

### 3.7 Изчислимост и разрешимост

В този параграф ще разгледаме машините на Тюринг от друга гледна точка.

Да предположим, както и досега, че  $V = \{a, b\}$ .

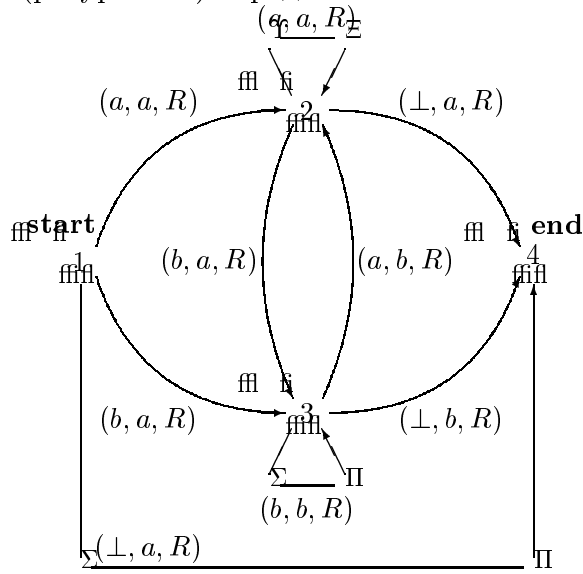
**Определение 3.7.1** Нека  $f(x_1, \dots, x_n)$  е функция дефинирана във  $V^{*n}$  със стойности във  $V^*$  т.е.  $f : V^{*n} \rightarrow V^*$ . Когато  $f$  е дефинирана за всяка  $n$ -орка  $(w_1, \dots, w_n)$  от думи над  $V$ , ще я наричаме **тотална**, а когато  $D_f \subseteq V^{*n}$ , ще я наричаме **частична**.

**Определение 3.7.2** Една частична функция  $f(x_1, \dots, x_n)$  е **изчислима (изчислима по Тюринг)**, ако съществува машина на Тюринг  $M$  над  $V \cup \{*\}$ , която пресмята функцията  $f$  т.е., която извършва следните действия с  $f(x_1, \dots, x_n)$  : за всеки  $n$  думи  $w_1, \dots, w_n$  над  $V$  машината  $M$  работи с входната дума  $w_1 * \dots * w_n$  и спира работа, ако  $f$  е дефинирана за  $n$ -орката  $(w_1, \dots, w_n)$ , с разпознаване или отхвърляне на тази дума, като съдържанието на лентата става  $f(w_1, \dots, w_n)$ , а когато  $f$  не е дефинирана за  $n$ -орката  $(w_1, \dots, w_n)$  машината  $M$  зацикля върху входната дума  $w_1 * \dots * w_n$ .

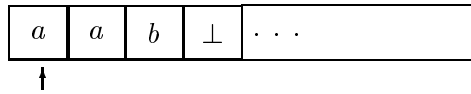
Оказва се, че повечето от функциите в множеството на целите числа могат да се изчислят с машина на Тюринг. Понеже функциите са основни обекти във всяка математическа теория, то въпросът за това, кои функции са изчислими по Тюринг и кои не има важно значение не само за основите на информатиката, но и за основите на самата математика. По-нататък ще направим една "класификация" на функциите, относно тяхната изчислимост с машини на Тюринг.

**Пример 3.7.1** Машината на Тюринг  $M_7$  зададена, чрез графа на Фиг.3.7.1 е на един аргумент  $x$  и тя му съпоставя  $ax$ , т.е. думата която се получава от  $x$ , като отляво се добави буквата  $a$ . Да означим тази функция с  $\mathbf{ad}_a(x)$ . Например, ако  $w = aab$ , то  $M_7$  започвайки работа върху лентата със съдържанието, показано на Фиг.3.7.2. преминава последователно през конфигурациите, представени на Фиг.3.7.3.

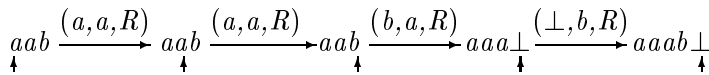
Един важен клас от функции, свързани с машините на Тюринг е класът на примитивно рекурсивните функции. Подобно на регулярните изрази, този клас от функции се въвежда, чрез рекурентно (рекурсивно) определение.



Фиг. 3.7.1



Фиг. 3.7.2



Фиг. 3.7.3

### Определение 3.7.3

(i) Базови примитивно рекурсивни функции над  $V$  са функциите:

$$\text{nil}(x) = \varepsilon, \quad \text{ad}_a(x) = ax \quad \text{и} \quad \text{ad}_b(x) = bx.$$

(ii) Ако  $h, g_1, \dots, g_m$  са примитивно рекурсивни функции над  $V$ , то такава е и функцията

$$f(x_1, \dots, x_n) = h(g(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n)),$$

която се нарича суперпозиция на  $h, g_1, \dots, g_m$ .

(iii) Ако  $h, g_1$  и  $g_2$  са примитивно рекурсивни функции над  $V$ , то такава е и функцията  $f(x_1, \dots, x_n)$ , за която:

при  $n = 1$ ,

$$f(\varepsilon) = w, \quad w \in V^*, \quad f(\mathbf{ad}_a(x)) = g_1(x, f(x)) \quad \text{и} \quad f(\mathbf{ad}_b(x)) = g_2(x, f(x)),$$

а при  $n \geq 2$   $f$  удовлетворява равенствата:

$$f(\varepsilon, x_2, \dots, x_n) = h(x_2, \dots, x_n),$$

$$f(\mathbf{ad}_a(x_1), x_2, \dots, x_n) = g_1(x_1, \dots, x_n, f(x_1, \dots, x_n)) \quad \text{и}$$

$$f(\mathbf{ad}_b(x_1), x_2, \dots, x_n) = g_2(x_1, \dots, x_n, f(x_1, \dots, x_n)).$$

(iv) Други примитивно рекурсивни функции над  $V$ , освен описаните в (i), (ii) и (iii) няма.

Да отбележим, че във функциите  $h, g_1$  и  $g_2$ , някои от аргументите могат да липсват т.е. да са фиктивни.

Да разгледаме някои примери на функции, които са примитивно рекурсивни над  $V = \{a, b\}$ .

1. Константите (букви)  $\mathbf{a}$  и  $\mathbf{b}$  са примитивно рекурсивни, защото

$$\mathbf{a} = \mathbf{ad}_a(\mathbf{nil}(x)) \quad \text{и} \quad \mathbf{b} = \mathbf{ad}_b(\mathbf{nil}(x)).$$

2. Тъждествената функция  $\mathbf{id}(x) = x$  е примитивно рекурсивна, защото може да се определи, по следния начин:

$$\mathbf{id}(\varepsilon) = \varepsilon, \quad \mathbf{id}(\mathbf{ad}_a(x)) = \mathbf{ad}_a(x) \quad \text{и} \quad \mathbf{id}(\mathbf{ad}_b(x)) = \mathbf{ad}_b(x).$$

3. Функцията  $\mathbf{con}(x_1, x_2) = x_1 x_2$ , която съединява две думи е примитивно рекурсивна, тъй като

$$\mathbf{con}(\varepsilon, x_2) = \mathbf{id}(x_2), \quad \mathbf{con}(\mathbf{ad}_a(x_1), x_2) = \mathbf{ad}_a(\mathbf{con}(x_1, x_2)) \quad \text{и}$$

$$\mathbf{con}(\mathbf{ad}_b(x_1), x_2) = \mathbf{ad}_b(\mathbf{con}(x_1, x_2)).$$

4. Функцията ”условен преход”, определена с равенството

$$\mathbf{if}(x_1, x_2, x_3) = \begin{cases} x_2 & \text{ако } x_1 \neq \varepsilon \\ x_3 & \text{ако } x_1 = \varepsilon \end{cases}$$

е примитивно рекурсивна, защото

$$\mathbf{if}(\varepsilon, x_2, x_3) = \mathbf{id}(x_3), \quad \mathbf{if}(\mathbf{ad}_a(x_1), x_2, x_3) = \mathbf{id}(x_2)$$

$$\text{и } \mathbf{if}(\mathbf{ad}_b(x_1), x_2, x_3) = \mathbf{id}(x_2).$$

Тази функция дава възможност за условен преход в програми и блок-схеми и тя е особено важна в програмирането.

**Определение 3.7.4** *Под  $n$ -местен предикат над  $V$  се разбира всяка функция  $p$  на  $V^{*n}$  в множеството от верностни стойности  $\{true, false\}$  т.е.*

$$p: V^{*n} \rightarrow \{true, false\}.$$

**Определение 3.7.5** *Един предикат се нарича примитивно рекурсивен, ако функцията*

$$f_p(x_1, \dots, x_n) = \begin{cases} \varepsilon & \text{при } p(x_1, \dots, x_n) = false \\ a & \text{при } p(x_1, \dots, x_n) = true, \end{cases}$$

*наречена **характеристична функция** на  $p$ , е примитивно рекурсивна.*

Да разгледаме някои примери на примитивно рекурсивни предикати над  $V$ .

1. Предикатите (константи) **true** и **false** са примитивно рекурсивни.

Наистина, ако  $p = \mathbf{true}(x) = \mathbf{true}$ , то характеристичната функция е  $f_{\mathbf{true}}(x) = a$  и аналогично, ако  $p = \mathbf{false}$ , то  $f_{\mathbf{false}}(x) = \mathbf{nil}(x)$ , които са примитивно рекурсивни функции.

2. Предикатът  $\mathbf{null}(x) = "x \text{ равно ли е на } \varepsilon"$  е примитивно рекурсивен, защото функцията

$$f_{\mathbf{null}}(x) = \begin{cases} a & \text{при } x = \varepsilon \\ \varepsilon & \text{при } x \neq \varepsilon \end{cases}$$

е примитивно рекурсивна функция.

Съвместното използване на примитивно рекурсивни предикати и функции, дава възможност да се дефинират едни от най-използуваните конструкции в програмирането.

Да положим  $\bar{x} = (x_1, \dots, x_n)$ , и нека  $p(\bar{x})$  е примитивно рекурсивен предикат, а  $f(\bar{x})$  и  $g(\bar{x})$  са примитивно рекурсивни функции. Тогава функцията

$$\mathbf{if}(p, f, g) = \begin{cases} f(\bar{x}) & \text{ако } p(\bar{x}) = true \\ g(\bar{x}) & \text{ако } p(\bar{x}) = false \end{cases}$$

е примитивно рекурсивна. В програмирането тази функция представя условния оператор и се записва по следния начин

$$\mathbf{if } p(\bar{x}) \text{ then } f(\bar{x}) \quad \text{else } g(\bar{x}).$$

Функцията условен оператор по естествен начин се обобщава до следната примитивно рекурсивна функция

$$\begin{aligned} &\mathbf{if } p_1(\bar{x}) \text{ then } f_1(\bar{x}) \quad \text{else } \mathbf{if } p_2(\bar{x}) \text{ then } f_2(\bar{x}) \\ &\text{else } \dots \mathbf{if } p_{k-1}(\bar{x}) \text{ then } f_{k-1}(\bar{x}) \quad \text{else } f_k(\bar{x}), \end{aligned}$$

където  $p_1, \dots, p_{k-1}$  са примитивно рекурсивни предикати, а  $f_1, \dots, f_k$  са примитивно рекурсивни функции.

Нека  $p(\bar{x})$  и  $q(\bar{x})$  са примитивно рекурсивни предикати. Тогава следните предикати са примитивно рекурсивни:

$$p(\bar{x}), \quad p(\bar{x}) \wedge q(\bar{x}), \quad p(\bar{x}) \vee q(\bar{x}), \quad p(\bar{x}) \leftrightarrow q(\bar{x}),$$

които се наричат съответно *отрицание*, *конюнкция*, *дизюнкция* и *равнозначност* на  $p$  и  $q$ .

Например, за дизюнкцията имаме

$$p(\bar{x}) \vee q(\bar{x}) = \begin{cases} true & \text{ако } p(\bar{x}) = true \text{ или} \\ & q(\bar{x}) = true \text{ или} \\ & p(\bar{x}) = q(\bar{x}) = true \\ false & \text{ако } p(\bar{x}) = false \text{ и} \\ & q(\bar{x}) = false. \end{cases}$$

Характеристичната функция на този предикат е

$$f_{p \vee q}(\bar{x}) = \text{if null}(\text{con}(f_p(\bar{x}), f_q(\bar{x}))) \text{ then } \varepsilon \text{ else } a,$$

която очевидно е примитивно рекурсивна.

Едно обобщение на примитивно рекурсивните функции е класът на *частично рекурсивните функции*. В този клас попадат освен всички примитивно рекурсивни функции и функциите, получени още чрез следните две възможности:

(i) ако  $g_1$  и  $g_2$  са частично рекурсивни функции, то такава е и функцията  $f$ , за която

$$f(\mathbf{ad}_a(x)) = g_1(x, f(x)) \quad \text{и} \quad f(\mathbf{ad}_b(x)) = g_2(x, f(x))$$

и евентуално  $f(\varepsilon)$  може да не е дефинирана, е частично рекурсивна;

(ii) ако  $p$  е частично рекурсивен предикат, то функцията  $f$ , за която  $f(\bar{x}) = y$ , където  $p(\bar{x}, y) = true$  и от  $\text{Nom}(w) < \text{Nom}(y)$  да следва, че  $p(\bar{x}, w) = false$ , е частично рекурсивна.

В случая (ii) казваме, че функцията  $f$  е получена, чрез *неограничена минимизация*. Да забележим, че ако  $p(\bar{x}, y) = false$  за всяко  $y$ , то  $f$  не е дефинирана за  $\bar{x}$ . Също така, ако  $p(\bar{x}, y)$  не е дефиниран и за всяко  $w \in V^*$  от  $\text{Nom}(w) < \text{Nom}(y)$  следва, че  $p(\bar{x}, w) = false$ , то  $f$  отново не е дефинирана за  $\bar{x}$ .

**Пример 3.7.2** Да разгледаме функцията  $\mathbf{tail}(x)$  ”опашката на  $x$ , т.е. думата  $x$  без първата буква”. Тя е частично рекурсивна функция защото

Очевидно всяка примитивно рекурсивна функция е и частично рекурсивна.

Тоталните частично рекурсивни функции се наричат *общо рекурсивни функции*.

Следващата теорема на Клини дава яснота за йерархията на рекурсивните функции и връзката им с машините на Тюринг.

**Теорема 3.7.1 (Теорема на Клини).** *Една функция е частично рекурсивна точно тогава, когато е изчислима по Тюринг.*

Доказателството на теоремата на Клини, ще предоставим за курса по математическа логика.

От казаното следва, че може да се направи следната йерархична схема за рекурсивните функции (Фиг.3.7.4):



**Фиг. 3.7.4**

Особен интерес за математиката представляват проблемите, на които отговора е "да" или "не". Такива проблеми ще наричаме *да/не проблеми*. Не по-малко интересен е и въпросът за установяване, дали съществува алгоритъм (програма, машина на Тюринг) за решаването на определен клас от да/не проблеми. В предходната глава разгледахме няколко такива проблеми, отнасящи се до автоматните граматиките, а в тази глава и за

безконтекстните граматики, като посочихме конкретните алгоритми, с които се решават тези проблеми (Виж Теорема 2.8.2, Теорема 2.8.3, Теорема 2.8.4, Теорема 3.5.1, Теорема 3.5.4).

**Определение 3.7.6** *Ще казваме, че един клас от да/не проблеми е **разрешим**, ако съществува краен алгоритъм (машина на Тюринг), който за всеки проблем от този клас, дава решението му. Всеки клас от да/не проблеми, за който не съществува такъв алгоритъм ще наричаме **неразрешим**.*

Оказва се, че една голяма част от да/не проблемите свързани с информатиката са неразрешими. Ето защо е важно да се опишат тези класове от проблеми, за да се избегне ненужното търсене на несъществуващи алгоритми. Ще направим едно кратко описание на част от най-важните неразрешими проблеми не само за информатиката, но и за цялата математика. За по-компактна формулировка на тези проблеми, ще използваме апарата на машините на Тюринг.

Най-напред, да се убедим, че всяка машина на Тюринг над азбуката  $V = \{a, b\}$  може да се представи, като дума над същата азбука. За осъществяването на такова представяне, съществуват различни начини, но ние ще се спрем на един от тях. Лесно се съобразява, че за всяка машина на Тюринг съществува еквивалентна на нея машина, която има само едно заключително състояние. Да приемем, че разглежданите машини на Тюринг имат единствено заключително състояние.

Нека  $M$  е произволна машина на Тюринг и нека на всяко ребро от нейната програма

$$\begin{array}{ccc} \text{ff} & \text{fi} & \\ \text{fff} & \xrightarrow{(\alpha, \beta, \gamma)} & \text{fff} \\ & i & j \end{array}$$

да съпоставим думата  $w = u(i)v(\alpha)v(\beta)w(\gamma)u(j)$  над  $V$ , където

$$u(k) = \begin{cases} \underbrace{abab \cdots b}_{k\text{-пъти}}aaaa & \text{ако } k \text{ е заключително състояние} \\ \underbrace{abab \cdots ba}_{k\text{-пъти}} & \text{в противния случай, } k \text{ — естествено;} \end{cases}$$



$$v(c) = \begin{cases} bbb & \text{ако } c = b \\ bba & \text{ако } c = a, \ c \in V; \end{cases} \quad w(T) = \begin{cases} bab & \text{ако } T = R \\ baa & \text{ако } T = L. \end{cases}$$

Предоставяме за самостоятелно упражнение, да се докаже, че това съпоставяне е взаимно-еднозначно.

Да забележим, че всяка машина на Тюринг се определя напълно от своята програма и следователно не възниква опасност от двусмислие, когато вместо *програма* пишем *машина* на Тюринг.

И така, всяка машина на Тюринг може да се смята за една дума над азбуката  $V$ . Да припомним, че множеството от думи над  $V$  е изброимо (Виж Теорема 2.1.1) и следователно всички машини на Тюринг могат да бъдат номерирани ("подредени" в редица). Да означим с  $Code(M)$  думата  $w$  от  $V^*$ , която се съпоставя на машината на Тюринг  $M$ .

От казаното, по естествен начин възниква въпросът, дали съществува "универсална" машина на Тюринг  $U$ , която като работи с  $Code(M)w$  за произволна машина  $M$  на Тюринг и за произволна входна дума  $w$  за  $M$  да извежда същия резултат, който извежда  $M$  при работата си върху думата  $w$ , т.е. дали съществува машина на Тюринг, която да може да пресмята всичко, което може да се пресмята? Отговорът на този въпрос, макар и неочакван е утвърдителен.

Нека да си представим, какво би извършил човек, ако трябваше да пресмята това, което искаме от машината  $U$ .

Като прегледа предоставената му лента с  $a$ -та и  $b$ -та, ще възстанови еднозначно програмата до първата поява на три последователни  $a$ -та и ще приложи възстановената програма върху входната дума, разположена върху лентата, непосредствено след  $Code(M)$ . Ясно е, че човек теоритически може да изпълнява функциите на универсалната машина на Тюринг. Тези доводи нямат сериозна математическа стойност, но те дават достатъчно надежда, че универсална машина на Тюринг наистина съществува. Всъщност, всеки добър програмист може да състави програма за работата на универсална машина, а пър-

вата такава програма е написана от самия А.Тюринг в 1936 г. така, че съществуването ѝ не подлежи на съмнение.

Съществуването на универсална машина на Тюринг води, непосредствено до идеята, че може да се построи универсален компютър, който да пресмята всичко, каквото може да се пресмята. За съжаление, техническите ограничения на компютрите не позволяват за особено голям оптимизъм в това отношение.

Непосредствено от направените по-горе разсъждения, може да се изведе така нареченият *стоп-проблем* за машините на Тюринг. Той се състои в намирането на алгоритъм, чрез който да може да се определи, дали произволна машина на Тюринг, като работи върху произволна входна дума ще спре работата си в заключително състояние или ще пресмята вечно т.е. ще зацikli. Ще покажем, че стоп-проблемът за машините на Тюринг е неразрешим.

**Теорема 3.7.2** *Стой-проблемът за машините на Тюринг е неразрешим.*

**Доказателство.** Ще докажем теоремата, като допуснем противното твърдение и достигнем до противоречие. Да предположим, че съществува алгоритъм (машина на Тюринг)  $A$ , който решава стой-проблема. Следователно за всяка входна дума  $w$ , алгоритъмът  $A$  дава верен отговор на въпроса, дали универсалната машина  $U$  спира работата си върху  $w$  в заключително състояние или не. Да разгледаме машината на Тюринг  $M$ , която извършва следните оператори върху произволна входна дума  $x$ :

1. Опитва се да възстанови машина на Тюринг  $M_1$ , за която  $Code(M_1) = x$ . Ако съществува такава машина отива на стъпка 2, а в противен случай се връща на стъпка 1.

2. Прави копие на  $x$  и го разполага върху лентата, непосредствено след последната буква на  $x$ , така върху лентата на  $M_1$  се получава думата  $w = xx = Code(M_1)Code(M_1)$  и преминава на стъпка 3.

3. Използва машината на Тюринг  $A$ , за да провери дали универсалната машина на Тюринг  $U$  спира в заключително състояние, работейки върху входната дума  $xx$ . Ако  $U$  спира,  $M$  отива на стъпка 3, а в противен случай  $M$  спира в свое заключително състояние.

Следователно машината  $M$  спира, ако са налице следните изисквания:

- а) съществува машина  $M_1$ , за която  $Code(M_1) = x$ ;
- б)  $U$  не спира в заключително състояние при входната дума  $xx$ .

Тъй като универсалната машина  $U$ , когато работи с думата  $Code(M_1)x$ , имитира поведението на  $M_1$  при работата ѝ върху думата  $x$  следва, че  $U$  не разпознава  $xx$  точно тогава, когато  $M_1$  не разпознава  $x$ . Следователно  $M$  спира в заключително състояние, работейки върху  $x$  точно тогава, когато съществува машина на Тюринг  $M_1$ , която зацикля при работа върху думата  $x$  и  $Code(M_1) = x$ . Ето защо, ако  $U$  разпознава една дума  $yy$ , то  $M$  не спира, започвайки работа върху думата  $y$ .

Нека  $Code(M) = z$  и да проверим, дали  $U$  спира, започвайки работа върху думата  $zz$ . От казаното по-горе следва, че

- (i)  $M$  спира върху входната дума  $z$ , ако  $U$  не спира върху думата  $zz$ .

От друга страна, понеже  $U$  е универсална машина, то тя имитира поведението на машината  $M$  и следователно

- (ii)  $U$  спира, работейки върху входната дума  $zz$  точно тогава, когато  $M$  спира върху думата  $z$ .

Тъй като (i) и (ii) взаимно се изключват следва, че допускането ни за съществуване на алгоритъма  $A$  не е вярно, с което теоремата е доказана.  $\square$

От това, че машините на Тюринг са еквивалентни на машините на Пост и на крайните машини с два стека следва, че стоп-проблемът и за тези машини също е неразрешим.

Всъщност стоп-проблемът има доста проста формулировка, допуска удивително лесно доказателство и в същото време представлява един много силен резултат в теорията на алгоритмите. Изводът, че стоп-проблема е неразрешим е много по-силен от това да считаме, че за сега нямаме решение или, че то е трудно. Проблемът има определена аналогия с трите прочути задачи на древността – квадратурата на кръга, трисекцията на ъгъла и удвояването на куба, но тази аналогия е само в отрицателния им отговор за решаване на самия проблем. Същественото различие на стоп-проблема от тези задачи е в това, че те са нерешими с линейка и пергел, докато стоп-проблемът е неразрешим с всички средства, достъпни за човека.

Доказателството, че стоп-проблемът е неразрешим, даде възможност да се отговори на редица важни въпроси в цялата математика, като се установи неразрешимостта на много да/не проблеми. Обикновено това се постига, като се допусне, че даденият проблем е разрешим и се достигне до противоречието, че от неговата разрешимост следва разрешимост на стоп-проблема. По-нататък ще изброим без доказателства, някои от по-важните неразрешими да/не проблеми в математиката и информатиката.

**Проблемът за диофантовите уравнения.** Един от първите неразрешими да/не проблеми е поставен от Д.Хилберт в знаменития му списък от проблеми пред математиката на двадесети век и е известен, като проблема за диофантовите уравнения.

Едно алгебрично уравнение се нарича *диофантово*, ако коефициентите му са цели и решенията му се търсят измежду целите числа. Проблемът за диофантовите уравнения се състои в това дали съществува алгоритъм, чрез който може да се даде отговор на въпроса има ли произволно диофантово уравнение поне едно решение.

Окончателното доказателство, че проблемът за диофантовите уравнения е неразрешим е достигнато от М.Дейвис и Ю.Матиясевич през 1970 г.

**Проблемът за полу-туевските системи.** Всяко крайно множество  $\Sigma$  от наредени двойки думи над  $V$ , се нарича полу-туевска система от думи т.е.

$$\Sigma = \{\langle \alpha_1, \beta_1 \rangle, \langle \alpha_2, \beta_2 \rangle, \dots, \langle \alpha_n, \beta_n \rangle\} \subset V^* \times V^*.$$

Ако  $u$  и  $v$  са две думи от  $V^*$ , ще казваме, че  $v$  е *изводима* (се *извежда*) от  $u$ , ако съществува редица от думи  $w_i$ ,  $i = 0, 1, \dots, p$ ,  $p \geq 0$  такива, че  $w_0 = u$ ,  $w_p = v$  и за всяко  $i$ ,  $0 < i \leq p$ ,  $w_i$  се получава от  $w_{i-1}$ , като в  $w_{i-1}$  е заместено  $\alpha_j$  с  $\beta_j$ , за някое  $j$ ,  $j \in \{1, 2, \dots, n\}$ . Проблемът за думите при полу-туевски системи е: съществува ли алгоритъм, чрез който да се установи дали  $v$  се извежда от  $u$  в  $\Sigma$ , за произволни думи  $u$  и  $v$  от  $V^*$ . В сила е следната теорема на Пост:

**Теорема 3.7.3 (Пост)** *Проблемът за думите при полу-туевски системи е неразрешим.*

**Проблемът за съответствието на Пост.** Ще казваме, че редицата от цели числа  $k_1, \dots, k_t$ ,  $t \geq 1$ ,  $1 \leq k_i \leq n$ ,  $i \leq t$  е решение на *системата от думи на Пост* над  $V$

$$S = \{\langle \alpha_1, \beta_1 \rangle, \langle \alpha_2, \beta_2 \rangle, \dots, \langle \alpha_n, \beta_n \rangle\} \subset V^* \times V^*,$$

ако

$$\alpha_{k_1} \alpha_{k_2} \dots \alpha_{k_t} = \beta_{k_1} \beta_{k_2} \dots \beta_{k_t}.$$

Проблемът за съответствието на Пост е: съществува ли алгоритъм, с който да се установи, дали произволна система на Пост над  $V$ , има решение.

**Теорема 3.7.4 (Пост)** *Проблемът за съответствието на Пост е неразрешим.*

В сила са и следващите теореми.

**Теорема 3.7.5** *Проблемът за установяване дали две безконтекстни граматика са еквивалентни, е неразрешим.*

Да отбележим, че същият проблем за автоматните граматика е разрешим (Виж Теорема 2.8.3).

**Теорема 3.7.6** *Проблемът за установяване дали дадена контекстна граматика поражда празния език, е неразрешим.*

Да отбележим, че същият проблем за безконтекстните граматика е разрешим (Виж Теорема 3.4.2).

**Теорема 3.7.7** *Проблемът за установяване дали произволна дума може да се изведе в произволна граматика от общ вид, е неразрешим.*

### З а д а ч и

1. Да се докаже, че следните функции са примитивно рекурсивни:

а)  $\text{rev}(x) = w$ ,  $w$  се състои от буквите на  $x$  написани в обратен ред;

а)  $\text{first}(x) = \alpha$ ,  $\alpha \in V$ , е първата буква на  $x$ ;

б)  $\text{tail}(x) = w$ , ако  $x = \alpha w$  за някое  $\alpha \in V$ ;

в)  $\text{right}(x) = \alpha$ ,  $\alpha \in V$  и  $\alpha$  е последната буква на  $x$ ;

г)  $\text{code}(x) = a^i$ , ако  $\text{Nom}(x) = i$  (вж. Глава 2);

д)  $\text{decode}(a^i) = x$ , ако  $\text{Nom}(x) = i$ ;

е)  $\text{sub}(a^i, a^j) = a^{i-j}$ , ако  $i \geq j$ ;

ж)  $\text{begin}(x) = w$ , ако  $x = w\alpha$  за някое  $\alpha \in V$ ;

з)  $\text{succ}(x) = w$ , ако  $\text{Nom}(w) = \text{Nom}(x) + 1$ ;

и)  $\text{pred}(x) = w$ , ако  $\text{Nom}(w) = \text{Nom}(x) - 1$ .

2. Да се докаже, че следните предикати са примитивно рекурсивни:

а)  $\text{eqa}(x) = "x \text{ равно ли е на } a?"$ ;

б)  $\text{eqb}(x) = "x \text{ равно ли е на } b?"$ ;

в)  $\mathbf{eq}(x, y) = "x \text{ равно ли е на } y?"$  за който

$$\mathbf{eq}(x, y) = \begin{cases} true & \text{ако } x = y \\ false & \text{ако } x \neq y. \end{cases}$$

3. Да се докаже, че следните функции са частично рекурсивни:

- а)  $\mathbf{first}(x) = \alpha$ ,  $\alpha \in V$ , е първата буква на  $x$ ;
- а)  $\mathbf{nodef}(x)$  е функция, която не е дефинирана за всяко  $x$ ,  $x \in V^*$ .

4\*. Да се докаже, че всяка примитивно рекурсивна функция е:

- а) тотална;
- б) изчислима по Тюринг.

5\*. Една система от думи  $\Sigma$ ,  $\Sigma \subset V^* \times V^*$  се нарича *туевска*, ако от  $\langle \alpha_i, \beta_i \rangle \in \Sigma$  следва, че  $\langle \beta_i, \alpha_i \rangle \in \Sigma$ . Да се докаже, че проблемът за думите в туевски системи е неразрешим.

6\*. Да се докаже, че проблемът за разпознаване дали произволна изчислима по Тюринг функция е тотална, е неразрешим.