenti, hogy a nemdeterminisztikus Turing-gép mint számítási modell legalább olyan erős, mint a determinisztikus. Kérdés, hogy ténylegesen erősebb-e, azaz van-e olyan feladat, ami megoldható nemdeterminisztikus Turing-géppel, de nem oldható meg determinisztikussal. Az állítás pontos megfogalmazásához szükségünk van a szimuláció fogalmának megfelelő kiterjesztésére. A nemdeterminisztikus Turing-gépek kimenete nem egyetlen szó, hanem szavak halmaza, így kicsit nehézségebb lenne megfogalmazni a szimuláció definícióját teljesen általános esetre. Mivel nincs is igazán nagy szükség a későbbiek megértéséhez, ezzel nem is foglalkozunk a jegyzet keretein belül. Felismerő Turing-gépek esetében azonban a fogalom minden különösebb gond nélkül leírható.

6.14. Definíció

Legyen \hat{T}_1 egy nemdeterminisztikus és \hat{T}_2 egy determinisztikus felismerő Turing-gép. Azt mondjuk, hogy \hat{T}_2 szimulálja \hat{T}_1 -et, ha $L(\hat{T}_1) = L(\hat{T}_2)$.

A nemdeterminisztikus Turing-gép modell Turing-ekvivalenciája a következőképpen fogalmazható meg.

6.15. Tétel

Minden \hat{T}_1 nemdeterminisztikus felismerő Turing-géphez létezik \hat{T}_2 determinisztikus Turing-gép, amelyik szimulálja \hat{T}_1 -t.

Bizonyítás

A téTEL igazolását konstruktív módon végezzük. Mivel azonban a pontos bizonyítás sok apró technikai részlet leírásával járna, csak a vázlatát adjuk meg.

Legyen

$$\hat{T}_1 = (\mathcal{Q}_1, \Sigma, s_1, \hat{\delta}_1, H_1)$$

a szimulálandó nemdeterminisztikus Turing-gép.

A szimuláció alapötlete az, hogy a szimuláló \hat{T}_2 Turing-gép szélességi kereséssel bejárja \hat{T}_1 teljes számításfaját. Egészen pontosan, addig keres a számításfában, amíg

- a.) vagy meg nem talál egy elfogadó állapotú konfigurációt,
- b.) vagy el nem fogynak a folytatható konfigurációk.

\hat{T}_2 -nek két szalagja lesz. Az elsőn fogja tárolni azt a konfigurációt, amelyikből a továbblépései lehetőségeket számolja, míg a második egy FIFO szerkezetű tárolóként szolgál, amin a megvizsgalandó konfigurációkat tárolja.

A működés pontosabb leírása a következő:

1. Az 1. szalagon levő bemenő szó alapján a 2. szalagra előállítja a szimulálandó \hat{T}_1 kezdőkonfigurációját, majd mögé ír egy megfelelő elválasztójelet pl. #).
2. Megvizsgálja, hogy üres-e a 2. szalag. Ha üres, megáll nem elfogadó állapotban, egyébként folytatja a 3. lépésnél.
3. A 2. szalagról az 1-re másolja az első ott található konfigurációt, miközben le is törli onnan.
4. Megvizsgálja, hogy az 1. szalagon levő konfiguráció elfogadó végállapotban van-e. Ha igen, megáll elfogadó állapotban, egyébként folytatja az 5. lépéssel.
5. Megvizsgálja, hogy az 1. szalagon levő konfiguráció nemelfogadó végállapotban van-e. Ha igen, letörli az 1. szalagot, és folytatja az 2. lépéssel, egyébként továbblép a 6. lépésre.
6. Egy előre rögzített sorrendben végigmegy a \hat{T}_1 átmenetfüggvénye által meghatározott konfigurációátmeneteken, de nem végrehajtja őket, hanem egymás után elválasztó jelekkel elkülönítve a 2. szalag végére írja őket. Ha mindenki írt, folytatja a 2. lépéssel.

Az így megadott \hat{T}_2 determinisztikus Turing-gép megfelelő sorrendben szélességi bejárással végiglélked a \hat{T}_1 összes lehetséges számításának összes konfigurációján mindaddig, míg elfogadó állapotot tartalmazó konfigurációhoz nem ér. Ha ilyet talál, elfogadja a bemenetet és megáll. Ha elfogynak a bejárható konfigurációk (azaz a számításfa véges) nemelfogadó állapotban megáll. Ha a számításfa nem véges, de nincs benne elfogadó állapotú konfiguráció, végétlen ciklusba kerül.

A leírás alapján világos, hogy \hat{T}_2 pontosan azokat a w szavakat fogja elfogadni, amelyeket \hat{T}_1 , azaz $L(\hat{T}_1) = L(\hat{T}_2)$. ✓

6.16. Megjegyzés

1. A téTEL alapján tehát azt mondhatjuk, hogy a nemdeterminisztikus Turing-gép modell sem erősebb, mint a determinisztikus, azaz ha egy probléma megoldható nemdeterminisztikus Turing-géppel, akkor megoldható determinisztikussal is.

2. Ha L egy nyelv, és létezik \hat{T} nemdeterminisztikus Turing-gép, amelyikre $L = L(\hat{T})$ akkor $L \in R_f$.

Chapter 7. Bonyolultságfogalmak

A Turing-gép modell általánossága és egységesisége miatt nem csak a kiszámíthatósággal kapcsolatos kérdések megfogalmazására és vizsgálatára alkalmas.

Segítségével az egyes feladatok nehézségére is megfelelően kifejező mértéket definiálhatunk. A hétköznapi életben akkor nevezünk egy feladatot nehéznek, ha a megoldásához olyan műveleteket kell végrehajtanunk, amelyek megértése vagy kivitelezése a megszokottól eltérő, illetve meghaladja (vagy legalábbis súrolja) a képességeink határát. Egy számítógép számára azonban ilyen jellegű gondok nem léteznek. Nem kell megértenie a műveleteket, és amennyiben megfelelően leprogramoztuk, a végrehajtásuk sem okozhat gondot. Algoritmikus szempontból tehát teljesen át kell értelmezni egy feladat nehézségét. Ha egy számítógéptől várjuk a feladat megoldását, akkor mondjuk, hogy nehéz a gép számára, ha sokat kell várni a válaszra. Alapvetően ezt a megközelítést fogjuk kiterjeszteni és egzakt módon megfogalmazni a következőkben. Az elemzések és elméleti kutatások legfontosabb területe szintén az algoritmusok és feladatok időszükségletének vizsgálata, bár a pontosabb leírások során a társzükséglet is szerepet kaphat.

1. Idő-, tár- és programbonyolultság

Először definiálni fogjuk a determinisztikus és nemdeterminisztikus Turing-gépek idő-, tár- és programbonyolultságát. Ezekkel a fogalmakkal tudjuk kifejezni, hogy egy-egy algoritmusnak (Turing-gépnek) a különböző erőforrásokból milyen mennyiségre van szüksége. A Turing-gép fogalma lehetővé teszi, hogy a mennyiségi egységet nagyon pontosan meghatározzuk.

Mivel az algoritmusainkat általában nem egy feladat megoldására szeretnénk használni, hanem egy egész feladatosztály minden feladatára, ezért az erőforrásszükségletet úgy kell tudnunk kifejezni, hogy a feladatosztály minden elemére kielégítő pontossággal rendelkezzen.

Először a legfontosabb bonyolultságfogalommal, az időbonyolultsággal foglalkozunk.

7.1. Definíció

Legyen T egy determinisztikus Turing-gép és w egy szó. Legyen továbbá a T számítása a w bemeneten K_0, K_1, K_2, \dots .
Tegyük fel, hogy van olyan n amelyre K_n állapotkomponense végállapot. Ekkor azt a legkisebb n értéket amire ez a tulajdonság teljesül, a számítás hosszának nevezzük.
Ha nincs ilyen n , akkor a számítás hosszát végtelennek tekintjük.
A T Turing-gép által a w bemeneten végrehajtott számítás hossza jelekben: $\tau_T(w) = n$, illetve $\tau_T(w) = \infty$.

A számítás hosszának definíciójával tulajdonképpen az idő fogalmát tudjuk helyettesíteni. Anélkül, hogy tudnánk mit is jelent, definiáltuk az időegységet: egy konfigurációátmenet végrehajtásához szükséges erőforrás. Ezáltal egyben függetlenítjük magunkat attól, hogy a valós idő műlásával fejezzük ki az algoritmusaink, programjaink ilyen jellegű erőforrásigényét. A számítás hosszának segítségével definiálhatjuk a Turing-gépek általános időszükségletét.

7.2. Definíció

Legyen $T = (Q, \Sigma, s, \delta, H)$ egy determinisztikus Turing-gép.
A T Turing-gép időbonyolultsága a $t: \mathbb{N} \rightarrow \mathbb{N}$ függvény, amelyre:
 $t_T(n) = \max\{\tau_T(w) \mid w \in \Sigma^* \text{ ahol } l(w) \leq n\}$

Amennyiben az egyértelműség nem sérül, az időbonyolultság jelöléséből elhagyhatjuk a Turing-gép megadását. Az így definiált időbonyolultság segítségével azt tudjuk kifejezni, hogy egy adott Turing-gép (algoritmus) egy rögzített korláttnál nem nagyobb méretű feladaton legrosszabb esetben mennyi ideig számol. Emiatt szokás ezt az értéket a legrosszabb esethez tartozó időbonyolultságnak is nevezni.

Hasonló módon értelmezhető az átlagos időbonyolultság, ami a hétköznapi programozás szempontjából lényegesen kifejezőbb mérték, viszont jóval nehezebben kezelhető. Szükséges hozzá többek között a bemenő adatok eloszlásának ismerete, ami a valós esetekben ritkán határozható meg pontosan.

7.3. Megjegyzés

Észrevehetjük, hogy a definíció alapján ha egy Turing-gép időbonyolultsága minden n esetén definiált, akkor az minden bemeneten megáll, vagyis az általa felismert nyelv rekurzív.

7.4. Megjegyzés

Legyen T egy determinisztikus Turing-gép és t az időbonyolultsága.
 $t(n)$ értéke a leghosszabb számítás hossza a legfeljebb n hosszúságú, míg $t(n+1)$ értéke a leghosszabb számítás hossza a legfeljebb $n+1$ hosszúságú bemeneteken. Mivel az utóbbi eset tartalmaz minden bemenetet az előbbiből ezért ennek értéke nem lehet kisebb, azaz $\forall n \in \mathbb{N} : t(n) \leq t(n+1)$. Ez azt jelenti, hogy $t(n)$ monoton növekvő.

Hasonló módon határozhatjuk meg egy algoritmus (Turing-gép) számítás során jelentkező tárigényét.

7.5. Definíció

Legyen T egy determinisztikus Turing-gép és w egy szó. Legyen továbbá a T számítása a w bemeneten K_0, K_1, K_2, \dots .
Tegyük fel, hogy $K_n = (q_n, w_{1,n}, a_n, w_{2,n})$ és legyen $l_n = l(w_{1,n} \cdot a_n \cdot w_{2,n})$.
Ha létezik ilyen a $\max_{n=0,1,\dots} (l_n)$ értéket a számítás tárigényének nevezzük.
Ha nincs ilyen, akkor a számítás tárigényét végtelennek tekintjük.
A T Turing-gép által a w bemeneten végrehajtott számításhoz szükséges tárigényét $\sigma_T(w)$ -vel jelöljük.

Az időbonyolultságnál alkalmazott módszerrel definiálhatjuk a tárbonyolultságot is.

7.6. Definíció

Legyen $T = (Q, \Sigma, s, \delta, H)$ egy determinisztikus Turing-gép. A T Turing-gép tárbonyolultsága a $s : \mathbb{N} \rightarrow \mathbb{N}$ függvény, amelyre:
 $s_T(n) = \max\{\sigma_T(w) | w \in \Sigma^* \text{ ahol } l(w) \leq n\}$

A Turing-gép jelölését természetesen ebben az esetben is elhagyhatjuk, ha nem megy az érthetőség rovására.

Az időbonyolultságéhoz hasonló tulajdonság itt is megfigyelhető.

7.7. Megjegyzés

Legyen T egy determinisztikus Turing-gép és s a tárbonyolultsága.
 $s(n)$ értéke a legnagyobb tárigény a legfeljebb n hosszúságú, míg $s(n+1)$ értéke a legnagyobb tárigény a legfeljebb $n+1$ hosszúságú bemeneteken. Mivel az utóbbi eset tartalmaz minden bemenetet az előbbiből ezért ennek értéke nem lehet kisebb, azaz hasonlóan az időbonyolultság esetéhez $\forall n \in \mathbb{N} : s(n) \leq s(n+1)$. Ez azt jelenti, hogy $s(n)$ monoton növekvő.

Végül definiáljuk a Turing-gép összetettségét is.

7.8. Definíció

Legyen T egy determinisztikus Turing-gép és \hat{T}_T a Turing-gép programja. programbonyolultságának az értéket nevezzük.

7.9. Megjegyzés

A definíció alapján a programbonyolultság érteleke nem függ a bemenet hosszától. Az algoritmusok implementációjáról minden már nem mondható el egyértelműen. Amennyiben ugyanis a bemenet érteleke egy bizonyos korlátot meghalad, más szerkezetű programot kell irnunk. Értelelm szerűen hosszabb bemenethez hosszabb program tartozik. Más módszereket kell ugyanis használni a bemenet beolvasásához, az adatok memoriában való tárolásához és előfordulhat, hogy az egyes műveletek is más formában jelennek meg.

A determinisztikus Turing-gépekhez hasonlóan nemdeterminisztikus Turing-gépek esetén is definiálhatunk idő- és tárbonbonyolultságot. Mivel azonban a teljesen általános definíció lényegesen bonyolultabb és kevésbé kifejező lenne, így a megfelelő fogalmakat csak felismerő Turing-gépekre vezetjük be.

7.10. Definíció

Legyen \hat{T} egy nemdeterminisztikus felismerő Turing-gép és w egy szó.

Tegyük fel, hogy \hat{T} -nek van olyan számítása a w bemeneten amelyik elfogadó állapotban megáll. A legrövidebb ilyen lépések számát a számítás hosszának nevezzük.

Ha nincs elfogadó állapotban megálló számítása, akkor a számítás hosszát végtelennek tekintjük.

A \hat{T} által a w bemeneten végrehajtott számítás hosszát a determinisztikus Turing-gépekhez hasonló módon $\tau_{\hat{T}}(w)$ -vel jelöljük.

Nemdeterminisztikus Turing-gépek esetén a számítás hosszát csak olyan bemenő szavakra definiáltuk, amelyeket a Turing-gép elfogad. Ennek megfelelően az időbonbonyolultság fogalmán is módosítani kell egy kicsit.

7.11. Definíció

Legyen $\hat{T} = (Q, \Sigma, s, \hat{\delta}, H)$ egy nemdeterminisztikus Turing-gép.

A \hat{T} Turing-gép időbonbonyolultsága a $t: \mathbb{N} \rightarrow \mathbb{N}$ függvény, amelyre:

$$t_{\hat{T}}(n) = \max \{ \tau_{\hat{T}}(w) \mid w \in L(\hat{T}) \text{ és } l(w) \leq n \}$$

A nemdeterminisztikus Turing-gépek tárbonbonyolultságát a következőképen definiálhatjuk.

7.12. Definíció

Legyen \hat{T} egy nemdeterminisztikus felismerő Turing-gép és w egy szó.

Tegyük fel, hogy \hat{T} -nek van olyan számítása a w bemeneten amelyik elfogadó állapotban megáll.

Legyen \hat{T} egy elfogadó állapotban megálló lehetséges számítása a w bemeneten $K_0, K_1, K_2, \dots, K_n$. Tegyük fel, hogy $K_i = (q_i, w_{1,i}, a_i, w_{2,i})$ és legyen $l_i = l(w_{1,i} \cdot a_i \cdot w_{2,i})$.

Az $K_0, K_1, K_2, \dots, K_n$ elfogadó számításhoz tartozó tárígény:

$$l = \max_{i=0, \dots, n} \{ l_i \}$$

A lehetséges elfogadó számításokhoz tartozó tárígények közül a legkisebbet a Turing-gép w bementhet tartozó tárígényének nevezzük. Ha nincs elfogadó állapotban megálló számítása, akkor a számítás tárígényét végtelennek tekintjük.

A \hat{T} által a w bemeneten végrehajtott számítás tárígényét a determinisztikus Turing-gépekhez hasonló módon $\sigma_{\hat{T}}$ -vel jelöljük.

Hasonlóan az időbonyolultsághoz, a nemdeterminisztikus Turing-gépek esetén a számítás tárígényét csak olyan bemenő szavakra definiáltuk, amelyeket a Turing-gép elfogad. Ennek megfelelően az tárbonolultság fogalmán is módosítani kell egy kicsit.

7.13. Definíció

Legyen $\hat{T} = (Q, \Sigma, s, \hat{\delta}, H)$ egy nemdeterminisztikus Turing-gép.
A \hat{T} Turing-gép tárbonolultsága a $s : \mathbb{N} \rightarrow \mathbb{N}$ függvény, amelyre:
 $s_{\hat{T}}(n) = \max \{ \sigma_{\hat{T}}(w) | w \in L(\hat{T}) \text{ és } l(w) \leq n \}$

A következő tételek a rekurzív nyelvek alaposabb megisméréséhez nyújt segítséget. Azt láthatjuk be általa, hogy ha egy nyelv rekurzív felsorolható és időbonyolultsága nem túl nagy, akkor a nyelv egyben rekurzív is. Ez azt jelenti, hogy ha egy nyelv nem rekurzív, akkor (ha van egyáltalán) az őt felismerő Turing-gép időbonyolultsága kiszámolhatatlanul nagy. A kiszámolhatatlanul nagy az tényleg valami nagyon gyorsan növő függvényt jelent, hiszen még az Ackermann-függvény is kiszámolható.

7.14. Tétel

Legyen L egy nyelv. Tegyük fel, hogy létezik egy $t(n)$ Turing-géppel kiszámolható függvény és egy T determinisztikus Turing-gép, amelyikre $L = L(T)$.
Amennyiben T az L elemein legfeljebb $t(n)$ lépéssben megáll, akkor $L \in R$.

Bizonyítás

Legyen T' az a Turing-gép, amelyik kiszámolja $t(n)$ -t, azaz $T'(n) = t(n)$ és legyen $T_{korl,x}$ a korábbiakban megismert elven T -ből készített korlátozott futásidéjű Turing-gép. Ezek segítségével készítsük el a következő működésű Turing-gépet:

1. A w bemenet alapján az új Turing-gép 2. szalagjára írunk $t(n)$ darab 1-t.
2. A w bemeneten indítsuk el a $T_{korl,x}$ Turing-gépet. Ha a futás során a T rész áll meg előbb, akkor annak döntése lesz a tényleges végállapot, ha a korlátozó rész áll meg előbb akkor nemleges választ adunk.

Világos, hogy az új Turing-gép minden bemeneten megáll, és pontosan azokat a szavakat fogadja el, amelyeket amúgy T is elfogadott, hiszen ha T elfogadja, akkor azt kevesebb mint $t(n)$ lépéssben teszi. Definíció szerint ez azt jelenti, hogy $L \in R$. ✓

2. Bonyolultsági osztályok

A Turing-gépek bonyolultságfogalmai alapján feladatok nehézségét is definiálhatjuk. Ezek a definíciók lényegében azt fogalmazzák meg, hogy egy feladat olyan nehéz, mint az őt megoldó megfelelő szempontból vizsgált legjobb algoritmus (Turing-gép) bonyolultsága.

7.15. Definíció

Legyen $f : \mathbb{N} \rightarrow \mathbb{N}$ egy monoton növekvő függvény.
Ekkor a $\{f(n)\} = \{x | \exists x \text{ minden } n \in \mathbb{N} \text{ teljesül } f(x) \leq f(n) \text{ és } f(x) < f(n+1)\}$
halmazt az $f(n)$ időben determinisztikus Turing-géppel megoldható feladatok osztályának nevezzük.

7.16. Definíció

Legyen $f : \mathbb{N} \rightarrow \mathbb{N}$ egy monoton növekvő függvény.
Ekkor a

MÖRHA($f(n)$): $\{x \mid \text{az } f\text{-determinisztikus Turing-gép, melyre } L_0 \in L(x) \text{ és } t_0(x) \in O(f(n))\}$

halmazt az $f(n)$ tárral determinisztikus Turing-géppel megoldható feladatok osztályának nevezük.

7.17. Definíció

Legyen $f : \mathbb{N} \rightarrow \mathbb{N}$ egy monoton növekvő függvény.

Ekkor az

MÖRHO($f(n)$): $\{x \mid \text{az } f\text{-monoton növekvő Turing-gép, melyre } L_0 \in L(x) \text{ és } t_0(x) \in O(f(n))\}$

halmazt az $f(n)$ időben determinisztikus Turing-géppel megoldható feladatok osztályának nevezük.

7.18. Definíció

Legyen $f : \mathbb{N} \rightarrow \mathbb{N}$ egy monoton növekvő függvény.

Ekkor az

MÖRHA($f(n)$): $\{x \mid \text{az } f\text{-monoton növekvő Turing-gép, melyre } L_0 \in L(x) \text{ és } t_0(x) \in O(f(n))\}$

halmazt az $f(n)$ tárral determinisztikus Turing-géppel megoldható feladatok osztályának nevezük.

Az egyes időbonyolultsági osztályok között a következő egyszerű összefüggés állapítható meg.

7.19. Tétel

Legyen $f : \mathbb{N} \rightarrow \mathbb{N}$ és $g : \mathbb{N} \rightarrow \mathbb{N}$ két monoton függvény, amelyekre $f \in O(g)$.

Ekkor $\text{TIME}(f(n)) \subseteq \text{TIME}(g(n))$.

Bizonyítás

Mivel $f \in O(g)$, ezért $O(f) \subseteq O(g)$. Ez azt jelenti, hogy ha egy h függvényre $h \in O(f)$ akkor $h \in O(g)$ is teljesül. Az fentebbi definíció szerint, ha $L \in \text{TIME}(f(n))$ akkor létezik $t(n) \in O(f(n))$ időbonyolultságú determinisztikus Turing-gép, amelyikre $L = L(T)$. Mivel ekkor $t(n) \in O(g(n))$ is teljesül, ezért $L \in \text{TIME}(g(n))$, azaz $\text{TIME}(f(n)) \subseteq \text{TIME}(g(n))$. ✓

Hasonló eredmény a többi bonyolultsági osztályra is bizonyítható.

7.20. Tétel

Legyen $f : \mathbb{N} \rightarrow \mathbb{N}$ és $g : \mathbb{N} \rightarrow \mathbb{N}$ két monoton függvény, amelyekre $f \in O(g)$.

Ekkor

1. $\text{SPACE}(f(n)) \subseteq \text{SPACE}(g(n))$,

2. $\text{NTIME}(f(n)) \subseteq \text{NTIME}(g(n))$,

3. $\text{NSPACE}(f(n)) \subseteq \text{NSPACE}(g(n))$.

Bizonyítás

A bizonyítás teljesen hasonló az előző tételehez. ✓ A determinisztikus és nemdeterminisztikus bonyolultsági osztályok között is megállapítható egy egyszerű összefüggés. ✓

7.21. Tétel

Legyen $f : \mathbb{N} \rightarrow \mathbb{N}$ egy monoton függvény.

Ekkor

1. $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$;
2. $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$.

Bizonyítás

Ha $L \in \text{TIME}(f(n))$, akkor definíció szerint L -hez létezik $t(n) \in O(f(n))$ időbonyolultságú T determinisztikus Turing-gép, amelyre $L = L(T)$. Mivel minden determinisztikus Turing-gép tekinthető nemdeterminisztikusnak, ezért ugyanezzel a Turing-géppel a $L \in \text{NTIME}(f(n))$ tulajdonság is teljesül, vagyis $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$.

Hasonlóan bizonyítható a $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$ állítás is. ✓

A nyelvosztályok közül kiemelhetünk néhányat, amelyek az algoritmikus vizsgálatok során kiemelt szerepet játszik. Ezek a következők:

1. $\text{TIME}(n)$: a lehető legegyszerűbb feladatok. Ha ugyanis valamelyen feladatnak $t(n)$ időbonyolultsága van úgy, hogy $n \notin O(t(n))$ (azaz $t(n)$ határozottan lassabban nő mint n), akkor a feladatot megoldó Turing-gép nem olvashatja végig a teljes bemenetet.
Néhány ide tartozó feladat: összeadás, keresés, egy szó tükrözése, negálás, paritásellenőrzés és általában a reguláris nyelvek felismerése.

2. $\text{TIME}(n^2)$: az előző osztályhoz képest egy kicsit nehezebb feladatok. Ide tartozik pl. szorzás, osztás, mátrix-szorzás, polinom szorzás, rendezés, két szó leghosszabb közös részsorozatának keresése.

3. $P = \bigcup_{k=0}^{\infty} \text{TIME}(n^k)$ azaz a determinisztikus Turing-géppel polinomiális időben megoldható feladatok osztálya.
Ide tartoznak a viszonylag egyszerűen megoldható feladatok:
pl. legnagyobb közös osztó megkeresése, gráfban legrövidebb utak keresése, gráfban minimális súlyú feszítőfa keresése, legnagyobb áteresztőképesség számítása.

4. $EXP = \bigcup_{k=0}^{\infty} \text{TIME}(k^n)$ azaz a determinisztikus Turing-géppel exponenciális időben megoldható feladatok osztálya.
Lényegében a gyakorlatban előforduló feladatok nagy többsége ide tartozik. Ez csalóka megfigyelés, ugyanis megszámlálhatóan végtelen sok megoldható feladat nem tartozik ebbe az osztályba, viszont ilyenek a gyakorlatban nem nagyon kerülnek a látókörünkbe.

5. $\text{NSPACE}(n)$: a lineáris tárral nemdeterminisztikus Turing-géppel megoldható feladatok. Pl. a környezetfüggő nyelvek felismerése.

6. $PSPACE = \bigcup_{k=0}^{\infty} \text{SPACE}(n^k)$

7. $\text{NTIME}(n)$: a nemdeterminisztikus Turing-géppel lineáris időben megoldható feladatok. Meglepően sok feladat tartozik ide.

8. $NP = \bigcup_{k=0}^{\infty} \text{NTIME}(n^k)$ a nemdeterminisztikus Turing-géppel polinomiális időben megoldható feladatok.
Bonyolultságelméleti szempontból az egyik legérdekesebb nyelvosztály.

3. Tár-idő tételek

A tár- és időbonyolultsági osztályok közötti kapcsolatot az alábbi két tételel fejezhetjük ki.

7.22. Tétel

Legyen $f: \mathbb{N} \rightarrow \mathbb{N}$ egy monoton függvény. Ekkor $\text{TIME}(f(n)) \subseteq \text{SPACE}(f(n))$.

Bizonyítás

Legyen $L \in \text{TIME}(f(n))$. Ekkor létezik egy $t(n) \in O(f(n))$ időbonyolultságú T determinisztikus Turing-gép, amelyikre $L = L(T)$. Legyen w egy bemenő szó, aminek a hossza $l(w) = n$. Mivel egy Turing-gép minden lépéseinél legfeljebb egy új cellába írhat jelet a szalagján, ezért a w bemeneten végrehajtott számítás során felhasznált szalag mérete nem lehet nagyobb mint a Turing-gép lépéseinak a száma, vagyis $t(n)$. Ez azt jelenti, hogy $s(n) \leq t(n)$, azaz $s(n) \in O(f(n))$. Ez viszont definíció szerint azt jelenti, hogy $L \in \text{SPACE}(f(n))$, vagyis $\text{TIME}(f(n)) \subseteq \text{SPACE}(f(n))$. ✓

A két osztálytípus közötti fordított összefüggés ettől jóval összetettebb.

7.23. Tétel

Legyen $f: \mathbb{N} \rightarrow \mathbb{N}$ egy monoton függvény és legyen $L \in \text{SPACE}(f(n))$. Ekkor létezik $c > 0$ konstans, amelyikre $L \in \text{TIME}(c^{f(n)})$.

Bizonyítás

Legyen $T = (Q, \Sigma, s, \delta, H)$ egy $s(n) \in O(f(n))$ tárbonyolultságú determinisztikus Turing-gép, amelyikre $L = L(T)$.

Tegyük fel, hogy w egy olyan n hosszúságú szó, amelyiken a T Turing-gép megáll.

T tulajdonsága miatt ez azt jelenti, hogy a w bemeneten történő K_0, \dots, K_m számítás során, ha $K_i = (q_i, w_{1,i}, a_i, w_{2,i})$, akkor $l(w_{1,i} \cdot a_i \cdot w_{2,i}) \leq s(n)$ minden $0 \leq i \leq m$ esetén.

Ha létezne $0 \leq i < j \leq m$ amelyekre $K_i = K_j$, akkor a Turing-gép sohasem állna meg, hiszen ebben az esetben $K_{i+1} = K_{j+1}$ illetve általánosabban $K_{i+k} = K_{j+k}$ teljesülne, ami azt jelentené, hogy a Turing-gép periodikusan körbe-körbe lépeget ugyanazokon a konfigurációkon.

Ez alapján azt mondhatjuk, hogy a K_0, \dots, K_m konfigurációk között nincs két egyforma, vagyis m értéke nem lehet nagyobb, mint azon konfigurációk száma, amelyhez tartozó szalagtartalom legfeljebb $s(n)$.

Legyen Σ elemszáma S . A Σ feletti pontosan i \$i\$ hosszúságú szavak száma S^i , vagyis a legfeljebb i \$i\$ hosszúságú szavak száma $\sum_{k=0}^i S^k \leq S^{i+1}$.

Ha a Q elemszáma N , akkor az előzőek alapján a feltételeknek megfelelő különböző konfigurációk száma legfeljebb $N \cdot S^{s(n)+1} \cdot (s(n)+2)$. A formula abból adódik, hogy egy konfigurációban N különböző állapot, $S^{s(n)+1}$ különböző szalagtartalom és $s(n)+2$ különböző író-olvasó fej pozíció lehet.

Mivel N állandó, ezért $N < 2^{s(n)}$, ha n elegendően nagy. Hasonlóan az $s(n) + 2 < 2^{s(n)}$ is igaz megfelelően nagy n -ek esetén. Nagy n -ekre az is igaz, hogy

A három egyenlőtlenséget összeolvasva azt kapjuk, hogy

$$N \cdot S^{s(n)+1} \cdot (s(n) + 2) < 2^{s(n)} \cdot (S+1)^{s(n)} \cdot 2^{s(n)} = (4 \cdot (S+1))^{s(n)}.$$

Legyen $c = 4 \cdot (S+1)$. Az előzőek alapján a számítás hosszára igaz, hogy $m < c^{s(n)}$.

Azt mondhatjuk tehát, hogy T minden olyan bemenő szóról amin megáll, legfeljebb $c^{s(n)}$ időben eldönti, hogy benne van-e L -ben vagy sem. Az egyetlen probléma, hogy T nem feltétlenül áll meg minden bemeneten. Ezért készítük el a következő T' Turing-gépet:

T' -nek legyen egy második szalagja is, amire minden lépés után elválasztójelekkel elkülönítve felírja az aktuális konfigurációját, és leellenőrzi, hogy ez a konfigurációja szerepelt-e már korábban a szalagon. Ha szerepelt, az azt jelenti, hogy T végtelen ciklusba keveredett, vagyis nem fogadja el a bemenetét. Ebben az esetben T' nemleges válasszal megáll.

T' számítása természetesen hosszabb mint T számítása, mivel minden lépés előtt végig kell mennie a 2. szalagon. A második szalonon tárolt szó hossza azonban nem lehet nagyobb mint a különböző konfigurációk száma szorozva a konfigurációk maximális méretével, azaz kisebb mint $c^{s(n)} \cdot c_1 \cdot s(n)$ ami megfelelően nagy n -ek esetén felülről becsülhető $(c+1)^{s(n)}$ -nel. Így ha T' időbonyolultsága $t'(n)$, akkor $t'(n) \leq (c \cdot (c+1))^{s(n)}$. Mivel $s(n) \in O(f(n))$, ezért megfelelően nagy n -ek esetén $s(n) \leq c_2 \cdot f(n)$, valamelyen $0 < c_2$ -re. Legyen $C = (c \cdot (c+1))^{c_2}$ ezzel a C -vel igaz, hogy elegendően nagy n -ekre $t'(n) \leq (c \cdot (c+1))^{s(n)} \leq C^{f(n)}$, azaz $t'(n) \in O(C^{f(n)})$, vagyis $L \in \text{TIME}(C^{f(n)})$. ✓

Chapter 8. Az NP nyelvosztály

Az $\text{NTIME}(f(n))$ nyelvosztályok vizsgálata érdekes megfigyelésekre vezet. Ahogy az előző fejezetben láthattuk, hogy $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$. Felvetődik azonban a kérdés, hogy vajon az összefüggés valódi részhalmazt jelöl-e, vagy vannak olyan esetek, ahol a két nyelvosztály között egyenlőség áll fenn. A gyakorlati alkalmazások szempontjából az egyik legfontosabb a P nyelvosztály, hiszen ide tartoznak a viszonylag elfogadható időben megoldható feladatok. Ennek kapcsolata NP -vel egy igen sokat kutatott területe az algoritmuselméletnek.

A következőkben először is azzal foglalkozunk, hogy hogyan értelmezhetjük az NP -be tartozó feladatokat.

1. A Tanú-tétel

Elsőre nyilvánvalónak tűnik, hogy vannak feladatok, amelyek esetén a megoldás (vagy a megoldásra utaló valamilyen információ) ismeretében könnyebben meg tudjuk oldani a kérdéses feladatot - az előző fejezet alapján azt mondhatjuk, hogy algoritmikus szempontból a könnyebb megoldás a gyorsabbat jelenti. Óvatosan kell azonban a megoldás jelentésével bánni. Egy feladatot ugyanis csak akkor tekinthetünk megoldottnak, ha nem egyszerűen ismerjük a megoldást, hanem biztosak lehetünk benne, hogy valóban az. Ilyen megközelítésben már egyáltalán nem nyilvánvaló, hogy mely feladatok megoldását tudjuk bizonyos ismeretek birtokában gyorsabban meghatározni.

Hiába mondja meg például valaki, hogy $31 \cdot 13 = 403$, ahhoz, hogy biztosak legyünk az eredmény helyességében, lényegében újra ki kell számolnunk a szorzatot. Ha viszont szeretnénk meghatározni 403 prímtényezős felbontását, és valaki elárulja, hogy $31 \cdot 13 = 403$, akkor a keresgélés helyett nekünk már csak egyszerűen egy szorzást kell elvégezni, illetve ellenőrizni, hogy a tényezők valóban prímszámok.

Az NP -be tartozó nyelvek hasonló megközelítéssel írhatók körül. A következő tételben be fogjuk látni, hogy lényegében akkor tartozik egy feladat NP -be, ha megadható a megoldására vonatkozó nem túl sok információ, ami alapján viszonylag gyorsan meg tudjuk határozni a megoldást.

8.1. Tétel (Tanú-tétel)

Legyen $L \subseteq \Sigma^*$ egy nyelv. Ekkor a következő állítások ekvivalensek:

1. $L \in NP$.

2. Létezik egy $L' \subseteq \Sigma^P$ -beli nyelv és egy $p(x)$ polinom, amelyekre

$\forall w \in L$ akkor és csak akkor, ha $\exists v \in \Sigma^*$ úgy, hogy $l(v) \leq p(l(w))$ és $w \# v \in L'$.

Bizonyítás

A tétel két állítást foglal magába: az első szerint 1. implikálja 2-t, a második szerint pedig 2. implikálja 1-t. Ennek megfelelően a bizonyítást is két részre bontjuk.

A. 1. \Rightarrow 2.

Mivel $L \in NP$, ezért létezik egy $\hat{T} = (\mathcal{Q}, \Sigma, s, \hat{\delta}, H)$ polinomiális időbonyolultságú nemdeterminisztikus Turing-gép, amelyikre $L = L(\hat{T})$.

Legyen \hat{T} időbonyolultsága $t(n)$. Ez azt jelenti, hogy minden $w \in L$ esetén \hat{T} -nek létezik egy K_0, \dots, K_m lehetséges számítása w -n, ahol K_m elfogadó állapotot tartalmaz, és $m \leq t(n)$.

$\hat{\delta}(q, \alpha) = \{(\hat{q}, \alpha_1), \dots, (\hat{q}, \alpha_m)\}$ minden esetén rögzítsük elemeinek a sorrendjét, azaz tegyük fel, hogy

Ez alapján, ha adott \hat{T} egy K_0, \dots, K_m számítása, azt leírhatjuk egy $\alpha_1, \dots, \alpha_m \in \mathbb{N}$ sorozattal, ahol α_i annak az utasításnak (állapotátmenetnek) az indexe, amelyiket a $K_i \rightarrow K_{i+1}$ lépésben ténylegesen végrehajtottunk. Világos, hogy $\alpha_i \leq |\mathcal{Q} \times (\Sigma \cup \{\leftarrow, \Rightarrow\})|$ minden $1 \leq i \leq m$ esetén. Ez azt jelenti, hogy α_i ábrázolható Σ -fölötti szóként legfeljebb c hosszon, ahol $c = \log_N(|\mathcal{Q} \times (\Sigma \cup \{\leftarrow, \Rightarrow\})|) + 1$. Itt $N = |\Sigma|$. (Ha pontosabbak akarunk lenni, Σ -ból el kell hagynunk néhány vezérlő jelet az α_i -k ábrázolásához, így N értéke helyett egy kisebbet kell dolgoznunk, de ez nem változtat a tényen, hogy létezik egy megfelelő c .) A konstans c értéke csak \hat{T} -től függ.

Egy $w \in L$ szóhoz \hat{T} alapján rendeljük hozzá azt a v szót, amit a w legrövidebb elfogadó K_0, \dots, K_m számításához tartozó $\alpha_1, \dots, \alpha_m$ indexsorozat leírásával kapunk, azaz legyen $v = \alpha_1 \# \alpha_2 \# \dots \# \alpha_m$. Ekkor $l(v) \leq c \cdot m \leq c \cdot t(l(w))$. Legyen a téTELben szereplő polinom $p(x) = c \cdot t(x)$.

Legyen továbbá $L' = \{w \# v | w \in L \text{ és } v \text{ a } w \text{ legrövidebb elfogadó számításának index sorozata}\}$.

A téTEL első felének igazolásához már csak annyi kell belátni, hogy $L' \in P$.

Legyen T' egy determinisztikus Turing-gép, amit \hat{T} alapján definiálunk.

Eltekintve a teljesen pontos leírástól, a T' állapothalmazát egyszerűen $\mathcal{Q}' = \mathcal{Q} \times \{0, \dots, c\}$ -ként, míg átmenetfüggvényét $\delta'((q, \alpha_i), a) = (q_a, b_a)$ -ként definiáljuk, ahol (q_a, b_a) a $\hat{\delta}(q, a)$ halmaz α_i -dik eleme.

A T' Turing-gép lényegében szimulálja a \hat{T} működését. Az első szalagján a \hat{T} szalagtartalma, a másodikon a v segédszó található. Számítása során a v szót egyszer olvassa végig, de szakaszosan. minden szimulált lépés előtt megnézi v következő α_i komponensét, és ennek értékétől függően hajtja végre az első szalagról beolvastott a jelre a $\delta'((q, \alpha_i), a)$ által meghatározott konfigurációátmenetet.

A T' Turing-gép így pontosan a K_0, \dots, K_m konfigurációsorozatnak megfelelő számítást hajtja végre, vagyis pontosan azokat a szavakat fogadja el, amelyek L' -be tartoznak.

Időbonyolultsága lineáris, hiszen a bemenő szót (Pontosabban annak jelentős részét, v -t) csak egyszer olvassa el.

Ezzel a téTEL első felét beláttuk.

B. 2. \Rightarrow 1.

Legyen T' az L' -t felismerő $t'(n)$ polinomiális időbonyolultságú determinisztikus Turing-gép, és legyen \hat{T}_{Σ^*} az a nemdeterminisztikus Turing-gép, amelyik a bemenő szava mögé ír egy $\#$ jelet és az összes lehetséges Σ^* -beli szót.

Definiáljuk a következő nemdeterminisztikus Turing-gépet:

$$\boxed{\bar{T}} = \boxed{\bar{T}_{\Sigma^*}} \quad \boxed{T'}$$

Ez lényegében annyit csinál, hogy a bemenő w szó mögé írja az össze lehetséges $v \in \Sigma^*$ szót, és \hat{T} kipróbálja, hogy benne van-e az így kapott szó v -ben. Mivel a téTEL állításában szereplő $\$v\$$ hossza $t(n)$, a időbonyolultsága a következőképpen határozható meg:

Legyen $l(w) = n$. Amíg a \hat{T}^* komponens számol, addig $l(v) \leq p(l(w)) \leq p(n)$ lépést tesz meg. A második szakaszban, T' végrehajtása során legfeljebb $t(l(w)+1+l(v)) \leq t(n+1+p(n))$ ideig számol. Legyen $t(n)$ fokszáma d_t , $p(n)$ fokszáma pedig d_p . Ekkor $t(n+1+p(n))$ szintén polinom, aminek fokszáma $d_t \cdot d_p$, ezért a \hat{T} nemdeterminisztikus Turing-gép időbonyolultsága $t(n) \in O(p(n) + t(n+1+p(n))) = O(n^{d_t \cdot d_p})$, vagyis $L \in P$.

8.2. Megjegyzés

1. A téTELben szereplő v szót a w egy tanújának nevezzük.
 2. Az előbbi téTEL alapján azt mondhatjuk, hogy a tanú segítségével ellenőrizhetjük, hogy egy szó beletartozik-e az adott nyelvbe.
- Emiatt a tulajdonsága miatt az NP -be tartozó nyelveket polinomiális időben ellenőrizhető nyelveknek is szokták nevezni.

2. NP -teljesség

Az előzőekben megvizsgáltuk hogyan tudjuk ellenőrizni egy nyelv NP -be tartozását. Még mindig nem tisztáztuk azonban, hogy szükséges-e ez egyáltalán. Azt tudjuk ugyan, hogy $P \subseteq NP$ de nem beszéltünk arról, hogy a tartalmazás valódi-e, vagy egyenlőség áll fenn. Bár elvileg semmi nem zárja ki a bizonyítás lehetőségét, minden napig nem ismert sem a bizonyítása, sem a cáfolata annak, hogy $P = NP$.

Az állítást cáfolni úgy lehetne, hogy keresünk egy NP -beli nyelvet, amiről belájtuk, hogy determinisztikus Turing-géppel nem lehet polinomiális időben felismerni.

A $P = NP$ cáfolatára és igazolására is komoly erőfeszítéseket tettek. Ez utóbbi érdekében vezették be az NP -tejesség fogalmát. Ha röviden akarjuk meghatározni, az NP -teles nyelvek az NP különleges tulajdonságú, legnehezebb feladatinak tekinthetők. Ha akár csak egyről is sikerülne belátni, hogy P -be tartozik, akkor az összes NP -beli nyelvre igaz lenne.

Az első szükséges fogalom a következő.

8.3. Definíció (Karp-redukció)

Legyen $L_1, L_2 \subseteq \Sigma^*$ két nyelv. Azt mondjuk, hogy L_1 Karp-redukálható L_2 -re (jelekben: $L_1 \preceq L_2$), ha létezik egy T polinomiális időbonyolultságú determinisztikus Turing-gép, amelyikkel $\forall w \in \Sigma^*$ esetén $w \in L_1$ akkor és csak akkor, ha $T(w) \in L_2$.

8.4. Megjegyzés

A Karp-redukálhatóság megközelítőleg annyit jelent, hogy ha van egy feladatunk, azt könnyen át tudjuk fogalmazni egy másik feladattá, illetve ha ezt a másikat - egyszerűen - meg tudjuk oldani, akkor az eredeti feladatot is - egyszerűen - meg tudjuk oldani. Erre utal a jelölés is: a könnyebb feladat felől a nehezebb irányába nyilik a jel.

A Karp-redukciót arra tudjuk használni, hogy a nyelveket (feladatokat) nehézség szerint összehasonlítsuk. Ehhez belájtuk a Karp-redukálhatóság, mint reláció néhány tulajdonságát.

8.5. Tétel

Legyenek $L_1, L_2, L_3 \subseteq \Sigma^*$ három nyelv. Ekkor

1. $L_1 \prec L_2$ (reflexívitás), $L_1 \prec L_3$
2. ha és akkor (tranzitivitás).

Bizonyítás

1. A Karp-redukció definíciójában a T leképező Turing-gép legyen egy olyan Turing-gép, ami amikor elindul, egyből megáll, anélkül, hogy a szalagjának tartalmát megváltoztatta volna. Ez a Turing-gép az identikus leképezést valósítja meg, azaz $T(w) = w$, minden bemeneten. A definíció alapján ekkor pontosan a bizonyítandó állítást kapjuk.

2. Legyen T_1 és T_2 két olyan Turing-gép, amelyikkel a Karp-redukció definíciójában levő átalakítást végezzük az $L_1 \prec L_2$ és $L_2 \prec L_3$ relációknak megfelelő sorrendben, és legyen a két Turing-gép időbonyolultsága $t_1(n)$ és $t_2(n)$, az indexeknek megfelelően, melyek fokszáma d_1 és d_2 . Legyen $T = T_1 \cdot T_2$. Erre a T Turing-gépre igaz az, hogy ha $T_1(w) = w_1$ akkor $T(w) = T_2(w_1)$. Ez alapján, a definíciót felhasználva $\forall w \in \Sigma^* w \in L_1 \Leftrightarrow T_1(w) = w_1 \in L_2 \Leftrightarrow T(w) = T_2(w_1) \in L_3$.

Ha egy $w \in \Sigma^*$ hossza $n = l(w)$, akkor T_1 legfeljebb $t_1(n)$ időben kiszámolja w -et. w hossza legfeljebb annyi, mint amennyi ideig számol a Turing-gép, hiszen egy lépésben csak egy új jelet írhat a szalagra (egészen pontosan csak minden második lépésben írhat egy jelet), azaz $l(w_1) \leq t_1(l(w)) \leq t_1(n)$. A T_2 Turing-gép számításának hossza w -en legfeljebb $t_2(l(w_1)) \leq t_2(t_1(n))$. Legyen $t(n)$ a T időbonyolultsága. Az előzőek alapján egy tetszőleges legfeljebb n hosszúságú w bemeneten a T Turing-gép számításának hossza legfeljebb $t_1(n) + t_2(t_1(n))$, azaz $t(n) \in O(t_1(n) + t_2(t_1(n))) = O(n^{d_1+d_2})$. Beláttuk tehát, hogy T polinomiális időbonyolultságú Turing-gép és teljesülnek rá a Karp-redukció definíciójánál előírt feltételek, vagyis $L_1 \prec L_3$. ✓

A Karp-redukálhatóság szűkítésével egy ekvivalenciarelációt definiálhatunk.

8.6. Definíció

Legyenek L_1 és L_2 két nyelv. Azt mondjuk, hogy L_1 és L_2 Karp-ekviavalensek (jelekben $L_1 \equiv L_2$), ha $L_1 \prec L_2$ és $L_2 \prec L_1$.

8.7. Tétel

A Karp-ekviavalencia tényleg ekvivalenciareláció,

azaz $\forall L_1, L_2, L_3 \subseteq \Sigma^*$ nyelvek esetén

1. $L_1 \equiv L_1$;
2. ha $L_1 \equiv L_2$, akkor $L_2 \equiv L_1$;
3. ha $L_1 \equiv L_2$ és $L_2 \equiv L_3$, akkor $L_1 \equiv L_3$.

Bizonyítás

Az 1. és 3. eset a Karp-redukálhatóság 1. és 2. tulajdonságainak egyszerű következménye, a 2. eset pedig a definíció alapján könnyen látható, hiszen a két reláció $L_1 \prec L_2$ és $L_2 \prec L_1$ a nyelvek sorrendjétől függetlenül igazak. ✓

8.8. Megjegyzés

Két nyelv (feladat) akkor Karp-ekvivalens, ha polinomiális időben egymásba alakíthatók, azaz a nehézségük polinomiális átalakítástól eltekintve azonos.

A Karp-redukálhatóság értelmezése alapján nem meglepő a következő téTEL.

8.9. TéTEL

Legyen L_1 és L_2 két nyelv, amelyikre $L_1 \prec L_2$. Ekkor

1. ha $L_2 \in P$, akkor $L_1 \in P$ is igaz;

2. ha $L_2 \in NP$, akkor $L_1 \in NP$ is igaz.

BIZONYÍTÁS

Legyen T egy polinomiális $t(n)$ időbonyolultságú Turing-gép, amelyikkel a $L_1 \prec L_2$ Karp-redukálhatóság igazolható.

1.

Mivel $L_2 \in P$, ezért létezik egy T_2 polinomiális $t_2(n)$ időbonyolultságú Turing-gép, amelyikre $L_2 = L(T_2)$.

Legyen $T_1 = T \cdot T_2$. A Karp-redukció definíciója alapján $w \in L(T_1) \Leftrightarrow w \in L(T \cdot T_2) \Leftrightarrow T(w) \in L(T_2) \Leftrightarrow T(w) \in L_2 \Leftrightarrow w \in L_1$, azaz $L_1 = L(T_1)$.

Legyen $\deg(t(n)) = d$ és $\deg(t_2(n)) = d_2$, valamint w egy n hosszúságú szó és $T(w) = w'$. A w bemeneten T legfeljebb $t(n)$ ideig számol, és $l(w') \leq t(n)$. A w' bemeneten T_2 legfeljebb $t_2(l(w)) \leq t_2(t_1(n))$ ideig számol, amiből az előző tételekhez hasonlóan T_1 Turing-gép $t_1(n)$ időbonyolultságára azt kapjuk, hogy $t_1(n) \in O(n^{d+d_2})$, azaz T_1 polinomiális időbonyolultságú. Ezzel az állítást igazoltuk.

2.

Hasonlóan az előző esethez, mivel $L_2 \in NP$, ezért létezik egy \hat{T}_2 polinomiális $t_2(n)$ időbonyolultságú nemdeterminisztikus Turing-gép, amelyikre $L_2 = L(\hat{T}_2)$. Legyen $\hat{T}_1 = T \cdot \hat{T}_2$. A Karp-redukció definíciója alapján $w \in L(\hat{T}_1) \Leftrightarrow w \in L(T \cdot \hat{T}_2) \Leftrightarrow T(w) \in L(\hat{T}_2) \Leftrightarrow T(w) \in L_2 \Leftrightarrow w \in L_1$, azaz $L_1 = L(\hat{T}_1)$.

Legyen $\deg(t(n)) = d$ és $\deg(t_2(n)) = d_2$, valamint $w \in L_1$ egy n hosszúságú szó és $T(w) = w'$. A w bemeneten T legfeljebb $t(n)$ ideig számol, és $l(w') \leq t(n)$. Mivel az előzőek miatt $w' \in L_2$ ezért T_2 legfeljebb $t_2(l(w)) \leq t_2(t_1(n))$ időben elfogadja, amiből az előző tételekhez hasonlóan \hat{T}_1 nemdeterminisztikus Turing-gép $t_1(n)$ időbonyolultságára azt kapjuk, hogy $t_1(n) \in O(n^{d+d_2})$, azaz \hat{T}_1 polinomiális időbonyolultságú. Ezzel az állítást igazoltuk. ✓

Az alábbi téTEL meghatározza a Karp-ekvivalencia alapján legkönnyebb feladatok osztályát.

8.10. TéTEL

Legyen $L_1, L_2 \in P$ két nem triviális nyelv (azaz nem üresek és a komplementerük sem az). Ekkor $L_1 \equiv L_2$.

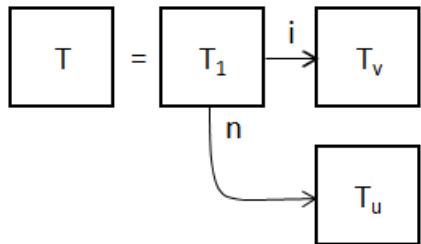
BIZONYÍTÁS

Elegendő azt belátni, hogy $L_1 \prec L_2$, ekkor ugyanis a szimmetria miatt $L_2 \prec L_1$ is teljesül.

Mivel L_2 nem triviális nyelv, ezért léteznek $u \notin L_2$ és $v \in L_2$ szavak. Továbbá, mivel $L_1 \in P$, ezért létezik egy T_1 polinomiális időbonyolultságú determinisztikus Turing-gép, amelyikre $L_1 = L(T_1)$.

Legyen T_u és T_v olyan Turing-gépek, amelyek letörlik a szalagjukat és u -t illetve v -t írnak rá.

Definiáljuk a következő Turing-gépet:



Ekkor egy tetszőleges w szó esetén $w \in L_1 \Leftrightarrow T_1(w) = v \Leftrightarrow T_1(w) \in L_2$. Mivel T_1 polinomiális T_u és T_v pedig konstans időbonyolultságú, ezért T is polinomiális időbonyolultságú Turing-gép. Ez viszont definíció szerint pontosan a bizonyítandó állítás. ✓

8.11. Megjegyzés

Ha bármely két nem triviális NP -beli nyelv Karp-ekvivalens lenne, akkor a $P = NP$ egyenlőség igaz lenne. Legyen ugyanis az egyik nyelv egy rögzített $L_1 \in P$, a másik pedig egy tetszőleges $L_2 \in NP$. Mivel $L_2 \prec L_1$, ez azt jelentné, hogy $L_2 \in P$.

Mivel a $P = NP$ állítás igazságát nem ismerjük, ezért nem tudjuk, hogy Karp-ekvivalens-e bármely két NP -beli nyelv. Definiálható viszont egy ehhez szorosan kötődő tulajdonság.

8.12. Definíció

1. Egy L nyelvet NP -nehéznek nevezünk, ha minden NP -beli nyelv Karp-redukálható rá.
2. Egy L nyelvet NP -teljesnek nevezünk, ha NP -beli és NP -nehéz.

Egy nyelv NP -teljessége azt jelenti, hogy az NP -beli nyelvek (feladatok) közül a legnehezebb.

8.13. Megjegyzés

Ha egy NP -teljes nyelvről sikerülne belátni, hogy P -beli, akkor a $P = NP$ igaz lenne.

Egyáltalán nem nyilvánvaló, hogy létezik NP -teljes nyelv. Ha viszont sikerül egyet találni, akkor egy újabb nyelv NP -teljességének igazolásához már csak azt kell ellenőrizni, hogy NP -beli legyen és Karp-redukálható legyen rá a másik NP -teljes nyelv. (A tranzitív tulajdonság miatt ugyanis ekkor minden NP -beli nyelv redukálható rá.) Ezen az elven már sok NP -beli nyelvről sikerült igazolni az NP -teljességet. Néhányat közülük a következő listában adunk meg.

NP -teljes nyelvek:

1. SAT - nyelv: a kielégíthető konjunktív normálformák nyelve.
2. A Hamilton-körrel rendelkező gráfok nyelve.
3. A 3 színnel színezhető gráfok nyelve.
4. k -klikk probléma (k -elemű teljes részgráf létezése)
5. k -csúcsú lefedés gráfokban

Irodalomjegyzék

[1] Ivanyos G., Rónyai L., Szabó R. Algoritmusok, Műszaki Könyvkiadó, Budapest, 1996

[2] Harry R. Lewis, Christos H. Papadimitriou : ELEMENTS OF THE THEORY OF COMPUTATION 1998

Error: no bibliography entry: d0e14231 found in
http://docbook.sourceforge.net/release/bibliography/bibliography.xml