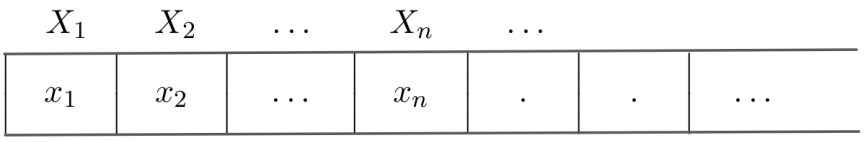
Лекция 5: Машини с неограничени регистри

2.1 Машини с неограничени регистри

Машините с неограничени регистри, или съкратено МНР (на англ. Unlimited register machines, URM), са абстрактен математически модел, с помощта на който в нашия курс ще дефинираме *изчислимите функции*. Тези машини са въведени през 1963 г. от Шефердсън и Стърджис.

2.1.1 Синтаксис на МНР



Лентата на една МНР

Всяка машина с неограничени регистри има изброимо много *регистри* $X_1, X_2, \ldots$, като във всеки регистър стои естествено число. Съдържанието на регистъра X_n ще означаваме с x_n. Да отбележим, че за разлика от *Машините на Тюринг*, при които във всяка клетка стои *буква* от лентовата азбука на машината, в регистрите на една МНР могат да се съхраняват произволно големи числа. Впрочем прилагателното "неограничени" в наименованието на тези машини е точно за да покаже, че съдържанието на техните регистри е неограничено (а не, че броят на регистрите е неограничен, както може да си помисли човек). Това, разбира се, отличава МНР и от истинските компютри и показва още веднъж, че тези машини са *идеализирани* изчислителни устройства.

Но какво може да прави една машина с неограничени регистри? Всъщност единственото, което може да прави, е да изпълнява много прости *инструкции*, които ще наричаме още и *оператори*. Тези оператори са общо 4 вида, разделени в две групи — *аритметични оператори* и *оператори за преход*.

Аритметични оператори:

$S(n)$ за всяко $n \geq 1$. Ще го записваме и като $X_n := X_n + 1$

$Z(n)$ за всяко $n \geq 1$. Запис: $X_n := 0$

$T(m, n)$ за всяко $m \geq 1$ и $n \geq 1$. Запис: $X_m := X_n$

Оператор за условен преход (jump):

$J(m, n, q)$ за всяко $m \geq 1, n \geq 1$ и $q \geq 0$,

който ще записваме по-човешки така: `if` $X_m = X_n$ `then goto` q

Разбира се, с този оператор можем да моделираме и безусловен `goto`, когато вземем $m = n$:

$$\texttt{if } X_m = X_m \texttt{ then goto } q.$$

Интуитивно е съвсем ясно какво прави всеки от тези 4 вида оператори. Ние ще дефинираме формално тяхната семантика в следващия раздел, след като въведем още няколко синтактични понятия.

Но най-напред да кажем как ще изглеждат програмите за нашите машини.

Определение 2.1. *Програма за машина с неограничени регистри* (или МНР програма) е всяка крайна редица от инструкции

$$I_0, \ldots, I_k.$$

Индексът l на I_l ще наричаме *адрес* (или *етикет*) на инструкцията.

Програмите ще означаваме с $P, Q, R, \ldots$, евентуално с индекси. Ще пишем

$$P\colon\ I_0, \ldots, I_k,$$

за да означим, че програмата P се състои от инструкциите $I_0, \ldots, I_k$.

2.1.2 Семантика на МНР

Нашите програми ще пресмятат функции на краен брой аргументи в естествените числа. Обаче за нашите цели е по-удобно да въведем тяхната семантика, като първоначално ги разглеждаме като преобразуватели на *всички* регистри $X_1, X_2, \ldots$ (или по-точно, на тяхното съдържание).

Както вече казахме, регистрите съдържат естествени числа. Следователно *текущото съдържание на регистрите* в даден момент от изчислението е безкрайната редица от естествени числа

$$(x_1, x_2, \ \ldots \ , x_n, \ldots).$$

(Да напомним, че с x_i означаваме съдържанието на i-тия регистър X_i.) Една такава безкрайна редица ще означаваме с $\widetilde{x}$. Ще я разглеждаме като елемент на декартовото произведение

$$\mathbb{N}^{\mathbb{N}} \ = \ \mathbb{N} \times \mathbb{N} \times \ldots.$$

Конфигурация (или моментна снимка) на дадена програма е наредената двойка $(l, \widetilde{x})$, където $l \in \mathbb{N}$ е адресът на оператора, който се изпълнява в дадения момент, а $\widetilde{x} = (x_1, x_2, \ \ldots)$ е текущото състояние на паметта в този момент. Конфигурацията $(l, (x_1, x_2, \ \ldots))$ ще отъждествяваме с безкрайната редица $(l, x_1, x_2, \ \ldots)$.

Начална конфигурация е конфигурацията $(0, \widetilde{x})$.

Заключителна (финална) конфигурация за програма с инструкции $I_0, \ldots, I_k$ е конфигурация от вида $(l, \widetilde{x})$, където $l > k$.

Да фиксираме произволна програма P: $I_0, \ \ldots \ , I_k$. Най-напред ще дефинираме *едностъпковото преобразование step* (или *функцията "стъпка"*) за тази програма. Изображението *step* ще преработва конфигурации в конфигурации, т.е. ще е от вида

$$step\colon \ \mathbb{N} \times \mathbb{N}^{\mathbb{N}} \ \longrightarrow \ \mathbb{N} \times \mathbb{N}^{\mathbb{N}}.$$

Идеята е да дефинираме семантиката на P като итерираме функцията *step* дотогава, докато се стигне до заключителна конфигурация (точно както при машините на Тюринг се итерира δ-функцията на преходите, докато се достигне финално състояние).

Стойността на *step* върху конфигурация $(l, x_1, x_2, \ldots)$ дефинираме с разглеждане на различните случаи за вида на оператора I_l:

$$step(l,\widetilde{x}) = \begin{cases} (l{+}1, x_1, \ldots x_{n-1}, x_n{+}1, x_{n+1}\ldots), & \text{ако } \ l \leq k \ \& \ I_l = S(n) \\ (l{+}1, x_1, \ldots x_{n-1}, 0, x_{n+1}\ldots), & \text{ако } \ l \leq k \ \& \ I_l = Z(n) \\ (l{+}1, x_1, \ldots x_{m-1}, x_n, x_{m+1}\ldots), & \text{ако } \ l \leq k \ \& \ I_l = T(m,n) \\ (q, \widetilde{x}), & \text{ако } \ l \leq k \ \& \ I_l = J(m,n,q) \\ & \& \ x_m = x_n \\ (l{+}1, \widetilde{x}), & \text{ако } \ l \leq k \ \& \ I_l = J(m,n,q) \\ & \& \ x_m \neq x_n \\ (l, \widetilde{x}), & \text{ако } \ l > k. \end{cases}$$

Типът на входа и на изхода на *step* е един и същ, следователно можем да итерираме тази функция. Да напомним означението

$$step^t(l,\widetilde{x}) \ = \ \underbrace{step(\ldots step}_{t \text{ пъти}}(l,\widetilde{x})\ldots).$$

Ясно е, че $step^t(l,\widetilde{x})$ ще е конфигурацията, до която е достигнала P след t стъпки, тръгвайки от конфигурацията $(l,\widetilde{x})$.

Ще казваме, че *P спира върху вход $\widetilde{x}$* (и ще пишем $P(\widetilde{x})\downarrow$), ако програмата P, тръгвайки от началната конфигурация $(0,\widetilde{x})$, след краен брой стъпки излезе от адресната си област $[0,k]$, т.е. попадне в заключителна конфигурация. Формално:

$$P(\widetilde{x})\downarrow \iff \exists t \exists l \exists \tilde{y}\ (step^t(0,\widetilde{x}) = (l,\tilde{y}) \ \& \ l > k).$$

Ако P не спира върху $\widetilde{x}$, ще пишем $P(\widetilde{x})\uparrow$.

Ако P спира върху $\widetilde{x}$, това, което ще ни интересува, е съдържанието на първия регистър. То ще бъде резултатът от работата на P върху $\widetilde{x}$.

Определение 2.2. Ще казваме, че МНР програмата P *спира върху вход $\widetilde{x}$ с резултат y* и ще пишем $P(\widetilde{x})\downarrow y$, когато

$$\exists t \exists l \exists \tilde{z}\ (step^t(0,\widetilde{x}) \ = \ (l,y,\tilde{z}) \ \& \ l > k).$$

Разбира се, очакваме ако P спре върху вход $\widetilde{x}$, резултатът да е еднозначно определен. Но доколкото в горната дефиниция на $P(\widetilde{x})\downarrow y$ има квантори за съществуване, този факт се нуждае от формална проверка. Да я направим.

Твърдение 2.1. (Коректност на дефиницията.)

$$P(\widetilde{x}) \downarrow y \;\&\; P(\widetilde{x}) \downarrow z \quad \Longrightarrow \quad y = z.$$

Доказателство. Ще използваме, че за всяка функция $f\colon\ M \longrightarrow M$ е вярно, че

$$f^{n+k}(x) \stackrel{\text{деф}}{=} \underbrace{f(\dots f}_{n+k \text{ пъти}}(x)\dots) = \underbrace{f(\dots f}_{n \text{ пъти}}(\underbrace{f\dots f}_{k \text{ пъти}}(x)\dots) = f^n(f^k(x)).$$

Ще покажем по-общата импликация

$$step^t(0,\widetilde{x}) = (l,\tilde{y}) \;\&\; l > k \;\&\; step^{t'}(0,\widetilde{x}) = (l',\tilde{y}') \;\&\; l' > k \quad \Longrightarrow \quad \tilde{y} = \tilde{y}'.$$

Без ограничение на общността можем да предполагаме, че $t' \geq t$. Тогава

$$step^{t'}(0,\widetilde{x}) \;=\; step^{t'-t}(step^t(0,\widetilde{x})) \;=\; step^{t'-t}(l,\tilde{y}) \;=\; (l,\tilde{y}).$$

За последното равенство използвахме, че върху заключителната конфигурация $(l,\tilde{y})$ функцията *step* действа като идентитет.

Получихме $step^{t'}(0,\widetilde{x}) = (l,\tilde{y})$; освен това имаме $step^{t'}(0,\widetilde{x}) = (l',\tilde{y}')$. Следователно $(l,\tilde{y}) = (l',\tilde{y}')$, и в частност, $\tilde{y} = \tilde{y}'$. □

2.1.3 Изчислимост на функция с програма за МНР

Вече имаме всичко необходимо, за да въведем едно от централните понятия в нашия курс — понятието *изчислима функция*.

Определение 2.3. Нека $n \geq 1$. Казваме, че програмата P *пресмята* n-местната частична функция $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ (или f *се пресмята от програмата* P), ако за всички естествени $x_1,\dots,x_n,y$ е в сила еквивалентността:

$$f(x_1,\dots,x_n) \simeq y \iff P(x_1,\dots,x_n,0,0,\dots) \downarrow y.$$

Определение 2.4. Казваме, че функцията $f\colon\ \mathbb{N}^n \rightharpoonup \mathbb{N}$ е *изчислима*, ако съществува програма за МНР, която я пресмята.

По-нататък ще пишем $P(\bar{x}) \downarrow y$ вместо $P(\bar{x},0,0,\dots) \downarrow y$, и аналогично $P(\bar{x}) \downarrow$ и $P(\bar{x}) \uparrow$ вместо $P(\bar{x},0,0,\dots) \downarrow$ и $P(\bar{x},0,0,\dots) \uparrow$.

Целта на този курс далеч не е да програмираме на езика за МНР, но все пак да разгледаме един пример за програма на този език.

Пример. Да се напише програма за МНР, която пресмята функцията събиране.

Решение. Началната конфигурация е $(0, x_1, x_2, 0, 0, \dots)$, т.е. всички регистри, освен входните X_1 и X_2, са 0. Резултатът трябва да получим в първия регистър.
Ще използваме, че

$$x_1 + x_2 \;=\; x_1 + \underbrace{1 + \cdots + 1}_{x_2 \text{ пъти}}.$$

Идеята ни е във for-цикъл да прибавяме x_2 пъти единица към x_1. Да означим с $I_0,\ I_1,\ I_2$ и I_3 операторите:

$I_0:$ `if` $X_2 = X_3$ `then goto` 4
$I_1:$ $X_3 := X_3 + 1$
$I_2:$ $X_1 := X_1 + 1$
$I_3:$ `if` $X_1 = X_1$ `then goto` 0

Тогава е ясно, че програмата $P: I_0,\ I_1,\ I_2,\ I_3$ пресмята функцията $f(x_1, x_2) = x_1 + x_2$. В официалния синтаксис тя ще изглежда така:

$$P\colon\ J(2,3,4),\ S(3),\ S(1),\ J(1,1,0).$$

□

Ако разглеждаме тази програма като пресмятаща функция на 3 аргумента, т.е. ако я извикаме с вход $(x_1, x_2, x_3, 0, 0, \dots)$, тя ще пресмята съвсем друга функция. Съобразете, че това ще е функцията

$$f(x_1, x_2, x_3) \simeq \begin{cases} x_1 + x_2 - x_3, & \text{ако } x_2 \geq x_3 \\ \neg!, & \text{ако } x_2 < x_3. \end{cases}$$

Понеже в P участват само регистрите X_1, X_2 и X_3, при $n \geq 4$ входни променливи тя очевидно ще продължава да пресмята горната функция f. По-точно, P ще пресмята n-местната функция, която на всяка n-торка $(x_1, \dots, x_n)$ съпоставя $f(x_1, x_2, x_3)$.

Когато $n = 1$, веднага се вижда, че програмата P ще пресмята идентитета $I(x) = x$. Така че една и съща програма може да пресмята съвършено различни функции, в зависимост от това каква е местността на тези функции.

По-надолу с $\mathcal{C}_n$ ще означаваме съвкупността от всички изчислими функции на n аргумента, т.е.

$$\mathcal{C}_n \;=\; \{f \mid f \in \mathcal{F}_n \;\&\; f \text{ е изчислима}\}.$$

2.2 Кодиране на програмите за МНР

В този раздел ще дефинираме *кодиране* на инструкциите и програмите за МНР с естествени числа. Това е стъпка, която има фундаментално значение за Теорията на изчислимостта. Като концепция, кодирането е въведено за пръв път от Гьодел като *аритметизация*, което в случая означава цифровизирането на всички аксиоми и правила за извод на определена формална система, както и кодирането на доказателства, теореми и пр. (т.е. превръщането на всичко това в естествени числа). По-късно в негова чест тази дейност се нарича още *гьоделизация*. В този дух може да се тълкува и Първият принцип на фон Нойман, който казва че "данните и инструкциите се съхраняват в една и съща оперативна памет и *няма логическа разлика между записа на данни и инструкции* (всичко е поредица от 0 и 1)".

Ние ще кодираме програмите за МНР с естествени числа, защото това са данните, върху които работят тези програми. Да отбележим, че когато изчислителният модел е базиран на *машини на Тюринг*, не възниква необходимост те да бъдат кодирани, защото самите машини на Тюринг са думи над определена азбука, точно както и входните данни за тези машини.

2.2.1 Кодиране на инструкциите

Програмите са редици от инструкции, тъй че преди да кодираме тях, е логично първо да се заемем с кодиране на инструкциите. Това може да стане по много начини; тук просто ще си изберем един от тях.

Да означим с $\mathbb{I}$ съвкупността от всички инструкции, т.е.

$$\mathbb{I} \;=\; \{I \mid I \text{ е инструкция за МНР}\}.$$

Ще дефинираме изображение

$$\beta\colon\ \mathbb{I} \longrightarrow \mathbb{N},$$

за което ще покажем, че е биекция. От доказателството ще се види още, че ако знаем числото $\beta(I)$, можем алгоритмично да възстановим инструкцията I.

Да си припомним четирите вида инструкции за МНР програмите:

1) $S(n),\ n \geq 1,$

2) $Z(n),\ n \geq 1,$

3) $T(m,n),\ m,n \geq 1,$

4) $J(m,n,q),\ m,n \geq 1, q \geq 0.$

Инструкциите са 4 вида, затова решаваме да ги кодираме с числа, даващи съответно остатък 0,1,2 и 3 при деление на 4. Да положим:

$$\begin{aligned} \beta(S(n)) &= 4(n-1) \\ \beta(Z(n)) &= 4(n-1)+1 \\ \beta(T(m,n)) &= 4\Pi(m-1,n-1)+2 \\ \beta(J(m,n,q)) &= 4\Pi_3(m-1,n-1,q)+3, \end{aligned} \tag{2.1}$$

където $\Pi(x,y) \stackrel{\text{деф}}{=} 2^x(2y+1)-1$ е кодирането на $\mathbb{N}\times\mathbb{N}$, а Π_3 е биекцията (1.8) между $\mathbb{N}^3$ и $\mathbb{N}$, които въведохме в раздел 1.8.1.

Да отбележим, че дефинирахме $\beta(S(n))$ като $4(n-1)$, а не $4n$, както може би ви се струва по-естествено, защото номерацията на регистрите X_n започва от $n=1$ (а не от $n=0$), а ние ще искаме да осигурим всяко число да е код на някаква инструкция. По тази причина в последния случай имаме $\beta(J(m,n,q)) = \Pi_3(m-1,n-1,q)+3$, защото q има смисъл на адрес, а адресирането започва от 0.

Числото $\beta(I)$ ще наричаме *код на инструкцията* I. От дефиницията на β е почти очевидно, че всяко естествено число е код на единствена инструкция, но да го проверим все пак.

Твърдение 2.2. Изображението $\beta\colon \mathbb{I} \longrightarrow \mathbb{N}$ е биекция.

Доказателство. Трябва да видим, че за всяко $z \in \mathbb{N}$ съществува единствена инструкция I, такава че $\beta(I) = z$. Ясно е, че най-напред трябва да разгледаме остатъка на z по модул 4. Ако той е 0, z е код на инструкцията $S(n)$, където $n = \frac{z}{4}+1$ и тази инструкция е единствената с това свойство.

Ще пропуснем случаите, когато z кодира други аритметични оператори и ще разгледаме направо случая $z = 4t+3$. В този случай единствената възможност е z да е код на оператор за преход. Нека

$$m = J_1^3(t)+1, \quad n = J_2^3(t)+1, \quad q = J_3^3(t),$$

където $J_i^3, 1 \leq i \leq 3$ са декодиращите за кодирането Π_3. Да се убедим, че кодът на инструкцията $J(m,n,q)$ е точно z:

$$\begin{aligned} \beta(J(m,n,q)) &\stackrel{\text{деф}}{=} 4\Pi_3(m-1,n-1,q)+3 = \\ 4\Pi_3(J_1^3(t),J_2^3(t),J_3^3(t))+3 &= 4t+3 = z. \end{aligned}$$

Да обърнем внимание, че ние не само показахме, че всяко естествено число z е код на единствена инструкция, но и във всеки от 4-те случая за z посочихме тази инструкция. □

2.2.2 Кодиране на програмите

Всяка програма за МНР е просто една непразна редица от инструкции

$$I_0, \ldots, I_k.$$

Ние, обаче, вече знаем как да кодираме инструкциите. Освен това знаем и как да кодираме непразни редици от естествени числа — посредством изображението $\tau\colon \mathbb{N}^* \longrightarrow \mathbb{N}$, което в раздел 1.8.3 дефинирахме ето така:

$$\tau(\langle x_0, \ldots, x_n\rangle) \;=\; \Pi(n, \Pi_{n+1}(x_0, \ldots, x_n)).$$

Затова не буди изнада и следващата дефиниция за *код на програма* P : $I_0, \ldots, I_k$. Този код, който ще означаваме с $\gamma(P)$, която дефинираме по следния начин:

$$\gamma(P) \stackrel{\text{деф}}{=} \tau(\langle \beta(I_0), \ldots, \beta(I_k)\rangle). \tag{2.2}$$

Съвкупността от всички програми ще означаваме с $\mathbb{P}$:

$$\mathbb{P} \;=\; \{P \mid P \text{ е програма за МНР}\}.$$

Да се убедим, че всяко естествено число е код на единствена програма, с други думи:

Твърдение 2.3. Изображението $\gamma\colon\ \mathbb{P} \longrightarrow \mathbb{N}$ е биекция.

Доказателство. Следва от факта, че τ и β са биекции. Ние все пак ще го разпишем, за да си представим как точно по дадено число a възстановяваме операторите на програмата с код a. Това ще ни е от полза по-нататък, когато ще строим универсалната функция.

Наистина, да вземем произволно $a \in \mathbb{N}$. Понеже τ е биекция, ще съществуват единствени числа $x_0, \ldots, x_k$, такива че

$$\tau(\langle x_0, \ldots, x_k\rangle) \;= a.$$

Съгласно предишното *Твърдение* 2.2, съществуват единствени инструкции $I_0, \ldots, I_k$, за които

$$\beta(I_l) \;=\; x_l \quad \text{за всяко } l = 0, \ldots, k.$$

Ясно е, че програмата $P\colon\ I_0, \ldots, I_k$ ще има код a и тя ще е единствената с това свойство. Да отбележим, че както в предишното твърдение, и тук успяхме *да конструираме* тази програма. □

2.2.3 Ефективно номериране на програмите и изчислимите функции

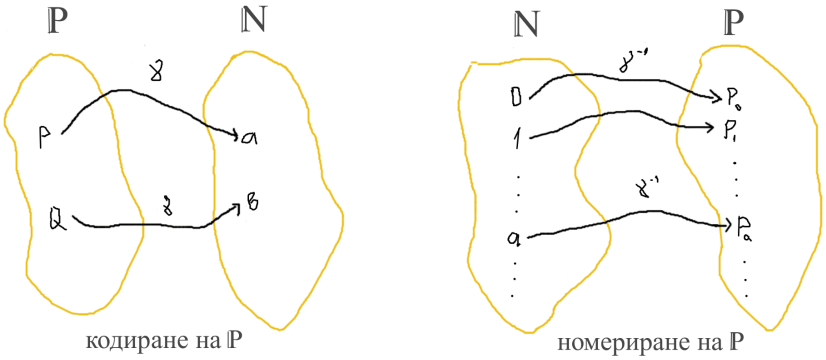
Току-що доказаното *Твърдение* 2.3 ни гарантира, че за всяко $a \in \mathbb{N}$ съществува единствена програма P, такава че $\gamma(P) = a$. Да означим тази програма с P_a, т.е.

$$P_a \overset{\text{деф}}{=} \text{програмата с код } a.$$

Иначе казано, P_a е $\gamma^{-1}(a)$.
Ясно е, че редицата

$$P_0,\ P_1,\ \ldots,P_a,\ \ldots \tag{2.3}$$

включва всички МНР програми и само тях, т.е. тя е едно фиксирано *изброяване* или *номериране* (enumeration) на множеството $\mathbb{P}$ на всички програми. Ще я наричаме *стандартно изброяване* на $\mathbb{P}$.



Разбира се, винаги можем да подредим МНР програмите в една безкрайна редица, защото те са изброимо много. Това, което отличава изброяването (2.3), е че то е *ефективно*, т.е. имаме алгоритъм, който по всяко a възстановява програмата с код a. Да отбележим, че в това изброяване няма *синтактични* повторения, с други думи, няма две едни и същи програми. В него, обаче, има *семантични* повторения (т.е. срещат се еквивалентни програми) и този факт, както ще видим по-нататък, няма как да бъде избегнат. Да, теоретично можем да наредим в една редица всички семантично различни програми (просто защото те са изброимо много), но всяка такава редица ще бъде със сигурност неалгоритмична.

Номерирането на всички МНР програми ни дава възможност да номерираме и всички изчислими функции. Това става със следното определение.

Определение 2.5. Да фиксираме $n \geq 1$. За всяко $a \in \mathbb{N}$ полагаме

$$\varphi_a^{(n)} \ = \ n\text{-местната функция, която се пресмята от програмата } P_a.$$

При $n = 1$ горният индекс обикновено се пропуска, т.е.

$$\varphi_a \ = \ \text{едноместната функция, която се пресмята от програмата } P_a.$$

Горните означения ни дават възможност да наредим в редица и всички n-местни изчислими функции.

Твърдение 2.4. За всяко $n \geq 1$ редицата

$$\varphi_0^{(n)},\ \varphi_1^{(n)},\ \ldots\ ,\varphi_a^{(n)},\ \ldots \tag{2.4}$$

включва всички n-местни изчислими функции и само тях.

Доказателство. По определение всички функции от тази редица са n-местни и изчислими. Обратно, ако $f \in \mathcal{F}_n$ е изчислима, дали ще се срещне в редицата? Очевидно да, защото щом тя е изчислима, значи съществува поне една програма, която я пресмята. Ако означим с a кода на тази програма, то по дефиниция $f = \varphi_a^{(n)}$. □

Така получихме, че редицата (2.4) е изброяване на класа $\mathcal{C}_n$ на всички n-местни изчислими функции. Ще го наричаме *стандартно изброяване (номериране)* на $\mathcal{C}_n$.

Това изброяване е ефективно в смисъл, че по дадено a можем да разберем коя е функцията $\varphi_a^{(n)}$, по-точно, можем да разберем коя е програмата, която пресмята тази функция.

Да отбележим, че всяка функция f от редицата (2.4) се повтаря безброй много пъти, защото ако f се пресмята от програмата $P:\ I_0, \ldots, I_k$, то очевидно същото ще прави, например, и всяка програма Q_c, където

$$Q_c:\ I_0, \ldots, I_k, \underbrace{X_1 := X_1, \ldots, X_1 := X_1}_{c \text{ пъти}}$$

Тези повторения не могат да бъдат избегнати. Ако наредим без повторения изчислимите функции от $\mathcal{C}_n$, тя ще загуби други важни свойства и на практика ще бъде една съвсем безполезна редица.

Ще завършим този раздел с важното понятие за *индекс* на изчислима функция.

Определение 2.6. За произволна n-местна изчислима функция f, *индекс (номер)* на f ще наричаме всяко число a, за което

$$f \ = \ \varphi_a^{(n)}.$$

Ясно е, че ако една функция има един индекс, то тя има безброй много индекси — това са кодовете на всички програми, които я пресмятат. По-нататък ще видим, че съвкупността от всички индекси на дадена изчислима функция образуват доста сложно множество.

2.3 Универсални функции

2.3.1 Определение

Нека $\mathcal{K}$ е съвкупност (или клас) от n-местни функции. Ще дефинираме важното понятие за *универсална функция (УФ)* за този клас.

Определение 2.7. Ще казваме, че функцията $U(a, x_1, \ldots, x_n)$ е *универсална* за класа $\mathcal{K} \subseteq \mathcal{F}_n$, ако са изпълнени условията:

0) U е изчислима;

1) за всяка $f \in \mathcal{K}$ съществува $a \in \mathbb{N}$: $f(\bar{x}) \simeq U(a, \bar{x})$ за всяко $\bar{x} \in \mathbb{N}$;

2) за всяко $a \in \mathbb{N}$ съществува $f \in \mathcal{K}$: $U(a, \bar{x}) \simeq f(\bar{x})$ за всяко $\bar{x} \in \mathbb{N}$.

Ако използваме λ-означенията, условията 1) и 2) можем да запишем по-кратко така:

1) за всяка $f \in \mathcal{K}$ съществува $a \in \mathbb{N}$: $f = \lambda\bar{x}.U(a, \bar{x})$;

2) за всяко $a \in \mathbb{N}$ функцията $\lambda\bar{x}.U(a, \bar{x}) \in \mathcal{K}$.

От условие 1) се вижда, че за да притежава универсална функция, класът $\mathcal{K}$ трябва да е най-много изброим. Впрочем, за всеки такъв клас има функция U, за която са изпълнени горните изисквания 1) и 2). За да я конструираме, тръгваме от произволно изброяване, евентуално с повторения, на функциите от класа $\mathcal{K}$:

$$f_0,\ f_1,\ \ldots\ , f_a,\ \ldots$$

Сега да положим

$$U(a, \bar{x}) \overset{\text{деф}}{\simeq} f_a(\bar{x})$$

за всяко $a \in \mathbb{N}, \bar{x} \in \mathbb{N}$. Тогава условията 1) и 2) следват директно от дефиницията на U.

След малко в няколко задачи ще илюстрираме как се построява универсална функция за някои прости класове от функции. Такива ще са класът на всички полиноми с коефициенти от $\mathbb{N}$, класът на всички крайни едноместни функции и др. Дали класът $\mathcal{C}_n$ на всички n-местни изчислими функции притежава универсална функция? За щастие — да, само че това вече няма да бъде тема на една задача, а на цял раздел (3.2). Фактът, че изчислимите функции имат УФ обикновено се нарича *Теорема за универсалната функция*.

2.3.2 Диагонален метод на Кантор

За задачите от следващия раздел ще ни е нужна конструкция, известна като *диагонален метод на Кантор*. Той се използва за конструиране на безкраен обект (функция, множество, реално число и пр.), който не принадлежи на дадено изброимо множество от подобни обекти.

Да разгледаме като пример случая, в който това множество се състои от едноместни тотални функции. Щом е изброимо, можем да подредим елементите му в редица:

$$f_0, f_1, \ \ldots\ , f_n, \ \ldots \tag{2.5}$$

Ще построим нова функция $d(x)$, която ще се различава от всяка функция f_n от тази редица точно в т. n. Функцията $d(x)$ е свързана с *диагонала* на една безкрайна таблица, в която можем да разположим стойностите на функциите от редицата (2.5).

	0	1	…	a	…
f_0	$f_0(0)$	$f_0(1)$	…	$f_0(a)$	…
f_1	$f_1(0)$	$f_1(1)$	…	$f_1(a)$	…
⋮	…	…	…	…	…
f_a	$f_a(0)$	$f_a(1)$	…	$f_a(a)$	…
⋮	…	…	…	…	…

Нека вземем например

$$d(x) \stackrel{\text{деф}}{=} f_x(x) + 1.$$

Ако допуснем, че $d = f_n$ за някое n, то би трябвало $d(x) = f_n(x)$ за всяко x. Но при $x = n$ това равенство очевидно се нарушава, защото съгласно избора на d:

$$d(n) = f_n(n) + 1.$$

2.3.3 Задачи

В следващите задачи ще покажем, че два интересни класа от функции имат универсална функция. Техните решения ще са сравнително лесни,

защото функциите от тези два класа могат да бъдат *кодирани*, а не само *номерирани*, както беше при изчислимите функции.

Като използваме *диагоналния метод на Кантор*, ще докажем също, че никоя универсална функция за тези два класа не принадлежи на тези класове (за разлика от УФ за изчислимите функции, която *е* изчислима функция).

Задача 2.1. Докажете, че

а) Класът $\mathcal{P}$ на всички полиноми на една променлива с коефициенти от $\mathbb{N}$ има универсална функция.

б) Никоя универсална функция за класа $\mathcal{P}$ не е полином.

Решение. а) Всеки полином от класа $\mathcal{P}$ изглежда така:

$$p(x) = a_0x^n + a_1x^{n-1} + \ \cdots + \ a_n$$

и следователно може да бъде кодиран с редицата от своите коефициенти $(a_0, \ldots a_n)$.

Да наречем *код на полинома* $a_0x^n + a_1x^{n-1} + \ \cdots + \ a_n$ числото $a = \tau(\langle a_0, \ldots a_n\rangle)$. Ясно е, че изображението $\tau\colon \mathcal{P} \longrightarrow \mathbb{N}$ наистина е кодиране, т.е. всяко естествено число a е код на единствен полином. Да означим

$$p_a \overset{\text{деф}}{=} \text{полинома с код } a. \tag{2.6}$$

Тогава редицата $p_0, p_1, \ldots, p_a, \ldots$ включва всички полиноми от класа $\mathcal{P}$ и само тях. Да дефинираме функцията U като:

$$U(a, x) \overset{\text{деф}}{=} p_a(x)$$

за всяко $a, x \in \mathbb{N}$. Условията 1) и 2) от *Определение* 3.1 за УФ се проверяват непосредствено, а фактът, че U е изчислима, следва от представянето

$$\begin{aligned} U(a,x) \ &= \ mem(a,0).x^{lh(a)} + mem(a,1).x^{lh(a)-1} + \cdots + mem(a, lh(a)) \\ &= \sum_{i=0}^{lh(a)} mem(a,i).x^{lh(a)-i}. \end{aligned}$$

б) Сега да допуснем, че класът $\mathcal{P}$ притежава някаква универсална функция $U(a, x)$, която е полином на променливите си a и x. Разглеждаме диагоналната функция

$$d(x) = U(x, x) + 1,$$

която очевидно също е полином. От условие 1) от дефиницията на УФ, за тази функция d ще съществува естествено число a, такова че

$$d(x) \ = \ U(a, x)$$

за всяко x. Но тогава при $x = a$ ще имаме $d(a) = U(a, a)$, което противоречи на избора на d, според който $d(a) = U(a, a) + 1$.

Забележка. Да обърнем внимание, че за полученото по-горе противоречие

$$d(a) = U(a, a) \quad \text{и} \quad d(a) = U(a, a) + 1$$

от изключителна важност беше фактът, че функцията U е тотална, което се вижда от определението ѝ (2.6). Ясно е, че същото разсъждение няма как да мине за универсалната функция Φ_1, защото тя е частична. □

Какво ще стане, ако от трите базисни операции — суперпозиция, примитивна рекурсия и минимизация — оставим само първата? Какъв клас от функции ще се получи? Оказва се, че функциите от този клас са съвсем прости — те са или константи, или са от вида $\lambda\bar{x}.x_i + b$. Да се убедим:

Задача 2.2. Нека

$$\mathcal{SUP} = \{f \mid f \text{ се получава от базисните функции с краен брой суперпозиции}\}.$$

Докажете, че една n-местна функция f принадлежи на класа $\mathcal{SUP}$ тогава и само тогава, когато f има вида

$$f(x_1, \ldots, x_n) = ax_i + b$$

за някои $1 \leq i \leq n$, $a \in \{0, 1\}$ и $b \in \mathbb{N}$.

Доказателство. Нека $f \in \mathcal{SUP}$. Ако f е базисна функция, твърдението се проверява непосредствено. Нека сега

$$f = g(h_1, \ldots, h_k).$$

От индуктивната хипотеза за g съществуват константи i, A и B, такива че $g(x_1, \ldots x_k) = Ax_i + B$, а от и.х. за h_i съществуват константи j, a и b, такива че $h_i(x_1, \ldots, x_n) = ax_j + b$. Тогава

$$f(x_1, \ldots, x_n) = A.h_i(x_1, \ldots, x_n) + B = A(ax_j + b) + B = cx_j + d$$

за някои константи c и d. Понеже $c = A.a$, то c е 0 или 1, защото по и.х. A и a са от същия вид.

Сега обратно, нека $f(x_1, \ldots, x_n) = ax_i + b$. Имаме два случая за a:

1 сл. $a = 0$, т.е. $f(\bar{x}) = b$ за всяко $\bar{x}$. Тогава f можем да представим по следния начин:

$$f(\bar{x}) = \underbrace{\mathcal{S}(\ldots\mathcal{S}}_{b \text{ пъти}}(\mathcal{O}(I_1^n(\bar{x}))) \ \ldots),$$

и значи $f \in \mathcal{SUP}$.

2 сл. $a = 1$. Тогава $f(x_1, \ldots, x_n) = x_i + b$. Тогава за f имаме представянето

$$f(\bar{x}) = \underbrace{\mathcal{S}(\ldots \mathcal{S}}_{b \text{ пъти}}(I_i^n(\bar{x}))\ \ldots)$$

и следователно f отново е от $\mathcal{SUP}$. □

Задача 2.3. Нека $\mathcal{SUP}_n = \mathcal{SUP} \cap \mathcal{F}_n$, т.е. $\mathcal{SUP}_n$ е класът на всички n-*местни* функции, които могат да се получат от базисните с операцията суперпозиция.

а) Да фиксираме $n \geq 1$. Докажете, че класът $\mathcal{SUP}_n$ има универсална функция.

б) Докажете, че никоя универсална функция за този клас не принадлежи на $\mathcal{SUP}$.

Решение. а) Една възможна дефиниция на универсална функция U за $\mathcal{SUP}_n$ е следната:

$$U(e, x_1, \ldots, x_n) = \begin{cases} sg((e)_1).x_1 + (e)_2, & \text{ако } (e)_0 \leq 1 \\ \ldots\ldots\ldots\ldots & \\ sg((e)_1).x_n + (e)_2, & \text{ако } (e)_0 \geq n. \end{cases}$$

Тази функция очевидно е изчислима (дори е примитивно рекурсивна). За всяко фиксирано e функцията $\lambda x_1, \ldots, x_n.U(e, x_1, \ldots, x_n)$ има вида $\underbrace{sg((e)_1)}_{0 \text{ или } 1}.x_i + (e)_2$, където индексът i се определя от $(e)_0$. Сега прилагаме *Задача* 2.2 и получаваме, че $\lambda\bar{x}.U(e, \bar{x}) \in \mathcal{SUP}_n$.

Обратно, нека f е произволна функция от $\mathcal{SUP}_n$. Тогава

$$f(x_1, \ldots, x_n) = ax_i + b$$

за някои $1 \leq i \leq n$, $a \in \{0, 1\}$ и $b \in \mathbb{N}$. Да вземем, например,

$$e = 2^i.3^a.5^b.$$

Тогава $f(\bar{x}) \overset{\text{деф}}{=} ax_i + b = (e)_1.x_i + (e)_2 = sg((e)_1).x_i + (e)_2$. Понеже $(e)_0 = i$, то и $U(e, \bar{x}) = sg((e)_1).x_i + (e)_2$, което означава, че $f = \lambda\bar{x}.U(e, \bar{x})$.

б) Сега нека U е някаква универсална функция за класа $\mathcal{SUP}_n$ (може да не е непременно тази, която построихме в подточка а)). Ако допуснем, че $U \in \mathcal{SUP}$, тогава функцията

$$f(x_1, \ldots, x_n) \overset{\text{деф}}{=} U(x_1, x_1, \ldots, x_n) + 1 \tag{2.7}$$

ще бъде в класа $\mathcal{SUP}_n$, защото $f = \mathcal{S} \circ U(I_1^n, \ldots, I_n^n)$. Тогава от условие 1) от дефиницията 3.1 за УФ ще съществува $e \in \mathbb{N}$, такова че $f = \lambda \bar{x}.U(e, \bar{x})$. Оттук при $(x_1, \ldots, x_n) = (e, \ldots, e)$ ще имаме, че

$$f(e, \ldots, e) = U(e, e, \ldots, e).$$

От друга страна, съгласно избора (2.7) на f, $f(e, \ldots, e) = U(e, e, \ldots, e)+1$. Противоречие. □

Задача 2.4. (Задача за ЕК) Нека $\Theta = \{\theta \mid \theta \in \mathcal{F}_1 \ \& \ \theta \text{ е крайна функция}\}$. Докажете, че:

1) Класът Θ има универсална функция.

2) Никоя универсална функция за Θ не е крайна функция.