AKADEMIA SZTABU GENERALNEGO
im. generała broni K. Świerczewskiego

19

dr Stanisław WALIGÓRSKI

ELEMENTY TEORII ALGORYTMÓW

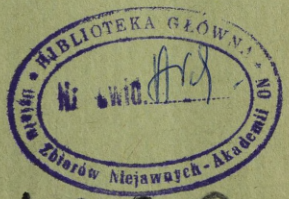
WARSZAWA LUTY 1969

AKADEMIA SZTABU GENERALNEGO
im. generała broni K. Świerczewskiego

19

dr Stanisław WALIGÓRSKI

ELEMENTY TEORII ALGORYTMÓW



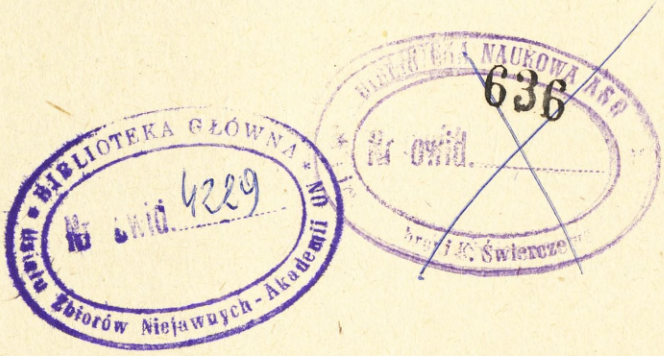
4229

WARSZAWA LUTY 1969

A K A D E M I A S Z T A B U G E N E R A L N E G O
im. gen. broni K. Świerczewskiego

dr Stanisław WALIGÓRSKI

E L E M E N T Y T E O R I I A L G O R Y T M Ó W



WARSZAWA Luty 1969 r

Wstęp

Ci wszyscy, którzy w jakikolwiek sposób zetknęli się z maszynami cyfrowymi, wiedzą, że na to, aby maszyna wykonała jakieś zadanie, trzeba jej podać program, czyli wyrażony w odpowiedni sposób pełny i ścisły opis metody rozwiązywania tego zadania. Tak jest zresztą nie tylko w przypadku stosowania maszyn cyfrowych, ale wogóle przy rozwiązywaniu prawie wszystkich bardzo skomplikowanych zagadnień, i to niezależnie od środków, jakie są przy tym stosowane. Aby móc rozwiązać takie złożone zagadnienie, zajmujący się nim człowiek /lub zespół ludzi/ musi znać dokładnie sposób rozwiązywania. Co prawda w działalności ludzi może czasem odgrywać dość istotną i pozytywną rolę intuicja, oparta na wprawie, doświadczeniu lub po prostu na zdrowym rozsądku , jednak jeżeli takie przypadki pominiemy, to nie ulega wątpliwości, że im dokładniejszy jest opis metody i im pełniej uwzględnia się w nim wszystkie możliwe warianty zadania, tym bardziej niezawodne jest postępowanie człowieka /lub ludzi/, rozwiązujących ten problem i tym pewniejsze jest otrzymanie bezbłędnych wyników.

Taki ścisły opis metody postępowania, całkowicie i jednoznacznie określający, co należy robić w każdej fazie rozwiązywania zadania i w każdej sytuacji, jaka może przy tym powstać, który nie zostawia miejsca na żadne działanie czynników przypadkowych lub ubocznych, nie związanych z zadaniem, nazywamy algorytmem.

Jest oczywiste, że w takim ścisłym formułowaniu przepisów postępowania byli i są zainteresowani przede wszystkim matematycy i ci, którzy z matematyki korzystają. Dlatego w ramach matematyki powstała tzw. teoria algorytmów, zajmująca się algorytmami, ich własnościami, różnymi sposobami ich formułowania, dobieraniem algorytmów do różnych problemów matematycznych, porównywaniem różnych algorytmów, ich mocy i skuteczności itp.

Przede wszystkim jednak teoria algorytmów zajmuje się tak zwanymi problemami rozstrzygalności.

Badania w zakresie podstaw matematyki, prowadzone w latach trzydziestych, doprowadziły do odkrycia, że istnieją zagadnienia matematyczne, które są nierozstrzygalne, to znaczy nie mogą być rozwiązane przy pomocy żadnych dostępnych człowiekowi środków. Niemożność ich rozwiązania nie wynika z tego, że obecne możliwości ludzi są niedostateczne, lecz wynika z samego charakteru zadania: nie ma metody, przy pomocy której można by je rozwiązać.

Powstało w ten sposób zagadnienie o dużym znaczeniu praktycznym: które problemy matematyczne są rozstrzygalne, a które nie - w szczególności, które funkcje są obliczalne, a które nie. Obecnie pierwszym pytaniem, które zadaje matematyk, stykający się z nowym i trudnym problemem, jest, czy ten problem jest rozstrzygalny. Jeżeli odpowiedź jest negatywna, to wszelkie próby rozwiązania takiego problemu są bezcelowe, bo są z góry skazane na niepowodzenie. Ponieważ dzięki stosowaniu maszyn matematycznych zakres zastosowań matematyki w różnych dziedzinach życia stale i szybko wzrasta, a wykorzystywane przy tym metody matematyczne stają się coraz bardziej skomplikowane, zagadnienia rozstrzygalności mają również duże znaczenie dla wszystkich tych , którzy z matematyki wyższej korzystają.

Niemniej ważnym dla praktyki zagadnieniem jest pytanie, czy dany problem, o którym skądinąd wiadomo, że jest rozstrzygalny, może być rozwiązany przy pomocy środków, którymi aktualnie dysponujemy. Środkami tymi są najczęściej maszyny cyfrowe ; ich wydajność obliczeniowa wiąże się z takimi własnościami maszyny, jak szybkość, pojemność pamięci, rodzaj, ilość i wydajność urządzeń wejścia i wyjścia maszyny /dalekopisy lub elektryczne maszyny do pisania, czytniki kart, taśmy perforowanej, drukarki, urządzenia rysujące, wyjścia ekranowe, pamięci pomocnicze lub inne urządzenia/. Tak więc praktyczne problemy efektywnej obliczalności wchodzą raczej w zakres wiedzy o maszynach cyfrowych; w zakres teorii algorytmów wchodzą tylko te zagadnienia, które da się wyrazić i rozwiązywać w sposób czysto matematyczny, nie korzystając przy tym z żadnych założeń ani informacji natury technicznej.

Z problemem efektywnej obliczalności wiąże się ściśle inne zagadnienie: wybieranie spośród wszystkich możliwych algorytmów rozwiązania danego zadania takiego algorytmu, który byłby możliwie jak najlepszy - na przykład pozwalał rozwiązać zadanie w możliwie najmniejszej ilości kroków. Chociaż praktyczne rozwiązywanie tego problemu także wchodzi w zakres wiedzy o maszynach cyfrowych, jednak

jego strona teoretyczna należy do teorii algorytmów.

Taki podział zagadnień pomiędzy teorię algorytmów a wiedzę o maszynach matematycznych rzutuje w pewien sposób na metody formułowania i rozwiązywania problemów, stosowane w obu tych dziedzinach. W teorii algorytmów wszystkie pojęcia są sprowadzane do możliwie prostej postaci, tak, żeby jak najbardziej ułatwić przeprowadzenie podstawowych wywodów teoretycznych, a ponadto, żeby wyniki tych wywodów były możliwie najbardziej ogólne. Z tego powodu może się nawet na pierwszy rzut oka wydawać, że wyniki teorii algorytmów mają stosunkowo mało wspólnego z tym, co wiemy o maszynach matematycznych. Tak jednak nie jest, choć prawdą jest, że przeniesienie wyników jednej dziedziny do drugiej i konfrontacja obu tych dziedzin wymaga pewnej wprawy i obycia matematycznego.

Rozważmy to zagadnienie bliżej na przykładzie funkcji rekurencyjnych. Będą one szczegółowo omówione niżej, na razie interesuje nas tylko sama zasada ich wprowadzania.

Wiadomo, że maszyna cyfrowa może wykonywać pewną ilość operacji elementarnych na liczbach, takich jak dodawanie, odejmowanie, mnożenie, dzielenie, badanie, czy liczba jest równa zeru, przesyłanie liczb z jednego rejestru lub miejsca pamięci do drugiego i inne. Wiadomo również, że składając te operacje i ustawiając je w określonej kolejności /podyktowanej przez program/ można obliczać nawet bardzo skomplikowane funkcje. Powstaje wobec tego pytanie: jakie funkcje można na maszynie cyfrowej obliczać? Dla prostoty abstrahujemy od tego, że czas działania maszyny i objętość jej pamięci są ograniczone, oraz zakładamy, że ani maszyna, ani obsługujący jej ludzie nie popełniają błędów.

Teoria algorytmów formułuje to pytanie prościej, choć bardziej abstrakcyjnie: dana jest pewna ilość funkcji i pewna ilość prostych operacji , które pozwalają z jednych funkcji tworzyć inne. Do takich operacji na funkcjach należy na przykład złożenie, które pozwala, dajmy na to, z funkcji $\sin x$ i $\exp x$ tworzyć nowe funkcje $\sin \exp x$ oraz $\exp \sin x$. Pytamy, jakie funkcje można w ten sposób otrzymać ? Aby rozważania, prowadzące do odpowiedzi były możliwie proste, zarówno funkcje, z których konstruujemy inne, jak same zasady konstrukcji /operacje na funkcjach/ są możliwie najbardziej prymitywne. W tym sensie nawet dodawanie i mnożenie są zbyt skomplikowane i konstruuje się je z funkcji bardziej elementarnych. Okazuje się jednak, że tak daleko posunięte uproszczenia nie wpływają istotnie na wynik końcowy - klasa funkcji konstruowalnych w ten sposób /tzw. funkcje częściowo rekurencyjne/

pokrywa

się z klasą funkcji, które można obliczyć na dowolnej wyidealizowanej maszynie cyfrowej. Ujęcie zagadnienia w sposób abstrakcyjny zwalnia nas od konieczności rozważania szczegółów budowy konkretnych maszyn cyfrowych i wogóle wszelkich szczegółów technicznej, a nie matematycznej natury.

Oczywiście te wszystkie uwagi o związku pomiędzy wiedzą o maszynach matematycznych a teorią algorytmów dotyczą stanu obecnego - warto pamiętać o tym, że teoria algorytmów powstała znacznie wcześniej, niż maszyny matematyczne, ale obecnie w pewnym sensie obie te dziedziny wzajemnie się uzupełniają.

Celem niniejszego opracowania jest zapoznanie Czytelników z podstawami teorii algorytmów w zakresie elementarnym. Charakter tej pracy wyklucza wszelkie zastosowanie zbyt zaawansowanych pojęć i metod; Czytelnicy, którzy chcieli by pogłębić swoje wiadomości, mogą skorzystać z literatury podanej w spisie na końcu pracy. Opracowanie to jest dostępne dla osób, nie posiadających specjalistycznego przygotowania matematycznego. Będzie tu jednak potrzebna znajomość takich pojęć, jak liczba naturalna, zbiór, element zbioru, funkcja jednej i wielu zmiennych, funkcja całkowicie i częściowo określona, ciąg skończony i nieskończony, a także potrzebne będzie pewne obycie z metodami rozumowania oraz wprowadzania i opisywania pojęć matematycznych, stosowanymi w matematyce elementarnej. Pożyteczne są także wiadomości o zasadach działania i programowaniu maszyn cyfrowych.

1. Funkcje pierwotnie rekurencyjne

W tym rozdziale będziemy się zajmować tylko takimi funkcjami, że zarówno ich argumenty, jak one same, przyjmują tylko wartości naturalne 0, 1, 2, ... Do klasy takich funkcji należą na przykład suma i iloczyn /bo zarówno suma jak iloczyn dwu liczb naturalnych jest znowu liczbą naturalną/ , a nie należą na przykład różnica i iloraz /bo mogą dać wynik, który nie jest liczbą naturalną, jak 2 - 3 = -1 lub 3:2 = 1,5 /.

Wprowadźmy następujące funkcje, które dalej będziemy nazywać funkcjami najprostszymi:

Funkcja jednej zmiennej $o(x)$, równa 0 dla każdej wartości x.

Funkcja jednej zmiennej $s(x) = x + 1$. Jest ona zwana "następnikiem", gdyż dla każdej liczby naturalnej x oblicza liczbę, która następuje bezpośrednio po niej w ciągu wszystkich liczb naturalnych 0, 1, 2, ...

Funkcje n zmiennych /dla n=1,2, ... / zależne od parametru m /gdzie 1 m n /

$$I_m^n (x_1, \dots , x_n) = x_m$$

Są to funkcje "wybierające" z ciągu swoich argumentów m-ty z kolei.

Stosując do tych funkcji operację superpozycji /złożenia funkcji/ można otrzymać nowe funkcje, jak na przykład

$$s(s(s(x))) = x + 3$$

$o(I_1^3(x, y, z))$ = 0 dla każdego zestawienia wartości zmiennych x, y, z .

$s(s(o(I_1^2(x, y))))$ = 2 dla wszystkich x, y .

Wszystkie funkcje , które można otrzymać z funkcji najprostszych przy pomocy operacji superpozycji, będziemy nazywać funkcjami elementarnymi.

Do funkcji elementarnych należą między innymi wszystkie funkcje stałe i wszystkie funkcje, które można otrzymać przez dodawanie do zmiennej dowolnej liczby . Ale zasób funkcji elementarnych nie jest zbyt urozmaicony i nawet taka prosta funkcja, jak suma $x + y$ już nie jest funkcją elementarną.

O wiele większe możliwości tworzenia nowych funkcji daje inna operacja: rekurencja pierwotna.

Przypuśćmy, że mamy dane dwie funkcje : funkcję n zmiennych $g(x_1, \dots, x_n)$ oraz funkcję n+2 zmiennych $h(x_1, \dots, x_n, y, z)$. Wtedy możemy utworzyć nową funkcję n+1 zmiennych w następujący sposób:

$$f(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n) \qquad /1/$$

$$f(x_1, \dots, x_n, y+1) = h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y)) \qquad /2/$$

Z wzoru /1/ otrzymujemy wartość funkcji $f(x_1, \dots, x_n, y)$ dla dowolnych wartości zmiennych $x_1, \dots, x_n$ i $y=0$. Wartość funkcji dla tych samych wartości $x_1, \dots, x_n$ oraz dla $y = 1, 2, \dots$ możemy obliczyć, stosując odpowiednią ilość razy wzór /2/.

W szczególnym przypadku może być n = 0 ; wtedy funkcja g redukuje się do liczby i te wzory przyjmują postać

$$f(0) = g$$

$$f(y+1) = h(y, f(y))$$

Niech na przykład $n=0$, $g=0$, $h(x, y) = I_1^2(x, y) = x$

Wtedy

$$f(0) = 0$$

$$f(y+1) = y$$

Stąd mamy $f(1) = 0$, $f(2) = 1$, $f(3) = 2$ itd, a ogólnie $f(x) = 0$ dla $x = 0$ i $f(x) = x-1$ dla $x > 0$. Oznaczymy tę funkcję przez $x \dot{-} 1$.

Funkcje, które można otrzymać z funkcji najprostszych przez zastosowanie do nich dowolną /skończoną/ ilość razy operacji superpozycji i rekurencji pierwotnej, będziemy nazywać funkcjami pierwotnie rekurencyjnymi.

Funkcjami pierwotnie rekurencyjnymi są, między innymi, suma $x+y$, iloczyn $x \cdot y$, potęga o wykładniku naturalnym x^y, różnica ograniczona $x \dot{-} y$, wartość bezwzględna różnicy $|x-y|$, funkcja signum /znak/ $sg(y)$, funkcja antysignum /znak przeciwny/ $\overline{sg}(y)$, część całkowita ilorazu $\left[\frac{x}{y}\right]$, reszta z dzielenia x przez y $rest(x, y)$ i wiele innych.

A oto w jaki sposób można to /dla wymienionych wyżej funkcji/ sprawdzić.

Suma:

$$x + 0 = x$$

$$x + (y+1) = (x+y) + 1$$

W tym przypadku $g(x) = I_1^1(x)$, $h(x, y, z) = s(I_3^3(x, y, z))$

czyli g jest funkcją najprostszą, zaś h złożeniem dwu takich funkcji.

Iloczyn:

$$x \cdot 0 = 0$$
$$x \cdot (y + 1) = x + x \cdot y$$

W tym przypadku $g(x) = o(x)$, czyli jest funkcją najprostszą, zaś $h(x, y, z) = x + z = I_1^3(x,y,z) + I_3^3(x,y,z)$, czyli jest złożeniem sumy /która jest pierwotnie rekurencyjna, jak to pokazaliśmy bezpośrednio wyżej/ oraz funkcji najprostszych.

Potęga o wykładniku naturalnym x^y:

$$x^0 = 1$$
$$x^{y+1} = x \cdot x^y$$

Tutaj $g(x) = s(o(x))$, a więc jest złożeniem funkcji najprostszych, oraz $h(x, y, z) = x \cdot z = I_1^3(x,y,z) \cdot I_3^3(x,y,z)$, a więc jest złożeniem iloczynu /o którym już wiemy, że jest pierwotnie rekurencyjny/ oraz funkcji najprostszych.

Różnica ograniczona $x \dot{-} y$:

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

Tu mamy $g(x) = I_1^1(x)$, $h(x, y, z) = z \dot{-} 1 = I_3^3(x,y,z) \dot{-} 1$. Funkcja $z \dot{-} 1$ jest pierwotnie rekurencyjna, co wynika z przykładu podanego poprzednio. Obliczając na podstawie tej definicji funkcji $x \dot{-} y$ jej wartości dla różnych wartości zmiennych x i y możemy się przekonać, że

$$x \dot{-} y = \begin{cases} x - y & , \text{gdy } x \geqslant y \\ 0 & , \text{gdy } x \leqslant y \end{cases}$$

Stąd pochodzi nazwa tej funkcji: różnica ograniczona.

Wartość bezwzględna różnicy $|x - y|$:

Korzystając z podanej wyżej własności różnicy ograniczonej możemy napisać

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

Jako złożenie funkcji pierwotnie rekurencyjnych, jest to także funkcja pierwotnie rekurencyjna.

Funkcja signum $sg(y)$:

$$sg(0) = 0$$
$$sg(y + 1) = 1$$

To , że to jest funkcja pierwotnie rekurencyjna, jest oczywiste. Zauważmy, że

$$sg(y) = \begin{cases} 0, & \text{gdy } y = 0 \\ 1, & \text{gdy } y > 0 \end{cases}$$

czyli $sg(y) = 1$ wtedy i tylko wtedy, gdy y jest liczbą dodatnią - stąd nazwa funkcji.

Funkcja $\overline{sg}(y)$:

$$\overline{sg}(0) = 1$$

$$\overline{sg}(y + 1) =$$

Stąd

$$\overline{sg}(y) = \begin{cases} 1, & \text{gdy } y = 0 \\ 0, & \text{gdy } y > 0 \end{cases}$$

czyli $\overline{sg}(y) = 1$ wtedy i tylko wtedy, gdy y nie jest liczbą dodatnią /to znaczy jest zerem/.

Część całkowita ilorazu $\left[\frac{x}{y}\right]$

W tym przypadku sprawdzenie, że to jest funkcja pierwotnie rekurencyjna, jest znacznie bardziej skomplikowane. Byłoby to wogóle niewykonalne, gdybyśmy nie określili dodatkowo wartości $\left[\frac{x}{0}\right]$ - w teorii funkcji rekurencyjnych wygodnie jest przyjąć, że taka wartość istnieje i że $\left[\frac{x}{0}\right] = x$ dla dowolnego x.

Jeżeli $y > 0$ i jeżeli przez n oznaczymy wartość $\left[\frac{x}{y}\right]$, to w ciągu

$$1y \dot{-} x,\ 2y \dot{-} x,\ \ldots,\ ny \dot{-} x,\ \ldots,\ xy \dot{-} x$$

będzie dokładnie n zer. A zatem dla dowolnego $y > 0$ jest

$$\left[\frac{x}{y}\right] = \sum_{i=1}^{x} \overline{sg}(iy \dot{-} x)$$

czyli, inaczej mówiąc, możemy policzyć wszystkie te zera, obliczając wartości funkcji $\overline{sg}$ dla wszystkich elementów tego ciągu i dodając je do siebie. Jeżeli $y = 0$, to też możemy ułożyć taki ciąg, ale wtedy on będzie miał dokładnie n elementów i wszystkie będą równe zeru ; podany wzór będzie w tym przypadku także prawdziwy. Udało nam się więc przedstawić badaną funkcję w postaci sumy funkcji pierwotnie rekurencyjnych; co prawda suma ta jest skończona, ale ilość jej elementów zależy od x ; nie można więc tu skorzystać z poprzednio dowiedzionego faktu, że suma dwuargumentowa /a więc także suma dowolnej stałej ilości składników/ jest pierwotnie rekurencyjna. Ale taką sumę o zmiennej górnej granicy sumowania można także obliczać przy pomocy rekurencji pierwotnej. Rzeczywiście, niech $a(x, y, i)$ będzie dowolną funkcją pierwotnie rekurencyjną i niech

$$S(x, y, m) = \sum_{i=1}^{m} a(x, y, i)$$

oraz $S(x, y, 0) = 0$

Wtedy

$$S(x,y,m+1) = S(x,y,m) + a(x,y,m+1)$$

czyli dla tej rekurencji pierwotnej

$$g(x, y) = o(I_1^2(x, y))$$

$$h(x, y, m, z) = a(x,y,s(m)) + z =$$
$$= a(I_1^4(x,y,m,z), I_2^4(x,y,m,z), s(I_3^4(x,y,m,z))) +$$
$$+ I_4^4(x,y,m,z)$$

Jak widać, obie funkcje, g i h, są złożeniami funkcji najprostszych i funkcji pierwotnie rekurencyjnych, a więc S jest także funkcją pierwotnie rekurencyjną. W szczególności, gdy $a(x,y,i) = \overline{sg}(iy \dot{-} x)$, $S(x, y, x) = \left[\frac{x}{y}\right]$, a więc część całkowita ilorazu też jest funkcją pierwotnie rekurencyjną.

Reszta z dzielenia, rest(x, y):
Podobnie, jak w przypadku części całkowitej ilorazu, musimy uprzednio umówić się, co będziemy rozumieć przez rest(x, 0) ; w przeciwnym razie dalsze rozważania nie miałyby sensu. W teorii funkcji rekurencyjnych przyjmuje się, że rest(x,0) = x dla dowolnego x. Przy tym założeniu nie trudno sprawdzić, że

$$\text{rest}(x,y) = x \dot{-} y \cdot \left[\frac{x}{y}\right]$$

dla dowolnych wartości zmiennych x i y. Jak widać, jest to złożenie funkcji pierwotnie rekurencyjnych, a zatem ta funkcja jest też pierwotnie rekurencyjna.

Przytoczone rozważania stanowią próbkę metod /przynajmniej tych najbardziej elementarnych/, jakie stosuje się do sprawdzania, czy jakaś funkcja należy do klasy funkcji pierwotnie rekurencyjnych, czy też nie. Najczęściej stosowanym chwytem jest konstruowanie coraz to **nowych funkcji z innych funkcji, poprzednio otrzymanych i przechodzenie** w ten sposób od funkcji prostszych do coraz bardziej złożonych. Oprócz operacji superpozycji i rekurencji pierwotnej można przy tym korzystać z następujących zasad:

Jeżeli funkcja $g(x_1, \dots, x_n)$ jest pierwotnie rekurencyjna, to funkcje

$$s(x_1, \dots, x_n) = \sum_{i=0}^{x_n} g(x_1, \dots, x_{n-1}, i) \qquad /3/$$

$$p(x_1, \dots, x_n) = \prod_{i=0}^{x_n} g(x_1, \dots, x_{n-1}, i) \qquad /4/$$

są także pierwotnie rekurencyjne.

Jeżeli oprócz tego $a(x_1, \dots, x_n)$, $b(x_1, \dots, x_n)$ są funkcjami pierwotnie rekurencyjnymi, to funkcje

$$r(x_1, \ldots, x_n) = \sum_{i=a(x_1,\ldots,x_n)}^{a(x_1,\ldots,x_n)+\,b(x_1,\ldots,x_n)} g(x_1,\ldots,i) \qquad /5/$$

$$t(x_1, \ldots, x_n) = \sum_{i=a(x_1,\ldots,x_n)}^{a(x_1,\ldots,x_n)+\,b(x_1,\ldots,x_n)} g(x_1,\ldots,i) \qquad /6/$$

są pierwotnie rekurencyjne.

Jeżeli mamy s+1 funkcji pierwotnie rekurencyjnych n zmiennych $f_1(x_1,\ldots,x_n)$, ... , $f_{s+1}(x_1,\ldots,x_n)$ oraz s funkcji n zmiennych $a_1(x_1,\ldots,x_n)$, ... , $a_s(x_1,\ldots,x_n)$ takich, że dla każdego zestawienia wartości zmiennych $x_1,\ldots, x_n$ co najwyżej jedna z tych funkcji a_i przyjmuje wartość 0 , to funkcja f, określona w następujący sposób:

$$f(x_1, \ldots, x_n) = \begin{cases} f_1(x_1,\ldots,x_n) & \text{gdy } a_1(x_1,\ldots,x_n)=0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ f_s(x_1,\ldots,x_n) & \text{gdy } a_s(x_1,\ldots,x_n)=0 \\ f_{s+1}(x_1,\ldots,x_n) & \text{w pozostałych przypadkach} \end{cases} \qquad /7'$$

jest również pierwotnie rekurencyjna.

W rozważanym wyżej przypadku funkcji $\left[\frac{x}{y}\right]$ sprawdziliśmy w pewnym momencie, że funkcja 3 zmiennych S jest pierwotnie rekurencyjna. Wzór /3/ jest uogólnieniem tego i dowodzi się go analogicznie, stosując schemat rekurencji pierwotnej względem zmiennej x_n. Podobnie sprawdza się poprawność /4/. Pełne dowody Czytelnik może znaleźć w podręczniku Malcewa [3] . /5/ i /6/ są nieznaczną modyfikacją wniosków z zależności /3/ i /4/, przytoczonych w tej samej książce. Prawdziwość /7/ wynika stąd, że tak określoną funkcję f można przedstawić w postaci

$$f = f_1\, \overline{sg}(a_1) + \ldots + f_s\, \overline{sg}(a_s) + f_{s+1}\, sg(a_1 \cdot \ldots \cdot a_s)$$

Dla uproszczenia zapisu pomijamy w tym wzorze ~~xxixxxxx~~ argumenty funkcji.

Ze wszystkimi tymi sposobami konstruowania funkcji na podstawie innych Czytelnik miał już zapewne sposobność spotkać się przy innej okazji, ponieważ są to chwyty stosowane często w analizie matematycznej /zwłaszcza w teorii ciągów i szeregów/, w kombinatoryce, w metodach numerycznych oraz związanych z tymi metodami zasadach programowania maszyn liczących. Na przykład w znanej definicji silni

$$0! = 1$$
$$(n+1)! = (n+1)\cdot n!$$

wykorzystuje się właśnie schemat rekurencji pierwotnej.

Suma k pierwszych wyrazów ciągu potęgowego o współczynnikach naturalnych ma postać

$$S(x, k) = \sum_{i=0}^{k} a_i \cdot x^i$$

Zamiast a_i możemy także napisać $a(i)$, gdyż w gruncie rzeczy mamy tu do czynienia z funkcją argumentu naturalnego. Jeżeli te współczynniki będą zależne jeszcze od innych zmiennych $x_1, \ldots, x_n$, to suma również będzie zależna od tych zmiennych

$$S(x, x_1, \ldots, x_n, k) = \sum_{i=0}^{k} a(x_1, \ldots, x_n, i) \cdot x^i$$

Z wzoru /3/ i z poprzednich rozważań wynika, że w każdym **przypadku, gdy a jest funkcją pierwotnie rekurencyjną,** S jest też pierwotnie rekurencyjna.

Bardzo często, gdy funkcja jest zbyt skomplikowana, aby można ją było opisać jednym wzorem, definiuje się ją"kawałkami". Weźmy jakiś prosty przykład:

$$f(x) = \begin{cases} x + 2 & \text{gdy } 0 \leqslant x \leqslant 2 \\ x^2 & \text{gdy } 3 \leqslant x \leqslant 5 \\ \left[\frac{x}{2}\right] & \text{gdy } 6 \leqslant x \leqslant 10 \\ x^3 & \text{gdy } x \geqslant 11 \end{cases}$$

Otóż wszystkie warunki postaci $i \leqslant x \leqslant j$ można zastąpić innymi, tworząc funkcję , która jest równa 0 tylko dla takich x i jest różna od zera dla $x < i$ oraz $x > j$. Funkcją taką jest, na przykład,

$$i \dot{-} x + x \dot{-} j$$

i wobec tego opis funkcji f można zapisać tak:

$$f(x) = \begin{cases} x + 2 & \text{gdy } 0 \dot{-} x + x \dot{-} 2 = 0 \\ x^2 & \text{gdy } 3 \dot{-} x + x \dot{-} 5 = 0 \\ \left[\frac{x}{2}\right] & \text{gdy } 6 \dot{-} x + x \dot{-} 10 = 0 \\ x^3 & \text{w przeciwnym przypadku .} \end{cases}$$

Jak widać , jest to zapis postaci /7/ , a zatem, w szczególności, funkcja f , zdefiniowana w ten sposób, jest pierwotnie rekurencyjna. Wymyślenie bardziej skomplikowanego , konkretnego przykładu zastosowania /7/ do określenia funkcji 2 lub więcej zmiennych pozostawiamy Czytelnikowi.

Jak już założyliśmy na początku tego rozdziału, rozważamy tylko te funkcje, które dla dowolnych naturalnych wartości swoich argumentów mają określone wartości , będące liczbami naturalnymi. Właściwie wszystkie takie funkcje, z jakimi możemy się zetknąć w praktyce, są funkcjami pierwotnie rekurencyjnymi; w każdym razie na pewno do nich należą te, które w jakikolwiek sposób można opisać przy pomocy wzorów, definicji rekurencyjnych dowolnej postaci, lub definiowania "kawałkami". Istnieją jednak takie funkcje, które nie są pierwotnie rekurencyjne, chociaż w codziennej praktyce obliczeniowej się ich nie spotyka. Aby się o istnieniu takich funkcji przekonać, możemy przeprowadzić następujące rozumowanie.

Funkcji pierwotnie rekurencyjnych jednej zmiennej jest nieskończenie wiele, ale można udowodnić / choć w dość skomplikowany sposób - zob. np. [3]/, że da się je wszystkie ponumerować kolejnymi **liczbami naturalnymi 0,1,2 ... . .** Inaczej mówiąc, da się ustawić wszystkie funkcje pierwotnie rekurencyjne 1 zmiennej w ciąg

$$f_0, f_1, f_2, \ldots$$

Utwórzmy teraz funkcję $D(k, x)$ taką , że dla każdego naturalnego k

$$D(k, x) = f_k(x)$$

Jak widać, dla każdej funkcji jednej zmiennej $g(x)$, która jest pierwotnie rekurencyjna, istnieje taka liczba naturalna m , że

$$g(x) = D(m, x)$$

dla wszystkich naturalnych wartości x. Odwrotnie, dla każdej ustalonej wartości m funkcja $D(m, x)$ jest pierwotnie rekurencyjną funkcją zmiennej x .

Zbadajmy teraz funkcję 1 zmiennej

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

Dla każdej **naturalnej** wartości x wartość funkcji d jest określona i jest liczbą naturalną. Gdyby $d(x)$ była funkcją pierwotnie rekurencyjną, to istniałaby liczba naturalna m taka, że $d(x) = D(m, x)$ dla wszystkich naturalnych x, czyli w szczególności

$$d(m) = D(m, m)$$

Ale wtedy z definicji funkcji d wynikałoby, że

$$D(m, m) + 1 = D(m, m)$$

co jest niemożliwe. Przypuszczenie, że $d(x)$ jest funkcją pierwotnie rekurencyjną, doprowadziło nas do niedorzeczności, a zatem to przypuszczenie jest fałszywe.

Stąd wynika, że funkcja $d(x)$ nie jest pierwotnie rekurencyjna. Z tego wynika także wniosek, że funkcja $D(k, x)$ /jako funkcja 2 zmiennych/ także nie jest pierwotnie rekurencyjna, bo gdyby tak

było, to z tego wynikałoby, że $d(x) = D(x, x)+1 =$
$= s'D(I_2^2(k, x), x))$ jako złożenie funkcji D i funkcji najprostszych, także musiałaby być pierwotnie rekurencyjna, a tak nie jest.

Otrzymaliśmy w ten sposób przykłady funkcji jednej i dwu zmiennych, które nie są pierwotnie rekurencyjne. Zupełnie analogicznie można otrzymać funkcje dowolnej ilości zmiennych, które nie są pierwotnie rekurencyjne, gdyż dla każdej liczby naturalnej n zbiór wszystkich funkcji pierwotnie rekurencyjnych n zmiennych można ułożyć w ciąg, a zatem istnieje funkcja $n+1$ zmiennych

$$D(k, x_1, \ldots, x_n)$$

/zwana funkcją uniwersalną dla zbioru pierwotnie rekurencyjnych funkcji n zmiennych/ taka, że dla każdego ustalonego naturalnego k $D(k, x_1, \ldots, x_n)$ jest funkcją pierwotnie rekurencyjną zmiennych $x_1, x_2, \ldots, x_n$ i odwrotnie, dla każdej funkcji $g(x_1, \ldots, x_n)$, która jest pierwotnie rekurencyjna, istnieje liczba m taka, że równość

$$g(x_1, \ldots, x_n) = D(m, x_1, \ldots, x_n)$$

zachodzi dla wszystkich wartości $x_1, \ldots, x_n$.
Przy tych założeniach ani funkcja $D(k, x_1, \ldots, x_n)$, ani $d(k, x_2, \ldots, x_n) = D(k, k, x_2, \ldots, x_n) + 1$ nie są pierwotnie rekurencyjne.

2. Funkcje ogólnie rekurencyjne

Funkcje uniwersalne zbiorów funkcji pierwotnie rekurencyjnych n zmiennych należą do szerszej klasy funkcji: są to tak zwane funkcje ogólnie rekurencyjne. Aby tę klasę bliżej określić, musimy wprowadzić jeszcze jedną operację na funkcjach: minimum ograniczone.

Niech $f(x_1, \dots, x_n)$ będzie dowolną funkcją n zmiennych taką, że dla dowolnych naturalnych wartości jej argumentów jej wartość jest liczbą naturalną. Przez

$$M_y\,(f(x_1, \dots, x_{n-1}, y) = x_n) \qquad /8/$$

oznaczymy najmniejszą wartość liczby y, dla której jest spełnione równanie

$$f(x_1, \dots, x_{n-1}, y) = x_n$$

/dla danych $x_1, \dots, x_n$ /, pod warunkiem, że równanie to dla wszystkich wartości zmiennych $x_1, \dots, x_n$ ma rozwiązanie. Jeżeli istnieje chociażby jedno zestawienie wartości tych zmiennych takie, że to równanie nie ma rozwiązań, to przyjmuje się, że wartość wyrażenia /8/ dla żadnej wartości zmiennych $x_1, \dots, x_n$ nie jest określona.

Jeżeli wartość wyrażenia /8/ jest określona, to wyznacza ono nową funkcję n zmiennych

$$g(x_1, \dots, x_n) = M_y(f(x_1, \dots, x_{n-1}, y) = x_n) \qquad /9/$$

Jeżeli sobie ustalimy którąkolwiek ze zmiennych, występujących w tym wzorze po lewej stronie - w szczególności może to być zmienna x_n, to otrzymamy funkcję g, zależną od odpowiednio mniejszej ilości argumentów, albo nawet stałą, jak to widać chociażby z pierwszego z poniższych przykładów.

Jeżeli przyjmiemy n=1, to ustalając zmienną x_1: $x_1=1$ i przyjmując $f(y) = sg(y)$ otrzymamy wyrażenie postaci /8/

$$M_y\,(sg(y) = 1)$$

którego wartość jest równa 1, jako że to właśnie jest najmniejsza z liczb naturalnych, dla których wartość funkcji sg jest równa 1.

Gdybyśmy nie ustalali zmiennej x, otrzymalibyśmy inne wyrażenie

$$M_y(sg(y) = x)$$

Ale, jak to wynika z definicji funkcji sg , występujące w tym wyrażeniu równanie nie ma rozwiązań dla x różnego od 0 lub 1; zatem wartość całego wyrażenia nie jest określona.

Podobnie nie jest określona wartość wyrażenia

$$M_y\ (x + y = z)$$

gdyż na to, żeby to równanie nie miało rozwiązania y, będącego liczbą naturalną, wystarczy dobrać jakąkolwiek wartość x większą od wartości z /na przykład x=1, z=0 /.

Równanie

$$y \dot{-} x = z$$

ma rozwiązanie y dla każdej pary wartości x i z. Jeżeli z = 0, to, jak wynika z definicji różnicy ograniczonej, $y \dot{-} x$; najmniejszym y spełniającym tę zależność jest liczba 0. Jeżeli $z > 0$ to wtedy $y \dot{-} x = y - x$ i stąd $y = x + z$. Wobec tego

$$g(x, z) = M_y\ (y \dot{-} x = z) = \begin{cases} 0 & \text{gdy } z = 0 \\ x+z & \text{w przeciwnym przypadku} \end{cases}$$

Zmienna y w /8/ i /9/ jest tzw. zmienną związaną - pozostałe zmienne, $x_1, \ldots, x_n$, są zmiennymi **wolnymi** . Nazwy te pochodzą stąd, że zmiennym $x_1, \ldots, x_n$ można w tym wyrażeniu nadawać dowolne wartości , lub ustalać je według upodobania, a ze zmienną y tego zrobić nie można, bo jej rola i sposób nadawania jej wartości jest ściśle określona przez postać wyrażenia. Jeżeli wyrażenie /8/ jest określone, **to jego wartość zależy tylko od wszystkich zmiennych wolnych, które w nim występują.**

Oczywiście zmienne wolne i związane nie muszą być zawsze oznaczane w ten sam sposób; na przykład w wyrażeniu $M_z\ (|x - z) = y)$ zmienną związaną jest z , zaś zmiennymi wolnymi x i y . Czytelnik może się sam przekonać, że występujące w tym wyrażeniu równanie ma rozwiązanie z /jedno lub więcej/ dla każdej pary wartości x i y , a więc wyrażenie to określa funkcję swoich zmiennych wolnych x, y .

W tych z przytoczonych wyżej przykładów, w których funkcja /9/ była określona, można ją było wyznaczyć bez większych trudności. Jest jednak zrozumiałe, że nie zawsze obliczenia przebiegają tak łatwo. W trudniejszym przypadku wartość funkcji g z wzoru /9/ /o ile ona istnieje/ można dla danych wartości zmiennych $x_1, \ldots, x_n$ obliczać metodą prób: zamiast y podstawia się 0 i sprawdza się, czy równość jest spełniona; jeżeli nie, zwiększa się y o 1 i znów bada spełnienie równości , i tak dalej, aż do znalezienie pierwszej napotkanej /a więc także najmniejszej / wartości y, która to równanie spełnia.

Jeżeli to obliczenie przeprowadzamy ręcznie, to taki sposób jest żmudny i niedogodny, ale jeżeli mamy możność wykorzystania do tego maszyny cyfrowej, to właściwie jest to zupełnie naturalny sposób liczenia. Wiadomo, że dość często stosowanym chwytem programowym jest tworzenie tak zwanych "pętli" programu: pewne obliczenie powtarza się wielokrotnie dopóty, dopóki nie będzie spełniony pewien warunek, zwany przez programistów warunkiem wyjścia z pętli, który jest dobierany zależnie od tego, co chcemy otrzymać jako ostateczny efekt działania takiej pętli. W tym przypadku powtarzaną operacją jest dodawanie jedynki do zmiennej y /której na początku nadano wartość 0/, zaś warunkiem wyjścia z pętli jest spełnienie równania, występującego w /8/. Wartość y w momencie wychodzenia z pętli jest poszukiwaną wartością funkcji g z wzoru /9/.

Funkcją ogólnie rekurencyjną nazywamy każdą funkcję, którą można otrzymać z funkcji najprostszych przez zastosowanie skończoną ilość razy operacji superpozycji, rekurencji pierwotnej, lub minimum ograniczonego.

Ponieważ wszystkie trzy operacje na funkcjach : superpozycja, rekurencja pierwotna i minimum ograniczone mogą być wykonywane na maszynie cyfrowej /w tym sensie, że jeżeli znamy wartości jakichś danych funkcji dla zadanych wartości ich argumentów, to możemy dla tych samych wartości argumentów obliczyć wartości funkcji, które z tych funkcji danych powstają przez zastosowanie tych operacji/ , zatem, gdyby nie było ograniczeń, spowodowanych ograniczoną objętością pamięci, ograniczonym czasem wykonywania obliczeń , ograniczoną niezawodnością działania maszyny itp - krótko mówiąc, gdybyśmy mieli do czynienia z maszynami idealnymi, to każdą funkcję ogólnie rekurencyjną można by na takich maszynach obliczyć.

Z definicji funkcji ogólnie rekurencyjnych wynika, że każda funkcja pierwotnie rekurencyjna jest także ogólnie rekurencyjna. Jak już wspomnieliśmy wyżej, można wykazać /choć dowód jest bardzo skomplikowany, zob. np. [3] / , że funkcje uniwersalne zbiorów wszystkich funkcji pierwotnie rekurencyjnych n zmiennych /dla n= 1, 2, ... / są także funkcjami ogólnie rekurencyjnymi. Jak już wiemy, nie są one pierwotnie rekurencyjne, a zatem klasa funkcji ogólnie rekurencyjnych jest szersza od klasy funkcji pierwotnie rekurencyjnych.

3. Funkcje częściowo rekurencyjne

W poprzednich rozważaniach mieliśmy do czynienia tylko z funkcjami całkowicie określonymi, to znaczy z takimi, że ich wartości były określone dla wszystkich możliwych wartości ich argumentów. Czasem jednak jest wygodnie posługiwać się funkcjami częściowo określonymi /krócej: funkcjami częściowymi/, których wartości dla pewnych wartości ich argumentów mogą nie być określone. Funkcjami częściowo określonymi posługujemy się bardzo często; wystarczy wspomnieć takie funkcje rzeczywiste, jak

$$\frac{x^2 + 1}{x + 2} \qquad \frac{4}{x^2 - 1} \qquad \text{lub} \qquad \sqrt{(x - 2)(3 - x)}$$

pierwsza z nich jest określona dla wszystkich rzeczywistych wartości zmiennej x oprócz $x = -2$, druga dla wszystkich rzeczywistych x poza 1 i -1, trzecia jest określona dla wszystkich x spełniających zależność $2 \leqslant x \leqslant 3$. W naszych rozważaniach ograniczamy się do funkcji, których argumenty przyjmują wartości naturalne i których wartości, jeżeli są określone, to także są liczbami naturalnymi. Ale i tutaj nie byłoby zbyt trudno skonstruować odpowiedni przykład, korzystając chociażby z tego, co było powiedziane wyżej o operacji minimum ograniczonego. Wyrażenie

$$M_y(sg(y) = x)$$

nie jest określone, bo to równanie nie ma rozwiązań dla wszystkich wartości x. Gdybyśmy jednak zrezygnowali z tego ograniczenia i przyjęli, że takie wyrażenie /oznaczone dla odróżnienia nieco inaczej/

$$\mu_y(sg(y) = x)$$

jest funkcją, której wartość jest określona dla tych wszystkich wartości zmiennych wolnych, dla których istnieje rozwiązanie tego równania, i nie jest określona w przeciwnym przypadku, to otrzymalibyśmy funkcję częściową g taką, że

$$g(x) = \begin{cases} x & \text{gdy } x \leqslant 1 \\ \text{nie jest określona,} & \text{gdy } x \geqslant 2 \end{cases}$$

Wyrażenie $M_y(x + y = z)$, jak to widzieliśmy wyżej, także nie jest określone, ale znowu, jeżeli zrezygnujemy z tego ograniczenia, otrzymamy częściową funkcję

$$g(x, z) = \mu_y(x + y = z)$$

taką, że

$$g(x, z) = \begin{cases} z - x & \text{gdy } x \leqslant z \\ \text{nie jest określona, gdy } x > z \end{cases}$$

Możemy się umówić, że jeśli jakieś wyrażenie algebraiczne może mieć wartość ujemną lub ułamkową po podstawieniu zamiast występujących w nim zmiennych liczb naturalnych, to uważamy je za opis funkcji częściowej, której wartość jest określona tylko dla tych wartości swoich argumentów, które po podstawieniu do tego wyrażenia dają wartość naturalną. Tak na przykład wyrażenie $x^2 + 2y$ opisuje całkowicie określoną funkcję dwu zmiennych, zaś funkcje opisane wyrażeniami $\frac{x}{2}$, $\sqrt{x}$ są częściowo określone: pierwsza jest określona tylko dla parzystych wartości x, druga tylko dla tych wartości x, które są pełnymi kwadratami: 0, 1, 4, 9, 16, ... Funkcja opisana wyrażeniem $-2(x+1)$ nie jest określona dla żadnej wartości x.

Przy tej umowie możemy napisać

$$\mu_y(x + y = z) = z - x$$

Wprowadziliśmy tu nowe działanie na funkcjach, oznaczane symbolem μ. Wyrażenie

$$\mu_y(f(x_1, \ldots, x_{n-1}, y) = x_n) \qquad /10/$$

wyznacza nam dla danej funkcji n zmiennych, częściowo określonej, nową częściowo określoną funkcję n swoich zmiennych wolnych $x_1, \ldots, x_{n-1}, x_n$. Ustalmy teraz dokładnie zasady obliczania tej nowej funkcji.

Jeżeli funkcja f nie jest skomplikowana, to można dość łatwo zorientować, czy równanie, występujące w /10/ ma rozwiązania dla danych wartości zmiennych wolnych i jeżeli funkcja f jest określona dla wszystkich wartości swoich argumentów, wyznaczyć najmniejsze wartości y w zależności od $x_1, \ldots, x_n$, o ile te wartości istnieją. Ale taka próba bezpośredniego określenia poszukiwanej funkcji może zawieść, jeśli ona sama jest skomplikowana lub jeżeli istnieją wartości jej argumentów, dla których ona nie jest określona. Natomiast opisane już wyżej postępowanie, polegające na tym, że dla danych wartości $x_1, \ldots x_{n-1}, x_n$ podstawiamy zamiast y 0, sprawdzamy, czy równanie jest spełnione, jeżeli nie, to zwiększamy y o 1 i znowu sprawdzamy, aż do znalezienia najmniejszego rozwiązania /o ile wogóle do niego można dotrzeć/, jest w każdym przypadku niezawodne.

Przyjmujemy je jako obowiązującą regułę obliczania wartości szukanej funkcji. Proces ten doprowadzi do znalezienia tej wartości tylko wtedy, gdy równanie w wyrażeniu /10/ ma rozwiązanie i dla wszystkich wartości y, mniejszych od tego rozwiązania, wartości $f(x_1, \ldots, x_{n-1}, y)$ są określone. Funkcja

$$g(x_1, \ldots, x_n) = \mu_y (f(x_1, \ldots, x_{n-1}, y) = x_n) \qquad /11/$$

nie jest określona /dla danych wartości $x_1, \ldots, x_{n-1}, x_n$ / w następujących przypadkach:

1. gdy wartość $f(x_1, \ldots, x_{n-1}, 0)$ nie jest określona
2. gdy wartość $f(x_1, \ldots, x_{n-1}, a)$ nie jest określona dla pewnego $a > 0$, zaś dla wszystkich $y = 0, 1, \ldots, a-1$ wartość $f(x_1, \ldots, x_{n-1}, y)$ jest określona, ale różna od x_n.
3. gdy wartości $f(x_1, \ldots, x_{n-1}, y)$ są określone dla wszystkich $y = 0, 1, 2, \ldots$, ale żadna z nich nie jest równa x_n.

Weźmy na przykład funkcję

$$g(x, y) = \mu_z (z - x = y)$$

Postępując według opisanej reguły, podstawiamy najpierw do równania $z = 0$:

$$0 - x = y$$

Jak widać, lewa strona tego równania jest określona tylko wtedy, gdy $x = 0$ - w przeciwnym razie byłaby to liczba ujemna, czego nie dopuszczamy. Jeżeli $x = 0$, to $z = 0$ jest rozwiązaniem równania tylko wtedy, gdy $y = 0$, a w przeciwnym przypadku musimy podstawić zamiast $z = 0$, $z = 1$:

$$1 - 0 = y \qquad \text{i tak dalej.}$$

Nietrudno zauważyć, że

$$\mu_z (z - x = y) = \begin{cases} y & \text{gdy } x = 0 \\ \text{jest nieokreślona, gdy } x \neq 0 \end{cases}$$

Widać stąd, że ta reguła obliczania funkcji /11/ nakłada istotne ograniczenia. W rozważanym przypadku byłoby grubym błędem obliczyć sobie "na boku" rozwiązanie równania $z - x = y$, czyli $z = x + y$ /które to rozwiązanie zawsze istnieje/ i przyjąć, że $g(x, y) = x+y$. Chodzi o to, żeby nie tylko przynajmniej jedno rozwiązanie istniało, /dla danych wartości zmiennych wolnych/, ale także, żeby ono dało się efektywnie obliczyć przy pomocy opisanego procesu. Tylko wtedy będziemy mogli mieć pewność, że jeżeli poszukiwana wartość funkcji g z /11/ istnieje, to będziemy ją mogli obliczyć na przykład przy pomocy maszyny cyfrowej /chyba, żeby nam stanęły na przeszkodzie takie czynniki techniczne, jak ograniczony czas liczenia, ograniczona pojemność pamięci maszyny, lub podobne/.

Tak określoną operację μ nazywamy minimum efektywnym.

Dla funkcji częściowych można zdefiniować złożenie /superpozycję/ i rekurencję pierwotną w podobny sposób , jak to zostało zrobione dla funkcji całkowicie określonych. Trzeba jednak uwzględnić w tej definicji fakt, że teraz funkcje, na których te operacje są wykonywane, mogą nie być określone dla pewnych wartości swoich argumentów.

Przypuśćmy, że mamy dane n funkcji częściowych m zmiennych

$$f_1(x_1, \ldots, x_m), \ \ldots \ , f_n(x_1, \ldots, x_m)$$

oraz jedną funkcję częściową n zmiennych

$$f(x_1, \ldots, x_n)$$

Przez superpozycję /złożenie/ funkcji f i funkcji $f_1, \ldots, f_n$ rozumiemy funkcję częściową m zmiennych

$$g(x_1, \ldots, x_m) = f(f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m)) \qquad /12/$$

Jej wartość dla danych wartości zmiennych $x_1, \ldots, x_m$ jest określona wtedy i tylko wtedy, gdy są określone wartości

$$y_1 = f_1(x_1, \ldots, x_m)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$y_n = f_n(x_1, \ldots, x_m)$$

oraz wartość

$$f(y_1, \ldots, y_n)$$

Jeżeli mamy dane dwie funkcje częściowe odpowiednio n i n+2 zmiennych: $g(x_1, \ldots, x_n)$ i $h(x_1, \ldots, x_n, y, z)$ i przez rekurencję pierwotną /porównaj wzory /1/,/2// względem zmiennej x_n utworzymy z nich nową funkcję n zmiennych , także częściową

$$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n) \qquad /13/$$

$$f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)) \qquad /14/$$

to przyjmujemy, że wartość tej funkcji dla danych wartości zmiennych $x_1, \ldots, x_n, y$ jest określona wtedy i tylko wtedy, gdy jest określona wartość $g(x_1, \ldots, x_n)$ oraz wartości $h(x_1, \ldots, x_n, z, f(x_1, \ldots, x_n, z))$ dla wszystkich $z = 0, 1, \ldots, y - 1$.

Funkcje, które można otrzymać z funkcji najprostszych, stosując do nich skończoną ilość razy operacje superpozycji, rekurencji pierwotnej i minimum efektywnego, nazywamy funkcjami częściowo rekurencyjnymi.

Z tego określenia wynika, że wszystkie funkcje pierwotnie rekurencyjne są częściowo rekurencyjne; jak wiemy, powstają one z funkcji najprostszych przez stosowanie tylko superpozycji i rekurencji pierwotnej , a więc zgodnie z tą definicją /która mówi, że operacja minimum efektywnego może, ale nie musi być stosowana/. Przy okazji możemy zauważyć, że funkcje pierwotnie rekurencyjne można opisywać także przy użyciu minimum efektywnego ; wystarczy podać przykład funkcji

$$\widetilde{sg}(x) = \mu_y((1 \dot{-} x) \dot{-} y = 0)$$

Jeżeli operacja minimum efektywnego daje funkcję całkowicie określoną /i działa na funkcję całkowicie określoną/, to wynik jest taki sam, jak gdyby zastosować operację minimum ograniczonego. Dlatego wszystkie funkcje ogólnie rekurencyjne są także częściowo rekurencyjne.

Klasa funkcji częściowo rekurencyjnych jest szersza od klasy funkcji ogólnie rekurencyjnych. Wszystkie funkcje ogólnie rekurencyjne /tak samo jak funkcje pierwotnie rekurencyjne/ są całkowicie określone, bo są budowane z funkcji najprostszych, które są całkowicie określone, a stosowane przy tej konstrukcji operacje superpozycji, relurencji pierwotnej i minimum ograniczonego, zastosowane do funkcji całkowicie określonych, dają w wyniku także funkcje całkowicie określone. Istnieją funkcje częściowo rekurencyjne, które nie są całkowicie określone , jak chociażby różnica $z - x$. Jest to funkcja częściowo rekurencyjna, bo, jak widzieliśmy wyżej, powstała przez zastosowanie operacji minimum efektywnego do sumy, która jest pierwaotnie rekurencyjna i której wyprowadzenie z funkcji najprostszych przy pomocy rekurencji pierwotnej widzieliśmy poprzednio.

Funkcje częściowo rekurencyjne, które są całkowicie określone, nazywa się funkcjami rekurencyjnymi. Można udowodnić , że funkcje rekurencyjne są ogólnie rekurencyjne i odwrotnie, czyli te dwie klasy funkcji się pokrywają.

Wszystkie funkcje ogólnie rekurencyjne n zmiennych można ustawić w ciąg f_1 , f_2, ...

Dowodu tego ważnego faktu nie będziemy tu przeprowadzać /zob. np. [3]/ ale zajmiemy się jego konsekwencjami: Podobnie, jak to zrobiliśmy dla funkcji pierwotnie rekurencyjnych, możemy dla tych funkcji utworzyć funkcję uniwersalną zbioru wszystkich funkcji ogólnie rekurencyjnych n zmiennych

$$R(k, x_1, \dots, x_n) \qquad /15/$$

Funkcja ta ma następujące własności:

1. Dla każdej ustalonej liczby naturalnej k funkcja n zmiennych $R(k, x_1, \ldots, x_n)$ jest funkcją ogólnie rekurencyjną.

2. Dla każdej funkcji ogólnie rekurencyjnej n zmiennych g istnieje taka liczba naturalna m , że

$$g(x_1, \ldots, x_n) = R(m, x_1, \ldots, x_n)$$

dla wszystkich naturalnych wartości zmiennych $x_1, \ldots, x_n$

Jeżeli utworzymy funkcję n zmiennych

$$r(x_1, x_2, \ldots, x_n) = R(x_1, x_1, x_2, \ldots, x_n) + 1 \qquad /16/$$

to zupełnie analogicznie, jak to robiliśmy w przypadku funkcji pierwotnie rekurencyjnych, możemy wykazać, że funkcja r nie jest ogólnie rekurencyjna. Stąd wynika /co znowu można udowodnić analogicznie jak poprzednio/, **że R /k, x_1, ... , x_n / , jako funkcja** wszystkich swoich n+1 argumentów, nie jest ogólnie rekurencyjna. Zarówno r jak R są funkcjami całkowicie określonymi. Ponieważ nie są ogólnie rekurencyjne, to nie są rekurencyjne, a zatem nie są także funkcjami częściowo rekurencyjnymi.

Ze sposobu wprowadzenia operacji superpozycji, rekurencji pierwotnej i minimum efektywnego dla funkcji częściowo określonych widać, że jeżeli tylko dlą funkcji , na których te operacje są wykonywane, istnieją ich wartości i potrafimy je obliczać, to potrafimy także obliczyć wartości funkcji /o ile one istnieją/, będących wynikiem zastosowania tych operacji, dla tych samych wartości argumentów, dla których obliczaliśmy wartości funkcji wyjściowych. Możemy również obliczać wartości funkcji najprostszych dla dowolnych wartości ich argumentów. Wobec tego, jeżeli tylko **znamy sposób konstruowania funkcji częściowo rekurencyjnej** z funkcji najprostszych przy pomocy operacji superpozycji, rekurencji pierwotnej i minimum efektywnego, to potrafimy obliczyć wartość tej funkcji dla dowolnych wartości jej argumentów, dla których ta wartość funkcji jest określona. Słowo "potrafimy" należy tu rozumieć w ten sposób, że moglibyśmy wykonać te obliczenia na maszynie cyfrowej, o ile nie staną nam na przeszkodzie ograniczenia pojemności dostępnej pamięci, czasu obliczeń, lub podobne czynniki natury technicznej.

4. Pojęcie algorytmu i teza Church'a

"Algorytm" jest w matematyce pojęciem pierwotnym, analogicznie jak pojęciami pierwotnymi są "zbiór" lub "element zbioru". Większość stosowanych w matematyce pojęć można ściśle zdefiniować przy pomocy pojęć prostszych, ale tego procesu przechodzenia do pojęć coraz bardziej prymitywnych nie można kontynuować bez końca. Pojęcia pierwotne nie mogą być definiowane - co najwyżej można w sposób mniej lub bardziej ścisły opisywać ich własności, aby ułatwić posługiwanie się nimi i zrozumienie /przynajmniej do pewnego stopnia/ ich sensu.

Jako taki niezbyt formalny opis pojęcia algorytmu można podać następujący zestaw jego cech:

1. Dyskretność: Algorytm jest to proces stopniowego budowania pewnych wielkości, wykonywany w następujących po sobie krokach. Przed wykonaniem pierwszego kroku tego procesu jest zadany pewien system wielkości, które będziemy nazywać danymi. W każdym kroku powstaje pewien system wielkości, otrzymywanych zgodnie z określoną zasadą /programem/, które to wielkości będziemy nazywać wynikami tego kroku.

2. Deterministyczność: Wyniki pierwszego kroku zależą jednoznacznie od danych i tylko od nich, a wyniki każdego kroku następnego zależą jednoznacznie od wyników kroku poprzedzającego.

3. Elementarność: Zasada otrzymywania wyników każdego kroku na podstawie danych lub wyników kroku poprzedniego powinna być prosta i lokalna.

4. Celowość: Jeżeli zastosowanie zasady otrzymywania wyników jakiegoś kroku z wyników kroku poprzedzającego lub z danych nie daje żadnego rezultatu, to powinno być powiedziane, co należy uważać za wynik algorytmu.

5. Masowość : Dane algorytmu można wybierać ze zbioru, który jest potencjalnie nieskończony.

Spróbujmy bliżej zbadać te cechy.
W każdym obliczeniu , wykonywanym przez człowieka ręcznie, lub przy pomocy jakichkolwiek **przyrządów** pomocniczych, lub na maszynie cyfrowej, można wyodrębnić pewne kroki czy etapy, przy czym można zupełnie dokładnie powiedzieć, jakimi danymi dysponujemy na początku i co się rozumie przez wyniki każdego kroku. Na przykład, jeżeli wykonujemy na maszynie cyfrowej jakieś obliczenie według zadanego programu, to jako taki elementarny krok możemy uważać każde wykonanie instrukcji, zawartej w tym programie . Można ściśle określić stan rejestrów i pamięci maszyny **oraz stan jej urządzeń** wejścia i wyjścia przed i po wykonaniu dowolnej instrukcji, i można określić ogólną zasadę, która mówi, jak wykonanie instrukcji te stany zmienia. Warto tu podkreślić, że przez wyniki każdego kroku rozumiemy cały system wielkości, jaki po tym kroku istnieje , a nie tylko te wielkości, które właśnie w czasie tego kroku powstały lub uległy zmianie. W tym sensie przez **wynik kroku, polegają** - cego na wykonaniu jakiejś instrukcji przez maszynę cyfrową, rozumiemy cały **stan jej pamięci rejestrów, urządzeń wejścia i wyjścia.**

Przyjmuje się, że wykonanie każdego kroku i jego wyniki zależą tylko od wyników wykonania kroku poprzedniego, jeżeli taki był, lub od danych początkowych, i od niczego więcej. Inaczej mówiąc, zakłada się, że nie ma tu żadnego wpływu czynników ubocznych ani przypadkowych. To że wyniki każdego kroku zależą tylko od systemu wielkości, który istniał przed wykonaniem tego kroku, oznacza , że cały program działania /wykonywania poszczególnych kroków/, o ile jest w jakiś sposób wyrażony, to również jest ujęty w tym systemie wielkości.

Człowiek, wykonując jakąś czynność, w szczególności jakiekolwiek obliczenia, może w każdym momencie swego postępowania poznawać, uwzględniać, kontrolować , wykorzystywać lub przekształcać tylko dość ograniczony zestaw wielkości, niezależnie od tego, czy one przedstawiają liczby, funkcje liczbowe, informacje zapisane przy pomocy tekstów , sygnały i tak dalej. Możliwości wykonywania bardzo skomplikowanych czynności też są raczej ograniczone, i to w znacznym stopniu. Dokładnie tak samo jest z wszelkimi przyrządami lub maszynami /w szczególności: maszynami cyfrowymi/, którymi człowiek może się posługiwać. Każde z tych urządzeń jest w stanie zapamiętać, przyjąć , rozpoznać, przekształcić lub w jakikolwiek sposób wykorzystać tylko bardzo ograniczoną ilość danych na raz. Dla naszych rozważań nie jest istotne, **czy np. w pamięci maszyny cyfrowej można** zawrzeć tysiąc, czy miliard liczb lub słów - istotne jest to, że jakakolwiek by to była liczba, zawsze będą istniały zadania , dla których taka pojemność pamięci jest niewystarczająca.

Tak samo zawsze będą istniały procesy - w szczególności procesy obliczeniowe - które będą tak skomplikowane, że nie będzie możliwości wykonania wszystkich związanych z nimi czynności jednocześnie, w tym samym momencie, niezależnie od tego, jak potężnymi środkami będziemy dysponowali. Nie będziemy na przykład nigdy potrafili obliczyć lub ocenić wszystkich wartości funkcji jednej lub wielu zmiennych , dla wszystkich wartości jej argumentów jednocześnie, jeżeli tylko funkcja ta będzie dostatecznie skomplikowana. Przyczyna tego jest prosta: zbiór wszystkich wartości jej argumentów jest nieskończony. Ale jeżeli nawet zechcemy ocenić wartość takiej skomplikowanej funkcji dla skończonej ilości zestawów wartości jej argumentów, to zawsze można tak tę ilość dobrać, żeby dla dostępnych środków wykonanie tych obliczeń jednocześnie nie było możliwe. Możemy za to takie zadanie z powodzeniem wykonać, jeżeli jeżeli będziemy te obliczenia robić stopniowo, rozkładając je na dłuższy czas i odpowiednią dużą ilość kroków obliczeniowych. Zadanie stanie się wtedy mniej złożone i łatwiej wykonalne dlatego, że w każdym momencie trzeba będzie uwzględniać tylko ograniczony zestaw wielkości, to znaczy w całym systemie wielkości , z jakimi mamy do czynienia, działać tylko w stosunkowo niewielkiej jego części , lokalnie, a same działania mogą być raczej proste, elementarne. Oczywiście lokalność i elementarność są tu pojęciami w pewnym sensie umownymi; nie chodzi nam o to, by zasady otrzymywania wyników jakiegokolwiek kroku algorytmu były **maksymalnie elementarne i lokalne.** W sformułowaniu cechy 3. akcentuje się sam fakt istnienia ograniczeń , które z konieczności muszą być dość ostre ze względu na ograniczone możliwości ludzi i środków, jakimi dysponują, ale dokładniejsze sprecyzowanie tych ograniczeń zależy już od rodzaju algorytmu.

W każdym procesie dla każdych danych istnieją dwie możliwości, jeżeli chodzi o przebieg tego procesu: albo po pewnym kroku on zostanie przerwany lub zatrzymany, i po tym kroku nie będzie już wykonany krok następny, albo takiego zatrzymania nie będzie, po każdym kroku będzie wykonany następny i proces będzie trwał nieskończenie. Jeżeli proces jest nieskończony, wtedy uważamy, że on nie daje wyniku, lub, inaczej mówiąc, jego wynik nie jest określony. Jeżeli natomiast proces w pewnym kroku się przerywa , to trzeba wiedzieć, co się wtedy rozumie przez wynik algorytmu /cecha 4/. Gdyby to nie było wiadome, użyteczność całego algorytmu byłaby wielce problematyczna.

W końcu cecha 5. żąda, żeby algorytm można było stosować dla dostatecznie bogatego zbioru danych - inaczej mówiąc, żąda się, żeby **proces był wykonalny nie tylko dla wyraźnie ograniczonej,** skończonej ilości przypadków

Znowu to się wiąże z użytecznością ; na ogół nikomu nie jest potrzebny proces obliczeniowy, który może obliczyć wartości jakiejś funkcji tylko dla pięciu pierwszych z brzegu wartości jej argumentów, ale dąży się do opracowywania i stosowania narzędzi możliwie mocnych i uniwersalnych. Jest to ważne szczególnie dla teorii algorytmów, która bada te zagadnienia w sposób możliwie szeroki i z punktu widzenia skrajnych możliwości algorytmów wogóle. Procesy, które nie miałyby dostatecznie szerokich możliwości, nie są po prostu z punktu widzenia tej teorii interesujące.

W algorytmach, które ta teoria omawia w swoich ogólnych rozważaniach, wymagania na lokalność i elementarność zasad wykonywania kroków algorytmów są mocniejsze, niż w teorii maszyn matematycznych lub w wielu zagadnieniach praktycznych . O przyczynach tego mówiliśmy już we wstępie: dzięki temu wszystkie rozważania teoretyczne stają się prostsze. Z drugiej strony na ogólności się przez to nic nie traci, bo wiadomo, że zmniejszenie tych wymagań na lokalność i elementarność nie doprowadziłoby do żadnych nowych i interesujących wyników ogólnych , przynajmniej w zakresie zagadnień rozstrzygalności i obliczalności, którymi się teoria algorytmów przede wszystkim zajmuje.

Mówiąc, że jakaś funkcja częściowo określona jest obliczalna przy pomocy zadanego **algorytmu**, rozumiemy, że jeżeli przyjmiemy jako dane do tego **algorytmu** dowolne wartości argumentów tej funkcji i jej wartość będzie dla nich określona, to po pewnej ilości kroków algorytm się zatrzyma i jego wynikiem będzie wartość tej funkcji dla tych wartości argumentów. Jeżeli natomiast wartość funkcji dla tych argumentów nie jest określona, to algorytm się nie zatrzyma i będzie działał nieskończenie.

Mówimy, że funkcja jest obliczalna, jeżeli istnieje algorytm, który ją oblicza.

Fundamentalną rolę w teorii algorytmów odgrywa tak zwana Teza Church'a, sformułowana przez amerykańskiego logika o tym nazwisku:

Klasa wszystkich funkcji obliczalnych jest identyczna z klasą wszystkich funkcji częściowo rekurencyjnych.

Zauważmy, że teza Church'a nie jest twierdzeniem matematycznym, które można by udowodnić lub obalić, gdyż jej treść opiera się na pojęciu algorytmu, którego nie można ściśle zdefiniować ani określić jednoznacznie wszystkich jego własności. Z tego względu nie jest

i nie może być ściśle zdefiniowane pojęcie funkcji obliczalnej, bo w jej"definicji"został użyty zwrot "istnieje algorytm, który ją oblicza". Potrafimy podać konkretne przykłady algorytmów i zrobimy to niżej, ale nie potrafimy nic dostatecznie konkretnego powiedzieć o klasie wszystkich algorytmów. Ponieważ teza Church'a nie może być poddana dowodzeniu **matematycznemu**, można powiedzieć, że należy ona raczej do filozofii ,niż do samej matematyki , choć dotyczy pojęć ściśle matematycznych oraz takich, które w matematyce są stosowane. W każdym razie bardzo szczegółowe i głębokie badania, przeprowadzane od powstania teorii algorytmów aż do dni obecnych, nie stworzyły żadnych przesłanek , które pozwoliłyby podważyć zasadność tej tezy - przeciwnie, raczej umocniły wiarę w jej słuszność.

Wartość tezy Church'a polega między innymi na tym, że pozwala ona przejść od raczej mgliście sprecyzowanego pojęcia funkcji obliczalnej do dokładnie i precyzyjnie wprowadzonego pojęcia funkcji częściowo rekurencyjnej, dzięki czemu można stosować w badaniach cały dostępny aparat logiki i matematyki. Z kolei postęp w tych badaniach pozwala bardziej przybliżyć pojęcie obliczalności i funkcji obliczalnych.

Jeżeli przyjmiemy tezę Church'a , to opierając się na tym, co zostało powiedziane w poprzednim rozdziale, możemy podać przykłady funkcji , które nie są obliczalne. Są to funkcje uniwersalne zbiorów wszystkich funkcji ogólnie rekurencyjnych n zmiennych, dla n = 1, 2, ... oraz wszystkie funkcje r /będące funkcjami n zmiennych / , określone wzorem /16/.

5. Maszyny Turinga

Pojęcie maszyny Turinga zostało wprowadzone w latach 1936-37 przez A.M. Turinga oraz /w nieco innej formie/ przez E.L. Posta. Jest to maszyna abstrakcyjna i nikt nigdy nie próbował jej budować w postaci rzeczywistego urządzenia, gdyż byłoby to zbędne ; jest to po prostu pewna forma algorytmu, użyteczna w różnych rozważaniach teoretycznych. Nazwa "maszyna" powstała stąd, że u podstaw tej koncepcji leżała idea "mechanizacji", czy,jak raczej powiedzielibyśmy dzisiaj, "algorytmizacji" obliczeń. Zauważmy, że pierwsze maszyny cyfrowe / w tym sensie, w jakim teraz rozumiemy to pojęcie/, powstały znacznie później. W świetle dzisiejszych pojęć można by z pewną dozą słuszności powiedzieć, że maszyna Turinga jest to maszyna cyfrowa, w której elementarność i lokalność zasad wykonywania każdego kroku obliczeniowego została posunięta prawie do kresu możliwości.

Maszyna Turinga jest wyposażona w "taśmę", będącą rodzajem pamięci. Jest to skończony ciąg komórek, ustawionych obok siebie w ten sposób, że każda komórka, oprócz dwu skrajnych, sąsiaduje z jedną komórką z lewej strony i z jedną komórką z prawej. Dla każdej maszyny Turinga jest określony alfabet - skończony zbiór, którego elementy będziemy nazywać symbolami. Jeden, wyróżniony element tego alfabetu jest nazywany symbolem pustym. W każdej komórce taśmy jest zawsze zapisany jeden symbol. Komórki, w których jest zapisany symbol pusty, będziemy nazywać pustymi. W czasie pracy maszyny symbole, zapisane w komórkach taśmy, można zastępować innymi. W szczególności można wpisywać nowe symbole do komórek pustych. W każdej chwili jedna komórka taśmy jest wyróżniona - jest to "pole widzenia" maszyny. To pole widzenia może przesuwać się wzdłuż taśmy o jedną komórkę w prawo lub w lewo. Przesunięcie pola widzenia poza skraj taśmy powoduje dobudowanie do niej nowej komórki pustej, sąsiadującej z tą, która była przedtem komórką skrajną i polem widzenia - po przesunięciu polem widzenia staje się ta nowo dołączona komórka. W ten sposób pole widzenia pozostaje zawsze w obrębie taśmy.

Każda maszyna Turinga posiada skończony zbiór stanów oraz program. Program maszyny Turinga jest to skończony ciąg rozkazów, a każdy rozkaz jest ciągiem, złożonym z pięciu następujących elementów /w kolejności ich zapisania/: stan, symbol, stan, symbol, jedna z liter: P , L lub N. W czasie pracy maszyny Turinga jej rozkazy są wykonywane w następujący sposób : jeżeli maszyna jest w stanie, który jest podany na pierwszym miejscu rozkazu, a w jej polu widzenia znajduje się symbol, wymieniony na drugim miejscu, to maszyna przechodzi do stanu, wymienione go na trzecim miejscu rozkazu, wpisuje w swoim polu widzenia symbol, zapisany na czwartym miejscu i przesuwa swoje pole widzenia stosownie do litery , znajdującej się na ostatnim miejscu rozkazu: w lewo, jeżeli to jest litera L, w prawo, jeśli P , lub nie przesuwa go, pozostawiając w tej samej komórce, jeżeli to jest litera N. Praca maszyny Turinga jest podzielona na kroki. W każdym kroku są przeglądane kolejne rozkazy programu, zaczynając od pierwszego, aż natrafi się na taki, w którym dwa pierwsze elementy są identyczne odpowiednio z aktualnym stanem maszyny i z zawartością jej pola widzenia ; ten rozkaz jest wykonywany, po czym przechodzi się do wykonania następnego kroku. Jeżeli takiego rozkazu nie ma, maszyna staje.

W zbiorze stanów maszyny jest wyróżniony jeden stan, zwany stanem początkowym. Przyjmuje się, że na początku pracy maszyna jest właśnie w tym stanie; zmiany stanów w kolejnych krokach są jednoznacznie określone przez program i zawartość taśmy maszyny.

Jako przykład weźmy maszynę Turinga, której alfabet składa się z liter a, g, l, m, o, p, r, t, y oraz symbolu pustego § . Ma ona dwa stany, które oznaczymy 1, 2 . Jej program:

```
1a  1§P
1g  1rP
1l  1pP
1m  1mP
1o  1oP
1p  1pP
1r  1gP
1t  1aP
1y  1rP
1§  2§N
```

Zobaczmy, jak ta maszyna będzie działać, jeżeli na początku w kolejnych komórkach taśmy, licząc od lewej, są zapisane symbole a, l, g, o, r, y, t, m, i w polu widzenia maszyny jest symbol a.

Aby ułatwić sobie śledzenie pracy maszyny , będziemy zapisywać dane i wyniki kolejnych kroków w następujących po sobie wierszach, pisząc obok siebie stan maszyny i aktualną zawartość taśmy, czyli symbole, znajdujące się w jej kolejnych komórkach. Symbol, będący w polu widzenia maszyny, jest podkreślony.

stan	taśma
1	algorytm
1	§lgorytm
1	§pgorytm
1	§prorytm
1	§prorytm
1	§progytm
1	§progrtm
1	§programm
1	§program§
2	§program§

Ponieważ nie ma w programie maszyny rozkazu, który miałby na początkowym miejscu stan 2, działanie maszyny na tym się kończy, a wynikiem jej działania jest ostatnia zawartość taśmy. Ale prześledźmy szczegółowo działanie maszyny od samego początku.

Pierwszy wiersz opisuje sytuację początkową: maszyna jest w stanie 1 i widzi literę a słowa algorytm, zapisanego na taśmie. W tej sytuacji zostaje wykonany pierwszy rozkaz programu: stan maszyny się nie zmienia, w polu widzenia maszyny zostaje wpisany pusty symbol §, a samo pole widzenia przesuwa się o jedną komórkę w prawo. Sytuacja **po wykonaniu tego kroku jest pokazana w drugim** wierszu. Teraz maszyna jest w stanie 1 i widzi symbol l, a zatem zostaje wykonany rozkaz 1l 1pP : stan maszyny się nie zmienia, l zostaje zastąpione przez p, pole widzenia przesuwa się o 1 komórkę w prawo i powstaje sytuacja , pokazana w trzecim wierszu. Następnie zostaje wykonany rozkaz 1g 1rP i tak dalej, aż pole widzenia maszyny dotrze do litery m. Teraz zostaje wykonany rozkaz 1m 1mP , czyli zarówno stan maszyny, jak symbol w polu widzenia pozostają bez zmian, a samo pole widzenia przesuwa się w prawo. Ale ponieważ

komórka z symbolem m jest na prawym krańcu taśmy, takie przesunięcie musi być połączone z dobudowaniem do taśmy nowej komórki pustej, co zostało pokazane w przedostatnim wierszu. Po tym kroku zostaje wykonany rozkaz 1§ 2§N , który powoduje tylko zmianę stanu maszyny. Dalej, jak powiedzieliśmy już wyżej, praca maszyny się kończy.

Stosując tę samą metodę, Czytelnik może sam sprawdzić, jak będzie działać ta maszyna, jeżeli na początku będzie sytuacja

algol

Często stosuje się skróconą formę zapisywania programów. Pierwsze dwa elementy każdego rozkazu pozostają niezmienione, ale następnie opuszcza się stan /na trzecim miejscu/ jeżeli rozkaz go nie zmienia, symbol wpisywany, jeżeli rozkaz powoduje wpisanie takiego samego symbolu, jaki był poprzednio, oraz opuszcza się literę N. Wygodnie jest oddzielać dwa pierwsze elementy każdego rozkazu od pozostałych wyraźnym odstępem lub jakimś znakiem, różnym od oznaczenia stanu, symbolu i od liter P,L . Takim znakiem oddzielającym może być na przykład skośna kreska / lub pozioma strzałka → .

W przytoczonym przykładzie programu nie było rozkazów, zaczynających się od stanu 2. Oznacza to, że w każdej sytuacji, w której maszyna znajdzie się w stanie 2 , zostanie zatrzymana. Stan , posiadający tę własność, można nazwać stanem końcowym. W każdym zbiorze stanów maszyny Turinga można wyodrębnić dokładnie jeden taki stan końcowy, bo gdyby w jej programie było więcej takich stanów, od których rozkazy się nie zaczynają, to można by je oznaczyć tym samym znakiem , łącząc je w ten sposób w jeden stan - działania maszyny w dowolnej sytuacji to nie zmieni. Często oznacza się stan końcowy jakimś specjalnym znakiem wyróżniającym, na przykład wykrzyknikiem !

Podany wyżej program będzie w zapisie skróconym wyglądał tak:

```
1a  §P
1g  rP
1l  pP
1m  P
1o  P
1p  P
1r  gP
1t  aP
1y  rP
1§  !
```

Maszyny Turinga można wykorzystywać do wyznaczania wartości funkcji liczbowych dla zadanych wartości ich argumentów. Oto przykład: maszyna, dodająca 1 do liczby, zapisanej w systemie dziesiętnym.

Alfabet maszyny składa się z cyfr 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 i znaku pustego ø . Stany maszyny wygodniej będzie tym razem oznaczyć literami: a /stan początkowy/, b, c, d /stan końcowy/. Program.

a0	P	b0	c1L	c0	L
a1	P	b1	c2L	c1	L
a2	P	b2	c3L	c2	L
a3	P	b3	c4L	c3	L
a4	P	b4	c5L	c4	L
a5	P	b5	c6L	c5	L
a6	P	b6	c7L	c6	L
a7	P	b7	c8L	c7	L
a8	P	b8	c9L	c8	L
a9	P	b9	0L	c9	L
aø	bL	bø	c1L	cø	dP

Na początku pracy maszyna ma w polu widzenia lewą skrajną cyfrę liczby, zapisanej na taśmie.

Zbadajmy działanie maszyny na przykładach:

stan	taśma
a	21309
a	21309
a	21309
a	21309
a	21309
a	21309ø
b	21309ø
b	21300ø
c	21310ø/
c	21310ø
c	21310ø
c	ø21310ø
d	ø21310ø

Wynikiem działania maszyny jest 21310 /znaki puste, stojące przy krańcach taśmy, pomijamy/

stan	taśma
a	999
a	999
a	999
a	999ø
b	999ø
b	990ø
b	900ø
b	ø000ø
c	ø1000ø
d	ø1000ø

stan	taśma
a	0
a	0ø
b	0ø
c	ø1ø
d	ø1ø

Jak widać, maszyna nie **tylko** dodaje 1 do liczby, zapisanej na taśmie, ale także za każdym razem przesuwa swoje pole widzenia na pierwszą /najstarszą/ cyfrę wyniku. W tej maszynie każdy stan określa jednoznacznie rodzaj czynności, jaką maszyna , będąc w nim, wykonuje:

a poszukiwanie prawego końca liczby

b dodanie 1 do cyfry, znajdującej się w polu widzenia maszyny i przesunięcie pola widzenia w lewo

c powrót do lewego końca liczby

d stop.

Z tych przykładów widać, że nawet tak proste działanie arytmetyczne, jak dodawanie jedynki do liczby, musiało być rozłożone na szereg elementarnych czynności, bo maszyna w żadnym momencie nie może objąć całej liczby, a tylko widzi jej pojedyncze cyfry. W związku tym do właściwego dodawania jedynki, z ewentualnym przeniesieniem do wyższej pozycji /jak to było w przypadku liczby 999/ , dochodzą jeszcze czynności pomocnicze, zresztą bardzo proste: odszukanie miejsca, od którego należy zacząć dodawanie, a następnie miejsca, na które należy"wrócić" .

Ta maszyna Turinga oblicza funkcję $s(x) = x + 1$. Czytelnik

może się sam przekonać, że maszyna Turinga, mająca ten sam alfabet, której stanami są : a /stan początkowy/ , b oraz c /stan końcowy/ , i która ma program podany niżej, oblicza funkcję $o(x)$.

a0	P	b0	øL
a1	P	b1	øL
a2	P	b2	øL
a3	P	b3	øL
a4	P	b4	øL
a5	P	b5	øL
a6	P	b6	øL
a7	P	b7	øL
a8	P	b8	øL
a9	P	b9	øL
aø	bL	bø	c0

Tak, jak w poprzednim przypadku, symbole puste w wyniku odrzucamy.

Gdybyśmy chcieli utworzyć maszynę Turinga, która wykonuje dodawanie jedynki do liczby, zapisanej w systemie dwójkowym,to otrzymalibyśmy maszynę prostszą od poprzedniej, choć ogólna zasada pozostaje taka sama. Alfabet tej maszyny składa się z symboli 0, 1, ø /symbol pusty/. Zbiór stanów: a /stan początkowy/ , b, c, d /stan końcowy/. Program:

a0	P	b0	c1L	c0	L
a1	P	b1	0L	c1	L
aø	bL	bø	c1L	cø	dP

A oto przykłady jej działania:

stan	taśma
a	11011
a	11011
a	11011
a	11011
a	11011
a	11011ø
b	11011ø
b	11010ø
b	11000ø
c	11100ø
c	11100ø
c	øT1100ø
d	ø11100ø

stan	taśma
a	1̲11
a	11̲1
a	111̲
a	111ø̲
b	111̲ø
b	110̲ø
b	100̲ø
b	ø̲000ø
c	ø̲1000ø
d	ø1̲000ø

Sprawdzmy poprawność tych wyników; w tym celu wystarczy przeliczyć liczby, zapisane w systemie dwójkowym, na system dziesiętny.

Liczba, zapisana dwójkowo 11011 , jest równa
$1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 = 1 + 2 + 8 + 16 = 27$.
Korzystamy tutaj z następującego ogólnego wzoru: jeżeli liczba, zapisana w systemie dwójkowym , ma n cyfr i jej cyframi, poczynając od najmniej znaczącej, są a_0, a_1, ... ,a_{n-1} , to jest ona równa $\sum_{i=0}^{n-1} 2^i \cdot a_i$.
Liczba, zapisana dwójkowo 11100 , jest równa
$0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 = 4 + 8 + 16 = 28$.
Podobnie 111 oznacza w zapisie dziesiętnym 7, zaś 1000 oznacza 8.

Napisanie programu maszyny Turinga, obliczającej funkcję o(x) dla liczb, zapisanych w systemie dwójkowym, pozostawiamy Czytelnikowi.

Najprostszym - z punktu widzenia złożoności maszyn Turinga - jest zapis liczb "pałeczkowy" . Każda liczba naturalna n zostaje zapisana jako ciąg napisanych obok siebie n+1 jedynek. Tak na przykład 1 oznacza liczbę 0, 111 liczbę 2 , 11111 oznacza 4 i tak dalej. W tym systemie zapisu, aby dodać jedynkę do jakiejkolwiek liczby, wystarczy dopisać symbol 1 do którejkolwiek strony liczby.

Maszyna Turinga, obliczająca funkcję s(x) w systemie "pałeczkowym" będzie miała alfabet, złożony z symbolu 1 i symbolu pustego 0 , będzie miała dwa stany: a /początkowy/ i b /końcowy/ , oraz program

a1 L
a0 b1

Obliczenie s(5) na tej maszynie:

stan	taśma
a	111111
a	0111111
b	1111111

Obliczenie s(0):

stan	taśma
a	1
a	01
b	11

Maszyna Turinga, obliczająca sumę dwu liczb w zapisie pałeczkowym, ma alfabet złożony z symbolu pustego O, cyfry 1 i symbolu oddzielającego składniki sumy + . Stany: a /początkowy/, b, c, d /końcowy/. Program:

a1 P
a+ P
a0 bL
b1 c0L
c1 L
c+ 1L
c0 dP

Zakładamy, że na początku maszyna ma w polu widzenia najbardziej skrajną lewą jedynkę pierwszego składnika. W programie nie ma rozkazów, zaczynających się od b+ i b0, bo przy prawidłowo zapisanych danych one nie mają zastosowania. Gdyby jednak wskutek jakiejś nieprawidłowości doszło do tego, że na taśmie byłyby takie dane i że maszyna w stanie b zobaczyłaby 0 lub +, to jej działanie zostanie przerwane.

Przykład działania tej maszyny:

stan	taśma
a	111+1111
a	111+1111
a	111+1111
a	111+1111
a	111+1111
a	111+1111
a	111+1111
a	111+1111
a	111+1111
a	111+11110

```
b        111+11110
c        111+11100
c        111+11100
c        111+11100
c        111+11100
c        111111100
c        111111100
c        111111100
c        011111100
d        011111100
```

Jak widać, maszyna wykonuje to dodawanie w ten sposób, że znajduje prawy koniec drugiego składnika, odrzuca od niego jedną jedynkę, po czym wstawia 1 zamiast + i wraca na początek wyniku. Ze stanami maszyny są związane następujące czynności:

a poszukiwanie prawego końca drugiego składnika

b usunięcie końcowej jedynki drugiego składnika i przesunięcie pola widzenia w lewo o jedną komórkę

c powrót na początek i zastąpienie + przez 1

d stop.

Mówiąc, że maszyna Turinga oblicza funkcję n zmiennych, częściowo określoną, $f(x_1, \ldots, x_n)$, mamy na myśli następującą własność tej maszyny:

Jeżeli na początku pracy maszyny na taśmie jest zapisane n liczb $x_1, \ldots, x_n$, oddzielonych przecinkami, lub innymi symbolami, różnymi od symbolu pustego i symboli, używanych do zapisu liczb,

jeżeli na początku pracy w polu widzenia maszyny znajduje się skrajna lewa cyfra pierwszej z tych liczb,

jeżeli dla takich liczb wartość $f(x_1, \ldots, x_n)$ jest określona,

to maszyna po pewnej ilości kroków dojdzie do stanu "stop" /stanu końcowego/ i wtedy na taśmie będzie zapisana wartość tej funkcji.

Jeżeli natomiast wartość funkcji dla zapisanych na taśmie wartości jej argumentów nie będzie określona, to maszyna będzie działać nieskończenie i nigdy do stanu końcowego nie dojdzie /ani jej działanie w żaden inny sposób nie będzie przerwane/.

Sposób zapisywania liczb na taśmie maszyny Turinga nie odgrywa przy tym roli - może to być zapis dziesiętny, dwójkowy, pałeczkowy lub jakikolwiek inny, ale jeżeli już na jakiś się zdecydujemy, to we wszystkich rozważaniach należy stosować ten sam system zapisu, żeby nie doprowadzić do niepotrzebnych komplikacji i nieporozumień. Można udowodnić, że jeżeli maszyna Turinga oblicza funkcję przy jakimkolwiek z tych zapisów liczb, to istnieje także maszyna Turinga, która oblicza tę samą funkcję w innym z tych zapisów.

Widzieliśmy, że przy pomocy maszyn Turinga można obliczać funkcje najprostsze $s(x)$ i $o(x)$. Można udowodnić, że istnieją maszyny Turinga , obliczające funkcje najprostsze I_m^n /po jednej maszynie na każdą z tych funkcji/

jeżeli istnieją maszyny Turinga, obliczające jakieś funkcje częściowo określone, to istnieją także maszyny Turinga, obliczające funkcje , które powstają z tych poprzednich przez zastosowanie operacji superpozycji , rekurencji pierwotnej i minimum efektywnego.

Stąd już bezpośrednio wynika wniosek, że dla kazdej funkcji częściowo rekurencyjnej istnieje maszyna Turinga, która ją oblicza.

Można także udowodnić, że każda funkcja, obliczalna na maszynie Turinga /to znaczy taka, że istnieje maszyna Turinga, która ją oblicza/ jest częściowo rekurencyjna.

Szczegółowe dowody Czytelnik może znaleźć w literaturze wymienionej na końcu.

Jak widać, twierdzenie o tym, że klasa funkcji obliczalnych na maszynach Turinga jest identyczna z klasą wszystkich funkcji częściowo rekurencyjnych, jest całkowicie zgodne z tezą Church'a.

6. Algorytmy Markowa

A.A. Markow, matematyk radziecki, wprowadził w latach 1950-54 nowy rodzaj algorytmu, opartego na zupełnie innej zasadzie, niż maszyna Turinga. Markow nazwał te algorytmy normalnymi, ale są one również powszechnie znane pod nazwą: algorytmy Markowa.

Niech będzie dany dowolny skończony zbiór, który nazwiemy alfabetem. Jego elementy będziemy nazywać **symbolami** lub literami. Załóżmy, że do tego alfabetu nie należą znaki . /kropka/ ani → /strzałka/. Dowolne skończone ciągi symboli /należących do tego alfabetu/ będziemy nazywać słowami. Słowem jest także ciąg bez elementów, to znaczy nie zawierający żadnej litery - takie słowo nazywamy pustym.

Algorytmem Markowa /w zadanym alfabecie/ nazywamy każdy skończony ciąg uporządkowanych par słów, przy czym słowa w każdej parze są przedzielone znakiem → lub znakami →. /strzałka z kropką/. Dla skrócenia wysłowienia elementy algorytmu Markowa /czyli te pary uporządkowane/ będziemy nazywać regułami . Pierwsze słowo każdej reguły nazywamy jej poprzednikiem, drugie następnikiem. Poprzednik i następnik reguły może być słowem pustym.

Regułę algorytmu Markowa można zastosować do danego słowa, jeżeli poprzednik tej reguły jest jego częścią. Wtedy przekształcenie słowa przy pomocy tej reguły polega na tym, że przeglądając je od lewej strony, znajdujemy w nim pierwsze występowanie poprzednika reguły i podstawiamy zamiast tej części słowa następnik tej reguły. Jeżeli ten następnik jest słowem pustym, to podstawienie go zamiast jakiejś części słowa oznacza po prostu usunięcie tej części. Jeżeli poprzednik reguły jest słowem pustym, to można ją zastosować do każdego słowa i jej wykonanie polega na dopisaniu jej następnika do początku tego słowa. W szczególności, jeżeli jej następnik jest też słowem pustym, to zastosowanie jej nie daje żadnych zmian. Do słowa pustego można zastosować tylko regułę, której poprzednik jest słowem pustym. Wtedy wynikiem działania tej reguły jest jej następnik.

Wykonanie algorytmu Markowa na danym słowie polega na tym, że przeglądając kolejno wszystkie reguły algorytmu, znajdujemy pierwszą z nich, którą można do tego słowa zastosować. Przekształcamy je przy pomocy tej reguły i w rezultacie otrzymujemy jakieś nowe słowo. Jeżeli zastosowana reguła miała tylko strzałkę bez kropki, to całe postępowanie powtarza się od nowa, wykonując algorytm na tym nowym słowie. Jeżeli zastosowana reguła miała znaki →. /strzałka z kropką/, to na jej wykonaniu działanie algorytmu się kończy , a wynikiem algorytmu jest wynik jej zastosowania. Reguły ze znakami →. nazywamy regułami kończącymi.

Jako przykład weźmy alfabet, składający się ze wszystkich cyfr 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, oraz z liter a, b .
Weźmy następujący algorytm Markowa:

a0 → 0a
a1 → 1a
a2 → 2a
a3 → 3a
a4 → 4a
a5 → 5a
a6 → 6a
a7 → 7a
a8 → 8a
a9 → 9a
a → b
0b →. 1
1b →. 2
2b →. 3
3b →. 4
4b →. 5
5b →. 6
6b →. 7
7b →. 8
8b →. 9
9b → b0
b →. 1
→ a

Poprzednikiem ostatniej reguły tego algorytmu jest słowo puste.
Zbadajmy działanie tego algorytmu na przykładach.

Słowo 21309 nie zawiera litery **a ani b, a więc jedyną regułą,** którą można do niego zastosować, jest ostatnia. Jej wynikiem będzie słowo a21309 . Ponieważ nie jest to reguła kończąca, stosujemy algorytm do tego słowa. Pierwszą z reguł, którą można do niego zastosować, jest a2 → 2a - w wyniku jej zastosowania otrzymujemy słowo 2a1309 . Teraz znowu pierwszą regułą, którą tu można zastosować, jest a1 → 1a i wobec tego zastępujemy część a1 tego słowa przez 1a , otrzymując słowo 21a309 . Wypiszmy wyniki wszystkich kroków algorytmu:

21309
a21309
2a1309
21a309
213a09
2130a9
21309a
21309b
2130b0
21310

A oto wyniki kolejnych kroków algorytmu, działającego na słowie 999:

999
a999
9a99
99a9
999a
999b
99b0
9b00
b000
1000

I w końcu działanie algorytmu na słowo 0 :

0
a0
0a
0b
1

Śledząc kolejne kroki algorytmu w tych przykładach, widzimy, że jeżeli litera jest na lewym krańcu lub w środku liczby, to znajduje zastosowanie jedna z pierwszych dziesięciu reguł, powodując przesunięcie litery a dalej w prawo. Jeżeli a jest na prawym krańcu liczby, wtedy żadnej z tych dziesięciu reguł zastosować nie można, ale za to może być zastosowana reguła jedenasta, zamieniająca a na b. Zauważmy, że reguła ostatnia, która w zasadzie może być zastosowana do każdego słowa, jest w tych przykładach wykorzystana tylko raz, w pierwszym kroku. Później już do jej wykorzystania nie dochodzi, bo zawsze znajduje zastosowanie któraś z reguł poprzedzających. Pojawienie się b na prawym krańcu liczby powoduje dodanie 1 do cyfry, poprzedzającej tę literę, a jeżeli w wyniku tego dodawania powstało przeniesienie, to b przesuwa się o jedno miejsce w lewo . W przeciwnym przypadku b znika i algorytm kończy działanie. Jeżeli przeniesienia sięgną poza liczbę, to w momencie likwidowania b zostaje do niej dopisana z lewej strony jedynka.

Jak widać z tych przykładów /i jak to można udowodnić dla każdej liczby/ ten algorytm Markowa oblicza funkcję s(x) dla liczb w zapisie dziesiętnym. Algorytm,obliczający tę samą funkcję dla liczb w zapisie dwójkowym, jest znacznie prostszy, bo wystarczy ograniczyć się do alfabetu 0, 1, a, b i ilość reguł znacznie zmaleje.

a0 → 0a
a1 → 1a
a → b
0b →. 1
1b → b0
b →. 1
→ a

Przykłady działania tego algorytmu:

11011
a1101
1a101
11a01
110a1
1101a
11011a
11011b
1101b0
110b00
11100

Ten sam algorytm, działający na słowo 111:

111
a11
1a11
11a1
111a
111b
11b0
1b00
b000
1000

Jak widać, w obu tych zapisach liczb algorytm dodawania jedynki do liczby działa w ten sposób, że najpierw odszukuje koniec liczby, od którego należy zacząć dodawanie, a następnie wykonuje je. W zapisie pałeczkowym takie odszukiwanie końca liczby jest niepotrzebne. W tym przypadku alfabet składa się z jednego symbolu 1 , a algorytm tylko z jednej reguły

1 →. 11

Podobnie prosty jest algorytm dodawania dwu liczb, zapisanych w systemie pałeczkowym; alfabet składa się z dwóch symboli 1,+ , zaś algorytm z jednej reguły kończącej

+ →.

powodującej po prostu usunięcie plusa, dzielącego obie liczby.

W algorytmie obliczania funkcji o(x) dla zapisu pałeczkowego wykorzystujemy alfabet dwuelementowy 1, a .

a1 → a
a →. 1
→ a

Oto przykład działania tego algorytmu: obliczenie o(4).

11111
a1111
a1111
a111
a11
a1
a
1

a jest tu symbolem pomocniczym, podobnie jak były pomocniczymi symbolami a i b w algorytmach dla funkcji s(x).
Opierając się na tej samej zasadzie wykorzystania symbolu pomocniczego a Czytelnik może napisać algorytmy obliczania funkcji o(x) dla dwójkowego i dziesiętnego zapisu liczb.

A oto przykład bardziej skomplikowany: obliczanie iloczynu dwu liczb w systemie pałeczkowym. Alfabet: 1, x, X, a, b, c .
Algorytm:

b1 → 1ab
a1 → 1a
1X → Xb
X → c
c1 → c
cb → c
ca → 1c
c → .
1x1 → xa

Przykład mnożenia dwu liczb, różnych od 0 i 1:

1111x11
111Xa11
111X1a1
111X11a
11Xb11a
11X1ab1a
11X1a1ab
11X11aaba
1Xb11aaba
1X1ab1aaba
1X1a1abaaba
1X11aabaaba
Xb11aabaaba
X1ab1aabaaba
X1a1abaabaaba
X11aabaabaaba
c11aabaabaaba
c1aabaabaaba
caabaabaaba
1cabaabaaba
11cbaabaaba
11caabaaba
111cabaaba
1111cbaaba
1111caaba

```
      11111caba
      111111cba
      111111ca
      1111111c
      1111111
```

Mnożenie przez zero /2 przypadki/:

```
      1x11111                 111x1
      Xa1111                  11Xa
      X1a111                  1Xba
      X11a11                  Xbba
      X111a1                  cbba
      X1111a                  cba
      c1111a                  ca
      c111a                   1c
      c11a                    1
      c1a
      ca
      1c
      1
```

Czytelnik może sam sprawdzić działanie tego algorytmu dla innych liczb, w szczególności dla mnożenia liczb przez 1.

Mówiąc , że algorytm Markowa oblicza funkcję częściowo określoną n zmiennych $f(x_1, x_2, \ldots, x_n)$ rozumiemy przez to, że algorytm, działający na ciąg liczb $x_1, x_2, \ldots, x_n$, zapisanych obok siebie i odzielonych przecinkami,/lub innymi symbolami, nie używanymi do zapisu liczb/ tak, że tworzą jedno słowo w alfabecie algorytmu, po wykonaniu pewnej ilości kroków zatrzymuje się, a jego wynikiem jest wartość funkcji, jeżeli ta wartość dla tych wartości jej argumentów jest określona, oraz nie zatrzymuje się, działając nieskończenie, jeżeli wartość funkcji dla tych liczb nie jest określona.

Sposób zapisywania liczb nie odgrywa przy tym roli - może być wybrany dowolnie, ale powinien być ustalony.

Mówimy, że funkcja częściowo określona jest obliczalna przy pomocy algorytmu Markowa, jeżeli istnieje algorytm Markowa, który ją oblicza.

Można udowodnić /zob. np. [3] / , że jeżeli istnieje maszyna Turinga, która oblicza daną funkcję częściowo określoną przy jakiejś reprezentacji liczb, to istnieje także algorytm Markowa, który oblicza tę samą funkcję przy tym samym sposobie przedstawiania liczb. Także prawdziwe jest twierdzenie odwrotne: każda funkcja , obliczalna

przy pomocy algorytmu Markowa, jest obliczalna na maszynie Turinga przy tym samym przedstawieniu liczb.

Stąd bezpośrednio wynika wniosek: klasa wszystkich funkcji, obliczalnych przy pomocy algorytmów Markowa, klasa wszystkich funkcji, obliczalnych na maszynie Turinga oraz klasa wszystkich funkcji częściowo rekurencyjnych są identyczne.

Ten wniosek jest znowu zgodny z tezą Church'a.

Wydrukowano w 100 egz.

Egz. nr 1 - 100 - Bibl.Jawna

Dnia 14.02.1969r.
Nr ks. 3670/WW
Druk ASG-O-XV-4021

Literatura:

1. Davis M. Computability and unsolvability, New York 1958. /w jęz. ang/
2. Grzegorczyk A. Zagadnienia rozstrzygalności , Warszawa 1957.
3. Malcew A.I. Algoritmy i rekursiwnye funkcii , Moskwa 1965. /w jęz. ros./
4. Nagel E., Newman J.R. Twierdzenie Gödla . Biblioteka Omega, Warszawa 1966.
5. Roger H. Theory of recursive functions and effective computability. New York 1967 /w jęz. ang./
6. Trachtenbrot B.A. Algorytmy i automatyczne rozwiązywanie zadań . Warszawa 1961.

Spis treści