Съдържание

Глава 1

Частично рекурсивни функции

1.1 Специфично за частичните функции

Ще разглеждаме функции в множеството на естествените числа

$$\mathbb{N} = \{0, 1, \dots\},$$

които са *частични*. Това означава, че в някои точки те могат да не са дефинирани, т.е. да нямат стойност. Такива ще са изчислимите функции, които основно ще изучаваме в този курс. Това ще са функциите, които се пресмятат – най-общо казано – с някаква програма. И тъй като програмите, както е известно, невинаги завършват, то значи и функциите, които те пресмятат, в общия случай трябва да са частични.

Ще пишем $f : \mathbb{N}^n \multimap \mathbb{N}$, за да означим, че f е частична функция на n аргумента в $\mathbb{N}$. Съвкупността от всички такива функции ще отбелязваме с $\mathcal{F}_n$, с други думи

$$\mathcal{F}_n = \{f \mid f : \mathbb{N}^n \multimap \mathbb{N}\}.$$

По-надолу ще предполагаме, че f е произволна n-местна частична функция. Ако тя е дефинирана в точката $(x_1, \dots, x_n)$, това ще отбелязваме така:

$$!f(x_1, \dots, x_n),$$

а ако не е дефинирана — ще пишем съответно $\neg!f(x_1, \dots, x_n)$.

Множеството от всички точки, в които f е дефинирана, ще наричаме *дефиниционно множество (домейн)* на f и ще означаваме с $Dom(f)$, или формално:

$$Dom(f) = \{(x_1, \dots, x_n) \mid !f(x_1, \dots, x_n)\}.$$

Ако $Dom(f) = \mathbb{N}^n$, ще казваме, че f е *тотална (навсякъде дефинирана)*. Разбира се, всяка тотална функция може да се разглежда и като частична, т.е. тя също принадлежи на $\mathcal{F}_n = \{f \mid f : \mathbb{N}^n \multimap \mathbb{N}\}$. Когато казваме *функция*, в общия случай ще имаме предвид частична функция. Ако става въпрос за тотална функция, това ще бъде отбелязвано експлицитно, ако не се подразбира от контекста.

По-нататък n-торките $(x_1, \ldots, x_n)$ ще съкращаваме до $\bar{x}$, когато това не води до някаква неяснота.

1.1.1 Условно равенство

Когато пишем равенство между изрази, в които участват частични функции, е необходимо да уточним какво ще разбираме в случаите, когато някоя от двете страни (или и двете едновременно) не са дефинирани. За тази цел ще използваме нова релация, която ще наричаме *условно равенство* и ще означаваме с $\simeq$.

Определение 1.1. Нека $\alpha(\bar{x})$ и $\beta(\bar{x})$ са изрази, в които участват частични функции. Тогава

$$\alpha(\bar{x}) \simeq \beta(\bar{x}) \quad \overset{\text{деф}}{\iff} \quad !\alpha(\bar{x}) \;\&\; !\beta(\bar{x}) \;\&\; \alpha(\bar{x}) = \beta(\bar{x}) \\ \vee \;\neg!\alpha(\bar{x}) \;\&\; \neg!\beta(\bar{x}).$$

С други думи, условното равенство има стойност *истина* или когато и двете му страни са дефинирани и имат една и съща стойност, или когато и двете му страни не са дефинирани. В останалите случаи то е *лъжа*. В частност, $f(\bar{x}) \simeq y$ ще е вярно точно когато f е дефинирана в $\bar{x}$ и нейната стойност е y.

Графиката G_f на частичната функция f въвеждаме по обичайния начин:

$$G_f = \{(x_1, \ldots, x_n, y) \mid f(x_1, \ldots, x_n) \simeq y\}.$$

Определение 1.2. За две n-местни частични функции f и g ще казваме, че са *равни* (и ще пишем $f = g$), ако $f(\bar{x}) \simeq g(\bar{x})$ за всяко $\bar{x} \in \mathbb{N}^n$.

Ясно е, че ако $f = g$, то $Dom(f) = Dom(g)$ и $f(\bar{x}) = g(\bar{x})$ за всяко $\bar{x} \in Dom(f)$. Равенството на две функции може да се разпише и така:

$$\begin{aligned} f = g &\iff \forall x_1 \ldots \forall x_n \; f(x_1, \ldots, x_n) \simeq g(x_1, \ldots, x_n) \\ &\iff \forall x_1 \ldots \forall x_n \forall y (f(x_1, \ldots, x_n) \simeq y \iff g(x_1, \ldots, x_n) \simeq y) \\ &\iff \forall x_1 \ldots \forall x_n \forall y ((x_1, \ldots, x_n, y) \in G_f \iff (x_1, \ldots, x_n, y) \in G_g) \\ &\iff G_f = G_g. \end{aligned}$$

Излезе (без да е изненадващо), че две частични функции са равни точно тогава, когато имат едни и същи графики.

1.1.2 Релацията включване

Сега ще въведем една релация между частични функции, която няма аналог при тоталните функции. Релацията е *включване* ($\subseteq$) и смисълът ѝ е, че ако $f \subseteq g$, то g "знае повече" от f, или g "носи повече информация" от f. Ето точното определение:

Определение 1.3. Нека $f, g \in \mathcal{F}_n$. Тогава

$$f \subseteq g \overset{\text{деф}}{\iff} \forall x_1 \ldots \forall x_n \forall y \; (f(x_1, \ldots, x_n) \simeq y \implies g(x_1, \ldots, x_n) \simeq y).$$

Ако $f \subseteq g$, ще казваме още, че f е *подфункция* на g или обратно — че g е *продължение* на f. Преразказано, една функция се продължава от друга, ако там, където първата е дефинирана (и значи има някаква стойност), там и втората е дефинирана и има същата стойност.

От определението се вижда, че

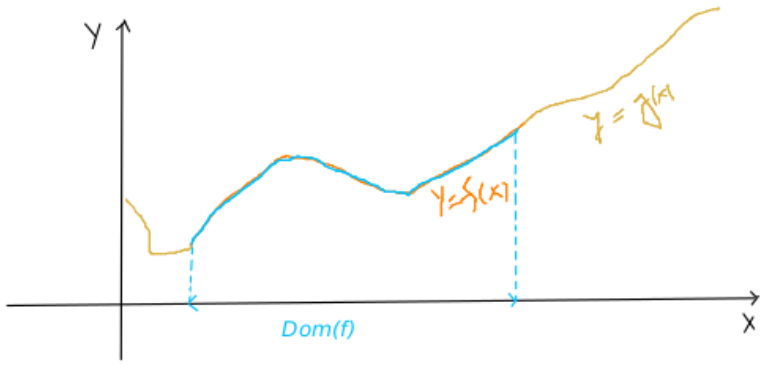
$$f \subseteq g \iff G_f \subseteq G_g,$$

което обяснява защо използваме теоретико-множествения символ $\subseteq$.

Да отбележим и още един очевиден факт, който ще използваме често:

$$f \subseteq g \implies Dom(f) \subseteq Dom(g).$$

Ето как изглеждат схематично графиките на f и g, такива че $f \subseteq g$:



Ако f е тотална и $f \subseteq g$, то очевидно $f = g$, т.е. върху тоталните функции релацията включване съвпада с релацията равенство.

Когато задаваме някаква функция f и искаме да кажем, че в т. $(x_1, \ldots, x_n)$ тя няма стойност, това ще записваме и така: $f(x_1, \ldots, x_n) \simeq \neg!$.

Ето един пример за две функции f и g, такива че f е подфункция на g:

Пример 1.1. Да дефинираме функциите f и g както следва:

$$f(x,y) \simeq \begin{cases} \lfloor \frac{x}{y} \rfloor, & \text{ако } y > 0 \\ \neg !, & \text{ако } y = 0, \end{cases}$$

$$g(x,y) = \begin{cases} \lfloor \frac{x}{y} \rfloor, & \text{ако } y > 0 \\ 0, & \text{ако } y = 0. \end{cases}$$

Ясно е, че в точките, в които е дефинирана, f има същата стойност като g, с други думи, $f \subseteq g$.

Релацията *строго включване* ($\subset$) се дефинира от $\subseteq$ по обичайния начин:

$$f \subset g \overset{\text{деф}}{\iff} f \subseteq g \ \& \ f \neq g.$$

За функциите f и g от *Пример* 1.1 от по-горе всъщност имаме $f \subset g$.

От наблюдението, че две функции са равни точно когато графиките им съвпадат, получаваме следната връзка между релациите $=$ и $\subseteq$:

$$\begin{aligned} f = g &\iff G_f = G_g \\ &\iff G_f \subseteq G_g \ \& \ G_g \subseteq G_f \\ &\iff f \subseteq g \ \& \ g \subseteq f. \end{aligned}$$

Излезе, че

$$f = g \iff f \subseteq g \ \& \ g \subseteq f.$$

От тази еквивалентност се вижда един начин да покажем, че две *частични* функции са равни — като проверим, че едната е подфункция на другата и обратно. Оказва се, че можем леко да отслабим това условие, като заменим включването $g \subseteq f$ с по-слабото $Dom(g) \subseteq Dom(f)$. Тази дребна наглед корекция в бъдеще ще ни спестява писане. Но да се убедим първо, че можем да направим това:

Задача 1.1. Нека f и g са n-местни функции. Докажете, че $f = g$ тогава и само тогава, когато са изпълнени условията:

1) $f \subseteq g$;

2) $Dom(g) \subseteq Dom(f)$.

Решение. Ако $f = g$, то $f \subseteq g$ и $g \subseteq f$ и от последното, в частност, следва и включването между домейните $Dom(g) \subseteq Dom(f)$.

Обратно, нека са верни 1) и 2). Трябва да покажем, че $f \subseteq g$ и $g \subseteq f$. Първото включване е точно условието 1). За да покажем, че и $g \subseteq f$, да приемем, че за произволни $\bar{x}, y$ $g(\bar{x}) \simeq y$. Тогава $\bar{x} \in Dom(g)$, а оттук съгласно 2) ще имаме и $\bar{x} \in Dom(f)$, т.е. $f(\bar{x}) \simeq z$ за някое z.

Сега от условието 1) получаваме, че и $g(\bar{x}) \simeq z$, и значи $y = z$. И така, получихме, че за произволни $\bar{x}, y$:

$$g(\bar{x}) \simeq y \implies f(\bar{x}) \simeq y,$$

което по дефиницията на $\subseteq$ означава, че $g \subseteq f$. □

От еквивалентността

$$f = g \iff G_f = G_g$$

се вижда, че релацията включване между функции се изразява чрез включване между *множества*, за което знаем, че е частична наредба. Следователно и релацията "подфункция" е *частична наредба*. Да обърнем внимание, че тя също е *частична*, т.е. не всеки две функции от $\mathcal{F}_n$ са свързани чрез нея. Такива са например константните функции f_0 и f_1, които за всяко $\bar{x} \in \mathbb{N}^n$ връщат 0 и 1, съответно. Не се заблуждавайте: вярно е, че $\forall \bar{x}\ f_0(\bar{x}) \leq f_1(\bar{x})$, обаче не е вярно, че $f_0 \subseteq f_1$. ☺

Интуитивно, $f \subseteq g$ означава, че f е "по-малко информативна" от g. Тогава "най-малко информативна" ще е функцията, която не е дефинирана в нито една точка.

Всъщност има безброй много такива функции, в зависимост от броя на аргументите им. За фиксирано $n \geq 1$ с $\emptyset^{(n)}$ ще означаваме n-местната функция, която не е дефинирана за нито една n-торка $\bar{x} \in \mathbb{N}^n$ и тази функция ще наричаме *никъде недефинираната* (или *празната*) функция на n аргумента. Да отбележим, че за всяка $f \in \mathcal{F}_n$ е в сила включването

$$\emptyset^{(n)} \subseteq f,$$

с други думи, никъде недефинираната функция $\emptyset^{(n)}$ е най-малкият (относно $\subseteq$) елемент на $\mathcal{F}_n$.

1.2 Примитивно рекурсивни функции

Дефиницията на примитивно рекурсивните функции идейно прилича на дефиницията на регулярните множества — тръгваме от някакви начални прости обекти и ги затваряме относно някакви прости операции. В нашия случай началните обекти се наричат *изходни (базисни) примитивно рекурсивни функции*, които въвеждаме по-долу заедно с операциите, относно които ще ги затворим. Така ще определим *примитивно рекурсивните* и *частично рекурсивните* функции, въведени от Ербран, Гьодел и Клини.

1.2.1 Изходни (базисни) примитивно рекурсивни функции

Идеята е това да са възможно най-простите функции над естествените числа, чиято "изчислимост" не оставя съмнение в никого. Разбира се, тези функции трябва да са и достатъчно изразителни, за да можем, тръгвайки от тях, да получим всички възможни изчислими функции.

Определение 1.4. *Изходните (базисните) примитивно рекурсивни функции* са следните:

1) функцията $\mathcal{S}$ (от successor), която дава наследника на всяко $x \in \mathbb{N}$:
$$\mathcal{S}(x) = x + 1;$$

2) *едноместната константна функция* $\mathcal{O}$, която за всяко $x \in \mathbb{N}$ връща 0:
$$\mathcal{O}(x) = 0;$$

3) *проектиращите функции* I_k^n, $n = 1, 2, \ldots$ и $1 \leq k \leq n$, дефинирани като:
$$I_k^n(x_1, \ldots, x_n) = x_k$$
за всяка n-торка $(x_1, \ldots, x_n)$ от $\mathbb{N}^n$.

В частност, при $k = n = 1$ получаваме $I_1^1(x) = x$ за всяко $x \in \mathbb{N}$, т.е. изходната функция I_1^1 е *идентитетът* в $\mathbb{N}$. Предназначението на проектиращите функции е по-скоро техническо.

1.2.2 Изходни (базисни) операции

Както при избора на изходните функции, и тук целта е тези начални операции да са възможно най-прости и едновременно с това — достатъчно мощни, за да може чрез тях да се зададат всички изчислими функции.

Изходните операции, които ще въведем, следвайки дефиницията на Ербран–Гьодел–Клини, са три — *суперпозиция, примитивна рекурсия и минимизация*. В този раздел ще определим първите две от тях, а в раздел 1.3 — третата.

Определение 1.5. Нека f е произволна n-местна функция, а $g_1, \ldots, g_n$ са n на брой функции, всички на k аргумента. *Суперпозицията* на тези функции е k-местната функция h, която се дефинира по следния начин:

$$h(x_1, \ldots, x_k) \simeq y \overset{\text{деф}}{\iff} \exists z_1 \ldots \exists z_n \, (g_1(x_1, \ldots, x_k) \simeq z_1 \;\&\; \ldots \;\&\; g_n(x_1, \ldots, x_k) \simeq z_n \;\&\; f(z_1, \ldots, z_n) \simeq y) \qquad (1.1)$$

за всяко $(x_1, \ldots, x_k)$ от $\mathbb{N}^k$

Суперпозицията на f и $g_1, \ldots, g_n$ ще означаваме с

$$f(g_1, \ldots, g_n).$$

При $n = 1$ функцията $f(g)$ ще наричаме *композиция* на f и g и ще бележим с обичайното $f \circ g$.

От еквивалентността (1.1) следва в частност, че

$$!f(g_1, \ldots, g_n)(\bar{x}) \iff !g_1(\bar{x}) \ \& \ \ldots \& \ !g_n(\bar{x}) \ \& \ !f(g_1(\bar{x}), \ldots, g_n(\bar{x})).$$

Ако приемем, че така разбираме дефинираността на $f(g_1, \ldots, g_n)(\bar{x})$, определението за суперпозиция можем да запишем и по-кратко като:

$$f(g_1, \ldots, g_n)(\bar{x}) \overset{\text{деф}}{\simeq} f(g_1(\bar{x}), \ldots, g_n(\bar{x})).$$

Втората изходна операция е *примитивна рекурсия*. Най-общо, една функция f се задава с *рекурсия*, ако се определя "чрез себе си", което можем да си представим схематично така:

$$f(x) \ \simeq \ \ldots \ f, \ x \ \ldots.$$

Най-простата схема за дефиниция по рекурсия на *едноместна* функция f е следната: по дадени константа $c \in \mathbb{N}$ и $g \in \mathcal{F}_2$, функцията f определяме посредством равенствата:

$$\left| \begin{array}{l} f(0) = c \\ f(x+1) \simeq g(x, f(x)). \end{array} \right.$$

Горната рекурентна връзка обикновено се нарича *проста схема на примитивна рекурсия*. За f ще казваме, че е получена с *примитивна рекурсия* от c и g. Тук константата c задава *дъното* на рекурсията, а функцията g ни казва как да пресметнем $f(x+1)$, ако знаем $f(x)$.

Буквалният пример за дефиниция чрез тази схема е рекурсивната дефиниция на функцията $f(x) = x!$. За нея имаме

$$\left| \begin{array}{l} f(0) = 1 \\ f(x+1) = (x+1).f(x). \end{array} \right.$$

Тук константата $c = 1$, а $g(x, y) = (x+1)y$.

Да се опитаме да обобщим ситуацията за функция на повече променливи. Как би изглеждала възможно най-простата схема на рекурсия, ако f е на два аргумента примерно? Как да дефинираме рекурсивно $f(x, y)$ — с рекурсия по x, по y или и по двата аргумента?

Известно е, че рекурсия *и по двата аргумента* може да доведе до функции-чудовища. Пример за това е *функцията на Акерман*, която се задава със следната двойна рекурсия:

$$\left|\begin{array}{l} f(0,y) \simeq y+1 \\ f(x+1,0) \simeq f(x,1) \\ f(x+1,y+1) \simeq f(x,f(x+1,y)). \end{array}\right.$$

Тази функция расте с шеметна скорост — например $f(4,2)$ е число с 19 729 цифри в десетичния си запис! (Но дали има *единствена* функция, която удовлетворява тези рекурсивни изисквания? Погледнете решението на *Задача* 1.18, ако се съмнявате в това.)

Малко офтопик: ако си мислите, че горната рекурсивна схема *поначало* е сложна, защото рекурсията е двойна — ами не винаги е така. Да вземем ето тази рекурсивна схема, която се различава с всичко на всичко "една единица" от дефиницията на Акерман (в дъното на рекурсията):

$$\left|\begin{array}{l} g(0,y) \simeq y \\ g(x+1,0) \simeq g(x,1) \\ g(x+1,y+1) \simeq g(x,g(x+1,y)). \end{array}\right.$$

Изненадващо, тази рекурсивна схема определя функция, която е почти константа (погледнете решението на *Задача*

Всъщност подходящото обобщение на простата схема на примитивна рекурсия е рекурсия само *по един* от аргументите на определяемата функция f. Ние ще изберем това да бъде последният аргумент.

Определение 1.6. Нека $g(x_1,\ldots,x_n)$ и $h(x_1,\ldots,x_n,y,z)$ са фиксирани частични функции, съответно на n и $n+2$ аргумента. Казваме, че f се получава с *примитивна рекурсия* от g и h, ако за всички $\bar{x},y$ са изпълнени равенствата:

$$\begin{aligned} &f(x_1,\ldots,x_n,0) \simeq g(x_1,\ldots,x_n) \\ &f(x_1,\ldots,x_n,y+1) \simeq h(x_1,\ldots,x_n,y,f(x_1,\ldots,x_n,y)). \end{aligned} \tag{1.2}$$

За функцията f ще казваме още, че *е* примитивна рекурсия на g и h.

Равенствата 1.6 определят *общата схема* на примитивната рекурсия. При $n=0$ от нея получаваме простата схема, която въведохме по-горе.

Да се убедим най-напред, че *Определение* 1.6 наистина определя точно една функция.

Твърдение 1.1. Нека $g \in \mathcal{F}_n$, а $h \in \mathcal{F}_{n+2}$. Съществува единствена функция f, която е примитивна рекурсия на g и h.

Доказателство. Ще покажем, че за всяко $\bar{x} \in \mathbb{N}^n$ и всяко $y \in \mathbb{N}$, $f(\bar{x},y)$ е *еднозначно определена*. Последното означава, че или $f(\bar{x},y)$ няма стойност, или има стойност и тази стойност е единствена.

За целта да фиксираме $\bar{x} \in \mathbb{N}^n$. С индукция по y ще покажем, че $\forall y P(y)$, където P е следното свойство:

$$P(y) \quad \overset{\text{деф}}{\iff} \quad f(\bar{x}, y) \text{ е еднозначно определена.}$$

Базовият случай $P(0)$ е ясен, защото тогава $f(\bar{x}, 0) \simeq g(\bar{x})$.
Сега да допуснем, че за някое y е в сила $P(y)$, т.е. $f(\bar{x}, y)$ е еднозначно определена. Но тогава същото ще е вярно и за $f(\bar{x}, y+1)$, тъй като в този случай

$$f(\bar{x}, y+1) \overset{(1.2)}{\simeq} h(\bar{x}, y, f(\bar{x}, y)).$$

Така показахме и $P(y+1)$, с което приключва проверката на $\forall y P(y)$, а оттук и доказателството на твърдението. □

С разсъждение, подобно на горното, лесно се убеждаваме, че операцията примитивна рекурсия запазва тоталността, т.е. приложена върху тотални функции, тя връща отново тотална функция.

Твърдение 1.2. Нека $g \in \mathcal{F}_n$ и $h \in \mathcal{F}_{n+2}$ са тотални функции. Тогава и тяхната примитивна рекурсия е тотална функция.

Доказателство. Да означим с f примитивната рекурсия на g и h. Сега трябва да видим, че за всяко $\bar{x} \in \mathbb{N}^n$ и всяко $y \in \mathbb{N}$, $!f(\bar{x}, y)$. Както в предишното доказателство, фиксираме $\bar{x} \in \mathbb{N}^n$ и с рутинна индукция по y показваме, че $\forall y Q(y)$, където Q се определя така:

$$Q(y) \quad \iff \quad !f(\bar{x}, y).$$

□

Задача 1.2. Намерете явния вид на функцията f, която се определя с примитивно рекурсивната схема

$$\begin{cases} f(x, 0) = g(x) \\ f(x, y+1) = h(x, y, f(x, y)) \end{cases}$$

от функциите g и h, където:

а) $g(x) = 1$ и $h(x, y, x) = x.z$;

б) $g(x) = x$ и $h(x, y, x) = z^x$;

в) $g(x) = 1$ и $h(x, y, x) = x^z$.

Отговор: а) $f(x, y) = x^y$, б) $f(x, y) = x^{x^y}$, в) $f(x, y) = \underbrace{x^{\cdot^{\cdot^{x}}}}_{y \text{ пъти}}$.

1.2.3 Примитивно рекурсивни функции

Определение 1.7. Казваме, че една функция е *примитивно рекурсивна (пр. р.)*, ако тя може да се получи от изходните примитивно рекурсивни функции чрез краен брой прилагания на операциите суперпозиция и примитивна рекурсия.

Да отбележим, че горната дефиниция всъщност е *индуктивна*. Тя може да бъде изказана и по следния начин:

1) Всяка от изходните функции $\mathcal{S}, \mathcal{O}$ и I_k^n е примитивно рекурсивна.

2) Ако f и $g_1, \dots g_n$ са примитивно рекурсивни, то и тяхната суперпозиция $f(g_1, \dots, g_n)$ е примитивно рекурсивна.

3) Ако f и g са примитивно рекурсивни, а h е получена с примитивна рекурсия от тях, то и h е примитивно рекурсивна.

Индуктивният характер на тази дефиниция означава, че всяко свойство, отнасящо се до примитивно рекурсивните функции, ще трябва да се доказва с индукция, следваща пунктовете на дефиницията. Такъв тип индукция се нарича *структурна индукция*. За илюстрация да докажем следващото твърдение.

Твърдение 1.3. Всяка примитивно рекурсивна функция е тотална.

Доказателство. Нека h е примитивно рекурсивна. Ако тя е получена по т. 1) от горната дефиниция, то h е някоя от изходните функции $\mathcal{S}, \mathcal{O}$ и I_k^n, които са тотални. Нека сега h е получена по т. 2) от дефиницията. Това означава, че $h = f(g_1, \dots, g_n)$, като за f и $g_1, \dots, g_n$ индуктивната хипотеза е вярна, т.е. те са тотални функции. Но тогава и h ще е такава, защото суперпозицията очевидно запазва тоталността.

По подобен начин разсъждаваме ако h е получена по последния пункт 3) от дефиницията, като се възползваме от току-що доказаното *Твърдение* 1.2. □

По-нататък ще се убедим, че макар и примитивна по име, с такава рекурсия могат да се дефинират супербързорастящи функции — примерно експоненти (двойни, тройни и всякакви). Всъщност всяка тотална изчислима функция в естествените числа, за която се досетите, със сигурност ще е примитивно рекурсивна. Както ще видим по-нататък в курса, ще се изискват доста знания, за да конструираме тотална изчислима функция, която да не е примитивно рекурсивна.

1.3 Частично рекурсивни функции

Сега се насочваме към последната изходна операция — *минимизация*, която участва в дефиницията на частично рекурсивните функции.

Определение 1.8. Нека $f : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ е произволна частична функция. Казваме, че n-местната функция g се получава с *минимизация* (или с *μ-операция*) от f, и пишем

$$g(x_1, \ldots, x_n) \simeq \mu y[f(x_1, \ldots, x_n, y) \simeq 0],$$

ако за g е изпълнено:

$$g(x_1, \ldots, x_n) \simeq y \iff f(x_1, \ldots, x_n, y) \simeq 0 \ \& \ \forall z_{z<y}\ f(x_1, \ldots, x_n, z) > 0$$

за всички естествени $x_1, \ldots, x_n$ и y.

Да обърнем внимание на следната особеност в горната дефиниция: ако $g(\bar{x}) \simeq y$, то y е не просто първото естествено число, за което $f(\bar{x}, y) \simeq 0$; за него трябва да е вярно още, че $f(\bar{x}, z) > 0$ за всяко $z < y$. С други думи, ако $g(\bar{x}) \simeq y$, то за всички $z < y$, $f(\bar{x}, z)$ *има стойност* и тя е различна от 0.

Сигурно се питате защо да не вземем минимизация, която просто връща първото естествено y, такова че $f(\bar{x}, y) \simeq 0$, иначе казано, защо да не вземем следната μ-операция:

$$\mu y[f(\bar{x}, y) \simeq 0] \overset{\text{деф}}{\simeq} min\{y \mid f(\bar{x}, y) \simeq 0\}.$$

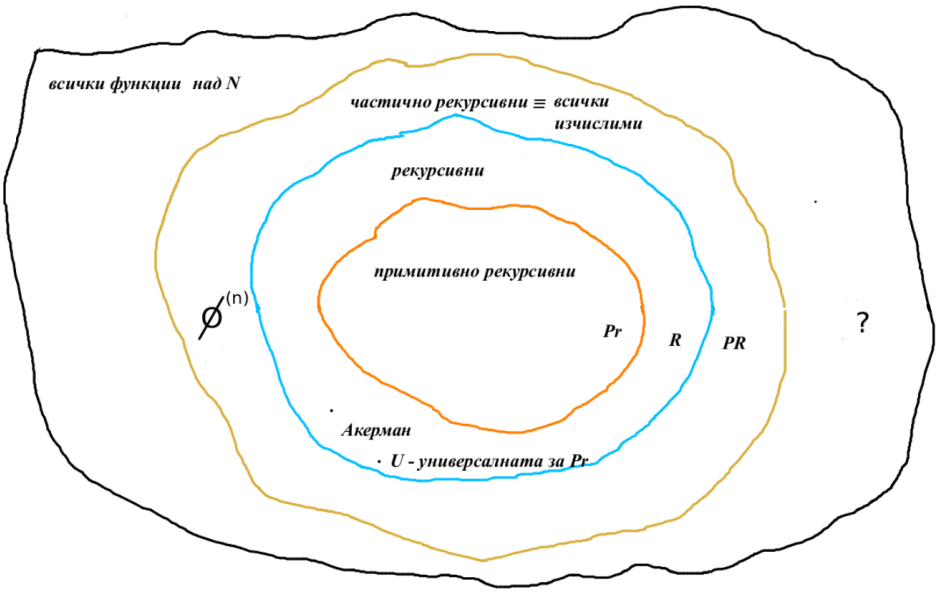
По-нататък в курса ще покажем, че при такава минимизация ще съществуват функции, които са изчислими (с някаква програма), докато тяхната минимизация вече не е изчислима с никаква програма. Това означава, че тази операция извежда извън класа на изчислимите функции, а това е последното нещо, което бихме искали да имаме за една базисна операция.

Определение 1.9. Казваме, че една функция е *частично рекурсивна (ч. р.)*, ако тя може да се получи от изходните примитивно рекурсивни функции чрез краен брой прилагания на операциите суперпозиция, примитивна рекурсия и минимизация.

Определение 1.10. Казваме, че една функция е *рекурсивна*, ако тя е частично рекурсивна и тотална.

Ясно е, че всички примитивно рекурсивни функции са и рекурсивни, защото те са:
– частично рекурсивни (в частност);
– тотални, съгласно *Твърдение* 1.3.

Ето как изглеждат на картинка класовете от функции, които въведохме, като в нея под *изчислима* функция засега ще разбираме функция, изчислима с *някаква* програма. В следващата глава ще дадем строга дефиниция на това понятие и ще докажем, че изчислимите функции съвпадат с частично рекурсивните.



В тази диаграма всички включвания са строги. Примери за функции, които са рекурсивни, но не са примитивно рекурсивни са, да кажем, функцията на Акерман, както и универсалната функция за всички примитивно рекурсивни функции (нали не очаквате в този момент да формулираме и докажем такова твърдение? ☺). Съвсем лесно за доказване е, обаче, че частично рекурсивните функции се включват строго в рекурсивните, просто защото те могат да са частични. Тривиални примери са празната функция $\emptyset^{(n)}$ или да кажем функцията f от *Пример* 1.1 (ще го докажем следващия час).

Разбира се, и най-външното включване е строго, което се вижда найлесно с мощностни съображения. За целта първо съобразяваме, че всички частично рекурсивни функции са изброимо много, а после използваме, че множеството от всички функции над $\mathbb{N}$ е с мощността на континуума (както е добре известно).

Ето и едно директно доказателство на този факт, което се базира на *диагоналния метод на Кантор.*

Задача 1.3. Докажете, че съществуват функции, които не са частично рекурсивни (и следователно не могат да се пресметнат с никаква програма).

Решение. Ще конструираме едноместна функция, която не е частично рекурсивна.

Както вече отбелязахме, всички частично рекурсивни функции са изброимо много, а оттук и *едноместните* ч.р.ф. също ще са изброимо много. Да ги подредим в редичка:

$$f_0, f_1, \ \ldots \ f_n, \ \ldots$$

Ще конструираме функция $d(x)$ — *диагонална функция*, такава че

$$d \neq f_n$$

за всяко n, и следователно d не може да е частично рекурсивна.

Условието $d \neq f_n$ означава, че $d(x) \not\simeq f_n(x)$ за поне едно x. Ние ще осигурим това различие за $x = n$, т.е. функцията d ще се различава от f_n в точката n.

За целта да вземем например

$$d(x) \simeq \begin{cases} \neg !, & \text{ако } !f_x(x) \\ 0, & \text{ако } \neg !f_x(x). \end{cases}$$

Да допуснем, че d е частично рекурсивна. Тогава $d = f_n$ за някое n и значи

$$d(x) \simeq f_n(x)$$

за всяко x. Но при $x = n$ имаме проблем, защото от една страна, би трябвало

$$d(n) \simeq f_n(n),$$

а от друга — функцията d избрахме точно с цел това да не се случва. $\square$

1.4 Примитивна рекурсивност на някои функции

В този раздел ще докажем примитивната рекурсивност на една дълга редица от функции в естествените числа. Фактът, че те са примитивно рекурсивни, ще е важен за това, което следва.

Ще започнем с едно спомагателно твърдение, което нататък ще използваме систематично. Ще го формулираме за трите типа функции, които въведохме, макар че основно ще го използваме за примитивно рекурсивните.

Твърдение 1.4. Нека $f(x_1, \ldots, x_k)$ е произволна примитивно рекурсивна/частично рекурсивна/рекурсивна функция. Нека още $i_1, \ldots, i_k$ са някакви числа между 1 и n (допускаме и повтарящи се). Да дефинираме n-местната функция g по следния начин:

$$g(x_1, \ldots, x_n) \simeq f(x_{i_1}, \ldots, x_{i_k})$$

за всяко $x_1, \ldots, x_n \in \mathbb{N}^n$. Тогава g също е примитивно рекурсивна/частично рекурсивна/рекурсивна.

Доказателство. Функцията g можем да препишем и така:

$$g(x_1, \ldots, x_n) \overset{\text{деф}}{\simeq} f(x_{i_1}, \ldots, x_{i_k}) \simeq f(I^n_{i_1}(x_1, \ldots, x_n), \ldots, I^n_{i_k}(x_1, \ldots, x_n)),$$

откъдето се вижда, че $g = f(I^n_{i_1}, \ldots, I^n_{i_k})$. Тъй като $I^n_{i_1}, \ldots, I^n_{i_k}$ са изходни пр.р. функции, то ясно е, че ако f е примитивно (частично) рекурсивна, то и g ще е такава, а ако f е рекурсивна, то и g ще е рекурсивна, защото изходните функции са тотални. □

Това твърдение ще използваме в ситуации като следните:

- $g(x_1, x_2) = f(x_1)$ — въвеждане на фиктивна променлива (тук $i_1 = 1$);
- $g(x_1, x_2) = f(x_2, x_1)$ — разместване на променливи (тук $i_1 = 2, i_2 = 1$);
- $g(x_1) = f(x_1, x_1)$ — удвояване на променлива (тук $i_1 = i_2 = 1$).

С C^n_a ще означаваме n-местната константна функция, която връща винаги a:

$$C^n_a(x_1, \ldots, x_n) \overset{\text{деф}}{=} a$$

за всяка $(x_1, \ldots, x_n) \in \mathbb{N}^n$. Всички константни функции са примитивно рекурсивни, както се вижда от задачата по-долу.

Задача 1.4. Докажете, че за всяко $a \in \mathbb{N}$ и $n \in \mathbb{N}^+$ константната функция C^n_a е примитивно рекурсивна.

Решение. За всяко $\bar{x} \in \mathbb{N}^n$ имаме $C^n_a(\bar{x}) = \underbrace{\mathcal{S}(\ldots\mathcal{S}}_{a \text{ пъти}}(\mathcal{O}(I^n_1(\bar{x})))\ldots)$ и значи C^n_a е следната композиция на изходни функции:

$$C^n_a = \underbrace{\mathcal{S} \circ \cdots \circ \mathcal{S}}_{a \text{ пъти}} \circ \mathcal{O} \circ I^n_1.$$

□

Предстои ни да видим как, тръгвайки от съвсем скромната функция "прибавяне на единица", можем да получим всички аритметични действия — събиране, изваждане, умножение, деление and much more ☺.

Твърдение 1.5. Следните функции са примитивно рекурсивни:

а) $f(x,y) = x + y$ (събиране).

Доказателство. Ще конструираме примитивно рекурсивна схема за f. Избираме си рекурсията да е по втория аргумент на f (тя е комутативна, тъй че няма значение кой от двата ще изберем). Базисният случай е ясен:

$$f(x,0) = x + 0 = x.$$

Сега трябва да намерим връзка между $f(x,y)$ и $f(x,y+1)$. В случая тя се вижда веднага:

$$f(x,y+1) = x + (y+1) = (x+y) + 1 = f(x,y) + 1 = \mathcal{S}(f(x,y)).$$

Получаваме общо

$$\begin{cases} f(x,0) = I_1^1(x) \\ f(x,y+1) = h(x,y,f(x,y)) \end{cases}$$

за $h(x,y,z) = \mathcal{S}(z)$. Тази функция е примитивно рекурсивна, защото се получава от изходната $\mathcal{S}$ с добавяне на две фиктивни променливи (тук използваме доказаното по-горе *Твърдение* 1.4). Финално, f се получава с примитивна рекурсия от пр.р. функции I_1^1 и h, и следователно f е примитивно рекурсивна. □

Въпрос: Как мислите, защо не използвахме по-краткото разсъждение, че събирането може да се представи като композиция на изходните функции $\mathcal{S}$ и I_1^2, което се вижда от следните равенства:

$$x + y = x + \underbrace{1 + \cdots + 1}_{y \text{ пъти}} = \underbrace{\mathcal{S}(\ldots\mathcal{S}}_{y \text{ пъти}}(x)\ldots) = \underbrace{\mathcal{S}(\ldots\mathcal{S}}_{y \text{ пъти}}(I_1^2(x,y))\ldots)?$$

б) $g(x,y) = x.y$ (умножение).

Доказателство. Отново ще използваме примитивна рекурсия. Имаме

$$\begin{aligned} x.0 &= 0 \\ x.(y+1) &= x.y + x, \end{aligned}$$

и значи схемата за g е такава:

$$\begin{cases} g(x,0) = 0 = \mathcal{O}(x) \\ g(x,y+1) = g(x,y) + x = h(x,y,g(x,y)). \end{cases}$$

Тук функцията $h(x,y,z) = x + z$ е примитивно рекурсивна, съгласно а) и *Твърдение* 1.4, следователно g ще е примитивно рекурсивна също. □

в) $h(x,y) = x^y$ (степенуване). Тук по дефиниция $0^0 = 1$).

Доказателство. Използваме, че

$$x^0 = 1$$
$$x^{y+1} = x^y.x,$$

откъдето

$$\begin{cases} h(x,0) = 1 = C_1^1(x) \\ h(x,y+1) = h(x,y).x = H(x,y,h(x,y)), \end{cases}$$

където $H(x,y,z) = x.z$ е примитивно рекурсивна, съгласно вече доказаното в б). □

Въпрос: Как ще изглежда функцията $u(x,y)$, която се получава от функцията h (степенуването) по начина, по който h беше получена от умножението? А ако повторите това разсъждение, като вместо h вземете новата функция u?

г) $p(x) = x \dot{-} 1$ — функцията *предшественик* (**p**redecessor), като по дефиниция:

$$x \dot{-} 1 = \begin{cases} x-1, & \text{ако } x > 0 \\ 0, & \text{ако } x = 0. \end{cases}$$

Доказателство. Можем да запишем

$$0 \dot{-} 1 = 0$$
$$(x+1) \dot{-} 1 = x,$$

или все едно

$$\begin{cases} p(0) = 0 \\ p(x+1) = x = I_1^2(x, p(x)). \end{cases}$$

□

д) $u(x,y) = x \dot{-} y$. Това е функцията *коригирана разлика*, която се дефинира като:

$$x \dot{-} y = \begin{cases} x-y, & \text{ако } x \geq y \\ 0, & \text{ако } x < y. \end{cases}$$

Доказателство. Целта ни е да изразим $x \dot{-} (y+1)$ чрез $x \dot{-} y$. Да се убедим, че

$$x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1.$$

Разглеждаме случаите от дефиницията на $x \dot{-} (y+1)$:
1 сл. $x \geq y+1$. Тогава $x - y \geq 1$ и следователно за израза вдясно ще имаме:

$$(x \dot{-} y) \dot{-} 1 \;=\; (x-y) \dot{-} 1 \;=\; x-y-1.$$

Понеже $x \geq y+1$, вляво ще имаме:

$$x \dot{-} (y+1) = \; x-(y+1) \;=\; x-y-1$$

и значи в този случай двете страни са равни.

2 сл. $x < y+1$. Тогава $x \leq y$ и лесно се вижда, че двете страни на горното равенство са 0. Сега преписваме тържествено:

$$\begin{cases} u(x,0) = x \\ u(x,y+1) = u(x,y) \dot{-} 1 = p(u(x,y)), \end{cases}$$

откъдето се вижда, че функцията u е примитивно рекурсивна. □

е) функцията $sg(x)$ (*сигнум*), където по определение:

$$sg(x) = \begin{cases} 0, & \text{ако } x = 0 \\ 1, & \text{ако } x > 0. \end{cases}$$

Доказателство. Да отбележим, че $sg(x)$ различава не знака на аргумента си, а дали той е равен или различен от 0. Имаме

$$\begin{cases} sg(0) = 0 \\ sg(x+1) = 1 = C_1^2(x, sg(x)). \end{cases}$$

□

ж) функцията $\bar{sg}(x)$:

$$\bar{sg}(x) \;=\; \begin{cases} 1, & \text{ако } x = 0 \\ 0, & \text{ако } x > 0. \end{cases}$$

Доказателство. Очевидно

$$\bar{sg}(x) = 1 \dot{-} sg(x).$$

□

з) $|x-y|$ — абсолютната стойност на разликата на x и y.

Доказателство. Следва от равенството

$$|x-y| = (x \dot{-} y) + (y \dot{-} x),$$

което се проверява директно с разглеждане на случаите $x \geq y$ и $x < y$. Следователно $|x-y|$ ще е примитивно рекурсивна като суперпозиция на функции, за които вече доказахме, че са примитивно рекурсивни.
□

и) $max(x, y)$.

Доказателство. Да се убедим, че

$$max(x, y) \;=\; x + (y \dot{-} x).$$

Наистина, ако $x \geq y$, стойността на израза вдясно ще е

$$x + (y \dot{-} x) = x + 0 = x,$$

което е точно $max(x, y)$ в този случай. Ако пък $x < y$, тогава

$$x + (y \dot{-} x) = x + (y - x) = y,$$

което отново е по-голямото от двете числа x и y. □

к) $min(x, y)$.

Доказателство. Следва от равенството

$$min(x, y) \;=\; x \dot{-} (x \dot{-} y),$$

което се проверява както по-горе. □

1.5 Примитивно рекурсивни предикати

Предикат на n аргумента в множеството $\mathbb{N}$ е тотално изображение

$$p\colon\ \mathbb{N}^n \longrightarrow \{\mathbf{t}, \mathbf{f}\}.$$

Вместо $p(\bar{x}) = \mathbf{t}$ обикновено ще пишем само $p(\bar{x})$ и ще казваме, че предикатът p е верен/е истина в $\bar{x}$, а вместо $p(\bar{x}) = \mathbf{f}$ ще пишем $\neg p(\bar{x})$ и ще казваме, че предикатът p е лъжа в $\bar{x}$.

Примери:

1) $p(x) \overset{\text{деф}}{\Longleftrightarrow} x$ е просто — *унарен* предикат;

2) $gt(x, y) \overset{\text{деф}}{\Longleftrightarrow} x > y$ — *бинарен* предикат;

3) $q(x, y, z) \overset{\text{деф}}{\Longleftrightarrow} z > 1 \;\&\; x \equiv y \;(mod\ z)$ — *тернарен* предикат.

1.5.1 Характеристична функция на предикат

За да можем говорим за примитивна рекурсивност на предикат p, е удобно преди това да въведем *числова* функция, която го представя. Тази функция се нарича *характеристична функция* на предиката и се означава с χ_p. По определение

$$\chi_p(\bar{x}) = \begin{cases} 0, & \text{ако } p(\bar{x}) = \mathbf{t} \\ 1, & \text{ако } p(\bar{x}) = \mathbf{f}. \end{cases}$$

Да обърнем внимание, че *истината* кодираме с 0, а *лъжата* — с 1, в което има известни технически предимства. Ако сте свикнали наобратно — ами пишете си ги наобратно $\ddot{\smile}$.

Когато разполагаме с числова функция, която характеризира един предикат, следващото определение изглежда съвсем естествено

Определение 1.11. Казваме, че предикатът p е *примитивно рекурсивен*, ако неговата характеристична функция χ_p е примитивно рекурсивна функция.

В редки случаи ще ни се налага да говорим само за *рекурсивни* предикати. Определението им е аналогично на горното:

Определение 1.12. Казваме, че предикатът p е *рекурсивен*, ако характеристичната му функция χ_p е рекурсивна функция.

1.5.2 Основни свойства на примитивно рекурсивните предикати

Следват няколко съвсем прости твърдения за предикати, които ще използваме често. Ще ги формулираме за примитивно рекурсивните предикати, макар че те остават в сила и когато заменим "примитивно рекурсивен" с "рекурсивен".

Твърдение 1.6. (НДУ за примитивна рекурсивност.) Предикатът p е примитивно рекурсивен тогава и само тогава, когато съществува примитивно рекурсивна функция f, такава че за всяко $\bar{x} \in \mathbb{N}^n$:

$$p(\bar{x}) \iff f(\bar{x}) = 0.$$

Доказателство. $\Longrightarrow$ Ако p е примитивно рекурсивен, то в качеството на f можем да вземем характеристичната му функция χ_p (която има това допълнително свойство, че е 0-1 функция).

$\Longleftarrow$ Обратно, ако има функция f с горното свойство, с разглеждане на двете възможности за $\bar{x}$ — $p(\bar{x})$ и $\neg p(\bar{x})$, непосредствено се вижда, че

$$\chi_p(\bar{x}) \;=\; sg(f(\bar{x})),$$

т.е. $\chi_p = sg \circ f$ и следователно χ_p е примитивно рекурсивна. □

Задача 1.5. Докажете, че предикатите $=$ и $\geq$ са примитивно рекурсивни.

Решение. Лесно се вижда, че

$$x = y \iff \underbrace{|x - y|}_{f(x,y)} = 0 \qquad \text{и} \qquad x \geq y \iff \underbrace{y \dot{-} x}_{g(x,y)} = 0,$$

откъдето по горното твърдение получаваме, че предикатите $=$ и $\geq$ са примитивно рекурсивни. □

Твърдение 1.7. Нека p е k-местен примитивно рекурсивен предикат, а $f_1, \ldots, f_k$ са примитивно рекурсивни функции, на n аргумента всяка. Тогава и предикатът q, който се дефинира с еквивалентността

$$q(x_1, \ldots, x_n) \overset{\text{деф}}{\iff} p(f_1(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n))$$

също е примитивно рекурсивен.

Доказателство. Следва от факта, че

$$\chi_q \;=\; \chi_p(f_1, \ldots, f_k).$$

□

Пример. Примитивно рекурсивен е предикатът

$$q(x, y, z) \iff x + y = z^2.$$

Твърдение 1.8. (Булевите операции запазват примитивната рекурсивност.) Нека p и q са примитивно рекурсивни предикати. Тогава са примитивно рекурсивни и предикатите $p \,\&\, q$, $p \vee q$ и $\neg p$.

Доказателство. По определение:

$$(p \,\&\, q)(\bar{x}) \iff p(\bar{x}) \,\&\, q(\bar{x}) \iff \chi_p(\bar{x}) = 0 \,\&\, \chi_q(\bar{x}) = 0 \iff \underbrace{\chi_p(\bar{x}) + \chi_q(\bar{x})}_{f(\bar{x})} = 0,$$

където f е примитивно рекурсивна, откъдето съгласно *Твърдение* 1.6 получаваме, че $p \,\&\, q$ е примитивно рекурсивен.

Аналогично, за $p \vee q$ ще имаме:

$$(p \vee q)(\bar{x}) \iff p(\bar{x}) \vee q(\bar{x}) \iff \chi_p(\bar{x}) = 0 \vee \chi_q(\bar{x}) = 0 \iff \underbrace{\chi_p(\bar{x}).\chi_q(\bar{x})}_{g(\bar{x})} = 0,$$

където g очевидно е примитивно рекурсивна.
За характеристичната функция на $\neg p$ можем да запишем:

$$\chi_{\neg p} = \bar{sg} \circ \chi_p.$$

□

От това твърдение, приложено към вече установения факт, че предикатите $=$ и $\geq$ са примитивно рекурсивни (*Задача* 1.5), получаваме примитивната рекурсивност на още основни предикати.

Задача 1.6. Докажете, че всеки от предикатите $<$, $\leq$, $>$ и $\neq$ е примитивно рекурсивен.

Решение. Използваме горното твърдение и еквивалентностите

$$x < y \iff \neg\,(x \geq y), \qquad x \leq y \iff x < y \ \vee \ x = y,$$

$$x > y \iff \neg\,(x \leq y), \qquad x \neq y \iff \neg\,(x = y).$$

□

1.5.3 Ограничени квантори

Определение 1.13. Нека $p(\bar{x}, y)$ е произволен предикат. Дефинираме предикатите q и r, които се получават от p посредством *ограничените квантори за съществуване и всеобщност* както следва:

$$q(\bar{x}, y) \iff \exists z_{z \leq y}\ p(\bar{x}, z)$$

$$r(\bar{x}, y) \iff \forall z_{z \leq y}\ p(\bar{x}, z).$$

С други думи, $q(\bar{x}, y)$ е истина, ако *за поне едно* $z \leq y$ предикатът p в $(\bar{x}, z)$ е истина, докато $r(\bar{x}, y)$ е истина, ако *за всяко* $z \leq y$ предикатът p в $(\bar{x}, z)$ е истина.

Тези дефиниции можем да препишем и така:

$$q(\bar{x}, y) \iff p(\bar{x}, 0) \vee p(\bar{x}, 1) \vee \cdots \vee p(\bar{x}, y)$$

$$r(\bar{x}, y) \iff p(\bar{x}, 0)\ \&\ p(\bar{x}, 1)\ \&\ \ldots\ \&\ p(\bar{x}, y).$$

Ограничените квантори запазват примитивната рекурсивност, за разлика от неограничените, където определено няма да е така.

Твърдение 1.9. (Ограничените квантори запазват примитивната рекурсивност.) Нека p е примитивно рекурсивен предикат. Тогава са примитивно рекурсивни и предикатите q и r, дефинирани като:

$$q(\bar{x}, y) \iff \exists z_{z \leq y}\ p(\bar{x}, z)$$

$$r(\bar{x}, y) \iff \forall z_{z \leq y}\ p(\bar{x}, z).$$

Доказателство. От определението на q се вижда, че

$$q(\bar{x}, y+1) \iff \underbrace{\exists z_{z \leq y}\ p(\bar{x}, z)}_{q(\bar{x}, y)} \vee p(\bar{x}, y+1),$$

откъдето

$$\chi_q(\bar{x}, y+1) = 0 \iff \chi_q(\bar{x}, y) = 0 \ \vee\ \chi_p(\bar{x}, y+1) = 0.$$

Оттук за χ_q можем да запишем

$$\chi_q(\bar{x}, y+1) \ = \ min(\chi_q(\bar{x}, y),\ \chi_p(\bar{x}, y+1)).$$

Освен това при $y = 0$ очевидно

$$q(\bar{x}, 0) \iff p(\bar{x}, 0).$$

Така получихме следната примитивно рекурсивна схема за χ_q:

$$\left|\begin{array}{l} \chi_q(\bar{x}, 0) \ = \ \chi_p(\bar{x}, 0) \\ \chi_q(\bar{x}, y+1) \ = \ min(\chi_q(\bar{x}, y),\ \chi_p(\bar{x}, y+1)). \end{array}\right.$$

Но характеристичната функция χ_p е примитивно рекурсивна по условие, а функцията $min(x, y)$ е такава съгласно *Твърдение* 1.5 и), следователно и χ_q ще е примитивно рекурсивна.

За предиката r, който се определя чрез ограничения квантор за всеобщност, разсъждаваме по подобен начин, като забележим, че този път

$$r(\bar{x}, y+1) \iff \underbrace{\forall z_{z \leq y}\ p(\bar{x}, z)}_{r(\bar{x}, y)}\ \&\ p(\bar{x}, y+1),$$

и значи

$$\chi_r(\bar{x}, y+1) = 0 \iff \chi_r(\bar{x}, y) = 0\ \&\ \chi_p(\bar{x}, y+1) = 0.$$

Тогава за χ_r ще имаме следната дефиниция по рекурсия:

$$\left|\begin{array}{l} \chi_r(\bar{x}, 0) \ = \ \chi_p(\bar{x}, 0) \\ \chi_r(\bar{x}, y+1) \ = \ max(\chi_r(\bar{x}, y),\ \chi_p(\bar{x}, y+1)). \end{array}\right.$$

□

Задача 1.7. Докажете, че е примитивно рекурсивен следният предикат sq, различаващ дали аргументът му е точен квадрат:

$$sq(x) \iff x \text{ е точен квадрат.}$$

Решение. По определение

$$sq(x) \iff \exists z(z^2 = x).$$

Яно е, че горният квантор $\exists z$ се ограничава от x, т.е. всъщност имаме

$$sq(x) \iff \exists z_{z \leq x} \ (\underbrace{z^2 = x}_{q(x,z)}),$$

и значи sq е примитивно рекурсивен, съгласно *Твърдение* 1.7 и *Твърдение* 1.9. □

Често ще ни се налага да използваме следното обобщение на горното *Твърдение* 1.9:

Следствие 1.1. Нека $p(\bar{x}, y)$ е примитивно рекурсивен предикат, а $b(\bar{x})$ е примитивно рекурсивна функция. Тогава са примитивно рекурсивни и предикатите q^* и r^*, дефинирани с еквивалентностите:

$$q^*(\bar{x}) \iff \exists z_{z \leq b(\bar{x})} \ p(\bar{x}, z)$$

$$r^*(\bar{x}) \iff \forall z_{z \leq b(\bar{x})} \ p(\bar{x}, z).$$

Доказателство. Нека $q(\bar{x}, y)$ е предикатът, който се получава от $p(\bar{x}, y)$ с ограничения квантор за съществуване:

$$q(\bar{x}, y) \iff \exists z_{z \leq y} \ p(\bar{x}, z).$$

Тогава очевидно

$$q^*(\bar{x}) \iff q(\bar{x}, b(\bar{x}))$$

и следователно q^* е примитивно рекурсивен като суперпозиция, съгласно *Твърдение* 1.7.

Аналогично разсъждаваме и за предиката r^*. □

1.6 Функционални операции, запазващи примитивната рекурсивност

В този раздел ще въведем няколко допълнителни операции над функции, за които ще покажем, че се изразяват чрез изходните операции суперпозиция и примитивна рекурсия. Това означава, че те не разширяват класа на примитивно рекурсивните функции, а по-скоро служат за улеснение.

1.6.1 Операцията "разглеждане на случаи" (case)

Определение 1.14. Нека са дадени k на брой частични n-местни функции $f_1, \ldots, f_k$ и k на брой n-местни предиката $p_1, \ldots, p_k$, за които е изпълнено условието

$$\forall \bar{x}\ \exists! i\ _{1 \leq i \leq k}\ p_i(\bar{x}) = \mathbf{t}.$$

Дефинираме функция $g : \mathbb{N}^n \multimap \mathbb{N}$ *с разглеждане на случаи* както следва:

$$g(\bar{x}) \simeq \begin{cases} f_1(\bar{x}), & \text{ако } p_1(\bar{x}) \\ \ldots\ldots\ldots\ldots & \ldots\ldots \\ f_k(\bar{x}), & \text{ако } p_k(\bar{x}). \end{cases}$$

Твърдение 1.10. (Операцията case запазва примитивната рекурсивност.) При означенията по-горе, ако функциите $f_1, \ldots, f_k$ и предикатите $p_1, \ldots, p_k$ са примитивно рекурсивни, то и g е примитивно рекурсивна.

Доказателство. Следва от следното представяне за g:

$$g(\bar{x}) \ = \ f_1(\bar{x}).\bar{sg}(\chi_{p_1}(\bar{x})) \ + \ \ldots \ + f_k(\bar{x}).\bar{sg}(\chi_{p_k}(\bar{x})).$$

За да го проверим, да вземем произволно $\bar{x} \in \mathbb{N}^n$. Нека i е (единственото) такова, че $p_i(\bar{x}) = \mathbf{t}$. Тогава $\bar{sg}(\chi_{p_i}(\bar{x})) = 1$, а за всяко $j \neq i$, $\bar{sg}(\chi_{p_j}(\bar{x})) = 0$, и значи

$$g(\bar{x}) = f_1(\bar{x}).\underbrace{\bar{sg}(\chi_{p_1}(\bar{x}))}_{0} + \cdots + f_i(\bar{x}).\underbrace{\bar{sg}(\chi_{p_i}(\bar{x}))}_{1} + \cdots + f_k(\bar{x}).\underbrace{\bar{sg}(\chi_{p_k}(\bar{x}))}_{0} = f_i(\bar{x}).$$

□

Типичният случай, в който ще прилагаме горното твърдение, е при $k = 2$, когато g можем да си представяме и така:

$\mathtt{g}(\bar{\mathtt{x}})$ = `if` $\mathtt{p}(\bar{\mathtt{x}})$ `then` $\mathtt{f_1}(\bar{\mathtt{x}})$ `else` $\mathtt{f_2}(\bar{\mathtt{x}})$

Всъщност за този случай ще ни трябва и твърдение, което казва, че `if then else` конструкцията запазва и частичната рекурсивност. Този факт не може да се докаже с разсъждения, подобни на тези от доказателството на *Твърдение* 1.10, както не е трудно да се забележи, затова го формулираме и доказваме отделно:

Твърдение 1.11. (Операцията if then else запазва частичната рекурсивност.) Нека функцията g се дефинира чрез частичните функции f_1 и f_2 и предиката p както следва:

$$g(\bar{x}) \simeq \begin{cases} f_1(\bar{x}), & \text{ако } p(\bar{x}) \\ f_2(\bar{x}), & \text{ако } \neg p(\bar{x}). \end{cases}$$

Тогава ако f_1 и f_2 са частично рекурсивни, а p е рекурсивен, то g е частично рекурсивна.

Доказателство. Първо ще дефинираме две спомагателни функции F_1 и F_2:

$$\begin{cases} F_1(\bar{x}, 0) = 0 \\ F_1(\bar{x}, y+1) \simeq f_1(\bar{x}) \end{cases}$$

и

$$\begin{cases} F_2(\bar{x}, 0) = 0 \\ F_2(\bar{x}, y+1) \simeq f_2(\bar{x}). \end{cases}$$

Функциите F_i, $i = 1, 2$, се получават с примитивна рекурсия от частично рекурсивните f_i, следователно те също са частично рекурсивни. Да се убедим, че за g е в сила представянето

$$g(\bar{x}) \simeq F_1(\bar{x}, s\bar{g}(\chi_p(\bar{x}))) \ + \ F_2(\bar{x}, \chi_p(\bar{x})).$$

Наистина, ако $\bar{x} \in \mathbb{N}^n$ е такова, че $p(\bar{x})$ е истина, то

$$g(\bar{x}) \ \simeq \ F_1(\bar{x}, \underbrace{s\bar{g}(\chi_p(\bar{x}))}_{1}) \ + \ F_2(\bar{x}, \underbrace{\chi_p(\bar{x})}_{0}) \ \simeq \ F_1(\bar{x}, 1) + F_2(\bar{x}, 0) \simeq f_1(\bar{x}).$$

Аналогично, ако за $\bar{x} \in \mathbb{N}^n$, $p(\bar{x})$ е лъжа, то ще имаме

$$g(\bar{x}) \ \simeq \ F_1(\bar{x}, \underbrace{s\bar{g}(\chi_p(\bar{x}))}_{0}) \ + \ F_2(\bar{x}, \underbrace{\chi_p(\bar{x})}_{1}) \ \simeq \ F_1(\bar{x}, 0) + F_2(\bar{x}, 1) \simeq f_2(\bar{x}).$$

Следователно g е частично рекурсивна като суперпозиция на такива функции. □

Разбира се, горното твърдение е в сила и за случая на произволно k. Тъй като по-нататък в курса ще го получим като следствие от по-общо твърдение, няма да го формулираме тук.

Задача 1.8. Нека тоталната функция g се различава от f в краен брой точки. Докажете, че ако f е примитивно рекурсивна, то и g е такава.

Упътване. Да разгледаме случая, когато f и g са едноместни (в общия случай разсъждението е аналогично). Нека те се различават в точките $a_1, \ldots, a_k$. Тогава g можем да представим по следния начин:

$$g(x) = \begin{cases} g(a_1), & \text{ако } x = a_1 \\ \ldots\ldots\ldots\ldots\ldots & \ldots\ldots \\ g(a_k), & \text{ако } x = a_k \\ f(x), & \text{в останалите случаи.} \end{cases}$$

Остана да докажете формално, че g е примитивно рекурсивна. □

1.6.2 Операциите ограничена сума и ограничено произведение

Определение 1.15. По дадена функция $f(\bar{x}, y)$ дефинираме функция

$$g(\bar{x}, y) \simeq \sum_{z<y} f(\bar{x}, z),$$

за която ще казваме, че се получава от f с *ограничено сумиране* (или е *ограничена сума* на f).

В това определение имаме предвид, че

$$g(\bar{x}, y) \simeq \begin{cases} 0, & \text{ако } y = 0 \\ f(\bar{x}, 0) + \cdots + f(\bar{x}, y-1), & \text{ако } y > 0. \end{cases}$$

Аналогично определяме и операцията ограничено произведение:

Определение 1.16. Казваме, че $h(\bar{x}, y) \simeq \prod_{z<y} f(\bar{x}, z)$ е *ограничено произведение* на $f(\bar{x}, y)$, ако за нея е изпълнено:

$$h(\bar{x}, y) \simeq \begin{cases} 1, & \text{ако } y = 0 \\ f(\bar{x}, 0). \ \ldots \ .f(\bar{x}, y-1), & \text{ако } y > 0. \end{cases}$$

Да се убедим, че тези операции също запазват примитивната рекурсивност.

Твърдение 1.12. (Операциите ограничена сума и произведение запазват примитивната рекурсивност.) Ако функцията f е примитивно рекурсивна, то ограничената сума и ограниченото произведение, които се дефинират чрез нея, също са примитивно рекурсивни.

Доказателство. За ограничената сума g можем да запишем:

$$g(\bar{x}, y+1) \ = \ \underbrace{f(\bar{x}, 0) + \cdots + f(\bar{x}, y-1)}_{g(\bar{x}, y)} + f(\bar{x}, y),$$

откъдето

$$\left| \begin{array}{l} g(\bar{x}, 0) \ = \ 0 \\ g(\bar{x}, y+1) \ = \ g(\bar{x}, y) + f(\bar{x}, y) \ = \ G(\bar{x}, y, g(\bar{x}, y)), \end{array} \right.$$

където $G(\bar{x}, y, z) = f(\bar{x}, y) + z$.

Аналогично за ограниченото произведение h имаме схемата

$$\left| \begin{array}{l} h(\bar{x}, 0) \ = \ 1 \\ h(\bar{x}, y+1) \ = \ h(\bar{x}, y).f(\bar{x}, y) \ = \ H(\bar{x}, y, h(\bar{x}, y)), \end{array} \right.$$

където $H(\bar{x}, y, z) = f(\bar{x}, y).z$. □

Като следствие от горното твърдение получаваме следното

Следствие 1.2. Нека $b(\bar{x})$ е тотална функция. По дадена функция $f(\bar{x}, y)$ дефинираме две функции g^* и h^* както следва:

$$g^*(\bar{x}) \simeq \sum_{z<b(\bar{x})} f(\bar{x}, z) \quad \text{и}$$
$$h^*(\bar{x}) \simeq \prod_{z<b(\bar{x})} f(\bar{x}, z).$$

Твърдим, че ако f и b са примитивно рекурсивни, то g^* и h^* също са примитивно рекурсивни.

Доказателство. Следва от равенствата

$$g^*(\bar{x}) = g(\bar{x}, b(\bar{x})) \quad \text{и}$$
$$h^*(\bar{x}) = h(\bar{x}, b(\bar{x})),$$

където g и h са функциите от определенията по-горе. □

Можем да отидем още по-нататък и да сложим функция и в долната граница на сумата/произведението:

Задача 1.9. Нека f, b_1 и b_2 са примитивно рекурсивни функции. Докажете, че тогава е примитивно рекурсивна и функцията

$$g(\bar{x}) = \begin{cases} \sum_{z=b_1(\bar{x})}^{b_2(\bar{x})} f(\bar{x}, z), & \text{ако } b_1(\bar{x}) \leq b_2(\bar{x}) \\ 0, & \text{иначе.} \end{cases}$$

Решение. Да разгледаме най-напред спомагателната функция

$$h(\bar{x}, y_1, y_2) = \begin{cases} \sum_{z=y_1}^{y_2} f(\bar{x}, z), & \text{ако } y_1 \leq y_2 \\ 0, & \text{иначе.} \end{cases}$$

Лесно се вижда, че за h е в сила представянето:

$$h(\bar{x}, y_1, y_2) = \sum_{z\leq y_2} f(\bar{x}, z) \dot{-} \sum_{z<y_1} f(\bar{x}, z).$$

(Разгледайте поотделно случаите $y_1 \leq y_2$ и $y_1 > y_2$.) Следователно h е примитивно рекурсивна като суперпозиция на примитивно рекурсивни.

Сега вече примитивната рекурсивност на g следва от равенството

$$g(\bar{x}) = h(\bar{x}, b_1(\bar{x}), b_2(\bar{x})).$$

□

1.6.3 Операцията ограничена минимизация

Определение 1.17. Нека е дадена *тотална* функция $f(\bar{x}, y)$. Ще казваме, че функцията $g(\bar{x}, y)$ се получава с *ограничена минимизация* от f и ще пишем

$$g(\bar{x}, y) \;=\; \mu z_{z<y}[f(\bar{x}, z) = 0],$$

ако за g е изпълнено

$$g(\bar{x}, y) \;=\; \begin{cases} min\{z|\ z < y \ \&\ f(\bar{x}, z) = 0\}, & \text{ако } \exists z_{z<y} f(\bar{x}, z) = 0 \\ y, & \text{в противен случай.} \end{cases}$$

Да отбележим, че горната дефиниция можем да препишем посредством *неограничената минимизация* по следния начин:

$$g(\bar{x}, y) \;=\; \begin{cases} \mu z[f(\bar{x}, z) = 0], & \text{ако } !\mu z[f(\bar{x}, z) = 0] \ \&\ \mu z[f(\bar{x}, z) = 0] < y \\ y, & \text{в противен случай.} \end{cases}$$

За разлика от неограничената минимизация, ограничената вече запазва примитивната рекурсивност.

Твърдение 1.13. (Операцията ограничена минимизация запазва примитивната рекурсивност.) Нека f е примитивно рекурсивна. Тогава и функцията

$$g(\bar{x}, y) \;=\; \mu z_{z<y}[f(\bar{x}, z) = 0]$$

е примитивно рекурсивна.

Доказателство. По определение $g(\bar{x}, 0) = 0$. Лесно се съобразява, че $g(\bar{x}, y)$ и $g(\bar{x}, y+1)$ са свързани по следния начин:

$$g(\bar{x}, y+1) \;=\; \underbrace{\begin{cases} g(\bar{x}, y), & \text{ако } g(\bar{x}, y) < y \\ y, & \text{ако } g(\bar{x}, y) = y \ \&\ f(\bar{x}, y) = 0 \\ y+1, & \text{в останалите случаи.} \end{cases}}_{G(\bar{x}, y, g(\bar{x}, y))}$$

За $G(\bar{x}, y, z)$ имаме, че:

$$G(\bar{x}, y, z) \;=\; \begin{cases} z, & \text{ако } z < y \\ y, & \text{ако } z = y \ \&\ f(\bar{x}, y) = 0 \\ y+1, & \text{в останалите случаи.} \end{cases}$$

G се дефинира с разглеждане на случаи, които очевидно са примитивно рекурсивни, и значи по *Твърдение* 1.10 тя ще е примитивно рекурсивна. Тогава и нашата функция g ще е примитивно рекурсивна, защото за нея можем да напишем схемата:

$$\left|\begin{array}{l} g(\bar{x},0) \;=\; 0 \\ g(\bar{x},y+1) \;=\; G(\bar{x},y,g(\bar{x},y)). \end{array}\right.$$

□

По-общо, когато границата на минимизиране е примитивно рекурсивна функция, отново получаваме примитивно рекурсивна функция.

Следствие 1.3. Нека $f(\bar{x},y)$ и $b(\bar{x})$ са примитивно рекурсивни функции. Тогава и функцията

$$g^*(\bar{x}) \;=\; \mu z_{z<b(\bar{x})}[f(\bar{x},z)=0]$$

е примитивно рекурсивна.

Доказателство. Нека

$$g(\bar{x},y) \;=\; \mu z_{z<y}[f(\bar{x},z)=0].$$

Тогава очевидно

$$g^*(\bar{x}) \;=\; g(\bar{x},b(\bar{x})).$$

□

Ще завършим този раздел с проверката за примитивна рекурсивност на още няколко важни функции и предикати. Това е задължителният минимум от функции, чиято примитивна рекурсивност ще ни трябва нататък за теорията, и за задачите.

Твърдение 1.14. Следните функции и предикати са примитивно рекурсивни:

а)
$$rem(x,y) = \begin{cases} \text{остатъка от делението на } y \text{ на } x, & \text{ако } x \neq 0 \\ 0, & \text{ако } x = 0. \end{cases}$$

Доказателство. Нека за $x>0$ да означим

$$\{\frac{y}{x}\} = rem(x,y).$$

Тогава очевидно

$$\{\frac{y+1}{x}\} = \begin{cases} \{\frac{y}{x}\}+1, & \text{ако } \{\frac{y}{x}\}+1 < x \\ 0, & \text{ако } rem(x,y)+1 = x. \end{cases}$$

Сега добавяме и случая $x=0$ (при който по дефиниция $rem(0,y)=0$) и пишем следната рекурсивна зависимост между $rem(x,y+1)$ и $rem(x,y)$:

$$rem(x,y+1) = \begin{cases} rem(x,y)+1, & \text{ако } x>0 \;\&\; rem(x,y)+1<x \\ 0, & \text{в останалите случаи.} \end{cases}$$

Нека $F(x,y,z)$ е функцията:

$$F(x,y,z) = \begin{cases} z+1, & \text{ако } x>0 \ \& \ z+1<x \\ 0, & \text{в останалите случаи.} \end{cases}$$

Така за rem получаваме следната примитивно рекурсивна схема:

$$\left| \begin{array}{l} rem(x,0) = 0 \qquad \% \text{ това важи и за двата случая } x=0 \text{ и } x \neq 0 \\ rem(x,y+1) = F(x,y,rem(x,y)). \end{array} \right.$$

Но F е примитивно рекурсивна, следователно и rem ще е примитивно рекурсивна. □

б)

$$qt(x,y) \ = \ \begin{cases} [\frac{y}{x}], & \text{ако } x \neq 0 \\ 0, & \text{ако } x = 0. \end{cases}$$

Доказателство. За $x \neq 0$ лесно се съобразява, че

$$[\frac{y+1}{x}] = \begin{cases} [\frac{y}{x}], & \text{ако } \{\frac{y+1}{x}\} \neq 0 \\ [\frac{y}{x}]+1, & \text{ако } \{\frac{y+1}{x}\} = 0. \end{cases}$$

Тогава за $qt(x,y+1)$ можем да запишем:

$$qt(x,y+1) = \begin{cases} qt(x,y), & \text{ако } x>0 \ \& \ rem(x,y+1) \neq 0 \\ qt(x,y)+1, & \text{ако } x>0 \ \& \ rem(x,y+1) = 0 \\ 0, & \text{в останалите случаи.} \end{cases}$$

Сега нека $G(x,y,z)$ е функцията

$$G(x,y,z) = \begin{cases} z, & \text{ако } x>0 \ \& \ rem(x,y+1) \neq 0 \\ z+1, & \text{ако } x>0 \ \& \ rem(x,y+1) = 0 \\ 0, & \text{в останалите случаи.} \end{cases}$$

G се дефинира с разглеждане на случаи, които се определят от примитивно рекурсивни условия, следователно тя е примитивно рекурсивна.

Да отбележим, че при $y=0$ имаме $qt(x,0)=0$ и в двата случая — $x=0$ и $x \neq 0$. Тогава за qt получаваме следната схема:

$$\left| \begin{array}{l} qt(x,0) = 0 \qquad \% \text{ отново е вярно и при } x=0, \text{ и при } x \neq 0 \\ qt(x,y+1) = G(x,y,qt(x,y)) \end{array} \right.$$

и следователно qt е примитивно рекурсивна. □

в) $div(x,y) \iff x > 0$ & x дели y.

Доказателство. Лесно се съобразява, че

$$div(x,y) \iff x > 0 \ \& \ rem(x,y) = 0$$

и следователно div е примитивно рекурсивен предикат. □

г) $pr(x) \iff x \geq 2$ & x е просто число.

Доказателство. По определение

x е просто число $\iff x \geq 2$ & броят на делителите на x е равен на 2.

С непосредствена проверка се установява, че броят на делителите на число $x > 0$ се дава от функцията

$$d(x) = \sum_{z=1}^{x} \bar{sg}(rem(z,x)),$$

която е примитивно рекурсивна като ограничена сума. Тогава

$$pr(x) \iff x \geq 2 \ \& \ d(x) = 2$$

и следователно предикатът pr е примитивно рекурсивен. □

д) $p(x) = x$-тото просто число,

с други думи, $p(0) = 2,\ p(1) = 3,\ p(2) = 5,\ \ldots$

По-нататък за по-кратко ще пишем p_x вместо $p(x)$, т.е. ще имаме $p_0 = 2,\ p_1 = 3,\ p_2 = 5,\ \ldots$

Доказателство. Ако знаем $p(x)$, то $p(x+1)$ можем да определим така:

$$p(x+1) = \underbrace{\mu z[z > p(x) \ \& \ pr(z)]}_{F(x,p(x))}.$$

Да се убедим, че горната функция F е примитивно рекурсивна. За целта първо да ограничим z в минимизацията. Това може да стане например с $p(x)! + 1$ (защото ако d е прост делител на $p(x)! + 1$, то очевидно $d > p(x)$ и следователно $d \geq p(x+1)$, с други думи, $p(x+1) \leq d \leq p(x)! + 1$). Така за F ще имаме представянето

$$F(x,y) = \mu z_{z \leq y!+1}[z > y \ \& \ pr(z)] \ = \ \mu z_{z \leq 2y} \underbrace{[\chi_{>}(z,y) + \chi_{pr}(z)}_{f(x,y,z)} = 0].$$

Тъй като функцията f от по-горе е примитивно рекурсивна, по *Следствие* 1.2 и F ще е примитивно рекурсивна, а оттук и функцията $p(x)$, тъй като за нея имаме примитивно рекурсивната схема:

$$\left| \begin{array}{l} p(0) = 2 \\ p(x+1) = F(x, p(x)). \end{array} \right.$$

□

е)

$$(x)_y \ = \ \begin{cases} \text{степента, с която } p_y \text{ участва в разлагането на } x, & \text{ако } x > 0 \\ 0, & \text{ако } x = 0. \end{cases}$$

Забележка. Ако се чудите защо използваме това странно означение, имайте търпение, скоро ще се изясни $\ddot{\smile}$.

Доказателство. При $x = 18$, например, ще имаме

$$18 \ = \ p_0^1 \cdot p_1^2 \cdot p_2^0 \cdot p_3^0 \ \ldots$$

и следователно $(18)_0 = 1,\ (18)_1 = 2,\ (18)_2 = 0,\ (18)_3 = 0 \ldots$.

Лесно се вижда, че при $x > 0$

$$(x)_y = \mu z[p_y^{z+1} \text{ не дели } x] = \mu z_{z \leq x}[p_y^{z+1} \text{ не дели } x] =$$

$$\mu z_{z \leq x}[\bar{sg}(rem(p(y)^{z+1}, x)) = 0].$$

□

1.7 Рекурсивни конструкции, запазващи примитивната рекурсивност

Предстои ни да разгледаме няколко схеми за рекурсия, за които ще покажем, че не извеждат от класа на примитивно рекурсивните функции, т.е. приложени върху примитивно рекурсивни функции, връщат отново такива функции. Това означава, че тяхната изразителна сила не е по-голяма от тази на примитивната рекурсия.

Да започнем с рекурсията от дефиницията на функцията на Фибоначи. Тя ще ни даде идея как да процедираме в една по-обща ситуация, която ще възникне малко по-нататък.

Задача 1.10. Докажете, че функцията на Фибоначи, която се дефинира с равенствата

$$\begin{array}{l} f(0) = 1 \\ f(1) = 1 \\ f(x+2) = f(x) + f(x+1) \end{array} \tag{1.3}$$

е примитивно рекурсивна.

Решение. Ще кодираме в една функция стойностите на f в две "съседни" точки x и $x+1$. Това може да стане по много начини. Един от тях е да използваме функцията

$$\pi(x,y) = 2^x \,.\, 3^y.$$

Ако знаем числото $z = 2^x.3^y$, от него можем да възстановим x и y, защото те са точно степените, с които 2 и 3 участват в разлагането на z на прости множители. Нещо повече, това можем да направим с примитивно рекурсивни средства, защото функциите

$$(z)_0 \qquad \text{и} \qquad (z)_1,$$

които дават показателите на 2 и 3 в това разлагане, са примитивно рекурсивни, съгласно *Твърдение* 1.14 е).

Да дефинираме

$$g(x) = 2^{f(x)} \,.\, 3^{f(x+1)}.$$

Тогава

$$\begin{aligned} f(x) &= (g(x))_0 \\ f(x+1) &= (g(x))_1. \end{aligned} \tag{1.4}$$

От първото равенство следва, в частност, че ако g е примитивно рекурсивна, то и функцията на Фибоначи f ще е такава. Да се опитаме да напишем примитивно рекурсивна схема за g. Базовият случай е ясен:

$$g(0) \overset{\text{деф}}{=} 2^{f(0)}.3^{f(1)} \overset{(1.3)}{=} 2^1.3^1 = 6.$$

По-нататък:

$$\begin{aligned} g(x+1) &\overset{\text{деф}}{=} 2^{f(x+1)} \,.\, 3^{f(x+2)} \overset{(1.3)}{=} 2^{f(x+1)} \,.\, 3^{f(x)+f(x+1)} \\ &\overset{(1.4)}{=} 2^{(g(x))_1}.3^{(g(x))_0+(g(x))_1} = G(x, g(x)), \end{aligned}$$

където $G(x,y) = 3^{(y)_0} \,.\, 6^{(y)_1}$ е примитивно рекурсивна. Сега от представянето

$$\left| \begin{array}{l} g(0) = 6 \\ g(x+1) = G(x, g(x)) \end{array} \right.$$

следва примитивната рекурсивност на g, а оттук и на функцията на Фибоначи. $\square$

1.7.1 Индукция и рекурсия в естествените числа

Най-общо, с *индукция* в множеството на естествените числа доказваме *твърдения* от вида $\forall n P(n)$, докато *рекурсията* служи за дефиниране на функции, множества, рекурсивни програми и пр.

Най-напред да си припомним двата принципа за индукция в естествените числа, които са ни известни от училищната математика:

Принцип на обичайната индукция:

$$\frac{P(0) \quad \forall n\ (P(n) \implies P(n+1))}{\forall n P(n)} \qquad \textbf{(1)}$$

Принцип на пълната индукция:

$$\frac{P(0) \quad \forall n\ (\ (P(0)\ \&\ \ldots\ \&\ P(n)) \implies P(n+1)\ )}{\forall n P(n)} \qquad \textbf{(2)}$$

Забележка. Тази индукция е известна още като *силна* индукция или *възвратна* индукция (на англ.: *course-of-values induction*).

Вторият принцип определено изглежда по-мощен от първия. Всъщност само изглежда. Двата принципа са напълно еквивалентни, т.е. всичко, което може да се докаже по индукция с единия принцип, може да се докаже и с другия. Доказателството на тази еквивалентност ще ни подсети как да действаме в аналогичната ситуация с *рекурсията*, затова да го проведем.

Задача 1.11. Докажете, че принципът на обичайната индукция е еквивалентен на принципа на пълната индукция.

Решение. Ясно е, че ако $\forall n P(n)$ можем да докажем с обичайна индукция, то (толкова повече) можем и с пълна.

Сега обратно, нека $\forall n P(n)$ е изведено с втория изндуктивен принцип. Това означава, че са били изпълнени условията над чертата на правилото **(2)**:

$$P(0), \qquad \forall n\ (\ (P(0)\ \&\ \ldots\ \&\ P(n)) \implies P(n+1)\ ). \qquad (1.5)$$

Трябва да покажем, че $\forall n P(n)$, като използваме принципа **(1)**. За тази цел ще се наложи да усилим P. Да пробваме със свойството $Q(n)$, където

$$Q(n) \iff P(0)\ \&\ \ldots\ \&\ P(n).$$

Сега със "слабия" принцип **(1)** ще покажем, че е вярно "силното" свойство $Q(n)$ за всяко естествено n.

База $n = 0$. Имаме, че

$$Q(0) \overset{\text{деф}}{\iff} P(0),$$

а $P(0)$ е вярно, съгласно (1.5), значи и $Q(0)$ ще е вярно.

Сега да допуснем, че за някое n е вярно $Q(n)$, т.е. вярно е $P(0)$ & ... & $P(n)$. Но тогава отново от (1.5) ще имаме, че е в сила и $P(n+1)$. Значи общо можем да твърдим, че

$$P(0) \ \& \ \ldots \ \& \ P(n) \ \& \ P(n+1)$$

е вярно. Но това е точно $Q(n+1)$, т.е. успяхме да направим индукционната стъпка и вече можем да твърдим, че $\forall n Q(n)$. Но очевидно

$$Q(n) \implies P(n),$$

в частност, от $\forall n Q(n)$ следва $\forall n P(n)$, което и трябваше да покажем.

Забележка. Да обърнем внимание, че $Q(n)$ е по-силно от $P(n)$, обаче

$$\forall n Q(n) \text{ е еквивалентно на } \forall n P(n),$$

тъй че не сме доказали нищо повече от това, което трябваше ☺. □

1.7.2 Пълна (course-of-values) рекурсия

Да видим какво съответства на двете индуктивни схеми, когато говорим за рекурсия при *едноместни* функции в $\mathbb{N}$. Аналогът на обичайната индукция е ясен — това е добре познатата ни проста схема на примитивна рекурсия:

$$\left| \begin{array}{l} f(0) = c \\ f(n+1) \ \underbrace{\ldots n, f(n) \ldots}_{F(n,f(n))} \end{array} \right.$$

Какво ще съответства на *пълната* индукция, обаче? Ясно е, че трябва най-общо да е схема от вида

$$\left| \begin{array}{l} f(0) = c \\ f(n+1) = \underbrace{\ldots n, f(0), \ldots, f(n) \ldots}_{F(n,f(0),\ldots,f(n))?} \end{array} \right.$$

Тук записът $F(n, f(0), \ldots, f(n))$ очевидно не е коректен. Тогава как да изразим формално, че искаме определяемата функция f в точката $n+1$ да зависи (от една или повече) *предишни* нейни стойности $f(0), \ldots, f(n)$? На помощ ни идва трикът с *кодирането*, който приложихме в *Задача*

1.10. Разликата е, че там кодирахме две стойности на f, а тук ще ни се наложи да кодираме цели n.

Функция, която в т. x "помни" стойностите на дадена f във всички точки от 0 до x, се нарича *история* на f. Има много начини да дефинираме функция-история на f. Този, който предлагаме тук, е един от най-простите:

Нека $f : \mathbb{N} \longrightarrow \mathbb{N}$ е *тотална* функция. *История* на f е функцията

$$\hat{f} : \mathbb{N} \longrightarrow \mathbb{N},$$

дефинирана с равенството:

$$\hat{f}(x) = p_0^{f(0)}. \ \ldots \ . p_x^{f(x)}.$$

Тогава очевидно

$$\begin{aligned} f(0) &= (\hat{f}(x))_0 \\ &\ldots\ldots\ldots\ldots \\ f(x) &= (\hat{f}(x))_x. \end{aligned}$$

Следователно ако знаем $\hat{f}(x)$, можем да възстановим стойностите на f във всяка точка, по-малка или равна на x.

Определение 1.18. Казваме, че f се дефинира с *пълна (възвратна) рекурсия* от константата c и тоталната функция $F(x, y)$, ако f удовлетворява равенствата:

$$\left| \begin{array}{l} f(0) = c \\ f(x+1) = F(x, \hat{f}(x)). \end{array} \right.$$

Следващото твърдение показва, че пълната рекурсия не извежда от класа на примитивно рекурсивните функции. И тъй като тази рекурсия е обобщение на примитивната рекурсия, това означава, че двете рекурсии имат еднаква изразителна сила — нещо, което наблюдавахме и при двата принципа — за обичайната и пълната индукция.

Твърдение 1.15. (Пълната рекурсия запазва примитивната рекурсивност.) Нека за *тоталната* функция $f : \mathbb{N} \longrightarrow \mathbb{N}$ е изпълнено:

$$\left| \begin{array}{l} f(0) = c \\ f(x+1) = F(x, \hat{f}(x)). \end{array} \right.$$

Тогава ако F е примитивно рекурсивна, то и f е примитивно рекурсивна.

Доказателство. Ще покажем, че "по-сложната" функция $\hat{f}$ е примитивно рекурсивна. Тогава от равенството

$$f(x) = (\hat{f}(x))_x$$

ще следва, че и f е примитивно рекурсивна.

Забележка. Всъщност $\hat{f}$ не е по-сложна от f, защото ако f е примитивно рекурсивна, то и $\hat{f}$ ще е такава, което се вижда от представянето

$$\hat{f}(x) \;=\; \prod_{y\le x} p_y^{f(y)} \;=\; \prod_{y\le x} p(y)^{f(y)}.$$

За $\hat{f}(x+1)$ можем да запишем:

$$\hat{f}(x+1) \overset{\text{деф}}{=} \underbrace{p_0^{f(0)}.\ \ldots\ .p_x^{f(x)}}_{\hat{f}(x)}.\ p_{x+1}^{f(x+1)} \;=\; \hat{f}(x).p_{x+1}^{F(x,\hat{f}(x))}.$$

Нека

$$G(x,y) \;=\; y.p(x+1)^{F(x,y)}.$$

Така за $\hat{f}$ получаваме следната примитивно рекурсивна схема:

$$\begin{array}{l} \big|\ \hat{f}(0) \overset{\text{деф}}{=} p_0^{f(0)} \;=\; 2^c \;=\; c_1 \\ \big|\ \hat{f}(x+1) \;=\; G(x,\hat{f}(x)). \end{array}$$

И тъй като функцията G е примитивно рекурсивна, то и $\hat{f}$, а оттам и f, ще са примитивно рекурсивни. □

Като непосредствено приложение на току-що доказаното, да решим следната задача:

Задача 1.12. Нека $g(x), h(x)$ и $F(x,y)$ са дадени тотални функции, а f се дефинира чрез тях както следва:

$$f(x) \;=\; \begin{cases} g(x), & \text{ако } h(x) \ge x \\ F(x, f(h(x))), & \text{ако } h(x) < x. \end{cases}$$

Докажете, че ако g, h и F са примитивно рекурсивни, то и f е примитивно рекурсивна.

Решение. От дефиницията на f се вижда, че

$$f(x+1) = \begin{cases} g(x+1), & \text{ако } h(x+1) \ge x+1 \\ F(x+1, \underbrace{f(h(x+1))}_{(\hat{f}(x))_{h(x+1)}}), & \text{ако } h(x+1) < x+1. \end{cases}$$

Сега дефинираме

$$G(x,y) = \begin{cases} g(x+1), & \text{ако } h(x+1) \geq x+1 \\ F(x+1,(y)_{h(x+1)}), & \text{ако } h(x+1) \leq x. \end{cases}$$

Функцията G е примитивно рекурсивна, откъдето и f ще е примитивно рекурсивна, тъй като тя се изразява чрез G чрез следната схема на пълна рекурсия:

$$\begin{aligned} \mid f(0) &= g(0) \\ \mid f(x+1) &= G(x, \hat{f}(x)). \end{aligned}$$

□

Пълната рекурсия се обобщава директно за функции на повече променливи. За целта първо обобщаваме дефиницията за *история* на f, когато f е на повече променливи. Ясно е, че дефиницията трябва да върви по една от променливите; ние ще вземем това да е последната променлива. Нека $f(x_1, \ldots, x_n, y)$ е *тотална* функция. *История* на f е функцията

$$\hat{f} : \mathbb{N}^{n+1} \longrightarrow \mathbb{N},$$

която се дефинира с равенството:

$$\hat{f}(\bar{x}, y) = p_0^{f(\bar{x},0)} . \ \ldots \ . p_x^{f(\bar{x},y)}.$$

Сега вече да фиксираме две тотални функции $G(\bar{x})$ и $H(\bar{x}, y, z)$, съответно на n и $n+2$ аргумента. За функцията $f(\bar{x}, y)$ ще казваме, че се получава с *пълна рекурсия* от G и H, ако за нея са изпълнени равенствата:

$$\begin{aligned} \mid f(\bar{x}, 0) &= G(\bar{x}) \\ \mid f(\bar{x}, y+1) &= H(\bar{x}, y, \hat{f}(\bar{x}, y)). \end{aligned}$$

Разбира се, отново е в сила твърдение, аналогично на 1.15, чието доказателство ще пропуснем, тъй като е съвсем идентично на предишното.

1.7.3 Взаимна рекурсия

В този раздел ще въведем още една рекурсивна схема, която обобщава схемата на примитивната рекурсия, но е със същата изразителна сила.

Определение 1.19. Нека са дадени функциите $g_1(\bar{x})$, $g_2(\bar{x})$, $h_1(\bar{x}, y, z, t)$ и $h_2(\bar{x}, y, z, t)$. Казваме, че $f_1(\bar{x}, y)$ и $f_2(\bar{x}, y)$ се дефинират с *взаимна рекурсия* (или *съвместна рекурсия; mutual recursion* на англ.) от тези функции, ако за f_1 и f_2 е изпълнено:

$$\left|\begin{array}{l} f_1(\bar{x},0) \simeq g_1(\bar{x}) \\ f_1(\bar{x},y+1) \simeq h_1(\bar{x},y,f_1(\bar{x},y),f_2(\bar{x},y)); \end{array}\right.$$

$$\left|\begin{array}{l} f_2(\bar{x},0) \simeq g_2(\bar{x}) \\ f_2(\bar{x},y+1) \simeq h_2(\bar{x},y,f_1(\bar{x},y),f_2(\bar{x},y)). \end{array}\right.$$

Твърдение 1.16. (Взаимната рекурсия запазва примитивната рекурсивност.) При предположенията по-горе, ако функциите g_1, g_2, h_1 и h_2 са примитивно рекурсивни, то f_1 и f_2 са примитивно рекурсивни също.

Доказателство. Ключовата дума отново е *кодиране*. Този път ще кодираме двете функции f_1 и f_2 в една обща функция F. Можем да вземем

$$F(\bar{x},y) \stackrel{\text{деф}}{=} 2^{f_1(\bar{x},y)} \cdot 3^{f_2(\bar{x},y)}.$$

Функциите f_1 и f_2 се изразяват примитивно рекурсивно чрез F:

$$f_1(\bar{x},y) = (F(\bar{x},y))_0 \qquad \text{и} \qquad f_2(\bar{x},y) = (F(\bar{x},y))_1,$$

следователно отново е достатъчно да видим, че е примитивно рекурсивна тази по-обща функция F. Това се вижда по следния начин: базовият случай е ясен, имаме

$$F(\bar{x},0) \stackrel{\text{деф}}{=} 2^{f_1(\bar{x},0)}.3^{f_2(\bar{x},0)} = \underbrace{2^{g_1(\bar{x})}.3^{g_2(\bar{x})}}_{G(\bar{x})},$$

където функцията G очевидно е примитивно рекурсивна. Нататък:

$$\begin{array}{l} F(\bar{x},y+1) \stackrel{\text{деф}}{=} 2^{f_1(\bar{x},y+1)} \cdot 3^{f_2(\bar{x},y+1)} = \\ 2^{h_1(\bar{x},y,f_1(\bar{x},y),f_2(\bar{x},y))} \cdot 3^{h_2(\bar{x},y,f_1(\bar{x},y),f_2(\bar{x},y))} = \\ 2^{h_1(\bar{x},y,(F(\bar{x},y))_0,(F(\bar{x},y))_1)} \cdot 3^{h_2(\bar{x},y,(F(\bar{x},y))_0,(F(\bar{x},y))_1)}. \end{array}$$

Нека

$$H(\bar{x},y,z) = 2^{h_1(\bar{x},y,(z)_0,((z)_1)} \cdot 3^{h_2(\bar{x},y,(z)_0,(z)_1)}.$$

Тогава примитивно рекурсивната схема за F изглежда така:

$$\left|\begin{array}{l} F(\bar{x},0) = G(\bar{x}) \\ F(\bar{x},y+1) = H(\bar{x},y,F(\bar{x},y)). \end{array}\right.$$

Тъй като функциите G и H са примитивно рекурсивни, то F също ще е примитивно рекурсивна. □

Задача 1.13. (Задача за ЕК) Казваме, че функцията h се определя с *вложена рекурсия (nested recursion)* от функциите f, g и π, ако за нея е изпълнено:

$$\left|\begin{array}{l} h(0,y) \simeq f(y) \\ h(x+1,y) \simeq g(x,y,h(x,\pi(x,y))). \end{array}\right.$$

Докажете, че ако f, g и π са примитивно рекурсивни, то и h е примитивно рекурсивна.

1.7.4 Итерация на функция

За произволна едноместна функция f да положим

$$f^n(x) \simeq \underbrace{f(\ldots f}_{n \text{ пъти}}(x)\ldots),$$

като при $n = 0$ имаме предвид, че $f^0(x) = x$.

Последната функционална операция, която ще дефинираме, има много имена. Освен като *итерация,* ще я срещнете още като *ротация, степенуване* или *експоненциация (exponentiation)* на функция.

Определение 1.20. За произволна едноместна функция $f : \mathbb{N} \rightharpoonup \mathbb{N}$ дефинираме *итерацията* ѝ $f^* : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ по следния начин:

$$f^*(n, x) \simeq f^n(x).$$

Оказва се, че тази операция също запазва примитивната рекурсивност. Всъщност тя също има изразителната сила на примитивната рекурсия. Както ще видим по-нататък, (*Теорема* 3.9), в дефиницията на класа на примитивно рекурсивните функции примитивната рекурсия може да се замени с итерация.

Твърдение 1.17. (Итерацията запазва примитивната рекурсивност.) Ако f е примитивно рекурсивни, то и f^* е примитивно рекурсивна.

Доказателство. Примитивно рекурсивната схема за f^* се пише директно от определението на функцията:

$$\left|\begin{array}{l} f^*(0, x) = x \\ f^*(n+1, x) = \underbrace{f(\ldots f}_{n+1 \text{ пъти}}(x)\ldots) = f(\underbrace{f(\ldots f}_{n \text{ пъти}}(x)\ldots)) = f(f^*(n, x)). \end{array}\right.$$

□

Примери. 1) Итерацията на функцията наследник $\mathcal{S}$ е събирането:

$$\mathcal{S}^*(n, x) = \mathcal{S}^n(x) = \underbrace{\mathcal{S}(\ldots \mathcal{S}}_{n \text{ пъти}}(x)\ldots) = x + n.$$

2) Итерацията на функцията предшественик $p(x) = x \dot{-} 1$ е изваждането (в случая — коригираното изваждане):

$$p^*(n, x) = p^n(x) = \underbrace{p(\ldots p}_{n \text{ пъти}}(x)\ldots) = x \dot{-} n.$$

Ако f е на повече аргументи, итерацията можем да правим по кой да е от тях. Нека f е на 2 аргумента. Да дефинираме

$$f^*(n,x,y) \simeq \underbrace{f(x,f(x,\dots f}_{n \text{ пъти}}(x,y)\dots)\ )$$

$$f^{**}(n,x,y) \simeq \underbrace{f(f(\dots f}_{n \text{ пъти}}(x,y)\dots,y),y).$$

Задача 1.14. Докажете, че ако $f(x,y)$ е примитивно рекурсивна, то f^* и f^{**} също са примитивно рекурсивни.

Решение. За f^* имаме представянето

$$\left|\begin{array}{l} f^*(0,x,y) = y \\ f^*(n+1,x,y) = f(x,f^*(n,x,y)), \end{array}\right.$$

докато за f^{**} схемата е такава:

$$\left|\begin{array}{l} f^{**}(0,x,y) = x \\ f^{**}(n+1,x,y) = f(f^*(n,x,y),y). \end{array}\right.$$

□

Задача 1.15. Определете итерациите на събирането, умножението и степенуването.

Решение. За $f(x,y) = x+y$ да пресметнем например f^*:

$$f^*(n,x,y) = f(x,f^*(n-1,x,y)) = x + f^*(n-1,x,y) = \dots =$$

$$\underbrace{x+\dots+x}_{n \text{ пъти}} + f^*(0,x,y) = n.x + y.$$

Забележка. Тези разсъждения не са съвсем формални. Те се използват по-скоро като евристика — за да се ориентираме какъв е отговорът, след което би трябвало с индукция относно n да докажем, че

$$\forall n f^*(n,x,y) = n.x + y.$$

Направете го за упражнение. Да отбележим, че функцията умножение се получава от итерацията на събирането при $y = 0$:

$$f^*(n,x,0) = n.x.$$

По подобен начин получаваме, че итерацията на $g(x,y) = x.y$ е

$$g^*(n,x,y) = x^n.y$$

От нея при $y = 1$ получаваме степенуването:

$$g^*(n, x, 1) \;=\; x^n.$$

Тъй като функцията $h(x, y) \;=\; x^y$ не е комутативна, очакваме двете ѝ итерации да са съвсем различни функции. Нещо повече, това ще са функции от съвсем различен порядък. Да видим:

$$h^*(n, x, y) \;=\; h(x, h^*(n-1, x, y)) \;=\; x^{f^*(n-1,x,y)} \;=\; \cdots = \underbrace{x^{x^{\cdot^{\cdot^{\cdot^{x^y}}}}}}_{n \text{ пъти}}.$$

За h^{**} ще имаме:

$$h^{**}(n, x, y) \;=\; h(h^{**}(n-1, x, y), y) \;=\; h^{**}(n-1, x, y)^y \;=\; \ldots \;=$$

$$h^{**}(0, x, y)^{y^n} \;=\; x^{y^n}.$$

□

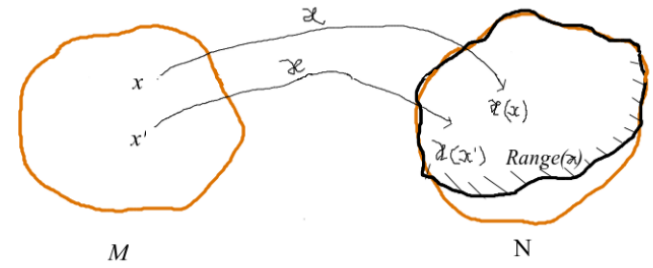
1.8 Кодиране

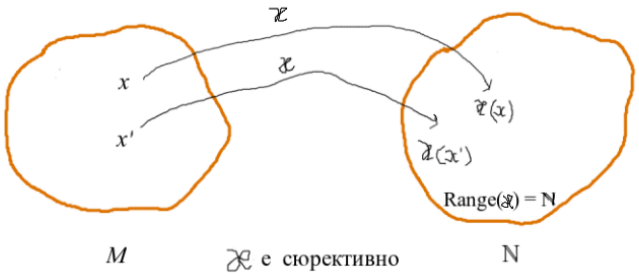
В този раздел ще дефинираме кодиране на наредени двойки, на n-торки и на крайни редици от естествени числа, посредством които по-нататък ще можем да кодираме — оператори, програми, изчисления и пр.

Нека M е произволно множество. *Кодиране* на M ще наричаме всяко инективно изображение

$$\kappa\colon\ M \longrightarrow \mathbb{N}.$$

Ясно е, че за да можем да кодираме с естествени числа едно множество M, то трябва да е най-много изброимо. Обикновено M е множество от *конструктивни обекти* — числа, низове, формули, дървета и пр.





Числото $\kappa(x)$ ще наричаме *код* на x. Инективността на κ е задължителна, с други думи, искаме всяко число да е код на *най-много* един обект. За нашите цели ще бъде удобно *всяко* число да е код на някакъв обект, затова ще се грижим конкретните кодирания, които дефинираме, да са сюрективни. И това, което е от изключителна важност — ще осигуряваме тези кодирания да са *ефективни*, което означава, най-общо казано, по кода $\kappa(x)$ да можем да възстановяваме *алгоритмично* обекта x. (Без последното условие въпросът за съществуване на кодиране на дадено множество M се решава тривиално: НДУ за съществуване на такова кодиране очевидно е изискването M да е най-много изброимо.)

1.8.1 Кодиране на наредени двойки

Оттук до края на курса с Π ще означаваме следното изображение $\Pi : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$:

$$\Pi(x, y) \stackrel{\text{деф}}{=} 2^x(2y + 1) - 1.$$

Да отбележим, че тъй като $2^x(2y+1) \geq 1$, то $\Pi(x, y) \in \mathbb{N}$ за всички естествени x и y. Да се убедим, че това изображение е кодиране на $\mathbb{N}\times\mathbb{N}$, като при това то е сюрективно. Да си спомним, че когато едно изображение е едновременно инективно и сюрективно, то се нарича *биективно* (или *биекция*).

Твърдение 1.18. Изображението Π: $\mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ е биекция.

Доказателство. Трябва да покажем, че за всяко $z \in \mathbb{N}$ съществуват единствени (x, y) : $\Pi(x, y) = z$.

Ще използваме, че всяко *положително* естествено число се представя по единствен начин във вида $2^x(2y + 1)$. Сега да вземем произволно $z \in \mathbb{N}$. Тогава $z + 1 > 0$ и следователно за *единствени* естествени x и y ще имаме, че

$$2^x(2y + 1) \ = \ z + 1.$$

Но тогава $2^x(2y + 1) - 1 \ = \ z$, т.е. $\Pi(x, y) \ = \ z$. □

Щом за всяко $z \in \mathbb{N}$ съществуват единствени x, y, за които $\Pi(x, y) \ = \ z$, то значи съществуват и *функции*, които за всяко z връщат тези x и y. Тези обратни функции обикновено се наричат *декодиращи функции*. Ние ще ги означаваме с L и R. По определение

$$L(\Pi(x, y)) \ = \ x \qquad \text{и} \qquad R(\Pi(x, y)) \ = \ y.$$

За наредената тройка $(\Pi, \ L, \ R)$ ще казваме, че е *кодираща тройка*, или по-общо — *кодираща схема*. Ще я наричаме *примитивно рекурсивна* кодираща тройка, тъй като функциите, участващи в нея, са примитивно рекурсивни.

Твърдение 1.19. Наредената тройка $(\Pi, \ L, R)$ е примитивно рекурсивна кодираща тройка.

Доказателство. Изображението Π е примитивно рекурсивно, защото можем да го препишем като $\Pi(x, y) \ = \ 2^x(2y + 1) \dot{-} 1$, като всички функции, участващи в това представяне, са примитивно рекурсивни.

Сега нека $\Pi(x, y) \ = \ z$, т.е. $2^x(2y+1)-1 \ = \ z$. Тогава $2^x(2y+1) \ = \ z+1$ и следователно $x \ = \ (z + 1)_0$, с други думи

$$L(z) \ = \ (z + 1)_0$$

и значи L е примитивно рекурсивна.

По-нататък, от $2^x(2y+1) \;=\; z+1$ ще имаме

$$2y+1 \;=\; \frac{z+1}{2^x} \;=\; \frac{z+1}{2^{L(z)}},$$

и следователно

$$y \;=\; \frac{\frac{z+1}{2^{L(z)}} - 1}{2}.$$

Сега примитивната рекурсивност на R следва от това, че R можем да препишем като суперпозицията

$$R(z) \;=\; y \;=\; qt(2, qt(2^{L(z)}, z+1) \dot{-} 1),$$

в която всички участващи функции са примитивно рекурсивни. □

Забележка. Да обърнем внимание на един на пръв поглед дребен факт, който обаче ще е важен при някои дефиниции по рекурсия, а именно, това, че за всяко z:

$$L(z) \leq z \qquad \text{и} \qquad R(z) \leq z,$$

като равенство се достига само при $z = 0$ и $z = 1$. За тези две числа имаме, че:

$$\Pi(0,0) = 2^0.(2.0+1) - 1 = 0 \quad \text{и} \quad \Pi(1,0) = 2^1.(2.0+1) - 1 = 1.$$

1.8.2 Кодиране на $\mathbb{N}^n$

За всяко фиксирано $n \geq 1$ ще въведем изображение

$$\Pi_n \colon\; \mathbb{N}^n \;\longrightarrow\; \mathbb{N},$$

за което ще докажем, че е кодиране на наредените n-торки от естествени числа. Дефиницията е с индукция по n:

Определение 1.21.

$$\begin{aligned} \Pi_1(x_1) &\stackrel{\text{деф}}{=} x_1 \\ \Pi_{n+1}(x_1, \ldots, x_{n+1}) &\stackrel{\text{деф}}{=} \Pi(\Pi_n(x_1, \ldots, x_n), x_{n+1}). \end{aligned} \tag{1.6}$$

От тази дефиниция веднага получаваме, че

$$\Pi_2(x_1, x_2) \stackrel{\text{деф}}{=} \Pi(\Pi_1(x_1), x_2) \;= \Pi(x_1, x_2),$$

с други думи, Π_2 съвпада с кодирането Π на наредените двойки естествени числа, което въведохме по-горе.

Забележка. В случай, че се питате защо не дефинирахме кодирането по този начин:

$$\Pi_{n+1}(x_1, \ldots, x_{n+1}) \;=\; \Pi(x_1, \Pi_n(x_2, \ldots, x_{n+1})) :$$

ами да, може е и така да се дефинира, но ние предпочетохме другото определение, защото има малки технически предимства $\ddot\smile$.

Най-напред да съобразим, че Π_n наистина е кодиране на $\mathbb{N}^n$, при това — отново примитивно рекурсивно.

Твърдение 1.20. За всяко $n \geq 1$ изображението $\Pi_n : \mathbb{N}^n \longrightarrow \mathbb{N}$ е биективно и примитивно рекурсивно.

Доказателство. Индукция по n. Пропускаме като очеваден случая $n = 1$ и приемаме, че за произволно $n \geq 1$, Π_n е биективно и примитивно рекурсивно. Тогава от

$$\Pi_{n+1}(x_1, \ldots, x_{n+1}) \;=\; \Pi(\Pi_n(x_1, \ldots, x_n), x_{n+1})$$

и *Твърдение* 1.19 веднага следва, че и Π_{n+1} ще е примитивно рекурсивно.

Да видим, че то е и биективно, т.е. за всяко $z \in \mathbb{N}$ съществуват единствени $(x_1, \ldots, x_{n+1})$, за които $\Pi_{n+1}(x_1, \ldots, x_{n+1}) = z$. Наистина, да изберем произволно естествено z. От *Твърдение* 1.18 знаем, че за единствена двойка числа (x, y) ще е вярно, че $\Pi(x, y) = z$. От индуктивната хипотеза, приложена за x, ще съществуват единствени $x_1, \ldots, x_n$, такива че

$$\Pi_n(x_1, \ldots, x_n) \;=\; x.$$

Но тогава

$$\Pi_{n+1}(x_1, \ldots, x_n, y) \;=\; \Pi(\Pi_n(x_1, \ldots, x_n), y) \;=\; \Pi(x, y) \;=\; z.$$

□

Щом изображението $\Pi_n : \mathbb{N}^n \longrightarrow \mathbb{N}$ е биекция, значи съществуват обратните му функции, които ще означаваме с J_i^n, $i = 1, \ldots, n$. По определение

$$J_i^n(\Pi_n(x_1, \ldots, x_n)) \;=\; x_i.$$

Тези функции сигурно ви напомнят на проектиращите функции I_i^n. Приликата не е само в означението, всъщност декодиращите J_i^n действат върху *кодовете* на n-торките точно както проектиращите I_i^n действат върху самите n-торки.

Задача 1.16. Докажете, че за всяко n и $1 \leq i \leq n$ функциите J_i^n са примитивно рекурсивни.

Решение. Индукция по n. При $n = 1$ имаме $J_1^1(z) = z$.
Да допуснем, че за някое n всички $J_i^n, 1 \leq i \leq n$, са примитивно рекурсивни. Тогава за $n+1$ ще имаме следното:

ако $1 \leq i \leq n$, то очевидно

$$J_i^{n+1}(z) \;=\; J_i^n(L(z))$$

и значи J_i^{n+1} е примитивно рекурсивна, съгласно индукционната хипотеза. Ако пък $i = n+1$, то по определение $J_{n+1}^{n+1}(z) = R(z)$ и значи отново J_{n+1}^{n+1} е примитивно рекурсивна. □

Тази задача, заедно с *Твърдение* 1.20 ни дават, че $(\Pi_n, J_1^n, \ldots J_n^n)$ е примитивно рекурсивна кодираща схема. На нас, обаче, ще ни трябва нещо по-силно от факта, че функциите $J_1^n, \ldots J_n^n$ са примитивно рекурсивни, а именно — ще ни трябва техния *явен вид*, изразен чрез декодиращите L и R. Да се убедим, че той е следният:

Твърдение 1.21. За всяко $n \geq 1$ и $1 \leq i \leq n$:

$$J_i^n \;=\; \begin{cases} R \circ L^{n-i}, & \text{ако } 1 < i \leq n \\ L^{n-1}, & \text{ако } i = 1. \end{cases}$$

Доказателство. Ясно е, че отново ще трябва да разсъждаваме с индукция относно n.

По-горе видяхме, че $J_1^1(z) = z$, т.е. $J_1^1 = I_1^1$, което се съгласува с $J_1^1 \;=\; L^0 \overset{\text{деф}}{=} I_1^1$. Така базата $n = 1$ е проверена.

Сега да приемем, че за някое $n \geq 1$ всяка от функциите J_i^n има горния вид. За $n+1$ разглеждаме следните два случая:
1 сл. $i = n+1$. От определението на Π_{n+1} имаме

$$J_{n+1}^{n+1}(z) \;=\; R(z) \;=\; R(\underbrace{L^{(n+1)-(n+1)}(z)}_{z}).$$

2 сл. $1 \leq i \leq n$. По-горе съобразихме, че $J_i^{n+1}(z) \;=\; J_i^n(L(z))$. Прилагаме индуктивната хипотеза и получаваме

$$J_i^{n+1}(z) \;=\; J_i^n(L(z)) \overset{\text{и.х.}}{=} \begin{cases} R(L^{n-i}(L(z))), & \text{ако } 1 < i \leq n \\ L^{n-1}(L(z)), & \text{ако } i = 1. \end{cases}$$

С други думи,

$$J_i^{n+1}(z) \;=\; \begin{cases} R(L^{n+1-i}(z)), & \text{ако } 1 < i \leq n \\ L^n(z), & \text{ако } i = 1, \end{cases}$$

което заедно с полученото по-горе за $i = n+1$ довършва доказателството на твърдението. □

Ще усилим още малко горното твърдение, като докажем, че всъщност $J_i^n(z)$ зависи "примитивно рекурсивно" не само от z, но и от индексите си n и i, по-точно:

Твърдение 1.22. Функцията

$$F(n,i,z) \;=\; \begin{cases} J_i^n(z), & \text{ако } n \geq 1 \;\&\; 1 \leq i \leq n \\ 0, & \text{в останалите случаи} \end{cases}$$

е примитивно рекурсивна.

Доказателство. За F можем да запишем, използвайки току-що доказаното *Твърдение* 1.21, че

$$F(n,i,z) \;=\; \begin{cases} R(L^*(n-i,z)), & \text{ако } n \geq 1 \;\&\; 1 < i \leq n \\ L^*(n-1,z), & \text{ако } n \geq 1 \;\&\; i = 1 \\ 0, & \text{в останалите случаи.} \end{cases}$$

Сега примитивната рекурсивност на F следва от факта, че итерацията запазва примитивната рекурсивност (*Твърдение* 1.17). □

1.8.3 Кодиране на $\mathbb{N}^*$

В този курс с $\mathbb{N}^*$ ще означаваме множеството на крайните *непразни* редици от естествени числа, с други думи

$$\mathbb{N}^* \;=\; \bigcup_{n=1}^{\infty} \mathbb{N}^n.$$

Да отбележим, че това означение се разминава със стандартното $A^* \stackrel{\text{деф}}{=} \bigcup_{n=0}^{\infty} A^n$, което включва и празния низ в A^*.

Ние няма да включваме ε в $\mathbb{N}^*$, защото идеята ни е $\mathbb{N}^*$ да е множеството от редиците, които са кодове на операторите на нашите програми (програмите ни ще бъдат просто крайни редици от оператори). И тъй като всяка програма има поне един оператор, празният низ очевидно няма да е от този вид.

Всяка *непразна* редица от естествени числа ще означаваме със счупени скобки по ето този начин:

$$\langle x_0, \ldots, x_n \rangle,\; n \geq 0.$$

Тук $n \geq 0$, т.е. започваме броенето от нула, за да осигурим, че n пробягва всички естествени числа, включително и нулата. Разбира се, всичко това е изцяло заради технически удобства.

Определение 1.22. В множеството на всички непразни редици с елементи от $\mathbb{N}$ дефинираме следното изображение $\tau\colon\ \mathbb{N}^* \longrightarrow \mathbb{N}$:

$$\tau(\langle x_0, \ldots, x_n\rangle) \ = \ \Pi(n, \Pi_{n+1}(x_0, \ldots, x_n)). \tag{1.7}$$

От определението се вижда, че ако

$$z = \tau(\langle x_0, \ldots, x_n\rangle) \stackrel{\text{деф}}{=} \Pi(\underbrace{n}_{L(z)}, (\underbrace{\Pi_{n+1}(x_0, \ldots, x_n)}_{R(z)}),$$

то $L(z)+1$ е дължината на редицата с код z, а $R(z)$ е кодът на тази редица (разглеждана като $L(z)+1$-орка). С други думи, $L(z)$ "помни" дължината на редицата с код z, докато $R(z)$ "помни" елементите на тази редица.

Най-напред да покажем, че τ е *кодиране* на $\mathbb{N}^*$, което при това покрива цялото $\mathbb{N}$, т.е. τ е биективно.

Твърдение 1.23. Изображението $\tau\colon\ \mathbb{N}^* \longrightarrow \mathbb{N}$ е биекция.

Доказателство. Трябва да видим, че за всяко $z \in \mathbb{N}$ съществува единствена редица $\langle x_0, \ldots, x_n\rangle$, такава че

$$\tau(\langle x_0, \ldots, x_n\rangle) \ = \ z.$$

Наистина, да означим $L(z)$ с n. От *Твърдение* 1.20, приложено за $n+1$, съществуват единствени $x_0, \ldots, x_n$, такива че

$$\Pi_{n+1}(x_0, \ldots, x_n) \ = \ R(z).$$

Сега вече

$$\tau(\langle x_0, \ldots, x_n\rangle) \stackrel{\text{деф}}{=} \Pi(n, \Pi_{n+1}(x_0, \ldots, x_n)) \ = \ \Pi(L(z), R(z)) \ = \ z.$$

□

Ясно е, че ако знаем редицата $\langle x_0, \ldots, x_n\rangle$, алгоритмично можем да намерим нейния код $\tau(\langle x_0, \ldots, x_n\rangle)$. Обратно, ако знаем кода $z = \tau(\langle x_0, \ldots, x_n\rangle)$ на една редица $\langle x_0, \ldots, x_n\rangle$, очевидно по него можем да възстановим дължината на тази редица и нейните елементи. Но как да формализираме това "възстановяване"? При кодирането Π_n тази идея се формализираше лесно — просто съобразихме, че декодиращите функции на Π_n са примитивно рекурсивни. Тук, обаче, за декодиращи функции очевидно не можем да говорим.

Затова въвеждаме две други функции $lh(z)$ и $mem(z,i)$, които връщат дължината на редицата и нейните елементи. Ето и точните дефиниции:

Нека $z = \tau(\langle x_0, \ldots, x_n\rangle)$. Тогава

$$lh(z) \stackrel{\text{деф}}{=} n,$$

с други думи, lh връща *дължината на редицата* с код n (или по-скоро дължината минус 1, защото дължината на редицата $\langle x_0, \ldots, x_n \rangle$ е $n+1$).

Другата функция mem (от *member*) дефинираме по следния начин (отново предполагайки, че $z = \tau(\langle x_0, \ldots, x_n \rangle)$):

$$mem(z,i) \stackrel{\text{деф}}{=} \begin{cases} x_i, & \text{ако } i \leq lh(z) \\ 0, & \text{ако } i > lh(z). \end{cases} \tag{1.8}$$

Определение 1.23. Казваме, че едно кодиране $\tau : \mathbb{N}^* \longrightarrow \mathbb{N}$ е *ефективно*, ако функциите lh и mem са рекурсивни. Ако тези функции са примитивно рекурсивни, ще казваме, че τ е *примитивно рекурсивно* кодиране.

Сега ще покажем, че нашето кодиране τ е ефективно, като при това неговите функции lh и mem са примитивно рекурсивни.

Твърдение 1.24. Декодиращите функции lh и mem на кодирането τ, дефинирано чрез (1.7), са примитивно рекурсивни.

Доказателство. За функцията "дължина" вече забелязахме, че $lh(z) = L(z)$ и значи lh е примитивно рекурсивна.

Да се убедим, че и mem е примитивно рекурсивна. Наистина, нека $z = \tau(\langle x_0, \ldots, x_n \rangle)$, т.е.

$$z = \Pi(n, \Pi_{n+1}(x_0, \ldots, x_n)).$$

Тогава $R(z) = \Pi_{n+1}(x_0, \ldots, x_n)$ и значи за всяко $x_i, 0 \leq i \leq n$ ще имаме

$$x_i = J_{i+1}^{n+1}(R(z)).$$

Тук е моментът да си спомним, че съгласно *Твърдение* 1.22, функцията $F(n,i,z) = J_i^n(z)$ е примитивно рекурсивна. Следователно за mem можем да запишем:

$$\begin{aligned} mem(z,i) &= \begin{cases} J_{i+1}^{n+1}(R(z)), & \text{ако } i \leq lh(z) \\ 0, & \text{ако } i > lh(z) \end{cases} \\ &= \begin{cases} F(L(z)+1, i+1, R(z)), & \text{ако } i \leq lh(z) \\ 0, & \text{ако } i > lh(z), \end{cases} \end{aligned}$$

откъдето се вижда, че mem е примитивно рекурсивна. □

Ще завършим този раздел с една друга дефиницията на функция-история, различна от тази, която въведохме по-горе.

Нека $f : \mathbb{N} \longrightarrow \mathbb{N}$ е тотална функция. С H_f ще означаваме следната *функция-история* на f:

$$H_f(x) \stackrel{\text{деф}}{=} \tau(\langle f(0), \ldots, f(x) \rangle). \tag{1.9}$$

Твърдение 1.25. Ако $f : \mathbb{N} \longrightarrow \mathbb{N}$ е примитивно рекурсивна, то и нейната история H_f е примитивно рекурсивна.

Доказателство. Искаме да напишем примитивно рекурсивна схема за H_f. За целта да видим как са свързани $H_f(x+1)$ и $H_f(x)$. Имаме

$$\begin{aligned} H_f(x+1) \ &= \ \tau(\langle f(0), \ldots, f(x), f(x+1)\rangle) \ \stackrel{\text{деф } \tau}{=} \\ \Pi(x+1, \ \Pi_{x+2}(f(0), \ldots, f(x), \ f(x+1))) \ &\stackrel{\text{деф } \Pi_{x+2}}{=} \\ \Pi(x+1, \ \Pi(\Pi_{x+1}(f(0), \ldots, f(x)), \ f(x+1))). \end{aligned} \tag{1.10}$$

Но за $H_f(x)$ имаме:

$$H_f(x) = \tau(\langle f(0), \ldots, f(x)\rangle) = \Pi(x, \Pi_{x+1}(f(0), \ldots, f(x))),$$

откъдето

$$\Pi_{x+1}(f(0), \ldots, f(x)) \ = \ R(H_f(x)).$$

Заместваме в изразяването (1.10) за $H_f(x+1)$ и получаваме

$$H_f(x+1) \ = \Pi(x+1, \Pi(R(H_f(x)), f(x+1))).$$

Сега окончателно

$$\left| \begin{array}{l} H_f(0) \ = \ \tau(\langle f(0)\rangle) \ = \ \Pi(0, f(0)) = c \\ H_f(x+1) \ = \underbrace{\Pi(x+1, \Pi(R(H_f(x)), f(x+1)))}_{G(x, H_f(x))}, \end{array} \right.$$

където функцията $G(x,y) = \Pi(x+1, \Pi(R(y), f(x+1)))$ е примитивно рекурсивна. Следователно и H_f е примитивно рекурсивна. □

1.8.4 Задачи

Задача 1.17. (Задача за ЕК) Задачата е да се покаже програмна характеризация на примитивно рекурсивните функции. По-конкретно, да се докаже, че примитивно рекурсивните функции са точно функциите, които се пресмятат с програми на учебния език $LOOP$, който въвеждаме по-долу.

Синтаксис на езика LOOP:

променливи: $X_1, X_2, X_3, \ \ldots$

оператор за присвояване $::= \ X_i := X_i + 1 \ \mid \ X_i := X_j \ \mid \ X_i := 0$

оператор въвеждаме със следната индуктивна дефиниция:

1) Всеки оператор за присвояване е оператор.

2) Ако S_1 и S_2 са оператори, то и $(S_1; S_2)$ е оператор.

3) Ако S е оператор, а X_i е променлива, която не се среща в S, то и *do* X_i *times* S е оператор.

програма на езика *LOOP*: $input(X_1, \ldots, X_n); S$.

Променливите $X_1, \ldots, X_n$ са *входните променливи* на програмата P.

Семантика на езика LOOP:

Да означим с $\mathbb{N}^\omega$ множеството на всички безкрайни редици от естествени числа. С индукция по дефиницията на оператор S определяме функцията M_S (от *meaning*) — значение на S:

$$M_S\colon\ \mathbb{N}^\omega \longrightarrow \mathbb{N}^\omega$$

както следва:

1) – ако S е $X_i := X_i + 1$, то

$$M_S(x_1, x_2, \ldots, x_i \ldots) = (x_1, x_2, \ldots, \underbrace{x_i + 1}_{(i)} \ldots).$$

– ако S е $X_i := X_j$, то

$$M_S(x_1, x_2, \ldots, x_i \ldots) = (x_1, x_2, \ldots, \underbrace{x_j}_{(i)} \ldots).$$

– ако S е $X_i := 0$, то

$$M_S(x_1, x_2, \ldots, x_i \ldots) = (x_1, x_2, \ldots, \underbrace{0}_{(i)} \ldots).$$

2) Ако S е $(S_1; S_2)$, то

$$M_S(x_1, x_2, \ldots) = M_{S_2}(M_{S_1}(x_1, x_2, \ldots)).$$

3) Ако S е *do* X_i *times* S е оператор, то

$$M_S(x_1, x_2, \ldots) = M_{S_0}^{x_i}(x_1, x_2, \ldots).$$

По дефиниция програмата $input(X_1, \ldots, X_n); S$ пресмята функцията $f\colon \mathbb{N}^n \longrightarrow \mathbb{N}$, която се определя с еквивалентността:

$$f(x_1, \ldots, x_n) = y \iff M_S(x_1, \ldots, x_n, 0, 0, \ldots) = (y, y_2, y_3, \ldots)$$

за някои $y_2, y_3, \ldots$ от $\mathbb{N}$.

Упътване. Доказателството, че всяка примитивно рекурсивна функция се пресмята от някоя $LOOP$ програма върви леко по индукция по дефиницията на пр.р.ф. За да покажете обратното, ще ви е нужно следното спомагателно понятие:

Да наречем редицата $(x_1, \ldots, x_k, \ldots)$ *финитна*, ако само краен брой от нейните членове са различни от нула. За такива редици дефинираме *кодиране* α по следния начин:

$$\alpha(x_1, \ldots, x_k, \ldots) \;=\; p_1^{x_1}. \;\ldots\; .p_k^{x_k}. \;\ldots$$

Ако $F\colon\ \mathbb{N}^\omega \longrightarrow \mathbb{N}^\omega$, то с $\hat{F}$ ще означаваме *представящата* на F, която се дефинира като:

$$\hat{F}(z) \;\stackrel{\text{деф}}{=}\; \alpha(f((z)_1, (z)_2, \ldots)).$$

Сега вече с индукция по дефиницията на оператор S покажете, че функцията $\hat{M}_S$ е примитивно рекурсивна.

Задача 1.18. (Функция на Акерман) Докажете, че съществува единствена функция f, определена с равенствата:

$$\begin{aligned} F(0, y) \;&=\; y + 1 \\ F(x + 1, 0) \;&=\; F(x, 1) \\ F(x + 1, y + 1) \;&=\; F(x, F(x + 1, y)) \end{aligned} \tag{1.11}$$

и тази функция е тотална.

Решение. Нека f удовлетворява (1.11). С индукция по x ще покажем, че

$$\forall x \; \underbrace{\forall y \; F(x, y) \text{ е еднозначно определена}}_{P(x)}.$$

База $x = 0$. Следва от това, че

$$F(0, y) \;=\; y + 1.$$

Сега да приемем, че за някое $x, P(x)$ е вярно, т.е.

$\forall y \; F(x, y)$ е еднозначно определена *индуктивна хипотеза* $\mathbf{P(x)}$.

Трябва да покажем, че и $P(x + 1)$ е вярно, т.е. вярно е, че

$$\forall y \; F(x + 1, y) \text{ е еднозначно определена}).$$

Да означим

$$Q(y) \;\stackrel{\text{деф}}{\Longleftrightarrow}\; F(x + 1, y) \text{ е еднозначно определена.}$$

Сега с индукция относно y ще покажем, че $\forall y Q(y)$, което е точно $P(x+1)$. Наистина, при $y = 0$ ще имаме

$$F(x+1,0) \;=\; F(x,1) \overset{\text{и.х. } P(x)}{=} \text{ еднозначно определена.}$$

Допускайки, че за някое y е вярно $Q(y)$, за $y+1$ получаваме последователно

$$F(x+1,y+1) \;=\; F(x,F(x+1,y)) \overset{\text{и.х. } Q(y)}{=} F(x,z)) \overset{\text{и.х. } P(x)}{=} \text{ е еднозначно определена.}$$

□

Сега правим съвсем незначителна промяна в базисното условие в дефиницията на функцията на Акерман, като вместо $y+1$ пишем y. Получаваме функция, която е почти константа! Да видим:

Задача 1.19. Нека за функцията g е изпълнено:

$$\begin{aligned} g(0,y) &= y \\ g(x+1,0) &= g(x,1) \\ g(x+1,y+1) &= g(x,g(x+1,y)). \end{aligned} \tag{1.12}$$

Докажете, че g има следния явен вид:

$$g(x,y) = \begin{cases} y, & \text{ако } x = 0 \\ 1, & \text{иначе.} \end{cases}$$

Решение. Ясно е, че $g(0,y) = y$. Трябва да покажем, че

$$\forall x_{x \geq 1} \forall y \; g(x,y) = 1.$$

За целта с индукция по $x \geq 1$ да се убедим, че $\forall x P(x)$, където

$$P(x) \overset{\text{деф}}{\iff} \forall y \; g(x,y) = 1.$$

База $x = 1$, т.е. доказваме, че

$$\forall y \; \underbrace{g(1,y) = 1}_{Q(y)}.$$

Ще докажем, че $\forall y \; Q(y)$ с индукция относно y. При $y = 0$ от (1.12) получаваме:

$$g(1,0) \;=\; g(0,1) \;=\; 1.$$

Да допуснем, че за някое y е вярно $Q(y)$. Тогава за $Q(y+1)$ ще имаме, съгласно (1.12):

$$g(1,y+1) \;=\; g(0,g(1,y)) \overset{\text{и.х. } Q(y)}{=} g(0,1) \;=\; 1.$$

С това приключва проверката на $\forall y\, Q(y)$, или все едно — на твърдението $P(1)$. Сега да допуснем, че за някое $x \geq 1$ е изпълнено $P(x)$. Трябва да покажем, че и $P(x+1)$ е вярно, т.е.

$$\forall y \underbrace{\; g(x+1,y) = 1 \;}_{R(y)}.$$

Трябва да покажем, че $\forall y\, R(y)$. Действаме отново с индукция относно y.

При $y = 0$ ще имаме

$$g(x+1,0) \;=\; g(x,1) \overset{\text{и.х. } P(x)}{=} 1.$$

Сега да приемем, че $R(y)$ е вярно за някое y. Тогава за $y+1$ ще имаме, съгласно (1.12):

$$g(x+1,y+1) \;=\; g(x,g(x+1,y)) \overset{\text{и.х. } R(y)}{=} g(x,1) \overset{\text{и.х. } P(x)}{=} 1.$$

□

Нека f_k е едноместната функция, която се получава от функцията на Акерман при фиксиран първи аргумент, равен на k, т.е.

$$f_k(y) \;=\; F(k,y)$$

за всяко $y \in \mathbb{N}$. Следват няколко задачи за някои основни свойства на тези функции.

Задача 1.20. Докажете, че функцията f_{k+1} се получава от функцията f_k по следната примитивно рекурсивна схема:

$$\left|\begin{array}{l} f_{k+1}(0) = f_k(1) \\ f_{k+1}(y+1) = f_k(f_{k+1}(y)). \end{array}\right.$$

Докажете още, че всяка от функциите $f_0, f_1, \ldots$ е примитивно рекурсивна.

Доказателство. Това, че f_k удовлетворява горната примитивно рекурсивна схема следва непосредствено от дефиницията на функцията на Акерман:

$$f_{k+1}(0) \overset{\text{деф}}{=} F(k+1,0) \overset{(1.11)}{=} F(k,1) \;=\; f_k(1) \quad \text{и}$$
$$f_{k+1}(y+1) \overset{(1.11)}{=} F(k+1,y+1) \;=\; f_k(f_{k+1}(y)).$$

Оттук с лека индукция по k получаваме, че f_k е примитивно рекурсивна.

□

Задача 1.21. Докажете, че за всички естествени k и y са изпълнени равенствата:

1) $f_{k+1}(y) = f_k^{y+1}(1)$;

2) $f_{k+1}(1) = f_k(\dots f_1(f_0(2)))\dots)$.

Упътване. 1) Отново разсъждаваме с рутинна индукция, този път по y.
База $y = 0$: от предишната задача знаем, че:

$$f_{k+1}(0) = f_k(1) = f_k^{y+1}(1).$$

Сега ако допуснем, че 1) е вярно за някое y, за $y+1$, отново от *Задача* 1.20 ще имаме:

$$f_{k+1}(y+1) = f_k(f_{k+1}(y)) \overset{\text{и.х.}}{=} f_k(f_k^{y+1}(1)) = f_k^{y+2}(1).$$

□

Задача 1.22. Намерете явния вид на функциите f_k за $k = 0, 1, 2, 3, 4$.

Доказателство. По определение за всяко y:

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

т.е. първата функция е $f_0(y) = y + 1$.
За следващите функции ще използваме схемата за f_k, която получихме в *Задача* 1.20:

$$\left|\begin{array}{l} f_{k+1}(0) = f_k(1) \\ f_{k+1}(y+1) = f_k(f_{k+1}(y)). \end{array}\right.$$

Така за f_1 ще имаме:

$$\left|\begin{array}{l} f_1(0) = f_0(1) = 2 \\ f_1(y+1) = f_0(f_1(y)) = f_1(y) + 1 \end{array}\right.$$

и оттук лесно стигаме до извода, че $f_1(y) = y + 2$.

За f_2 получаваме:

$$\left|\begin{array}{l} f_2(0) = f_1(1) = 3 \\ f_2(y+1) = f_1(f_2(y)) = f_2(y) + 2. \end{array}\right.$$

Ясно е, че трябва да търсим f_2 във вида $f_2(y) = ay + b$. Това равенство е в сила за всяко y. Заместваме с възможно най-малките стойности $y = 0$ и $y = 1$ и получаваме следната система за a и b:

$$\left|\begin{array}{l} f_1(0) = f_2(0) = a.0 + b = 3 \\ f_1(y+1) = f_2(1) = a.1 + b = a + 3 = 5. \end{array}\right.$$

Оттук $a = 2, b = 3$, и значи $f_2(y) = 2y + 3$..

Условията за f_3 са:

$$\left| \begin{array}{l} f_3(0) = f_2(1) = 5 \\ f_3(y+1) = f_2(f_3(y)) = 2.f_3(y) + 3. \end{array} \right.$$

Тук вече търсим f_3 във вида $f_3(y) = 2^{ay+b} + c$. Както по-горе, заместваме в горните равенства с различни стойности на y (достатъчно е да вземем например $y = 0, 1$ и 2), за да получим система за a, b и c. От нея получаваме, че $f_3(y) = 2^{y+3} - 3$.

Да се опитаме да пресметнем и f_4. За начало имаме:

$$\left| \begin{array}{l} f_4(0) = f_3(1) = 13 \\ f_4(y+1) = f_3(f_4(y)) = 2^{f_4(y)+3} - 3. \end{array} \right.$$

Виждаме, че $f_4(y+1) \approx 2^{f_4(y)}$ и значи f_4 трябва да е от вида $\underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{y \text{ пъти}}$.

По-точният отговор е $f_4(y) = \underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{y+3 \text{ пъти}} - 3$. Довършете подробностите.

□

Задача 1.23. (Задача за ЕК) Докажете, че функцията на Акерман, която се дефинира като:

$$\left| \begin{array}{l} F(0, y) = y + 1 \\ F(x+1, 0) = F(x, 1) \\ F(x+1, y+1) = F(x, F(x+1, y)). \end{array} \right.$$

не е примитивно рекурсивна.

Упътване. Идеята е да се покаже, че функцията на Акерман расте по-бързо от всяка примитивно рекурсивна функция, или по-формално, да се докаже твърдението:

За всяка примитивно рекурсивна функция g съществува k, такова че

$$g(x_1, \ldots, x_n) < F(k, x_1, \ldots, x_n)$$

за всички естествени $x_1, \ldots, x_n$.

За да докажете горното, ще ви трябват следните свойства на F:

(1) $F(x+1, y) > y + 1$ за всяко x, y;

(2) F е монотонно растяща по двата си аргумента;

(3) $F(x+1, y) \geq F(x, y+1)$ за всяко x, y.

□

Следват две задачи за кодиране.

Задача 1.24. Нека

$$\pi(x,y) = 2^x.(2y+1).$$

Да дефинираме изображението κ, действащо върху множеството $\hat{\mathbb{N}}$ на всички крайни редици от $\mathbb{N}$ (включително и празната) както следва:

$$\kappa(\varepsilon) \;=\; 0; \qquad \kappa(\langle x_1,\ldots,x_n\rangle) \;=\; \pi(x_1,\kappa(\langle x_2,\ldots,x_n\rangle)).$$

Докажете, че κ е примитивно рекурсивно кодиране на $\hat{\mathbb{N}}$.

Доказателство. С пълна индукция по z да се убедим, че за всяко $z \in \mathbb{N}$ съществува единствена редица с код z.

Ако $z = 0$, то $\kappa(\varepsilon) \overset{\text{деф}}{=} 0$, и понеже $\pi(x,y) \geq 1$, то ε е единствената редица с това свойство.

Лесно се вижда, че π в биекция между $\mathbb{N}^2$ и $\mathbb{N}^+$. Тогава за $z > 0$ съществуват единствени x и y, за които $z = \pi(x,y)$. Понеже $y < z$, по индуктивната хипотеза съществува единствена редица $\langle x_1,\ldots,x_n\rangle$, такава че $\kappa(\langle x_1,\ldots,x_n\rangle) = y$. Тогава е ясно, че

$$\kappa(\langle x, x_1,\ldots,x_n\rangle) = \pi(x,\kappa(\langle x_1,\ldots,x_n\rangle)) = \pi(x,y) = z.$$

Нека l и r са декодиращите за кодирането π. (Тъй като 0 не е в областта от стойности на π, да приемем, че $l(0) = r(0) = 0$). Ясно е, че l и r са примитивно рекурсивни.

Сега за функцията $lh(z)$, даваща дължината на редицата с код z имаме следното представяне:

$$\left|\begin{array}{l} lh(0) = 0 \\ lh(z+1) = 1 + lh(r(z+1)). \end{array}\right.$$

Тъй като при $z > 0$ $r(z) < z$, то това е дефиниция с пълна рекурсия и следователно $lh(z)$ е примитивно рекурсивна.

Нека $\kappa(\langle x_1,\ldots,x_n\rangle) = z$. Да дефинираме mem като:

$$mem(z,i) \overset{\text{деф}}{=} \begin{cases} x_i, & \text{ако } z > 0 \;\&\; 1 \leq i \leq lh(z) \\ 0, & \text{иначе} \end{cases}$$

Нека $z > 0$ и $1 < i \leq lh(z)$. Тогава очевидно i-тият елемент на редицата с код z е $i-1$-ви елемент на редицата с код $r(z)$. Значи за mem ще имаме следната рекурсивна връзка:

$$mem(z,i) = \begin{cases} mem(r(z), i-1), & \text{ако } z > 0 \;\&\; 1 < i \leq lh(z) \\ l(z), & \text{ако } z > 0 \;\&\; i = 1 \\ 0, & \text{в останалите случаи.} \end{cases}$$

Има няколко начина да се убедим, че функцията mem е примитивно рекурсивна. Най-краткият е да забележим, че тя се дефинира с *вложена* рекурсия и да се възползваме от *Задача* 1.13, която казва, че тази рекурсия се изразява чрез примитивна рекурсия.

Другият начин се основава на наблюдението, че при рекурсивното обръщение *и двата* аргумента на mem намаляват. Да разгледаме *представящата* $m\hat{e}m$ на mem, дефинирана като:

$$m\hat{e}m(t) \;=\; mem(L(t), R(t))$$

Тогава рекурсивното обръщение

$$mem(z, i) \;\longrightarrow\; mem(r(z), i-1)$$

ще се превърне в

$$m\hat{e}m(t) \;\longrightarrow\; m\hat{e}m(\Pi(r(L(t)), R(t)-1)).$$

Така вече ще имаме $t > \Pi(r(L(t)), R(t)-1)$. Това означава, че можем да напишем схема за пълна рекурсия за $m\hat{e}m$ и значи тя ще е примитивно рекурсивна. Оттук, поради $mem(z,i) = m\hat{e}m(\Pi(z,i))$, ще имаме, че и mem е примитивно рекурсивна.

Но може би най-краткият начин да се убедим в примитивната рекурсивност на mem е да забележим, че можем да я изразим и *явно* по следния начин:

$$mem(z,i) = \begin{cases} l(r^{i-1}(z)), & \text{ако } z > 0 \;\&\; 1 \leq i \leq lh(z) \\ 0, & \text{иначе.} \end{cases}$$

Доказателството е с рутинна индукция по $1 \leq i \leq lh(z)$. Случаят $i = 1$ е очевиден, а допускайки, че за $1 \leq i < lh(z)$ това е така, за $i+1$ ще имаме:

$$mem(z,i) = mem(r(z), i-1) \overset{\text{и.х.}}{=} l(r^{(i-1)-1}(r(z))) = l(r^{i-1}(z)).$$

□

Задача 1.25. Да означим с Fin множеството от всички едноместни крайни функции в естествените числа. Изображението $\kappa : Fin \to N^+$ се дефинира като:

$$\kappa(\theta) = \begin{cases} 1, & \text{ако } Dom(\theta) = \emptyset \\ \prod_{i=1}^{n} p_{x_i}^{\theta(x_i)+1}, & \text{ако } Dom(\theta) = \{x_1, \ldots, x_n\}. \end{cases}$$

Докажете, че κ ефективно кодиране на крайните функции.

Доказателство. Да видим най-напред, че всяко $z > 0$ е код на единствена крайна функция. Наистина, ако $z = 1$ то z е може да е код само на $\emptyset^{(1)}$.

Ако $z > 1$, то z се разлага по единствен начин във вида

$$p_{x_1}^{t_1} \cdot \ldots \cdot p_{x_n}^{t_n},$$

където показателите $t_1, \ldots, t_n$ са положителни. Нека θ е крайната функция с дефиниционно множество $\{x_1, \ldots, x_n\}$, такава че $\theta(x_i) = t_i - 1$ за всяко $i = 1, \ldots, n$. Тогава очевидно $\kappa(\theta) = z$.

Сега трябва да покажем, че са примитивно рекурсивни функциите lh и mem, където

$lh(z) =$ броят на точките, в които крайната функция с код z е дефинирана,

$$mem(z,i) = \begin{cases} \Pi(x_i, \theta(x_i)), & \text{ако } z > 0 \ \& \ \kappa(\theta) = z \ \& \ x_i \text{ е } i\text{-тият} \\ & \text{по големина елемент на } Dom(\theta) \\ 0, & \text{ако } z = 0. \end{cases}$$

Лесно се вижда, че

$$lh(z) \ = \ \sum_{i=0}^{z} sg((z)_i).$$

За $mem(z,i)$ може да разсъждаваме така: нека

$$h(z) = \mu x_{x \leq z}[(z)_x > 0].$$

Ясно е, че при $z \geq 2$, $h(z)$ връща първия елемент от домейна на крайната функция с код z (по-горе видяхме, че всяко такова z е код на непразна крайна функция). Тогава за mem можем да запишем:

$$mem(z,i) = \begin{cases} \Pi(h(z), (z)_{h(z)} - 1), & \text{ако } z \geq 2 \ \& \ i = 1 \\ mem\left(\frac{z}{p_{h(z)}^{(z)_{h(z)}}}, i-1\right), & \text{ако } z \geq 2 \ \& \ i > 1 \\ 0, & \text{в останалите случаи.} \end{cases}$$

Като разсъждавате като в задачата по-горе, покажете, че представящата на mem

$$\hat{mem}(t) \ = \ mem(L(t), R(t))$$

е примитивно рекурсивна, откъдето и самата mem ще е такава. □

Задача 1.26. (бърз ЕК, срок: 10 ноември) Да означим с Fin множеството на всички *крайни* подмножества на $\mathbb{N}$. Дефинираме изображение

$$\kappa \colon \ Fin \ \longrightarrow \ \mathbb{N}$$

по следния начин: ако $A = \{x_1, \ldots, x_n\}$, то $\kappa(A)$ е числото, в чийто двоичен запис единиците са точно на позиции $x_1, \ldots, x_n$ (като броим позициите от дясно наляво, започвайки от позиция 0).

Забележка. Това кодиране е познато като *канонично кодиране* на крайните елементи на 2^M, където M е множество, в което има въведена наредба.

Примери: $\kappa(\emptyset) = 0_{(2)}, \quad \kappa(\{0,1\}) = 11_{(2)}, \quad \kappa(\{1,3,4\}) = 11010_{(2)}$.

1) Докажете, че κ е биекция.

2) Нека $\kappa(A) = z$. Докажете, че са примитивно рекурсивни функциите lh и mem, дефинирани като:

$$lh(z) = |A|$$

$$mem(z,i) = \begin{cases} i\text{-тия по големина елемент на } A, & \text{ако } 1 \leq i \leq lh(z) \\ 0, & \text{иначе.} \end{cases}$$

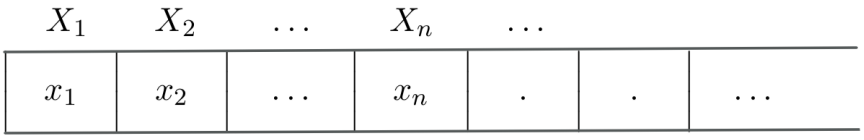
Глава 2

Изчислими функции

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 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 (всъщност прилагателното "unlimited" в наименованието на тези машини е за да покаже, че съдържанието на регистрите им е неограничено). Това, разбира се, отличава МНР и от истинските компютри и показва още веднъж, че те 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, разделени на два вида — аритметични оператори и оператори за преход:

— *аритметични оператори:*

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

2) $Z(n)$, където $n \geq 1$. Запис: $X_n := 0$

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

— *оператор за условен преход (jump):*

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

или `if` $X_m = X_n$ `then goto` q.

Разбира се, с оператор от последния тип можем да моделираме и безусловен `goto`, като вземем $m = n$:

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

Какво прави всеки от тези оператори е съвсем ясно; ние ще въведем формално семантиката им след още няколко синтактични понятия. Най-важното от тях е понятието *програма* за МНР.

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 инструкциите на P са $I_0, \ldots, I_k$.

2.1.2 Семантика на МНР

Нашите програми ще пресмятат функции на краен брой аргументи в естествените числа. Оказва се, обаче, че е по-удобно да въведем тяхната семантика, като първоначално ги разглеждаме като преобразуващи *всички* регистри $X_1, X_2, \ldots$ (или по-точно, 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

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

(Разбира се, тук x_i е съдържанието на i-тия регистър X_i.) По-надолу 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 програмата P 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 $(l, (x_1, x_2, \ \ldots))$ понякога ще отъждествяваме с безкрайната редица $(l, x_1, x_2, \ \ldots)$.

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

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

Да фиксираме произволна програма $P\colon \ I_0, \ \ldots \ , I_k$. 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 стигнем 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 както следва:

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

Типът на входа и на изхода на $step$ е един и същ, следователно можем да итерираме тази функция, т.е. да разглеждаме

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

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

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 $\widetilde{x}$, това ще отбелязваме с $P(\widetilde{x})\uparrow$.

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 $\widetilde{x}$. В този случай ще казваме, че P *спира върху вход* $\widetilde{x}$ *с резултат* y и ще пишем $P(\widetilde{x})\downarrow y$:

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

Разбира се, очакваме ако P спре върху вход $\widetilde{x}$, то резултатът де е еднозначно определен. Но тъй като 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 произволна функция $f\colon M \longrightarrow M$ е вярно следното:

$$f^{n+k}(x) \;=\; \underbrace{(f \circ \cdots \circ f)}_{n+k \text{ пъти}}(x) \;=\; \underbrace{(f \circ \cdots \circ f)}_{n \text{ пъти}} \circ \underbrace{(f \circ \cdots \circ f)}_{k \text{ пъти}}(x) \;=\; f^n(f^k(x)).$$

Твърдението, което което трябва да покажем, е еквивалентно на импликацията

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

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 факта, че върху заключителната конфигурация $(l, \tilde{y})$ функцията $step$ действа като идентитет.

И тъй, получихме, че $step^{t'}(0, \widetilde{x}) = (l, \tilde{y})$. Но по условие $step^{t'}(0, \widetilde{x}) = (l', \tilde{y}')$, което означава, че $(l, \tilde{y}) = (l', \tilde{y}')$, и в частност, $\tilde{y} = \tilde{y}'$. $\square$

2.1.3 Изчислимост на функция с програма за МНР

Вече имаме всичко необходимо, за да въведем едно от най-важните понятия в курса — понятието *изчислима функция*.

Определение 2.2. 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 $x_1, \ldots, x_n, y$ е в сила еквивалентността:

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

Определение 2.3. Казваме, че функцията $f: \mathbb{N}^n \rightharpoonup \mathbb{N}$ е *изчислима*, ако съществува програма за МНР, която я пресмята.

По-нататък ще пишем $P(\bar{x}) \downarrow y$, за да означим, че $P(\bar{x}, 0, 0, \ldots) \downarrow y$, и аналогично $P(\bar{x}) \downarrow$ и $P(\bar{x}) \uparrow$ вместо $P(\bar{x}, 0, 0, \ldots) \downarrow$ и $P(\bar{x}, 0, 0, \ldots) \uparrow$.

Да разгледаме един пример, макар че целта на този курс далеч не е да програмираме на езика за МНР.

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

Решение. Ще използваме, че

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

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

Нека $I_0, \ \ldots \ , I_3$ са следните оператори:

$I_0:$ `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

Лесно се съобразява, че програмата $P: \ I_0, \ \ldots \ , I_3$ връща стойността на израза $x_1 + x_2$, т.е. пресмята събирането. В официалния синтаксис горната програма ще изглежда така:

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

□

Ако разглеждаме горната програма P като пресмятаща функция на 3 аргумента, т.е. ако я стартираме с начална конфигурация $(0, x_1, x_2, x_3, 0, 0, \ldots)$,

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 $n \geq 4$ програмата очевидно ще продължава да пресмята горната функция f. (Е, ако трябва да сме точни, P ще изчислява n-местната функция, която на всяка n-торка $x_1, \ldots, x_n$ съпоставя $f(x_1, x_2, x_3)$.)

Последната възможност за n е $n = 1$. Тогава очевидно горната програма ще пресмята идентитета $I(x) = x$.

С други думи, една и съща програма може да пресмята съвършено различни функции, в зависимост от това какъв е броят на входните ѝ променливи.

Навсякъде в курса с $\mathcal{C}_n$ ще означаваме класа на всички изчислими функции на n аргумента, т.е.

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

При $n = 1$ горният индекс обикновено се изпуска, или все едно

$$\mathcal{C} \ = \ \mathcal{C}_1 = \ \{f \mid f \text{ е едноместна изчислима функция}\}.$$

2.2 Кодиране на програмите за МНР

В този раздел ще дефинираме *кодиране* на инструкциите и програмите за МНР с естествени числа. Тоива е стъпка, която има фундаментално значение за Теорията на изчислимостта. Като концепция, кодирането е въведено за пръв път от Гьодел като *аритметизация*, което в неговия случай означава дигитализирането на всички аксиоми и правила за извод в определена формална система, като и кодирането на доказателства, теореми и пр. (т.е. превръщането ма 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 заменим 0-те и 1-те с естествени числа, защото в нашия случай това са данните, върху които работят програмите за МНР. Да отбележим, че когато изчислителният модел е базиран на *машини на Тюринг*, не възниква необходимост от кодиране им, защото те самите 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 най-напред 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, и освен това е ефективно, т.е. знаейки кода $\beta(I)$, можем алгоритмично да възстановим инструкцията I.

Да напомним, че имаме следните типове инструкции:

1) $S(n),\ n \geq 1$, или $X_n := X_n + 1$

2) $Z(n),\ n \geq 1$, или $X_n := 0$

3) $T(m,n),\ m,n \geq 1$, или $X_m := X_n$

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

Инструкциите са 4 вида, затова решаваме да ги кодираме с числа, даващи различен остатък по модул 4. По-точно, нека

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

където $\Pi(x,y) \overset{\text{деф}}{=} 2^x(2y+1)-1$ е кодирането на $\mathbb{N}\times\mathbb{N}$, а Π_3 е биекцията (1.6) между $\mathbb{N}^3$ и $\mathbb{N}$, които въведохме миналия път.

Да отбележим, че взехме $\beta(S(n)) \overset{\text{деф}}{=} 4(n-1)$, а не $\beta(S(n)) = 4n$, защото номерацията на регистрите започва от $n = 1$ (а не от $n = 0$), а ние ще искаме да осигурим, че всяко число е код на някаква инструкция. По тази причина в последния случай имаме $\beta(J(m,n,q)) = \Pi_3(m-1,n-1,\underline{q})+3$, 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.

Аналогично се разсъждава и в останалите случаи. Все пак, да разгледаме един от тях — примерно последния, когато $z = 4t + 3$. Ясно е, че в този случай единствената възможност е z да е код на инструкция 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. Твърдим, че кодът на инструкцията $J(m, n, q)$ е точно z. Наистина,

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

□

От доказателството на горното твърдение се вижда още, че по число z *алгоритмично* можем да посочим инструкцията с код z

2.2.2 Кодиране на програмите

Всяка програма P за МНР е просто редица от инструкции:

$$I_0, \ldots, I_k.$$

Ние, обаче, вече знаем как да кодираме инструкции. Освен това знаем как да кодираме и редици от естествени числа — с изображението $\tau\colon \mathbb{N}^* \longrightarrow \mathbb{N}$, което в раздел 1.8.3 дефинирахме чрез равенството

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

Затова не е неочаквана следната дефиниция за *код* $\gamma(P)$ на програма $P: \; I_0, \ldots, I_k$:

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

Нека

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

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. Основава се на факта, че τ и β са биекции, но да го разпишем все пак. 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

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

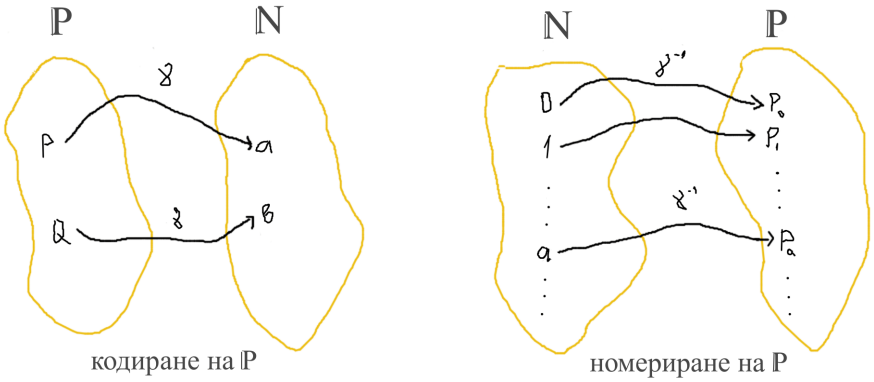
Тогава е ясно, че програмата $P\colon\ I_0, \ldots, I_k$ ще има код a и тя ще е единствената с това свойство. Да отбележим, че както в предишното твърдение, и тук успяхме *да конструираме* програмата с код произволното число a. □

2.2.3 Ефективно номериране на програмите и изчислимите функции

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.

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

С други думи, $P_a = \gamma^{-1}(a)$.



Ясно е тогава, 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 *изброяване* или *номериране* (enumeration) на множеството $\mathbb{P}$ на всички програми. Ще я наричаме *стандартно изброяване* на $\mathbb{P}$.

Разбира се, винаги можем да наредим 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 еднакви програми. В него, обаче, има *семантични* повторения (т.е. срещат се еквивалентни програми) и този факт, както ще видим по-нататък в курса, 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 гарантирано неефективна.

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

Определение 2.4. 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 ефективна редица всички функции от класа $\mathcal{C}_n$.

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

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

Тези повторения не могат да бъдат избегнати, в смисъл, че ако наредим изчислимите функции в редица без повторения, 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 2.5. 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. По-нататък ще видим, че съвкупността от всички индекси на дадена изчислима функция образуват доста сложно множество.

2.3 Еквивалентност между частична рекурсивност и изчислимост

В този раздел ще покажем, че двата подхода към изчислимостта, които въведохме дотук — подходът на Клини с частично рекурсивните функции и подходът с изчислителния модел, базиран на МНР, са еквивалентни, т.е. определят един и същ клас от функции. Твърдение от този тип обикновено се нарича *теорема за еквивалентност.*

2.3.1 От частична рекурсивност към изчислимост

Първо ще се заемем с по-лесната половина на теоремата за еквивалентност, а именно:

Твърдение 2.5. Всяка частично рекурсивна функция е изчислима.

Доказателство. Нека f е произволна частично рекурсивна функция. Ще разсъждаваме с индукция по дефиницията на f, за да покажем, че за нея съществува програма за МНР, която я пресмята.

Ако f е базисна функция, нещата са ясни:

– ако f е функцията $\mathcal{S}(x) = x + 1$, то f се пресмята от програмата $P\colon\ S(1)$;

– ако f е функцията $\mathcal{O}(x) = 0$, то f се пресмята от програмата $P\colon\ Z(1)$;

– ако f е проектиращата функция I_k^n (където $I_k^n(x_1,\ldots,x_n) \stackrel{\text{деф}}{=} x_k$), то f се пресмята от програмата P: $T(1,k)$.

Нека сега

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

като за g и $h_1,\ldots,h_k$, съгласно индукционната хипотеза, съществуват програми P и $Q_1,\ldots,Q_k$, които ги пресмятат. С тяхна помощ ще построим нова програма R, която пресмята f.

Тук ще направим една уговорка, чийто смисъл ще стане ясен след малко. Ще предполагаме, че горните програми $P, Q_1,\ldots,Q_k$ са от т. нар. "стандартен тип". По определение една програма $I_0,\ldots,I_k$ е *стандартна*, ако за всеки адрес q от оператор за преход $J(m,n,q)$ е вярно, че ако $q > k$, то $q = k+1$. Ясно е, че по всяка програма за МНР алгоритмично можем да получим еквивалентна на нея стандартна програма.

Нека горната функция f е на n аргумента. Тогава за всяко $\bar{x} \in \mathbb{N}^n$:

$$f(\bar{x}) \simeq y \stackrel{\text{деф}}{=} \exists z_1 \ldots \exists z_k\ (h_1(\bar{x}) \simeq z_1\ \&\ \ldots h_k(\bar{x}) \simeq z_k\ \&\ h_k(\bar{x}) \simeq z_k\ \&\ g(z_1,\ldots,z_k) \simeq y)$$

Ясно е, че програмата R първо трябва да извика $Q_1,\ldots,Q_k$ върху $\bar{x}$, за да се пресметнат (ако съществуват) стойностите $z_1,\ldots,z_k$, и после да извика P с вход $(z_1,\ldots,z_k)$.

В общи линии, алгоритъмът за f е този (а и трудно би могъл да бъде друг $\ddot\smile$), но тук възникват няколко технически въпроса. Единият е, че трябва да се погрижим да запазим входните стойности $x_1,\ldots,x_n$, защото те могат да бъдат изтрити още при извикването на Q_1. Освен това, междинните стойности $z_1,\ldots,z_k$ също трябва да бъдат съхранени в достатъчно далечни регистри, за да не бъдат загубени, докато дойде време да бъдат включени в изчислението. Тук под "далечен регистър" имаме предвид регистър, който няма да бъде засегнат при работата на програмите $Q_1,\ldots,Q_k$. Лесно е съобразява, че един такъв регистър е X_m за

$$m \ = \ max\{n, m_1,\ldots,m_k\},$$

където m_i е най-големият индекс на регистър, който се среща в операторите на никоя от програмите Q_i, $1 \le i \le k$.

Тогава нашата програма R трябва да започва със следните инструкции:

$$X_{m+1} := X_1,\ \ldots\ , X_{m+n} := X_n.$$

Идеята ни е веднага след тях да сложим инструкциите на първата програма Q_1. Само че те не могат да останат същите, защото преди тях вече сме написали горните n на брой трансфери. Значи сега всеки адрес q в

оператор за преход $J(m,n,q)$ трябва да бъде увеличен с n. Освен това, след като Q_1 приключи работата си върху $\bar{x}$, трябва да се погрижим да прехвърлим резултата от първия регистър в някой по-далечен — да кажем X_{m+n+1}.

Сега се насочваме към пресмятането на $Q_2(\bar{x})$. За целта зареждаме първите n регистъра с $x_1,\ldots,x_n$, а останалите до X_m инициализираме с 0:

$$X_{m+n+1} := X_1, X_1 := X_{m+1}, \ldots, X_n := X_{m+n}, X_{n+1} := 0, \ldots, X_m := 0,$$

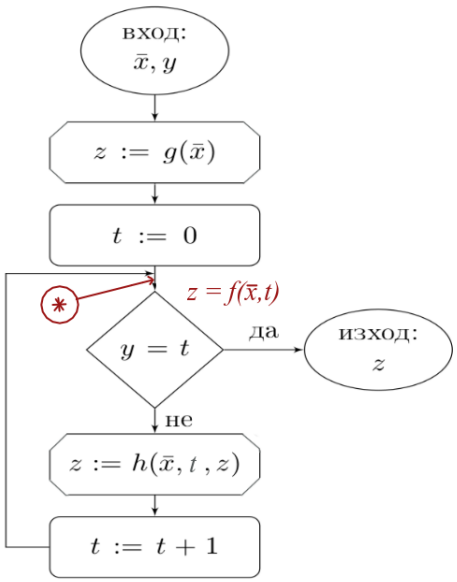
които са точно $m+1$ на брой. Следователно адресите в операторите за преход в следващата програма Q_2 трябва да бъдат увеличени с общо $n+(m+1)+l_1$, където l_1 е броят на инструкциите на Q_1. По същия начин процедираме и със следващите програми $Q_3,\ldots,Q_k$ и P, като за P, разбира се, инициализираме първите k регистъра $X_1,\ldots,X_k$ със стойностите на последните регистри $X_{m+n+1},\ldots,X_{m+n+k}$. Ясно е, че новопостроената програма R ще пресмята f.

Нека сега f се получава с примитивна рекурсия от g и h, като за тези две функции по индукционното предположение има програми за МНР, които ги пресмятат.

Нека по-конкретно за f е изпълнено:

$$\begin{aligned} f(\bar{x},0) &\simeq g(\bar{x}) \\ f(\bar{x},y+1) &\simeq h(\bar{x},y,f(\bar{x},y)). \end{aligned} \tag{2.5}$$

Да съобразим, че f се пресмята от следния алгоритъм:



За целта да видим, че когато, образно казано, изчислението "преминава" през контролната точка $(*)$, е изпълнено равенството

$$z = f(\bar{x}, t)$$

за текущите стойности на регистрите z и t (тези стойности също ще означаваме със z и t).

Това ще направим с индукция по броя на преминаванията през тази контролна точка. Когато изчислението за първи път премине през нея, ще имаме, в частност, че $g(\bar{x})$ е било дефинирано. Тогава от $z = g(\bar{x})$ получаваме

$$z = g(\bar{x}) \overset{(2.5)}{=} f(\bar{x}, 0) = f(\bar{x}, t).$$

Да приемем, че при някакво преминаване през $(*)$ за текущите стойности на z и t е било вярно, че $z = f(\bar{x}, t)$. Тогава при следващото преминаване през тази контролна точка за новите стойности z_{new} и t_{new} ще имаме $z_{new} = h(\bar{x}, t, z)$ и $t_{new} = t + 1$, откъдето

$$z_{new} = h(\bar{x}, t, z) \overset{\text{и.х.}}{=} h(\bar{x}, t, f(\bar{x}, t)) \overset{(2.5)}{=} f(\bar{x}, t + 1) = f(\bar{x}, t_{new}).$$

С това индукцията е приключена. Разбира се, при излизане от цикъла условието $z = f(\bar{x}, t)$ ще продължава да е в сила. Но тогава вече $t = y$, и значи $z = f(\bar{x}, y)$.

Получихме, че ако горната програма завърши при вход $(\bar{x}, y)$, то резултатът ще е $f(\bar{x}, y)$. Ако пък за някой вход $(\bar{x}, y)$ тя не върне резултат, лесно се съобразява, че в такъв случай f няма да е дефинирана в $(\bar{x}, y)$.

Да означим с F функцията, която се пресмята от горната програма. Погоре всъщност показахме, че за всяко $\bar{x}, y$ и z:

$$F(\bar{x}, y) \simeq z \implies f(\bar{x}, y) \simeq z,$$

което ще рече, че $F \subseteq f$.

Освен това видяхме, че

$$(\bar{x}, y) \notin Dom(F) \implies (\bar{x}, y) \notin Dom(f),$$

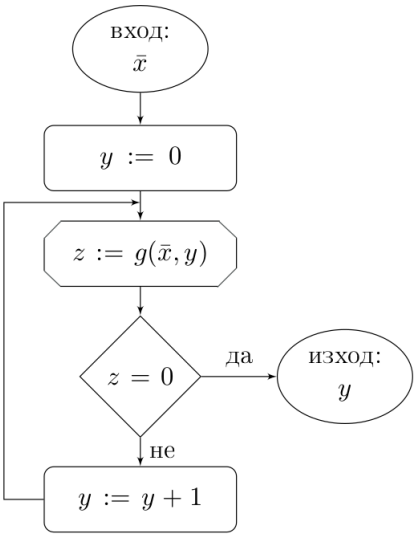
или все едно, $Dom(f) \subseteq Dom(F)$.

Сега вече можем да приложим *Задача* 1.1, която ни гарантира, че $f = F$, с други думи, горната блок схема наистина пресмята функцията f. Това, което остава, е да я напишем във вид на програма за МНР $\ddot{\smile}$.

Да разгледаме и последния случай, когато f е получена с минимизация от g, т.е. за нея е изпълнено

$$f(x_1, \ldots, x_n) \simeq \mu y[g(x_1, \ldots, x_n, y) \simeq 0].$$

Разбира се, съгласно индукционната хипотеза, за g съществува програма за МНР, която я пресмята. Да съобразим, че ето тази блок схема пресмята f.



За целта трябва да вземем под внимание една особеност на μ-операцията, която обсъдихме след нейната дефиниция 1.8. Става въпрос за това, че ако $f(\bar{x}) \simeq y$, то y не само е първото естествено число, за което $g(\bar{x}, y) \simeq 0$, но за него е вярно още, че за всяко $z < y$, $g(\bar{x}, z)$ *има стойност* (която, разбира се, трябва да е положителна). Лесно се вижда, че ако има y с тези свойства, то ще бъде намерено от горната блок схема и обратно, ако тя върне резултат y, то за него ще е вярно, че $f(\bar{x}) \simeq y$. □

2.3.2 От изчислимост към частична рекурсивност

За да покажем, че всяка изчислима функция е частично рекурсивна, ще трябва предварително да свършим известно количество техническа работа. За начало да си припомним най-основните понятия и означения, свързани с работата на една машина с неограничени регистри. *Конфигурация* за машината е безкрайна редица от естествени числа

$$(l, (x_1, x_2, \dots)),$$

която ще съкращаваме до $(l, \widetilde{x})$ и ще отъждествяваме с редицата $(l, x_1, x_2, \dots)$. В нея l е адресът на инструкцията, която предстои да се изпълни, а $(x_1, x_2, \dots)$ е текущото състояние на паметта на машината. *Начална конфигурация* (за входа $(x_1, \dots, x_n)$) е редицата $(0, x_1, \dots x_n, 0, 0, \dots)$.

По дефиниция n-местната функция f се пресмята от програмата $P\colon I_0, \ldots, I_k$, ако за всички естествени $x_1, \ldots, x_n, y$ е изпълнено:

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

Тук записът $P(\widetilde{x}) \downarrow y$, който въведохме в раздел 2.1.2, означава, че P спира върху паметта $\widetilde{x}$ с резултат y и се дефинира формално така:

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

Функцията *step* (или по-точно $step_P$) задаваше едностъпковото преобразование на програмата P. Стойността на $step(l, \widetilde{x})$ дефинирахме в зависимост от вида на l-тия оператор на програмата P. За нашите цели, обаче, се оказва по-удобно да дефинираме *step* посредством една друга функция *next*, която не зависи от конкретната програма. Тази функция по даден оператор I и конфигурация $(l, \widetilde{x})$ дава *следващата* конфигурация, към която се преминава, когато се приложи оператора I към конфигурацията $(l, \widetilde{x})$.

Формалната дефиниция на *next* е с разглеждане на четирите възможности за оператора I:

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

Сега функцията *step* за програмата $P\colon\ I_0, \ldots, I_k$, изразена чрез *next*, изглежда така:

$$step(l, \widetilde{x}) = \begin{cases} next(I_0, (l, \widetilde{x})), & \text{ако } l = 0 \\ \ldots\ldots\ldots\ldots & \\ next(I_k, (l, \widetilde{x})), & \text{ако } l = k \\ (l, \widetilde{x}), & \text{ако } l > k. \end{cases} \tag{2.6}$$

Идеята ни е да покажем, че *step* е примитивно рекурсивна. Тя, обаче, е изображение от вида

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

и въобще не е числова функция! За да стане такава, очевидно трябва да преминем от конфигурации към някакви техни *кодове*. Проблемът е, че конфигурациите са *безкрайни* редици от естествени числа, които очевидно няма как да бъдат "запомнени" с едно естествено число.

За щастие, конфигурациите, които участват в едно реално изчисление, са от по-специален вид. Нашите програми тръгват от начални конфигурации $(0, (x_1, \ldots, x_n, 0, 0, \ldots))$, в които само първите n регистъра могат да са различни от 0. Освен това, тъй като всяка програма е *краен* текст, в хода на изчисленията тя може да променя само *краен брой* от регистрите — само тези, които участват в нея. Следователно във всеки момент от едно изчисление само краен брой регистри ще имат съдържание, различно от 0.

Конфигурации $(l, (x_1, \ldots, x_n, 0, 0, \ldots)\ )$, в които условието $x_i \neq 0$ е изпълнено за краен брой i, ще наричаме *финитни*. Такива конфигурации вече можем да кодираме с естествени числа.

Код на финитната конфигурация $(l, \widetilde{x}) = (l, x_1, x_2,\ \ldots)$ ще наричаме числото

$$\delta(l, \widetilde{x}) \stackrel{\text{деф}}{=} 2^l.p_1^{x_1}.p_2^{x_2}.\ \ldots.$$

Ясно е, че всяко естествено число $z > 0$ е код на единствена конфигурация $(l, x_1, x_2,\ \ldots)$ — тази, за която

$$l = (z)_0, \quad x_1 = (z)_1, \quad x_2 = (z)_2,\ \ldots.$$

Когато казваме конфигурация, оттук нататък ще имаме предвид финитна конфигурация.

Да означим със $Step$ функцията, която действа върху кодовете на конфигурациите така, както $step$ действа върху самите конфигурации:

$$Step(\delta(l, \widetilde{x})) \stackrel{\text{деф}}{=} \delta(step(l, \widetilde{x})),$$

или все едно

$$Step(z) \ = \ \delta(step((z)_0, (z)_1, (z)_2,\ \ldots)).$$

Разбира се, горното равенство се отнася само за $z > 0$. За да бъде тотална функцията $Step$, полагаме $Step(0) \stackrel{\text{деф}}{=} 0$.

Да дефинираме с подобна идея и функция $Next$ чрез $next$. Тя ще преработва кодовете на конфигурациите точно както $next$ преработва самите конфигурации. По-точно, при фиксиран оператор I полагаме

$$Next(I, z) = \begin{cases} \delta(next(I, ((z)_0, (z)_1, (z)_2,\ \ldots))), & \text{ако } z > 0 \\ 0, & \text{ако } z = 0. \end{cases} \tag{2.7}$$

Да разпишем как изглежда $Next(I, z)$ във всеки от четирите случая за оператора I. Навсякъде по-долу ще предполагаме, че $z > 0$ (т.е. z е код на някаква конфигурация).

Имаме, че $next(S(n),(l,\widetilde{x})) = (l{+}1, x_1, \ldots, x_{n-1}, x_n{+}1, x_{n+1}, \ldots)$, следователно

$$Next(S(n),z) = \delta((z)_0{+}1, (z)_1, ..., \underbrace{(z)_n{+}1}_{(n)}, ...) = 2^{(z)_0+1}.p_1^{(z)_1} \ldots p_n^{(z)_n+1} \cdots = 2zp_n.$$

Аналогично, от $next(Z(n),(l,\widetilde{x})) = (l{+}1, x_1, \ldots, x_{n-1}, 0, x_{n+1}, \ldots)$ ще имаме

$$Next(Z(n),z) = \delta((z)_0{+}1, (z)_1, ..., \underbrace{0}_{(n)}, ...) = 2^{(z)_0+1}.p_1^{(z)_1} ... p_n^0 ... = \frac{2z}{p_n^{(z)_n}}.$$

За оператора $T(m,n)$ имаме по определение
$next(T(m,n),(l,\widetilde{x})) = (l{+}1, x_1, \ldots, x_{m-1}, x_n, x_{m+1}, \ldots))$, и значи

$$Next(T(m,n),z) = \delta((z)_0{+}1, (z)_1, ..., \underbrace{(z)_n}_{(m)}, ...) = 2^{(z)_0+1}.p_1^{(z)_1} ... p_m^{(z)_n} ... = \frac{2zp_m^{(z)_n}}{p_m^{(z)_m}}.$$

При оператор за преход $J(m,n,q)$ имаме, че

$$next(J(m,n,q),(l,\widetilde{x})) = \begin{cases} (q,\widetilde{x}), & \text{ако } x_m = x_n \\ (l{+}1,\widetilde{x}), & \text{ако } x_m \neq x_n. \end{cases}$$

Следователно $Next(J(m,n,q),z)$ можем да представим така:

$$Next(J(m,n,q),z) = \begin{cases} \frac{z.2^q}{2^{(z)0}}, & \text{ако } (z)_m = (z)_n \\ 2z, & \text{ако } (z)_m \neq (z)_n. \end{cases}$$

Така проверихме верността на следното

Твърдение 2.6. За всеки фиксиран оператор I, функцията $\lambda z.Next(I,z)$ е примитивно рекурсивна.

Оттук ще следва, че и функцията $Step$ ще е примитивно рекурсивна. Да видим:

Твърдение 2.7. Функцията $Step\colon \mathbb{N} \longrightarrow \mathbb{N}$ на всяка МНР програма P е примитивно рекурсивна.

Доказателство. Да фиксираме програмата $P\colon I_0, \ldots, I_k$ и да препишем представянето (2.6) за $step$ посредством $Next$. Ще получим

$$Step(z) = \begin{cases} Next(I_0,z), & \text{ако } z>0 \ \&\ (z)_0 = 0 \\ \ldots\ldots\ldots\ldots & \\ Next(I_k,z), & \text{ако } z>0 \ \&\ (z)_0 = k \\ z, & \text{ако } z>0 \ \&\ (z)_0 > k \\ 0, & \text{ако } z = 0. \end{cases}$$

Виждаме, че $Step$ се дефинира с разглеждане на случаи и следователно е примитивно рекурсивна, съгласно *Твърдение* 1.10 и горното *Твърдение* 2.6. □

Вече сме всъстояние да покажем твърдението, което беше основна цел на настоящия раздел, а именно:

Твърдение 2.8. Всяка изчислима функция е частично рекурсивна.

Доказателство. Да вземем произволна n-местна изчислима функция f. Това означава, че съществува МНР програма P: $I_0, \ldots, I_k$, такава че за всички естествени $x_1, \ldots, x_n, y$ е изпълнено:

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

където по дефиниция

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

Да означим с $t(\bar{x})$ функцията, която дава минималния брой тактове, за които програмата P, извикана върху $(\bar{x}, 0, 0, \ldots)$, попада в заключително състояние (ако въобще спре), и съответно $t(\bar{x})$ е недефинирана, ако P не спре върху $(\bar{x}, 0, 0, \ldots)$. Тогава $t(x_1, \ldots, x_n)$ можем да представим така:

$$t(x_1, \ldots, x_n) \simeq \mu t[(Step^t(\delta(0, x_1, \ldots, x_n, 0, 0, \ldots)))_0 > k]$$

$$\simeq \mu t[(Step^*(t, \delta(0, x_1, \ldots, x_n, 0, 0, \ldots)))_0 > k].$$

По определение $\delta(0, x_1, \ldots, x_n, 0, 0, \ldots) = p_1^{x_1}. \ \ldots \ .p_n^{x_n}$ и следователно функцията $d(\bar{x}) = \delta(0, \bar{x}, 0, 0, \ldots))$ е примитивно рекурсивна. Освен това итерацията $Step^*$ на функцията $Step$ също е примитивно рекурсивна, съгласно *Твърдение* 1.17 и *Твърдение* 2.7. Тогава функцията $t(\bar{x})$, която можем да препишем като

$$t(\bar{x}) \simeq \mu t[(Step^*(t, d(\bar{x})))_0 > k]. \tag{2.8}$$

ще е частично рекурсивна. Финално, за f можем да запишем:

$$f(\bar{x}) \simeq (Step^*(t(\bar{x}), d(\bar{x})))_1 \tag{2.9}$$

и следователно f също е частично рекурсивна. □

Сега събираме заедно *Твърдение* 2.5 и *Твърдение* 2.8, за да получим следния важен резултат.

Теорема 2.1. (Теорема за еквивалентност) Една функция е частично рекурсивна точно тогава, когато е изчислима.

Като непосредствено следствие от горната теорема получаваме следното любопитно наблюдение:

Следствие 2.1. Всяка частично рекурсивна функция може да бъде получена с прилагане на една единствена минимизация.

Доказателство. Ако f е частично рекурсивна, то съгласно горната теорема тя е изчислима, и значи може да се представи във вида (2.9). В това представяне всички функции, с изключение на $t(\bar{x})$, са примитивно рекурсивни, а самата $t(\bar{x})$ се получава от примитивно рекурсивни с една единствена минимизация, както е видно от равенството (2.8).

□

2.3.3 Тезис на Чърч-Тюринг

Теоремата за еквивалентност има и важно методологическо значение. Това, че съвпадат два класа от функции, определени посредством два принципно различни изчислителни модела ни говори, че тези класове не са случайни. Тази теорема е аргумент в подкрепа на *Тезиса на Тюринг*, който в първоначалния си вариант, изказан през 1936 г. от Тюринг, гласи: една функция е алгоритмично изчислима (в широк, неформален смисъл) точно когато е изчислима с машина на Тюринг. По същото време, независимо от Тюринг, и Чърч формулира свой *Тезис на Чърч*, който твърди подобно нещо, само че за λ-определимите функции, въведени от него.

Впоследствие възникват и много други изчислителни модели, за които се доказва, че функциите, определими във всеки един от тях съвпадат с функциите, изчислими с машини на Тюринг. С други думи, всеки независим опит да се въведе формално понятие за алгоритмично изчислима функция води до изчислимостта по Тюринг. Затова всички тези модели се наричат *тюрингово пълни.*

До ден днешен никой не е посочил функция, която да е изчислима в някакъв естествен, общоприет смисъл, но да не е изчислима с машина на Тюринг. Този факт, заедно с казаното по-горе, дава основание на хората, които се занимават с Теоретична информатика, да се обединят около следното твърдение, което е прието да се нарича

Тезис на Чърч-Тюринг. Една функция е алгоритмично изчислима тогава и само тогава, когато е изчислима с машина на Тюринг.

Разбира се, Тезисът на Чърч-Тюринг е *хипотеза*, която никога няма да бъде доказана, по простата причина, че в неговата формулировка участва нематематическото понятие "алгоритмично изчислима функция".

Глава 3

Основни теореми в Теория на изчислимостта

3.1 Универсална функция

Ще започнем с дефиницията на важното понятие за *универсална функция (УФ)*. Нашата непосредствена задача ще бъде да докажем, че класът $\mathcal{C}_n$ на всички n-местни изчислими функции има универсална функция.

Определение 3.1. Ще казваме, че функцията $U(a, x_1, \ldots, x_n)$ е *универсална* за класа от функции $\mathcal{K} \subseteq \mathcal{F}_n$, 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) бихме могли 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 $\lambda\bar{x}.U(a, \bar{x}) \in \mathcal{K}$.

При фиксирано $n \geq 1$ да положим

$$\Phi_n(a, x_1, \ldots, x_n) \simeq \varphi_a^{(n)}(x_1, \ldots, x_n) \tag{3.1}$$

за всяко $a \in \mathbb{N}$ и $\bar{x} \in \mathbb{N}^n$.

Оттук по *Твърдение* 2.4 получаваме, че за всяка функция f:

$$f \in \mathcal{C}_n \implies f = \varphi_a^{(n)} \text{ за някое } a \implies f = \lambda\bar{x}.\Phi_n(a, \bar{x}) \text{ за някое } a.$$

Следователно за Φ_n е в сила условието 1) от дефиницията за УФ за класа $\mathcal{C}_n$. Условието 2) е изпълнено автоматично, защото

$$\lambda\bar{x}.\Phi_n(a,\bar{x}) \stackrel{\text{деф}}{=} \varphi_a^{(n)},$$

която е в $\mathcal{C}_n$ по смисъла на определението си. Никак не е автоматична, обаче, проверката, че Φ_n удовлетворява и условието 0). Фактът, че Φ_n е изчислима е един от най-важните резултати в Теория на изчислимостта и обикновено се нарича *теорема за универсалната функция*.

3.1.1 Теорема за универсалната функция

Теорема 3.1. (Теорема за универсалната функция) За всяко $n \geq 1$ функцията Φ_n е изчислима.

Доказателство. Да фиксираме някакво $n \geq 1$. Ще покажем, че Φ_n е частично рекурсивна функция, което ще означава, че Φ_n е изчислима, съгласно теоремата за еквивалентност.

Да отбележим, че директното доказателство на изчислимостта на Φ_n се свежда на практика до построяването на *универсалната програма* за МНР — нещо, което технически е доста по-трудно за реализиране, защото трябва да програмираме на езика за МНР, а той далеч не е най-удобният език за тази цел.

Да разпишем отново определението на Φ_n:

$$\Phi_n(a,\bar{x}) \simeq y \iff \varphi_a^{(n)}(\bar{x}) \simeq y \iff P_a \text{ спира върху } \bar{x} \text{ с резултат } y.$$

С други думи, $\Phi_n(a,\bar{x})$ връща резултата от работата на програмата P_a върху вход $\bar{x}$. За да опишем формално как работи P_a върху $\bar{x}$ ще въведем следната спомагателна функция Q_n:

$$Q_n(a,\bar{x},t) \stackrel{\text{деф}}{=} \text{кода на конфигурацията, която се получава след } t \text{ такта от работата на } P_a \text{ върху } \bar{x}. \tag{3.2}$$

С примитивна рекурсия по t ще покажем, че Q_n е примитивно рекурсивна. Но как да изразим $Q_n(a,\bar{x},t+1)$ чрез $Q_n(a,\bar{x},t)$? Ясно е, че конфигурацията $Q_n(a,\bar{x},t+1)$ е "следващата конфигурация" след $Q_n(a,\bar{x},t)$. Тук на помощ ни идва функцията $next$ от по-горе, и по-точно, нейната "цифровизираната" версия $Next$, която въведохме с равенството (2.7). Преразказано, $Next(I,z)$ дава кода на конфигурацията, която се получава, когато приложим инструкцията I към конфигурацията с код z. В нашия случай ще искаме да извикаме $Next$ върху конфигурацията на

стъпка t, т.е. при $z = Q_n(a, \bar{x}, t)$. Инструкцията, която ще прилагаме към тази конфигурация, е текущата инструкция на стъпка t. Нейният адрес е точно $(Q_n(a, \bar{x}, t))_0$, а (кодът на) самата инструкция няма проблем да възстановим от кода a на програмата P_a.

За да направим нещата точни, ще трябва да въведем още една — този път изцяло числова функция от серията "$next$" — да я наречем $NEXT$. Тази функция, извикана върху кода $\beta(I)$ на инструкцията I и кода на (финитната) конфигурация $(l, \widetilde{x})$ връща $Next(I, \delta(l, \widetilde{x}))$. Ако си представяме $\delta(l, \widetilde{x}))$ като z, то ще имаме

$$NEXT(\beta(I), z) = Next(I, ((z)_0, (z)_1, (z)_2, \ \dots\ )).$$

Тъй като 0 не е код на конфигурация, можем да положим $NEXT(\beta(I), 0) \stackrel{\text{деф}}{=} 0$ за всяка инструкция I.

В доказателството на *Твърдение* 2.6 видяхме, че за всяка от четирите вида инструкции I имаме следните представяния за $Next(I, z)$ при $z > 0$:

$$Next(S(n), z) = 2zp_n$$

$$Next(Z(n), z) = \frac{2z}{p_n^{(z)_n}}$$

$$Next(T(m, n), z) = \frac{2zp_m^{(z)_n}}{p_m^{(z)_m}}$$

$$Next(J(m, n, q), z) \ = \ \begin{cases} \frac{z.2^q}{2^{(z)_0}}, & \text{ако } (z)_m = (z)_n \\ 2z, & \text{ако } (z)_m \neq (z)_n. \end{cases}$$

Следователно $NEXT$ можем да се препишем така:

$$NEXT(\beta(I), z) = \begin{cases} 2z.p_n, & \text{ако } \ I = S(n) \\ \frac{2z}{p_n^{(z)_n}}, & \text{ако } \ I = Z(n) \\ \frac{2z.p_m^{(z)_n}}{p_m^{(z)_m}}, & \text{ако } I = T(m, n) \\ \frac{z.2^q}{2^{(z)_0}}, & \text{ако } \ I = J(m, n, q) \ \& \ (z)_m = (z)_n \\ 2z, & \text{ако } \ I = J(m, n, q) \ \& \ (z)_m \neq (z)_n. \end{cases}$$

Разбира се, добре е и първия аргумент $\beta(I)$ да заменим с някаква буква, да кажем i. Правим го веднага:

$$NEXT(i, z) = \begin{cases} 2z.p_n, & \text{ако } \ i = \beta(S(n)) \\ \frac{2z}{p_n^{(z)_n}}, & \text{ако } \ i = \beta(Z(n)) \\ \frac{2z.p_m^{(z)_n}}{p_m^{(z)_m}}, & \text{ако } \ i = \beta(T(m, n)) \\ \frac{z.2^q}{2^{(z)_0}}, & \text{ако } \ i = \beta(J(m, n, q)) \ \& \ (z)_m = (z)_n \\ 2z, & \text{ако } \ i = \beta(J(m, n, q)) \ \& \ (z)_m \neq (z)_n. \end{cases}$$

Остана една последна стъпка — в дясната част на горното равенство да премахнем всички променливи, различни от i и z. За целта трябва да имаме пред себе си определението (2.1) на кода $\beta(I)$, който дефинирахме с различен остатък по модул 4, в зависимост от вида на инструкцията I:

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

Ясно е, че ще ни е нужно да изразим компонентите на инструкцията I (числата m, n или q) чрез нейния код $\beta(I)$. Знаем, че няма пречка да го направим, защото при дефинирането на кодирането β се погрижихме то да е ефективно. Да се убедим че това изразяване става с примитивно рекурсивни функции. Разглеждаме четирите случая за I:

— ако $i = \beta(S(n))$, т.е. $i = 4(n-1)$, то $n = [\frac{i}{4}] + 1$

— ако $i = \beta(Z(n))$, т.е. $i = 4(n-1)+1$, то $n = [\frac{i}{4}] + 1$

— ако $i = \beta(T(m,n))$, т.е. $i = 4\Pi(m-1,n-1)+2$,
то $m = L([\frac{i}{4}]) + 1$ и $n = R([\frac{i}{4}]) + 1$

— ако $i = \beta(J(m,n,q))$, т.е. $i = 4\Pi_3(m-1,n-1,q)+3$,
то $m = J^3_1([\frac{i}{4}]) + 1$, $n = J^3_2([\frac{i}{4}]) + 1$ и $q = J^3_3([\frac{i}{4}])$.

Да препишем финално дефиницията на $NEXT(i,z)$, като си представяме, че навсякъде в нея буквите m, n и q са заместени с горните изрази:

$$NEXT(i,z) = \begin{cases}
2z.p_n, & \text{ако } z>0 \ \&\ i \equiv 0 \ (mod\ 4) \\
\frac{2z}{p_n^{(z)n}}, & \text{ако } z>0 \ \&\ i \equiv 1 \ (mod\ 4) \\
\frac{2z.p_m^{(z)n}}{p_m^{(z)m}}, & \text{ако } z>0 \ \&\ i \equiv 2 \ (mod\ 4) \\
\frac{z.2^q}{2^{(z)0}}, & \text{ако } z>0 \ \&\ i \equiv 3 \ (mod\ 4) \ \&\ (z)_m = (z)_n \\
2z, & \text{ако } z>0 \ \&\ i \equiv 3 \ (mod\ 4) \ \&\ (z)_m \neq (z)_n \\
0, & \text{ако } z=0.
\end{cases}$$

Разбира се, по-горе всички деления са целочислени, следователно $NEXT$ е примитивно рекурсивна.

Сега да се върнем на функцията $Q_n(a,\bar{x},t)$, която дефинирахме с равенството (3.2). $Q_n(a,\bar{x},t)$ даваше кода на конфигурацията на t-та стъпка от работата на P_a върху $\bar{x}$. Ще покажем, че тя е примитивно рекурсивна, като напишем примитивно рекурсивна схема за нея. Рекурсията ще бъде по последната променлива t, разбира се. Базовият случай е ясен:

$$Q_n(a,\bar{x},0) = \delta(0,x_1,\dots,x_n,0,\dots) \overset{\text{деф}}{=} p_1^{x_1} \ \dots \ p_n^{x_n}.$$

Нека конфигурацията на стъпка t е $(l, \widetilde{x})$, т.е. $Q_n(a, \bar{x}, t) = \delta(l, \widetilde{x})$. Нека още $l \leq lh(a)$. Адресът l на инструкцията, която трябва да се изпълни на тази стъпка, е $(Q_n(a, \bar{x}, t))_0$, а самата инструкция I_l възстановяваме от кода a на програмата P_a. Ако P_a е програмата $I_0, \ldots, I_k$ (тук $k = lh(a)$), то по определение (2.2) за нейния код $\gamma(P_a)$ имаме:

$$\gamma(P_a) \overset{\text{деф}}{=} a = \tau(\langle \beta(I_0), \ldots, \beta(I_k) \rangle).$$

Тогава $\beta(I_l)$ просто ще е l-тият елемент от редицата с код a, който се даваше от функцията $mem(a, l)$, дефинирана с (1.8).

Значи можем да запишем:

$$Q_n(a, \bar{x}, t+1) = \begin{cases} NEXT(mem(a, l), Q_n(a, \bar{x}, t)), & \text{ако } l \leq lh(a) \\ Q_n(a, \bar{x}, t), & \text{ако } l > lh(a) \end{cases}$$

$$= \begin{cases} NEXT(mem(a, (Q_n(a, \bar{x}, t))_0), Q_n(a, \bar{x}, t)), & \text{ако } (Q_n(a, \bar{x}, t))_0 \leq lh(a) \\ Q_n(a, \bar{x}, t), & \text{ако } (Q_n(a, \bar{x}, t))_0 > lh(a). \end{cases}$$

Така получаваме примитивно рекурсивната схема

$$\begin{cases} Q_n(a, \bar{x}, 0) = p_1^{x_1} \ \ldots \ p_n^{x_n} \overset{\text{деф}}{=} g(x_1, \ldots, x_n) \\ Q_n(a, \bar{x}, t+1) = G(a, \bar{x}, t, Q_n(a, \bar{x}, t)), \end{cases}$$

където с G сме означили функцията

$$G(a, \bar{x}, t, z) = \begin{cases} NEXT(mem(a, (z)_0), z), & \text{ако } (z)_0 \leq lh(a) \\ z, & \text{ако } (z)_0 > lh(a). \end{cases}$$

Понеже g и G са примитивно рекурсивни, то и Q_n ще е примитивно рекурсивна.

От дефиницията на Q_n се вижда, че ако програмата P_a спре върху вход $\bar{x}$, то това ще стане за брой стъпки $t_n(a, \bar{x})$, където

$$t_n(a, \bar{x}) \simeq \mu t[(Q_n(a, \bar{x}, t))_0 > lh(a)].$$

Тогава очевидно t_n е частично рекурсивна функция.

Резултатът y от работата на P_a върху $\bar{x}$ по дефиниция е в първи регистър, т.е.

$$y \simeq (Q_n(a, \bar{x}, t_n(a, \bar{x})))_1.$$

Следователно функцията $\varphi_a^{(n)}$, която P_a пресмята, се изразява чрез Q_n по следния начин:

$$\varphi_a^{(n)}(\bar{x}) \simeq (Q_n(a, \bar{x}, t_n(a, \bar{x})))_1.$$

(Тук отчитаме, че ако P_a не спре върху $\bar{x}$, то $t_n(a,\bar{x})$ и $\varphi_a^{(n)}(\bar{x})$ са недефинирани; следователно и двете страни на горното условно равенство ще са недефинирани, и значи то отново ще е в сила.)

От това равенство и от дефиницията (3.1) на Φ_n получаваме финално

$$\Phi_n(a,\bar{x}) \simeq (Q_n(a,\bar{x},t_n(a,\bar{x})))_1.$$

Следователно Φ_n е частично рекурсивна, което значи и изчислима, съгласно *Твърдение* 2.5. □

Забележка. От това, че $\Phi_n(a,\bar{x})$ е частично рекурсивна, можем да заключим, че такива ще са и функциите $\varphi_a^{(n)} = \lambda\bar{x}.\Phi_n(a,\bar{x})$ за всяко фиксирано a. Но това означава, че всъщност всички изчислими функции са частично рекурсивни, което е точно *Твърдение* 2.8. Излиза, че можеше да го получим и като следствие от горната теорема. Така е, но ние предпочетохме да докажем теоремата за еквивалентност независимо от теоремата за универсалната функция, за да не смесваме двете явления — еквивалентността на двата подхода и съществуването на универсална функция. Освен това цялата техническа работа, която свършихме при доказателството на първата теорема, беше използвана в доказателството на втората, така че трудът ни не беше напразен.

3.1.2 Теорема за нормален вид на Клини

Следващото твърдение, известно като *теорема за нормален вид на Клини*, дава едно стандартизирано представяне на всяка изчислима функция, което се оказва полезно за много приложения.

Теорема 3.2. (Теорема за нормален вид на Клини) За всяко $n \geq 1$ съществува примитивно рекурсивна функция $T_n(a,\bar{x},z)$, такава че за всички естествени a и $\bar{x}$:

$$\varphi_a^{(n)}(\bar{x}) \simeq L(\mu z[T_n(a,\bar{x},z) = 0]).$$

Доказателство. Да означим с $A_n(a,\bar{x},y,t)$ следния предикат:

$$A_n(a,\bar{x},y,t) \overset{\text{деф}}{\iff} P_a \text{ спира върху вход } \bar{x} \text{ за } \leq t \text{ такта с резултат } y.$$

A_n е примитивно рекурсивен, защото можем да го запишем още така:

$$A_n(a,\bar{x},y,t) \iff (Q_n(a,\bar{x},t))_0 > lh(a) \text{ и } (Q_n(a,\bar{x},t))_1 = y.$$

Сега дефинираме следния предикат $T_n(a, \bar{x}, z)$, който често се нарича *предикат на Клини*:

$$T_n(a, \bar{x}, z) \overset{\text{деф}}{\iff} A_n(a, \bar{x}, L(z), R(z)).$$

T_n формално е предикат, но ние ще го отъждествяваме с характеристичната му функция

$$\chi_{T_n}(a, \bar{x}, z) = \begin{cases} 0, & \text{ако } T_n(a, \bar{x}, z) \\ 1, & \text{ако } \neg T_n(a, \bar{x}, z). \end{cases}$$

Да се убедим, че T_n е търсената от нас функция. За целта при фиксирано a да означим с g_a следната функция:

$$g_a(\bar{x}) \simeq L(\mu z[T_n(a, \bar{x}, z) = 0]).$$

Ще покажем, че $g_a = \varphi_a^{(n)}$. За целта отново ще се възползваме от *Задача* 1.1, според която е достатъчно да съобразим, че

$$g_a \subseteq \varphi_a^{(n)} \quad \text{и} \quad Dom(\varphi_a^{(n)}) \subseteq Dom(g_a).$$

1) $g_a \subseteq \varphi_a^{(n)}$: Нека $g_a(\bar{x}) \simeq y$. Тогава в частност $g_a(\bar{x})$ е дефинирано и значи съществува $z : T_n(a, \bar{x}, z) = 0$ и $L(z) = y$. От дефиницията на T_n ще имаме, че е вярно $A_n(a, \bar{x}, y, R(z))$, което означава, че $\varphi_a^{(n)}(\bar{x}) \simeq y$.

2) $Dom(\varphi_a^{(n)}) \subseteq Dom(g_a)$: Нека за произволно $\bar{x} \in \mathbb{N}^n$ е вярно, че $\bar{x} \in Dom(\varphi_a^{(n)})$, т.е. $!\varphi_a^{(n)}(\bar{x})$. Тогава за някои y и t ще е изпълнено

$$A_n(a, \bar{x}, y, t),$$

а оттук и $T_n(a, \bar{x}, \Pi(y, t))$. Получихме, че съществува z, за което $T_n(a, \bar{x}, z) = 0$ и следователно $!g_a(\bar{x})$.

С това показахме, че $g_a = \varphi_a^{(n)}$, с други думи

$$\varphi_a^{(n)}(\bar{x}) \simeq L(\mu z[T_n(a, \bar{x}, z) = 0]).$$

□

Ще завършим този раздел с едно интересно твърдение, което е непосредствено следствие от теоремата за универсалната функция.

Твърдение 3.1. Съществува частично рекурсивна функция, която не може да бъде продължена до рекурсивна.

Доказателство. Ще конструираме едноместна функция f с това свойство, като тръгнем от "най-сложната" функция — универсалната за едноместните изчислими функции $\Phi_1(a, x)$. Да вземем

$$f(x) \overset{\text{деф}}{\simeq} \Phi_1(x, x) + 1.$$

По теоремата за универсалната функция, f е изчислима. Ще покажем, че тя не може да се продължи до рекурсивна функция. Наистина, да допуснем, че съществува рекурсивна g, такава че $f \subseteq g$. Функцията g е рекурсивна, което ще рече и изчислима, и значи тя има индекс. Нека a е някакъв индекс на g. Понеже $g = \varphi_a$, то φ_a е тотална и в частност, $\varphi_a(a)$ е дефинирана. Но тогава е дефинирана и $f(a)$, тъй като

$$f(a) \simeq \Phi_1(a, a) + 1 \simeq \varphi_a(a) + 1.$$

От $f \subseteq g$ и $!f(a)$ ще имаме $f(a) = g(a)$, което обаче е невъзможно, тъй като съгласно избора на f имаме, че

$$f(a) = \varphi_a(a) + 1 = g(a) + 1 \neq g(a).$$

Забележка. Ако си мислите, че f не може да бъде продължена до рекурсивна, защото може да има много големи стойности, приложете горните разсъждения към 0-1-функцията $f(x) = \bar{sg}(\Phi_1(x, x))$. Пак ще стигнете до противоречие. □

3.1.3 Задачи

В следващите задачи ще покажем, че два интересни класа от функции имат универсална функция. Доказателствата, че това е така, ще са несравнимо по-лесни от доказателството на факта, че изчислимите функции имат УФ (теоремата за универсалната функция). Една от причините за това е, че функциите от тези два класа могат да бъдат *кодирани*, а не просто *номерирани*, както беше при изчислимите функции.

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

Задача 3.1. 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

$$p_a \stackrel{\text{деф}}{=} \text{полинома с код } a. \tag{3.3}$$

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:

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

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 (3.3). 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 3.2. Нека

$\mathcal{SUP} = \{f \mid f$ 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 3.3. 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* 3.2 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

$$f(x_1, \ldots, x_n) \overset{\text{деф}}{=} U(x_1, x_1, \ldots, x_n) + 1 \tag{3.4}$$

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 (3.4) на f, $f(e, \ldots, e) = U(e, e, \ldots, e)+1$. Противоречие. □

Задача 3.4. (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.

3.1.4 Диагонален метод на Кантор

Ще формулираме няколко твърдения, свързани с универсални функции, които се доказват с конструкция, известна като *диагонален метод на Кантор.* Въпреки че доказателствата им са елементарни, смисълът на тези твърдения е важен компютърната наука.

Да напомним определението за универсална функция от началото на тази глава: функцията $U(a, \bar{x})$ ще наричаме *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 (поне едно) $a \in \mathbb{N}$: $f = \lambda\bar{x}.U(a, \bar{x})$;

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

От условие 1) е ясно, че за да притежава универсална функция, класът $\mathcal{K}$ трябва да е най-много изброим. Всъщност за всеки такъв клас има функция, за която са изпълнени горните изисквания 1) и 2). За да я конструираме, да вземем произволно изброяване

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

на функциите от класа $\mathcal{K}$ (в изброяването може да има и повторения). 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) за тази функция U следват директно от дефиницията ѝ.

Това, което е наистина важно в горното определение, е изчислимостта на U.

Да означим с $\mathcal{R}_n$ класа на всички n-местни рекурсивни функции. Оказва се, че за разлика от класа на частично рекурсивните (изчислимите) функции, този клас няма универсална функция.

Твърдение 3.2. За всяко $n \geq 1$ класът $\mathcal{R}_n$ на n-местните рекурсивни функции няма универсална функция.

Доказателство. Първо ще разгледаме случая $n = 1$, с който ще илюстрираме и класическата диагонална конструкция на Кантор. Да допуснем, че $U(a, x)$ е универсална за $\mathcal{R}_1$. Тогава U е изчислима. Освен това тя е тотална, защото за всяко фиксирано a функцията $\lambda x.U(a, x)$ е от класа $\mathcal{R}_1$, и следователно $!U(a, x)$ за всяко x. Но изчислима + тотална означава рекурсивна, т.е. дотук имаме, че U е рекурсивна функция.

Нека за всяко фиксирано a с f_a да означим функцията

$$f_a = \lambda x.U(a,x).$$

От условие 1) на дефиницията за УФ имаме, че редицата

$$f_0, f_1, \ldots, f_a, \ldots \tag{3.5}$$

съдържа всички едноместни рекурсивни функции. Ще построим нова функция $d(x)$, която хем ще е рекурсивна, хем няма да се среща в редицата (3.5), което ще е търсеното противоречие. $d(x)$ е свързана с *диагонала* на таблицата от стойностите на функциите от горната редица. Да

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

положим

$$d(x) \;=\; f_x(x) + 1$$

за всяко $x \in \mathbb{N}$. Тази функция е рекурсивна, защото можем да я препишем като $d(x) = U(x,x) + 1$ (тук използваме, че по допускане U е рекурсивна). Но щом е рекурсивна, тя е изброена в горната редица (3.5) поне веднъж, т.е. съществува a, такова че $d = f_a$. Но тогава би трябвало $d(x) = f_a(x)$ за всяко x, което при $x = a$ очевидно се нарушава, защото съгласно избора на d имаме

$$d(a) = f_a(a) + 1.$$

По подобен начин разсъждаваме за произволно n. Допускаме, че класът $\mathcal{R}_n$ има УФ $U(a, x_1, \ldots, x_n)$ и полагаме

$$d(x_1, \ldots, x_n) \;=\; U(x_1, x_1, x_2, \ldots, x_n) + 1.$$

Разбира се, тази функция U също е рекурсивна. Тогава и d ще е рекурсивна, което означава, че $d = \lambda \bar{x}.U(a,\bar{x})$ за някое a. В частност, в точката $\underbrace{(a,\ldots,a)}_{n \text{ пъти}}$ ще имаме $d(a,\ldots,a) \;=\; U(a,a,\ldots,a)$, което противоречи на дефиницията на d, според която

$$d(a,\ldots,a) \overset{\text{деф}}{=} U(a,a,\ldots,a) + 1.$$

□

Това твърдение ни казва, че е безсмислено да се опитваме да пишем език за програмиране, чийто програми завършват при всеки вход, и който е такъв, че всяка тотална изчислима функция е изчислима и с програма на този език (без второто условие въпросният език може да е съвсем тривиален — примерно в него да няма цикли). Наистина, ако такъв език съществуваше, всички рекурсивни функции (и само те) щяха да се пресмятат с програмите от този език. Но тогава щяхме да можем да построим УФ за тези функции, следвайки идеята от доказателството на теоремата за универсалната функция за нашия език за МНР. Този факт влиза в противоречие с горното *Твърдение* 3.2.

Да означим с $\mathcal{PR}_n$ класа на n-местните примитивно рекурсивни функции. За разлика от рекурсивните функции, те имат универсална функция и ние ще я построим по-нататък в курса. Следващото твърдение казва, обаче, че никоя УФ за $\mathcal{PR}_n$ не може да бъде примитивно рекурсивна.

Твърдение 3.3. Нека U е универсална функция за класа $\mathcal{PR}_n$ на n-местните примитивно рекурсивни функции. Тогава U не е примитивно рекурсивна.

Доказателство. Следваме схемата от доказателството на предното твърдение. Допускаме, че U е примитивно рекурсивна и отново конструираме "диагоналната" функция

$$d(x_1, \ldots, x_n) = U(x_1, x_1, x_2, \ldots, x_n) + 1.$$

Тя също е примитивно рекурсивна и значи $d = \lambda\bar{x}.U(a, \bar{x})$ за някое a. В частност, $d(a, \ldots, a) = U(a, a, \ldots, a)$, което влиза в противоречие с това, което имаме по дефиниция за d: $d(a, \ldots, a) = U(a, a, \ldots, a) + 1$. □

Ако се питате как така същото диагонално разсъждение не може да се използва, за да се докаже, че никоя УФ за класа на изчислимите функции не е изчислима, краткият отговор е: защото изчислимите функции са частични.

Наистина, да дефинираме d като

$$d(x) \simeq \Phi_1(x, x) + 1$$

(разсъждавайки за $n = 1$ за по-просто). Функцията d е изчислима, съгласно теоремата за универсалната функция. Следователно $d = \lambda x.\Phi_1(a, x)$ за поне едно a. Оттук при $x = a$ ще е вярно, че

$$d(a) \simeq \Phi_1(a, a),$$

и едновременно с това от дефиницията на d ще имаме

$$d(a) \simeq \Phi_1(a,a) + 1.$$

Тези равенства, обаче, не си противоречат, точно защото са *условни,* и могат да са в сила едновременно, когато $\neg!\Phi_1(a,a)$. Нещо повече, понеже знаем, че Φ_1 *е* изчислима, можем със сигурност да твърдим, че $\Phi_1(a,a)$ не е дефинирана за никой индекс a на "диагоналната" функция d.

3.2 S_n^m-теорема

Тази теорема е известна още като *теорема за параметризацията* или *Лема за транслация.* По-нататък ще стане ясно откъде идват тези имена.

За да си изясним "на първо четене" смисъла на S_n^m-теоремата, ще започнем с едно частно наблюдение, което после ще обобщим.

Преди това, обаче, да си напомним едно означение, което въведохме преди. При фиксирани $n \geq 1$ и $a \in \mathbb{N}$:

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

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

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

Ако $f = \varphi_a^{(n)}$, то a нарекохме *индекс* на f.

Сега да вземем произволна изчислима функция на два аргумента $f(a,x)$. Ако за момент си представим, че първият ѝ аргумент a е фиксиран, получаваме едноместната функция

$$f_a = \lambda x.f(a,x).$$

Тази функция, разбира се, също е изчислима и значи за нея съществува индекс b, такъв че $f_a = \varphi_b$. Излезе, че

$$\forall a \exists b \; f_a = \varphi_b,$$

което означава (с използване на аксиомата за избора), че съществува функция, да кажем h, такава че за всяко a

$$f_a = \varphi_{h(a)}.$$

Това, което в добавка към това наблюдение ни дава S_n^m-теоремата е, че функцията h може да се избере така, че да бъде рекурсивна (и дори примитивно рекурсивна). От доказателството ще се види още, че функцията h се *конструира* (т.е. доказателството на теоремата е конструктивно).

За случая на двуместна изчислима функция $f(a,x)$ S_n^m-теоремата казва, че съществува примитивно рекурсивна функция h, такава че за всяко a и x:

$$\varphi_{h(a)}(x) \simeq f(a,x).$$

Този резултат би могъл да се обобщи по два начина. Първият е директен — вместо $f(a,x)$ да разглеждаме $f(a_1,\ldots,a_m,x_1,\ldots,x_n)$, т.е. параметрите на конструкцията да бъдат $a_1,\ldots,a_m$.

Вторият начин е да направим т. нар. *равномерно* обобщение. То се състои в следното: щом $f(a,x)$ е изчислима, значи можем да си я мислим във вида $\varphi_e^{(2)}(a,x)$ за някое e, с други думи, да разглеждаме f, зададена чрез своя индекс e. Е, оказва се, че "в играта" можем да включим и този индекс e. С други думи, вярно е не просто, че

$$\forall e \exists h\ \varphi_{h(a)} = \lambda x.\varphi_e^{(2)}(a,x),$$

а нещо доста по-силно — че функцията h може да се избере отнапред и да е *обща* за всички индекси e, т.е. горните два квантора могат да се разменят:

$$\exists h \forall e\ \varphi_{h(e,a)} = \lambda x.\varphi_e^{(2)}(a,x).$$

3.2.1 S_n^m-теорема

Ето и най-общата формулировка на S_n^m-теоремата:

Теорема 3.3. (S_n^m-теорема) Нека $m \geq 1, n \geq 1$. Съществува примитивно рекурсивна функция S_n^m, за която е изпълнено:

$$\varphi^{(n)}_{S_n^m(e,a_1,\ldots,a_m)}(x_1,\ldots,x_n) \simeq \varphi_e^{(m+n)}(a_1,\ldots,a_m,x_1,\ldots,x_n)$$

за всички естествени $e,a_1,\ldots,a_m,x_1,\ldots,x_n$.

Доказателство. За начало ще разгледаме случая $m=n=1$. Трябва да построим примитивно рекурсивна функция S_1^1, такава че за всяко e,a и x:

$$\varphi_{S_1^1(e,a)}(x) \simeq \varphi_e^{(2)}(a,x).$$

При фиксирани e и a, нека Q е МНР програма, която пресмята функцията $\lambda x.\varphi_e^{(2)}(a,x)$. Как работи Q? При вход $(x,0,0,\ldots)$ тя симулира P_e върху нейния вход $(a,x,0,0,\ldots)$

За целта най-напред Q трябва да прехвърли x във втория регистър, а в първия да зареди a. Това става с редицата от инструкции

$$X_2 := X_1,\ X_1 := 0,\ \underbrace{X_1 := X_1 + 1,\ \ldots\ , X_1 := X_1 + 1}_{a \text{ пъти}}$$

След тези инициализации Q трябва да започне да изпълнява инструкциите на P_e, които, обаче, преди това са били *преадресирани*. Това се налага заради тези $a+2$ на брой оператора, които слагаме в началото на програмата. Заради тях всеки оператор за преход $J(m,n,q)$ на P_e трябва да се замени с $J(m,n,q+a+2)$.

Нека за определеност $P_e\colon I_0,\ldots,I_k$. За всяко $l=0,\ldots,k$ нека

$$I_l^{'} = \begin{cases} I_l, & \text{ако } I_l \text{ е оператор за присвояване} \\ J(m,n,q+a+2), & \text{ако } I_l \text{ е } J(m,n,q). \end{cases}$$

Значи Q трябва да изглежда така:

$$T(2,1),\ Z(1),\ \underbrace{S(1),\ \ldots\ ,S(1),}_{a \text{ пъти}}\ \underbrace{I_0^{'},\quad I_1^{'},\quad \ldots\quad ,I_k^{'}}_{\text{преадресираните инстр. на } P_e}$$

Всъщност нашата програма Q зависи от двата параметъра e и a, затова нека да я означим като $Q_{e,a}$. Сега задачата ни се свежда до това да покажем, че кодът $\gamma(Q_{e,a})$ на програмата $Q_{e,a}$ зависи "гладко" (в случая — примитивно рекурсивно) от e и a. Тогава ако положим

$$S_1^1(e,a) = \gamma(Q_{e,a}),$$

то $S_1^1(e,a)$ ще е примитивно рекурсивна и едноместната функция, която програмата с код $S_1^1(e,a)$ пресмята, ще е точно тази, която се пресмята от $Q_{e,a}$, т.е. точно $\lambda x.\varphi_e^{(2)}(a,x)$. Така ще имаме, че за всяко x

$$\varphi_{S_1^1(e,a)}(x) \simeq \varphi_e^{(2)}(a,x).$$

И тъй като e и a са произволни, то горното условно равенство ще е изпълнено за всяко e, a и x, което е точно S_n^m-теоремата при $m=n=1$.

Сега вече имаме мотивация да се заемем с доказателството на примитивната рекурсивност на функцията $\lambda e,a.\gamma(Q_{e,a})$.

Да означим с tr ("tr" от *transform*) функцията, която преобразува кода на оператор за преход $J(m,n,q)$ по следния начин:

$$tr(\beta(J(m,n,q)),b) \;=\; \beta(J(m,n,q+b)).$$

Тази функция е примитивно рекурсивна, защото (като си спомним детайлите за код на оператор за преход (2.1)), можем да я запишем като

$$tr(z,b) \;=\; 4.\Pi_3(J_1^3([\frac{z}{4}]),J_2^3([\frac{z}{4}]),J_3^3([\frac{z}{4}])+b)+3.$$

Нека *prime* е функцията, която преобразува кода на всеки оператор I на P_e в кода на I'. Тази функция също е примитивно рекурсивна, защото

$$prime(z)=\begin{cases} z, & \text{ако } rem(4,z)<3\\ tr(z,a+2), & \text{ако } rem(4,z)=3.\end{cases}$$

Да напомним, че номерът k на последния оператор на $P_e: I_0,\ldots,I_k$ се дава от функцията lh, т.е. $k=lh(e)$. Освен това, ако I_l е l-тият оператор на P_e, то неговият код, съгласно определението (2.2), ще е $mem(e,l)$, и следователно кодът на I_l' ще е $prime(mem(e,l))$.

Да препишем отново редицата от инструкции на програмата $Q_{e,a}$:

$$T(2,1),\; Z(1),\; \underbrace{S(1),\;\ldots\;,S(1),}_{a\text{ пъти}}\; \underbrace{I_0',\quad I_1',\quad \ldots\quad ,I_k'}_{\text{преадресираните инстр. на } P_e}$$

Да означим с $f(e,a,l)$ функцията, която за всяко $l\le a+2+k$ дава кода на l-тия оператор на тази програма, по-точно нека

$$f(e,a,l)=\begin{cases}\text{кода на } l\text{-ия оператор на } Q_{e,a}, & \text{ако } l\le a+2+k\\ 0, & \text{ако } l>a+2+k.\end{cases}$$

Да съобразим, че и тази функция е примитивно рекурсивна. За целта да забележим, че за $l\in\{a+2,\ldots,a+2+k\}$ е вярно, че l-тият оператор на $Q_{e,a}$ е точно I'_{l-a-2}, и този оператор, съгласно това, което отбелязахме по-горе, е с код $prime(mem(e,l-a-2))$. Тогава за $f(e,a,l)$ имаме следната дефиниция с разглеждане на случаи:

$$f(e,a,l)=\begin{cases}\beta(T(2,1)), & \text{ако } l=0\\ \beta(Z(1)), & \text{ако } l=1\\ \beta(S(1)), & \text{ако } 2\le l<a+2\\ prime(mem(e,l-a-2)), & \text{ако } a+2\le l\le a+2+lh(e)\\ 0, & \text{ако } l>a+2+lh(e).\end{cases}$$

Оттук ясно се вижда, че f наистина е примитивно рекурсивна. Сега вече за кода $\gamma(Q_{e,a})$ на програмата $Q_{e,a}$ ще имаме, съгласно дефиницията (2.2) на код на програма:

$$\gamma(Q_{e,a})=\tau(\langle f(e,a,0),\;\ldots\;,f(e,a,a+2+lh(e))\rangle).$$

Но този израз е точно историята H_f на f, съгласно определение (1.9). Сега вече $\gamma(Q_{e,a})$ придобива вида

$$\gamma(Q_{e,a}) \;=\; H_f(e, a, a + 2 + lh(e)).$$

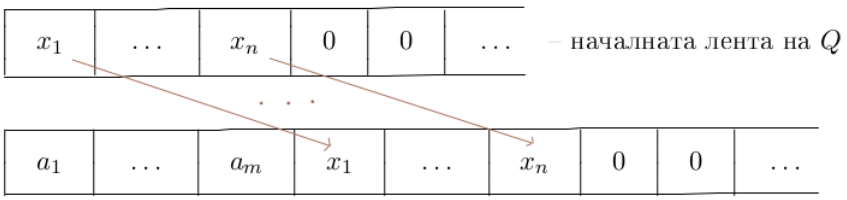
От *Твърдение* 1.25 знаем, че ако f е примитивно рекурсивна, то и нейната история H_f също ще е, откъдето финално получаваме, че функцията $\lambda e, a.\gamma(Q_{e,a})$ е примитивно рекурсивна.

С това приключва доказателството на S^m_n-теоремата за случая $m = n = 1$.

За произволни m и n разсъжденията са много подобни: при фиксирани $e, a_1, \ldots, a_m$ разглеждаме програмата $Q = Q_{e,a_1,\ldots,a_m}$, която пресмята функцията

$$\lambda x_1, \ldots, x_n.\varphi_e^{(m+n)}(a_1, \ldots, a_m, x_1, \ldots, x_n).$$

Този път началната конфигурация на Q е $(x_1, \ldots, x_n, 0, 0, \ldots)$ и тя работи така, както P_e работи върху $(a_1, \ldots, a_m, x_1, \ldots, x_n, 0, 0, \ldots)$, или сега картинката е:



началната лента на P_e

Значи Q трябва най-напред да прехвърли съдържанието на първите си n регистъра в регистри с номера $m + 1, \ldots, m + n$. Добре е да започне от последния към първия регистър (съобразете защо):

$$X_{m+n} := X_n, \;\ldots\; , X_{m+1} := X_1$$

Следва блок от инструкции, с които първите m регистъра се зареждат с $a_1, \ldots, a_m$:

$$X_1{:=}0, \underbrace{X_1{:=}X_1{+}1, \ldots, X_1{:=}X_1{+}1}_{a_1 \text{ пъти}}, \ldots, X_m{:=}0, \underbrace{X_m{:=}X_m{+}1, \ldots, X_m{:=}X_m{+}1}_{a_m \text{ пъти}}$$

Това са общо $a_1{+}\cdots{+}a_m{+}m$ на брой инструкции, които заедно с горните n правят общо $a_1 + \cdots + a_m + m + n$ начални инструкции. Оттук нататък доказателството върви по същия начин, както при случая $m = n = 1$, с тази разлика, че отместването в адресите на операторите за преход трябва да е не $a + 2$, а $a_1 + \cdots + a_m + m + n$. □

Задача 3.5. (Задача за ЕК) Докажете, че S^m_n-функцията в S^m_n-теоремата може да се избере така, че да не зависи от n.

3.2.2 Слаба S_n^m-теорема

За повечето от приложенията на S_n^m-теоремата ще ни е достатъчна следната "неравномерна" нейна версия, с която всъщност започнахме раздел 3.2. Тя се получава, когато не се интересуваме от индекса на функцията, която ще параметризираме.

Теорема 3.4. (Слаба S_n^m-теорема) Нека $f(a_1, \ldots, a_m, x_1, \ldots, x_n)$ е изчислима. Тогава съществува примитивно рекурсивна функция $h(a_1, \ldots, a_m)$, такава че за всяко $\bar{a} \in \mathbb{N}^m, \bar{x} \in \mathbb{N}^n$ е изпълнено:

$$\varphi^{(n)}_{h(a_1,\ldots,a_m)}(x_1, \ldots, x_n) \simeq f(a_1, \ldots, a_m, x_1, \ldots, x_n).$$

Доказателство. Щом $f(\bar{a}, \bar{x})$ е изчислима, то тя има индекс, т.е. съществува e_0, такова че $f = \varphi^{(m+n)}_{e_0}$. Да положим

$$h(\bar{a}) \overset{\text{деф}}{=} S_n^m(e_0, \bar{a}).$$

От общата S_n^m-теорема имаме, че

$$\varphi^{(n)}_{S_n^m(e,\bar{a})}(\bar{x}) \simeq \varphi^{(m+n)}_e(\bar{a}, \bar{x})$$

за всички естествени $e, \bar{a}, \bar{x}$. Оттук при $e = e_0$ получаваме

$$\varphi^{(n)}_{S_n^m(e_0,\bar{a})}(\bar{x}) \simeq \varphi^{(m+n)}_{e_0}(\bar{a}, \bar{x}) \overset{\text{деф}}{\simeq} f(\bar{a}, \bar{x}), \quad \text{или}$$

$$\varphi^{(n)}_{h(\bar{a})}(\bar{x}) \simeq f(\bar{a}, \bar{x}) \quad \text{за всички естествени } \bar{a}, \bar{x}.$$

□

Всъщност слабата S_n^m-теорема е еквивалентна на S_n^m-теоремата, макар привидно да изглежда по-слаба от нея (а и името ѝ да е такова ☺). Поради това и на двете формулировки ще се позоваваме като на "S_n^m-теоремата", а контекстът ще показва коя от двете имаме предвид.

Задача 3.6. Да се докаже, че от слабата S_n^m-теорема следва (първоначалната) S_n^m-теорема.

Решение. Да фиксираме $m \geq 1, n \geq 1$. Искаме да покажем, че съществува примитивно рекурсивна функция S_n^m, за която

$$\varphi^{(n)}_{S_n^m(e,\bar{a})}(\bar{x}) \simeq \varphi^{(m+n)}_e(\bar{a}, \bar{x}).$$

Да приложим слабата S_n^m-теорема към функцията

$$f(e, \bar{a}, \bar{x}) \overset{\text{деф}}{\simeq} \Phi_{m+n}(e, \bar{a}, \bar{x}),$$

като параметризираме по първите ѝ $m+1$ аргумента $(e, a_1, \ldots, a_m)$. Ще получим, че за някоя примитивно рекурсивна функция $h(e, \bar{a})$ е изпълнено

$$\varphi^{(n)}_{h(e,\bar{a})}(\bar{x}) \simeq f(e, \bar{a}, \bar{x}), \quad \text{или все едно}$$

$$\varphi^{(n)}_{h(e,\bar{a})}(\bar{x}) \simeq \underbrace{\Phi_{m+n}(e, \bar{a}, \bar{x})}_{\varphi_e^{(m+n)}(\bar{a},\bar{x})}.$$

Остана да вземем $S^m_n \overset{\text{деф}}{=} h$. □

3.2.3 Приложения

По-съществените приложения на S^m_n-теоремата ще наблюдаваме в теоремите, които ще доказваме по-нататък в курса. Сега ще решим няколко задачи, за да видим на практика как работи тази теорема.

Задача 3.7. Да се докаже, че съществува примитивно рекурсивна функция h, такава че за всяко естествено n програмата с код $h(n)$ пресмята функцията x^n.

Доказателство. Условието програмата $P_{h(n)}$ да пресмята x^n ще рече, че функцията $\varphi_{h(n)}$ е x^n, т.е. за всяко n и x имаме

$$\varphi_{h(n)}(x) = \underbrace{x^n}_{f(n,x)}.$$

Ако си представяме, че h е "дошла" от слабата S^m_n-теорема, то ясно е, че функцията, към която трябва да приложим тази теорема, е функцията вдясно в горното равенство. Да я означим с $f(n, x)$.

Знаем, че $f(n, x) = x^n$ е примитивно рекурсивна, следователно тя е и изчислима. Тогава по S^m_n-теоремата ще съществува примитивно рекурсивна функция h, такава че за всяко n и x

$$\varphi_{h(n)}(x) = f(n, x), \quad \text{или все едно,} \quad \varphi_{h(n)}(x) = x^n.$$

□

Задача 3.8. Докажете, че съществува примитивно рекурсивна функция l, такава че за всички естествени a и b

$$\varphi_{l(a,b)}(x) = ax + b.$$

Доказателство. Смисълът на функцията $l(a, b)$ е, че по всяко a и b тя връща код на програма, пресмятаща линейната функция $ax + b$.

За да получим l, тръгваме от функцията $f(a,b,x) = ax+b$, която очевидно е изчислима. Прилагаме S_n^m-теоремата към нея, като този път трябва да параметризираме по първите ѝ два аргумента. Така получаваме, че за някоя примитивно рекурсивна функция $l(a,b)$ ще е в сила

$$\varphi_{l(a,b)}(x) = f(a,b,x) = ax + b$$

за всяко a, b и x. □

Да напомним, че с C_a^n означавахме n-местната константна функция, която винаги връща a, т.е.

$$C_a^n(x_1, \ldots, x_n) = a \quad \text{за всяко} \quad (x_1, \ldots, x_n) \in \mathbb{N}^n.$$

Задача 3.9. Нека $n \geq 1$. Докажете, че съществува примитивно рекурсивна функция c_n, такава че за всички естествени a:

$$\varphi_{c_n(a)} = C_a^n.$$

С други думи, за всяко a, $c_n(a)$ връща някакъв индекс на C_a^n.

Решение. Да фиксираме някакво $n \geq 1$. Търсената от нас функция c_n трябва да е такава, че за всяко $\bar{x} \in \mathbb{N}^n$ да бъде вярно, че

$$\varphi_{c_n(a)}(\bar{x}) = C_a^n(\bar{x}), \quad \text{т.е.} \quad \varphi_{c_n(a)}(\bar{x}) = \underbrace{a}_{f(a,\bar{x})}.$$

Функцията $f(a,\bar{x}) \overset{\text{деф}}{=} a$ очевидно е изчислима и значи към нея може да се приложи S_n^m-теоремата. Според тази теорема ще съществува примитивно рекурсивна функция — да я наречем c_n, такава че за всяко a и $\bar{x}$:

$$\varphi_{c_n(a)}(\bar{x}) = f(a,\bar{x}), \quad \text{или все едно,} \quad \varphi_{c_n(a)}(\bar{x}) = a.$$

Но последното означава, че $\forall a\ \varphi_{c_n(a)} = C_a^n$. □

Тази задача може да се обобщи, като местността n на константната функция C_a^n стане аргумент на параметризиращата функция c.

Задача 3.10. (Бърз ЕК, срок: 24 ноември) Докажете, че съществува примитивно рекурсивна функция $c(n,a)$, такава че за всички естествени n и a:

$$\varphi_{c(n,a)} = C_a^n.$$

Задача 3.11. а) Нека f е фиксирана едноместна изчислима функция. Докажете, че съществува едноместна примитивно рекурсивна функция p, такава че за всяко естествено n:

$$\varphi_{p(n)} = f^n.$$

б) Докажете, че съществува двуместна примитивно рекурсивна функция p, такава че за всички естествени a и n:

$$\varphi_{p(a,n)} = \varphi_a^n.$$

Решение. Разбира се, от втората подточка следва първата, но е поучително да ги разгледаме в реда, в който са формулирани в задачата.

а) Търсената функция $p(n)$ трябва да е такава, че за всички a и x:

$$\varphi_{p(n)}(x) \simeq f^n(x).$$

От *Твърдение* 1.17 знаем, че ако f е примитивно рекурсивна, то и нейната итерация $f^*(n,x) \overset{\text{деф}}{=} f^n(x)$ ще е такава. Дали това остава в сила ако заменим "примитивно" с "частично" рекурсивна? Абсолютно, при това, и доказателството е съвсем същото: примитивно рекурсивна схема за f^*:

$$\left|\begin{array}{l} f^*(0,x) = x \\ f^*(n+1,x) \simeq \underbrace{f(\ldots f}_{n+1 \text{ пъти}}(x)\ldots) \simeq f(\underbrace{f(\ldots f}_{n \text{ пъти}}(x)\ldots)) \simeq f(f^*(n,x)). \end{array}\right.$$

Сега просто прилагаме S_n^m-теоремата към изчислимата функция $f^*(n,x)$, като параметризираме по първия ѝ аргумент n.

б) Ясно е, че тук трябва да параметризираме по n и a. За целта първо трябва да съобразим, че функцията $\lambda a,n,x.\varphi_a^n(x)$ е изчислима. Да положим $F(a,n,x) \overset{\text{деф}}{\simeq} \varphi_a^n(x)$. За F имаме следната пр. рекурсивна схема:

$$\left|\begin{array}{l} F(a,0,x) \simeq \varphi_a^0(x) = x \\ F(a,n+1,x) \simeq \varphi_a^{n+1}(x) \simeq \varphi_a(\varphi_a^n(x)) \simeq \Phi_1(a,F(a,n,x)). \end{array}\right.$$

Съобразете, че F е изчислима и приложете към нея S_n^m-теоремата. □

За всяко $a \in \mathbb{N}$, нека W_a е дефиниционната област на функцията φ_a:

$$W_a \;=\; \{x|\ !\varphi_a(x)\}.$$

Задача 3.12. Докажете, че съществува примитивно рекурсивна функция s, такава че за всяко a, $W_{s(a)} = \{a\}$.

Доказателство. Търсим функция $f(a,x)$, която е такава, че след прилагане към нея на S_n^m-теоремата и получаване на примитивно рекурсивна s със свойството:

$$\varphi_{s(a)}(x) \simeq f(a,x),$$

да се окаже, че s е търсената, т.е. за нея да е изпълнено $W_{s(a)} = \{a\}$. Последното равенство означава, че за всяко a и x е в сила еквивалентността

$$!\varphi_{s(a)}(x) \iff x = a.$$

Ние търсим f, за която да е изпълнено $\varphi_{s(a)}(x) \simeq f(a,x)$, затова горната еквивалентност преписваме като

$$!f(a,x) \iff x = a.$$

Има много изчислими функции f горното свойство; бихме могли да вземем например

$$f(a,x) \simeq \begin{cases} 0, & \text{ако } a = x \\ \neg!, & \text{иначе.} \end{cases}$$

Прилагаме S_n^m-теоремата към тази функция и получаваме, че за някоя примитивно рекурсивна s ще бъде изпълнено

$$\varphi_{s(a)}(x) \simeq f(a,x), \quad \text{и в частност,}$$

$$!\varphi_{s(a)}(x) \iff !f(a,x) \iff x = a$$

за всяко a и x. Това означава, че $W_{s(a)} = \{a\}$ за всяко a. □

В *Задача* 3.1 въведохме *код* на полином $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$ с коефициенти от $\mathbb{N}$. Това беше числото $a = \tau(\langle a_0, \ldots a_n\rangle)$, което кодира редицата от коефициентите му.
Да си припомним и означението

$$p_a \;=\; \text{полинома с код } a.$$

Разбира се, всеки полином е изчислима функция и значи той може да бъде зададен и със свой индекс. Следващата задача ни казва, че съществува примитивно рекурсивна функция, която по код на полином връща негов индекс, т.е. има "равномерен" преход от кодовете на полиномите към техните индекси.

Задача 3.13. Да се докаже, че съществува примитивно рекурсивна функция h, такава че за всяко a

$$\varphi_{h(a)} = p_a.$$

Доказателство. В *Задача* 3.1 показахме, че е изчислима функцията

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

Прилагаме S_n^m-теоремата към тази функция и получаваме, че за някоя примитивно рекурсивна h ще е изпълнено:

$$\varphi_{h(a)}(x) = U(a,x), \quad \text{или все едно} \quad \varphi_{h(a)}(x) = p_a(x)$$

за всяко a и x. □

3.3 Ефективни оператори

3.3.1 Определение и примери за оператори

Под *оператор* ще разбираме тотално изображение, което преработва функции във функции. Операторите ще означаваме с главни гръцки букви — $\Gamma, \Delta, ...$, евентуално с индекси.

Да се спрем първо на случая, когато тези оператори имат един аргумент. Тогава те ще са изображения от вида

$$\Gamma \colon \mathcal{F}_n \longrightarrow \mathcal{F}_k$$

за някои естествени $n \geq 1$ и $k \geq 1$. Константите n и k определят *типа на оператора* Γ, който ще означаваме с $(n \ \to \ k)$.

Примери:

1) операторът *идентитет*: $\Gamma_{id}(f) = f$, който е от тип $(k \ \to \ k)$.

2) *константният оператор* $\Gamma_c(f) = f_0$, за f_0 — фиксирана функция.
Този оператор може да е от произволен тип $(n \ \to \ k)$;

3) операторът за *диагонализация* $\Gamma_d(f)(x) \simeq f(x,x)$,
който е от тип $(2 \to 1)$. Ако се чудите откъде идва името на този оператор, представете си стойностите на f, разположени в таблица с безкрайно много редове и стълбове, в която (i,j)-тият елемент е $f(i,j)$.

4) $\Gamma(f)(x) \simeq$ `if` $x = 0$ `then` 1 `else` $x.f(x-1)$.
Този оператор е тясно свързан с рекурсивната дефиниция на функцията *факториел*; очевидно е от тип $(1 \ \to \ 1)$;

5) операторът за *минимизация* $\Gamma_\mu(f)(\bar{x}) \simeq \mu y[f(\bar{x}, y) \simeq 0]$
от тип $(n+1 \ \to \ n)$.

Оператор на *произволен брой аргументи* е изображение от вида

$$\Gamma \colon \ \mathcal{F}_{n_1} \times \cdots \times \mathcal{F}_{n_k} \ \longrightarrow \ \mathcal{F}_m.$$

Типа на Γ ще означаваме с $(n_1, \ldots, n_k \ \to \ m)$.

Примери:

1) операторът *композиция*: $\Gamma_{comp}(f,g)(\bar{x}) \simeq f(g(\bar{x}))$ от тип $(1, n \ \to \ n)$;

2) операторът $\Gamma_{mult}(f,g)(\bar{x}) \simeq f(\bar{x}).g(\bar{x})$ от тип $(n, n \ \to \ n)$;

3) операторът *суперпозиция* $\Gamma_{sup}(f, g_1, \ldots, g_n) = f(g_1, \ldots, g_n)$
от тип $(n, k, \ldots, k \ \to \ k)$;

4) операторът `if then else`: $\Gamma(f,g,h)(\bar{x}) \simeq$ `if` $f(\bar{x}) \simeq 0$ `then` $g(\bar{x})$ `else` $h(\bar{x})$,
който е от тип $(n, n, n \ \to \ n)$.

3.3.2 Ефективни оператори

Определение 3.2. Казваме, че операторът $\Gamma\colon \mathcal{F}_{n_1}\times\cdots\times\mathcal{F}_{n_k} \longrightarrow \mathcal{F}_m$ е *ефективен*, ако съществува рекурсивна функция h, такава че за всички естествени $a_1,\ldots,a_k$ е изпълнено:

$$\Gamma(\varphi_{a_1}^{(n_1)},\ldots,\varphi_{a_k}^{(n_k)}) \;=\; \varphi_{h(a_1,\ldots,a_k)}^{(m)}.$$

Функцията h ще наричаме *индексна функция* на Γ.

Ако $\Gamma\colon \mathcal{F}_n \longrightarrow \mathcal{F}_k$ е оператор на една променлива, горното определение се свежда до съществуването на *едноместна* рекурсивна функция h, такава че за всяко a:

$$\Gamma(\varphi_a^{(n)}) \;=\; \varphi_{h(a)}^{(k)}.$$

Да обърнем внимание, че операторът Γ действа върху *всички* частични функции, а условието за ефективност е само върху изчислимите. При това искаме Γ не просто да преработва изчислими в изчислими, но да го прави по *равномерен* начин, което ще рече — да има изчислима функция, която по индекса на аргументите на Γ да връща индекса на резултата.

Всички "естествени" оператори са ефективни, в частност, операторите от горните примери. Да проверим ефективността на добре познатия ни оператор за минимизация.

Задача 3.14. Докажете, че е ефективен операторът за минимизация $\Gamma_\mu\colon \mathcal{F}_2 \longrightarrow \mathcal{F}_1$, дефиниран като:

$$\Gamma_\mu(f)(x) \simeq \mu y[f(x,y)\simeq 0].$$

Решение. Търсим рекурсивна функция $h(a)$, такава че за всяко a и x

$$\varphi_{h(a)}(x) \simeq \Gamma_\mu(\varphi_a^{(2)})(x) \overset{\text{деф}}{\simeq} \underbrace{\mu y[\varphi_a^{(2)}(x,y)\simeq 0]}_{F(a,x)}.$$

Очакваме да получим функцията h от S_n^m-теоремата, затова разглеждаме функцията $F(a,x)$, както е означена по-горе. Тя е изчислима, защото можем да я препишем като

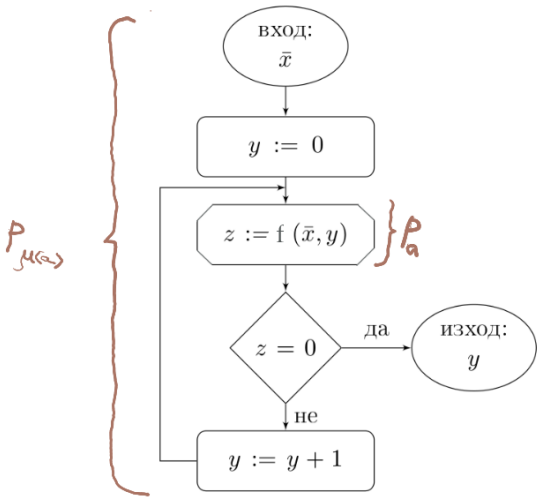
$$F(a,x) \simeq \mu y[\Phi_2(a,x,y)\simeq 0]$$

и да приложим теоремата за универсалната функция, според която УФ Φ_2 е изчислима. Сега вече, прилагайки S_n^m-теоремата към тази функция F, стигаме до (дори примитивно) рекурсивна функция h, такава че

$$\varphi_{h(a)}(x) \simeq \mu y[\varphi_a^{(2)}(x,y)\simeq 0]$$

за всяко a и x. Оттук $\varphi_{h(a)} = \Gamma_\mu(\varphi_a^{(2)})$ за всяко a и следователно Γ_μ е ефективен оператор. $\square$

Да напомним, че миналия път доказахме, че операцията минимизация запазва изчислимостта. Сега виждаме, че всъщност тя я запазва *равномерно*, т.е. по кода на програмата, пресмятаща $f(x,y)$, можем алгоритмично да получим кода на програмата, пресмятаща $\mu y[f(x,y) \simeq 0]$.



3.3.3 НДУ за ефективност на оператор

Разсъжденията в решението на предишната *Задача* 3.14 ни насочват към следващото твърдение, което се явява НДУ са ефективност на оператор. Ще го използваме, когато по-нататък се налага да доказваме, че даден оператор е ефективен.

Твърдение 3.4. (Необходимо и достатъчно условие за ефективност на оператор) Операторът $\Gamma\colon \mathcal{F}_{n_1} \times \cdots \times \mathcal{F}_{n_k} \longrightarrow \mathcal{F}_m$ е ефективен тогава и само тогава, когато е изчислима функцията

$$F(a_1, \ldots, a_k, x_1, \ldots, x_m) \overset{\text{деф}}{\simeq} \Gamma(\varphi_{a_1}^{(n_1)}, \ldots, \varphi_{a_k}^{(n_k)})(x_1, \ldots, x_m).$$

Доказателство. Нека Γ е ефективен оператор. Тогава за някоя рекурсивна функция h ще бъде изпълнено

$$\Gamma(\varphi_{a_1}^{(n_1)}, \ldots, \varphi_{a_k}^{(n_k)})(\bar{x}) \simeq \varphi^{(m)}_{h(a_1,\ldots,a_k)}(\bar{x})$$

за всички $\bar{a} \in \mathbb{N}^k, \bar{x} \in \mathbb{N}^m$. Тогава за F можем да запишем:

$$F(\bar{a}, \bar{x}) \overset{\text{деф}}{\simeq} \Gamma(\varphi_{a_1}^{(n_1)}, \ldots, \varphi_{a_k}^{(n_k)})(\bar{x}) \simeq \varphi_{h(\bar{a})}^{(m)}(\bar{x}) \simeq \Phi_m(h(\bar{a}), \bar{x}),$$

и значи F е изчислима, съгласно теоремата за универсалната функция.

Обратно, нека функцията $F(a_1, \ldots, a_k, x_1, \ldots, x_m)$ от условието на твърдението е изчислима. Прилагаме S_n^m-теоремата, като параметризираме по променливите $(a_1, \ldots, a_k)$. Така получаваме, че за някоя (дори) *примитивно* рекурсивна функция h ще е изпълнено

$$\varphi_{h(\bar{a})}^{(m)}(\bar{x}) \simeq F(\bar{a}, \bar{x}),$$

или все едно,

$$\varphi_{h(\bar{a})}^{(m)}(\bar{x}) \simeq \Gamma(\varphi_{a_1}^{(n_1)}, \ldots, \varphi_{a_k}^{(n_k)})(\bar{x}).$$

Горното равенство е за всички $\bar{a} \in \mathbb{N}^k$ и $\bar{x} \in \mathbb{N}^m$, следователно

$$\varphi_{h(\bar{a})}^{(m)} = \Gamma(\varphi_{a_1}^{(n_1)}, \ldots, \varphi_{a_k}^{(n_k)})$$

за всички $\bar{a} \in \mathbb{N}^k$. □

Забележка. От горното твърдение следва, в частност, че ако един оператор има рекурсивна индексна функция, той има и примитивно рекурсивна индексна функция.

Да приложим току-що доказаното, за да видим, че е ефективен операторът композиция.

Задача 3.15. Докажете, че е ефективен операторът
$\Gamma_{comp} \colon \mathcal{F}_1 \times \mathcal{F}_1 \longrightarrow \mathcal{F}_1$, който се дефинира с равенството

$$\Gamma_{comp}(f, g)(x) \simeq f(g(x)).$$

Решение. Разглеждаме функцията

$$F(a, b, x) \simeq \Gamma_{comp}(\varphi_a, \varphi_b)(x) \simeq \varphi_a(\varphi_b(x)).$$

Тя е изчислима, защото можем да я препишем като

$$F(a, b, x) \simeq \Phi_1(a, \Phi_1(b, x)).$$

Тогава съгласно *Твърдение* 3.4 операторът Γ_{comp} е ефективен, т.е. за някоя рекурсивна функция *comp* е изпълнено

$$\Gamma_{comp}(\varphi_a, \varphi_b) = \varphi_{comp(a,b)}$$

за всички естествени a и b. □

Задача 3.16. Докажете, че операторът за итерация $\Gamma_{it} : \mathcal{F}_1 \longrightarrow \mathcal{F}_2$, дефиниран като

$$\Gamma_{it}(\varphi_a) = \varphi_a^*$$

е ефективен.

Решение. В решението на *Задача* 3.11 б) вече съобразихме, че функцията $F(a,n,x) \overset{\text{деф}}{\simeq} \varphi_a^n(x)$ е изчислима. Но

$$\varphi_a^n(x) \simeq \varphi_a^*(n,x), \quad \text{а} \quad \varphi_a^*(n,x) \overset{\text{деф}}{\simeq} \Gamma_{it}(\varphi_a)(n,x).$$

Излезе, че $F(a,n,x) \simeq \Gamma_{it}(\varphi_a)(n,x)$ и тази функция е изчислима. Сега прилагаме *Твърдение* 3.4, за да получим, че операторът Γ_{it} е ефективен. □

Задача 3.17. Дайте пример за оператор $\Gamma\colon \mathcal{F}_1 \longrightarrow \mathcal{F}_1$, който не е ефективен.

Решение. Един тривиален ход е да изберем оператор, който извежда от класа на изчислимите функции. Бихме могли направо да вземем константния оператор

$$\Gamma(f) \overset{\text{деф}}{=} f_0,$$

където f_0 е някаква неизчислима функция.

Ако търсим по-интересен пример, може би първото нещо, което ни идва на ум, е да вземем някоя неизчислима функция h и да положим

$$\Gamma(\varphi_a) \overset{\text{деф}}{=} \varphi_{h(a)}.$$

Изглежда првдоподобно, но всъщност този пример не е пример за нищо, защото горната дефиниция е некоректна в общия случай. Нищо не ни гарантира, че за всяко a и b ще е изпълнено условието

$$\varphi_a = \varphi_b \implies \varphi_{h(a)} = \varphi_{h(b)}.$$

(Горното свойство се нарича *екстензионалност* на h.)

Да опитаме с този оператор:

$$\Gamma(\varphi_a) \overset{\text{деф}}{=} \varphi_{m(a)},$$

където $m(a) = min\{e \mid \varphi_e = \varphi_a\}$. Лесно се вижда, че функцията m вече е екстензионална. Това, че m не е изчислима, ще следва от един по-общ резултат, който ще покажем по-нататък. □

3.4 Теореми за рекурсия

Най-общо, теоремите за рекурсия са твърдения за съществуване на изчислими функции, удовлетворяващи някакви рекурсивни условия. Тези условия могат да бъдат изказани както в термините на самите функции, така и чрез програмите, които ги пресмятат. Как точно става това ще обсъдим в следващите примери.

3.4.1 Няколко примера

В първия пример са три рекурсивни дефиниции, които вече сме обсъждали по други поводи.

Пример 3.1. 1) Най-напред "буквarният" пример за дефиниция по рекурсия:

$$f(x) \simeq \begin{cases} 1, & \text{ако } x = 0 \\ x.f(x-1), & \text{иначе.} \end{cases}$$

Тук с обикновена индукция по xсе вижда, че единствената функция, удовлетворяваща това рекурсивно условие, е функцията $f(x) = x!$.

2) Следващият също тъй популярен пример е за рекурсивната дефиниция на функцията на Фибоначи:

$$f(x) \simeq \begin{cases} 1, & \text{ако } x \leq 1 \\ f(x-1) + f(x-2), & \text{иначе.} \end{cases}$$

И тук с лека (но вече пълна) индукция се показва, че има единствена функция, за която е в сила горното условие. В *Задача* 1.10 видяхме, че тази функция е примитивно рекурсивна.

3) Последният пример е за рекурсивната дефиниция на функцията на Акерман, която можем да препишем и по този начин:

$$f(x,y) \simeq \begin{cases} y+1, & \text{ако } x = 0 \\ f(x-1,1), & \text{ако } x > 0 \ \& \ y = 0 \\ f(x-1, f(x,y-1)), & \text{в останалите случаи.} \end{cases}$$

В решението на *Задача* 1.18 видяхме, че съществува единствена функция, за която горните условия са изпълнени и тази функция е тотална.

Това, че функцията на Акерман е изчислима (което ще рече, рекурсивна) ще докажем по-нататък в *Задача* 3.24. Сега само да отбележим, че и трите рекурсивни дефиниции са от вида

$$f(\bar{x}) \simeq \underbrace{\ldots f, \bar{x} \ldots}_{\Gamma(f)(\bar{x})}$$

За да можем да атакуваме общата задача за такъв тип дефиниция по рекурсия, ще трябва да изучим основните свойства на операторите Γ, чрез които се задава рекурсивно една функция f. Това ще направим обстойно в раздел 3.6.

А сега да видим какво става, ако f се задава рекурсивно не само чрез своите стойности, но и чрез алгоритъма, който я пресмята, в нашия случай — чрез МНР програмата ѝ. Тъй като спокойно можем да отъждествяваме програмите с естествените числа, тук вече нямаме нужда от предварителната подготовка на предишния подход.

Освен това сега вече можем да пишем далеч по-общи рекурсивни дефиниции, в които участват и самите определяеми *програми*.

Пример 3.2. 1) Да наречем програмата P_a *самовъзпроизвеждаща се*, ако за всеки вход x тя връща собствения си код, т.е. за всяко естествено x е изпълнено

$$P_a(x) = a.$$

(Тук пишем $P_a(x) = y$ вместо $P_a(x) \downarrow y$, за да изглежда повече като уравнение условието за P_a ☺.)

Разбира се, това условие можем да препишем и като условие за φ_a:

$$\varphi_a(x) = a.$$

За разлика от горните примери, тук вече не е толкова ясно, че съществува такава програма P_a (или все едно, такава изчислима функция φ_a). Това ще получим като следствие от една от теоремите за рекурсия.

2) Едно обобщение на горния пример е да поискаме изходът $P_a(x)$ да зависи и от входа x — например, нека за всяко x да имаме

$$P_a(x) = a + x.$$

3) Можем да си зададем въпроса дали съществува програма P_a, която за всеки вход x дава някаква информация, свързана с изчисленията си при този вход — например, $P_a(x)$ да връща кода на конфигурацията на стъпка x (тук подразбираме, че тази конфигурация е финалната, до която достига $P_a(x)$, ако е спряла за по-малко от x стъпки). Малко по-точно:

$$P_a(x) = \begin{cases} \text{кода на конфигурацията на стъпка } x, & \text{ако } P_a(x) \text{ не спира} \\ & \text{за } \leq x \text{ стъпки} \\ \text{кода на финалната конфигурация}, & \text{иначе.} \end{cases}$$

Отговори — при това, позитивни — на тези и други въпроси дава теоремата за определимост по рекурсия.

3.4.2 Теорема за определимост по рекурсия

Съществуването на програми с всяко от свойствата, изброени в *Пример* 3.2, ще е елементарно следствие от следващата теорема за рекурсия, принадлежаща на Клини.

Теорема 3.5. (Теорема за определимост по рекурсия) Нека $n \geq 1$. За всяка изчислима функция $f(a, x_1, \ldots, x_n)$ съществува естествено число a, такова че

$$\varphi_a^{(n)}(\bar{x}) \simeq f(a, \bar{x})$$

за всяко $\bar{x} \in \mathbb{N}^n$.

Доказателство. Да разгледаме функцията

$$g(a, \bar{x}) \overset{\text{деф}}{\simeq} f(S_n^1(a, a), \bar{x}).$$

Тя е изчислима и следователно има индекс. Да фиксираме един такъв индекс e. Тогава за всяко a и $\bar{x}$ ще е вярно, че

$$\varphi_e^{(n+1)}(a, \bar{x}) \simeq g(a, \bar{x}).$$

Да приложим S_n^m-теоремата към функцията $\varphi_e^{(n+1)}(a, \bar{x})$ с параметри e и a. Ще получим, че за всяко a и $\bar{x}$

$$\varphi_{S_n^1(e,a)}^{(n)}(\bar{x}) \simeq \varphi_e^{(n+1)}(a, \bar{x}).$$

Комбинираме с равенството по-горе и получаваме, че

$$\varphi_{S_n^1(e,a)}^{(n)}(\bar{x}) \simeq g(a, \bar{x}) \overset{\text{деф}}{\simeq} f(S_n^1(a, a), \bar{x})$$

за всяко a и $\bar{x}$. В частност, при $a = e$ достигаме до

$$\varphi_{\underbrace{S_n^1(e,e)}_{a_0}}^{(n)}(\bar{x}) \simeq f(\underbrace{S_n^1(e,e)}_{a_0}, \bar{x}).$$

Да означим $a_0 \overset{\text{деф}}{=} S_n^1(e, e)$. Заместваме в горното равенство и получаваме, че за всяко $\bar{x} \in \mathbb{N}^n$

$$\varphi_{a_0}^{(n)}(\bar{x}) \simeq f(a_0, \bar{x}),$$

което означава, че a_0 удовлетворява условието на теоремата. □

Да приложим тази теорема към всяка от задачите от *Пример* 3.2.

1) Търсим a, такова че за всяко x

$$\varphi_a(x) = \underbrace{a}_{f(a,x)} .$$

Затова прилагаме горната теорема към функцията $f(a,x) \overset{\text{деф}}{=} a$. Тя очевидно е изчислима и следователно за поне едно a ще бъде вярно, че за всяко x

$$\varphi_a(x) = f(a,x), \qquad \text{или все едно} \quad \varphi_a(x) = a.$$

Последното равенство, преписано чрез P_a, ни дава

$$P_a(x) = a \quad \text{за всяко } x.$$

Така показахме, че самовъзпроизвеждащи се програми съществуват.

2) В този пример за програмата P_a искаме при всеки вход x да е изпълнено

$$P_a(x) = \underbrace{a+x}_{f(a,x)}, \quad \text{което ще рече} \quad \varphi_a(x) = \underbrace{a+x}_{f(a,x)}.$$

Това, че такова a съществува се осигурява отново от горната теорема, приложена този път за $f(a,x) = a + x$.

3) Условието към функцията φ_a тук е:

$$\varphi_a(x) = \begin{cases} \text{кода на конфигурацията на стъпка } x, & \text{ако } P_a(x) \text{ не спира} \\ & \text{за } \leq x \text{ стъпки} \\ \text{кода на финалната конфигурация,} & \text{иначе.} \end{cases}$$

При доказателството на теоремата за универсалната функция показахме, че е примитивно рекурсивна функцията

$$Q_n(a,\bar{x},t) \overset{\text{деф}}{=} \text{кода на конфигурацията, която се получава след } t \text{ такта от работата на } P_a \text{ върху } \bar{x}.$$

Значи такава ще бъде и следната функция f:

$$f(a,x) = Q_1(a,x,x) \overset{\text{деф}}{=} \text{кода на конфигурацията след } x \text{ такта от работата на } P_a \text{ върху } x.$$

Да си спомним, че по определение, ако $P_a(x)$ спре за t_0 такта, то за всяко $t > t_0$ по дефиниция $Q_1(a,x,t) = Q_1(a,x,t_0)$. Но това означава, че φ_a удовлетворява точно равенството

$$\varphi_a(x) = f(a,x)$$

за всяко x. Сега просто прилагаме теоремата за определимост по рекурсия към функцията f и получаваме, че съществува a с горното свойство.

Задача 3.18. Докажете, че съществува a, за което

$$W_a = \{a\}.$$

(Или изказано в термините на програми: докажете, че съществува програма P_a, която спира само върху собствения си код.)

Решение. Трябва да приложим теоремата за определимост по рекурсия към подходяща изчислима функция $f(a, x)$. Ясно е, че за нея трябва да е изпълнено условието:

$$\text{ако } a \text{ е такова, че } \quad \varphi_a(x) \simeq f(a, x), \quad \text{ то } \quad W_a = \{a\}.$$

Тогава за всяко a и x ще имаме

$$!f(a, x) \iff !\varphi_a(x) \overset{\text{деф}}{\iff} x \in W_a \iff x = a.$$

Една изчислима функция f, за която горното условие е вярно, е например

$$f(a, x) \simeq \begin{cases} 0, & \text{ако } a = x \\ \neg!, & \text{иначе.} \end{cases}$$

Ясно е, че тази f ще ни свърши работа, но да се убедим формално, тръгвайки по обратен ред. Наистина, от теоремата за определимост по рекурсия ще съществува a, за което $\varphi_a(x) \simeq f(a, x)$. Оттук в частност, $!\varphi_a(x) \iff !f(a, x)$, което означава, че за това a ще е изпълнено

$$x \in W_a \iff !f(a, x) \iff x = a.$$

Разбира се, горната еквивалентност е за всяко x, което ни дава точно $W_a = \{a\}$. □

Задача 3.19. Докажете, че съществува рекурсивна функция g, такава че за всяко n, $g(n)$ е индекс на $\lambda x.ng(x)$.

Решение. Функцията g ще търсим във вида φ_a. Искаме за всяко n $\varphi_a(n)$ да е индекс на $\lambda x.n\varphi_a(x)$, което означава, че за всяко n и x:

$$\varphi_{\varphi_a(n)}(x) \simeq \underbrace{n.\varphi_a(x)}_{f(a,n,x)}.$$

Тук $f(a, n, x) \simeq n.\varphi_a(x) \simeq n.\Phi_1(a, x)$ е изчислима, съгласно теоремата за универсалната функция. Сега прилагаме към f S_n^m-теоремата и получаваме, че за някоя примитивно рекурсивна функция $h(a, n)$:

$$\varphi_{h(a,n)}(x) \simeq f(a, n, x)$$

за всяко a, n, x. Ние търсим a със свойството $\varphi_{\varphi_a(n)}(x) \simeq f(a, n, x)$, следователно за това a трябва да е вярно, че

$$\varphi_{\varphi_a(n)} = \varphi_{h(a,n)}$$

за всяко n. За да получим, че такова a съществува, е достатъчно да приложим теоремата за определимост по рекурсия към функцията h. □

3.4.3 Втора теорема за рекурсия

От теоремата за определимост по рекурсия лесно се извежда следващото твърдение, известно като *втора теорема за рекурсия:*

Теорема 3.6. (Втора теорема за рекурсия) Нека $n \geq 1$, а h е едноместна рекурсивна функция. Тогава съществува индекс a, такъв че

$$\varphi^{(n)}_{h(a)} = \varphi^{(n)}_a.$$

Доказателство. Да разгледаме функцията

$$f(a, \bar{x}) \overset{\text{деф}}{\simeq} \varphi^{(n)}_{h(a)}(\bar{x}).$$

Тя е изчислима, защото можем да я препишем като $f(a, \bar{x}) \simeq \Phi_n(h(a), \bar{x})$ и да вземем пред вид, че Φ_n е изчислима. Тогава към f можем да приложим теоремата за определимост по рекурсия. Така получаваме, че за поне едно a:

$$\varphi^{(n)}_a(\bar{x}) \simeq f(a, \bar{x}), \quad \text{или все едно,} \quad \varphi^{(n)}_a(\bar{x}) \simeq \varphi^{(n)}_{h(a)}(\bar{x}).$$

Последното равенство е изпълнено за всяко $\bar{x} \in \mathbb{N}^n$ и значи $\varphi^{(n)}_a = \varphi^{(n)}_{h(a)}$. □

От горното доказателство се вижда как "на една стъпка" от теоремата за определимост по рекурсия получихме втората теорема за рекурсия, като приложихме теоремата за универсалната функция. Да видим, че е вярно и обратното (като тук ще използваме другата важна теорема — S^m_n-теоремата). Поради това понякога и двете теореми 3.5 и 3.6 се наричат общо *втора теорема за рекурсия.*

Задача 3.20. Докажете, че от втората теорема за рекурсия следва теоремата за определимост по рекурсия.

Доказателство. Нека $f(a, \bar{x})$ е произволна изчислима функция. Към нея прилагаме S^m_n-теоремата и получаваме, че съществува рекурсивна функция h, такава че за всяко a и $\bar{x} \in \mathbb{N}^n$:

$$\varphi^{(n)}_{h(a)}(\bar{x}) \simeq f(a, \bar{x}).$$

Сега от втората теорема за рекурсия ще имаме, че за тази функция h съществува индекс a, такъв че

$$\varphi^{(n)}_{h(a)}(\bar{x}) \simeq \varphi^{(n)}_a(\bar{x})$$

за всяко $\bar{x}$. Тогава за това a и за всяко $\bar{x} \in \mathbb{N}^n$ ще е изпълнено

$$\varphi^{(n)}_a(\bar{x}) \simeq f(a, \bar{x}).$$

□

Нека h е едноместна рекурсивна функция. Ако за индекса a е изпълнено

$$\varphi^{(n)}_{h(a)} = \varphi^{(n)}_a,$$

то a ще наричаме *псевдонеподвижна точка* на h. Тогава втората теорема за рекурсия може да бъде изказана и така: *всяка едноместна рекурсивна функция има поне една псевдонеподвижна точка.*

Защо a се нарича *псевдо*неподвижна точка ще разберем по-нататък в лекцията, когато се запознаем с понятието "неподвижна точка на оператор". Сега да докажем, че псевдонеподвижните точки на всяка функция всъщност са безброй много.

Твърдение 3.5. Всяка едноместна рекурсивна функция h има безброй много псевдонеподвижни точки.

Доказателство. Трябва да покажем, че каквото и k да си вземем, ще съществува псевдонеподвижна точка a, която е по-голяма от k. За целта ще конструираме друга рекурсивна функция g, която е почти същата като h, и която няма "малки" неподвижни точки. Нека

$$g(a) = \begin{cases} h(a), & \text{ако } a > k \\ c, & \text{ако } a \leq k, \end{cases}$$

където c е такова, че $\varphi^{(n)}_c \notin \{\varphi^{(n)}_0, \ldots, \varphi^{(n)}_k\}$. Функцията g също е рекурсивна, следователно съществува a, такова че $\varphi^{(n)}_{g(a)} = \varphi^{(n)}_a$. От избора на c е ясно, че не може $a \leq k$. Значи остава $a > k$. Но тогава $g(a) = h(a)$ и оттук

$$\varphi^{(n)}_{h(a)} = \varphi^{(n)}_{g(a)} = \varphi^{(n)}_a.$$

Така конструирахме псевдонеподвижна точка на h, която е по-голяма от k. Понеже k беше произволно, можем да твърдим, че h има произволно големи псевдонеподвижни точки. □

Забележка. Разбира се, от това твърдение веднага следва, че са безброй много и индексите a от теоремата за определимост по рекурсия, т.е. индексите, за които $\forall \bar{x}\ \varphi^{(n)}_a(\bar{x}) \simeq f(a, \bar{x})$. За целта разсъждаваме както в доказателството на *Задача* 3.20: към дадената изчислима функция $f(a, \bar{x})$ прилагаме S^m_n-теоремата и получаваме рекурсивна h, такава че $\varphi^{(n)}_{h(a)}(\bar{x}) \simeq f(a, \bar{x})$. Ясно е, че за всички (безброй много) псевдонеподвижни точки на h ще е изпълнено $\varphi^{(n)}_a(\bar{x}) \simeq f(a, \bar{x})$.

Задача 3.21. Докажете, че съществуват безброй много a, за които са равни "съседните" функции φ_a и φ_{a+1} от редицата $\varphi_0, \varphi_1, \ldots$.

Решение. Искаме $\varphi_a = \varphi_{a+1}$, което означава, че търсим псевдонеподвижни точки на рекурсивната функция $h(a) = a + 1$. Е, вече видяхме, че те съществуват, и освен това са безброй много. □

В *Задача* 3.12 показахме, че съществува примитивно рекурсивна функция s, такава че $W_{s(a)} = \{a\}$ за всяко a. Като приложим втората теорема за рекурсия към функцията $s(a)$, получаваме, че за поне един индекс a ще имаме $\varphi_{s(a)} = \varphi_a$. Тогава, в частност, ще е изпълнено и $W_{s(a)} = W_a$, което заедно с $W_{s(a)} = \{a\}$ ни дава

$$W_a = \{a\}.$$

Така получихме по-кратко решение на *Задача* 3.18.

Задача 3.22. Докажете, че съществуват безброй много a, такива че

1) $\varphi_a = \varphi_a \circ \varphi_a$;

2) $\varphi_a = \varphi_{a+1} \circ \varphi_{a+2}$.

Решение. 1) Единият начин е да тръгнем от функцията

$$f(a, x) \simeq \varphi_a(\varphi_a(x)),$$

която е изчислима, защото можем да си я мислим като $\Phi_1(a, \Phi_1(a, x))$. Значи към f е приложима теоремата за определимост по рекурсия, според която съществуват естествени числа a, такива че

$$\varphi_a(x) \simeq f(a, x)$$

за всяко x. Според забележката след края на *Твърдение* 3.5, тези a са безброй много. От избора на f се вижда, че всички те удовлетворяват условието, защото

$$\varphi_a(x) \simeq f(a, x) \simeq \varphi_a(\varphi_a(x))$$

за всяко x. Оттук, разбира се, и $\varphi_a = \varphi_a \circ \varphi_a$ за безброй много a.

Вторият начин да решим задачата е да се възползваме от ефективността на оператора $\Gamma_{comp}\colon\ \mathcal{F}_1 \times \mathcal{F}_1 \longrightarrow \mathcal{F}_1$, който се дефинира с равенството

$$\Gamma_{comp}(f, g) = f \circ g.$$

От *Задача* 3.15 знаем, че съществува рекурсивна функция $comp$, такава че за всяко a и b:

$$\Gamma_{comp}(\varphi_a, \varphi_b) = \varphi_{comp(a,b)}, \quad \text{или все едно,} \quad \varphi_a \circ \varphi_b = \varphi_{comp(a,b)}.$$

Нека $h(a) \overset{\text{деф}}{=} comp(a, a)$. Тогава $\varphi_{h(a)} = \varphi_a \circ \varphi_a$ и значи за всяка псевдонеподвижна точка a на h ще имаме $\varphi_a = \varphi_{h(a)} = \varphi_a \circ \varphi_a$.

За подусловие 2) разсъждавайте по аналогия с първия начин за решаване на 1). Съобразете защо вторият начин тук е неприложим. □

Ето и едно любопитно приложение на втората теорема за рекурсия.

Задача 3.23. Докажете, че за всеки компютърен вирус съществуват безброй много програми, чието действие той не може да промени.

Решение. Ако си представяме (идеализирано) вируса като програма, която променя кодовете на програмите, то той всъщност е рекурсивна функция — да кажем, $v(a)$, такава че $P_{v(a)}$ е резултатът от действието на вируса върху P_a. Да фиксираме $n \geq 1$. Знаем, че има безброй много a, за които $\varphi^{(n)}_{v(a)} = \varphi^{(n)}_a$. Това означава, че програмите P_a и $P_{v(a)}$ са еквивалентни, т.е. пресмятат една и съща n-местна функция. Значи всяка такава P_a остава семантично непроменена от вируса. □

Задача за ЕК. Докажете, че съществува рекурсивна функция g, такава че за всяко n числото $g(n)$ е индекс на функцията g^n.

Задача за ЕК. Докажете, че съществува *инективна* и рекурсивна функция g, такава че за всяко n числото $g(n)$ е индекс на g.

Забележка. Искаме g да изброява *различни* свои индекси. Без изискването за инективност, едно очевидно решение е $g = \varphi_a$, където a е код на самовъзпроизвеждаща се програма от *Пример* 3.2 1). Тогава ще имаме, че за всяко n

$$g(n) = \varphi_a(n) = a.$$

Задача за ЕК. Измислете някакво автореферентно свойство на програма за МНР и докажете, че има безброй много програми с това свойство. ☺

3.5 Неподвижни точки на оператори

3.5.1 Неподвижни и най-малки неподвижни точки

Нека $\Gamma : \mathcal{F}_n \longrightarrow \mathcal{F}_k$ е произволен оператор. Функцията f наричаме *неподвижна точка* на оператора Γ, ако

$$\Gamma(f) = f.$$

Ясно е, че за да говорим за неподвижни точки на Γ, трябва броят на аргументите на f и на резултата $\Gamma(f)$ да е един и същ, т.е. трябва Γ да е оператор от тип $(k \rightarrow k)$.

Определение 3.3. Казваме, че f е *най-малка неподвижна точка (н.м.н.т.)* на оператора Γ, ако:

1) f е неподвижна точка на Γ;

2) за всяка неподвижна точка g на Γ е вярно, че $f \subseteq g$.

Ако съществува, най-малката неподвижна точка на Γ е единствена: наистина, ако Γ има две най-малки неподвижни точки f и g, то от второто условие на дефиницията ще имаме, че $f \subseteq g$ и $g \subseteq f$ и следователно $f = g$. Тази единствена най-малка неподвижна точка на Γ ще означаваме с f_Γ. Друго често срещано означение е $lfp(\Gamma)$ (от ***l**east **f**ixed **p**oint*).

Една основна мотивация за интереса към неподвижните точки на операторите са рекурсивните програми. Да разгледаме няколко примера.

Пример 3.3. Нека R е следната рекурсивна програма:

$$R: \quad f(x) \;=\; \underbrace{\texttt{if} \quad x \;=\; 0 \quad \texttt{then}\; 1 \quad \texttt{else}\; x.f(x-1)}_{\Gamma(f)(x)}$$

На тялото на R можем да съпоставим оператора $\Gamma\colon \mathcal{F}_1 \longrightarrow \mathcal{F}_1$, дефиниран като:

$$\Gamma(f)(x) \simeq \begin{cases} 1, & \text{ако } x = 0 \\ x.f(x-1), & \text{иначе.} \end{cases}$$

Ясно е, че функцията f, която R пресмята, удовлетворява условието

$$f(x) \simeq \begin{cases} 1, & \text{ако } x = 0 \\ x.f(x-1), & \text{иначе.} \end{cases}$$

С други думи

$$f(x) \simeq \Gamma(f)(x) \quad \text{за всяко} \quad x \in \mathbb{N},$$

или все едно, $f = \Gamma(f)$, т.е. f е *неподвижна точка* на оператора Γ.

Този този оператор има единствена неподвижна точка — функцията *факториел*. Наистина, нека f е произволна неподвижна точка на Γ, т.е. за f е изпълнено

$$f(x) \simeq \begin{cases} 1, & \text{ако } x \;=\; 0 \\ x.f(x-1), & \text{иначе.} \end{cases}$$

С индукция относно $x \in \mathbb{N}$ ще покажем, че $\forall x\ f(x) = x!$.

При $x = 0$ имаме $f(0) = 1 \stackrel{\text{деф}}{=} 0!$, а ако допуснем, че $f(x) = x!$ за някое $x \geq 0$, то за $x+1$ получаваме последователно:

$$f(x+1) \simeq (x+1).f(x) = (x+1).x! = (x+1)!.$$

Да изследваме и неподвижните точки на операторите, идващи от две съвсем прости програми:

Пример 3.4. 1) $R: \quad \texttt{f(x)} \;=\; \texttt{g(x)}, \quad$ където g е фиксирана функция. R формално не е рекурсивна, но не пречи да изследваме оператора, който тя определя — константният оператор, при който за всяка $f \in \mathcal{F}_1$ имаме

$$\Gamma(f) \stackrel{\text{деф}}{=} g.$$

Този оператор има единствена неподвижна точка и това е g (която, разбира се, е и функцията, която R ще пресметне).

2) $R:$ $\texttt{f(x)} = \texttt{f(x)}$
Тази програма пресмята никъде недефинираната функция $\emptyset^{(1)}$. Операторът, който тя определя, е операторът идентитет

$$\Gamma(f) \stackrel{\text{деф}}{=} f,$$

на който очевидно *всяка* функция е неподвижна точка, а най-малката неподвижна точка ще е точно $\emptyset^{(1)}$.

Горният оператор е пример за оператор с *континуум много* неподвижни точки. Ето и два последни примера на рекурсивни програми, които определят оператори с изброимо много неподвижни точки.

Пример 3.5. Нека R е програмата
$R:$ $\texttt{f(x)} = \texttt{if x = 0 then 0 else f(x + 1)}$
Да означим с Γ оператора, който R задава:

$$\Gamma(f)(x) \simeq \begin{cases} 0, & \text{ако } x = 0 \\ f(x+1), & \text{иначе.} \end{cases}$$

Ако f е н.т. на Γ, то за нея е вярно, че

$$f(x) \simeq \begin{cases} 0, & \text{ако } x = 0 \\ f(x+1), & \text{иначе.} \end{cases}$$

Следователно $f(0) = 0$, а при всяко $x > 0$ би трябвало $f(x) \simeq f(x+1)$, което означава, че

$$f(1) \simeq f(2) \simeq f(3) \simeq \dots \ .$$

Следователно f или трябва да има една и съща стойност при $x > 0$, или въобще да няма стойност. С други думи, f или е някоя от функциите f_c, където f_c (за $c \in \mathbb{N}$) има вида

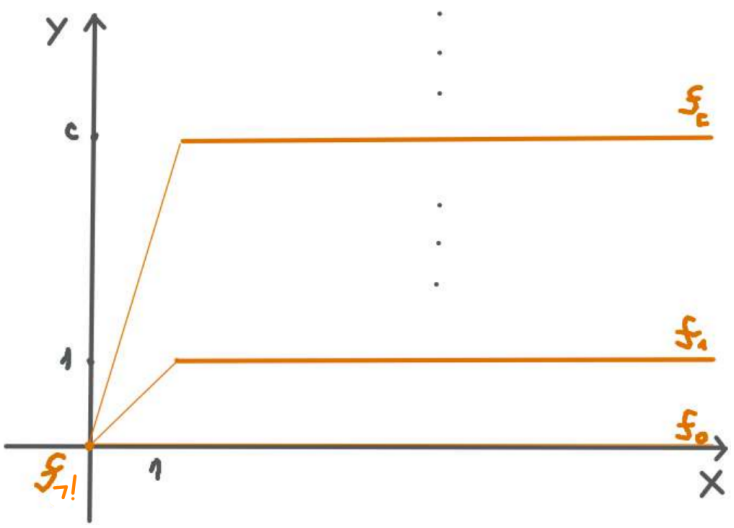
$$f_c(x) \simeq \begin{cases} 0, & \text{ако } x = 0 \\ c, & \text{ако } x > 0, \end{cases}$$

или f е $f_{\neg!}$, където

$$f_{\neg!}(x) \simeq \begin{cases} 0, & \text{ако } x = 0 \\ \neg!, & \text{ако } x > 0, \end{cases}$$

Ясно е, че най-малката н.т. на Γ ще е горната функция $f_{\neg!}$.

Ето как изглеждат графично тези функции.



Пример 3.6. Нека R е програмата

R: `f(x,y) = if x = 0 then 0 else f(x − 1, f(x, y))`

Операторът, който R задава, е следният:

$$\Gamma(f)(x,y) \simeq \begin{cases} 0, & \text{ако } x = 0 \\ f(x-1, f(x,y)), & \text{иначе.} \end{cases}$$

Нека $f = \Gamma(f)$, или все едно

$$f(x,y) \simeq \begin{cases} 0, & \text{ако } x = 0 \\ f(x-1, f(x,y)), & \text{иначе.} \end{cases}$$

Лесно се вижда, че най-малката функция, която удовлетворява това условие, е

$$f(x,y) \simeq \begin{cases} 0, & \text{ако } x = 0 \\ \neg!, & \text{ако } x > 0. \end{cases}$$

Същевременно и всяка от функциите f_c, $c > 0$, където

$$f_c(x,y) \simeq \begin{cases} 0, & \text{ако } x \leq c \\ \neg!, & \text{ако } x > c, \end{cases}$$

също е неподвижна точка на Γ. Най-голямата неподвижна точка е

$$f_\infty(x,y) \overset{\text{деф}}{=} 0 \quad \text{за всяко } x, y.$$

Видяхме, че разнообразието при неподвижните точки на операторите е голямо. Те могат да имат една, няколко или безброй много неподвижни точки. Обаче едно нещо се набиваше на очи — че всички те имат най-малка неподвижна точка.

Дали това винаги е така? Не. Ще завършим тази встъпителна част с още два примера — за оператор, който няма *най-малка* неподвижна точка (но има неподвижни точки) и за оператор, който въобще няма неподвижни точки. Особеното и при двата оператора е, че те, за разлика от вече разгледаните примери, "не идват" от рекурсивни програми.

Пример 3.7. Нека f_0 и f_1 са две различни тотални функции (бихме могли да си мислим за константните функции $\lambda x.0$ и $\lambda x.1$.) Да определим операторите Γ и Δ както следва:

$$\Gamma(f) = \begin{cases} f_0, & \text{ако } f = f_0 \\ f_1, & \text{ако } f \neq f_0, \end{cases}$$

$$\Delta(f) = \begin{cases} f_1, & \text{ако } f = f_0 \\ f_0, & \text{ако } f \neq f_0. \end{cases}$$

За да определим неподвижните точки на Γ, да приемем, че $\Gamma(f) = f$. Като разгледаме двете възможности за f — да е равна или да е различна от f_0, стигаме до извода, че $f = f_0$ или $f = f_1$. Следователно Γ има две неподвижни точки — f_0 и f_1, но няма най-малка неподвижна точка.

С подобни разсъждения се показва, че операторът Δ няма никакви неподвижни точки.

3.5.2 Неподвижни точки на ефективни оператори

Нека $\Gamma : \mathcal{F}_n \longrightarrow \mathcal{F}_k$ е ефективен оператор. Това, съгласно *Определение* 3.2 означава, че съществува рекурсивна функция h (индексната функция на Γ), такава че за всяко a:

$$\Gamma(\varphi_a^{(n)}) \;=\; \varphi_{h(a)}^{(k)}.$$

Оказва се, че всеки ефективен оператор от подходящия тип $(n \to n)$ има поне една изчислима неподвижна точка.

Твърдение 3.6. Нека $\Gamma : \mathcal{F}_n \longrightarrow \mathcal{F}_n$ е ефективен оператор. Тогава съществува изчислима функция f, такава че $\Gamma(f) = f$.

Доказателство. Нека рекурсивната функция h е индексна за оператора, т.е. за всяко a е изпълнено

$$\Gamma(\varphi_a^{(n)}) \;=\; \varphi_{h(a)}^{(n)}.$$

Съгласно втората теорема за рекурсия, съществува индекс a, такъв че $\varphi_{h(a)}^{(n)} = \varphi_a^{(n)}$. Следователно

$$\Gamma(\varphi_a^{(n)}) \overset{\text{деф}}{=} \varphi_{h(a)}^{(n)} \;=\; \varphi_a^{(n)},$$

с други думи, функцията $\varphi_a^{(n)}$ е неподвижна точка на Γ. □

Забележка. От *Твърдение* 3.5 знаем, че всяка рекурсивна функция h има безброй много псевдонеподвижни точки, т.е има безброй много индекси a, за които $\varphi_{h(a)}^{(k)} = \varphi_a^{(k)}$. Разбира се, това съвсем не означава, че и операторът Γ с индексна функция h ще има безброй много неподвижни точки. Може просто всички псевдонеподвижни точки на h да са индекси на една и съща функция. Такъв е случаят с оператора, свързан с функцията на Акерман от следващата задача:

Задача 3.24. Докажете, че функцията на Акерман, която се дефинира с условията

$$\left|\begin{array}{l} f(0,y) \simeq y+1 \\ f(x+1,0) \simeq f(x,1) \\ f(x+1,y+1) \simeq f(x,f(x+1,y)). \end{array}\right.$$

е рекурсивна.

Решение. От решението на *Задача* 1.18 знаем, че съществува единствена функция f, удовлетворяваща горните равенства. Да препишем дефиницията на f в следния по-удобен за нашите цели вид:

$$f(x,y) = \underbrace{\begin{cases} y+1, & \text{ако } x = 0 \\ f(x,1), & \text{ако } x > 0 \;\&\; y = 0 \\ f(x,f(x+1,y)), & \text{ако } x > 0 \;\&\; y > 0. \end{cases}}_{\Gamma(f)(x,y)} \tag{3.6}$$

Да означим с $\Gamma : \mathcal{F}_2 \longrightarrow \mathcal{F}_2$ оператора, определен от дясната част на горното равенство:

$$\Gamma(f)(x,y) \overset{\text{деф}}{\simeq} \begin{cases} y+1, & \text{ако } x = 0 \\ f(x,1), & \text{ако } x > 0 \;\&\; y = 0 \\ f(x,f(x+1,y)), & \text{ако } x > 0 \;\&\; y > 0. \end{cases}$$

Да се убедим най-напред, че този оператор е ефективен. Ще използваме критерия от *Твърдение* 3.4. За тази цел разглеждаме функцията

$$F(a,x,y) \overset{\text{деф}}{\simeq} \Gamma(\varphi_a^{(2)})(x,y) \simeq \begin{cases} y+1, & \text{ако } x=0 \\ \varphi_a^{(2)}(x,1), & \text{ако } x>0 \ \&\ y=0 \\ \varphi_a^{(2)}(x,\varphi_a^{(2)}(x+1,y)), & \text{ако } x>0 \ \&\ y>0. \end{cases}$$

Преписваме F чрез универсалната функция Φ_2:

$$F(a,x,y) \simeq \begin{cases} y+1, & \text{ако } x=0 \\ \Phi_2(a,x,1), & \text{ако } x>0 \ \&\ y=0 \\ \Phi_2(a,x,\Phi_2(a,x+1,y)), & \text{ако } x>0 \ \&\ y>0. \end{cases}$$

Сега вече никой не се съмнява, че F е изчислима и значи операторът Γ е ефективен. Тогава според *Твърдение* 3.6 той ще има поне една изчислима неподвижна точка $\varphi_a^{(2)}$.

Откъде, обаче, да сме сигурни, че това ще е точно функцията на Акерман? Ами всяка неподвижна точка на Γ удовлетворява условията (3.6), а знаем, че има само една функция която може да удовлетворява тези условия и това е функцията F на Акерман. Следователно $F = \varphi_a^{(2)}$, с други думи, F е изчислима. Но тя е и тотална, и значи общо е рекурсивна.

Да обърнем внимание, че за безброй много a, $\Gamma(\varphi_a^{(2)}) = \varphi_a^{(2)}$, но неподвижната точка на този оператор е само една.

Друг начин да решим задачата е като използваме директно теоремата за определимост по рекурсия. За целта разсъждаваме така: ако функцията на Акерман е рекурсивна, тя би трябвало да е от вида $\varphi_a^{(2)}$ за някое a. Значи е достатъчно да съобразим, че функция от вида $\varphi_a^{(2)}$ удовлетворява условието (3.6):

$$\varphi_a^{(2)}(x,y) \simeq \underbrace{\begin{cases} y+1, & \text{ако } x=0 \\ \varphi_a^{(2)}(x,1), & \text{ако } x>0 \ \&\ y=0 \\ \varphi_a^{(2)}(x,\varphi_a^{(2)}(x+1,y)), & \text{ако } x>0 \ \&\ y>0. \end{cases}}_{F(a,x,y)}$$

Да означим дясната част на това равенство с $F(a,x,y)$. Тази функция е изчислима, както вече отбелязахме по-горе. Следователно съществува естествено число a, такова че

$$\varphi_a^{(2)}(x,y) \simeq F(a,x,y).$$

Ясно е, че $\varphi_a^{(2)}$ ще е точно функцията на Акерман, защото тя единствена удовлетворява (3.6). □

Накрая да обясним защо индексите a, такива че

$$\varphi_{h(a)} = \varphi_a,$$

се наричат псевдонеподвижни точки на h.

Кое може да е изображението, на което φ_a да е неподвижна точка? Звучи логично това да е операторът Γ, който върху изчислимите функции се задава с равенството

$$\Gamma(\varphi_a) \stackrel{\text{деф}}{=} \varphi_{h(a)}.$$

Това определение, обаче, изобщо казано е некоректно. Ако такъв оператор съществуваше, то за него би трябвало да е изпълнено условието

$$\varphi_a = \varphi_b \implies \Gamma(\varphi_a) = \Gamma(\varphi_b),$$

което преписано чрез h изглежда така:

$$\varphi_a = \varphi_b \implies \varphi_{h(a)} = \varphi_{h(b)}.$$

Далеч не всяка рекурсивна функция има това много специално свойство (вече споменахме, че функциите, които го имат, се наричат екстензионални.)

Ако, обаче, разглеждаме оператор, който преработва *програми* в *програми*, вече ще имаме коректна дефиниция. Да си спомним за множеството от всички програми

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

и нека изображението $\Delta\colon \mathbb{P} \longrightarrow \mathbb{P}$ се дефинира така: за всяко a

$$\Delta(P_a) = P_{h(a)}.$$

Тук вече нямаме проблем с коректността, защото с всяка програма свързваме *единствено* число — нейния код.

Ако програмите P_a и P_b са еквивалентни (т.е. ако $\varphi_a = \varphi_b$), нека този факт отбелязваме така: $P_a \approx P_b$. Тогава е ясно, че ако $\varphi_a = \varphi_{h(a)}$, то $P_a \approx \Delta(P_a)$. Тъкмо заради факта, че имаме $P_a \approx \Delta(P_a)$, а не $P_a = \Delta(P_a)$, говорим за *псевдо*неподвижни (а не неподвижни) точки на h.

3.6 Първа теорема за рекурсия

Формулировката и доказателството на първата теорема за рекурсия изискват известна предварителна подготовка, с която ще се заемем за начало.

3.6.1 Компактни оператори

За начало ще дефинираме два типа оператори — монотонни и компактни и ще покажем връзката между тях.

Определение 3.4. Казваме, че операторът $\Gamma : \mathcal{F}_k \longrightarrow \mathcal{F}_m$ е *монотонен*, ако за всяка двойка функции $f, g \in \mathcal{F}_k$ е изпълнено условието:

$$f \subseteq g \implies \Gamma(f) \subseteq \Gamma(g).$$

За дефиницията на втория тип оператори — компактните, ще ни трябва понятието *крайна функция*. Да напомним, че една функция е крайна, ако е дефинирана само в краен брой точки. Всяка крайна функция носи само *крайна информация* — информация за стойностите си в точките от дефиниционното си множество. За сравнение: една *тотална* едноместна функция f се характеризира с безкрайната редица от стойностите си $f(0), f(1), \ldots$.

По-надолу с θ ще означаваме само крайни функции.

Определение 3.5. Операторът $\Gamma : \mathcal{F}_k \longrightarrow \mathcal{F}_m$ наричаме *компактен*, ако за всяка функция $f \in \mathcal{F}_k$, всяко $\bar{x} \in \mathbb{N}^k$ и всяко $y \in \mathbb{N}$ е в сила еквивалентността:

(3.7)

Интуитивно, за един компактен оператор Γ е вярно, че *ако* $\Gamma(f)(\bar{x})$ има стойност, то тази стойност се получава като се използва само крайна информация от аргумента f — това е точно крайната функция θ от горното определение. Разбира се, точките, в които тази крайна θ е дефинирана, могат да зависят както от f, така и от $\bar{x}$.

Например, при оператора за диагонализация Γ_d имаме, че ако

$$\Gamma_d(f)(x) \overset{\text{деф}}{\simeq} f(x,x) \simeq y,$$

то резултатът y зависи от стойността на f само в една точка — точката (x,x). Следователно най-малката функция $\theta \subseteq f$, от която се определя резултатът $\Gamma_d(f)(x)$ е с дефиниционна област $\{(x,x)\}$.

За оператора

$$\Gamma_{sum}(f)(x) \simeq f(0) + \cdots + f(x)$$

имаме, че ако $\Gamma_{sum}(f)(x) \simeq y$, то y се определя от стойностите на f в точките $0, 1, \ldots, x$, и следователно $Dom(\theta)$ *трябва* да включва точките $0, 1, \ldots, x$, а най-малката θ с това свойство е тази, за която $Dom(\theta) = \{0, 1, \ldots, x\}$.

При оператора за композиция, който се дефинира с условието $\Gamma_{comp}(f)(x) \simeq f(f(x))$ е ясно, че ако $\Gamma_{comp}(f)(x) \simeq y$, то $Dom(\theta)$ трябва да включва точките x и $f(x)$, като втората точка вече зависи и от f.

Въобще, всички оператори, които сме давали дотук като примери, са компактни.

За нашите цели се оказва удобна следната еквивалентна формулировка на дефиницията за компактност:

Твърдение 3.7. Операторът $\Gamma : \mathcal{F}_k \longrightarrow \mathcal{F}_m$ е компактен тогава и само тогава, когато са изпълнени условията:

1) Γ е монотонен;

2) За всички $f \in \mathcal{F}_k$, $\bar{x} \in \mathbb{N}^k$ и $y \in \mathbb{N}$ е в сила импликацията:

$$\Gamma(f)(\bar{x}) \simeq y \implies \exists\theta(\theta \subseteq f \ \& \ \theta \text{ е крайна } \& \ \Gamma(\theta)(\bar{x}) \simeq y).$$

Доказателство. Нека Γ е компактен. Имаме да проверим само монотонността на Γ. За целта да вземем две функции f и g, такива че $f \subseteq g$. За да покажем, че $\Gamma(f) \subseteq \Gamma(g)$, да приемем, че за някои $\bar{x}, y$

$$\Gamma(f)(\bar{x}) \simeq y.$$

Тогава от правата посока на (3.7) ще съществува крайна функция $\theta \subseteq f$, за която $\Gamma(\theta)(\bar{x}) \simeq y$. Имаме $\theta \subseteq f$ и $f \subseteq g$, и значи $\theta \subseteq g$, защото $\subseteq$ е транзитивна. Сега отново от условието за компактност на Γ, но прочетено наобратно, достигаме до $\Gamma(g)(\bar{x}) \simeq y$. Получихме общо, че

$$\Gamma(f)(\bar{x}) \simeq y \implies \Gamma(g)(\bar{x}) \simeq y,$$

и понеже $\bar{x}$ и y бяха произволни, то наистина $\Gamma(f) \subseteq \Gamma(g)$.

Нека сега са в сила условията 1) и 2). Трябва да проверим само обратната посока на условието за компактност (3.7). Ако се вгледаме в него, виждаме, че то е някаква специална монотонност на Γ, отнасяща се само за случаите, когато по-малката функция е крайна.
Наистина, нека дясната част на (3.7) е в сила, т.е. за някоя крайна $\theta \subseteq f$ е вярно, че

$$\Gamma(\theta)(\bar{x}) \simeq y.$$

Но операторът Γ е монотонен, и щом $\theta \subseteq f$, то и $\Gamma(\theta) \subseteq \Gamma(f)$. Оттук, имайки предвид, че $\Gamma(\theta)(\bar{x}) \simeq y$, веднага получаваме, че и $\Gamma(f)(\bar{x}) \simeq y$, което и трябваше да покажем. □

Следствие 3.1. Всеки компактен оператор е монотонен.

Както не е трудно да се предположи, обратната посока на горното следствие не е вярна. Ето един контрапример:

Пример 3.8. Следващият оператор Γ е монотонен, но не е компактен:

$$\Gamma(f) = \begin{cases} \emptyset^{(1)}, & \text{ако } f \text{ е крайна} \\ f, & \text{иначе.} \end{cases}$$

Доказателство. Монотонността на Γ се проверява непосредствено като се разгледат трите възможности за $f \subseteq g$:
– f и g – крайни;
– f – крайна, g – безкрайна;
– f и g – безкрайни.
За да се убедим, че Γ не е компактен, е достатъчно да вземем коя да е тотална функция f. За произволно естествено x имаме $\Gamma(f)(x) \overset{\text{деф}}{=} f(x)$ и следователно $\Gamma(f)(x)$ има стойност. От друга страна, за всяка крайна θ, $\Gamma(\theta) \overset{\text{деф}}{=} \emptyset^{(1)}$ и следователно $\Gamma(\theta)(x)$ няма стойност, т.е. условието за компактност (3.7) не може да е в сила. □

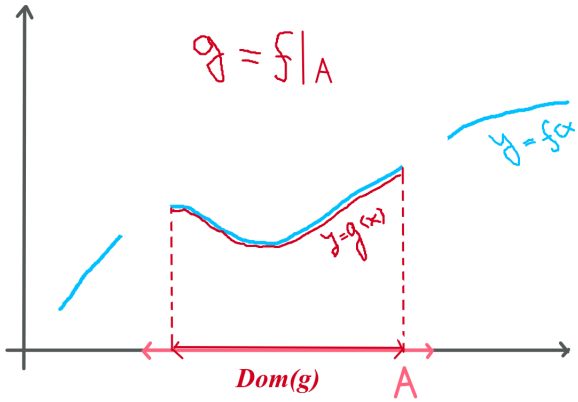
Да обърнем внимание, че горният оператор е доста неестествен от изчислителна гледна точка в следния смисъл: Да предположим, че разполагаме с програма за f. За да пресметнем $\Gamma(f)(x)$, трябва да проверим дали f е крайна функция — нещо, което интуитивно е ясно, че няма как да стане алгоритмично за краен брой стъпки.

Всички оператори, които разглеждахме досега (с изключение на контрапримерите от *Примери* 3.7 и 3.8) са компактни. В следващата задача ще проверим компакността на някои от тях. В решенията се оказва удобно следното означение:

Нека f е n-местна функция, а A е подмножество на $\mathbb{N}^n$. *Рестрикция на f до множеството A* ще наричаме функцията $g \in \mathcal{F}_n$, за която:

$$Dom(g) \;=\; Dom(f) \cap A \quad \& \quad g(\bar{x}) \simeq f(\bar{x}) \text{ за всяко } \bar{x} \in Dom(g).$$

Рестрикцията на f до множеството A ще означаваме с $f \restriction A$.



Задача 3.25. Докажете, че следващите оператори са компактни:

а) операторът за диагонализация $\Gamma_d : \mathcal{F}_2 \longrightarrow \mathcal{F}_1$, който се дефинира с $\Gamma_d(f)(x) \simeq f(x,x)$ за всяко $x \in \mathbb{N}$;

б) операторът Γ_{sq}: $\mathcal{F}_1 \longrightarrow \mathcal{F}_1$ със следната дефиниция: $\Gamma_{sq}(f) = f \circ f$;

в) операторът за сумиране Γ_{sum} : $\mathcal{F}_1 \longrightarrow \mathcal{F}_1$, който за всяко $x \in \mathbb{N}$:

$$\Gamma_{sum}(f)(x) \simeq \sum_{z=0}^{x} f(z);$$

г) операторът Γ, свързан с функцията на Акерман:

$$\Gamma(f)(x,y) \simeq \begin{cases} y+1, & \text{ако } x = 0 \\ f(x-1,0), & \text{ако } x > 0 \ \& \ y = 0 \\ f(x-1, f(x, y-1)), & \text{ако } x > 0 \ \& \ y > 0. \end{cases}$$

Решение. Ще се възползваме от НДУ, което формулирахме в *Твърдение* 3.7, т.е. за всеки от операторите ще покажем, че е монотонен и че за него е в сила правата посока на условието за компактност (3.7).

а) Монотонност: да вземем две функции f и g от $\mathcal{F}_2$, такива че $f \subseteq g$. За да видим, че и $\Gamma_d(f) \subseteq \Gamma_d(g)$, следваме определението на релацията $\subseteq$:

$$\Gamma_d(f) \subseteq \Gamma_d(g) \overset{\text{деф}}{\iff} \forall x \forall y \ (\Gamma_d(f)(x) \simeq y \implies \Gamma_d(g)(x) \simeq y).$$

Наистина, да вземем произволни естествени x и y и да приемем, че $\Gamma_d(f)(x) \simeq y$. Трябва да покажем, че и $\Gamma_d(g)(x) \simeq y$.

Условието $\Gamma_d(f)(x) \simeq y$ означава $f(x,x) \simeq y$. Но $f \subseteq g$, следователно и $g(x,x) \simeq y$, или все едно $\Gamma_d(g)(x) \simeq y$. Понеже x и y бяха произволни, можем да заключим, че $\Gamma_d(f) \subseteq \Gamma_d(g)$.

Да проверим, че за Γ_d е в сила импликацията

$$\forall f_{\in \mathcal{F}_2} \forall x \forall y \ (\Gamma_d(f)(x) \simeq y \implies \exists \theta (\theta \subseteq f \ \& \ \theta \text{ е крайна } \& \ \Gamma_d(\theta)(x) \simeq y)).$$

За целта фиксираме функция $f \in \mathcal{F}_1$ и естествени числа x и y и приемаме, че $\Gamma_d(f)(x) \simeq y$, което ще рече — $f(x,x) \simeq y$. Очевидно резултатът y зависи само от стойността на f в точката (x,x). Тогава е ясно коя крайна функция $\theta \subseteq f$ да изберем, така че да си осигурим $\Gamma_d(\theta)(x) \simeq y$ — полагаме θ да е *рестрикцията на* f до множеството $\{(x,x)\}$:

$$\theta := f \restriction \{(x,x)\}.$$

От избора на θ автоматично следва, че тя е подфункция на f, дефинирана в най-много една точка — точката (x,x). Но ние имаме, че $(x,x) \in Dom(f)$, откъдето

$$\theta(x,x) = f(x,x)\ (= y).$$

Оттук веднага $\Gamma_d(\theta)(x) \overset{\text{деф}}{\simeq} \theta(x,x) \simeq y$.

б) По дефиниция

$$\Gamma_{sq}(f)(x) \overset{\text{деф}}{\simeq} (f \circ f)(x) \simeq f(f(x)).$$

За да се убедим, че и този оператор е монотонен, вземаме отново произволни функции f, g от $\mathcal{F}_1$, такива че $f \subseteq g$. Да приемем, че $\Gamma_{sq}(f)(x) \simeq y$ за някои $x, y \in \mathbb{N}$. Това означава, че $f(f(x)) \simeq y$. В такъв случай, съгласно нашата дефиниция за суперпозиция, със сигурност $f(x)$ ще е дефинирано, т.е. $f(x) \simeq z$ за някое z. Понеже x и z са от $Dom(f)$, а $f \subseteq g$, то веднага $f(x) = g(x)$ и $f(z) = g(z)$. Но тогава

$$\Gamma_{sq}(g)(x) \simeq g(g(x)) \simeq g(\underbrace{f(x)}_{z}) \simeq f(\underbrace{f(x)}_{z}) \simeq y,$$

което и трябваше да покажем.

Насочваме се към проверка на импликацията

$$\forall f_{\in \mathcal{F}_1} \forall x \forall y\ (\Gamma_{sq}(f)(x) \simeq y \implies \exists \theta(\theta \subseteq f\ \&\ \theta \text{ е крайна } \&\ \Gamma_{sq}(\theta)(x) \simeq y)).$$

Избираме произволни $f \in \mathcal{F}_1$, x и y и приемаме, че $\Gamma_{sq}(f)(x) \simeq y$, т.е. $f(f(x)) \simeq y$. Вече видяхме, че оттук следва, в частност, че $f(x)$ е дефинирана. Можем да вземем

$$\theta := f \restriction \{x,\ f(x)\}.$$

Да отбележим, че това всъщност е единственият възможен избор за θ, ако искаме тя да е подфункция на f, защото само за точките x и $f(x)$ знаем със сигурност, че принадлежат на дефиниционната област на f. Ясно е, че $\theta(x) = f(x)$ и $\theta(f(x)) = f(f(x))$. Тогава

$$\Gamma_{sq}(\theta)(x) \overset{\text{деф}}{\simeq} \theta(\underbrace{\theta(x)}_{=f(x)}) \simeq \theta(f(x)) \simeq f(f(x)) \simeq y.$$

в) Оставяме проверката за монотонността на Γ_{sum} за упражнение и се насочваме директно към второто условие от *Твърдение* 3.7.

За целта, нека $\Gamma_{sum}(f)(x) \simeq y$, т.е. $f(0) + \cdots + f(x) \simeq y$ за някои f, x и y. В частност, $!f(0), \ldots, !f(x)$. Резултатът y се определя от стойностите на

f в точките $0, 1, \ldots, x$, и следователно $Dom(\theta)$ трябва да включва тези точки (и само тях, ако искаме θ да е подфункция на f). Наистина, нека

$$\theta := f \restriction \{0, \ldots, x\}.$$

Така ще имаме

$$\Gamma_{sum}(\theta)(x) \overset{\text{деф}}{\simeq} \theta(0) + \cdots + \theta(x) \simeq f(0) + \cdots + f(x) \simeq y.$$

г) Тъй като операторът

$$\Gamma(f)(x,y) \simeq \begin{cases} y+1, & \text{ако } x = 0 \\ f(x-1,0), & \text{ако } x > 0 \ \& \ y = 0 \\ f(x-1, f(x, y-1)), & \text{ако } x > 0 \ \& \ y > 0. \end{cases}$$

се дефинира с разглеждане на случаи, ще се наложи и ние да разгледаме тези случаи, когато доказваме неговата компактност.

За да видим, че Γ е монотонен, вземаме произволни двуместни функции f и g, такива че $f \subseteq g$ и приемаме, че $\Gamma(f)(x,y) \simeq z$ за някои x, y и z. Искаме да покажем, че $\Gamma(g)(x,y) \simeq z$. Разглеждаме поотделно трите случая от дефиницията на Γ.

1 сл. $x = 0$. Тук очевидно $\Gamma(f)(x,y) \overset{\text{деф}}{\simeq} 0 \simeq \Gamma(g)(x,y)$.

2 сл. $x > 0 \ \& \ y = 0$. В този случай $\Gamma(f)(x,y) \overset{\text{деф}}{\simeq} f(x-1,0) \simeq z$. Но $f \subseteq g$, значи и $g(x-1,0) \simeq z$, откъдето $\Gamma(g)(x,y) \overset{\text{деф}}{\simeq} g(x-1,0) \simeq z$.

3 сл. $x > 0 \ \& \ y > 0$. По определение $\Gamma(f)(x,y) \simeq f(x-1, f(x,y-1))$. От допускането $\Gamma(f)(x,y) \simeq z$ ще имаме $f(x-1, f(x,y-1)) \simeq z$. От дефиницията за суперпозиция следва, че и $f(x, y-1)$ ще е дефинирана. Понеже $f \subseteq g$, ще имаме, че

$$f(x,y-1) = g(x,y-1) \quad \text{и} \quad f(x-1, f(x,y-1)) = f(x-1, g(x,y-1)).$$

Оттук

$$\Gamma(g)(x,y) \overset{\text{деф}}{\simeq} g(x-1, g(x,y-1)) \simeq g(x-1, f(x,y-1)) \simeq f(x-1, f(x,y-1)) \simeq y.$$

Сега се насочваме към проверката на импликацията

$$\forall f_{\in \mathcal{F}_2} \forall x \forall y \, \forall z (\Gamma(f)(x,y) \simeq z \implies \exists \theta (\theta \subseteq f \ \& \ \theta \text{ е крайна } \& \ \Gamma(\theta)(x,y) \simeq z)).$$

Да приемем, че $\Gamma(f)(x,y) \simeq z$ за някои f, x и y. Отново се налага да следваме случаите от дефиницията на Γ.

1 сл. $x = 0$. В този случай $\Gamma(f)$ не зависи от f и значи ако вземем

$$\theta := \emptyset^{(2)}$$

ще имаме със сигурност, че $\theta \subseteq f$ и $\Gamma(\theta)(x,y) \simeq z$. Да отбележим, че това е единственият възможен избор на θ, защото допускането $\Gamma(f)(x,y) \simeq z$ не ни дава никаква информация за $Dom(f)$, в частност, напълно възможно е и f да е $\emptyset^{(2)}$.

2 сл. $x > 0 \,\&\, y = 0$. Условието $\Gamma(f)(x,y) \simeq z$ тук означава $f(x-1,0) \simeq z$. Тогава за

$$\theta := f \restriction \{(x-1,y)\}$$

очевидно ще е изпълнено $\Gamma(\theta)(x,y) \simeq z$.

3 сл. $x > 0 \,\&\, y > 0$. В този случай имаме, че $f(x-1, f(x,y-1)) \simeq z$. Съобразете, че за функцията

$$\theta := f \restriction \{(x,y-1),(x-1,f(x,y-1))\}$$

ще е в сила $\Gamma(\theta)(x,y) \simeq z$. □

3.6.2 Точни горни граници на редици

Нека $f_0, f_1, \ldots, f_n, \ldots$ (или само $\{f_n\}_n$) е редица от k-местни функции.

Определение 3.6. Ще казваме, че функцията g е *горна граница (мажоранта)* на редицата $\{f_n\}_n$, ако за всяко n е вярно, че

$$f_n \subseteq g.$$

g е *точна горна граница* (т.г.г.) на редицата $\{f_n\}_n$, ако:

1) g е горна граница на $\{f_n\}_n$;

2) за всяка горна граница h на тази редица е в сила $g \subseteq h$.

Ако съществува, точната горна граница на $\{f_n\}_n$ е единствена. Наистина, ако допуснем, че редицата $\{f_n\}_n$ има две т.г.гр. g и h, то от условие 2) на дефиницията ще имаме, че $g \subseteq h$ и $h \subseteq g$ и следователно $g = h$. Точната горна граница на редицата $\{f_n\}_n$ ще означаваме с

$$\bigcup_n f_n$$

или само с $\bigcup f_n$.

Оказва се, че ако една редица е *монотонно растяща*, то тя има точна горна граница, която при това се получава по съвсем естествен начин. Да се убедим:

Твърдение 3.8. Всяка монотонно растяща редица $f_0 \subseteq f_1 \subseteq \ldots$ от функции в $\mathcal{F}_k$ притежава точна горна граница f, която се дефинира с условието: за всички естествени $x_1, \ldots, x_k, y$:

$$f(x_1, \ldots, x_k) \simeq y \iff \exists n \; f_n(x_1, \ldots, x_k) \simeq y. \tag{3.8}$$

Доказателство. Най-напред да се убедим, че тази еквивалентност дефинира еднозначна функция. Наистина, нека за някои $\bar{x}, y$ и z е изпълнено

$$f(\bar{x}) \simeq y \quad \text{и} \quad f(\bar{x}) \simeq z.$$

Тогава ще съществуват индекси l и m, за които

$$f_l(\bar{x}) \simeq y \quad \text{и} \quad f_m(\bar{x}) \simeq z.$$

Без ограничение на общността можем да считаме, че $l \leq m$. Тогава $f_l \subseteq f_m$ и щом $f_l(\bar{x}) \simeq y$, то и $f_m(\bar{x}) \simeq y$. Но ние имаме $f_m(\bar{x}) \simeq z$, и значи наистина $y = z$.

Нека сега f_n е произволна функция от редицата $f_0, f_1, \ldots$ От определението на f се вижда, че $G_{f_n} \subseteq G_f$, което означава, че $f_n \subseteq f$. Понеже това е вярно за *всяко* n, то f е горна граница на редицата $\{f_n\}_n$.

За да видим, че тя е най-малката сред горните ѝ граници, да вземем друга горна граница — да кажем, h. Трябва да покажем, че $f \subseteq h$. За целта, нека за произволни $\bar{x}$ и y: $f(\bar{x}) \simeq y$. От определението на f имаме, че тогава за някое n трябва да е изпълнено $f_n(\bar{x}) \simeq y$. Но h е мажоранта на редицата $\{f_n\}_n$, а f_n е член на тази редица, следователно $f_n \subseteq h$, откъдето в частност $h(\bar{x}) \simeq y$. Получихме, че за произволните $\bar{x}, y$ е в сила импликацията:

$$f(\bar{x}) \simeq y \implies h(\bar{x}) \simeq y,$$

което по дефиниция означава, че $f \subseteq h$. Следователно f е точната горна граница на редицата $f_0, f_1, \ldots$ □

Забележка. От определението на f се вижда, че нейната графика е *обединение* на графиките на функциите от редицата $\{f_n\}_n$, което обяснява и означението $\bigcup$ за точна горна граница.

Ще докажем и една спомагателна лема, която ще използваме веднага след това при доказателството на важната теорема на Кнастер-Тарски.

Лема 3.1. Нека $f_0 \subseteq f_1 \subseteq \ldots$ е монотонно растяща редица от функции в $\mathcal{F}_k$ и нека за крайната функция θ е изпълнено:

$$\theta \subseteq \bigcup_n f_n.$$

Тогава $\theta \subseteq f_n$ за някое n.

Доказателство. Нека $Dom(\theta) = \{\bar{x}^1, \dots \bar{x}^l\}$. Можем да предполагаме, че $l \geq 1$, защото ако $l = 0$, т.е. $\theta = \emptyset^{(k)}$, то със сигурност $\theta \subseteq f_0$.

Да фиксираме $1 \leq i \leq l$ и нека

$$\theta(\bar{x}^i) \simeq y_i.$$

Понеже $\theta \subseteq \bigcup f_n$, значи и $(\bigcup f_n)(\bar{x}^i) \simeq y_i$. От последното, като използваме дефиницията за т.г.г. (3.8), получаваме, че съществува n_i, за което

$$f_{n_i}(\bar{x}^i) \simeq y_i.$$

Нека $n = max\{n_1, \dots, n_l\}$. Тогава очевидно $n_i \leq n$ и следователно $f_{n_i} \subseteq f_n$. Сега от $f_{n_i}(\bar{x}^i) \simeq y_i$ ще имаме, че и $f_n(\bar{x}^i) \simeq y_i$. Финално, за всяко $\bar{x}^i \in Dom(\theta)$ е изпълнено $\theta(\bar{x}^i) \simeq y_i \simeq f_n(\bar{x}^i)$, и следователно $\theta \subseteq f_n$. $\square$

3.6.3 Теорема на Кнастер-Тарски

Теоремата на Кнастер-Тарски е един общ резултат за съществуване на най-малка неподвижна точка на компактен оператор. Тя е известна още като *Теорема на Кнастер-Тарски-Клини*, защото Клини посочва начина, по който се *конструира* най-малката неподвижна точка f_Γ — като точна горна граница на подходяща монотонно растяща редицата от функции, които се явяват последователни приближения на f_Γ.

Нека Γ е оператор от тип $(k \to k)$, а f е произволна k-местна функция. За всяко естествено число n, с $\Gamma^n(f)$ ще означаваме функцията, която се получава след n-кратно прилагане на оператора Γ към f:

$$\Gamma^n(f) = \underbrace{\Gamma(\dots\Gamma}_{n \text{ пъти}}(f)\dots).$$

Тогава очевидно

$$\Gamma^0(f) = f$$
$$\Gamma^{n+1}(f) = \Gamma(\Gamma^n(f)).$$

Теорема 3.7. (Теорема на Кнастер-Тарски) Нека $\Gamma : \mathcal{F}_k \longrightarrow \mathcal{F}_k$ е компактен оператор. Тогава Γ има най-малка неподвижна точка f_Γ, която се получава по следния начин:

$$f_\Gamma = \bigcup_n \Gamma^n(\emptyset^{(k)}).$$

Доказателство. Да означим с f_n функцията $\Gamma^n(\emptyset^{(k)})$. Тогава

$$\Gamma^0(\emptyset^{(k)}) = \emptyset^{(k)} \quad \text{и} \quad \Gamma^{n+1}(\emptyset^{(k)}) = \Gamma(\Gamma^n(\emptyset^{(k)})) = \Gamma(f_n)$$

Следователно редицата $\{f_n\}_n$ удовлетворява рекурентната връзка

$$f_0 = \emptyset^{(k)}$$
$$f_{n+1} = \Gamma(f_n).$$

Най-напред да се убедим, че тази редица е монотонно растяща. С индукция по n ще покажем, че за всяко естествено n

$$f_n \subseteq f_{n+1}.$$

База $n = 0$: по определение $f_0 = \emptyset^{(k)}$ и тогава очевидно $f_0 \subseteq f_1$.
Сега да приемем, че за някое n

$$f_n \subseteq f_{n+1}.$$

Операторът Γ е компактен, и в частност — монотонен, съгласно *Следствие* 3.1. Тогава от горното включване ще имаме

$$\Gamma(f_n) \subseteq \Gamma(f_{n+1}),$$

или все едно $f_{n+1} \subseteq f_{n+2}$, с което индуктивната стъпка е приключена.

Щом редицата $f_0, f_1, \ldots$ е монотонно растяща, съгласно *Твърдение* 3.8 тя притежава точна горна граница — да я означим с g:

$$g = \bigcup_n f_n.$$

Нашата цел е да покажем, че g е най-малката неподвижна точка на Γ, с други думи, $g = f_\Gamma$.

Да видим първо, че тя е неподвижна точка на Γ, т.е. $\Gamma(g) = g$. Това означава да проверим двете включвания:

$$g \subseteq \Gamma(g) \quad \text{и} \quad \Gamma(g) \subseteq g.$$

За първото е достатъчно да съобразим, че $\Gamma(g)$ е горна граница за дефинираната по-горе редица $\{f_n\}_n$, т.е. $f_n \subseteq \Gamma(g)$ за всяко n. Наистина, при $n = 0$ това е очевидно, а ако $n > 0$, от определението на g имаме, че $f_{n-1} \subseteq g$, откъдето по монотонността на Γ получаваме

$$\underbrace{\Gamma(f_{n-1})}_{f_n} \subseteq \Gamma(g), \quad \text{т.е.} \quad f_n \subseteq \Gamma(g).$$

Следователно $\Gamma(g)$ е горна граница на редицата $\{f_n\}_n$. Но g е точната горна граница на тази редица, и значи $g \subseteq \Gamma(g)$.

За да видим обратното включване $\Gamma(g) \subseteq g$, да приемем, че $\Gamma(g)(\bar{x}) \simeq y$. От дефиницията за компактност (3.7) следва, че тогава за някоя крайна функция $\theta \subseteq g$ ще е изпълнено

$$\Gamma(\theta)(\bar{x}) \simeq y.$$

От $\theta \subseteq g \overset{\text{деф}}{=} \bigcup_n f_n$ по *Лема* 3.1 ще имаме, че съществува n, такова че $\theta \subseteq f_n$. Но тогава и $\Gamma(\theta) \subseteq \Gamma(f_n)$, и значи

$$\Gamma(f_n)(\bar{x}) \simeq y, \quad \text{т.е.} \quad f_{n+1}(\bar{x}) \simeq y.$$

Но $f_{n+1} \subseteq g$, следователно и $g(\bar{x}) \simeq y$. Така получихме, че за произволни $\bar{x}, y$:

$$\Gamma(g)(\bar{x}) \simeq y \implies g(\bar{x}) \simeq y,$$

което означава, че $\Gamma(g) \subseteq g$.

Нека сега h е друга неподвижна точка на Γ. Тъй като g е точна горна граница на $\{f_n\}_n$, за да покажем, че $g \subseteq h$, е достатъчно да видим, че h е горна граница на тази редица, с други думи, че $f_n \subseteq h$ за всяко $n \geq 0$. Това ще проверим с индукция относно n. За $n = 0$ имаме по определение

$$f_0 \overset{\text{деф}}{=} \emptyset^{(k)} \subseteq h.$$

Да предположим, че за някое n

$$f_n \subseteq h.$$

Прилагаме Γ към двете страни на неравенството и получаваме

$$\Gamma(f_n) \subseteq \Gamma(h) = h,$$

т.е. $f_{n+1} \subseteq h$. Сега вече можем да твърдим, че $f_n \subseteq h$ за всяко $n \geq 0$, с други думи, че h е мажоранта на редицата $\{f_n\}_n$ и значи h мажорира и точната ѝ горна граница g, т.е. $g \subseteq h$. □

Забележка. Функциите f_n от горното доказателство имат смисъл на последователни *приближения* (*апроксимации*) на f_Γ.

Преди да сме преминали към задачите, които илюстрират тази теорема, да съобразим следния факт, който се оказва полезен за някои от тях:

Задача 3.26. Нека $\Gamma\colon \mathcal{F}_k \longrightarrow \mathcal{F}_k$ е компактен оператор. За редицата $f_0, f_1, f_2, \ldots$ от последователните приближения на f_Γ да се докаже, че ако за някое n е вярно, че $f_n = f_{n+1}$, то тогава

$$f_n = f_{n+1} = f_{n+2} = \ldots$$

Забележка. Разбира се, в такъв случай ще имаме, че границата на редицата $\{f_n\}_n$ ще бъде тази функция f_n, с други думи $f_\Gamma = f_n$.

Решение. Нека за някое n е изпълнено $f_n = f_{n+1}$. С индукция относно $m \geq n$ ще покажем, че

$$f_n = f_m \quad \text{за всяко} \quad m \geq n.$$

Случаят $m = n$ е ясен, а приемайки, че

$$f_n \;=\; f_m$$

за някое $m \geq n$, след почленно прилагане на Γ ще имаме

$$\Gamma(f_n) \;=\; \Gamma(f_m),$$

или все едно, $f_{n+1} \;=\; f_{m+1}$. Но ние имаме $f_n = f_{n+1}$, откъдето веднага $f_n = f_{m+1}$, с което индуктивната стъпка е проведена.

Ако n е първото естествено число със свойството $f_n = f_{n+1}$, редицата $\{f_n\}_n$ ще изглежда така:

$$f_0 \overset{\text{деф}}{=} \emptyset^{(k)} \subset \; f_1 \; \cdots \subset f_n = \; f_{n+1} \; = f_{n+2} \ldots$$

В този случай се казва, че рекурсията "се затваря" на стъпка n. Разбира се, тогава $\bigcup_n f_n$ ще е тази функция f_n. □

Нашият първи пример ще бъде за рекурсия, която се затваря още на стъпка $n = 1$. Операторът е този от *Пример* 3.6.

Задача 3.27. Като използвате теоремата на Кнастер-Тарски намерете най-малката неподвижна точка на следния оператор Γ:

$$\Gamma(f)(x,y) \simeq \begin{cases} 0, & \text{ако } x = 0 \\ f(x-1, f(x,y)), & \text{иначе.} \end{cases}$$

Решение. Означаваме с f_n функцията $\Gamma^n(\emptyset^{(2)})$. Искаме да намерим *явния вид* на всяка f_n, а оттам — и на самата f_Γ.

Ще използваме, че редицата $\{f_n\}_n$ удовлетворява рекурентната схема

$$\begin{aligned} f_0 &= \emptyset^{(2)} \\ f_{n+1} &= \Gamma(f_n). \end{aligned}$$

Така за първата апроксимация f_1 на f_Γ ще имаме:

$$f_1(x,y) \simeq \Gamma(\emptyset^{(2)})(x,y) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 0, & \text{ако } x = 0 \\ \emptyset^{(2)}(x-1, \emptyset^{(2)}(x,y)), & \text{ако } x > 0 \end{cases} \simeq \begin{cases} 0, & \text{ако } x = 0 \\ \neg!, & \text{ако } x > 0. \end{cases}$$

За следващата апроксимация f_2 получаваме:

$$f_2(x,y) \simeq \Gamma(f_1)(x,y) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 0, & \text{ако } x = 0 \\ f_1(x-1, \underbrace{f_1(x,y)}_{\neg!}), & \text{ако } x > 0 \end{cases} \overset{\text{деф } f_1}{\simeq} \begin{cases} 0, & \text{ако } x = 0 \\ \neg!, & \text{ако } x > 0. \end{cases}$$

Оказа се, че двете апроксимации f_1 и f_2 съвпадат. Но тогава, съгласно *Задача* 3.26, всички следващи апроксимации ще са равни на f_1, т.е. редицата от последователните приближения на f_Γ изглежда така:

$$f_0 \overset{\text{деф}}{=} \emptyset^{(2)} \subset f_1 = f_2 = f_3 \ldots.$$

Ясно е, че границата на тази редица е f_1, и значи $f_\Gamma = f_1$. □

Да приложим теоремата на Кнастер-Тарски за оператора от *Пример* 3.3. Вече знаем, че неговата единствена неподвижна точка е функцията $x!$, но да видим как ще я получим с конструкцията от теоремата.

Задача 3.28. Като използвате теоремата на Кнастер-Тарски, намерете най-малката неподвижна точка на оператора

$$\Gamma(f)(x) \simeq \begin{cases} 1, & \text{ако } x = 0 \\ x.f(x-1), & \text{иначе.} \end{cases}$$

Решение. Означаваме, както по-горе, с f_n функцията $\Gamma^n(\emptyset^{(1)})$. Да напомним, че редицата $\{f_n\}_n$ удовлетворява рекурентната връзка

$$\begin{aligned} f_0 &= \emptyset^{(1)} \\ f_{n+1} &= \Gamma(f_n). \end{aligned}$$

Нашата цел ще бъде да намерим *явния вид* на всяка от тези функции.

Започваме с първата апроксимация f_1 на f_Γ:

$$f_1(x) \simeq \Gamma(\emptyset^{(1)})(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ x.\emptyset^{(1)}(x-1), & \text{ако } x > 0 \end{cases} \simeq \begin{cases} 1, & \text{ако } x = 0 \\ \neg!, & \text{ако } x > 0. \end{cases}$$

За следващата апроксимация f_2 ще имаме:

$$f_2(x) \simeq \Gamma(f_1)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ x.f_1(x-1), & \text{иначе} \end{cases} \overset{\text{деф } f_1}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ 1.1, & \text{ако } x = 1 \\ \neg!, & \text{ако } x > 1. \end{cases}$$

Функцията f_2 можем да препишем още по следния начин:

$$f_2(x) \simeq \begin{cases} x!, & \text{ако } x < 2 \\ \neg!, & \text{иначе,} \end{cases}$$

което ни дава идея какъв би могъл да е общият вид на f_n:

$$f_n(x) \simeq \begin{cases} x!, & \text{ако } x < n \\ \neg!, & \text{иначе.} \end{cases}$$

Ще използваме индукция относно $n \in \mathbb{N}$, за да се убедим, че това е така.

На практика вече проверихме случаите $n = 0, 1$ и 2. Да предположим сега, че f_n има горния вид. Тогава за f_{n+1} ще имаме последователно:

$$f_{n+1}(x) \simeq \Gamma(f_n)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ x.f_n(x-1), & \text{ако } x > 0 \end{cases}$$

$$\overset{\text{и.х. } f_n}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ x.(x-1)!, & \text{ако } x > 0 \ \& \ x-1 < n \\ \neg!, & \text{ако } x - 1 \geq n \end{cases} \simeq \begin{cases} x!, & \text{ако } x < n+1 \\ \neg!, & \text{иначе,} \end{cases}$$

и значи индукционната хипотеза се потвърждава и за $n + 1$.

Сега остава да намерим границата $\bigcup_n f_n$ на редицата $f_0, f_1, \ldots, f_n \ldots$.

Интуитивно е ясно, че тази редица трябва да клони към $x!$, защото f_n е рестрикцията на $x!$ върху множеството $\{0, 1, \ldots, n-1\}$, но да го докажем все пак.

Наистина, да означим с f точната горна граница на редицата $\{f_n\}_n$. По определение

$$f(x) \simeq y \iff \exists n \ f_n(x) \simeq y.$$

Да фиксираме произволно x и да изберем $n = x + 1$. Понеже $Dom(f_n) = \{0, \ldots, n-1\}$, то $x \in Dom(f_n)$. Но там, където е дефинирана, f_n се държи като $x!$; в частност, за нашето x ще имаме, че $f_n(x) = x!$. Тогава и $f(x)$ ще е $x!$. Но x беше произволно, следователно за всяко $x, f(x) = x!$, или все едно, $f_\Gamma(x) = x!$. □

Следващата задача се решава по много подобен начин на *Задача* 3.28.

Задача 3.29. Приложете теоремата на Кнастер-Тарски, за да намерите най-малката неподвижна точка на оператора

$$\Gamma(f)(x) \simeq \begin{cases} 1, & \text{ако } x = 0 \\ 2.f(x-1), & \text{иначе.} \end{cases}$$

Решение. Отново търсим явния вид на последователните приближения на f_Γ. По определение $f_0 = \emptyset^{(1)}$. За функцията f_1 ще имаме:

$$f_1(x) \simeq \Gamma(\emptyset^{(1)})(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ 2.\emptyset^{(1)}(x-1), & \text{ако } x > 0 \end{cases} \simeq \begin{cases} 1, & \text{ако } x = 0 \\ \neg!, & \text{ако } x > 0. \end{cases}$$

За следващата апроксимация f_2 получаваме:

$$f_2(x) \simeq \Gamma(f_1)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ 2.f_1(x-1), & \text{ако } x > 0 \end{cases} \overset{\text{деф } f_1}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ 2.1, & \text{ако } x = 1 \\ \neg!, & \text{ако } x > 1. \end{cases}$$

f_2 можем да препишем и така:

$$f_2(x) \simeq \begin{cases} 2^x, & \text{ако } x < 2 \\ \neg!, & \text{ако } x \geq 2, \end{cases}$$

което ни подсказва, че f_n може би ще е ето тази функция:

$$f_n(x) \simeq \begin{cases} 2^x, & \text{ако } x < n \\ \neg!, & \text{ако } x \geq n. \end{cases}$$

Ще използваме индукция относно n, за докажем, че това е така. Базовият случай $n = 0$ е ясен. Да предположим, че f_n има горния вид. Тогава за f_{n+1} ще имаме последователно:

$$f_{n+1}(x) \simeq \Gamma(f_n)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ 2.f_n(x-1), & \text{ако } x > 0 \end{cases}$$

$$\overset{\text{и.х. } f_n}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ 2.2^{x-1}, & \text{ако } x > 0 \ \& \ x - 1 < n \\ \neg!, & \text{ако } x - 1 \geq n \end{cases} \simeq \begin{cases} 2^x, & \text{ако } \ x < n+1 \\ \neg!, & \text{иначе,} \end{cases}$$

което потвърждава нашата хипотеза за f_{n+1}.

Остана да съобразим, че границата на редицата $\{f_n\}_n$ е функцията 2^x, което се вижда както в предишната задача. □

И в двата примера по-горе наблюдавахме, че n-тата апроксимация на f_Γ е с дефиниционна област множеството $\{0, \ldots, n-1\}$. Това е така, защото рекурсията при тях е примитивна, т.е. $\Gamma(f)(x)$ се определя чрез $f(x-1)$. В следващата задача, обаче, $Dom(f_n)$ е по-широко множество.

Задача 3.30. С помощта на теоремата на Кнастер-Тарски определете най-малката неподвижна точка на оператора

$$\Gamma(f)(x) \simeq \begin{cases} 1, & \text{ако } x \leq 1 \\ x.f(x-2), & \text{иначе;} \end{cases}$$

Решение. Отново търсим явния вид на всяка от апроксимациите $f_0, f_1, \ldots$ на f_Γ. Целта ни е да покажем, че $f_\Gamma = \lambda x.x!!$, където

$$x!! \overset{\text{деф}}{=} \begin{cases} 1, & \text{ако } x = 0 \\ 1.3 \ldots x, & \text{ако } x \text{ е нечетно} \\ 2.4 \ldots x, & \text{ако } x > 0 \text{ е четно.} \end{cases}$$

Започваме с първата апроксимация f_1:

$$f_1(x) \simeq \Gamma(\emptyset^{(1)})(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x \leq 1 \\ x.\emptyset^{(1)}(x-2), & \text{ако } x > 1 \end{cases} \simeq \begin{cases} x!!, & \text{ако } x < 2 \\ \neg!, & \text{ако } x \geq 2. \end{cases}$$

За следващата апроксимация f_2 ще имаме:

$$f_2(x) \simeq \Gamma(f_1)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x \leq 1 \\ x.\ \underbrace{f_1(x-2)}_{(x-2)!! \text{ за } x-2<2}, & \text{ако } x > 1 \end{cases}$$

$$\overset{\text{деф } f_1}{\simeq} \begin{cases} x!!, & \text{ако } x \leq 1 \\ x.(x-2)!!, & \text{ако } x > 1 \ \&\ x-2 < 2 \\ \neg!, & \text{ако } x - 2 \geq 2 \end{cases} \simeq \begin{cases} x!!, & \text{ако } x < 4 \\ \neg!, & \text{ако } x \geq 4. \end{cases}$$

Хипотеза за общия вид на f_n:

$$f_n(x) \simeq \begin{cases} x!!, & \text{ако } x < 2n \\ \neg!, & \text{ако } x \geq 2n. \end{cases}$$

Ще използваме индукция относно $n \in \mathbb{N}$, за да се убедим, че това е така.

На практика вече проверихме случаите $n = 0, 1$ и 2. Да предположим сега, че f_n има горния вид. Тогава за f_{n+1} можем да запишем:

$$f_{n+1}(x) \simeq \Gamma(f_n)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x \leq 1 \\ x.\ \underbrace{f_n(x-2)}_{(x-2)!! \text{ за } x-2<2n}, & \text{ако } x > 1 \end{cases}$$

$$\overset{\text{и.х. } f_n}{\simeq} \begin{cases} x!!, & \text{ако } x \leq 1 \\ x.(x-2)!!, & \text{ако } x > 1 \ \&\ x-2 < 2n \\ \neg!, & \text{ако } x - 2 \geq 2n \end{cases} \simeq \begin{cases} x!!, & \text{ако } x < 2(n+1) \\ \neg!, & \text{ако } x \geq 2(n+1). \end{cases}$$

Индукционната хипотеза се потвърди и за $n+1$. Остана да съобразим, че границата на редицата $\{f_n\}_n$ е функцията $x!!$, което следва съвсем директно от дефиницията за точна горна граница (3.8). □

Апроксимациите на операторите от следващите задачи вече са с по-разнообразни дефиниционни области.

Задача 3.31. С помощта на теоремата на Кнастер-Тарски да се намери най-малката неподвижна точка на оператора

$$\Gamma(f)(x,y) \simeq \begin{cases} 0, & \text{ако } x = y \\ f(x, y+1) + 1, & \text{иначе.} \end{cases}$$

Решение. Тръгвайки от $f_0 = \emptyset^{(2)}$, за f_1 ще имаме:

$$f_1(x,y) \overset{\text{деф}}{\simeq} \begin{cases} 0, & \text{ако } x = y \\ f_0(x, y+1) + 1, & \text{иначе} \end{cases} \simeq \begin{cases} 0, & \text{ако } x = y \\ \neg!, & \text{иначе.} \end{cases}$$

Сега за апроксимацията f_2 получаваме:

$$f_2(x,y) \simeq \Gamma(f_1)(x,y) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 0, & \text{ако } x = y \\ f_1(x, y+1) + 1, & \text{иначе} \end{cases}$$

$$\overset{\text{деф } f_1}{\simeq} \begin{cases} 0, & \text{ако } x = y \\ 0+1, & \text{ако } x+1 = y \\ \neg!, & \text{в останалите случаи} \end{cases} \simeq \begin{cases} x - y, & \text{ако } 0 \leq x - y < 2 \\ \neg!, & \text{в останалите случаи.} \end{cases}$$

Да приемем, че за произволно n, f_n изглежда по подобен начин:

$$f_n(x,y) \simeq \begin{cases} x - y, & \text{ако } 0 \leq x - y < n \\ \neg!, & \text{в останалите случаи.} \end{cases}$$

Базата на индукцията я имаме, така че пристъпваме директно към проверката за f_{n+1}:

$$f_{n+1}(x,y) \simeq \Gamma(f_n)(x,y) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 0, & \text{ако } x = y \\ f_n(x, y+1) + 1, & \text{иначе} \end{cases}$$

$$\overset{\text{и.х. } f_n}{\simeq} \begin{cases} 0, & \text{ако } x = y \\ x - (y+1) + 1, & \text{ако } x \neq y \ \& \ 0 \leq x - (y+1) < n \\ \neg!, & \text{в останалите случаи} \end{cases}$$

$$\simeq \begin{cases} x - y, & \text{ако } 0 \leq x - y < n+1 \\ \neg!, & \text{в останалите случаи,} \end{cases}$$

което потвърждава индуктивното ни предположение. Накрая съобразете, че f_Γ има вида:

$$f_\Gamma(x,y) \simeq \begin{cases} x - y, & \text{ако } x \geq y \\ \neg!, & \text{иначе.} \end{cases}$$

□

Задача 3.32. С теоремата на Кнастер-Тарски да се намери най-малката неподвижна точка на оператора

$$\Gamma(f)(x) \simeq \begin{cases} 1, & \text{ако } x = 0 \\ (f(\frac{x}{2}))^2, & \text{ако } x > 0 \text{ е четно} \\ 2(f(\frac{x-1}{2}))^2, & \text{ако } x \text{ е нечетно} \end{cases}$$

Решение. Ще действаме по схемата от предишната задача: най-напред ще намерим явния вид на всяка от апроксимациите $f_0, f_1, \ldots$, а после ще намерим границата на тази редица, която е точно f_Γ.

По дефиниция $f_0 = \emptyset^{(1)}$, а за f_1 получаваме последователно:

$$f_1(x) \simeq \Gamma(f_0)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ f_0^2(\frac{x}{2}), & \text{ако } x > 0 \text{ е четно} \\ 2f_0^2(\frac{x-1}{2}), & \text{ако } x \text{ е нечетно} \end{cases} \simeq \begin{cases} 1, & \text{ако } x = 0 \\ \neg!, & \text{ако } x > 0. \end{cases}$$

Като имаме предвид явния вид на f_1, за f_2 получаваме:

$$f_2(x) \simeq \Gamma(f_1)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ f_1^2(\frac{x}{2}), & \text{ако } x > 0 \text{ е четно} \\ 2f_1^2(\frac{x-1}{2}), & \text{ако } x \text{ е нечетно} \end{cases} \overset{\text{деф } f_1}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ 2.1^2, & \text{ако } x = 1 \\ \neg!, & \text{ако } x > 1. \end{cases}$$

Тази функция можем да препишем във вида

$$f_2(x) \simeq \begin{cases} 2^x, & \text{ако } x < 2 \\ \neg!, & \text{иначе,} \end{cases}$$

и тя е съвсем същата като функцията f_2 от *Задача* 3.29. Да не се подвеждаме, обаче; следваща апроксимация f_3 вече изглежда по-различно:

$$f_3(x) \simeq \Gamma(f_2)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ f_2^2(\frac{x}{2}), & \text{ако } x > 0 \text{ е четно} \\ 2f_2^2(\frac{x-1}{2}), & \text{ако } x \text{ е нечетно} \end{cases} \overset{\text{деф } f_2}{\simeq} \begin{cases} 2^x, & \text{ако } \mathbf{x < 4} \\ \neg!, & \text{иначе,} \end{cases}$$

Хипотезата ни за $f_n, n \geq 1$, е такава:

$$f_n(x) \simeq \begin{cases} 2^x, & \text{ако } x < 2^{n-1} \\ \neg!, & \text{иначе.} \end{cases}$$

Вече наблюдавахме, че при $n = 1, 2, 3$, f_n имаше този вид. Приемаме, че и за произволно n това е така и пресмятаме внимателно f_{n+1}:

$$f_{n+1}(x) \simeq \Gamma(f_n)(x) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ f_n^2(\frac{x}{2}), & \text{ако } x > 0 \text{ е четно} \\ 2f_n^2(\frac{x-1}{2}), & \text{ако } x \text{ е нечетно} \end{cases}$$

$$\overset{\text{и.х. } f_n}{\simeq} \begin{cases} 1, & \text{ако } x = 0 \\ (2^{\frac{x}{2}})^2, & \text{ако } x > 0 \text{ е четно } \& \ \frac{x}{2} < 2^{n-1} \\ 2(2^{\frac{x-1}{2}})^2, & \text{ако } x \text{ е нечетно } \& \ \frac{x-1}{2} < 2^{n-1} \end{cases}$$

$$\simeq \begin{cases} 1, & \text{ако } x = 0 \\ 2^x, & \text{ако } x > 0 \text{ е четно } \& \ x < 2^n \\ 2^x, & \text{ако } x \text{ е нечетно } \& \ x - 1 < 2^n \end{cases} \simeq \begin{cases} 2^x, & \text{ако } x < 2^n \\ \neg!, & \text{иначе.} \end{cases}$$

За последната еквивалентност използвахме, че при нечетно x имаме:

$$x - 1 < 2^n \implies x \leq 2^n \implies x < 2^n.$$

Сега с разсъждения, съвсем подобни на тези от *Задача* **??** а) показваме, че и тази редица $\{f_n\}_n$ има граница 2^x.

Забележете експоненциалната скорост, с която расте броят на елементите на $Dom(f_n)$. Това, разбира се, е в тясна връзка с логаритмичната сложност на бързия алгоритъм за степенуване, тъй като $Dom(f_n)$ на практика дава тези входове, за които рекурсивната програма, определена от оператора, спира за $\leq n$ рекурсивни обръщения. □

Задача 3.33. С теоремата на Кнастер-Тарски да се намери най-малката неподвижна точка на оператора

$$\Gamma(f)(x,y) \simeq \begin{cases} 0, & \text{ако } x < y \\ f(x-y,y)+1, & \text{ако } x \geq y. \end{cases}$$

Решение. Отново искаме да опишем общия вид на n-тата апроксимация f_n. Имайки предвид, че $f_0 = \emptyset^{(2)}$, за f_1 ще имаме:

$$f_1(x,y) \simeq \Gamma(f_0)(x,y) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 0, & \text{ако } x < y \\ f_0(x-y,y)+1, & \text{ако } x \geq y \end{cases} \simeq \begin{cases} 0, & \text{ако } x < y \\ \neg!, & \text{ако } x \geq y. \end{cases}$$

Както беше и в примерите по-горе, f_1 е дефинирана в точките, които са базови за оператора (в случая това са тези $(x,y) : x < y$). Това все още не може да ни ориентира за общия вид на f_n, затова продължаваме с експериментите:

$$f_2(x,y) \simeq \Gamma(f_1)(x,y) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 0, & \text{ако } x < y \\ f_1(x-y,y)+1, & \text{ако } x \geq y \end{cases}$$

$$\overset{\text{деф } f_1}{\simeq} \begin{cases} 0, & \text{ако } x < y \\ 0+1, & \text{ако } x \geq y \ \& \ x-y < y \\ \neg!, & \text{в останалите случаи} \end{cases} \simeq \begin{cases} \lfloor \frac{x}{y} \rfloor, & \text{ако } y \neq 0 \ \& \ \lfloor \frac{x}{y} \rfloor < 2 \\ \neg!, & \text{в останалите случаи.} \end{cases}$$

За последната еквивалентност използвахме, че

$$y \leq x < 2y \iff 1 \leq \frac{x}{y} < 2 \iff \lfloor \frac{x}{y} \rfloor = 1.$$

Освен това условията $x < y$ и $x - y < y$ ни гарантират, че $y \neq 0$ и следователно частното $\frac{x}{y}$ е дефинирано.

Звучи правдоподобно да предположим, че f_n има следния вид:

$$f_n(x,y) \simeq \begin{cases} \lfloor \frac{x}{y} \rfloor, & \text{ако } y \neq 0 \ \& \ \lfloor \frac{x}{y} \rfloor < n \\ \neg!, & \text{в останалите случаи.} \end{cases}$$

Наистина, да приемем, че това е така, и да видим какво можем да кажем за f_{n+1}:

$$f_{n+1}(x,y) \simeq \Gamma(f_n)(x,y) \overset{\text{деф } \Gamma}{\simeq} \begin{cases} 0, & \text{ако } x < y \\ f_n(x-y,y)+1, & \text{ако } x \geq y \end{cases}$$

$$\overset{\text{и.х. } f_n}{\simeq} \begin{cases} 0, & \text{ако } x < y \\ \lfloor \frac{x-y}{y} \rfloor + 1, & \text{ако } x \geq y \ \& \ y \neq 0 \ \& \ \lfloor \frac{x-y}{y} \rfloor < n \\ \neg!, & \text{в останалите случаи} \end{cases}$$

$$\simeq \begin{cases} \lfloor \frac{x}{y} \rfloor, & \text{ако } y \neq 0 \ \& \ \lfloor \frac{x}{y} \rfloor = 0 \\ \lfloor \frac{x}{y} \rfloor, & \text{ако } y \neq 0 \ \& \ 1 \leq \lfloor \frac{x}{y} \rfloor < n+1 \\ \neg!, & \text{в останалите случаи} \end{cases}$$

$$\simeq \begin{cases} \lfloor \frac{x}{y} \rfloor, & \text{ако } y \neq 0 \ \& \ \lfloor \frac{x}{y} \rfloor < n+1 \\ \neg!, & \text{в останалите случаи,} \end{cases}$$

с което индуктивната ни хипотеза се потвърди. В преобразованията по-горе използвахме наблюдението, че при $x \geq y > 0$:

$$\lfloor \frac{x-y}{y} \rfloor = \lfloor \frac{x}{y} - 1 \rfloor = \lfloor \frac{x}{y} \rfloor - 1.$$

Остана да намерим границата на редицата $f_0, f_1, \ldots$. Нека отново

$$f = \bigcup_n f_n.$$

Да фиксираме произволни x, y, като $y \neq 0$. Тогава $\lfloor \frac{x}{y} \rfloor$ е определено. Да изберем n така, че $\lfloor \frac{x}{y} \rfloor < n$ (бихме могли направо да вземем $n := \lfloor \frac{x}{y} \rfloor + 1$). Тогава точката (x,y) принадлежи на $Dom(f_n)$ и $f_n(x,y) = \lfloor \frac{x}{y} \rfloor$. Сега от дефиницията на точна горна граница (3.8) ще имаме, че и $f(x,y) = \lfloor \frac{x}{y} \rfloor$.

Ако $y = 0$, то каквото и да е x, от общия вид на f_n виждаме, че $f_n(x,0)$ не е дефинирано, като това е за всяко n. Тогава е ясно, че и граничната функция f няма да е дефинирана в $(x,0)$: ако допуснем, че съществува z,

такова че $f(x,0) \simeq z$, това би означавало, съгласно (3.8), че непременно за някое n и $f_n(x,0) \simeq z$ — противоречие.

Финално, за $f_\Gamma = \bigcup_n f_n$ получаваме:

$$f_\Gamma(x,y) \simeq \begin{cases} \lfloor \frac{x}{y} \rfloor, & \text{ако } y > 0 \\ \neg!, & \text{ако } y = 0. \end{cases}$$

□

Задачи за ЕК:

Задача 1. Дадени са компактните оператори

$$\Gamma : \mathcal{F}_k \times \mathcal{F}_m \longrightarrow \mathcal{F}_k \quad \text{и} \quad \Delta : \mathcal{F}_m \longrightarrow \mathcal{F}_m.$$

Докажете, че най-малкото решение на системата

$$\left| \begin{array}{lcl} f & = & \Gamma(f,\ g) \\ g & = & \Delta(g). \end{array} \right.$$

е двойката (f^*, g^*), където g^* е най-малката неподвижна точка на Δ, а f^* е най-малката неподвижна точка на оператора $\lambda f.\Gamma(f,\ g^*)$.

Задача 2. Нека h е фиксирана двуместна функция. Намерете най-малките неподвижни точки на операторите:

а)

$$\Gamma(f)(x,y) \simeq \begin{cases} 0, & \text{ако } h(x,y) \simeq 0 \\ f(x,y+1)+1, & \text{ако } h(x,y) > 0 \\ \neg!, & \text{ако } \neg! h(x,y) \end{cases}$$

б)

$$\Gamma(f)(x,y) \simeq \begin{cases} y, & \text{ако } h(x,y) \simeq 0 \\ f(x,y+1)1, & \text{ако } h(x,y) > 0 \\ \neg!, & \text{ако } \neg! h(x,y). \end{cases}$$

Вярно ли е, че тези оператори имат и други неподвижни точки? Обосновете се.

Задача 3. Опишете всички неподвижни точки на оператора Γ, дефиниран по следния начин:

$$\Gamma(f)(x,y) \simeq \begin{cases} x+y, & \text{ако } x = 0 \text{ или } y = 0 \\ f(x-1, f(x,y-1)), & \text{иначе.} \end{cases}$$

Задача 4. Докажете, че е тотална функция най-малката неподвижна точка на оператора

$$\Gamma(f)(x) \simeq \begin{cases} \frac{x}{2}, & \text{ако } x \text{ е четно} \\ f(f(3x+1)), & \text{иначе.} \end{cases}$$

3.6.4 Рекурсивни оператори. Първа теорема за рекурсия

Определение 3.7. Операторът $\Gamma\colon \mathcal{F}_k \longrightarrow \mathcal{F}_m$ наричаме *рекурсивен*, ако той е компактен и ефективен.

Теорема 3.8. (Първа теорема за рекурсия) Нека $\Gamma : \mathcal{F}_k \longrightarrow \mathcal{F}_k$ е рекурсивен оператор. Тогава Γ притежава *изчислима* най-малка неподвижна точка f_Γ, която се дефинира по следния начин:

$$f_\Gamma \ = \bigcup_n \Gamma^n(\emptyset^{(k)}).$$

Доказателство. Операторът Γ е рекурсивен и в частност — компактен. От теоремата на Кнастер-Тарски следва, че най-малката му неподвижна точка f_Γ съществува и има горното представяне. Това, което ни дава условието за ефективност на оператора, е изчислимостта на f_Γ. Да я съобразим.

За целта нека отново $f_n \overset{\text{деф}}{=} \Gamma^n(\emptyset^{(k)})$. Най-напред да се убедим, че съществува рекурсивна функция g, която "държи" индексите на функциите от редицата $\{f_n\}_n$, т.е. g е такава, че за всяко n

$$f_n = \varphi^{(k)}_{g(n)}.$$

Наистина, да фиксираме h — някаква рекурсивна индексна функция на Γ и a_0 — произволен индекс на $\emptyset^{(k)}$. Дефинираме функцията g с примитивна рекурсия както следва:

$$\left|\begin{array}{l} g(0) = a_0 \\ g(n+1) = h(g(n)). \end{array}\right.$$

Непосредствена индукция по n ни убеждава, че $f_n = \varphi^{(k)}_{g(n)}$ за всяко естествено n. Наистина, при $n = 0$ имаме, съгласно избора на a_0:

$$f_0 \overset{\text{деф}}{=} \emptyset^{(k)} = \varphi^{(k)}_{a_0} \overset{\text{деф } g}{=} \varphi^{(k)}_{g(0)}.$$

Сега ако приемем, че за някое n, $f_n = \varphi^{(k)}_{g(n)}$, то за $n+1$ ще имаме:

$$f_{n+1} \overset{\text{деф}}{=} \Gamma(f_n) \overset{\text{и.х.}}{=} \Gamma(\varphi^{(k)}_{g(n)}) = \varphi^{(k)}_{h(g(n))} \overset{\text{деф } g}{=} \varphi^{(k)}_{g(n+1)}.$$

Сега вече можем да твърдим, че

$$f_\Gamma \ = \bigcup_n \varphi^{(k)}_{g(n)}.$$

Тогава според *Твърдение* 3.8 ще имаме, че за всяко $\bar{x}$ и y е в сила еквивалентността

$$f_\Gamma(\bar{x}) \simeq y \iff \exists n \ \varphi^{(k)}_{g(n)}(\bar{x}) \simeq y. \tag{3.9}$$

Да си спомним, че съгласно теоремата за нормален вид на Клини, всяка функция $\varphi_a^{(k)}$ има следния вид:

$$\varphi_a^{(k)}(\bar{x}) \simeq L(\mu z[T_k(a, \bar{x}, z) = 0]),$$

където T_k е предикатът на Клини (за който видяхме, че е примитивно рекурсивен).

За нашите цели ще е по-удобно да модифицираме леко този предикат, така че той вече да притежава свойството: за всяко a и $\bar{x} \in \mathbb{N}^k$:

$$\exists z\ T_k(a, \bar{x}, z) = 0 \implies \exists! z\ T_k(a, \bar{x}, z) = 0.$$

Това става, като вместо T_k разглеждаме предиката T_k^*, дефиниран като

$$T_k^*(a, \bar{x}, z) = \begin{cases} T_k(a, \bar{x}, z), & \text{ако } t < z\ T_k(a, \bar{x}, z) > 0 \\ 1, & \text{в останалите случаи.} \end{cases}$$

Тогава очевидно $\mu z[T_k(a, \bar{x}, z) = 0] \simeq \mu z[T_k^*(a, \bar{x}, z) = 0]$ и значи

$$\varphi_a^{(k)}(\bar{x}) \simeq L(\mu z[T_k^*(a, \bar{x}, z) = 0]).$$

Оттук, като използваме представянето (3.9) на f_Γ, получаваме, че

$$f_\Gamma(\bar{x}) \simeq y \iff \exists n\ L(\mu z[T_k^*(g(n), \bar{x}, z) = 0]) \simeq y. \tag{3.10}$$

Като имаме предвид избора на T_k^*, условието вдясно можем да запишем и без минимизация. По-точно, твърдим, че

$$f_\Gamma(\bar{x}) \simeq y \iff \exists n \exists z\ T_k^*(g(n), \bar{x}, z) = 0\ \&\ L(z) = y. \tag{3.11}$$

Наистина, ако $f_\Gamma(\bar{x}) \simeq y$, то за някое n, $L(\mu z[T_k^*(g(n), \bar{x}, z) = 0]) \simeq y$ и значи $\exists n \exists z\ T_k^*(g(n), \bar{x}, z) = 0\ \&\ L(z) = y$.

Обратно, да приемем, че за някои n и z: $T_k^*(g(n), \bar{x}, z) = 0\ \&\ L(z) = y$. Понеже числото z с това свойство е единствено, то

$$\mu v[T_k^*(g(n), \bar{x}, v) = 0] = z.$$

Тогава, разбира се и $L(\mu v[T_k^*(g(n), \bar{x}, v) = 0]) = L(z)$. Но $L(z) = y$ и следователно

$$L(\mu v[T_k^*(g(n), \bar{x}, v) = 0]) = y.$$

Оттук, използвайки еквивалентността (3.10), можем да твърдим, че $f_\Gamma(\bar{x}) \simeq y$, с което приключва проверката на (3.11).

Сега вече можем да пристъпим към доказателството на изчислимостта на f_Γ. Ще покажем, че за нея имаме следното представяне:

$$f_\Gamma(\bar{x}) \simeq \underbrace{L(R(\mu t[T_k^*(g(L(t)), \bar{x}, R(t)) = 0]))}_{F(\bar{x})},$$

откъдето, разбира се, ще следва, че f_Γ е изчислима.

Да означим функцията вдясно с F, както е показано по-горе. Задачата ни е да покажем, че $F = f_\Gamma$. Да видим най-напред, че $F \subseteq f$. За целта да приемем, че $F(\bar{x}) \simeq y$ за някои $\bar{x}$ и y. Трябва да покажем, че и $f_\Gamma(\bar{x}) \simeq y$. От $F(\bar{x}) \simeq y$ следва, че за най-малкото t, такова че $T_k^*(g(L(t)), \bar{x}, R(t)) = 0$ ще имаме $L(R(t)) = y$. Нека $n := L(t), z := R(t)$. Тогава за тези n и z имаме, че $T_k^*(g(n), \bar{x}, z) = 0$ и $L(z) \overset{\text{деф}}{=} L(R(t)) = y$. Сега еквивалентността (3.11) ни дава $f_\Gamma(\bar{x}) \simeq y$.

За обратното включване $f_\Gamma \subseteq F$ ще се възползваме от *Задача* 1.1, според която е достатъчно да покажем по-слабото условие $Dom(f_\Gamma) \subseteq Dom(F)$. Наистина, нека да вземем произволно $\bar{x} \in Dom(f_\Gamma)$. Отново прилагаме еквивалентността (3.11), само че в обратна посока: щом $!f_\Gamma(\bar{x})$, то за някои n и z ще е вярно, че $T_k^*(g(n), \bar{x}, z) = 0$. Сега вземаме $t := \Pi(n, z)$. Ясно е, че за това t ще имаме $T_k^*(g(L(t)), \bar{x}, R(t)) = 0$ и следователно $F(\bar{x})$ е дефинирана. □

Задача 3.34. Като използвате първата теорема за рекурсия докажете, че функцията на Акерман е рекурсивна.

Решение. В доказателството на *Задача* 3.24 видяхме, че е ефективен операторът Γ, свързан с функцията на Акерман

$$\Gamma(f)(x,y) \simeq \begin{cases} y+1, & \text{ако } x = 0 \\ f(x-1,0), & \text{ако } x > 0 \ \& \ y = 0 \\ f(x-1, f(x, y-1)), & \text{ако } x > 0 \ \& \ y > 0. \end{cases}$$

От *Задача* 3.25 г) знаем, че този оператор е компактен. Значи общо той е рекурсивен. Знаем още, че Γ има единствена неподвижна точка — функцията на Акерман. Следователно най-малката неподвижна точка f_Γ е функцията на Акерман и тогава съгласно първата теорема за рекурсия тази функция е изчислима, което в случая значи и рекурсивна. □

Предимството на първата теорема за рекурсия е, че ни казва *коя точно* е изчислимата функция, която е неподвижна точка на рекурсивния оператор, като при това ни дава начин да я конструираме. За разлика от нея, *Твърдение* 3.6 (което е следствие от втората теорема за рекурсия) само твърди, че всеки ефективен оператор има поне една изчислима неподвижна точка. От друга страна, предимството на втората теорема за рекурсия (и на свързаната с нея теорема за определимост по рекурсия) е в това, че тя може да се прилага в рекурсивни дефиниции, в които участва и *програмата* на функцията, която се определя по рекурсия (като например самовъзпроизвеждащата се програма от *Пример* 3.2).

3.7 Програмна характеризация на примитивно рекурсивните функции

Ще завършим тази глава с описанието на един учебен език, на който се програмират всички едноместни примитивно рекурсивни функции (и само те). Ще построим транслатор от този език към езика на МНР, както и интерпретатор, който ще се явява универсална функция за едноместните примитивно рекурсивни функции. Основната ни цел ще бъде да наблюдаваме как се прилагат най-важните теореми, които доказахме дотук.

3.7.1 Езикът SL

Езикът SL, който ще въведем, е предложен от проф. Димитър Скордев. Той е съвсем прост функционален език. Програмите на този език ще наричаме *SL-изрази* (или *SL-програми*).

Понятието *SL-израз* определяме със следната индуктивна дефиниция:

Определение 3.8.

1) SL-изрази са $\mathbb{S}$, $\mathbb{O}$, $\mathbb{I}$, $\mathbb{L}$ и $\mathbb{R}$ (ще ги наричаме *базисни*).

2) Ако E_1 и E_2 са SL-изрази, то и $(E_1 \circ E_2)$ е SL-израз.

3) Ако E_1 и E_2 са SL-изрази, то и $\pi(E_1, E_2)$ е SL-израз.

4) Ако E е SL-израз, то и $\rho(E)$ е SL-израз.

SL-изразите можем да зададем по-кратко чрез следната граматика:

$$E \ ::= \ \mathbb{S} \ | \ \mathbb{O} \ | \ \mathbb{I} \ | \ \mathbb{L} \ | \ \mathbb{R} \ | \ (E \circ E) \ | \ \pi(E, E) \ | \ \rho(E)$$

Примери за SL-изрази: $\mathbb{S}$, $(\mathbb{S} \circ \mathbb{O})$, $\pi(\rho((\mathbb{S} \circ \mathbb{O})), \mathbb{L})$.

SL-изразите са синтактични обекти. Но какво се крие зад всеки SL-израз? За да определим смисъла (семантиката) на всеки такъв израз, най-напред ще дефинираме две нови операции над едноместни функции. Първата се нарича *свечетаване* на две функции; ще я означаваме с Π. За всяка $f, g \in \mathcal{F}_1$ полагаме $\Pi(f, g)$ да е функцията, дефинирана като:

$$\Pi(f, g)(x) \stackrel{\text{деф}}{=} \Pi(f(x), g(x)),$$

където $\Pi(x, y) = 2^x(2y + 1) - 1$ е кодирането на наредени двойки от числа, което въведохме в раздел 1.8.1. Да напомним, че декодиращите функции на това кодиране означавахме с L и R.

Забележка. Правете разлика между записите $\mathsf{\Pi}(f,g)$ и $\Pi(f,g)$. Първият е функционалната операция $\mathsf{\Pi}$ (съчетаване), приложена върху аргументите f и g, докато вторият е суперпозицията на функциите Π, f и g.

Да отбележим, че операцията $\mathsf{\Pi}$ действа като *кодиране на функции*. Ако f и g са *тотални*, функцията $h = \mathsf{\Pi}(f,g)$ "помни" f и g, т.е. ако знаем h, можем да възстановим f и g. Това е така, защото очевидно

$$f(x) = L(\Pi(f(x), g(x))) \quad \text{и} \quad g(x) = R(\Pi(f(x), g(x))),$$

откъдето веднага $f = L \circ h$ и $g = R \circ h$.

Втората функционална операция, която ще наричаме *ротация* и ще означаваме с R, всъщност не е съвсем нова за нас. Тя е на практика операцията итерация, която въведохме в раздел 1.7.4, но дефинирана така, че резултатът да бъде *едноместна* функция. Ето точното определение:
За всяка $f \in \mathcal{F}_1$ дефинираме нейната ротация $\mathsf{R}(f)$ както следва:

$$\mathsf{R}(f)(x) \overset{\text{деф}}{\simeq} f^{L(x)}(R(x)).$$

Когато прилагаме $\mathsf{R}(f)$ към аргументи от вида $\Pi(n,x)$, получаваме точно итерацията f^* на f, защото $\mathsf{R}(f)(\Pi(n,x)) \simeq f^n(x) \simeq f^*(n,x)$.

Сега вече можем да дефинираме семантиката на един SL-израз, т.е. функцията, която той определя. Разбира се, дефиницията ще е по индукция, следваща индуктивната дефиниция 3.8 на SL-израз:

1) Базисните изрази $\mathbb{S}$, $\mathbb{O}$, $\mathbb{I}$, $\mathbb{L}$ и $\mathbb{R}$ определят съответно функциите $\mathcal{S}, \mathcal{O}$, I, L и R, където $\mathcal{S}$ е $\lambda x.x+1$, $\mathcal{O}$ е $\lambda x.0$, I е $\lambda x.x$ (идентитетът), а L и R са декодиращите функции на кодирането Π.

2) Ако SL-изразите E_1 и E_2 определят функциите f и g, то SL-изразът $(E_1 \circ E_2)$ определя тяхната композиция $f \circ g$.

3) Ако SL-изразите E_1 и E_2 определят функциите f и g, то SL-изразът $\pi(E_1, E_2)$ определя тяхното съчетаване $\mathsf{\Pi}(f,g)$.

4) Ако SL-изразът E определя функцията f, то SL-изразът $\rho(E)$ определя ротацията $\mathsf{R}(f)$ на f.

Скобите в изразите от вида $(E_1 \circ E_2)$ са заради еднозначния синтактичен анализ — за да е ясно как се чете, примерно, изразът $E_1 \circ E_2 \circ E_3$ — дали той е $((E_1 \circ E_2) \circ E_3)$ или $(E_1 \circ (E_2 \circ E_3))$. Но тъй като композицията е асоциативна операция, за семантиката това не е от значение. Затова по-надолу обикновено ще пропускаме тези скоби.

Правете разлика между *израз* и *функция*, която този израз определя. Например изразите $\mathbb{S}$ и $\mathbb{S} \circ \mathbb{I}$ са различни, но имат една и съща семантика, т.е. определят една и съща функция $\lambda x.x+1$. За такива изрази ще казваме, че са *еквивалентни*.

Определение 3.9. Функцията $f \in \mathcal{F}_1$ наричаме *SL-определима*, ако тя се определя от някой SL-израз.

Съвкупността от тези функции ще означаваме с $\mathcal{SL}$, т.е.

$$\mathcal{SL} \;=\; \{f \mid f \text{ е } SL\text{-определима}\}.$$

Функциите от класа $\mathcal{SL}$ са точно тези, които се получават от базисните функции $\mathcal{S}, \mathcal{O}, I, L$ и R чрез краен брой прилагания на функционалните операции $\circ$, $\mathsf{\Pi}$ и R — убедете се самостоятелно, че това е така.

Ясно е, че ако f е SL-определима, има безброй много изрази, които я определят. Наистина, ако f се определя от израза E, то и всеки от изразите $E \circ \underbrace{\mathbb{I} \circ \cdots \circ \mathbb{I}}_{n \text{ пъти}}$ определя f (като, разбира се, това далеч не са всички изрази, еквивалентни на E.).

Нашата близка цел ще бъде е да покажем, че функциите, определими в езика SL, са точно едноместните примитивно рекурсивни функции.

Теорема 3.9. (Скордев) Едноместната функция f е SL-определима тогава и само тогава, когато тя е примитивно рекурсивна.

Доказателство. Нека f се определя от SL-израза E. С индукция по дефиницията на E ще покажем, че f е примитивно рекурсивна.

Наистина, ако E е базисен SL-израз, то той определя някоя от функциите $\mathcal{S}, \mathcal{O}, I, L$ и R, които са примитивно рекурсивни. Нека E е от вида $E_1 \circ E_2$, като за E_1 и E_2 е вярно, че функциите f и g, които те определят, са примитивно рекурсивни. Но $E_1 \circ E_2$ определя $f \circ g$, която очевидно също е примитивно рекурсивна. Ако $E = \pi(E_1, E_2)$ и E_1 и E_2 определят примитивно рекурсивните f и g, то $\pi(E_1, E_2)$ определя функцията $\mathsf{\Pi}(f, g) \overset{\text{деф}}{=} \Pi(f, g)$, която също ще е примитивно рекурсивна. Накрая, ако $E = \rho(E_1)$, като по индуктивната хипотеза E_1 определя примитивно рекурсивната функция f, то $\rho(E_1)$ определя функцията $\lambda x.f^*(L(x), R(x))$, която по *Твърдение* 1.17 е примитивно рекурсивна.

За обратната посока на твърдението не можем да разсъждаваме с индукция по дефиницията на класа на примитивно рекурсивните функции. Проблемът е, че с тази дефиниция се въвеждат *всички* примитивно рекурсивни функции, а на нас ни трябват само *едноместните*. Този технически проблем ще преодолеем, като се възползваме от понятието *представяща* на дадена функция f.

Да напомним, че ако f е произволна функция на n аргумента, *представяща* на f е едноместната функция $\hat{f} \in \mathcal{F}_1$, която действа върху *кодовете* на аргументите на f така, както f действа върху самите аргументи, с други думи

$$\hat{f}(\Pi_n(x_1, \ldots, x_n)) \simeq f(x_1, \ldots, x_n).$$

"Официалната" дефиниция на $\hat{f}$ е следната: за всяко $z \in \mathbb{N}$:

$$\hat{f}(z) \overset{\text{деф}}{\simeq} f(J_1^n(z), \ldots, J_n^n(z)).$$

Тук $J_1^n, \ldots, J_n^n$ са декодиращите функции за кодирането Π_n от раздел 1.8.2. При $n = 1$ очевидно $\hat{f} = f$.

Нашата цел ще е да покажем, че ако f е примитивно рекурсивна, то нейната представяща $\hat{f}$ е SL-определима. В частност, когато f е едноместна, от $\hat{f} = f$ ще получим, че f е SL-определима.

Оттук до края на доказателството на теоремата целта ни ще бъде да покажем на импликацията

$$f \text{ е примитивно рекурсивна } \implies \hat{f} \in \mathcal{SL}. \tag{3.12}$$

Това ще направим с индукция по дефиницията на класа на примитивно рекурсивните функции. Ако f е $\mathcal{S}$ или $\mathcal{O}$, няма какво да се доказва. Ако f е проектиращата функция I_k^n, то нейната представяща е функцията J_k^n. От *Твърдение* 1.21 знаем, че за J_k^n имаме следното представяне:

$$J_k^n = \begin{cases} R \circ L^{n-k}, & \text{ако } 1 < k \leq n \\ L^{n-1}, & \text{ако } k = 1. \end{cases}$$

То показва, че и в двата случая $k = 1$ и $1 < k \leq n$ — ще имаме, че J_k^n ще принадлежи на $\mathcal{SL}$.

Сега нека за $f \in \mathcal{F}_n$ е изпълнено $f = g(h_1, \ldots, h_k)$, като по индукционната хипотеза $\hat{g}, \hat{h}_1, \ldots, \hat{h}_k$ лежат в $\mathcal{SL}$. Трябва да покажем, че и $\hat{f} \in \mathcal{SL}$. Да разпишем дефиницията на $\hat{f}$:

$$\begin{aligned} \hat{f}(\Pi_n(\bar{x})) \overset{\text{деф}}{=} f(\bar{x}) \;&=\; g(h_1(\bar{x}), \ldots, h_k(\bar{x})) \;=\; \hat{g}(\Pi_k(h_1(\bar{x}), \ldots, h_k(\bar{x}))) \\ &= \hat{g}(\Pi_k(\hat{h}_1(\Pi_n(\bar{x})), \ldots, \hat{h}_k(\Pi_n(\bar{x})))) \;=\; \hat{g}(\Pi_k(\hat{h}_1, \ldots, \hat{h}_k))(\Pi_n(\bar{x})). \end{aligned}$$

Следователно $\hat{f} = \hat{g}(\Pi_k(\hat{h}_1, \ldots, \hat{h}_k))$. Съгласно индуктивната хипотеза $\hat{g}, \hat{h}_1, \ldots, \hat{h}_k$ са от $\mathcal{SL}$. Сега за да се убедим, че и $\hat{f} \in \mathcal{SL}$, е достатъчно да си спомним дефиницията на кодирането Π_k от раздел 1.8.2. Имаме

$$\Pi_k(\hat{h}_1, \ldots, \hat{h}_k) \overset{\text{деф}}{=} \underbrace{\Pi(\ldots\Pi}_{k-1 \text{ пъти}}(\hat{h}_1, \hat{h}_2), \ldots, \hat{h}_k) = \underbrace{\mathsf{\Pi}(\ldots\mathsf{\Pi}}_{k-1 \text{ пъти}}(\hat{h}_1, \hat{h}_2), \ldots, \hat{h}_k).$$

От това представяне се вижда, че функцията $\Pi_k(\hat{h}_1, \ldots, \hat{h}_k)$ се получава след $k-1$-кратно прилагане на операцията $\mathsf{\Pi}$ към функции, които са в класа $\mathcal{SL}$, следователно и тя е в $\mathcal{SL}$, откъдето веднага и $\hat{f} \in \mathcal{SL}$.

Нека сега f се получава с примитивна рекурсия от функциите g и h, като отново по индуктивната хипотеза $\hat{g}$ и $\hat{h}$ са в класа $\mathcal{SL}$. Трябва да

покажем, че и $\hat{f} \in \mathcal{SL}$. Да разгледаме най-напред случая, когато f е на 2 аргумента. Тогава тя удовлетворява примитивно рекурсивната схема

$$\begin{aligned} f(x,0) &= g(x) \\ f(x,y+1) &= h(x,y,f(x,y)). \end{aligned} \tag{3.13}$$

Ще покажем, че примитивната рекурсия може да се изрази чрез операцията R. За целта да разгледаме изображението

$$(x,y,z) \;\mapsto\; (x,\; y+1,\; h(x,y,z)).$$

При $y=0$ и $z=g(x)$ ще имаме

$$(x,\,0,\,g(x)) \;\mapsto\; (x,\,1,\,h(x,0,\underbrace{g(x)}_{f(x,0)})), \quad \text{т.е.} \quad (x,0,g(x)) \;\mapsto\; (x,1,f(x,1)).$$

Аналогично

$$(x,1,f(x,1)) \;\mapsto\; (x,2,\underbrace{h(x,1,f(x,1))}_{f(x,2)}), \text{ т.е. } (x,1,f(x,1)) \;\mapsto\; (x,2,f(x,2)).$$

Тези експерименти ни дават основание да предположим, че тръгвайки от $(x,0,g(x))$, след y итерации ще стигнем до наредената тройка $(x,\; y,\; f(x,y))$, от която ще можем да извлечем $f(x,y)$. За да направим нещата строги, ще трябва да разглеждаме горното преобразование не върху тройки от естествени числа, а върху техните *кодове*.

Да дефинираме функцията H по следния начин:

$$H(\Pi_3(x,y,z)) \overset{\text{деф}}{=} \Pi_3(x,y+1,h(x,y,z)). \tag{3.14}$$

С индукция по y ще покажем, че

$$H^y(\Pi_3(x,0,g(x))) \;=\; \Pi_3(x,y,f(x,y)). \tag{3.15}$$

Наистина, при $y=0$ имаме, че

$$H^0(\Pi_3(x,0,g(x))) \overset{\text{деф}}{=} \Pi_3(x,0,g(x)) \overset{(3.13)}{=} \Pi_3(x,0,f(x,0)).$$

Ако приемем, че за някое y е вярно (3.15), то за $y+1$ получаваме:

$$\begin{aligned} H^{y+1}(\Pi_3(x,0,g(x))) \;&=\; H(H^y(\Pi_3(x,0,g(x)))) \overset{\text{и.х.}}{=} H(\Pi_3(x,y,f(x,y))) \\ &\overset{\text{деф}}{=} \Pi_3(x,y{+}1,h(x,y,f(x,y))) \overset{(3.13)}{=} \Pi_3(x,y{+}1,f(x,y{+}1)). \end{aligned}$$

Така проверихме верността на равенството (3.15). От него получаваме следното изразяване за f:

$$f(x,y) \;=\; J_3^3(H^y(\Pi_3(x,0,g(x)))).$$

Виждаме, че примитивната рекурсия вече е заместена с итерация. Остана тази итерация да изразим чрез операцията R и освен това да видим, че функцията H е в класа $\mathcal{SL}$. Наистина, гледайки дефиницията (3.14), за H можем да запишем:

$$H(z) = \Pi_3(J_1^3(z), J_2^3(z)+1, h(J_1^3(z), J_2^3(z), J_3^3(z))) = \Pi_3(J_1^3(z), J_2^3(z)+1, \hat{h}(z))$$

$$= \Pi(\Pi(J_1^3(z), \mathcal{S}(J_2^3(z))), \hat{h}(z)) = \mathsf{\Pi}(\mathsf{\Pi}(J_1^3, \mathcal{S} \circ J_2^3), \hat{h})(z).$$

Следователно $H = \mathsf{\Pi}(\mathsf{\Pi}(J_1^3, \mathcal{S} \circ J_2^3), \hat{h}) \in \mathcal{SL}$, тъй като по индукционната хипотеза $\hat{h} \in \mathcal{SL}$, а и всяка от функциите J_1^3, J_2^3 и $\mathcal{S}$ също са в този клас. Финално, за представящата $\hat{f}$ на f ще имаме:

$$\begin{aligned}
\hat{f}(z) &= f(L(z), R(z)) = J_3^3(H^{R(z)}(\Pi_3(L(z), 0, g(L(z))))) \\
&= J_3^3(\mathsf{R}(H)(\Pi(R(z), \Pi_3(L(z), \mathcal{O}(z), g(L(z)))))) \\
&= J_3^3(\mathsf{R}(H)(\mathsf{\Pi}(R, \mathsf{\Pi}(\mathsf{\Pi}(L, \mathcal{O}), g \circ L))))).
\end{aligned}$$

Сега вече уверено можем да твърдим, че $\hat{f} \in \mathcal{SL}$.

По подобен начин разсъждаваме и в случая, когато f е на $n = 1$ или на $n \geq 3$ променливи, което оставяме като добро упражнение за читателя ☺. □

3.7.2 Кодиране на SL-изразите

Имаме 4 типа SL-изрази, затова решаваме да ги кодираме според остатъците им при деление на 4, по-точно — изразите от първия вид да имат остатък 0 при деление на 4, от втория вид — остатък 1 и т.н.

С индукция по дефиницията 3.8 на SL-израз полагаме:

1) $\alpha(\mathbb{S}) = 0,\ \alpha(\mathbb{O}) = 4,\ \alpha(\mathbb{I}) = 8,\ \alpha(\mathbb{L}) = 12,\ \alpha(\mathbb{R}) = 4k,\ k \geq 4$;

2) $\alpha(E_1 \circ E_2) = 4\Pi(\alpha(E_1), \alpha(E_2)) + 1$;

3) $\alpha(\pi(E_1, E_2)) = 4\Pi(\alpha(E_1), \alpha(E_2)) + 2$;

4) $\alpha(\rho(E)) = 4\alpha(E) + 3$.

Числото $\alpha(E)$ ще наричаме *код на* SL-израза E. От определението се вижда, че базисният SL-израз $\mathbb{R}$ има безброй много кодове, откъдето следва, че всички изрази, в които той участва, също ще имат безброй много кодове. Това, че изображението α е многозначно, не създава проблем, защото нас ще ни интересува по-скоро обратното изображение α^{-1}, чрез което ще номерираме всички SL-изрази, полагайки $E_a = \alpha^{-1}(a)$ (съвсем по аналогия с $P_a = \gamma^{-1}(a)$).

За да можем да направим това, обаче, трябва да сме сигурни, че на всяко естествено число a съответства точно един SL-израз. Да се убедим:

Твърдение 3.9. Всяко естествено число a е код на точно един SL-израз.

Доказателство. Ще разсъждаваме с пълна индукция по a. Базата на индукцията са числата, кратни на 4. За тях е ясно, че кодират по точно един (базисен) SL-израз.

Нека $a = 4k + 1$. В края на раздел 1.8.1 съобразихме, че за всяко естествено k,$L(k) \leq k$ и $R(k) \leq k$. Тогава със сигурност $L(k) < a$ и $R(k) < a$ и по индуктивната хипотеза ще съществуват единствени SL-изрази E_1 и E_2, такива че $\alpha(E_1) = L(k)$ и $\alpha(E_2) = R(k)$. В такъв случай

$$\alpha(E_1 \circ E_2) \;=\; 4\Pi(\alpha(E_1), \alpha(E_2)) + 1 \;=\; 4\Pi(L(k), R(k)) + 1 \;=\; 4k + 1 = a$$

и очевидно $E_1 \circ E_2$ е единственият SL-израз с това свойство. По същия начин разсъждаваме и когато $a = 4k + 2$. Ако $a = 4k + 3$, имаме $k < a$ и по индуктивната хипотеза ще има единствен SL-израз E, чийто код е k. Така за кода на израза $\rho(E)$ получаваме:

$$\alpha(\rho(E)) \;=\; 4\alpha(E) + 3 \;=\; 4k + 3 = a.$$

□

Сега вече, като сме сигурни, че на всяко $a \in \mathbb{N}$ съответства точно един SL-израз, можем да положим

$$E_a \stackrel{\text{деф}}{=} SL\text{-израза с код } a.$$

Нека още

$$f_a \stackrel{\text{деф}}{=} \text{функцията, която се определя от } SL\text{-израза } E_a.$$

Тогава очевидно редицата

$$E_0, E_1, \ldots, E_a, \ldots$$

изброява всички SL-изрази, а редицата

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

изброява всички SL-определими функции, или все едно — всички едноместни примитивно рекурсивни функции.

3.7.3 Построяване на компилатор за езика SL

Искаме да конструираме *изчислимо* изображение $\kappa : SL \longrightarrow$ МНР, което транслира всеки SL-израз в еквивалентна на него програма за МНР. За да твърдим строго, че κ е изчислима, тя трябва да е *числова* функция, с други думи — да преработва *кодовете* на SL-изразите в съответните *кодове* на програми за МНР.

Искаме SL-изразът E_a и преводът му $P_{\kappa(a)}$ да бъдат еквивалентни, което ще рече — да пресмятат една и съща функция. Формално това означава да е изпълнено равенството

$$f_a \;=\; \varphi_{\kappa(a)}. \tag{3.16}$$

Нашата идея е да дефинираме κ така, че тя да удовлетворява горното условие върху кодовете на *базисните* SL-изрази, а после да разширим дефиницията ѝ върху останалите кодове, като се грижим, разбира се, да поддържаме условието (3.16).

Върху базисните SL-изрази можем да дефинираме κ по най-различни начини. Ако това е истински компилатор, желателно е да превеждаме във възможно най-простите програми. Например изразът $\mathbb{S}$ да се транслира в програмата с единствен оператор $X_1 := X_1 + 1$, изразът $\mathbb{O}$ — в програмата $X_1 := 0$ и прочее.

В случая не се интересуваме от ефективност, затова просто фиксираме някакви МНР програми $P_{a_0}, \ldots, P_{a_4}$, които пресмятат функциите $\mathcal{S}, \mathcal{O}, I, L$ и R, съответно. Да положим

$$\kappa(0) = a_0,\ \kappa(4) = a_1,\ \kappa(8) = a_2,\ \kappa(12) = a_3 \text{ и } \kappa(4k) = a_4 \text{ за всяко } k \geq 4.$$

Нека $a = 4k + 1$. Тогава $E_a = E_{L(k)} \circ E_{R(k)}$ и съответно $f_a = f_{L(k)} \circ f_{R(k)}$. Да приемем за момент (в доказателството на следващото *Твърдение* 3.10 това ще бъде оформено строго като индуктивна хипотеза), че за изразите $E_{L(k)}$ и $E_{R(k)}$, чийто кодове $L(k)$ и $R(k)$ са *по-малки* от k, е в сила условието (3.16). Искаме да дефинираме κ за $a = 4k + 1$ така, че условието (3.16) да продължава да е в сила.

При $a = 4k + 1$ знаем, че $\kappa(a)$ е код на SL-израз, който е композиция на два други. И тук решаваща роля има *ефективността* на операцията (или оператора) композиция, която проверихме в *Задача* 3.15. Нещо повече, там показахме, че съществува и *примитивно рекурсивна* индексна функция, която нарекохме *comp*. Тази функция беше такава, че за всички естествени a и b:

$$\varphi_a \circ \varphi_b = \varphi_{comp(a,b)}.$$

Както вече казахме, ще предполагаме, че върху $L(k)$ и $R(k)$ компилаторът κ се държи коректно, т.е. $f_{L(k)} = \varphi_{\kappa(L(k))}$ и $f_{R(k)} = \varphi_{\kappa(R(k))}$. Тогава за f_a можем да запишем:

$$f_a = f_{L(k)} \circ f_{R(k)} \overset{\text{и.х.}}{=} \varphi_{\kappa(L(k))} \circ \varphi_{\kappa(R(k))} = \varphi_{comp(\kappa(L(k)),\kappa(R(k)))}.$$

Получихме, че $f_a = \varphi_{comp(\kappa(L(k)),\kappa(R(k)))}$, което ни навежда на мисълта да дефинираме κ за $a = 4k+1$ така:

$$\kappa(4k+1) \overset{\text{деф}}{=} comp(\kappa(L(k)), \kappa(R(k))).$$

Тогава за $a = 4k+1$ условието (3.16) ще бъде осигурено автоматично:

$$\varphi_{\kappa(a)} = \varphi_{comp(\kappa(L(k)),\kappa(R(k)))} = f_a.$$

За да определим κ и в останалите случаи, ще ни е нужно да знаем, че операциите Π и R също са ефективни и значи имат примитивно рекурсивни индексни функции.

Задача 3.35. Докажете, че съществуват примитивно рекурсивни функции pi и ro, такива че за всяко a и b:
а) $\varphi_{pi(a,b)} = \Pi(\varphi_a, \varphi_b)$;
б) $\varphi_{ro(a)} = \mathsf{R}(\varphi_a)$.

Решение. а) Разглеждаме функцията

$$F(a,b,x) \overset{\text{деф}}{\simeq} \Pi(\varphi_a, \varphi_b)(x) \simeq \Pi(\varphi_a(x), \varphi_b(x)).$$

Тя е изчислима, защото можем да я препишем като

$$F(a,b,x) \simeq \Pi(\Phi_1(a,x), \Phi_1(b,x)).$$

Тогава по S_n^m-теоремата ще съществува примитивно рекурсивна функция pi, такава че

$$\varphi_{pi(a,b)}(x) \simeq F(a,b,x) \simeq \Pi(\varphi_a, \varphi_b)(x)$$

за всяко a, b, x. Това означава, че

$$\varphi_{pi(a,b)} = \Pi(\varphi_a, \varphi_b)$$

за всяко a и b.

б) Разсъждаваме аналогично, като разглеждаме функцията

$$G(a,x) \overset{\text{деф}}{\simeq} \mathsf{R}(\varphi_a)(x) \simeq \varphi_a^{L(x)}(R(x)) \simeq \underbrace{\varphi_a(\ldots\varphi_a}_{L(x)\ \text{пъти}}(R(x))\ldots).$$

За да покажем, че G е изчислима, отново минаваме през универсалната функция Φ_1. Ще имаме

$$G(a,x) \simeq \underbrace{\Phi_1(a,\dots\Phi_1}_{L(x)\text{ пъти}}(a,R(x))\dots) \simeq \Phi_1^*(L(x),a,R(x)).$$

Като си спомним за *Задача* 1.14, можем да твърдим, че G е изчислима. Прилагаме и към нея S_n^m-теоремата и получаваме, че за някоя примитивно рекурсивна функция — да я наречем ro — ще е изпълнено

$$\varphi_{ro(a)}(x) \simeq G(a,x) \simeq \mathsf{R}(\varphi_a)(x)$$

за всяко a и x. Оттук за всяко a ще имаме

$$\varphi_{ro(a)} = \mathsf{R}(\varphi_a).$$

□

Като имаме опита от случая $a = 4k+1$, ни е съвсем ясно как да дефинираме $\kappa(a)$ при $a = 4k+2$ и $a = 4k+3$:

$$\kappa(4k+2) \overset{\text{деф}}{=} pi(\kappa(L(k)),\kappa(R(k))) \quad \text{и} \quad \kappa(4k+3) \overset{\text{деф}}{=} ro(\kappa(k)).$$

Като използваме свойствата на pi и ro от *Задача* 3.35, както и предположението, че върху по-малки индекси κ удовлетворява условието (3.16), ще имаме

$$f_{4k+2} = \mathsf{\Pi}(f_{L(k)}, f_{R(k)}) \overset{\text{и.х.}}{=} \mathsf{\Pi}(\varphi_{\kappa(L(k))}, \varphi_{\kappa(R(k))}) = \varphi_{pi(\kappa(L(k)),\kappa(R(k)))} = \varphi_{\kappa(4k+2)}$$

и аналогично

$$f_{4k+3} = \mathsf{R}(f_k) \overset{\text{и.х.}}{=} \mathsf{R}(\varphi_{\kappa(k)}) = \varphi_{ro(\kappa(k))} = \varphi_{\kappa(4k+3)}.$$

Така получихме, че $\kappa(a)$ удовлетворява условието (3.16) и при $a = 4k+2$ и $a = 4k+3$.

От всичко, казано дотук, със сигурност можем да твърдим, че

Твърдение 3.10. Функцията κ е примитивно рекурсивна и удовлетворява условието

$$f_a \;=\; \varphi_{\kappa(a)}$$

за всяко естествено a.

Доказателство. Това, че κ удовлетворява условието (3.16), осигурихме докато я конструирахме. Пробвайте да го покажете "начисто", като разсъждавате с пълна индукция относно a.

За да се убедим, че κ е примитивно рекурсивна, трябва да съберем в едно частите от нейната дефиниция за всеки от случаите за аргумента ѝ a. Така получаваме следната рекурсивна дефиниция:

$$\kappa(a) = \begin{cases} a_0, & \text{ако } a = 0 \\ a_1, & \text{ако } a = 4 \\ a_2, & \text{ако } a = 8 \\ a_3, & \text{ако } a = 12 \\ a_4, & \text{ако } a \equiv 0 \ (mod\ 4) \ \& \ \lfloor \frac{a}{4} \rfloor \geq 4 \\ comp(\kappa(L(\lfloor \frac{a}{4} \rfloor)), \kappa(R(\lfloor \frac{a}{4} \rfloor))), & \text{ако } a \equiv 1 \ (mod\ 4) \\ pi(\kappa(L(\lfloor \frac{a}{4} \rfloor)), \kappa(R(\lfloor \frac{a}{4} \rfloor))), & \text{ако } a \equiv 2 \ (mod\ 4) \\ ro(\kappa(\lfloor \frac{a}{4} \rfloor)), & \text{ако } a \equiv 3 \ (mod\ 4). \end{cases}$$

От нея се вижда, че κ се дефинира с пълна рекурсия, като в дефиницията ѝ участват функции и предикати, които са примитивно рекурсивни. Следователно и κ е примитивно рекурсивна. □

3.7.4 Универсална функция за примитивно рекурсивните функции

Целта ни е да покажем, че за всяко $n \geq 1$, класът $\mathcal{PR}_n$ на n-местните примитивно рекурсивни функции има универсална функция. Това ще получим като директно следствие от факта, че примитивно рекурсивните функциите са точно функциите, които са програмируеми на езика SL, и имаме транслатор от този език към езика за МНР (а МНР-изчислимите функции вече имат универсална функция).

Теорема 3.10. За всяко $n \geq 1$ класът на n-местните примитивно рекурсивни функции има универсална функция.

Доказателство. Най-напред ще конструираме универсална функция за едноместните примитивно рекурсивни функции. Да положим

$$U(a, x) \stackrel{\text{деф}}{=} f_a(x),$$

където f_a е функцията, която се определя от SL-израза E_a. Тъй като всички SL-изрази пробягват редицата

$$E_0, E_1, \ldots, E_a, \ldots$$

то всички SL-определими функции (и само те) са в редицата

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

Теорема 3.9 ни дава, че това са точно едноместните примитивно рекурсивни функции. Тогава условията 1) и 2) от дефиницията за УФ са изпълнени автоматично за функцията $U(a,x)$. Остана да покажем, че тя е изчислима. Наистина, от равенството (3.16) имаме, че

$$U(a,x) \stackrel{\text{деф}}{=} f_a(x) = \varphi_{\kappa(a)}(x) = \Phi_1(\kappa(a),x),$$

където Φ_1 е универсалната за едноместните изчислими функции. Но Φ_1 е изчислима, $\kappa(a)$ — също (тя дори е примитивно рекурсивна), което означава, че и U е изчислима. Всъщност U е рекурсивна функция, защото е тотална.

Да фиксираме сега произволно $n \geq 1$ и да положим

$$U_n(a,x_1,\ldots,x_n) \stackrel{\text{деф}}{=} U(a,\Pi_n(x_1,\ldots,x_n)).$$

Да се убедим, че U_n е универсална за класа $\mathcal{PR}_n$. Тя очевидно е рекурсивна, тъй че трябва да проверим само условията 1) и 2) от дефиницията за УФ.

Да си спомним, че представящата представяща $\hat{f}$ на всяка функция $f \in \mathcal{PR}_n$ удовлетворява условието:

$$\hat{f}(z) \stackrel{\text{деф}}{\simeq} f(J_1^n(z),\ldots,J_n^n(z)).$$

Ясно е, че $\hat{f}$ също ще е от класа $\mathcal{PR}_1$. Но тогава съгласно *Теорема* 3.9 ще съществува a, за което $\hat{f} = f_a$. Следователно

$$U_n(a,\bar{x}) \stackrel{\text{деф}}{=} U(a,\Pi_n(\bar{x})) \stackrel{\text{деф } U}{=} f_a(\Pi_n(\bar{x})) = \hat{f}(\Pi_n(\bar{x})) \stackrel{\text{деф}}{=} f(\bar{x}),$$

с други думи, $\lambda\bar{x}.U_n(a,\bar{x}) = f$, т.е. условието 1) от дефиницията за УФ е в сила. Обратно, за всяко фиксирано a имаме, че функцията

$$\lambda\bar{x}.U_n(a,\bar{x}) = \lambda\bar{x}.U(a,\Pi_n(\bar{x})) = \lambda\bar{x}.f_a(\Pi_n(\bar{x})) = f_a \circ \Pi_n$$

очевидно е примитивно рекурсивна, т.е. налице е и второто условие от дефиницията за УФ. □

Забележка. Да обърнем внимание, че съгласно *Твърдение* 3.3, никоя от универсалните функции U_n не може да е примитивно рекурсивна. Всяка от тях е пример за функция, която е рекурсивна, но не е примитивно рекурсивна. Така вече имаме строго доказателство, че класът на примитивно рекурсивните функции се включва строго класа на рекурсивните.

3.7.5 Построяване на интерпретатор за езика SL

Сега се насочваме към конструиране на интерпретатор $\boldsymbol{I}(a,x)$ за езика SL. Идеята е по дадени a и x, $\boldsymbol{I}$ да връща стойността на функцията, определена от SL-израза E_a в точката x. По-горе тази функция означихме с f_a. Следователно дефиницията на $\boldsymbol{I}$ трябва да е такава:

$$\boldsymbol{I}(a,x) \;=\; f_a(x).$$

Върху кодовете на базисните изрази е ясно как трябва да дефинираме $\boldsymbol{I}$:

$$\boldsymbol{I}(0,x) \stackrel{\text{деф}}{=} \mathcal{S}(x) = x+1; \quad \boldsymbol{I}(4,x) \stackrel{\text{деф}}{=} \mathcal{O}(x) = 0; \quad \boldsymbol{I}(8,x) \stackrel{\text{деф}}{=} I(x) = x;$$

$$\boldsymbol{I}(12,x) \stackrel{\text{деф}}{=} L(x); \quad \boldsymbol{I}(4k,x) \stackrel{\text{деф}}{=} R(x) \text{ за всяко } \; k \geq 4.$$

Идеята ни е, по подобие с конструкцията на κ в предишния раздел, да дефинираме рекурсивно $\boldsymbol{I}(a,x)$ за a такива, че E_a не са базисни. Тук, разбира се, отново ще предполагаме, че за $b < a$ вече сме осигурили $\boldsymbol{I}(b,x) = f_b(x)$ (и на това равенство по-долу ще се позоваваме като на индуктивна хипотеза).

Когато $a = 4k+1$, имаме по определение $f_a = f_{L(k)} \circ f_{R(k)}$, и значи в този случай:

$$f_a(x) = f_{L(k)}(f_{R(k)}(x)) \stackrel{\text{и.х.}}{=} f_{L(k)}(\boldsymbol{I}(R(k),x)) \stackrel{\text{и.х.}}{=} \boldsymbol{I}(L(k),\boldsymbol{I}(R(k),x)).$$

Това означава, че за да осигурим $\boldsymbol{I}(a,x) = f_a(x)$, за $\boldsymbol{I}(a,x)$ трябва да положим

$$\boldsymbol{I}(a,x) = \boldsymbol{I}(L(k),\boldsymbol{I}(R(k),x)),$$

или разписано само чрез a:

$$\boldsymbol{I}(a,x) = \boldsymbol{I}(L(\lfloor\frac{a}{4}\rfloor),\boldsymbol{I}(R(\lfloor\frac{a}{4}\rfloor),x)).$$

Аналогично, за $a = 4k+2$ по определение $f_a = \mathsf{\Pi}(f_{L(k)}, f_{R(k)})$ и следователно

$$f_a(x) = \mathsf{\Pi}(f_{L(k)}, f_{R(k)})(x) \stackrel{\text{и.х.}}{=} \Pi(\boldsymbol{I}(L(k),x),\boldsymbol{I}(R(k),x)).$$

Тогава за $\boldsymbol{I}(a,x)$ при $a = 4k+2$ полагаме

$$\boldsymbol{I}(a,x) = \Pi(\boldsymbol{I}(L(\lfloor\frac{a}{4}\rfloor),x),\boldsymbol{I}(R(\lfloor\frac{a}{4}\rfloor),x)).$$

Остана случая $a = 4k+3$. В този случай $E_a = \rho(E_k)$ и съответно $f_{4k+3} = \mathsf{R}(f_k)$. Следователно

$$f_a(x) = \mathsf{R}(f_k)(x) = f_k^{L(x)}(R(x)) = \underbrace{f_k(\ldots f_k}_{L(x)\text{ пъти}}(R(x)\ldots) \stackrel{\text{и.х.}}{=} \underbrace{\boldsymbol{I}(k,\ldots,\boldsymbol{I}(k}_{L(x)\text{ пъти}},R(x))\ldots).$$

Тогава за $\boldsymbol{I}(a,x)$ в този случай полагаме

$$\boldsymbol{I}(a,x) = \underbrace{\boldsymbol{I}(\lfloor\frac{a}{4}\rfloor,\ldots,\boldsymbol{I}(\lfloor\frac{a}{4}\rfloor}_{L(x)\text{ пъти}},R(x))\ldots).$$

Дали можем да твърдим, че в такъв случай $\boldsymbol{I}$ е точно функцията, която ни трябва? Това беше основният ни замисъл, докато я конструирахме, но да формулираме този факт като твърдение:

Твърдение 3.11. Ако $\boldsymbol{I}$ удовлетворява (3.17), то за нея е вярно, че $\boldsymbol{I}(a,x) = f_a(x)$ за всяко a и x.

Доказателство. С пълна индукция относно a показваме, че

$$\forall x\ \boldsymbol{I}(a,x) = f_a(x).$$

Ще пропуснем доказателството, защото всички необходими разсъждения вече проведохме по-горе. □

Така получаваме, че $\boldsymbol{I}$ трябва да удовлетворява следната рекурсивна схема:

$$\boldsymbol{I}(a,x) = \begin{cases} x+1, & \text{ако } a = 0 \\ 0, & \text{ако } a = 4 \\ x, & \text{ако } a = 8 \\ L(x), & \text{ако } a = 12 \\ R(x), & \text{ако } a \equiv 0\ (mod\ 4)\ \&\ \lfloor\frac{a}{4}\rfloor \geq 4 \\ \boldsymbol{I}(L(\lfloor\frac{a}{4}\rfloor),\boldsymbol{I}(R(\lfloor\frac{a}{4}\rfloor),x)), & \text{ако } a \equiv 1\ (mod\ 4) \\ \Pi(\boldsymbol{I}(L(\lfloor\frac{a}{4}\rfloor),x),\boldsymbol{I}(R(\lfloor\frac{a}{4}\rfloor),x)), & \text{ако } a \equiv 2\ (mod\ 4) \\ \underbrace{\boldsymbol{I}(\lfloor\frac{a}{4}\rfloor,\ldots,\boldsymbol{I}(\lfloor\frac{a}{4}\rfloor}_{L(x)\text{ пъти}},R(x))\ldots), & \text{ако } a \equiv 3\ (mod\ 4). \end{cases} \tag{3.17}$$

Идеята ни е да приложим теоремата за определимост по рекурсия и да получим, че съществува *изчислима* функция удовлетворяваща рекурсивното условие (3.17). Но за да сме сигурни, че точно тази изчислима функция ще е нашата $\boldsymbol{I}$, първо да се убедим, че (3.17) не може да е вярно за повече от една функция.

Твърдение 3.12. Ако функциите $\boldsymbol{I}$ и $\boldsymbol{J}$ удовлетворяват условието(3.17), то $\boldsymbol{I} = \boldsymbol{J}$.

Доказателство. Трябва да покажем, че за всяко a и x:

$$\boldsymbol{I}(a,x) = \boldsymbol{J}(a,x).$$

Да означим

$$P(a) \overset{\text{деф}}{\iff} \forall x\ \boldsymbol{I}(a,x) = \boldsymbol{J}(a,x).$$

С пълна индукция относно a ще покажем, че $\forall a\ P(a)$.

Когато $a \equiv 0\ (mod\ 4)$ е ясно, че $\boldsymbol{I}(a,x) = \boldsymbol{J}(a,x)$ за всяко x, защото в този случай те се определят явно.

Когато $a \equiv 1\ (mod\ 4)$, със сигурност $L(\lfloor \frac{a}{4} \rfloor) < a$ и $R(\lfloor \frac{a}{4} \rfloor) < a$. Фиксираме произволно $x \in \mathbb{N}$ и прилагаме индуктивната хипотеза $P(L(\lfloor \frac{a}{4} \rfloor))$ и $P(L(\lfloor \frac{a}{4} \rfloor))$. Ще имаме

$$\begin{aligned}\boldsymbol{I}(a,x) &\overset{(3.17)}{=} \boldsymbol{I}(L(\lfloor \tfrac{a}{4} \rfloor), \boldsymbol{I}(R(\lfloor \tfrac{a}{4} \rfloor), x)) \overset{\text{и.х.}}{=} \boldsymbol{I}(L(\lfloor \tfrac{a}{4} \rfloor), \boldsymbol{J}(R(\lfloor \tfrac{a}{4} \rfloor), x)) \\ &\overset{\text{и.х.}}{=} \boldsymbol{J}(L(\lfloor \tfrac{a}{4} \rfloor), \boldsymbol{J}(R(\lfloor \tfrac{a}{4} \rfloor), x)) \overset{(3.17)}{=} \boldsymbol{J}(a,x).\end{aligned}$$

Аналогично (и дори малко по-просто), за $a \equiv 2\ (mod\ 4)$ можем да запишем:

$$\begin{aligned}\boldsymbol{I}(a,x) &\overset{(3.17)}{=} \Pi(\boldsymbol{I}(L(\lfloor \tfrac{a}{4} \rfloor), x), \boldsymbol{I}(R(\lfloor \tfrac{a}{4} \rfloor), x)) \\ &\overset{\text{и.х.}}{=} \Pi(\boldsymbol{J}(L(\lfloor \tfrac{a}{4} \rfloor), x), \boldsymbol{J}(R(\lfloor \tfrac{a}{4} \rfloor), x)) \overset{(3.17)}{=} \boldsymbol{J}(a,x).\end{aligned}$$

При $a \equiv 3\ (mod\ 4)$ е вярно, че $\lfloor \frac{a}{4} \rfloor < a$, тъй че отново можем да приложим индуктивната хипотеза $P(\lfloor \frac{a}{4} \rfloor)$, този път — $L(x)$ пъти:

$$\begin{aligned}\boldsymbol{I}(a,x) &\overset{(3.17)}{=} \underbrace{\boldsymbol{I}(\lfloor \tfrac{a}{4} \rfloor, \ldots, \boldsymbol{I}(\lfloor \tfrac{a}{4} \rfloor}_{L(x)\ \text{пъти}}, R(x)) \ldots) \overset{\text{и.х.}}{=} \ \ldots \\ &\overset{\text{и.х.}}{=} \underbrace{\boldsymbol{J}(\lfloor \tfrac{a}{4} \rfloor, \ldots, \boldsymbol{J}(\lfloor \tfrac{a}{4} \rfloor}_{L(x)\ \text{пъти}}, R(x)) \ldots) \overset{(3.17)}{=} \boldsymbol{J}(a,x).\end{aligned}$$

□

Сега вече сме в състояние да докажем основното твърдение в този раздел:

Твърдение 3.13. Функцията $\boldsymbol{I}$ е рекурсивна.

Доказателство. Ще използваме теоремата за определимост по рекурсия, което означава, че ще търсим $\boldsymbol{I}$ във вида $\varphi_e^{(2)}$ за някое e. По-точно, целта ни е да видим, че една такава функция $\varphi_e^{(2)}$ удовлетворява рекурсивната схема (3.17), с която дефинирахме $\boldsymbol{I}$, т. е. за нея е вярно, че

$$\varphi_e^{(2)}(a,x) = \underbrace{\begin{cases} x+1, & \text{ако } a = 0 \\ 0, & \text{ако } a = 4 \\ x, & \text{ако } a = 8 \\ L(x), & \text{ако } a = 12 \\ R(x), & \text{ако } a \equiv 0 \ (mod\ 4) \ \&\ \lfloor \frac{a}{4} \rfloor \geq 4 \\ \varphi_e^{(2)}(L(\lfloor \frac{a}{4} \rfloor), \varphi_e^{(2)}(R(\lfloor \frac{a}{4} \rfloor), x)), & \text{ако } a \equiv 1 \ (mod\ 4) \\ \Pi(\varphi_e^{(2)}(L(\lfloor \frac{a}{4} \rfloor), x), \varphi_e^{(2)}(R(\lfloor \frac{a}{4} \rfloor), x)), & \text{ако } a \equiv 2 \ (mod\ 4) \\ \underbrace{\varphi_e^{(2)}(\lfloor \frac{a}{4} \rfloor, \ldots, \varphi_e^{(2)}(\lfloor \frac{a}{4} \rfloor}_{L(x) \text{ пъти}}, R(x)) \ldots), & \text{ако } a \equiv 3 \ (mod\ 4). \end{cases}}_{F(e,a,x)}$$

(3.18)

Да означим функцията вдясно с F. Искаме да покажем, че тя е изчислима. Минаваме стандартно през универсалната функция Φ_2:

$$F(e,a,x) \simeq \underbrace{\begin{cases} x+1, & \text{ако } a = 0 \\ 0, & \text{ако } a = 4 \\ x, & \text{ако } a = 8 \\ L(x), & \text{ако } a = 12 \\ R(x), & \text{ако } a \equiv 0 \ (mod\ 4) \ \&\ \lfloor \frac{a}{4} \rfloor \geq 4 \\ \Phi_2(e, L(\lfloor \frac{a}{4} \rfloor), \Phi_2(e, R(\lfloor \frac{a}{4} \rfloor), x)), & \text{ако } a \equiv 1 \ (mod\ 4) \\ \Pi(\Phi_2(e, L(\lfloor \frac{a}{4} \rfloor), x), \Phi_2(e, R(\lfloor \frac{a}{4} \rfloor), x)), & \text{ако } a \equiv 2 \ (mod\ 4) \\ \underbrace{\varphi_e^{(2)}(\lfloor \frac{a}{4} \rfloor, \ldots, \varphi_e^{(2)}(\lfloor \frac{a}{4} \rfloor}_{L(x) \text{ пъти}}, R(x)) \ldots), & \text{ако } a \equiv 3 \ (mod\ 4). \end{cases}}_{g(e,a,x)}$$

Да означим с g функцията от последния ред на дефиницията на F. Искаме да покажем, че тя е изчислима. Затова да разгледаме по-общата функция

$$G(n,e,a,x) \overset{\text{деф}}{\simeq} \underbrace{\varphi_e^{(2)}(a, \ldots \varphi_e^{(2)}}_{n \text{ пъти}}(a,x) \ldots).$$

За G имаме следната примитивно рекурсивна схема:

$$\begin{cases} G(0,e,a,x) = x \\ G(n+1,e,a,x) \simeq \varphi_e^{(2)}(a, G(n,e,a,x)) \simeq \Phi_2(e,a,G(n,e,a,x)). \end{cases}$$

откъдето се вижда, че G е изчислима. Но $g(e,a,x) \simeq G(L(x), e, \lfloor \frac{a}{4} \rfloor, R(x))$, следователно и g е изчислима. Сега вече можем да твърдим, че и F е

изчислима. Но тогава по теоремата за определимост по рекурсия ще съществува индекс e_0, такъв че

$$\varphi^{(2)}_{e_0}(a,x) \simeq F(e_0,a,x)$$

за всяко a и x. Това означава, че $\varphi^{(2)}_{e_0}$ удовлетворява равенството (3.18), а оттук и (3.17). Но *Твърдение* 3.12 казва, че има единствена функция, която го удовлетворява и това е $\boldsymbol{I}$. Така получаваме $\boldsymbol{I} = \varphi^{(2)}_{e_0}$, което значи, че $\boldsymbol{I}$ е изчислима. Но тази функция очевидно е тотална и значи тя е рекурсивна. □

Задача за ЕК. Докажете, че функцията $\boldsymbol{I}$ е рекурсивна като използвате първата теорема за рекурсия.

Глава 4

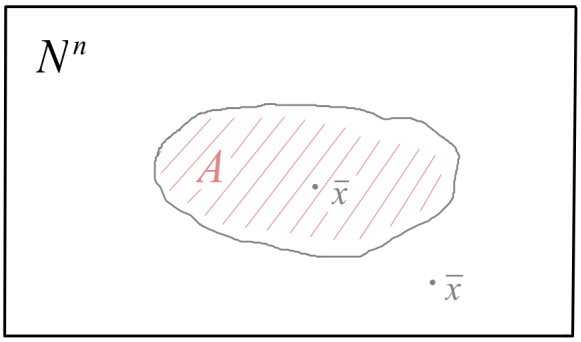
Разрешимост, полуразрешимост и неразрешимост

4.1 Разрешими множества

Днес ще изучим основните свойства на *разрешимите множества*, които се явяват математическа формализация на алгоритмично разрешимите проблеми. На английски ще ги срещнете под най-различни имена: *decidable, solvable, computable, recursive.*

4.1.1 Характеристична функция на множество

Навсякъде под *множество* ще разбираме подмножество на някаква декартова степен на $\mathbb{N}$. Интуитивно, едно множество е разрешимо, ако има алгоритъм, който може да каже (да разреши, to decide) дали дадена n-торка $\bar{x}$ принадлежи на A или не (като и в двата случая спира и дава някаква индикация).



За да дефинираме формално понятието разрешимост, най-напред ще въведем *характеристична функция* на множество A.

Определение 4.1. *Характеристична функция* на множеството $A \subseteq \mathbb{N}^n$ наричаме функцията $\chi_A \colon \mathbb{N}^n \longrightarrow \mathbb{N}$, която се дефинира с равенството:

$$\chi_A(\bar{x}) = \begin{cases} 0, & \text{ако } \bar{x} \in A \\ 1, & \text{ако } \bar{x} \notin A. \end{cases}$$

Да обърнем внимание, че и тук, както при дефиницията на характеристична функция на *предикат,* "позитивния" случай $\bar{x} \in A$ кодираме с 0, а негативния $\bar{x} \notin A$ — с 1.

Определение 4.2. Казваме, че множеството $A \subseteq \mathbb{N}^n$ е *разрешимо*, ако неговата характеристична функция χ_A е рекурсивна.

Да отбележим, че характеристичната функция на всяко множество A е *тотална*, тъй че условието за рекурсивност на χ_A съвпада с условието за изчислимост на χ_A.

Горната дефиниция за разрешимост на множество е съвсем аналогична на определението за разрешимост на *предикат*, което не е странно, защото едното понятие се свежда към другото.

Примери. Разрешими са множествата $\mathbb{N}^n$ за всяко $n \geq 1$, празното множество, множеството на четните числа, на простите числа, на съвършените числа и пр. Всички тези множества всъщност имат примитивно рекурсивни характеристични функции.

Въобще, вярно е, че ако предикатът p е разрешим, то и множеството $A = \{\bar{x} \mid p(\bar{x}) = true\}$ е разрешимо и обратно, ако A е разрешимо, то е разрешим и предикатът p, който се дефинира с еквивалентността

$$p(\bar{x}) \overset{\text{деф}}{\iff} \bar{x} \in A.$$

Задача 4.1. Докажете, че всяко крайно множество е разрешимо.

Решение. Нека $A \subseteq \mathbb{N}$. Ако A е празното множество, то χ_A е функцията $\lambda x.1$, която е примитивно рекурсивна. Ако $A = \{a_1, \ldots, a_k\}$, то неговата характеристична функция се дефинира с разглеждане на случаи и също е примитивно рекурсивна:

$$\chi_A(x) = \begin{cases} 0, & \text{ако } x = a_1 \\ . & . \ . \ . \ . \ . \ . \\ 0, & \text{ако } x = a_k \\ 1, & \text{в останалите случаи.} \end{cases}$$

Ако $A = \{\bar{a}^1, \ldots \bar{a}^k\} \subseteq \mathbb{N}^n$, за χ_A отново е вярно, че

$$\chi_A(\bar{x}) = \begin{cases} 0, & \text{ако } \bar{x} = \bar{a}^1 \\ . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ 0, & \text{ако } \bar{x} = \bar{a}^k \\ 1, & \text{в останалите случаи.} \end{cases}$$

Тук предикатът $p_i(\bar{x}) \iff \bar{x} = \bar{a}^i$ е примитивно рекурсивен, защото ако $\bar{a}^i = (a_1^i, \ldots, a_n^i)$, то p_i можем да препишем като

$$p_i(x_1, \ldots, x_n) \iff x_1 = a_1^i \ \& \ \ldots \ \& \ x_n = a_n^i.$$

Следователно χ_A е примитивно рекурсивна и значи A е разрешимо. □

4.1.2 Основни свойства на разрешимите множества

В този раздел ще покажем някои основни факти, отнасящи се до разрешими множества, които ще използваме многократно по-нататък.

Твърдение 4.1. (НДУ за разрешимост на множество) Множеството $A \subseteq \mathbb{N}^n$ е разрешимо тогава и само тогава, когато съществува рекурсивна функция f, такава че за всяко $\bar{x} \in \mathbb{N}^n$:

$$\bar{x} \in A \iff f(\bar{x}) = 0.$$

Доказателство. Ако A е разрешимо, то $f := \chi_A$ очевидно удовлетворява горното условие (като в добавка f е 0-1 функция).

Обратно, ако има рекурсивна функция f, за която горното условие е в сила, то очевидно за всяко $\bar{x} \in \mathbb{N}^n$ ще е изпълнено

$$\chi_A(\bar{x}) = sg(f(\bar{x}))$$

и следователно χ_A е рекурсивна. □

Твърдение 4.2. (Основни свойства на разрешимите множества)

1) Нека $A \subseteq \mathbb{N}^n$ и $B \subseteq \mathbb{N}^n$ са разрешими множества. Тогава са разрешими и множествата $A \cup B$, $A \cap B$ и $\bar{A}$.

2) Нека $A \subseteq \mathbb{N}^n$ е разрешимо. Тогава са разрешими и множествата

$$C = \{(\bar{x}, y) \mid \exists z_{z \leq y} \ (\bar{x}, z) \in A\} \quad \text{и}$$
$$D = \{(\bar{x}, y) \mid \forall z_{z \leq y} \ (\bar{x}, z) \in A\}.$$

3) Ако $A \subseteq \mathbb{N}^n$ и $B \subseteq \mathbb{N}^k$ са разрешими, то е разрешимо и тяхното декартово произведение $A \times B$.

Доказателство. 1) По определение:

$$\begin{aligned}\bar{x} \in A \cup B &\iff \bar{x} \in A \ \vee \ \bar{x} \in B \iff \chi_A(\bar{x}) = 0 \ \vee \ \chi_B(\bar{x}) = 0 \\ &\iff \underbrace{\chi_A(\bar{x}).\chi_B(\bar{x})}_{f(\bar{x})} = 0.\end{aligned}$$

Функцията f очевидно е рекурсивна, откъдето съгласно *Твърдение* 4.1 $A \cup B$ е разрешимо.

За $A \cap B$ ще имаме:

$$\begin{aligned}\bar{x} \in A \cap B &\iff \bar{x} \in A \ \& \ \bar{x} \in B \iff \chi_A(\bar{x}) = 0 \ \& \ \chi_B(\bar{x}) = 0 \\ &\iff \underbrace{\chi_A(\bar{x}) + \chi_B(\bar{x})}_{g(\bar{x})} = 0.\end{aligned}$$

където g е рекурсивна, и значи отново можем да приложим *Твърдение* 4.1.

За характеристичната функция на $\bar{A} = \mathbb{N}^n \setminus A$ очевидно

$$\chi_{\bar{A}} = \bar{sg} \circ \chi_A.$$

2) Условието за принадлежност към C можем да препишем така:

$$\begin{aligned}(\bar{x}, y) \in C &\iff (\bar{x}, 0) \in A \ \vee \ \ldots \ \vee \ (\bar{x}, y) \in A \\ &\iff \chi_A(\bar{x}, 0) = 0 \ \vee \ \ldots \ \vee \ \chi_A(\bar{x}, y) = 0 \iff \prod_{z \leq y} \chi_A(\bar{x}, z) \ = \ 0,\end{aligned}$$

откъдето се вижда, че $\chi_C(\bar{x}, y) = \prod\limits_{z \leq y} \chi_A(\bar{x}, z)$ е рекурсивна.

По същия начин за множеството D ще имаме:

$$\begin{aligned}(\bar{x}, y) \in D &\iff (\bar{x}, 0) \in A \ \& \ \ldots \ \& \ (\bar{x}, y) \in A \\ &\iff \chi_A(\bar{x}, 0) = 0 \ \& \ \ldots \ \& \ \chi_A(\bar{x}, y) = 0 \iff \sum_{z \leq y} \chi_A(\bar{x}, z) \ = \ 0.\end{aligned}$$

Понеже функцията $f(\bar{x}, y) = \sum\limits_{z \leq y} \chi_A(\bar{x}, z)$ е рекурсивна, то по *Твърдение* 4.1 множеството D ще е разрешимо.

3) От определението за декартово произведение имаме за произволни $\bar{x} \in \mathbb{N}^n$ и $\bar{y} \in \mathbb{N}^k$:

$$\begin{aligned}(\bar{x}, \bar{y}) \in A \times B &\iff \bar{x} \in A \ \& \ \bar{y} \in B \iff \chi_A(\bar{x}) = 0 \ \& \ \chi_B(\bar{y}) = 0 \\ &\iff \chi_A(\bar{x}) + \chi_B(\bar{y}) = 0,\end{aligned}$$

откъдето отново от *Твърдение* 4.1 ще следва, че $A \times B$ е разрешимо. $\square$

За в бъдеще ще ни трябва и следното обобщение на *Твърдение* 4.2 2):

Следствие 4.1. Нека A е разрешимо множество, а $b(\bar{x})$ е рекурсивна функция. Тогава са разрешими и следните множества C^* и D^*:

$$\begin{aligned} C^* &= \{\bar{x} \mid \exists z_{z \leq b(\bar{x})}\ (\bar{x}, z) \in A\} \\ D^* &= \{\bar{x} \mid \forall z_{z \leq b(\bar{x})}\ (\bar{x}, z) \in A\}. \end{aligned}$$

Доказателство. Като използваме, че характеристичните функции на множествата C и D са рекурсивни, веднага получаваме същото и за χ_{C^*} и χ_{D^*}, защото

$$\chi_{C^*}(\bar{x}) = \chi_C(\bar{x}, b(\bar{x})) \quad \text{и} \quad \chi_{D^*}(\bar{x}) = \chi_D(\bar{x}, b(\bar{x})).$$

□

Твърдение 4.3. (Първообразът на разрешимо множество чрез рекурсивна функция е разрешимо множество) Нека $A \subseteq \mathbb{N}$ е разрешимо, а f е n-местна рекурсивна функция. Тогава е разрешимо и множеството

$$f^{-1}(A) \overset{\text{деф}}{=} \{\bar{x} \mid f(\bar{x}) \in A\}.$$

Доказателство. По дефиниция

$$\bar{x} \in f^{-1}(A) \iff f(\bar{x}) \in A$$

и следователно $\chi_{f^{-1}(A)} = \chi_A \circ f$ ще е рекурсивна като композиция на рекурсивни функции. □

Ако се питате дали подобно твърдение е вярно и за *образ* на разрешимо множество, отговорът е НЕ. Контрапример ще можем да дадем следващия път, когато се запознаем с *полуразрешимите* множества.

Следващото твърдение дава връзка между изчислителната сложност на функция f и нейната графика G_f.

Твърдение 4.4. Нека f е тотална функция. Тогава f е рекурсивна тогава и само тогава, когато нейната графика е разрешимо множество.

Доказателство. Нека f е n-местна рекурсивна функция. Тогава за всяко $\bar{x} \in \mathbb{N}^n$ и $y \in \mathbb{N}$ е вярно, че

$$(\bar{x}, y) \in G_f \iff f(\bar{x}) = y \iff \underbrace{|f(\bar{x}) - y|}_{g(\bar{x},y)} = 0.$$

Тъй като функцията g е рекурсивна, то по *Твърдение* 4.1 множеството G_f ще е разрешимо.

За обратната посока използваме, че всяка функция f (включително и нетотална) можем да изразим чрез нейната графика по следния начин:

$$f(\bar{x}) \simeq \mu y[(\bar{x}, y) \in G_f] \simeq \mu y[\chi_{G_f}(\bar{x}, y) = 0].$$

От това изразяване се вижда, че ако G_f е разрешима, то χ_{G_f} е рекурсивна и следователно f ще е частично рекурсивна. Но по условие имаме, че тази функция е тотална, и значи тя е рекурсивна. $\square$

Забележка. Да отбележим, че изискването f да е тотална във формулировката на горното твърдение е съществено. Възможно е графиката на f да е разрешима, без непременно f да е тотална. Като най-прост пример можем да вземем графиката на никъде недефинираната функция $\emptyset^{(1)}$, която е разрешима (защото е празното множество), но $\emptyset^{(1)}$ не е тотална.

Задача 4.2. Нека графиката на n-местната функция f е разрешима. Нека още f за някоя рекурсивна функция b, е вярно, че за всяко $\bar{x} \in \mathbb{N}^n$:

$$!f(\bar{x}) \implies f(\bar{x}) \leq b(\bar{x}).$$

Докажете, че f е изчислима функция с разрешима дефиниционна област.

Забележка. Ако за функциите $f(\bar{x})$ и $b(\bar{x})$ е изпълнена горната импликация, ще казваме, че f *се мажорира* от b и ще пишем $f \leq b$.

Решение. За произволно $\bar{x} \in \mathbb{N}^n$ имаме

$$\bar{x} \in Dom(f) \iff \exists y\ f(\bar{x}) \simeq y \iff \exists y_{y \leq b(\bar{x})}\ f(\bar{x}) \simeq y.$$

Тъй като ограничените квантори запазват разрешимостта (*Следствие* 4.1), то $Dom(f)$ ще е разрешимо.

За изчислимостта на f използваме отново наблюдението от доказателството на горното твърдение, а именно, че

$$f(\bar{x}) \simeq \mu y[\chi_{G_f}(\bar{x}, y) = 0].$$

$\square$

4.1.3 Няколко задачи за разрешими множества

Нека A и B са множества от естествени числа. Дефинираме тяхната *директна сума* $A \oplus B$ по следния начин:

$$A \oplus B \stackrel{\text{деф}}{=} \{2x \mid x \in A\} \cup \{2x+1 \mid x \in B\}.$$

В следващата задача ще видим, че $A \oplus B$ "кодира" множествата A и B, като освен това запазва разрешимостта.

Задача 4.3. Нека $A \subseteq \mathbb{N}$ и $B \subseteq \mathbb{N}$. Докажете, че A и B са разрешими тогава и само тогава, когато тяхната директна сума $A \oplus B$ е разрешима.

Решение. Нека A и B са разрешими. Да разпишем условието за принадлежност към $A \oplus B$:

$$\begin{aligned} z \in A \oplus B &\iff (z \text{ е четно } \& \ \frac{z}{2} \in A) \ \vee \ (z \text{ е нечетно } \& \ \frac{z-1}{2} \in B) \\ &\iff rem(2,z)=0 \ \& \ \chi_A(\lfloor \frac{z}{2} \rfloor)=0 \ \vee \ \underbrace{rem(2,z)=1}_{\bar{sg}(rem(2,z))=0} \& \ \chi_B(\lfloor \frac{z}{2} \rfloor)=0 \\ &\iff \underbrace{(rem(2,z)+\chi_A(\lfloor \frac{z}{2} \rfloor)).(\bar{sg}(rem(2,z))+\chi_B(\lfloor \frac{z}{2} \rfloor))}_{f(z)} = 0 \\ &\iff f(z)=0. \end{aligned}$$

Понеже функцията f е рекурсивна, съгласно *Твърдение* 4.1, $A \oplus B$ ще е разрешимо.

Друг начин да решим тази посока на задачата е да представим $A \oplus B$ като израз, в който участват разрешими множества и теоретико-множествени операции, които запазват разрешимостта. За целта да означим с E множеството на четните числа. Нека още

$$A_0 = \{z \mid \lfloor \frac{z}{2} \rfloor \in A\} \quad \text{и} \quad B_0 = \{z \mid \lfloor \frac{z}{2} \rfloor \in B\}.$$

Множествата A_0 и B_0 са първообрази на A и B чрез $f(z) = \lfloor \frac{z}{2} \rfloor$ и съгласно *Твърдение* 4.3 те са разрешими. Тогава $A \oplus B$ можем да представим така:

$$A \oplus B \ = \ (E \cap A_0) \ \cup \ (\bar{E} \cap B_0).$$

Сега прилагаме *Твърдение* 4.2 1), за да получим, че $A \oplus B$ е разрешимо.

Всъщност може би най- краткият начин да решим тази посока на задачата е да забележим, че характеристичната функция на $A \oplus B$ се изразява чрез характеристичните на A и B по следния начин:

$$\chi_{A \oplus B}(z) = \begin{cases} \chi_A(\lfloor \frac{z}{2} \rfloor), & \text{ако } z \text{ е четно} \\ \chi_B(\lfloor \frac{z}{2} \rfloor), & \text{ако } z \text{ е нечетно.} \end{cases}$$

За обратната посока, да приемем, че директната сума $A \oplus B$ е разрешимо множество. Лесно се съобразява, че за всяко естествено x са в сила еквивалентностите:

$$x \in A \iff 2x \in A \oplus B \quad \text{и} \quad x \in B \iff 2x+1 \in A \oplus B.$$

Сега вече можем да приложим *Твърдение* 4.3 или пък директно да изразим характеристичните функции на A и B:

$$\chi_A(x) = \chi_{A \oplus B}(2x) \quad \text{и} \quad \chi_B(x) = \chi_{A \oplus B}(2x+1).$$

От тези равенства, в частност, се вижда, че ако имаме разрешаващ алгоритъм за $A \oplus B$, можем да разрешаваме алгоритмично A и B, и следователно $A \oplus B$ кодира в едно двете множества A и B. $\square$

Да дефинираме и *директното произведение* $A \otimes B$ на две множества от естествени числа A и B както следва:

$$A \otimes B \stackrel{\text{деф}}{=} \{\Pi(x, y) \mid x \in A \ \& \ y \in B\}.$$

Твърдим, че за директното произведение $A \otimes B$ е вярно същото като за $A \oplus B$, но само ако A и B не са празни множества.

Задача 4.4. Нека $\emptyset \neq A \subseteq \mathbb{N}$ и $\emptyset \neq B \subseteq \mathbb{N}$. Докажете, че A и B са разрешими тогава и само тогава, когато директното им произведение $A \otimes B$ е разрешимо.

Решение. Правата посока на задачата е ясна; имаме

$$z \in A \otimes B \iff L(z) \in A \ \& \ R(z) \in B \iff \chi_A(L(z)) + \chi_B(R(z)) = 0.$$

Да отбележим, че тук никъде не използваме, че A и B не са празни. За обратната посока, обаче, това ще е важно.

Наистина, ако $A = \emptyset$, то $A \otimes B$ също ще е $\emptyset$, и следователно ще е разрешимо, докато в същото време B може да е произволно сложно. Така че от $A \otimes B$ — разрешимо в общия случай *не следва*, че A и B ще са разрешими. Освен това, при $A = \emptyset$ директното произведение $A \otimes B = \emptyset$ и следователно то не може да "запомни" в себе си B.

Нека сега A и B са непразни множества с разрешимо декартово произведение $A \otimes B$. Ако разпишем директно условието за принадлежност към A от дефиницията на $A \otimes B$, ще получим

$$x \in A \iff \exists y(y \in B \ \& \ \Pi(x, y) \in A \otimes B),$$

което намесва и B и очевидно не дава начин да видим, че A е разрешимо.

Затова разсъждаваме другояче: щом $B \neq \emptyset$, ще съществува поне едно $y_0 \in B$. Да фиксираме едно такова y_0. Тогава очевидно

$$x \in A \iff \Pi(x, y_0) \in A \otimes B.$$

Получихме, че $\chi_A(x) = \chi_{A \otimes B}(\Pi(x, y_0))$ и значи χ_A е рекурсивна. Аналогично разсъждаваме, за да покажем, че и χ_B е рекурсивна.

Ако, обаче, поискаме да разберем дали дадено число x е в A, като разполагаме с програмата, разпознаваща $A \otimes B$ (т.е. като знаем $\chi_{A \otimes B}$), виждаме, че има проблем: за да пресметнем $\chi_A(x)$ ни трябва конкретно y_0 от B, а ние не го знаем (знаем само, че такова съществува).

Всъщност, като разполагаме с $\chi_{A\otimes B}$, все пак можем да намерим *алгоритмично* елемент $y_0 \in B$. За целта намираме първото $z \in A\otimes B$. Знаем, че то трябва да е във вида $\Pi(x_0, y_0)$ за някои $x_0 \in A$ и $y_0 \in B$. Тогава

$$y_0 = R(\mu z[\chi_{A\otimes B}(z) = 0]).$$

Сега преписваме χ_A само чрез $\chi_{A\otimes B}$ (без y_0):

$$\chi_A(x) = \chi_{A\otimes B}(\Pi(x, R(\mu z[\chi_{A\otimes B}(z) = 0]))).$$

Ясно е вече, че като разполагаме с програма, разпознаваща $A\otimes B$, можем да напишем програма, разпознаваща A. □

Следващата задача е още една илюстрация на явлението, което наблюдавахме току-що — че едно е да знаем, че дадено множество е разрешимо, а съвсем друго е разполагаме с алгоритъм, който го разрешава.

Задача 4.5. Разрешимо ли е множеството A, което се дефинира като:

$$A = \{n \mid \text{в десетичния запис на числото } \pi \text{ има блок от } n \text{ седмици }\}?$$

(Тук имаме предвид, че ако π е от вида $3,1415\ldots\underbrace{7\ldots7}_{n \text{ пъти}}\ldots$, то $n \in A$.)

Решение. Формално логически имаме два случая:

първи случай: A е крайно. Тогава A е разрешимо, съгласно *Задача* 4.1.

втори случай: A е безкрайно. Тогава всъщност $A = \mathbb{N}$, което се вижда веднага от факта, че

$$n \in A \implies \forall m_{m\leq n}\ m \in A.$$

Наистина, нека m е произволно естествено число. Щом A е безкрайно, ще съществува число $n \geq m$, което е в A. Но тогава от горното наблюдение получаваме, че числата по-малки от n също ще са в A, и значи там ще е и произволното m, от което тръгнахме.

Видяхме, че и в двата възможни случая за A излезе, че A е разрешимо. Но кой е алгоритъмът, който го разрешава? Отговорът на този въпрос по никакъв начин не следва от разсъжденията по-горе, защото те не определят дали множеството A е крайно или безкрайно (ако е крайно — кои са елементите му). □

Задача 4.6. Нека h е едноместна рекурсивна функция, за която е изпълнено условието $h(n) \geq n$ за всяко n. Докажете, че множеството $A = \{h(0), h(1), \ldots\}$ е разрешимо.

Решение. От дефиницията на h имаме, че за всяко x:

$$x \in A \iff \exists n\, h(n) = x \iff \exists n_{n\leq \underbrace{h(n)}_{x}}\ \underbrace{h(n) = x}_{(n.x)\in G_h} \iff \exists n_{n\leq x}\ (n.x) \in G_h.$$

Понеже h е рекурсивна, нейната графика ще е разрешима, съгласно *Твърдение* 4.4. Сега прилагаме *Следствие* 4.1 и стигаме до извода, че A наистина е разрешимо. □

4.1.4 Характеризации на непразните разрешими подмножества на $\mathbb{N}$

По-надолу с $Range(f)$ ще означаваме множеството от стойностите на функцията f, с други думи

$$Range(f) \stackrel{\text{деф}}{=} \{\ y \mid \exists \bar{x}\ f(\bar{x}) \simeq y\ \}.$$

Когато f е едноместна и тотална, очевидно $Range(f) = \{f(0), f(1), \dots\}$. Ако за едно множество $A \subseteq \mathbb{N}$ е изпълнено

$$A = \{f(0), f(1), \dots\},$$

ще казваме, че f *изброява* елементите на A. Ако функцията f е строго растяща, тя изброява елементите на A в строго растящ ред. Ако f е нестрого растяща (т.е. $\forall x \forall y\ (x \leq y \Longrightarrow f(x) \leq f(y))$, тя изброява елементите на A в ненамаляващ ред.

Оказва се, че *безкрайните* разрешими множества можем да характеризираме като точно тези множества, които могат да се изброят в строго растящ ред.

Твърдение 4.5. Безкрайното множество от естествени числа A е разрешимо тогава и само тогава, когато съществува рекурсивна строго растяща функция h, такава че $A = \{h(0), h(1), \dots\}$.

Доказателство. Нека $A \subseteq \mathbb{N}$ е безкрайно и разрешимо. Ще конструираме рекурсивна функция h, която го изброява във възходящ ред. Всъщност функцията h с това свойство е единствена и тя се определя от следната рекурсивна схема:

$$\left|\begin{array}{l} h(0) = \mu z[z \in A] \\ h(n+1) = \mu z[z \in A\ \&\ z > h(n)]. \end{array}\right.$$

Да се убедим. Най-напред, тъй като A е безкрайно, h очевидно ще е тотална функция. От определението ѝ се вижда още, че $h(n+1) > h(n)$ за всяко n, т.е. h е строго растяща. Освен това по дефиниция $Range(h) \subseteq A$. Обратното включване също е вярно (съобразете го). Следователно $A = Range(h)$.

Остана да видим, че е h изчислима. За целта да препишем дефиницията ѝ така:

$$\left|\begin{array}{l} h(0) = \mu z[\chi_A(z) = 0] = a_0 \\ h(n+1) = \underbrace{\mu z[\chi_A(z) + \chi_{>}(z, h(n)) = 0]}_{H(n,h(n))}, \end{array}\right.$$

където $H(n,y) \overset{\text{деф}}{\simeq} \mu z[\chi_A(z)+\chi_>(z,y)=0]$ очевидно е изчислима. Тогава и h ще е изчислима, и понеже тя е тотална, значи общо е рекурсивна.

Обратно, нека $A=\{h(0),h(1),\dots\}$ за някоя рекурсивна строго растяща функция h. По определение

$$x\in A \iff \exists n\ h(n)=x.$$

Знаем, че множеството $G_h=\{(n,x)\mid h(n)=x\}$ е разрешимо, но как да ограничим квантора за съществуване пред него, за да можем да използваме *Следствие* 4.1?

Една горна граница за всяко n, такова че $h(n)=x$, очевидно се явява функцията $b(x)=\mu n[h(n)\geq x]$. Тази функция е тотална, защото A е безкрайно. Освен това тя е изчислима, значи общо b е рекурсивна. Сега вече множеството A ще е разрешимо, защото за него ще е вярно, че

$$x\in A \iff \exists n\ h(n)=x \iff \exists n_{n\leq b(x)}\ h(n)=x.$$

Разбира се, можехме и директно да съобразим, че

$$x\in A \iff h(b(x))=x,$$

откъдето веднага $\chi_A(x)=sg(|h(b(x))-x|)$ ще е рекурсивна. □

Да отбележим, че тази посока на твърдението можехме да получим и като следствие от *Задача* 4.6, защото когато h е строго растяща, със сигурност $h(n)\geq n$. Проверката е с индукция по n: за $n=0$ то се превръща в очевидното $h(0)\geq 0$, а допускайки, че $h(n)\geq n$ за някое n, за $n+1$ ще имаме:

$$h(n+1)>h(n) \overset{\text{и.х.}}{\geq} n.$$

Но $h(n+1)$ и n са естествени числа, тъй че от $h(n+1)>n$ със сигурност $h(n+1)\geq n+1$.

Следващото твърдение характеризира *непразните* разрешими множества от естествени числа: това са точно множествата, които могат да се изброят алгоритмично в ненамаляващ ред.

Твърдение 4.6. Непразното множество от естествени числа A е разрешимо тогава и само тогава, когато съществува рекурсивна нестрого растяща функция h, такава че $A=\{h(0),h(1),\dots\}$.

Доказателство. Нека $\emptyset\neq A\subseteq\mathbb{N}$ е разрешимо. Искаме да построим рекурсивна и нестрого растяща функция h, която да изброява неговите елементи. Тук не можем да разсъждаваме както в предишното твърдение, полагайки

$$\left|\begin{array}{l} h(0)=\mu z[z\in A] \\ h(n+1)=\mu z[z\in A\ \&\ z>h(n)], \end{array}\right.$$

защото ако A е крайно множество, h няма да е тотална функция. Затова решаваме да разгледаме поотделно случаите, в които A е крайно и безкрайно.

Случай 1: A е крайно. Тогава $A = \{a_0, \ldots, a_k\}$ за някое $k \geq 0$, като предполагаме, че $a_0 < \cdots < a_k$. Тогава можем да вземем следната функция h:

$$h(n) = \begin{cases} a_0, & \text{ако } n = 0 \\ . \quad . \quad . \quad . \quad . \\ a_k, & \text{ако } n \geq k. \end{cases}$$

Имаме $h(0) < \cdots < h(k-1) = h(k) = h(k+1) = \ldots$, т.е. h е нестрого растяща. Тя очевидно е рекурсивна и изброява елементите на A.

Случай 2: A е безкрайно. Но тогава можем да използваме предишното *Твърдение* 4.5 и да получим, че $A = Range(h)$ дори за *строго растяща* рекурсивна h.

Видяхме, че и в двата възможни случая за A съществува рекурсивна растяща h, която изброява елементите на това множество. Горното разсъждение формално беше коректно, но то не ни помага да *конструираме* функцията h, ако разполагаме с програма, пресмятаща характеристичната функция на A. Програмата за χ_A в общия случай не ни дава възможност да определим кой от двата случая — A е крайно или A е безкрайно — е налице. Това, разбира се, не означава, че не може да се подходи другояче към задачата. Наистина, оказва се, че е възможно да се намери обща функция h, без да се разглеждат случаите за A. Това ще формулираме като задача за ЕК, след като завършим доказателството на твърдението.

Сега да се насочим към обратната посока. Нека h е рекурсивна растяща функция, която изброява елементите на A в ненамаляващ ред, т.е. имаме

$$A = \{h(0), h(1), \ldots\}, \quad \text{където} \quad h(0) \leq h(1) \leq \ldots$$

Тук отново е удобно да разгледаме двата случая за мощността на A.

Случай 1: A е крайно. Вече се убедихме, че всяко крайно множество е разрешимо.

Случай 2: A е безкрайно. Тук можем да действаме по аналогия с доказателството на предишното твърдение. Разликата е, че сега h е *нестрого* растяща, но въпреки това, щом $Range(h) = A$ е безкрайно, функцията

$$b(x) = \mu n[h(n) \geq x]$$

ще е тотална и изчислима, т.е. рекурсивна. Тогава отново за всяко x:

$$x \in A \iff \exists n_{n \leq b(x)}\ h(n) = x.$$

Следователно A е разрешимо множество. $\square$

Задача 4.7. (задача за ЕК) Нека $\emptyset \neq A \subseteq \mathbb{N}$ е разрешимо. *Конструирайте* чрез χ_A рекурсивна нестрого растяща функция h, такава че $A = \{h(0), h(1), \dots\}$.

Да използваме характеризацията на безкрайните разрешими множества от по-горе, за да решим следната задача:

Задача 4.8. Нека A и B са безкрайни и разрешими множества от естествени числа. Докажете, че съществува рекурсивна биекция $h\colon A \longrightarrow B$. При какво условие за A и B тази функция h може да се разшири до рекурсивна биекция върху цялото $\mathbb{N}$?

Решение. От *Твърдение* 4.5 знаем, че за множествата A и B съществуват рекурсивни строго растящи функции g и h, такива че $Range(f) = A$ и $Range(g) = B$, т.е.

$$A = \{f(0), f(1), \dots\} \quad \text{и} \quad B = \{g(0), g(1), \dots\}.$$

Идеята ни е ясна — да дефинираме h така, че n-тият елемент на A да отива в n-тия елемент на B, т.е.

$$h(f(n)) = g(n).$$

Тогава очевидно $h\colon A \longrightarrow B$ ще бъде инективна и сюрективна. Върху точките извън $Range(f)$ нямаме изисквания за h.

Значи можем да положим

$$h(x) = \begin{cases} g(\mu n[f(n) = x]), & \text{ако } x \in Range(f) \\ 0, & \text{иначе.} \end{cases}$$

Ако $x \in Range(f)$, то съществува единствено $n\colon f(n) = x$, и тогава минимизацията $\mu n[f(n) = x]$ го намира, а $h(x) \overset{\text{деф}}{=} g(\mu n[f(n) = x])$ изпраща този n-ти елемент на A в n-тия елемент на B. Така осигуряваме, че $h(f(n)) = g(n)$ за всяко n. Тъй като предикатът $p(x) \iff x \in Range(f)$ е разрешим, то h ще е рекурсивна.

Ако искаме $h\colon \mathbb{N} \longrightarrow \mathbb{N}$ да е биективна, очевидно трябва допълненията на A и B да са с една и съща мощност, т.е. или и двете да са безкрайни, или да са крайни и да имат един и същ брой елементи. Довършете конструкцията на h за тези два случая, като за първия случай използвате, че $\bar{A}$ и $\bar{B}$ също са разрешими. □

Задача 4.9. (задача за ЕК) Нека f и g са едноместни рекурсивни функции, като g е биективна. Нека още $f(x) \geq g(x)$ за всяко $x \in \mathbb{N}$. Докажете, че $Range(f)$ е разрешимо множество.

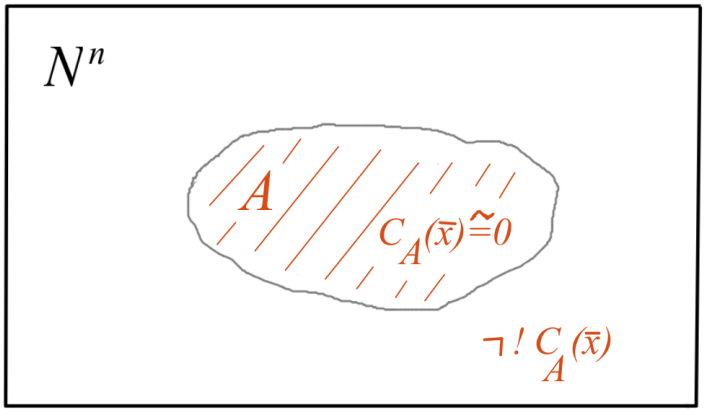
Задача 4.10. (задача за ЕК) Нека f и g са едноместни рекурсивни функции, като g е обратима и $Range(g)$ е разрешимо множество. Нека още $f(x) \geq x$ за всяко $x \in \mathbb{N}$. Докажете, че е разрешимо множеството $\{g(x) \mid x \in Range(f)\}$.

4.2 Полуразрешими множества

Интуитивно, едно множество $A \subseteq \mathbb{N}^n$ е *полуразрешимо*, ако има алгоритъм, който спира *точно* върху тези n-торки $\bar{x}$, които са от A. Какво точно връща, когато спре, е съвършено без значение. Обикновено се избира това да е "*да*", "*true*", "0" или "1", но може да е и всичко друго. Важното е да има някакъв резултат, който да ни сигнализира, че входът $\bar{x}$ е от A. За точките извън A алгоритъмът не завършва.

По този начин, образно казано, разполагаме само с *половината* от информацията за A, в случая — "позитивната" половина. Разбира се, със същият успех бихме могли да искаме алгоритъмът да спира само върху точките извън A, което ще е еквивалентно на това допълнението $\bar{A}$ да е полуразрешимо.

За да формализираме това понятие, отново разглеждаме подходяща числова функция, която — логично — се нарича *полухарактеристична* функция на A.



Определение 4.3. *Полухарактеристична функция* на множеството $A \subseteq \mathbb{N}^n$ наричаме функцията $C_A : \mathbb{N}^n \rightharpoonup \mathbb{N}$, която се дефинира по следния начин:

$$C_A(\bar{x}) \simeq \begin{cases} 0, & \text{ако } \bar{x} \in A \\ \neg!, & \text{ако } \bar{x} \notin A. \end{cases}$$

Определение 4.4. Казваме, че множеството $A \subseteq \mathbb{N}^n$ е *полуразрешимо*, ако неговата полухарактеристична функция C_A е изчислима.

Интуитивно е ясно, че ако за едно множество имаме разрешаваща процедура, ще имаме и полуразрешаваща. Да го докажем и формално:

Задача 4.11. Докажете, че всяко разрешимо множество е полуразрешимо.

Решение. Нека множеството $A \subseteq \mathbb{N}^n$ е разрешимо, т.е. характеристичната му функция χ_A е рекурсивна. За полухарактеристичната C_A имаме следната дефиниция с разглеждане на случаи:

$$C_A(\bar{x}) \simeq \begin{cases} \mathcal{O}(I_1^n(\bar{x})), & \text{ако } \chi_A(\bar{x}) = 0 \\ \emptyset^{(n)}(\bar{x}), & \text{ако } \chi_A(\bar{x}) \neq 0. \end{cases}$$

Тъй като χ_A е рекурсивна, съгласно *Твърдение* 1.11, C_A ще е изчислима.

Задачата можем да решим и по-кратко (без непременно това да означава и по-ясно $\ddot{\smile}$), като съобразим, че полухарактеристичната C_A се изразява чрез характеристичната χ_A по следния начин:

$$C_A(\bar{x}) \simeq \mu y[\chi_A(\bar{x}) = 0].$$

□

Дали е вярна и обратната посока на тази задача? Очаквано, не е. Нашият първи пример за полуразрешимо множество, което не е разрешимо, ще е множеството K, наречено на името на Стивън Клини, един от основоположниците на Теорията на изчислимостта (която тогава се е наричала Теория на рекурсията). Множеството K се дефинира по следния начин:

$$K \stackrel{\text{деф}}{=} \{\ x \mid !\varphi_x(x)\ \}.$$

По определение $!\varphi_x(x) \iff P_x(x)\downarrow$, значи можем да кажем, че

$$K = \{\ x \mid P_x(x) \downarrow\ \},$$

т.е. K се състои от кодовете на тези програми, които спират върху собствения си код.

Твърдение 4.7. Множеството K е полуразрешимо, но не е разрешимо.

Доказателство. За полухарактеристичната функция на K имаме следното представяне чрез универсалната функция за едноместните изчислими Φ_1: $C_K(x) \simeq \mathcal{O}(\Phi_1(x,x))$. Това се вижда от еквивалентностите:

$$C_K(x) \simeq 0 \iff x \in K \iff !\varphi_x(x) \iff !\Phi_1(x,x) \iff \mathcal{O}(\Phi_1(x,x)) \simeq 0 \quad \text{и}$$

$$\neg!C_K(x) \iff x \notin K \iff \neg!\varphi_x(x) \iff \neg!\Phi_1(x,x) \iff \neg!\mathcal{O}(\Phi_1(x,x)).$$

Тъй като универсалната функция Φ_1 е изчислима, то и C_K ще е изчислима и значи K е полуразрешимо.

Сега да допуснем, че K е разрешимо и да разгледаме функцията

$$f(x) \simeq \begin{cases} \neg!, & \text{ако } !\varphi_x(x) \\ 0, & \text{ако } \neg!\varphi_x(x). \end{cases}$$

Тя е изчислима, защото можем да я препишем като

$$f(x) \simeq \begin{cases} \emptyset^{(1)}(x), & \text{ако } x \in K \\ \mathcal{O}(x), & \text{ако } x \notin K \end{cases}$$

и да приложим отново *Твърдение* 1.11.

Щом f е изчислима, ще има поне едно a: $f = \varphi_a$. Тогава, в частност, би трябвало $f(a) \simeq \varphi_a(a)$, което очевидно не е възможно (f беше подбрана точно с цел да осуети това условно равенство). Полученото противоречие показва, че множеството K не е разрешимо. □

4.2.1 Еквивалентни характеризации на полуразрешимите множества

Дефиницията за полуразрешимост на множество, която въведохме по-горе, всъщност е една от няколкото възможни. Сега ще дадем още 4 дефиниции (или еквивалентни характеризации) на полуразрешимостта, всяка от които дава различна гледна точка към това понятие. Както ще наблюдаваме по-нататък, в различните ситуации се оказва по-удобно да използваме една или друга от тези еквивалентни дефиниции, а първоначалното *Определение* 4.4 ще използваме най-рядко.

Най-близко до определението с полухарактеристична функция е следното НДУ за полуразрешимост:

Твърдение 4.8. (Първо НДУ за полуразрешимост) Множеството $A \subseteq \mathbb{N}^n$ е полуразрешимо тогава и само тогава, когато съществува изчислима функция f, такава че

$$A \;=\; Dom(f).$$

Доказателство. Да отбележим, че условието $A = Dom(f)$, или все едно $A = \{\ \bar{x} \mid !f(\bar{x})\ \}$, може да се препише като еквивалентността

$$\bar{x} \in A \iff !f(\bar{x})$$

за всяко $\bar{x} \in \mathbb{N}^n$. Това означава, че програмата, която пресмята f, ще спира точно върху тези n-торки, които са от A (без значение с какъв резултат), което беше и нашата първоначална *интуитивна идея* за полуразрешимост.

Да съобразим, че тя отговаря точно на дефиницията за полуразрешимост. Проверката и на двете посоки е едноходова: ако A е полуразрешимо, то $f := C_A$ очевидно върши работа. Обратно, ако f е такава че $A = Dom(f)$, лесно се съобразява, че $C_A = \mathcal{O} \circ f$ и значи C_A е изчислима. □

Да отбележим, че ако малко по-нагоре разполагахме с това твърдение, полуразрешимостта на множеството K щеше да следва директно от представянето

$$K \stackrel{\text{деф}}{=} \{\ x \mid !\varphi_x(x)\ \} = \{\ x \mid !\underbrace{\Phi_1(x,x)}_{f(x)}\ \}$$

и факта, че функцията f е изчислима.

Следващото НДУ за полуразрешимост напомня на подобното НДУ *за разрешимост* (*Твърдение* 4.1).

Твърдение 4.9. (Второ НДУ за полуразрешимост) Множеството $A \subseteq \mathbb{N}^n$ е полуразрешимо тогава и само тогава, когато съществува изчислима функция f, такава че

$$A \ = \ \{\ \bar{x} \mid f(\bar{x}) \simeq 0\ \}.$$

Доказателство. Ако A е полуразрешимо, отново можем да вземем f да е полухарактеристичната функция на A. Тя има това допълнително свойство, че там, където не е 0, не е дефинирана.

Обратно, ако разполагаме с изчислима функция f, такава че

$$\bar{x} \in A \iff f(\bar{x}) \simeq 0,$$

за всяко $\bar{x} \in \mathbb{N}^n$, можем да твърдим, че C_A е изчислима, защото за нея имаме представянето:

$$C_A(\bar{x}) \simeq \mu y[f(\bar{x}) \simeq 0].$$

Да отбележим, че от това НДУ за точките извън A имаме следното:

$$\bar{x} \notin A \iff \neg!f(\bar{x}) \ \vee \ f(\bar{x}) > 0.$$

Това означава, че тези точки, които по идея са алгоритмично недостъпни за нас, изглежда, не са съвсем такива, защото ако $f(\bar{x}) > 0$, програмата, пресмятаща f ще спре и така ние ще знаем, че $\bar{x} \notin A$. Истината е, че нямаме никаква сигурност, че при $\bar{x} \notin A$ това непременно ще се случи, което на практика води до това, че A е полуразрешимо, както сочи и доказателството по-горе. □

Следващата характеризация на полуразрешимо множество A влиза в детайли от изчислението на C_A, което я прави много удобна за различни приложения.

Твърдение 4.10. (Трето НДУ за полуразрешимост) Множеството $A \subseteq \mathbb{N}^n$ е полуразрешимо тогава и само тогава, когато съществува $n+1$-местна *рекурсивна* функция ρ, такава че

$$A \ = \ \{\ \bar{x} \mid \exists t\ \rho(\bar{x}, t) = 0\ \}.$$

Забележка. От доказателството ще се види, че всъщност ρ може да се избере и *примитивно* рекурсивна.

Доказателство. Нека A е полуразрешимо множество и нека P е МНР програмата, която пресмята полухарактеристичната му функция C_A. Интуитивно можем да си мислим, че $\rho(\bar{x}, t)$ дава индикация за това дали на стъпка t от изчислението на P върху $\bar{x}$ тя все още работи или вече е спряла, по-точно. Наистина, нека дефинираме ρ така:

$$\rho(\bar{x}, t) \overset{\text{деф}}{=} \begin{cases} 0, & \text{ако } P \text{ спира върху } \bar{x} \text{ за } \leq t \text{ стъпки} \\ 1, & \text{иначе.} \end{cases}$$

За да видим, че ρ е примитивно рекурсивна, ще се възползваме от примитивно рекурсивната функция $Q_n(a, \bar{x}, t)$, която въведохме в началото на раздел 3.1.1. По определение, $Q_n(a, \bar{x}, t)$ дава кода на конфигурацията след t стъпки от работата на P_a върху $\bar{x}$. Значи имаме еквивалентността:

P_a спира върху $\bar{x}$ за $\leq t$ стъпки $\iff$ $Q_n(a, \bar{x}, t)$ е код на закл. конфигурация.

Да предположим, че програмата P, която пресмята C_A, има код a, т.е. $P = P_a$. Нека за определеност $P_a \colon I_0, \ldots, I_k$. Тогава

$Q_n(a, \bar{x}, t)$ е код на заключителна конфигурация $\iff$ $(Q_n(a, \bar{x}, t))_0 > k$.

Да видим, че тази функция ρ изпълнява условието от твърдението. Наистина, за произволно $\bar{x} \in \mathbb{N}^n$ ще имаме:

$$\bar{x} \in A \iff !C_A(\bar{x}) \iff P_a(\bar{x})\downarrow \iff \exists t\, (Q_n(a, \bar{x}, t))_0 > k \iff \exists t\, \rho(\bar{x}, t) = 0.$$

Освен това ρ е примитивно рекурсивна, защото очевидно

$$\rho(\bar{x}, t) \;=\; \chi_{>}((Q_n(a, \bar{x}, t))_0, k).$$

Да отбележим, че функцията ρ можем да конструираме и другояче. Доказателството, че тази друга ρ върши работа е по-кратко, но и по-неинтуитивно. То минава през теоремата за нормален вид на Клини, според която $\varphi_a^{(n)}$ има следния нормален вид:

$$\varphi_a^{(n)}(\bar{x}) \simeq L(\mu z[T_n(a, \bar{x}, z) = 0]),$$

където функцията T_n е примитивно рекурсивна.
В нашият случай $\varphi_a^{(n)} = C_A$, т.е. имаме

$$C_A(\bar{x}) \simeq L(\mu z[T_n(a, \bar{x}, z) = 0]).$$

Тогава

$$\bar{x} \in A \iff !C_A(\bar{x}) \iff \exists z \underbrace{T_n(a, \bar{x}, z)}_{\rho(\bar{x}, z)} = 0$$

и значи в качеството на ρ можем да вземем функцията $\lambda\bar{x}, z.T_n(a, \bar{x}, z)$.

Сега обратно, нека за множеството A разполагаме с рекурсивна функция ρ, такава че за всяко $\bar{x} \in \mathbb{N}^n$:

$$\bar{x} \in A \iff \exists t\ \rho(\bar{x}, t) = 0.$$

Как да разберем дали $\bar{x}$ е в A? Ами пресмятаме последователно

$$\rho(\bar{x}, 0),\ \rho(\bar{x}, 1),\ \rho(\bar{x}, 2),\ \ldots$$

докато (ако изобщо) се натъкнем на t, за което $\rho(\bar{x}, t) = 0$. Ако $\bar{x} \in A$, рано или късно ще намерим такова t; ако $\bar{x} \notin A$, такова t не съществува, но ние няма как да разберем това, докато пресмятаме още и още поредни стойности на $\rho(\bar{x}, t)$. Така се получава точно ефектът на полуразрешимостта — позитивната информация за A ни е достъпна алгоритмично, а за негативната не можем да кажем нищо.

След като се убедихме, че неформално твърдението в тази посока е вярно, да напишем и формалното доказателство. То е съвсем кратко: просто за всяко $\bar{x} \in \mathbb{N}^n$:

$$\bar{x} \in A \iff \exists t\ \rho(\bar{x}, t) = 0 \iff \underbrace{!\mu t[\rho(\bar{x}, t) = 0]}_{f(\bar{x})}.$$

С други думи, $A = Dom(f)$, като функцията f очевидно е изчислима, откъдето съгласно *Твърдение* 4.9, A ще е полуразрешимо. Да отбележим, че тук съществено използвахме, че ρ е тотална. Ако ρ не е тотална, в общия случай не е вярно, че $\exists t\ \rho(\bar{x}, t) = 0 \implies !\mu t[\rho(\bar{x}, t) = 0]$. □

От това твърдение получаваме следната връзка между разрешимите и полуразрешимите множества:

Следствие 4.2. Множеството $A \subseteq \mathbb{N}^n$ е полуразрешимо тогава и само тогава, когато съществува разрешимо множество $B \subseteq \mathbb{N}^{n+1}$, такова че

$$A = \{\ \bar{x} \mid \exists y\ (\bar{x}, y) \in B\ \}.$$

Доказателство. Ако A е полуразрешимо, съгласно горното твърдение ще съществува рекурсивна функция ρ, такава че $A = \{\ \bar{x} \mid \exists t\ \rho(\bar{x}, t) = 0\ \}$. Ясно е кое трябва да е множеството B — това са всички $(\bar{x}, t) : \rho(\bar{x}, t) = 0$. За характеристичната функция на B имаме

$$\chi_B(\bar{x}, t) = sg(\rho(\bar{x}, t))$$

и значи множеството B е разрешимо.

Обратно, ако A е от вида $\{\ \bar{x} \mid \exists y\ (\bar{x}, y) \in B\ \}$ за някое разрешимо B, то A можем да представим като

$$\{\ \bar{x} \mid \exists y\ \chi_B(\bar{x}, y) = 0\},$$

където функцията χ_B е рекурсивна, и значи A е полуразрешимо. $\square$

В предишния раздел видяхме, че ограничените квантори запазват разрешимостта (*Твърдение* 4.2). Ако кванторите са *неограничени*, това вече не е така. Да го съобразим, като използваме горната характеризация на полуразрешимите множества чрез разрешими.

Задача 4.12. Докажете, че кванторите за съществуване и всеобщност не запазват разрешимостта.

Решение. Да забележим, че ако A е разрешимо, то множеството

$$\exists y A \stackrel{\text{деф}}{=} \{\ \bar{x} \mid \exists y\ (\bar{x}, y) \in A\ \}$$

ще е полуразрешимо със сигурност. Ние искаме то да *не е* разрешимо, значи $\exists y A$ трябва да е полуразрешимо, но неразрешимо. Вече знаем едно такова множество — множеството K (*Твърдение* 4.7). Прилагаме горното *Следствие* 4.2 към полуразрешимото K и получаваме, че за някое разрешимо $A \subseteq \mathbb{N}^2$ ще е изпълнено

$$K \ = \ \{\ x \mid \exists y\ (x, y) \in A\ \}.$$

Така конструирахме пример за разрешимо множество A, такова че $\exists y A$ вече не е разрешимо.

За да се убедим, че и кванторът за всеобщност не запазва разрешимостта, тръгваме от допълнението $\bar{A}$ на горното множество A, което също е разрешимо. Тогава $\overline{K} = \forall y \bar{A}$: наистина, за произволно $x \in \mathbb{N}$ имаме, съгласно законите на Де Морган:

$$x \in \overline{K} \iff \neg(x \in K) \iff \neg(\exists y\ (x, y) \in A) \iff \forall y\ (x, y) \in \bar{A}.$$

Така получихме, че $\bar{A}$ е разрешимо, докато $\overline{K} = \forall y \bar{A}$ не е разрешимо, защото ако $\overline{K}$ беше разрешимо, то трябваше и K да е такова, а то не е. (Казано в скобки — $\overline{K}$ дори не е полуразрешимо, както ще забележим по-нататък. Това означава, че кванторът $\forall$, за разлика от квантора $\exists$, вече извежда и от класа на полуразрешимите множества.) $\square$

Последната характеризация на полуразрешимите множества се отнася за непразни множества от естествени числа (т.е. за подмножества на $\mathbb{N}$). Тя казва, че едно такова множество е полуразрешимо точно тогава, когато елементите му могат да се изброят алгоритмично, евентуално с повторения. Оттук идва и другото име на полуразрешимите множества — *рекурсивно изброими (recursively enumerable)*.

Твърдение 4.11. (Четвърто НДУ за полуразрешимост) Нека $\emptyset \neq A \subseteq \mathbb{N}$. Множеството A е полуразрешимо тогава и само тогава, когато съществува рекурсивна функция h, която го изброява, т.е. за която е изпълнено

$$A \ = \ \{h(0), h(1), \dots\}.$$

Доказателство. Ще използваме предишното НДУ и в двете посоки на доказателството.

Нека A е полуразрешимо. Съгласно *Твърдение* 4.10, можем да си представяме A във вида $A = \{ x \mid \exists t\ \rho(x,t) = 0 \}$ за някоя рекурсивна функция ρ. Ще използваме тази функция, за да конструираме h.

Понеже $A \neq \emptyset$, съществува $x_0 \in A$. (Ако искаме да го намерим алгоритмично по дадената ρ, можем да вземем $x_0 = L(\mu n[\rho(L(n), R(n)) = 0])$.)

Сега дефинираме h като:

$$h(n) = \begin{cases} L(n), & \text{ако } \rho(L(n), R(n)) = 0 \\ x_0, & \text{иначе.} \end{cases}$$

Ясно е, че h е рекурсивна. Всъщност бихме могли да я получим и примитивно рекурсивна, ако тръгнем от примитивно рекурсивна ρ. За приложенията обикновено е важно само, че h е *тотална* изчислима функция.

Да се убедим, че $Range(h) = A$. Наистина, ако $x \in Range(h)$, то $x = x_0$ или $x = L(n)$ за някое n, такова че $\rho(L(n), R(n)) = 0$. В първия случай $x_0 \in A$ по дефиниция. Във втория случай имаме $\rho(x, R(n)) = 0$, което означава, че $x \in A$, т.е. дотук $Range(h) \subseteq A$.

Обратно, ако $x \in A$, то ще съществува t, такова че $\rho(x,t) = 0$. Нека $n = \Pi(x,t)$. Понеже $L(n) = x$ и $R(n) = t$, все едно имаме $\rho(L(n), R(n)) = 0$. Тогава

$$h(n) \overset{\text{деф}}{=} L(n) = x$$

и значи $x \in Range(h)$, с което показахме и обратното включване $A \subseteq Range(h)$.

Да докажем и обратната посока на твърдението. За целта нека $A = Range(h)$ за някоя рекурсивна функция h. Тогава

$$x \in A \iff \exists n\ h(n) = x \iff \exists n\ |h(n) - x| = 0 \iff \exists n\ \rho(x,n) = 0,$$

където с ρ сме означили рекурсивната функция $|h(n) - x|$. Не ни остава нищо друго, освен отново да приложим *Твърдение* 4.10. □

Получихме две основни характеризации на полуразрешимите множества:

— като множества, за които има *полуразрешаващ* алгоритъм;

— като множества, чиито елементи могат *да се генерират* от алгоритъм.

Да отбележим, че подобно явление се наблюдава и в другите класове от Йерархията на Чомски.

Автоматните езици могат да се характеризират като тези езици, които:
— *се разпознават* от краен автомат;
— *се генерират* от автоматна граматика.

По подобен начин, *безконтекстните езици*:
— *се разпознават* от стеков автомат;
— *се генерират* от безконтекстна граматика.

И накрая, *контекстно-зависимите (контекстните) езици* са точно тези, които:
— *се разпознават* от линейно ограничен автомат (LBA);
— *се генерират* от контекстна граматика.

При изброяването на елементите на A от функцията h, която построихме по-горе, неизбежно има повторения. Дали е възможно това изброяване да е без повторения (ако A е безкрайно, разбира се)? Оказва се, че е възможно.

Задача 4.13. Нека $A \subseteq \mathbb{N}$ е безкрайно. Докажете, че множеството A е полуразрешимо тогава и само тогава, когато съществува рекурсивна и инективна функция g, такава че $Range(g) = A$, с други думи, g изброява елементите на A без повторение.

Решение. Обратната посока следва директно от *Твърдение* 4.11, затова да се съсредоточим върху правата посока.

Нека A е безкрайно разрешимо множество от естествени числа. Ще конструираме функцията g, като стъпим на функцията h от горното *Твърдение* 4.11, която изброява A евентуално с повторения.

Ще строим g с пълна рекурсия. За стойността на g в началната точка $x = 0$ можем да вземем

$$g(0) \;=\; h(0).$$

Да приемем, че сме дефинирали $g(0), \ldots, g(n)$. Идеята ни е да намерим първото $k : h(k) \neq g(0), \ldots, h(k) \neq g(n)$ и да положим $g(n+1) \stackrel{\text{деф}}{=} h(k)$. Да го напишем формално:

$$\begin{aligned} g(n+1) &= h(\mu k[h(k) \neq g(0) \ \& \ \ldots \& \ h(k) \neq g(n)]) \\ &= h(\mu k[\chi_{\neq}(h(k), g(0)) = 0 \ \& \ \ldots \ \& \ \chi_{\neq}(h(k), g(n)) = 0]) \\ &= h(\mu k[\sum_{i \leq n} \chi_{\neq}(h(k), g(i)) = 0]). \end{aligned}$$

Така получаваме следната схема с пълна рекурсия за g:

$$\left| \begin{array}{lcl} g(0) & = & h(0) \\ g(n+1) & = & h(\mu k[\sum_{i \leq n} \chi_{\neq}(h(k), g(i)) = 0]), \end{array} \right.$$

откъдето се вижда, че g е изчислима. Освен това тя е тотална, понеже A е безкрайно, и значи общо е рекурсивна.

Да се убедим, че g е инективна. Наистина, да допуснем, че съществуват m и n, такива че $g(m) = g(n)$. Без ограничение на общността можем да считаме, че $m < n$. Но по дефиниция $g(n)$ е различна от $g(k)$ за всички $k < n$, в частност, $g(n) \neq g(m)$ — противоречие.

От дефиницията на g се вижда, че $Range(g) \subseteq Range(h)$. Лесно се съобразява, че е вярно ии обратното (убедете се сами), откъдето $Range(g) = Range(h)$. Но $Range(h) = A$ и следователно $Range(g) = A$. □

4.2.2 Основни факти за полуразрешимите множества

Ще покажем някои най-важни свойства на полуразрешими множества, които ще са ни необходими при доказване на следващи твърдения, както и в задачите.

Твърдение 4.12. (Обединение, сечение и декартово произведение запазват полуразрешимостта)

1) Нека $A \subseteq \mathbb{N}^n$ и $B \subseteq \mathbb{N}^n$ са полуразрешими множества. Тогава са полуразрешими и $A \cap B$ и $A \cup B$.

2) Ако $A \subseteq \mathbb{N}^n$ и $B \subseteq \mathbb{N}^k$ са полуразрешими, то е полуразрешимо и тяхното декартово произведение $A \times B$.

Забележка. Допълнението на полуразрешимо множество невинаги е полуразрешимо. Контрапример ще можем да дадем по-нататък.

Доказателство. 1) За сечението на A и B разсъждаваме както при разрешимите множества. Имаме

$$\begin{aligned} \bar{x} \in A \cap B &\iff \bar{x} \in A \ \& \ \bar{x} \in B \iff C_A(\bar{x}) \simeq 0 \ \& \ C_B(\bar{x}) \simeq 0 \\ &\iff C_A(\bar{x}) + C_B(\bar{x}) \simeq 0. \end{aligned}$$

Следователно $C_{A\cap B}(\bar{x}) \simeq C_A(\bar{x}) + C_B(\bar{x})$. (Със същия успех бихме могли да вземем $C_{A\cap B}(\bar{x}) \simeq C_A(\bar{x}).C_B(\bar{x})$.)

При обединението, обаче, има проблем. Вече не можем да твърдим, както беше при характеристичните функции, че

$$C_A(\bar{x}) \simeq 0 \ \vee \ C_B(\bar{x}) \simeq 0 \iff C_A(\bar{x}).C_B(\bar{x}) \simeq 0.$$

Това е защото полухарактеристичните функции са *частични* функции. В случая, изразът $C_A(\bar{x}).C_B(\bar{x})$ е дефиниран когато са дефинирани *едновременно* $C_A(\bar{x})$ и $C_B(\bar{x})$, т.е. когато $\bar{x} \in A \cap B$.

За нашите цели се оказва удобно да използваме НДУ за полуразрешимост от *Твърдение* 4.10. Съгласно това твърдение, за полуразрешимите множества A и B ще съществуват рекурсивни функции ρ_1 и ρ_2, такива че за всяко $\bar{x} \in \mathbb{N}^n$:

$$\bar{x} \in A \iff \exists t \ \rho_1(\bar{x}, t) = 0 \quad \text{и} \quad \bar{x} \in B \iff \exists t \ \rho_2(\bar{x}, t) = 0.$$

Тогава

$$\begin{aligned} \bar{x} \in A \cup B &\iff \bar{x} \in A \ \vee \ \bar{x} \in B \iff \exists t \ \rho_1(\bar{x}, t) = 0 \ \vee \ \exists t \ \rho_2(\bar{x}, t) = 0 \\ &\iff \exists t \ \underbrace{\rho_1(\bar{x}, t).\rho_2(\bar{x}, t)}_{\rho(\bar{x}, t)} = 0. \end{aligned}$$

Понеже функцията ρ е рекурсивна, $A \cup B$ ще е полуразрешимо.

2) От дефиницията на $A \times B$ имаме, че за произволни $\bar{x} \in \mathbb{N}^n$ и $\bar{y} \in \mathbb{N}^k$:

$$\begin{aligned} (\bar{x}, \bar{y}) \in A \times B &\iff \bar{x} \in A \ \& \ \bar{y} \in B \iff C_A(\bar{x}) \simeq 0 \ \& \ C_B(\bar{y}) \simeq 0 \\ &\iff C_A(\bar{x}) + C_B(\bar{y}) \simeq 0. \end{aligned}$$

Така за $C_{A\times B}$ ще е изпълнено $C_{A\times B}(\bar{x}, \bar{y}) \simeq C_A(\bar{x}) + C_B(\bar{y})$ и следователно $A \times B$ е полуразрешимо. □

От това твърдение лесно следва, че обединението, сечението и декартовото произведение на *краен брой* полуразрешими множества е полуразрешимо също. Убедете се сами, че е така.

Следващото твърдение показва, че кванторът за съществуване запазва полуразрешимостта и е известно като *Теорема за проекцията*.

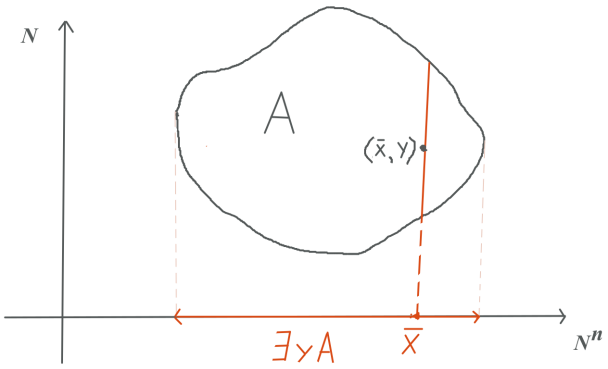
Твърдение 4.13. (Теорема за проекцията) Нека множеството $A \subseteq \mathbb{N}^{n+1}$ е полуразрешимо. Тогава е полуразрешимо и множеството

$$B = \{ \ \bar{x} \mid \exists y \ (\bar{x}, y) \in A \ \}.$$

Доказателство. Отново е подходящо да използваме НДУ от *Твърдение* 4.10, според което за полуразрешимото множество $A \subseteq \mathbb{N}^{n+1}$ съществува рекурсивна функция ρ, такава че

$$(\bar{x}, y) \in A \iff \exists t \ \rho(\bar{x}, y, t) = 0.$$

Картинката по-долу обяснява откъде идва името на това твърдение — множеството $\exists y A$ се явява проекция на множеството A върху оста $\mathbb{N}^n$.



За всяко $\bar{x} \in \mathbb{N}^n$ ще имаме, съгласно избора на ρ:

$$\bar{x} \in B \iff \exists y\ (\bar{x}, y) \in A \iff \exists y \exists t\ \rho(\bar{x}, y, t) = 0.$$

Двата квантора за съществуване можем да заменим с един: твърдим, че е в сила еквивалентността:

$$\exists y \exists t\ \rho(\bar{x}, y, t) = 0 \iff \exists z\ \rho(\bar{x}, L(z), R(z)) = 0.$$

Наистина, ако $\exists y \exists t\ \rho(\bar{x}, y, t) = 0$, то $z \stackrel{\text{деф}}{=} \Pi(y, t)$ удовлетворява дясната страна на тази еквивалентност. Обратно, ако $\rho(\bar{x}, L(z), R(z)) = 0$ за някое z, то за $y \stackrel{\text{деф}}{=} L(z)$ и $t \stackrel{\text{деф}}{=} R(z)$ ще е в сила лявата страна на еквивалентността. Така получихме, че за всяко $\bar{x} \in \mathbb{N}^n$

$$\bar{x} \in B \iff \exists z \underbrace{\rho(\bar{x}, L(z), R(z))}_{\rho'(\bar{x}, z)} = 0,$$

и понеже ρ' е рекурсивна, то B е полуразрешимо. □

От това твърдение веднага получаваме, че двата *ограничени* квантора запазват полуразрешимостта.

Твърдение 4.14. (Ограничените квантори запазват полуразрешимостта) Нека A е полуразрешимо. Тогава са полуразрешими и множествата

$$\begin{aligned} B &= \{(\bar{x}, y) \mid \exists z_{z \leq y}\ (\bar{x}, z) \in A\} \quad \text{и} \\ D &= \{(\bar{x}, y) \mid \forall z_{z \leq y}\ (\bar{x}, z) \in A\}. \end{aligned}$$

Доказателство. Условието за принадлежност към B можем да препишем така:

$$(\bar{x},y)\in B \iff \exists z_{z\le y}\ (\bar{x},z)\in A \iff \exists z(z\le y\ \&\ (\bar{x},z)\in A)$$
$$\iff \exists z\ \underbrace{\chi_{\le}(z,y)+C_A(\bar{x},z)}_{f(\bar{x},y,z)}\simeq 0.$$

Да означим:

$$B^* \ = \ \{(\bar{x},y,z)\mid f(\bar{x},y,z)\simeq 0\ \}.$$

Ясно е, че множеството B^* е полуразрешимо. За нашето B ще имаме $B=\{(\bar{x},y)\mid \exists z\ (\bar{x},y,z)\in B^*\}$, т.е. B се получава с прилагане на квантор за съществуване пред B^* и по теоремата за проекцията B ще е полуразрешимо.

За да покажем че и другият ограничен квантор запазва полуразрешимостта, разсъждаваме както при аналогичното *Твърдение* 4.2 за разрешимите множества:

$$(\bar{x},y)\in D \iff (\bar{x},0)\in A\ \&\ \ldots\ \&\ (\bar{x},y)\in A$$
$$\iff C_A(\bar{x},0)\simeq 0\ \&\ \ldots\ \&\ C_A(\bar{x},y)\simeq 0 \iff \sum_{z\le y} C_A(\bar{x},z)\simeq 0.$$

Тогава $C_D(\bar{x},y)\simeq \sum\limits_{z\le y} C_A(\bar{x},z)$ и следователно множеството D е полуразрешимо. □

Това твърдение обобщаваме по обичайния начин за произволна рекурсивна граница $b(\bar{x})$:

Следствие 4.3. Нека A е полуразрешимо множество, а $b(\bar{x})$ е рекурсивна функция. Тогава са полуразрешими и следните множества:

$$B^*=\{\bar{x}\mid \exists z_{z\le b(\bar{x})}\ (\bar{x},z)\in A\}\quad \text{и}$$
$$D^*=\{\bar{x}\mid \forall z_{z\le b(\bar{x})}\ (\bar{x},z)\in A\}.$$

Доказателство. Полухарактеристичните функции на B^* и D^* преписваме чрез полухарактеристичните функции на множествата B и D от горното *Твърдение* 4.14 по следния начин:

$$C_{B^*}(\bar{x})\simeq C_B(\bar{x},b(\bar{x}))\quad \text{и}\quad C_{D^*}(\bar{x})\simeq C_D(\bar{x},b(\bar{x})).$$

□

Твърдение 4.15. (Образът и първообразът на полуразрешимо множество чрез изчислима функция са полуразрешими) Нека $A\subseteq\mathbb{N}$ е полуразрешимо множество, а f е едноместна изчислима функция. Тогава са полуразрешими и множествата

$$f(A) \ = \ \{\ y\mid \exists x\in A: f(x)\simeq y\ \}\quad \text{и}$$
$$f^{-1}(A) \ = \ \{\ x\mid !f(x)\ \&\ f(x)\in A\ \}.$$

Забележка. Ако f е тотална функция, образът и първообразът на A можем да представим по-кратко така:

$$f(A) \;=\; \{f(x) \mid x \in A\ \} \quad \text{и} \quad f^{-1}(A) \;=\; \{x \mid f(x) \in A\ \}.$$

Доказателство. По дефиниция

$$C_{f^{-1}(A)}(x) \simeq 0 \iff x \in f^{-1}(A) \iff !f(x)\ \&\ f(x) \in A \iff C_A(f(x)) \simeq 0,$$

откъдето $C_{f^{-1}(A)} = C_A \circ f$ е изчислима като композиция на изчислими.

За $f(A)$ разсъждаваме така:

$$\begin{aligned} y \in f(A) &\iff \exists x(x \in A\ \&\ f(x) \simeq y) \iff \exists x(C_A(x) \simeq 0\ \&\ |f(x) - y| \simeq 0) \\ &\iff \exists x(\underbrace{C_A(x) + |f(x) - y|}_{\{(x,y)\ \mid\ (x,y) \in B\}} \simeq 0). \end{aligned}$$

Тъй като функцията $g(x,y) \overset{\text{деф}}{\simeq} C_A(x)+|f(x)-y|$ е изчислима, то множеството B е полуразрешимо, съгласно *Твърдение* 4.9. Тогава по теоремата за проекцията $f(A) = \exists x B$ също ще е полуразрешимо. □

Друг начин да покажем, че $f(A)$ е полуразрешимо, е като съобразим, че

$$f(A) \;=\; \exists x((A \times \mathbb{N}) \;\cap\; G_f)$$

и приложим твърдението, което следва.

Това твърдение дава връзка между изчислимостта на дадена функция и полуразрешимостта на нейната графика. То е известно като *Теорема за графиката.*

Твърдение 4.16. (Теорема за графиката) Функцията f е изчислима тогава и само тогава, когато графиката ѝ е полуразрешимо множество.

Доказателство. Правата посока е ясна: за всяко $\bar{x} \in \mathbb{N}^n$ и $y \in \mathbb{N}$ е изпълнено:

$$(\bar{x}, y) \in G_f \iff f(\bar{x}) \simeq y \iff \underbrace{|f(\bar{x}) - y|}_{g(\bar{x},y)} \simeq 0.$$

Тъй като функцията g е изчислима, съгласно *Твърдение* 4.9, множеството G_f ще е полуразрешимо.

Сега нека си представим, че разполагаме с алгоритъм, пресмятащ полухарактеристичната функция на G_f. Как да пресметнем f? Имаме, че

$$f(\bar{x}) \simeq min\{\ y \mid (\bar{x}, y) \in G_f\ \} \simeq min\{\ y \mid C_{G_f}(\bar{x}, y) \simeq 0\ \}.$$

Вече знаем, че

$$min\{\ y \mid C_{G_f}(\bar{x}, y) \simeq 0\ \} \;\neq\; \mu y[C_{G_f}(\bar{x}, y) \simeq 0].$$

Следователно не можем да представим $f(\bar{x})$ като $\mu y[C_{G_f}(\bar{x}, y) \simeq 0]$.

Затова ще подходим по-внимателно. Ясно е, че трябва да използваме твърдение, което представя G_f чрез *тотална* функция. За целта отново можем да използваме *Твърдение* 4.10, според което за някоя рекурсивна функция ρ ще е вярно, че за всяко $(\bar{x}, y) \in \mathbb{N}^{n+1}$:

$$(\bar{x}, y) \in G_f \iff \exists t\ \rho(\bar{x}, y, t) = 0.$$

Да съобразим, че

$$\exists t\ \rho(\bar{x}, y, t) = 0 \iff \exists z\ \rho(\bar{x}, L(z), R(z)) = 0\ \&\ L(z) = y.$$

Това е лесно: за правата посока полагаме $z := \Pi(y, t)$ а за обратната — $t := R(z)$.

От горните две еквивалентности получаваме

$$f(\bar{x}) \simeq y \iff \exists z\ \rho(\bar{x}, L(z), R(z)) = 0\ \&\ L(z) = y. \tag{4.1}$$

Твърдим, че за f имаме представянето:

$$f(\bar{x}) \simeq \underbrace{L(\mu z[\rho(\bar{x}, L(z), R(z)) = 0])}_{g(\bar{x})}.$$

Наистина, да означим функцията вдясно с g. Трябва да покажем, че $f = g$.

Да видим най-напред включването $g \subseteq f$. За целта да приемем, че $g(\bar{x}) \simeq y$. Искаме да покажем, че и $f(\bar{x}) \simeq y$.

От $g(\bar{x}) \simeq y$, в частност, $!g(\bar{x})$, което означава, че съществува най-малко z, такова че $\rho(\bar{x}, L(z), R(z)) = 0$. Тогава $g(\bar{x})$ ще е равна на $L(z)$, но ние имаме, че $g(\bar{x}) \simeq y$, откъдето достигаме до извода, че $L(z) = y$. Да резюмираме: получихме общо

$$\rho(\bar{x}, L(z), R(z)) = 0\ \&\ L(z) = y,$$

и това е точно условието в дясната част на еквивалентността (4.1). Следователно $f(\bar{x}) \simeq y$.

За обратното включване $f \subseteq g$ ще се възползваме за пореден път от *Задача* 1.1, според която е достатъчно да покажем по-слабото условие $Dom(f) \subseteq Dom(g)$. Наистина, нека да вземем произволно $\bar{x} \in Dom(f)$. Тогава за някое y ще имаме $f(\bar{x}) \simeq y$. В такъв случай, съгласно (4.1), трябва да съществува z, за което $\rho(\bar{x}, L(z), R(z)) = 0$. Но това означава, че $\mu z[\rho(\bar{x}, L(z), R(z)) = 0]$ е дефинирана и следователно $g(\bar{x})$ е дефинирана, т.е. $\bar{x} \in Dom(g)$. Получихме, че за произволното $\bar{x} \in \mathbb{N}^n$ е в сила

$$\bar{x} \in Dom(f) \implies \bar{x} \in Dom(g),$$

което означава, че $Dom(f) \subseteq Dom(g)$. С това приключва доказателството на равенството на f и g. Но функцията g очевидно е изчислима, откъдето финално получаваме, че и f е изчислима. $\square$

Като непосредствено следствие от теоремата за графиката получаваме, че изчислимостта се запазва при вземане на обратна функция.

Следствие 4.4. Нека f е едноместна изчислима и обратима функция. Тогава обратната ѝ функция f^{-1} също е изчислима.

Доказателство. По определение

$$f^{-1}(y) \simeq x \iff f(x) \simeq y.$$

Следователно $G_{f^{-1}} = \{\ (y,x) \mid (x,y) \in G_f\ \}$, откъдето за характеристичните функции на двете графики ще имаме следната връзка:

$$C_{G_{f^{-1}}}(y,x) \simeq C_{G_f}(x,y).$$

Щом f е изчислима, по теоремата за графиката множеството G_f ще е полуразрешимо, което по дефиниция означава, че C_{G_f} е изчислима. Следователно и $C_{G_{f^{-1}}}$ ще е изчислима, а оттук отново по теоремата за графиката и функцията f^{-1} ще е изчислима. $\square$

Ще завършим този раздел с още едно полезно твърдение, което обобщава дефиницията с разглеждане на случаи от раздел 1.6.1.

Твърдение 4.17. (Операцията "case" запазва изчислимостта) Нека $f_1, \ldots, f_k$ са n-местни функции, а $A_1, \ldots, A_k$ са две по две непресичащи се подмножества на $\mathbb{N}^n$. Да дефинираме функцията $g : \mathbb{N}^n \rightharpoonup \mathbb{N}$ както следва:

$$g(\bar{x}) \simeq \begin{cases} f_1(\bar{x}), & \text{ако } \bar{x} \in A_1 \\ \ldots\ldots\ldots\ldots\ldots & \ldots\ldots\ldots \\ f_k(\bar{x}), & \text{ако } \bar{x} \in A_k \\ \neg!, & \text{в останалите случаи.} \end{cases}$$

Твърдим, че ако $f_1, \ldots, f_k$ са изчислими, а $A_1, \ldots, A_k$ са полуразрешими, то и функцията g е изчислима.

Доказателство. Лесно се съобразява, че за графиката на g е изпълнено

$$G_g \ = \ G_{f_1} \cap (A_1 \times \mathbb{N}) \ \cup \ \ldots \ \cup \ G_{f_k} \cap (A_k \times \mathbb{N}).$$

Според *Твърдение* 4.12, графиката на g е полуразрешима, откъдето по теоремата за графиката ще имаме, че g е изчислима. $\square$

Най-типичният частен случай на горното твърдение, с който ще се сблъскваме в бъдеще, е при $k = 1$. Той изглежда така:

Следствие 4.5. Ако f е n-местна изчислима функция, а $A \subseteq \mathbb{N}^n$ е полуразрешимо множество, то е изчислима и функцията

$$g(\bar{x}) \simeq \begin{cases} f(\bar{x}), & \text{ако } \bar{x} \in A \\ \neg!, & \text{иначе.} \end{cases}$$

Само да отбележим, че в този по-прост случай имаме и по-просто доказателство — g можем да представим например така: $g(\bar{x}) \simeq C_A(\bar{x}) + f(\bar{x})$.

4.2.3 Още задачи за полуразрешими множества

Да започнем с едно типично полуразрешимо множество, свързано със *спор-проблема (the halting problem)* за МНР.

Задача 4.14. Докажете, че множеството

$$HP \stackrel{\text{деф}}{=} \{\ (a,x) \mid P_a \text{ спира върху } x\ \}$$

е полуразрешимо.

Решение. По определение

$$P_a \text{ спира върху } x \iff !\varphi_a(x) \iff !\Phi_1(a,x).$$

Следователно множеството HP се явява точно дефиниционната област на универсалната функция Φ_1, която е изчислима, и съгласно първото НДУ за полуразрешимост от *Твърдение* 4.8, HP ще е полуразрешимо. В следващия раздел ще покажем, че множеството HP, подобно на K, не е разрешимо. □

Да решим задачите за директна сума и директно произведение от раздел 4.1.3, този път във версиите за полуразрешими множества.

Задача 4.15. Докажете, че $A \subseteq \mathbb{N}$ и $B \subseteq \mathbb{N}$ са полуразрешими тогава и само тогава, когато е полуразрешима тяхната директна сума

$$A \oplus B \ = \ \{2x \mid x \in A\} \cup \{2x+1 \mid x \in B\}.$$

Решение. Нека A и B са полуразрешими. Лесно се вижда, че полухарактеристичната функция на $A \oplus B$ се изразява чрез C_A и C_B по следния начин:

$$C_{A\oplus B}(z) \simeq \begin{cases} C_A(\lfloor \frac{z}{2} \rfloor) & \text{ако } z \text{ е четно} \\ C_B(\lfloor \frac{z}{2} \rfloor) & \text{ако } z \text{ е нечетно.} \end{cases}$$

Ако не се сетим за това решение, бихме могли за начало да препишем условието за принадлежност към $A \oplus B$:

$$\begin{aligned} z \in A \oplus B &\iff \exists x(x \in A \ \&\ z = 2x) \ \vee \ \exists x(x \in B \ \&\ z = 2x+1) \\ &\iff \exists x \underbrace{C_A(x) + \chi_=(z, 2x) \simeq 0}_{A^*} \ \vee \ \exists x \underbrace{C_B(x) + \chi_=(z, 2x+1) \simeq 0}_{B^*}. \end{aligned}$$

Оттук $A \oplus B = \exists x A^* \cup \exists x B^*$. Довършете доказателството, като използвате някои свойства на полуразрешимите множества, които вече доказахме.

Нека сега $A \oplus B$ е полуразрешимо. Тогава за всяко естествено x:

$$x \in A \iff 2x \in A \oplus B \quad \text{и} \quad x \in B \iff 2x+1 \in A \oplus B,$$

откъдето

$$C_A(x) \simeq C_{A \oplus B}(2x) \quad \text{и} \quad C_B(x) \simeq C_{A \oplus B}(2x+1).$$

□

Задача 4.16. Нека $\emptyset \neq A \subseteq \mathbb{N}$ и $\emptyset \neq B \subseteq \mathbb{N}$. Докажете, че A и B са полуразрешими тогава и само тогава, когато е полуразрешимо директното им произведение

$$A \otimes B = \{\Pi(x, y) \mid x \in A \ \&\ y \in B\}.$$

Решение. Решението е много подобно на това от *Задача* 4.4. В правата посока имаме

$$z \in A \otimes B \iff L(z) \in A \ \&\ R(z) \in B,$$

откъдето

$$\begin{aligned} C_{A \otimes B}(z) \simeq 0 &\iff C_A(L(z)) \simeq 0 \ \&\ C_B(R(z)) \simeq 0 \\ &\iff C_A(L(z)) + C_B(R(z)) \simeq 0. \end{aligned}$$

Нека сега A и B са непразни множества с полуразрешимо декартово произведение $A \otimes B$. Избираме някакво $y_0 \in B$ (знаем, че $B \neq \emptyset$). Тогава очевидно

$$x \in A \iff \Pi(x, y_0) \in A \otimes B$$

и значи $C_A(x) \simeq C_{A \otimes B}(x, y_0)$. Аналогично разсъждаваме за другото множество B. □

Първото НДУ за полуразрешимост, което показахме (*Твърдение* 4.8) всъщност ни казва, че

$$A \subseteq \mathbb{N}^n \text{ е полуразрешимо} \iff A = Dom(f) \text{ за някоя изчислима } f.$$

Да видим, че имаме подобна характеризация на полуразрешимите подмножества на $\mathbb{N}$, но този път като *области от стойности* на изчислими функции.

Задача 4.17. Докажете, че $A \subseteq \mathbb{N}$ е полуразрешимо тогава и само тогава, когато $A = Range(f)$ за някоя едноместна изчислима функция f.

Решение. Нека $A \subseteq \mathbb{N}$ е полуразрешимо. Ако $A = \emptyset$, то очевидно $A = Range(\emptyset^{(1)})$. Ако пък A е непразно, то съгласно *Твърдение* 4.11, то ще се изброява от някоя (дори тотална) изчислима функция f.

Нека сега $A = Range(f)$ за някоя изчислима f. Тогава

$$x \in A \iff \exists n\ f(n) \simeq x \iff \exists n\ (n, x) \in G_f.$$

Сега прилагаме теоремата за графиката и теоремата за проекцията, за да заключим, че A е полуразрешимо. □

В *Твърдение* 4.3 видяхме, че първообраз на разрешимо множество чрез рекурсивна функция е разрешимо. За образ това не е така и вече сме в състояние да дадем контрапример.

Задача 4.18. Дайте пример за рекурсивна функция f и разрешимо множество A, такива че $f(A)$ не е разрешимо.

Решение. Тъй като всяко разрешимо множество е и полуразрешимо, съгласно *Твърдение* 4.15, $f(A)$ ще е полуразрешимо. Ние искаме то да не е разрешимо. Да тръгнем отново от множеството K. То е непразно, и съгласно *Твърдение* 4.11, ще съществува рекурсивна (и дори примитивно рекурсивна) функция f, която го изброява, т.е. за която $Range(f) = K$. Но $Range(f)$ е всъщност образът на цялото $\mathbb{N}$ чрез f, т.е. имаме

$$f(\mathbb{N}) = K.$$

Така получихме търсените рекурсивна функция f и разрешимо множество $A = \mathbb{N}$. □

Задача 4.19. Нека $A \subseteq \mathbb{N}$ е безкрайно и полуразрешимо. Докажете, че съществува безкрайно разрешимо множество B, такова че $B \subseteq A$.

Решение. A е непразно и полуразрешимо, следователно по *Твърдение* 4.11 ще съществува рекурсивна функция f, която го изброява (без значение със или без повторения). Като използваме f, ще конструираме *строго растяща* рекурсивна функция h, такава че

$$Range(h) \subseteq Range(f) = A.$$

Накрая ще се възползваме от *Твърдение* 4.5, което казва, че $Range(h)$ е разрешимо и ще положим $B := Range(h)$.

Изборът на h е общо взето прозрачен: строим я с примитивна рекурсия, като тръгваме от $h(0) := f(0)$, и после искаме $h(n+1)$ да е от вида $f(z)$, където $f(z) > h(n)$. Формално:

$$\left|\begin{array}{lcl} h(0) & = & f(0) \\ h(n+1) & = & f(\mu z[f(z) > h(n)]). \end{array}\right.$$

Това, че h е рекурсивна, строго растяща и $Range(h) \subseteq Range(f)$ следва непосредствено от дефиницията ѝ. Разбира се, $Range(h)$ също така е безкрайно. □

4.2.4 Теорема на Пост и приложения

Теорема 4.1. (Теорема на Пост) Ако множеството A и неговото допълнение $\bar{A}$ са полуразрешими, то A е разрешимо.

Доказателство. Ако разполагаме с алгоритми, които полуразпознават A и $\bar{A}$, интуитивно е ясно как ще разпознаваме A. За да разберем дали $\bar{x}$ е или не е в A, просто пускаме едновременно двата алгоритъма върху $\bar{x}$ и гледаме кой ще спре (ясно е, че *точно* един от двата със сигурност ще спре).

Формалното доказателство е съвсем кратко: характеристичната функция на A

$$\chi_A(\bar{x}) = \begin{cases} 0, & \text{ако } \bar{x} \in A \\ 1, & \text{ако } \bar{x} \in \bar{A} \end{cases}$$

е изчислима, съгласно *Твърдение* 4.17, откъдето получаваме, че A е разрешимо. □

Едно непосредствено обобщение на теоремата на Пост е следното:

Задача 4.20. Нека $A_1, \ldots, A_k$ са непресичащи се полуразрешими подмножества на $\mathbb{N}^n$, такива че $A_1 \cup \cdots \cup A_k = \mathbb{N}^n$. Докажете, че $A_1, \ldots, A_k$ са разрешими.

Решение. Да видим, например, че A_1 е разрешимо (за останалите разсъждението е аналогично).

Наистина, от $A_1 \cup \cdots \cup A_k = \mathbb{N}^n$ ще имаме $\bar{A_1} = A_2 \cup \cdots \cup A_k$, и понеже обединението запазва полуразрешимостта, то $\bar{A_1}$ е полуразрешимо. Сега по теоремата на Пост A_1 ще е разрешимо.

Разбира се, можехме да напишем директно характеристичната функция на A_1 и да приложим отново *Твърдение* 4.17:

$$\chi_{A_1}(\bar{x}) = \begin{cases} 0, & \text{ако } \bar{x} \in A_1 \\ 1, & \text{ако } \bar{x} \in A_2 \\ \ldots\ldots\ldots\ldots\ldots & \ldots\ldots\ldots \\ 1, & \text{ако } \bar{x} \in A_k. \end{cases}$$

□

Следващата задача също може да се разглежда като някакъв вид обобщение на теоремата на Пост.

Задача 4.21. Нека A и B са полуразрешими множества от естествени числа, такива че $A \cup B$ и $A \cap B$ са разрешими. Докажете, че тогава A и B са разрешими.

Решение. Лесно се съобразява, че

$$\begin{aligned} x \notin A &\iff x \notin A \cup B \ \vee \ x \in B \ \& \ x \notin A \\ &\iff x \in \overline{A \cup B} \ \vee \ x \in B \ \& \ x \notin A \cap B \\ &\iff x \in \overline{A \cup B} \ \vee \ x \in B \ \& \ x \in \overline{A \cap B}. \end{aligned}$$

Следователно

$$\bar{A} \ = \ \overline{A \cup B} \ \cup \ B \cap (\overline{A \cap B})$$

и значи $\bar{A}$ е полуразрешимо. Оттук по теоремата на Пост, A ще е разрешимо. За B разсъждаваме аналогично. □

Задача 4.22. Нека $A \subseteq \mathbb{N}$ е разрешимо множество, а f е едноместна рекурсивна функция, такава че $Range(f) = \mathbb{N}$ и $f(A) \cap f(\bar{A}) = \emptyset$. Докажете, че множеството $f(A)$ е разрешимо.

Решение. Имаме, че A и $\bar{A}$ са разрешими, и в частност — полуразрешими. Оттук по *Твърдение* 4.15, $f(A)$ и $f(\bar{A})$ ще са полуразрешими също. Ясно е, че $f(A) \cup f(\bar{A}) = Range(f) = \mathbb{N}$. Освен това по условие $f(A)$ и $f(\bar{A})$ са непресичащи се. Това означава, че едното множество е допълнение на другото и значи по теоремата на Пост и двете множества са разрешими. □

От теоремата на Пост лесно следва, че допълнението на множеството K не е полуразрешимо.

Задача 4.23. Докажете, че множеството $\overline{K}$ не е полуразрешимо.

Решение. Знаем, че K е полуразрешимо, но не е разрешимо (*Твърдение* 4.7). Ако допуснем, че $\overline{K}$ е полуразрешимо, то по теоремата на Пост ще получим, че K е разрешимо — противоречие. □

Забележка. Тази задача показва още, че допълнението не запазва полуразрешимостта.

Задача 4.24. Докажете, че кванторът за всеобщност не запазва полуразрешимостта.

Решение. Следва от решението на *Задача* 4.12 и факта, че множеството $\overline{K}$ не е полуразрешимо. □

По-горе отбелязахме (*Задача* 4.20), че теоремата на Пост е в сила за произволен брой непресичащи се полуразрешими подмножества, чието обединение "запълва" $\mathbb{N}^n$. Дали същото може да се твърди за безкраен брой множества със същите свойства? Оказва се, че не. Доказателството — малко по-надолу.

За да формулираме следващата задача за ЕК, ще въведем едно спомагателно понятие.

Нека $A \subseteq \mathbb{N}^{n+1}$. *Селектор* за A ще наричаме n-местната функция f, за която е изпълнено:

$$\exists y\ (\bar{x}, y) \in A \implies !f(\bar{x})\ \&\ (\bar{x}, f(\bar{x})) \in A$$

за всяко $\bar{x} \in \mathbb{N}$.

Задача 4.25. (Задача за ЕК) Докажете, че всяко полуразрешимо множество $A \subseteq \mathbb{N}^n, n \geq 2$, притежава изчислим селектор.

Задача 4.26. (Задача за ЕК) Нека A е разрешимо. Докажете, че следните две условия са еквивалентни:

1) A има рекурсивен селектор;

2) множеството $B = \{\ \bar{x} \mid \exists y\ (\bar{x}, y) \in A\ \}$ е разрешимо.

4.2.5 Ефективно изброяване (номериране) на полуразрешимите множества

За фиксирани $n \geq 1$ и $a \in \mathbb{N}$, с $W_a^{(n)}$ ще означаваме дефиниционната област на функцията $\varphi_a^{(n)}$, а с $E_a^{(n)}$ — множеството от нейните стойности:

$$W_a^{(n)} \ = \ Dom(\varphi_a^{(n)}) \stackrel{\text{деф}}{=} \{\ \bar{x} \mid !\varphi_a^{(n)}(\bar{x})\ \}$$
$$E_a^{(n)} \ = \ Range(\varphi_a^{(n)}) \stackrel{\text{деф}}{=} \{\ y \mid \exists \bar{x} \in \mathbb{N}^n\colon\ \varphi_a^{(n)}(\bar{x}) \simeq y\ \}.$$

От първото НДУ за полуразрешимост (*Твърдение* 4.8) знаем, че едно множество $A \subseteq \mathbb{N}^n$ е полуразрешимо тогава и само тогава, когато съществува изчислима функция f, такава че $A = Dom(f)$. Оттук:

$$A \subseteq \mathbb{N}^n \text{ е полуразрешимо } \iff \exists a\ A = Dom(\varphi_a^{(n)}) \iff \exists a\ A = W_a^{(n)}.$$

Ако $A = W_a^{(n)}$, то числото a ще наричаме *индекс* на A. Понеже всяка изчислима функция има безброй много индекси, то и всяко полуразрешимо множество ще има безброй много индекси.

При $n = 1$ ще пишем за по-кратко

$$W_a \ = \ Dom(\varphi_a) \quad \text{и} \quad E_a \ = \ Range(\varphi_a).$$

Ясно е, че редицата

$$W_0,\ W_1,\ \ldots\ , W_a,\ \ldots$$

се състои от всички полуразрешими подмножества на $\mathbb{N}$. Чрез тях множеството $K \stackrel{\text{деф}}{=} \{x \mid !\varphi_x(x)\}$ можем да препишем още и така:

$$K\ =\ \{\ x \mid x \in W_x\ \}.$$

Тогава $\overline{K}\ =\ \{\ x \mid x \notin W_x\ \}$ и сега вече е ясно и без теоремата на Пост защо $\overline{K}$ не може да е полуразрешимо — защото по дефиниция

$$x \in \overline{K}\ \iff\ x \notin W_x,$$

и значи $\overline{K}$ и W_x се различават в точката x. С други думи, $\overline{K} \neq W_x$ за всяко x, следователно $\overline{K}$ не е полуразрешимо.

От *Твърдение* 4.12 знаем, че обединението на две, а оттук и на краен брой полуразрешими множества, е полуразрешимо. Когато имаме безкраен брой полуразрешими множества, обединението им вече може и да не е такова. Най-простият контрапример е да вземем някакво "сложно" множество $D = \{\ a_0,\ a_1,\ \ldots\ \}$ и да разгледаме едноелементните множества $A_n = \{a_n\}$ за $n = 0, 1, \ldots$. Всяко такова A_n е крайно, и в частност — полуразрешимо, обаче обединението на тези множества, което е точно D, вече не е полуразрешимо.

Когато, обаче, обединяваме множества, чийто индекси принадлежат на някое полуразрешимо множество A, това обединение със сигурност ще е полуразрешимо. Да се убедим:

Задача 4.27. Нека $A \subseteq \mathbb{N}$ е полуразрешимо. Докажете, че е полуразрешимо и множеството

$$B\ =\ \bigcup_{a \in A} W_a.$$

Решение. По определение

$$\begin{aligned} x \in B\ &\iff\ \exists a\ (a \in A\ \&\ x \in W_a)\ \iff\ \exists a(a \in A\ \&\ !\Phi_1(a,x)) \\ &\iff\ \exists a\ (a \in A\ \&\ (a,x) \in Dom(\Phi_1)) \\ &\iff\ \exists a\ (a,x) \in (A \times \mathbb{N}) \cap Dom(\Phi_1). \end{aligned}$$

Тъй като множеството $(A \times \mathbb{N}) \cap Dom(\Phi_1)$ е полуразрешимо, то ще е полуразрешимо и множеството B, съгласно теоремата за проекцията. $\square$

Знаем, че всяко полуразрешимо $A \subseteq \mathbb{N}$ има индекс, т.е. можем да си го мислим във вида $A = W_b$ за някое b. Това ни дава възможност да обобщим горната задача по следния начин:

Задача 4.28. Докажете, че е полуразрешимо множеството

$$B^* \;=\; \{\; (b,x) \mid x \in \bigcup_{a \in W_b} W_a \;\}.$$

Решение. По определение

$$\begin{aligned}(b,x) \in B^* &\iff \exists a(a \in W_b \;\&\; x \in W_a) \iff \exists a(!\varphi_b(a) \;\&\; !\varphi_a(x)) \\ &\iff \exists a\, !(\varphi_b(a) + \varphi_a(x)) \iff \exists a\, !\underbrace{\Phi_1(b,a) + \Phi_1(a,x)}_{f(a,b,x)}.\end{aligned}$$

Понеже f е изчислима, то множеството $C = Dom(f)$ ще е полуразрешимо. Оттук по теоремата за проекцията ще е полуразрешимо и B^*.

□

Всички твърдения за полуразрешими множества, които доказахме в предишния раздел, имат и т. нар. "равномерни" версии, чиито доказателства се основават на (слабата) S^m_n-теорема.

Равномерната версия на *Твърдение* 4.12 1) ще изглежда така:

Задача 4.29. Докажете, че съществуват примитивно рекурсивни функции cut и uni, такива че за всяко a и b:

$$W_{cut(a,b)} \;=\; W_a \cap W_b \quad \text{и} \quad W_{uni(a,b)} \;=\; W_a \cup W_b.$$

Решение. Първо да решим задачата за сечението. Искаме да конструираме изчислима функция $f(a,b,x)$, която е такава, че след прилагане на S^m_n-теоремата към нея и намиране на функция $cut(a,b)$, такава че $\varphi_{cut(a,b)}(x) \simeq f(a,b,x)$, да се окаже, че $cut(a,b)$ търсената, т.е. за нея е изпълнено $W_{cut(a,b)} \;=\; W_a \cap W_b$.

За целта можем да вземем f да е следната функция:

$$f(a,b,x) \simeq \varphi_a(x) + \varphi_b(x).$$

f е изчислима, защото можем да я препишем чрез универсалната функция като $f(a,b,x) \simeq \Phi_1(a,x) + \Phi_1(b,x)$. Прилагаме към нея S^m_n-теоремата и получаваме, че за някоя примитивно рекурсивна функция $cut(a,b)$ ще е изпълнено $\varphi_{cut(a,b)}(x) \simeq f(a,b,x)$, или все едно

$$\varphi_{cut(a,b)}(x) \simeq \varphi_a(x) + \varphi_b(x).$$

Оттук получаваме, че за произволни a, b и x:

$$x \in W_{cut(a,b)} \iff !\varphi_{cut(a,b)}(x) \iff !\varphi_a(x) \;\&\; !\varphi_b(x) \iff x \in W_a \cap W_b,$$

и следователно $W_{cut(a,b)} \;=\; W_a \cap W_b$.

При обединението на W_a и W_b нещата са по-сложни (знаем го още от "неравномерната" версия на това твърдение). Затова започваме отдалече — с полуразрешимото множество $U = Dom(\Phi_1)$. За него съществува рекурсивна функция ρ, такава че за всяко a и x:

$$(a,x) \in U \iff \exists t\ \rho(a,x,t) = 0.$$

Тогава

$$\begin{aligned} x \in W_a \cup W_b &\iff x \in W_a \ \vee\ x \in W_b \iff (a,x) \in U \ \vee\ (b,x) \in U \\ &\iff \exists t\ \rho(a,x,t) = 0 \ \vee\ \exists t\ \rho(b,x,t) = 0 \\ &\iff \exists t\ \rho(a,x,t).\rho(b,x,t) = 0 \iff !\underbrace{\mu t[\rho(a,x,t).\rho(b,x,t) = 0]}_{f(a,b,x)}. \end{aligned}$$

Към функцията f вдясно прилагаме S_n^m-теоремата и получаваме, че съществува примитивно рекурсивна функция $uni(a,b)$, такава че

$$\varphi_{uni(a,b)}(x) \simeq f(a,b,x).$$

Тогава за всяко a, b и x ще е изпълнено:

$$x \in W_a \cup W_b \iff !f(a,b,x) \iff !\varphi_{uni(a,b)}(x) \iff x \in W_{uni(a,b)},$$

което означава, че функцията uni има исканото свойство. □

Да докажем и равномерната версия на *Твърдение* 4.15, което казваше, че образът и първообразът на полуразрешимо множество чрез изчислима функция също е полуразрешимо множество.

Задача 4.30. Докажете, че съществуват двуместни примитивно рекурсивни функции im и $preim$, такива че за всяко a и b е изпълнено:

1) $W_{preim(a,b)} = \varphi_a^{-1}(W_b)$;

2) $W_{im(a,b)} = \varphi_a(W_b)$.

Решение. 1) За първообраза на W_b чрез φ_a имаме:

$$x \in \varphi_a^{-1}(W_b) \iff !\varphi_a(x)\ \&\ \varphi_a(x) \in W_b \iff !\varphi_b(\varphi_a(x)) \iff !\underbrace{\Phi_1(b, \Phi_1(a,x))}_{f(a,b,x)}$$

за всяко a, b и x. Функцията f е изчислима и по S_n^m-теоремата ще съществува примитивно рекурсивна функция $preim(a,b)$, такава че

$$\varphi_{preim(a,b)}(x) \simeq f(a,b,x).$$

Тогава за всяко a, b и x ще е изпълнено:

$$x \in \varphi_a^{-1}(W_b) \iff !f(a,b,x) \iff !\varphi_{preim(a,b)}(x) \iff x \in W_{preim(a,b)},$$

и значи функцията $preim$ е търсената.

2) За образа $\varphi_a(W_b)$ разсъждаваме така:

$$\begin{aligned} y \in \varphi_a(W_b) &\iff \exists x(x \in W_b \ \&\ \varphi_a(x) \simeq y) \\ &\iff \exists x(\mathcal{O}(\Phi_1(b,x)) \simeq 0 \ \&\ |(\Phi_1(a,x) - y| \simeq 0) \\ &\iff \exists x(\underbrace{\mathcal{O}(\Phi_1(b,x)) + |(\Phi_1(a,x) - y| \simeq 0}_{\{(a,b,y,x) \mid (a,b,x,y) \in A\}}). \end{aligned}$$

Функцията $\mathcal{O}(\Phi_1(b,x)) + |(\Phi_1(a,x) - y|$ е изчислима и следователно множеството A е полуразрешимо. Сега по теоремата за проекцията ще е полуразрешимо и множеството $A^* = \exists x A = \{(a,b,y) \mid \exists x (a,b,x,y) \in A \ \}$. Тогава ще е изчислима полухарактеристичната функция C_{A^*} на това множество и значи по S_n^m-теоремата ще съществува примитивно рекурсивна функция $im(a,b)$, такава че

$$\varphi_{im(a,b)}(y) \simeq C_{A^*}(a,b,y).$$

Така ще имаме, че за всяко a, b и y:

$$\begin{aligned} y \in \varphi_a(W_b) &\iff \exists x \ (a,b,y,x) \in A \iff (a,b,y) \in A^* \\ &\iff !C_{A^*}(a,b,y) \iff !\varphi_{im(a,b)}(y) \iff y \in W_{im(a,b)} \end{aligned}$$

и следователно $\varphi_a(W_b) = W_{im(a,b)}$. □

Да покажем, че теоремата на Пост не може да се обобщи за *безкраен* брой непресичащи се полуразрешими подмножества на $\mathbb{N}^n$:

Задача 4.31. Дайте пример за множества от естествени числа $A_0, A_1, \ldots,$ които са са непресичащи се и полуразрешими, за които още $A_0 \cup A_1 \cup \cdots = \mathbb{N}$, но не всички те са разрешими.

Решение. Знаем, че $\overline{K}$ не е полуразрешимо, следователно е безкрайно. Нека $\overline{K} = \{a_1, a_2, \ldots\}$. Разглеждаме следните множества A_i:

$$A_0 = K, \ A_1 = \{a_1\}, \ A_2 = \{a_2\}, \ \ldots$$

Ясно е, че те удовлетворяват условието на задачата — две по две са непресичащи се, полуразрешими са и обединението им е точно $\mathbb{N}$. Обаче първото множество $A_0 = K$ очевидно не е разрешимо. □

Оказва се, че теоремата на Пост все пак е вярна и за безкраен брой множества $A_0, A_1, \ldots$. Необходимо е, обаче, редицата от тези множества да е *ефективна*. Какво означава това?

Нека $A_0, A_1, \ldots$ е редица от полуразрешими подмножества на $\mathbb{N}^k$. Казваме, че тази редица е *ефективна*, ако съществува рекурсивна функция h, такава че за всяко n:

$$A_n = W^{(k)}_{h(n)}.$$

Функцията h ще наричаме *индексна функция* за тази редица.

Задача 4.32. Нека $A_0, A_1, \ldots$ е ефективна редица от непресичащи се полуразрешими подмножества на $\mathbb{N}^k$, такива че $A_0 \cup A_1 \cup \cdots = \mathbb{N}^k$. Докажете, че всяко от тези множества е разрешимо.

Решение. За произволно n ще покажем, че допълнението $\overline{A_n}$ на множеството A_n е полуразрешимо. И тъй като и A_n е полуразрешимо, по теоремата на Пост ще следва, че A_n е разрешимо.

Нека h е индексната функция на дадената редица. Тогава за произволно $\bar{x} \in \mathbb{N}^k$ ще е изпълнено:

$$\begin{aligned}
\bar{x} \in \overline{A_n} &\iff \exists m(m \neq n \ \&\ \bar{x} \in A_m) \iff \exists m(m \neq n \ \&\ \bar{x} \in W^{(k)}_{h(m)}) \\
&\iff \exists m(\chi_{\neq}(m,n) = 0 \ \&\ \mathcal{O}(\varphi^{(k)}_{h(m)}(\bar{x})) \simeq 0) \\
&\iff \exists m(\underbrace{\chi_{\neq}(m,n) + \mathcal{O}(\Phi_k(h(m),\bar{x}))}_{f(m,n,\bar{x})} \simeq 0).
\end{aligned}$$

Функцията f е изчислима, следователно множеството $A = \{(m,n,\bar{x}) \mid f(m,n,\bar{x}) \simeq 0\}$ ще е полуразрешимо, откъдето по теоремата за проекцията и $\overline{A_n}$ ще е полуразрешимо. □

Задача 4.33. Докажете, че една редица $A_0, A_1, \ldots$ от полуразрешими подмножества на $\mathbb{N}^k$ е ефективна тогава и само тогава, когато множеството

$$U \ = \ \{(n,\bar{x}) \mid \bar{x} \in A_n\}$$

е полуразрешимо.

Забележка. Да отбележим, че множеството U е нещо като универсално множество за редицата $\{A_n\}_n$.

Решение. В правата посока: нека h е рекурсивна функция, такава че за всяко n: $A_n = W^{(k)}_{h(n)}$. Тогава

$$(n,\bar{x}) \in U \overset{\text{деф}}{\iff} \bar{x} \in A_n \iff \bar{x} \in W^{(k)}_{h(n)} \iff !\varphi^{(k)}_{h(n)}(\bar{x}) \iff !\Phi_k(h(n),\bar{x}).$$

Понеже функцията $\lambda n, \bar{x}.\Phi_k(h(n),\bar{x})$ е изчислима, то множеството U е полуразрешимо.

Обратно, ако горното множество U е полуразрешимо, то полухарактеристичната му функция $C_U(n,\bar{x})$ е изчислима. Прилагаме към нея S^m_n-теоремата и получаваме, че за някоя рекурсивна функция h ще е изпълнено:

$$\varphi^{(k)}_{h(n)}(\bar{x}) \simeq C_U(n,\bar{x}).$$

Тогава за всяко n и $\bar{x} \in \mathbb{N}^k$ ще имаме:

$$\bar{x} \in A_n \iff (n,\bar{x}) \in U \iff !C_U(n,\bar{x}) \iff !\varphi^{(k)}_{h(n)}(\bar{x}) \iff \bar{x} \in W^{(k)}_{h(n)},$$

и следователно редицата $A_0, A_1, \ldots$ е ефективна. $\square$

От *Задача* 4.17 знаем, че едно множество $A \subseteq \mathbb{N}$ е полуразрешимо тогава и само тогава, когато $A = Range(f)$ за някоя едноместна изчислима функция f,или все едно:

$$A \subseteq \mathbb{N} \text{ е полуразрешимо } \iff \exists a\ A = E_a.$$

С други думи, множеството $A \subseteq \mathbb{N}$ е полуразрешимо точно когато е от вида E_a за някое a. Така получаваме друга система за индексиране на полуразрешимите множества от естествени числа. Да видим, че между двете системи за индексиране има равномерен преход:

Задача 4.34. Докажете, че съществуват примитивно рекурсивни функции α и β, такива че за всяко a:

а) $W_{\alpha(a)} = E_a$

б) $E_{\beta(a)} = W_a$.

Решение. а) Ще ни трябва изчислима функция $f(a,x)$, такава че $Dom(\lambda x.f(a,x))$ да е точно E_a. Да вземем например

$$f(a,x) \simeq \begin{cases} 0, & \text{ако } x \in E_a \\ \neg!, & \text{иначе.} \end{cases}$$

Всъщност f е полухарактеристична функция на множеството $A = \{(a,x) \mid x \in E_a\}$. Да се убедим, че A е полуразрешимо. Имаме

$$(a,x) \in A \iff \exists y\ \varphi_a(y) \simeq x \iff \exists y \underbrace{|\Phi_1(a,y) - x| \simeq 0}_{g(a,x,y)}.$$

Функцията g е изчислима и значи множеството $A_0 = \{(a,x,y) \mid g(a,x,y) \simeq 0\}$ ще е е полуразрешимо, откъдето по теоремата за проекцията и A ще е полуразрешимо. Тогава $f = C_A$ е изчислима. По S_n^m-теоремата ще съществува примитивно рекурсивна функция α, такава че

$$\varphi_{\alpha(a)}(x) \simeq f(a,x),$$

откъдето получаваме, че за всяко a и x:

$$x \in W_{\alpha(a)} \iff !\varphi_{\alpha(a)}(x) \iff$$

$$x \in W_{\alpha(a)} \iff !f(a,x) \iff x \in E_a.$$

Следователно $W_{\alpha(a)} = E_a$ за всяко a.

б) Ще ни е нужна изчислима функция $f(a,x)$, която този път трябва да е такава, че $Range(\lambda x.f(a,x))$ да е W_a. Един начин да изберем f е следният:

$$f(a,x) \simeq \begin{cases} x, & \text{ако } x \in W_a \\ \neg!, & \text{иначе.} \end{cases}$$

Условието $x \in W_a$ от нейната дефиниция е еквивалентно на $(a,x) \in Dom(\Phi_1)$, което е полуразрешимо, и значи съгласно *Следствие* 4.5, функцията f е изчислима. Сега отново от S_n^m-теоремата ще имаме, че за някоя примитивно рекурсивна функция β ще е изпълнено:

$$\varphi_{\beta(a)}(x) \simeq g(a,x).$$

Оттук за всяко a и y ще имаме

$$\begin{aligned} y \in E_{\beta(a)} &\iff \exists x\ \varphi_{\beta(a)}(x) \simeq y \iff \exists x\ g(a,x) \simeq y \\ &\iff \exists x\ (y = x\ \&\ x \in W_a) \iff y \in W_a, \end{aligned}$$

и значи $E_{\beta(a)} = W_a$ за всяко a. □

Задача 4.35. (Задача за ЕК) Нека $A \subseteq \mathbb{N}$ и $R \subseteq \mathbb{N}^2$ са полуразрешими множества. С индукция по n дефинираме следната редица от подмножества на $\mathbb{N}$:

$$A_0 = A, \quad A_{n+1} = \{\ x \mid \exists y((x,y) \in R\ \&\ y \in A_n\ \}.$$

Докажете, че:

а) всяко от множествата A_n е полуразрешимо;

б) редицата $A_0,\ A_1,\ \ldots$ е ефективна.

4.3 Алгоритмично неразрешими проблеми

4.3.1 m-сводимост

За да сравняваме алгоритмичната сложност на *масови проблеми*, които ще представяме с множества от естествени числа, въвеждаме следната релация между две числови множества A и B:

Определение 4.5. Нека $A \subseteq \mathbb{N}$ и $B \subseteq \mathbb{N}$. Ще казваме, че A е *m-сводимо* към B (и ще пишем $A \leqslant_m B$), ако съществува рекурсивна функция f, такава че за всяко $x \in \mathbb{N}$ е в сила еквивалентността:

$$x \in A \iff f(x) \in B.$$

Забележка. Терминът е m-сводимост, защото функцията f в общия случай е "many-to-one" (т.е. неинективна), откъдето идва и името на сводимостта. Когато f е "one-to-one" (инективна), се говори за 1-сводимост, която се означава с $\leqslant_1$. Тъй като за нашите цели ще ни е нужна само m-сводимостта, ще я наричаме просто *сводимост* и съответно ще пишем $A \leqslant B$ вместо $A \leqslant_m B$.

Когато искаме да означим, че A се свежда към B чрез функцията f, ще пишем

$$A \underset{f}{\leqslant} B.$$

Ясно е, че ако $A \underset{f}{\leqslant} B$, то и $\bar{A} \underset{f}{\leqslant} \bar{B}$. Да споменем също, че ако $A \underset{f}{\leqslant} B$, то всъщност $A = f^{-1}(B)$, т.е. A е първообразът на B чрез f.

Ако $A \underset{f}{\leqslant} B$, то очевидно

$$\chi_A = \chi_B \circ f \quad \text{и} \quad C_A = C_B \circ f.$$

Оттук следва, че ако B е разрешимо (полуразрешимо), то и A ще е разрешимо (полуразрешимо). С други думи, ако имаме разрешаваща (полуразрешаваща) процедура за B, то ще имаме подобна процедура и за A. В този смисъл, ако $A \leqslant B$, то множеството A е алгоритмично по-просто от множеството B (което обяснява означението $\leqslant$).

Една от типичните задачи в този раздел ще бъде да показваме, че дадено множество *не* е разрешимо или полуразрешимо. За тази цел се оказва по-удобно да използваме контрапозицията на наблюдението по-горе. Понеже ще го цитираме често, да го формулираме като отделно твърдение:

Твърдение 4.18. Нека $A \leqslant B$. Тогава ако A не е разрешимо (полуразрешимо), то и B не е разрешимо (полуразрешимо).

От *Твърдение* 4.7 и *Задача* 4.23 знаем, че множеството $K \stackrel{\text{деф}}{=} \{\, x \mid !\varphi_x(x) \,\}$ не е разрешимо, а допълнението му $\overline{K}$ не е и полуразрешимо. K и $\overline{K}$ ще бъдат еталонни множества за нас. Ако за някое множество A успеем да покажем, че $K \leqslant A$, това със сигурност ще означава, че A не е разрешимо, а ако стигнем до неравенството $\overline{K} \leqslant A$, то A няма да е дори полуразрешимо.

Твърдение 4.19. Релацията $\leqslant$ е рефлексивна и транзитивна.

Доказателство. Рефлексивност: ясно е, че A се свежда към себе си чрез функцията идентитет.

Транзитивност: нека $A \underset{f}{\leqslant} B$ и $B \underset{g}{\leqslant} C$. Тогава за всяко $x \in \mathbb{N}$:

$$x \in A \iff f(x) \in B \iff g(f(x)) \in C.$$

Така получаваме, че A се свежда към C чрез функцията $g \circ f$. □

Забележка. Да отбележим, че релацията $\leqslant$ не е антисиметрична. Понадолу ще имаме много примери на *различни* множества A и B, такива че $A \leqslant B$ и $B \leqslant A$. Ако искате да имате контрапример още сега, вземете например множествата $\mathbb{N}$ и $\mathbb{N}^+$. За всяко x ще са в сила еквивалентностите

$$x \in \mathbb{N} \iff \mathcal{S}(x) \in \mathbb{N}^+ \quad \text{и} \quad x \in \mathbb{N}^+ \iff x \dot{-} 1 \in \mathbb{N},$$

които ни дават $\mathbb{N} \leqslant \mathbb{N}^+$ и $\mathbb{N}^+ \leqslant \mathbb{N}$. В същото време $\mathbb{N} \neq \mathbb{N}^+$.

Определението за сводимост се обобщава по един съвсем естествен начин и за подмножества на *произволни* декартови степени на $\mathbb{N}$:

Определение 4.6. Нека $A \subseteq \mathbb{N}^n$ и $B \subseteq \mathbb{N}^k$. Ще казваме, че A е *сводимо* към B (и отново ще пишем $A \leqslant B$), ако съществуват рекурсивни функции $f_1, \ldots, f_k$, такива че за всяко $\bar{x} \in \mathbb{N}^n$ е изпълнено:

$$\bar{x} \in A \iff (f_1(\bar{x}), \ldots, f_k(\bar{x})) \in B.$$

От тази еквивалентност веднага получаваме, че за всяко $\bar{x}$:

$$\chi_A(\bar{x}) = \chi_B(f_1(\bar{x}), \ldots, f_k(\bar{x})) \quad \text{и} \quad C_A(\bar{x}) \simeq C_B(f_1(\bar{x}), \ldots, f_k(\bar{x})),$$

откъдето

$$\chi_A = \chi_B(f_1, \ldots, f_k) \quad \text{и} \quad C_A = C_B(f_1, \ldots, f_k).$$

Следователно *Твърдение* 4.18 остава вярно и за случая на произволни множества A и B. *Твърдение* 4.19 също продължава да е вярно (убедете се сами).

Да напомним дефиницията и на множеството U, което въведохме миналия път:

$$U \;=\; \{(a,x) \mid x \in W_a\} \;=\; \{(a,x) \mid !\Phi_1(a,x)\}.$$

Множеството U е дефиниционна област на най-сложната изчислима функция — универсалната функция Φ_1, поради което очакваме да е най-сложното полуразрешимо множество. Да се убедим:

Твърдение 4.20. Всяко полуразрешимо множество $A \subseteq \mathbb{N}$ е сводимо към U.

Решение. Щом $A \subseteq \mathbb{N}$ е полуразрешимо, то $A = W_a$ за някое a. Но тогава за всяко $x \in \mathbb{N}$ ще имаме, че

$$x \in W_a \iff (\underbrace{a}_{f_1(x)}, \underbrace{x}_{f_2(x)}) \in U.$$

Следователно A е сводимо към U чрез функциите $f_1 = \lambda x.a$ и $f_2 = \lambda x.x$, които очевидно са рекурсивни. □

Забележка. Множеството U се нарича *универсално* множество за всички полуразрешими подмножества на $\mathbb{N}$.

От горното твърдение следва, в частност, че и $K \leqslant U$, защото $K \subseteq \mathbb{N}$ е полуразрешимо. Оказва се, че е вярно и обратното неравенство $U \leqslant K$. Ще го получим като следствие от твърдението по-долу, според което и K е най-сложното сред полуразрешимите подмножества на $\mathbb{N}$.

Твърдение 4.21. За всяко полуразрешимо $A \subseteq \mathbb{N}$ е вярно, че $A \leqslant K$.

Доказателство. Да дефинираме функцията f както следва:

$$f(a,x) \simeq \begin{cases} 0, & \text{ако } a \in A \\ \neg!, & \text{иначе.} \end{cases}$$

f е изчислима, защото A е полуразрешимо ($f(a,x) \simeq C_A(a)$). Тогава по S^m_n-теоремата ще съществува рекурсивна функция h, такава че за всяко a и x: $\varphi_{h(a)}(x) \simeq f(a,x)$, с други думи, за всяко $a, x \in \mathbb{N}$:

$$\varphi_{h(a)}(x) \simeq \begin{cases} 0, & \text{ако } a \in A \\ \neg!, & \text{иначе.} \end{cases}$$

Това можем да запишем и така:

$$\varphi_{h(a)} = \begin{cases} \mathcal{O}, & \text{ако } a \in A \\ \emptyset^{(1)}, & \text{иначе.} \end{cases}$$

Твърдим, че A се свежда към K чрез функцията h. Наистина, за всяко $a \in \mathbb{N}$ имаме:

$$a \in A \iff \varphi_{h(a)} = \mathcal{O} \iff !\varphi_{h(a)}(h(a)) \overset{\text{деф}}{\iff} h(a) \in K.$$

□

От релацията $\leqslant$ по стандартния начин въвеждаме *еквивалентност* на две множества A и B:

$$A \equiv B \overset{\text{деф}}{\iff} A \leqslant B \ \& \ B \leqslant A.$$

Ясно е, че тази релация е релация на еквивалентност (тя очевидно е симетрична, а от *Твърдение* 4.19 лесно следва, че е и рефлексивна и транзитивна). Една хубава задача в тази връзка е следната:

Задача 4.36. (Задача за ЕК) Разглеждаме $\equiv$ върху множеството $\mathcal{R}$ на всички разрешими подмножества на $\mathbb{N}$. Опишете класовете на еквивалентност на тази релация.

По-горе лесно съобразихме, че $\mathbb{N} \equiv \mathbb{N}^+$. Ето и първият ни не толкова очевиден пример за двойка еквивалентни множества — множеството K и универсалното множество U.

Задача 4.37. Докажете, че $U \equiv K$.

Решение. Като следствие от *Твърдение* 4.20 получихме, че $K \leqslant U$. За да съобразим обратното, да разгледаме множеството $\hat{U}$, което на практика има същата сложност като U, но вече е подмножество на $\mathbb{N}$:

$$\hat{U} \overset{\text{деф}}{=} \{\ \Pi(a,x) \mid (a,x) \in U\ \}.$$

Ясно е, че U е полуразрешимо и следователно $\hat{U} \leqslant K$, съгласно *Твърдение* 4.21. Освен това имаме, че $U \leqslant \hat{U}$, което се вижда от еквивалентността:

$$(a,x) \in U \iff \Pi(a,x) \in \hat{U}.$$

Сега вече от $U \leqslant \hat{U}$ и $\hat{U} \leqslant K$ по транзитивност достигаме до неравенството $U \leqslant K$, което заедно с обратното $K \leqslant U$ ни дава $U \equiv K$. □

4.3.2 Неразрешимост на стоп-проблема за МНР

За едно множество $A \subseteq \mathbb{N}^n$ ще казваме, че е *неразрешимо*, ако то не е разрешимо.

При фиксирано $n \geq 1$ да положим:

$$HP_n \overset{\text{деф}}{=} \{(a, x_1, \ldots, x_n) \mid P_a \text{ спира върху } (x_1, \ldots, x_n)\}.$$

Теорема 4.2. (Стоп-проблемът за МНР е неразрешим) Нека $n \geq 1$. Множеството $HP_n = \{ (a, x_1, \ldots, x_n) \mid P_a \text{ спира върху } (x_1, \ldots, x_n) \}$ не е разрешимо.

Доказателство. Най-напред ще се спрем на случая $n = 1$. В този случай HP_1 е всъщност множеството, което по-горе означихме с U, и за което видяхме (*Задача* 4.23), че $K \leqslant U$. И понеже K не е разрешимо, то и U не може да е разрешимо, съгласно *Твърдение* 4.18.

Нека сега $n \geq 1$ е произволно. Ще покажем, че $HP_1 \leqslant HP_n$, откъдето ще следва, че и множеството HP_n е неразрешимо.

Да разгледаме функцията

$$f(a, x_1, \ldots, x_n) \overset{\text{деф}}{\simeq} \varphi_a(x_1).$$

Тя е изчислима и следователно за някоя рекурсивна функция h, съгласно S_n^m-теоремата, ще е изпълнено

$$\varphi^{(n)}_{h(a)}(x_1, \ldots, x_n) \simeq f(a, x_1, \ldots, x_n), \text{ т.е. } \varphi^{(n)}_{h(a)}(x_1, \ldots, x_n) \simeq \varphi_a(x_1)$$

за всяко $a \in \mathbb{N}, \bar{x} \in \mathbb{N}^n$. При $x_1 = \cdots = x_n = x$ ще имаме

$$\varphi^{(n)}_{h(a)}(\underbrace{x, \ldots, x}_{n \text{ пъти}}) \simeq \varphi_a(x),$$

откъдето в частност

$$!\varphi_a(x) \iff !\varphi^{(n)}_{h(a)}(\underbrace{x, \ldots, x}_{n \text{ пъти}}).$$

Преписана чрез HP_1 и HP_n, горната еквивалентност изглежда така:

$$(a, x) \in HP_1 \iff (h(a), \underbrace{x, \ldots, x}_{n \text{ пъти}}) \in HP_n.$$

Тя е в сила за всяко a и x, и значи $HP_1 \leqslant HP_n$. □

Забележка. Да отбележим, че множеството HP_n все пак е полуразрешимо, защото

$$HP_n = \{(a, \bar{x}) \mid P_a \text{ спира върху } \bar{x}\} = \{(a, \bar{x}) \mid !\Phi_n(a, \bar{x})\}.$$

Това означава, че стоп-проблемът за МНР е полуразрешим.

Да означим с RHP_n (от *restricted halting problem*) следното множество:

$$RHP_n \overset{\text{деф}}{=} \{(a, \bar{x}, t) \mid P_a \text{ спира върху } \bar{x} \text{ за } \leq t \text{ стъпки}\}.$$

Твърдение 4.22. (Ограниченият стоп-проблем за МНР е разрешим) За всяко $n \geq 1$ множеството RHP_n е разрешимо.

Доказателство. Най-напред да си припомним, че съгласно определението за код на програма, ако P_a: $I_0, \ldots, I_k$, то номерът k на последния оператор на P_a е $lh(a)$.

Разполагаме и с примитивно рекурсивна функция $Q_n(a, \bar{x}, t)$, която дава кода на конфигурацията на стъпка t от работата на P_a върху $\bar{x}$. Тогава очевидно

$$P_a \text{ спира върху } \bar{x} \text{ за } \leq t \text{ стъпки } \iff (Q_n(a, \bar{x}, t))_0 > lh(a),$$

или все едно

$$(a, \bar{x}, t) \in RHP_n \iff (Q_n(a, \bar{x}, t))_0 > lh(a).$$

Следователно множеството RHP_n е разрешимо. □

4.3.3 Теорема на Райс-Успенски

Тук основната ни задача ще бъде да изследваме важни *семантични* свойства на програмите от нашия изчислителен модел и да ги класифицираме в зависимост от тяхната сложност — разрешими, полуразрешими, неразрешими, неполуразрешими и пр. Ще разглеждаме само програми с една входна променлива, тъй като всички резултати, които са или не са в сила за тези програми, съответно са или не са в сила и за програмите с n входни променливи.

Ще ни интересуват т.нар. *масови* проблеми, т.е. задачи, отнасящи се до *безкрайни* съвкупности от обекти — функции, множества, програми. Разбира се, това, че даден проблем е алгоритмично неразрешим, съвсем не означава, че в някои *частни случаи* той не може да бъде разрешен. Алгоритмичната неразрешимост означава, че няма *общ метод*, който да работи за *всички* случаи.

Например, ако се интересуваме от проблема дали произволна програма P_a спира за всеки вход x, в някои конкретни случаи бихме могли да си отговаряме на този въпрос — например, ако в програмата няма цикли, е ясно, че тя ще завършва винаги. Това, което ни интересува, е дали съществува алгоритъм, който работи при *всеки* конкретен случай.

Да напомним, че с $\mathcal{C}_1$ означаваме съвкупността от всички едноместни изчислими функции, т.е.

$$\mathcal{C}_1 = \{\varphi_0, \varphi_1, \ldots\}.$$

Да разглеждаме някакво *семантично* свойства на програмите за МНР с една входна променлива означава да разглеждаме свойство на *функциите*, които тези програми пресмятат, т.е. свойство на функциите от

класа $\mathcal{C}_1$. Всяко такова свойство може да се зададе със съответно множество $\mathcal{A} \subseteq \mathcal{C}_1$ — това е множеството от всички функции, имащи това свойство. Следователно проблемът дали една функция от $\mathcal{C}_1$ има свойството P е еквивалентен на проблема дали тази функция принадлежи на множеството $\mathcal{A} = \{f \mid f \text{ има свойството } P\}$. Формално този проблем ще записваме така:

$$"\varphi_a \in \mathcal{A}?"$$

Например проблемът дали P_a завършва за всяко x е проблем дали φ_a е тотална, или все едно — дали φ_a принадлежи на класа

$$\mathcal{A}_{tot} = \{f \mid f \text{ е тотална и изчислима}\}.$$

Следователно проблемът дали P_a завършва за всяко x може да се запише така: "$\varphi_a \in \mathcal{A}_{tot}$?".

Друг пример е *проблемът за коректността*, т.е. дали програмата P_a пресмята конкретна функция f_0. Него можем да формализираме така: имаме

$$P_a \text{ пресмята } f_0 \iff \varphi_a = f_0 \iff \varphi_a \in \mathcal{A}_0 \stackrel{\text{деф}}{=} \{f_0\}.$$

Значи проблемът за коректността е проблемът "$\varphi_a \in \mathcal{A}_0$?".

Нека $\mathcal{A} \subseteq \mathcal{C}_1$. *Индексното множество* $I_{\mathcal{A}}$ на $\mathcal{A}$ дефинираме по следния начин:

$$I_{\mathcal{A}} \;=\; \{\; a \mid \varphi_a \in \mathcal{A} \;\}.$$

С други думи, в $I_{\mathcal{A}}$ влизат всевъзможните индекси на всяка функция от $\mathcal{A}$. Да отбележим, че $I_{\mathcal{A}}$ вече е множество от *числа*. Това ни дава възможност да говорим за *сложност* на проблема "$\varphi_a \in \mathcal{A}$?", като разглеждаме сложността на индексното му множество $I_{\mathcal{A}}$.

Определение 4.7. Ще казваме, че проблемът "$\varphi_a \in \mathcal{A}$?" *е разрешим (полуразрешим, неразрешим),* ако индексното му множество $I_{\mathcal{A}}$ е разрешимо (полуразрешимо, неразрешимо).

Ще казваме, че един проблем от вида "$\varphi_a \in \mathcal{A}$?" е *нетривиален*, ако има поне една функция, която е в $\mathcal{A}$, и поне една извън $\mathcal{A}$, с други думи, ако $\emptyset \subsetneqq \mathcal{A} \subsetneqq \mathcal{C}_1$.
Следващата теорема показва, че всички проблеми от вида "$\varphi_a \in \mathcal{A}$?", с изключение на два — за $\mathcal{A} = \emptyset$ и $\mathcal{A} = \mathcal{C}_1$, са неразрешими.

Теорема 4.3. (Теорема на Райс-Успенски) Нека $\emptyset \subsetneqq \mathcal{A} \subsetneqq \mathcal{C}_1$. Тогава проблемът "$\varphi_a \in \mathcal{A}$?" е неразрешим.

Доказателство. По-краткото (но и по-малко информативно) доказателство минава през втората теорема за рекурсия.

Да допуснем, че проблемът "$\varphi_a \in \mathcal{A}$?" е разрешим, т.е. множеството $I_{\mathcal{A}} = \{a \mid \varphi_a \in \mathcal{A}\}$ е разрешимо. Да фиксираме два индекса b и c, такива че $\varphi_b \in \mathcal{A}$ и $\varphi_c \notin \mathcal{A}$, и да разгледаме функцията

$$h(a) = \begin{cases} c, & \text{ако } \varphi_a \in \mathcal{A} \\ b, & \text{ако } \varphi_a \notin \mathcal{A}. \end{cases}$$

Условието $\varphi_a \in \mathcal{A}$ е еквивалентно на $a \in I_{\mathcal{A}}$, което е разрешимо по допускане, и значи h е рекурсивна функция. Тогава ще съществува индекс a, такъв че

$$\varphi_a \;=\; \varphi_{h(a)}. \tag{4.2}$$

Да разгледаме двете възможности за стойностите на $h(a)$:

1 сл.: $h(a) = b$, което означава, че $\varphi_a \notin \mathcal{A}$. Тогава от равенството (4.2) ще имаме $\varphi_a = \varphi_b$ и следователно $\varphi_a \in \mathcal{A}$ — противоречие.

2 сл.: $h(a) = c$, което съгласно дефиницията на h означава, че $\varphi_a \in \mathcal{A}$. Тогава отново от (4.2) ще имаме $\varphi_a = \varphi_c$, и понеже $\varphi_c \notin \mathcal{A}$, то и $\varphi_a \notin \mathcal{A}$ — пак противоречие, което показва, че допускането ни, че проблемът "$\varphi_a \in \mathcal{A}$?" е разрешим, е погрешно.

Друг начин да покажем, че проблемът "$\varphi_a \in \mathcal{A}$?" е неразрешим, е чрез *свеждане*. По-точно, ще покажем, че $K \leqslant I_{\mathcal{A}}$, което не само ще следва, че множеството $I_{\mathcal{A}}$ е неразрешимо, но то ще ни казва още, че сложността на $I_{\mathcal{A}}$ е над тази на K (в частност, ако $I_{\mathcal{A}}$ е полуразрешимо, то ще е със сложността на K).

Без ограничение на общността можем да си мислим, че $\emptyset^{(1)} \notin \mathcal{A}$, защото ако не е така, можем да разгледаме проблема "$\varphi_a \in \overline{\mathcal{A}}$?" и за него да покажем, че не е разрешим. Тогава множеството $I_{\overline{\mathcal{A}}}$ няма да е разрешимо, откъдето и $I_{\mathcal{A}}$, като негово допълнение, също няма да е разрешимо.

Нека отново приемем, че b е индекс на функция от класа $\mathcal{A}$, т.е. $b \in I_{\mathcal{A}}$. Да дефинираме функцията f по следния начин:

$$f(a,x) \simeq \begin{cases} \varphi_b(x), & \text{ако } a \in K \\ \neg!, & \text{иначе.} \end{cases}$$

Тази функция очевидно е изчислима, значи към нея можем да приложим S^m_n-теоремата. Така ще получим, че има рекурсивна функция h, такава че за всяко a и x, $\varphi_{h(a)}(x) \simeq f(a,x)$. Оттук веднага

$$\varphi_{h(a)} = \begin{cases} \varphi_b, & \text{ако } a \in K \\ \emptyset^{(1)}, & \text{иначе.} \end{cases}$$

Като имаме предвид, че $\varphi_b \neq \emptyset^{(1)}$, лесно съобразяваме, че

$$a \in K \iff \varphi_{h(a)} \in \mathcal{A} \iff h(a) \in I_{\mathcal{A}},$$

което означава, че $K \underset{h}{\leqslant} I_{\mathcal{A}}$. И тъй като K не е разрешимо, по *Твърдение* 4.18 и $I_{\mathcal{A}}$ няма да е разрешимо. □

Забележка. Да обърнем внимание, че случаите за $\mathcal{A}$, които се изключват от горната теорема — $\mathcal{A} = \emptyset$ и $\mathcal{A} = \mathcal{C}_1$, са очевидно разрешими, защото индексните им множества са $\emptyset$ или $\mathbb{N}$, съответно.

Да отбележим, че ако индексното множество $I_{\mathcal{A}}$ е полуразрешимо, то от $I_{\mathcal{A}} \leqslant K$ (*Задача* 4.23) и свеждането $K \leqslant I_{\mathcal{A}}$, до което достигнахме по-горе, ще получим общо $I_{\mathcal{A}} \equiv K$. Това означава, че ако един проблем "$\varphi_a \in \mathcal{A}$?" е полуразрешим, то индексното му множество $I_{\mathcal{A}}$ е от най-сложните полуразрешими множества.

От доказателството на теоремата на Райс-Успенски (по втория начин) се вижда още едно полезно следствие:

Следствие 4.6. Нека класът от функции $\mathcal{A} \subsetneqq \mathcal{C}_1$ е такъв, че $\emptyset^{(1)} \in \mathcal{A}$. Тогава проблемът "$\varphi_a \in \mathcal{A}$?" не е полуразрешим.

Доказателство. Разглеждаме допълнителния проблем "$\varphi_a \in \overline{\mathcal{A}}$?". Той очевидно е непразен (защото $\mathcal{A} \subsetneqq \mathcal{C}_1$), а също и $\overline{\mathcal{A}} \neq \mathcal{C}_1$ (защото $\emptyset^{(1)} \in \mathcal{A}$). За него имаме още, че $\emptyset^{(1)} \notin \overline{\mathcal{A}}$ и следователно $K \leqslant I_{\overline{\mathcal{A}}}$. Тогава $\overline{K} \leqslant I_{\mathcal{A}}$, и понеже $\overline{K}$ не е полуразрешимо, то и $I_{\mathcal{A}}$ не може да е полуразрешимо. □

Като директно приложение на теоремата на Райс-Успенски да си дадем сметка, че са неразрешими следните проблеми:

Задача 4.38. Докажете, че следните проблеми са неразрешими:

1) *Проблемът за тоталността*: дали програмата P_a завършва за всяко x, т.е. проблемът "φ_a е тотална?"

2) *Проблемът за ограничената тоталност*: дали P_a завършва за всяко $x \leq c$, където c е фиксирано, който се записва формално като "$\forall x_{x \leq c}\, !\varphi_a(x)$?"

3) Проблемът дали програмата P_a спира при безкрайно много входове, те дали "$Dom(\varphi_a)$ е безкрайно?"

4) Проблемът за коректността, т.е. дали програмата P_a пресмята фиксирана функция f_0, или все едно "$\varphi_a = f_0$?"

5) Проблемът дали програмата P_a пресмята примитивно рекурсивна функция, или все едно "$\varphi_a \in \mathcal{PR}_1$?", където $\mathcal{PR}_1$ е класът на всички едноместни примитивно рекурсивни функции.

Решение. Единственото, което трябва да съобразим е, че всички тези проблеми са нетривиални, което е съвсем очевидно ⌣̈. Тогава по теоремата на Райс-Успенски те автоматично ще бъдат неразрешими. □

Задача 4.39. Докажете, че следните проблеми не са полуразрешими:

1) "φ_a е крайна функция?"

2) "φ_a не е тотална?"

3) "$\varphi_a = \emptyset^{(1)}$?"

4) За фиксирано $c \in \mathbb{N}$, "$\exists x_{x \leq c}\, \neg!\varphi_a(x)$?"

Решение. Лесно се съобразява, че всеки от тези проблеми е в сила за никъде недефинираната функция $\emptyset^{(1)}$. Освен това никой от тях не е валиден *за всяка* функция. Тогава съгласно *Следствие* 4.6 никой от тях не е полуразрешим. □

Да припомним, че програмите P_a и P_b са еквивалентни ($P_a \approx P_b$), ако пресмятат една и съща функция, т.е. ако $\varphi_a = \varphi_b$. Индексното множество на проблема "$\varphi_a = \varphi_b$?" е множеството $E = \{(a, b) \mid \varphi_a = \varphi_b\}$.

Твърдение 4.23. (Проблемът за еквивалентността на МНР програмите не е полуразрешим) Множеството

$$E = \{(a, b) \mid \varphi_a = \varphi_b\}$$

не е полуразрешимо.

Доказателство. Да допуснем, че E е полуразрешимо. Твърдим, че тогава ще бъде полуразрешимо и множеството

$$E_0 = \{a \mid \varphi_a = \emptyset^{(1)}\}.$$

Това звучи правдоподобно, защото проблемът "$\varphi_a = \emptyset^{(1)}$?" е частен случай на проблема "$\varphi_a = \varphi_b$?", но да го съобразим все пак.

Наистина, имаме, че $C_{E_0}(a) \simeq C_E(a, b_0)$, където b_0 е фиксиран индекс на $\emptyset^{(1)}$, следователно C_{E_0} също е изчислима, и значи E_0 наистина е полуразрешимо.

Обаче множеството E_0 е индексното множество на проблема "$\varphi_a = \emptyset^{(1)}$?", който не е полуразрешим, съгласно *Задача* 4.39 3). Полученото противоречие показва, че и E не може да бъде полуразрешимо. □

Следват две дефиниции за коректност на програма относно дадена спецификация. Целта ни е да покажем, че проблемът за коректността не е полуразрешим.

Нека P е произволна програма с една входна променлива, а $I(x)$ и $O(x, y)$ са съответно някакво *входно* и някакво *изходно условие* за P (или *спецификация* за P).

Казваме, че програмата P е *частично коректна* относно дадената спецификация $I\,/\,O$, ако е изпълнено, че за всеки вход x, такъв че входното условие $I(x)$ е в сила, *ако* програмата завърши върху x с резултат y, то този резултат ще е коректен, т.е ще е изпълнено изходното условие $O(x, y)$.

Функцията f, която тази програма пресмята, очевидно ще удовлетворява свойството $P_{p.c.}(f)$, което се дефинира по следния начин:

$$P_{p.c.}(f) \iff \forall x(\ I(x)\ \&\ !f(x) \implies O(x, f(x))\ ).$$

Програмата P е *тотално коректна* относно спецификацията $I\,/\,O$, ако при всеки коректен вход тя *задължително* завършва и резултатът отново е коректен, разбира се. Тогава за функцията f, която тя пресмята, ще е изпълнено свойството $P_{t.c.}$, където

$$P_{t.c.}(f) \iff \forall x(\ I(x) \implies !f(x)\ \&\ O(x, f(x))\ ).$$

Задача 4.40. Докажете, че всеки нетривиален проблем за частична коректност на МНР програмите не е полуразрешим.

Решение. Да фиксираме някаква спецификация $I\,/\,O$ и нека

$$\mathcal{A}_{p.c.} = \{f \mid f \text{ е изчислима } \&\ P_{p.c.}(f)\}.$$

Тогава

$$P_a \text{ е частично коректна относно } I\,/\,O \iff \varphi_a \in \mathcal{A}_{p.c.}.$$

При смислена спецификация със сигурност проблемът "$\varphi_a \in \mathcal{A}_{p.c.}$?" ще е нетривиален. Пример за тривиален проблем е да кажем този с входно условие $I(x) \iff x = x + 1$. Това условие е тъждествената *лъжа*, и следователно свойството свойствата $P_{p.c.}(f)$ и $P_{t.c.}(f)$ ще са в сила за всяка функция f, т.е. ще имаме $\mathcal{A}_{p.c.} = \mathcal{C}_1$ и $\mathcal{A}_{t.c.} = \mathcal{C}_1$.

Лесно се вижда, свойството $P_{p.c.}$ е в сила за $\emptyset^{(1)}$. Следователно ако $\mathcal{A}_{p.c.} \neq \mathcal{C}_1$, проблемът за частичната коректност няма да е полуразрешим, съгласно *Следствие* 4.6. □

Забележка. Проблемът за тоталната коректност също не е полуразрешим (за нетривиалните спецификации), но това ще следва от един по-общ критерий за неполуразрешимост, който предстои да докажем — теоремата на Райс-Шапиро.

Задачата по-горе показва, че не може да се съществува универсална програма-верификатор, която да проверява дали всяка МНР програма е коректна относно дадена спецификация.

4.3.4 Теорема на Райс-Шапиро

Вече се убедихме, че всеки нетривиален проблем от вида "$\varphi_a \in \mathcal{A}$?" е неразрешим. Нека сега да видим, че съществуват поне *полуразрешими* проблеми от този тип. Следващата задача ще ни подскаже, че тези проблеми трябва да отговарят на едно специално условие.

Задача 4.41. Докажете, че проблемът "φ_a не е инективна?" е полуразрешим.

Решение. По определение една *частична* функция $f : \mathbb{N} \rightharpoonup \mathbb{N}$ е инективна, ако е изпълнено условието:

$$\forall x \forall y (x \neq y \ \& \ !f(x) \ \& \ !f(y) \implies f(x) \neq f(y)).$$

Тогава

$$\begin{aligned} \varphi_a \text{ не е инективна} &\iff \exists x \exists y \exists z (x \neq y \ \& \ \varphi_a(x) \simeq z \ \& \ \varphi_a(y) \simeq z) \\ &\iff \exists x \exists y \exists z \underbrace{\chi_{\neq}(x,y) + |\Phi_1(a,x) - z| + |\Phi_1(a,y) - z|}_{f(a,x,y,z)} \simeq 0 \end{aligned}$$

f е изчислима, значи множеството $A = \{(a,x,y,z) \mid f(a,x,y,z) \simeq 0\}$ е полуразрешимо, а оттам по теоремата за проекцията (приложена трикратно) ще е полуразрешимо и множеството $I = \{a \mid \varphi_a \text{ не е инективна}\}$. Следователно проблемът "φ_a не е инективна?" е полуразрешим. □

Следващата теорема ни дава *необходимо* (но недостатъчно) условие един проблем от типа "$\varphi_a \in \mathcal{A}$?" да е полуразрешим. Затова няма как да я използваме, за да доказваме, че даден проблем е полуразрешим. Но това няма да бъде и нашата цел; ние ще я прилагаме с контрапозиция, за да показваме, че даден проблем *не* е полуразрешим.

Теорема 4.4. (Теорема на Райс-Шапиро) Нека проблемът "$\varphi_a \in \mathcal{A}$?" е полуразрешим. Тогава за всяка функция $f \in \mathcal{C}_1$ е в сила еквивалентността:

$$f \in \mathcal{A} \iff \exists \theta (\theta \text{ е крайна } \& \ \theta \subseteq f \ \& \ \theta \in \mathcal{A}). \tag{4.3}$$

Доказателство. Условието (4.3) ни казва, че за да е полуразрешим даден проблем "$\varphi_a \in \mathcal{A}$?", той трябва да се локализира до някоя крайна част θ от функцията $f \in \mathcal{A}$.

Да вземем за пример свойството "неинективност" от *Задача* 4.41. За да не е инективна една функция f, е достатъчно да има две различни точки x и y от дефиниционната ѝ област, за които $f(x) = f(y)$. Значи в качеството на $\theta \subseteq f$ можем да вземем функцията θ, която е дефинирана само в тези две точки.

Най-напред да се заемем с правата посока на еквивалентността (4.3). За целта да фиксираме изчислима функция $f \in \mathcal{A}$ и да допуснем, че условието в дясната страна на (4.3) не е изпълнено. Това означава, че за всяка крайна $\theta \subseteq f$ ще е вярно, че $\theta \notin \mathcal{A}$. Да видим, че тогава $\overline{K} \leqslant I_{\mathcal{A}}$, което ще противоречи на полуразрешимостта на $I_{\mathcal{A}}$.

От третото НДУ за полуразрешимост (*Твърдение* 4.10), приложено за полуразрешимото множество K, ще съществува рекурсивна функция ρ, такава че за всяко a:

$$a \in K \iff \exists t\ \rho(a,t) = 0. \tag{4.4}$$

Чрез тази функция ρ дефинираме функцията g по следния начин:

$$g(a,x) \simeq \begin{cases} f(x), & \text{ако } \forall t \leq x\ \rho(a,t) > 0 \\ \neg!, & \text{ако } \exists t \leq x\ \rho(a,t) = 0. \end{cases}$$

Тъй като f е изчислима, а условието $\forall t \leq x\ \rho(a,t) > 0$ е разрешимо, то и g ще е изчислима. S_n^m-теоремата, приложена към нея, ще ни даде рекурсивна функция h, такава че за всяко a и x,

$$\varphi_{h(a)}(x) \simeq g(a,x). \tag{4.5}$$

Нашата цел ще бъде да покажем, че $\overline{K}$ се свежда към $I_{\mathcal{A}}$ чрез горната функция h (т.е. $a \in \overline{K} \iff h(a) \in I_{\mathcal{A}}$).
Да разгледаме най-напред случая $a \in \overline{K}$. Тогава от (4.4) ще имаме, че $\forall t\ \rho(a,t) > 0$. Но това означава, че $g(a,x) \simeq f(x)$ за всяко x. Тогава според (4.5) ще имаме, че

$$\varphi_{h(a)}(x) \simeq f(x)$$

за всяко x и значи $\varphi_{h(a)} = f$. Но съгласно нашия избор, $f \in \mathcal{A}$, с други думи, $\varphi_{h(a)} \in \mathcal{A}$ и значи $h(a) \in I_{\mathcal{A}}$. Така получихме, че

$$a \in \overline{K} \implies h(a) \in I_{\mathcal{A}}. \tag{4.6}$$

Нека сега $a \in K$. Тогава от (4.4) ще имаме, че $\rho(a,t) = 0$ за някое t. Нека t_0 е първото с това свойство. Лесно се вижда, че

$$\varphi_{h(a)}(x) \simeq g(a,x) \simeq \begin{cases} f(x), & \text{ако } x < t_0 \\ \neg!, & \text{иначе.} \end{cases}$$

Следователно в този случай $\varphi_{h(a)}$ е крайна и очевидно $\varphi_{h(a)} \subseteq f$. Но по допускане никоя такава функция не може да е в класа $\mathcal{A}$, т.е. имаме $\varphi_{h(a)} \notin \mathcal{A}$ и съответно — $h(a) \notin I_{\mathcal{A}}$. Така стигнахме до импликацията

$$a \in K \implies h(a) \notin I_{\mathcal{A}},$$

която заедно с обратната ѝ импликация (4.6) ни дава общо $a \in \overline{K} \iff h(a) \in I_{\mathcal{A}}$. Това означава, че $\overline{K} \underset{h}{\leqslant} I_{\mathcal{A}}$ — противоречие с факта, че $I_{\mathcal{A}}$ е полуразрешимо.

Сега да проверим и обратната посока на еквивалентността (4.3). Нека за изчислимата функция f е вярно, че съществува крайна функция θ, такава че $\theta \subseteq f$ и $\theta \in \mathcal{A}$. Да допуснем, че $f \notin \mathcal{A}$. Целта ни е отново да стигнем до свеждането $\overline{K} \leqslant I_{\mathcal{A}}$, което ще ни даде търсеното противоречие.

Сега дефинираме функцията g както следва:

$$g(a,x) \simeq \begin{cases} f(x), & \text{ако } x \in Dom(\theta) \ \vee \ a \in K \\ \neg!, & \text{иначе.} \end{cases}$$

Да означим с A множеството $\{(a,x) \mid x \in Dom(\theta) \ \vee \ a \in K\}$ от дефиницията на g. Тогава

$$(a,x) \in A \iff x \in Dom(\theta) \vee a \in K \iff (a,x) \in (\mathbb{N} \times Dom(\theta)) \cup (K \times \mathbb{N}).$$

Така $A = (\mathbb{N} \times Dom(\theta)) \cup (K \times \mathbb{N})$ и значи A е полуразрешимо, следователно g е изчислима. Отново прилагаме S_n^m-теоремата и получаваме, че има рекурсивна функция h, такава че за всяко a и x:

$$\varphi_{h(a)}(x) \simeq g(a,x).$$

Ако $a \in K$, тогава за всяко x, $g(a,x) \overset{\text{деф}}{\simeq} f(x)$, което според горното равенство означава, че за всяко x, $\varphi_{h(a)}(x) \simeq f(x)$, т.е. $\varphi_{h(a)} = f \notin \mathcal{A}$. Тогава $h(a) \notin I_{\mathcal{A}}$, или дотук проверихме, че

$$a \in K \implies h(a) \notin I_{\mathcal{A}}.$$

Нека сега $a \notin K$. Тогава от дефиницията на g ще имаме:

$$\varphi_{h(a)}(x) \simeq g(a,x) \simeq \begin{cases} f(x), & \text{ако } x \in Dom(\theta) \\ \neg!, & \text{иначе.} \end{cases}$$

Понеже $\theta \subseteq f$, излезе, че в този случай $\varphi_{h(a)} = \theta$, като $\theta \in \mathcal{A}$, т.е. получихме, че $\varphi_{h(a)} \in \mathcal{A}$ и съответно — $h(a) \in I_{\mathcal{A}}$. Така проверихме импликацията $a \in \overline{K} \implies h(a) \in I_{\mathcal{A}}$, която заедно с обратната импликация от по-горе, ни дава общо $a \in \overline{K} \iff h(a) \in I_{\mathcal{A}}$ и търсеното противоречие. □

От тази теорема на момента получаваме, че проблемът за тоталността не е полуразрешим, защото няма крайни функции, за които той да е в сила:

Задача 4.42. Докажете, че проблемът "$\varphi_a \in \mathcal{A}_{tot}$?" не е полуразрешим.

Решение. Ако допуснем, че е полуразрешим, то трябва да е изпълнено условието (4.3). Но няма крайна функция, която принадлежи на $\mathcal{A}_{tot}$, и значи допускането ни е погрешно. □

От теоремата на Райс-Шапиро получаваме следното удобно за използване ДУ за неполуразрешимост:

Следствие 4.7. Нека проблемът "$\varphi_a \in \mathcal{A}$?" е полуразрешим. Тогава класът $\mathcal{A}$ е затворен нагоре, т.е. за всяка $f, g \in \mathcal{C}_1$ е в сила импликацията:

$$f \in \mathcal{A} \;\&\; f \subseteq g \implies g \in \mathcal{A}.$$

Доказателство. Нека $f \in \mathcal{A}$. Тогава от еквивалентността (4.3), прочетена в правата посока, ще имаме, че за някоя крайна функция $\theta \subseteq f$ ще е вярно, че $\theta \in \mathcal{A}$. Сега ако $f \subseteq g$, по транзитивността на $\subseteq$ получаваме, че $\theta \subseteq g$, и оттук отново чрез (4.3), но този път приложена в обратната посока, достигаме до $g \in \mathcal{A}$. □

Оттук лесно се вижда как теоремата на Райс-Успенски следва леко от теоремата на Райс-Шапиро:

Задача 4.43. Изведете теоремата на Райс-Успенски като следствие на теоремата на Райс-Шапиро.

Решение. Нека $\emptyset \subsetneqq \mathcal{A} \subsetneqq \mathcal{C}_1$ и да приемем, че проблемът "$\varphi_a \in \mathcal{A}$?" е разрешим. Тогава и двата проблема — "$\varphi_a \in \mathcal{A}$?" и "$\varphi_a \in \overline{\mathcal{A}}$?" ще са полуразрешими. Разглеждаме поотделно двата случая:
1 сл.: $\emptyset^{(1)} \in \mathcal{A}$. Тъй като за всяка функция $g \in \mathcal{C}_1$ е вярно, че $\emptyset^{(1)} \subseteq g$, то от горното следствие веднага следва, че $\mathcal{A} = \mathcal{C}_1$, което противоречи на условието $\mathcal{A} \subsetneqq \mathcal{C}_1$.
2 сл.: $\emptyset^{(1)} \notin \mathcal{A}$. Но тогава $\emptyset^{(1)} \in \overline{\mathcal{A}}$ и разсъждавайки както в сл. 1, ще получим, че $\overline{\mathcal{A}} = \mathcal{C}_1$, откъдето $\mathcal{A} = \emptyset$ — отново противоречие с условието.
Следователно допускането ни, че проблемът "$\varphi_a \in \mathcal{A}$?" е разрешим, е било погрешно. □

От това следствие с контрапозиция получаваме, че ако $\mathcal{A}$ не е затворен нагоре, то проблемът "$\varphi_a \in \mathcal{A}$?" не е полуразрешим. Така можем да покажем, че не са полуразрешими редица проблеми.

Задача 4.44. Нека f_0 е фиксирана едноместна изчислима функция. Докажете, че проблемът "$\varphi_a = f_0$?" не е полуразрешим.

Решение. Този проблем можем да разпишем като "$\varphi_a \in \mathcal{A}$?", където $\mathcal{A} = \{f_0\}$. Да видим, че той не е полуразрешим. Ще разгледаме следните два случая за f_0:

1 сл.: f_0 е крайна. Тогава според горното *Следствие* 4.7, класът $\mathcal{A}$ трябва да е затворен нагоре, а той очевидно не е — противоречие.

2 сл.: f_0 не е крайна. Тогава според (4.3) класът $\mathcal{A}$ трябва да съдържа крайна функция, а той очевидно не съдържа такава — отново противоречие. □

Видяхме, че проблемът "$\varphi_a = f_0$?" не е полуразрешим. Дали това ще бъде вярно и за допълнителния проблем "$\varphi_a \neq f_0$?"? Оказва се, че не е.

Задача 4.45. Има ли изчислима функция f_0, за която проблемът "$\varphi_a \neq f_0$?" да е полуразрешим?

Решение. Нека $\mathcal{A} = \{f \mid f \neq f_0\}$. Тогава

$$\varphi_a \neq f_0 \iff \varphi_a \in \mathcal{A}$$

и можем да разсъждаваме за $\mathcal{A}$.

Ако f_0 не е $\emptyset^{(1)}$, то $\emptyset^{(1)} \in \mathcal{A}$ и следователно проблемът "$\varphi_a \neq f_0$?" няма да е полуразрешим, съгласно *Следствие* 4.6.

Ако $f_0 = \emptyset^{(1)}$, се оказва, че проблемът "$\varphi_a \neq f_0$?" , т.е "$\varphi_a \neq \emptyset^{(1)}$?", вече е полуразрешим. Наистина, в този случай ще имаме, че за произволно a:

$$a \in I_{\mathcal{A}} \iff \varphi_a \neq \emptyset^{(1)} \iff \exists x\ !\varphi_a(x) \iff \exists x\ !\Phi_1(a, x)$$

и значи $I_{\mathcal{A}}$ е полуразрешимо множество. □

Глава 5

Аксиоматична теория на сложността

Възниква през 60-те години на XX век в дисертацията на Мануел Блум. Известна е още като *Машинно независима теория на сложността*. При нея не се предполага конкретен изчислителен модел. Единственото, което се изисква, е да има ефективна номерация

$$P_0,\ P_1,\ \ldots$$

на всички програми от този модел. Мерките за сложност, които се разглеждат, се наричат абстрактни мерки за сложност и се въвеждат с двете *аксиоми на Блум*.

За разлика от Аксиоматичната теория на сложността, в *Структурната теория на сложността* се разглежда конкретен изчислителен модел — този на машините на Тюринг, както и две конкретни мерки за сложност — времевата и пространствената мярка.

5.1 Аксиоми на Блум

Въпреки че в аксиоматичната теория на сложността изчислителният модел няма значение, за удобство ще предполагаме, че продължаваме да работим в нашия модел с МНР, който изучавахме дотук, т.е. навсякъде под "програма" ще разбираме програма за МНР. Принципно ще разглеждаме програми с една входна променлива.

Определение 5.1. Функцията $M\colon\ \mathbb{N}\times\mathbb{N} \multimap \mathbb{N}$ наричаме *абстрактна мярка за сложност*, ако са изпълнени условията:

B1: за всяко a и x: $\ !M(a,x) \iff P_a(x)\downarrow$;

B2: множеството $R = \{(a,x,y) \mid M(a,x) \simeq y\}$ е разрешимо.

Ако $!M(a,x)$, числото $M(a,x)$ ще тълкуваме като M-ресурса, необходим при работата на P_a върху x. Така например $M(a,x)$ може да бъде времето за работа на P_a върху x, или паметта, която се използва при това изчисление и пр. Детайлите ще уточним след малко в раздел 5.2. Сега да обърнем внимание на факта, че говорим за сложност на *програмата* P_a, а не на функцията φ_a. Както ще отбележим по-нататък, някои функции нямат най-добра сложност.

От аксиома B1 се вижда, че $Dom(M) = Dom(\Phi_1)$, а от аксиома B2 следва, че графика на M (която всъщност е множеството R), е разрешима. В частност, тя е и полуразрешима, и съгласно теоремата за графиката, функцията M ще е изчислима. Оттук следва, че функцията "абстрактна мярка" е много особена функция — тя има проста (разрешима) графика и с възможно най-сложния домейн — този на универсалната функция. Дали въобще съществуват такива функции? Да, многобройните примери за абстрактни мерки, които ще дадем по-долу, ще го потвърдят.

Вместо аксиомата B2 понякога е по-удобно да използваме следната аксиома B2′:

B2′ : множеството $R' = \{(a,x,y) \mid M(a,x) \leq y\}$ е разрешимо.

Разрешимостта на R' означава, че има да/не алгоритъм, който за всяка програма P_a, за всеки вход x и за всяко $y \in \mathbb{N}$ ни казва дали абстрактният ресурс M, който е в количество y, ще "стигне" за работата на P_a върху x (това се разбира под $M(a,x) \leq y$). Оказва се, че условието ресурсът да "стигне" е еквивалентно на условието да "стигне точно", което всъщност беше смисълът на аксиома B2. Да се убедим:

Твърдение 5.1. Аксиомите B2 и B2′ са еквивалентни.

Доказателство. Нека е в сила B2, т.е. $R = \{(a,x,y) \mid M(a,x) \simeq y\}$ е разрешимо множество. Тогава условието за принадлежност към R' на произволна тройка $(a,x,y) \in \mathbb{N}^3$ можем да изразим чрез принадлежност към R по следния начин:

$$(a,x,y) \in R' \iff \exists z_{z \leq y}\ (a,x,z) \in R.$$

Тъй като ограниченият квантор за съществуване запазва разрешимостта (*Твърдение* 4.2), то и R' ще е разрешимо.

Обратно, ако $R' = \{(a,x,y) \mid M(a,x) \leq y\}$ е разрешимо, то за произволни $(a,x,y) \in \mathbb{N}^3$ ще имаме:

$$(a,x,y) \in R \iff (a,x,y) \in R' \ \&\ (\underbrace{y > 0 \implies (a,x,y-1) \notin R'}_{y=0\ \vee\ (a,x,y-1) \notin R'}).$$

Следователно множеството R също е разрешимо. □

5.2 Примери за мерки за сложност

При фиксирано a, да означим с M_a функцията, която за за всяко x се дефинира като:

$$M_a(x) \overset{\text{деф}}{\simeq} M(a,x).$$

Функцията M_a ще тълкуваме като *сложност на програмата P_a относно мярката M*. От аксиома B1 имаме, че

$$!M_a(x) \iff !M(a,x) \iff P_a(x)\downarrow,$$

т.е. M_a е дефинирана в точките, в които P_a спира.

Следващите няколко примера показват, че двете естествени понятия за сложностни мерки — памет и време, удовлетворяват аксиомите на Блум.

Твърдение 5.2. Следните функции са мерки за сложност:

1) *Времевата* мярка за сложност

$$T(a,x) \simeq \begin{cases} \text{броя стъпки, за които } P_a \text{ спира върху } x, & \text{ако } P_a(x)\downarrow \\ \neg!, & \text{иначе.} \end{cases}$$

Доказателство. Аксиомата B1 очевидно е в сила.

Аксиома B2 интуитивно е ясна: за да разберем дали P_a ще спре върху x за t такта, я пускаме да работи за тези t такта и гледаме дали междувременно е спряла или не.

За строгото доказателство можем да се възползваме от еквивалентната аксиома B2′, т.е. да покажем, че е разрешимо множеството

$$R' = \{(a,x,t) \mid P_a \text{ спира върху } x \text{ за } \leq t \text{ такта}\}.$$

Всъщност R' е същото като множеството RHP_1 (ограничения стоп-проблем), който е разрешим, съгласно *Твърдение* 4.22. Следователно и R' е разрешимо, и значи аксиома B2 също е вярна за тази мярка. □

2) Времевата мярка T^*, която използва физическото време, а не абстрактните тактове от изчислението на P_a:

$$T^*(a,x) \simeq \begin{cases} \text{времето (примерно в сек), за което } P_a \text{ спира върху } x, & \text{ако } P_a(x)\downarrow \\ \neg!, & \text{иначе.} \end{cases}$$

Доказателство. Разсъждаваме както при предишната времева мярка. □

3) *Пространствената* мярка за сложност

$$L(a,x) \simeq \begin{cases} \text{мах стойност на регистър при работата на } P_a \text{ върху } x, & \text{ако } P_a(x)\downarrow \\ \neg!, & \text{иначе.} \end{cases}$$

Доказателство. Аксиомата B1 отново е ясна. За да проверим B2, да фиксираме произволна програма P_a. Нека тя има q на брой оператора, а m е максималният номер на регистър, който се среща в P_a, т.е. всички регистри, които се срещат в P_a са сред $X_1, \ \ldots \ , X_m$.

Можем да си мислим, че паметта на P_a е $(X_1, \ \ldots \ , X_m)$. Тогава всяка конфигурация е от вида $(l, x_1, \ldots, x_m)$. Да си дадем сметка, че броят на незаключителните конфигурации, на които съдържанието на регистрите не надвишава y, е точно

$$q(y+1)^m.$$

(За всяко x_i имаме $y+1$ възможни стойности — от 0 до y, а възможните стойностите на адресите l са колкото броя на операторите, т.е. q.) Сега нашата стратегия да разбираме дали $(a,x,y) \in R$, е следната: пускаме програмата P_a да работи върху x точно $t = q(y+1)^m + 1$ такта. Имаме две възможности: за това време P_a да е спряла или да не е спряла.

Ако е спряла, проследяваме изчислението и гледаме дали максималното съдържание на регистрите $X_1, \ \ldots \ , X_m$ в хода на изчислението е било y. Ако да, то $(a,x,y) \in R$, ако не — $(a,x,y) \notin R$.

Ако $P_a(x)$ не е спряла за тези t такта, твърдим, че $(a,x,y) \notin R$. За целта разглеждаме двата възможни случая:

1 сл. По време на изчислението в някой от регистрите се е появила стойност, която е по-голяма от y. Тогава е ясно, че $(a,x,y) \notin R$.

2 сл. За тези t такта от изчислението няма регистър, чието съдържание да надхвърля y. Тук си даваме сметка, че щом P_a не е спряла върху x, то тя не е попадала във заключително състояние. Но броят на всички незаключителни конфигурации е по-малък от броя на тактовете t и значи P_a е попаднала поне два пъти в една и съща конфигурация. Следователно тя ще зацикли върху x и значи отново $(a,x,y) \notin R$.

Разбира се, тази процедура е "равномерна" по a, защото броят q на операторите на P_a е $lh(a)+1$, а като горна граница за паметта на P_a можем да вземем самото a. С други думи, сега алгоритъмът ни започва така: пресмятаме функцията $b(a,x,y) = (lh(a)+1)(y+1)^a$. После пускаме P_a да работи върху x $b(a,x,y)+1$ такта. Доказателството, че това е достатъчно, за да можем да определим дали $(a,x,y) \in R$ вече го имаме по-горе. □

Има още много примери за ресурси, които са мерки за сложност. Ето два примера, които оставям като задача за допълнителни точки.

Задача 5.1. (Задача за ЕК) Докажете, че следните функции са мерки за сложност:

$$S(a,x) \simeq \begin{cases} \text{броя на изпълнените команди от вида } \mathcal{S}(n) & \text{ако } P_a(x)\downarrow \\ \text{при работата на } P_a \text{ върху } x, & \\ \neg!, & \text{иначе} \end{cases}$$

$$J(a,x) \simeq \begin{cases} \text{броя на изпълнените команди от вида } J(m,n,q) & \text{ако } P_a(x)\downarrow \\ \text{при работата на } P_a \text{ върху } x, & \\ \neg!, & \text{иначе.} \end{cases}$$

Функциите, за които дотук показахме, че са абстрактни мерки за сложност, в широк смисъл наистина представляваха някакъв критерий за сложност на една програма. Това може да се счита за аргумент в подкрепа на факта, че аксиомите на Блум са адекватни. Друг аргумент в тази посока е да покажем, че функции, които нямаме основание да считаме за критерий за сложност, не удовлетворяват аксиомите на Блум.

За целта се оказва полезно едно наблюдение, което съобразихме в *Задача* 4.2, а именно, че ако една функция f има разрешима графика и се мажорира от някоя рекурсивна функция b, т.е. за всяко $\bar{x}$ е изпълнено:

$$!f(\bar{x}) \implies f(\bar{x}) \leq b(\bar{x}),$$

то нейната дефиниционна област $Dom(f)$ също е разрешимо множество.

Ние ще се възползваме от този резултат така: за да покажем, че дадена функция M не е мярка, ще е достатъчно да видим, че M се мажорира от някаква рекурсивна функция. Тогава ако допуснем, че M е мярка, то по аксиома B2 тя трябва да има разрешима графика, откъдето по *Задача* 4.2 ще следва, че $Dom(M)$ е разрешимо множество — противоречие с аксиома B1. Следователно функцията M не може да е мярка.

Твърдение 5.3. 1) Не е мярка за сложност функцията

$$M_1(a,x) \stackrel{\text{деф}}{=} \text{дължината на работната памет на } P_a.$$

Доказателство. Тази функция е тотална и следователно не удовлетворява B1. □

2) Следващата функция M_2 също не е мярка:

$$M_2(a,x) \simeq \begin{cases} \text{дължината на работната памет на } P_a & \text{ако } P_a(x)\downarrow \\ \neg!, & \text{иначе.} \end{cases}$$

Доказателство. Тази функция удовлетворява B1, но за нея B2 пропада. За целта е достатъчно да забележим, че M_2 се мажорира от рекурсивната функция $b(a,x) \stackrel{\text{деф}}{=} a$, защото ако P_a има m регистъра, то със сигурност $m \leq a$. □

3) Следващата функция M_3 не е мярка, което показва, че няма безплатни изчисления ☺:

$$M_3(a,x) \simeq \begin{cases} 0, & \text{ако } P_a(x)\downarrow \\ \neg!, & \text{иначе.} \end{cases}$$

Доказателство. Тази функция се мажорира от константата 0 и съгласно *Задача* 4.2 не може да бъде абстрактна мярка. □

4) *Резултатът* от работата на P_a не е мярка, т.е. не е мярка функцията M_4, която се дефинира така:

$$M_4(a,x) \simeq \begin{cases} \varphi_a(x), & \text{ако } P_a(x)\downarrow \\ \neg!, & \text{иначе.} \end{cases}$$

Доказателство. Наистина, резултатът от едно изчисление не би трябвало да е критерий за сложността на това изчисление. За целта вземете едно сложно множество A. Тогава неговата характеристична функция χ_A ще е проста функция — със стойности само 0 и 1, и тези стойности по никакъв начин не ни говорят за сложността на изчислението на $\chi_A(x)$.

Всъщност M_4 е точно универсалната функция Φ_1. Да допуснем, че Φ_1 е мярка за сложност. Не можем да ограничим с рекурсивна функция нейните стойности, но все пак ще успеем да стигнем до противоречие с факта, че графиката ѝ е разрешимо множество.

За тази цел да разгледаме програмата P_{a_0}, която пресмята полухарактеристичната функция на множеството K. Тогава $\varphi_{a_0} = C_K$ и за всяко x ще бъде изпълнено:

$$x \in K \iff \varphi_{a_0}(x) \simeq 0 \iff (a_0, x, 0) \in G_{\Phi_1}.$$

Излезе, че K трябва да е разрешимо, а то не е, и значи допускането ни е погрешно. □

Задача 5.2. (Задача за ЕК) Докажете, че следната функция не е мярка за сложност:

$$Z(a,x) \simeq \begin{cases} \text{броят на изпълнените команди от вида } \mathcal{O}(n) & \text{ако } P_a(x)\downarrow \\ \text{при работата на } P_a \text{ върху } x, & \\ \neg!, & \text{иначе} \end{cases}$$

5.3 Теорема за рекурсивна свързаност на мерките

Ще казваме, че функцията $f(x)$ е *повишаваща*, ако е вярно, че:

1) f е растяща (може и нестрого);

2) $\lim\limits_{x \longrightarrow \infty} f(x) = \infty$.

Следващото твърдение дава начин да образуваме все нови и нови мерки за сложност, тръгвайки от дадена мярка.

Твърдение 5.4. Нека $C(a,x)$ е мярка за сложност, а f е рекурсивна повишаваща функция. Да дефинираме нова функция D както следва:

$$D(a,x) \simeq f(C(a,x))$$

за всяко a и x. Тогава функцията $D(a,x)$ също е мярка.

Доказателство. Тъй като f е тотална, а C е мярка, аксиомата B1 е в сила и за D.
За да проверим B2, да разпишем условието за принадлежност към графиката на $D = f \circ C$. Имаме

$$D(a,x) \simeq y \iff \exists z\ (\underbrace{C(a,x) \simeq z}_{\text{разрешимо}}\ \&\ \underbrace{f(z) = y}_{\text{разрешимо}}).$$

За да ограничим квантора по z, да дефинираме функцията

$$b(y)\ = \mu z[f(z) > y].$$

Тя е изчислима и тотална (второто защото f е повишаваща). Сега лесно се съобразява, че за графиката на D е изпълнено:

$$D(a,x) \simeq y \iff \exists z_{z \leq b(y)}\ (\underbrace{C(a,x) \simeq z}_{\text{разрешимо}}\ \&\ \underbrace{f(z) = y}_{\text{разрешимо}})$$

и следователно графиката на D е разрешимо множество. □

От това твърдение следва, че ако тръгнем от функция, която е мярка — примерно от времевата мярка T, то мерки ще са и функциите $2T(a,x)$, $T^2(a,x)$, $2^{T(a,x)}$ и прочее.

Оказва се, че е в сила и резултат, в известен смисъл обратен на горния. Той показва, че всеки две мерки за сложност са свързани и ако една програма P_a има малка сложност по отношение на една мярка, то тя има малка сложност и по отношение на всяка друга мярка.

Теорема 5.1. (Теорема за рекурсивната свързаност на мерките) Нека C и D са мерки за сложност. Съществува рекурсивна функция $f(x,y)$, строго растяща по y, такава че за всяко a:

$$!C_a(x) \implies C_a(x) \leq f(x, D_a(x)) \quad \text{за всяко } x \geq a;$$

$$!D_a(x) \implies D_a(x) \leq f(x, C_a(x)) \quad \text{за всяко } x \geq a.$$

Забележка. От това, че f е растяща по y следва, че ако $D_a(x)$ има малка стойност, то и $C_a(x)$, което е ограничена от $f(x, D_a(x))$ също трябва да има малка стойност. С други думи, ако по отношение на една мярка програмата P_a е проста, то по отношение на произволна друга мярка тя също трябва да е проста.

Доказателство. Да дефинираме

$$h(a,x,y) = \begin{cases} max(C(a,x), D(a,x)), & \text{ако } C_a(x) \simeq y \ \vee \ D_a(x) \simeq y \\ 0, & \text{иначе.} \end{cases}$$

Тъй като графиките на C и D са разрешими функции, то h е изчислима. Тя очевидно е и тотална, тъй че общо е рекурсивна.

Сега да вземем f да е следната функция:

$$f(x,y) \ = \ y + \max_{a \leq x}(\max_{z \leq y} h(a,x,z)). \tag{5.1}$$

Лесно се вижда, че f е рекурсивна и строго растяща по y. Да се убедим, че тя има свойството от теоремата. За целта да фиксираме a, вземем произволно $x \geq a$ и да приемем, че $!C_a(x)$. Тогава $!D_a(x)$ и

$$\begin{aligned} f(x, D_a(x)) \ &\overset{\text{деф}}{=} \ D_a(x) + \max_{i \leq x}(\max_{z \leq D_a(x)(x)} h(i,x,z)) \ \geq \max_{i \in \{0,\ldots,a,\ldots,x\}}(\max_{z \leq D_a(x)} h(i,x,z)) \\ &\geq \max_{z \leq D_a(x)} h(a,x,z) \ \geq \ h(a,x,C_a(x)) \ \overset{\text{деф}}{=} \ max(C_a(x), D_a(x)) \\ &\geq \ C_a(x). \end{aligned}$$

Аналогично се доказва и другото неравенство. □

Дали може горната теорема да бъде във вид, по-близък до предишното *Твърдение* 5.4? Може, но има допълнително условие за това: $C_a(x) \geq x$ и $D_a(x) \geq x$.

Забележка. Лесно се вижда, че ако в качеството на C и D вземем времевата мярка T и пространствената мярка L, то например за програмата $P_a\colon X_1 := X_1 + 1$ ще е вярно, че

$$T_a(x) = 1 \quad \text{и} \quad L_a(x) = x + 1$$

и значи не е възможно за всяко x

$$L_a(x) \leq r(T_a(x)) = r(1) = const.$$

Следствие 5.1. Нека $C_a(x) \geq x$ и $D_a(x) \geq x$ за всяко x. Тогава съществува строго растяща рекурсивна функция r, такава че за всяко a:

$$!C_a(x) \implies C_a(x) \leq r(D_a(x)) \quad \text{за всяко } x \geq a;$$

$$!D_a(x) \implies D_a(x) \leq r(C_a(x)) \quad \text{за всяко } x \geq a.$$

Доказателство. Тръгваме от функцията f, която дефинирахме по-горе с равенството (5.1) и чрез нея конструираме следната функция r:

$$r(y) \;=\; \max_{z \leq y} f(z, y).$$

Ясно е, че и новата функция r е рекурсивна и строго растяща. Да се убедим, че тя върши работа. Отново фиксираме a, вземем произволно $x \geq a$ и приемаме, че $!C_a(x)$ (което значи и $!D_a(x)$). Тогава, тъй като $D_a(x) \geq x$, ще имаме:

$$r(D_a(x)) \stackrel{\text{деф}}{=} \max_{z \in \{0,\ldots,x,\ldots,D_a(x)\}} f(z, D_a(x)) \geq f(x, D_a(x)) \geq C_a(x).$$

Аналогично разсъждаваме и за второто неравенство. □

Нека $P(x)$ е произволен предикат. Ще казваме, че P е в сила *почти навсякъде (п.н.)*, ако съществува x_0, такова че $P(x)$ е вярно за всяко $x \geq x_0$.

Да отбележим, че неравенствата в горната теорема и следствието също са в сила почти навсякъде. Въобще, ако във важните резултати теореми от тази теория се изпусне "почти навсякъде", се получава почти нищо ☺.

Ще завършим този раздел с едно означение: ако M е мярка за сложност и $M_a(x)$ няма стойност, е логично да се приеме, че тази стойност е безкрайност. Така е, например, когато имаме времевата мярка: ако $\neg!T_a(x)$, по аксиома B1 това означава, че $P_a(x)\uparrow$, т.е. програмата P_a не спира върху x и следователно $T_a(x) = \infty$.

Тогава например неравенството $M_a(x) \leq y$ ще е изпълнено, когато

$$!M_a(x) \;\&\; M_a(x) \leq y,$$

а неравенството $M_a(x) \geq y$ ще означава

$$\neg!M_a(x) \vee !M_a(x) \;\&\; M_a(x) \geq y.$$

Съобразете, че с тази уговорка неравенствата в теоремата за рекурсивна свързаност на мерките могат да бъдат изказани по-кратко така:

$$C_a(x) \leq f(x, D_a(x)) \quad \text{и} \quad D_a(x) \leq f(x, C_a(x)).$$

5.4 Класове на сложност. Теорема за пропастта

Нека M е мярка за сложност, а b е рекурсивна функция. Да си припомним определението за това кога M_a се мажорира от b ($M_a \leq b$):

$$M_a \leq b \iff \forall x(!M_a(x) \implies M_a(x) \leq b(x)).$$

Като имаме предвид уговорката за това как трябва да се чете неравенството $M_a(x) \leq b(x)$, която току-що направихме, определението на $M_a \leq b$ можем да съкратим до:

$$M_a \leq b \iff \forall x\ M_a(x) \leq b(x).$$

Тогава $M_a \leq b$ п.н. ще означава, че за почти всички x, $M_a(x) \leq b(x)$.

Ако M е мярка за сложност, а b е рекурсивна функция, с C_b^M ще означаваме следния клас от едноместни рекурсивни функции:

$$C_b^M \ = \ \{f \mid f \text{ е тотална } \& \ \exists a\ (\varphi_a = f \ \&\ M_a \leq b \text{ п.н.})\ \}.$$

С други думи, в класа C_b^M са всички едноместни рекурсивни функции, които могат да се сметнат с M-сложност под b.

Класът C_b^M ще наричаме *сложностен клас* (или *клас на сложност)* относно мярката M.

Забележете, че сложностният клас C_b^M съдържа само рекурсивни функции. Какво ще стане, ако в него включим *всички* изчислими едноместни функции? Ето какво:

Задача 5.3. Нека

$$\hat{C}_b^M \ = \ \{f \mid \exists a\ (\varphi_a = f \ \&\ M_a \leq b \text{ п.н.})\ \}.$$

Докажете, че всяка функция от този клас е с разрешимо дефиниционното множество (и следователно тя е потенциално рекурсивна функция).

Доказателство. Нека $f \in \hat{C}_b^M$ и нека a е такова, че $\varphi_a = f$ и $M_a \leq b$ п.н. Нека още x_0 е такова, че за всяко $x \geq x_0$ е изпълнено $M_a(x) \leq b(x)$. Ще покажем, че $Dom(f)$ е разрешимо множество. Наистина, да изберем произволно $x \geq x_0$. Тогава

$$x \in Dom(f) \iff !\varphi_a(x) \overset{\text{B1}}{\iff} !M_a(x) \iff \exists z M_a(x) \simeq z \iff \exists z_{\leq b(x)} M_a(x) \simeq z.$$

Съгласно аксиома B2, множеството $\{(x,z) \mid M_a(x) \simeq z\ \}$ е разрешимо, Следователно $\{x \mid x \geq x_0\ \&\ x \in Dom(f))$ е разрешимо, откъдето и цялото множество $Dom(f)$ е разрешимо. $\square$

Любопитно е да се отбележи, че класът на всички едноместни функции е сложностен клас относно времевата и пространствената мярка, т.е. съществуват рекурсивни функции b и b', такива че

$$\mathcal{PR}_1 \;=\; C_b^T \quad \text{и} \quad \mathcal{PR}_1 \;=\; C_{b'}^L.$$

Нека b и b' са произволни рекурсивни функции. Интуицията ни подсказва, че ако b' мажорира b, при това многократно (примерно ако $b'(x) = 2^{b(x)}$), то би трябвало класът $C_{b'}^M$ да включва строго класа C_b^M. Оказва се, че ако функцията b се избере много "специална", това няма да е така. Този резултат ще е следствие от следващата теорема, доказана от Бородин.

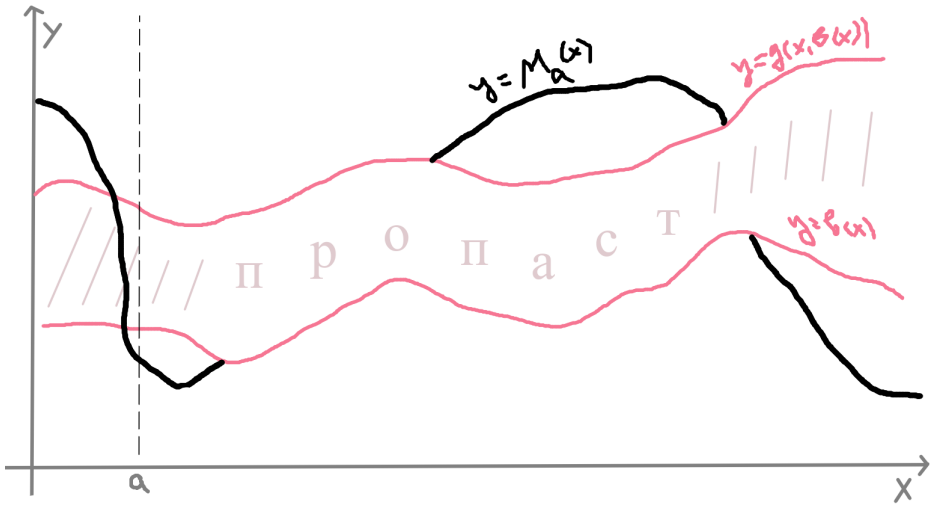
Теорема 5.2. (Теорема за пропастта / gap theorem) Нека M е сложностна мярка, а $g(x, y)$ е рекурсивна функция. Съществува рекурсивна функция $b(x)$, такава че за всяко a и всяко $x \geq a$

$$M_a(x) \leq g(x, b(x)) \implies M_a(x) \leq b(x). \tag{5.2}$$

Забележка. Като директно следствие от тази теорема получаваме, че например класовете C_b^M и $C_{2^b}^M$ съвпадат. Наистина, нека $b(x)$ е функцията от теоремата за пропастта за $g(x, y) = 2^y$. От тази теорема ще имаме, че за всяко a и всяко $x \geq a$

$$M_a(x) \leq 2^{b(x)} \implies M_a(x) \leq b(x).$$

Оттук, ако $f \in C_{2^b}^M$, то ще съществува програма P_a, пресмятаща f, такава че $M_a \leq 2^b$ п.н.. Но тогава съгласно горното условие ще имаме, че всъщност $M_a \leq b$ п.н., което означава, че $f \in C_b^M$.



Доказателство. Да разгледаме множеството

$$A \;=\; \{\; (x,y) \mid \forall a \le x \; (M_a(x) \le y \;\vee\; M_a(x) > g(x,y)) \;\}.$$

Условието $M_a(x) \le y$ е разрешимо по аксиома B2, а за другото условие $M_a(x) > g(x,y)$ можем да запишем, че

$$M_a(x) > g(x,y) \iff \forall z \le g(x,y)\; \neg M_a(x) \simeq z.$$

Следователно множеството A е разрешимо. Нека

$$b(x) \stackrel{\text{деф}}{=} \mu y[\chi_A(x,y) = 0].$$

Тази функция е изчислима. За да покажем, че е тотална, трябва да видим, че за всяко x съществува $y : (x,y) \in A$.

Наистина, да фиксираме едно естествено число x и да разгледаме множеството $B = \{\; M_a(x) \mid !M_a(x) \;\&\; a \le x \;\}$, с други думи,

$$B \;=\; \{M_0(x), \ldots, M_x(x)\},$$

като приемаме, че ако $\neg!M_a(x)$, то не добавяме стойност в B. Множеството B е крайно и следователно има най-голям елемент y (ако $B = \emptyset$, нека $y \stackrel{\text{деф}}{=} 0$). Ще покажем, че $(x,y) \in A$, откъдето ще следва, че $!b(x)$. И понеже x е произволно, то значи функцията b ще е тотална и следователно — изчислима.

Наистина, да вземем някакво $a \le x$. Ако $!M_a(x)$, то $!M_a(x) \le y$, защото y беше максималният елемент на B. Следователно $(x,y) \in A$. Ако $\neg!M_a(x)$, то $M_a(x)$ е по-голямо от всяко нещо и в частност, $M_a(x) > g(x,y)$. Значи отново $(x,y) \in A$.

Сега пристъпваме към доказателството на импликацията (5.2) от теоремата. За целта фиксираме някакво $a \in \mathbb{N}$ и нека $x \ge a$ е произволно. Да приемем, че предпоставката на (5.2) е налице, т.е. $M_a(x) \le g(x, b(x))$.

От друга страна, по определение $(x, b(x)) \in A$, освен това $a \le x$, и значи е вярно, че

$$M_a(x) \le y \;\vee\; M_a(x) > g(x,y).$$

Второто условие $M_a(x) > g(x,y)$ не е изпълнено, и остава да е вярно първото — $M_a(x) \le y$. $\square$

5.5 Теорема за ускорението

Теоремата за пропастта, която доказахме, е единият от двата т. нар. "странни резултата" в абстрактната теория на сложността. Вторият резултат (който всъщност е по-странният), е *теоремата за ускорението*, доказана най-напред от Блум.

При фиксирана мярка M е логично да очакваме, че за всяка изчислима функция съществува най-добра (оптимална) по отношение на M програма, която я пресмята. Това, обаче, не е така. Оказва се, че съществуват функции (които могат дори да са 0−1-функции), такива че *всяка* програма, която ги пресмята, може да бъде оптимизирана (или ускорена), при това колкото си искаме много. Ето и точната формулировка.

Теорема 5.3. (Теорема за ускорението / speedup theorem) Нека M е мярка за сложност, а $r(x, y)$ е произволна рекурсивна функция. Съществува рекурсивна функция f със свойството: за всяко a, такова че P_a пресмята f, съществува b, такова че P_b пресмята f, за което

$$r(x, M_b(x)) \leq M_a(x) \quad \text{почти навсякъде.}$$

Сега ако вземем $r(x, y) = 100y$, то ще излезе, че

$$100M_b(x) \leq M_a(x) \quad \text{п.н.,}$$

или все едно

$$M_b(x) \leq \frac{M_a(x)}{100} \quad \text{п.н.,}$$

т.е. постигнали сме стократно намаление на сложността.

Ако вземем $r(x, y) = 2^y$, ще имаме

$$2^{M_b(x)} \leq M_a(x) \quad \text{п.н.,}$$

което означава, че сме получили логаритмично намаление на сложността на P_a:

$$M_b(x) \leq log_2(M_a(x)) \quad \text{п.н.}$$

Азбучен указател