Reprezentacja dyskretnych procesów obliczeniowych w wybranych modelach obliczeń analogowych

Rozprawa doktorska

Monika Piekarz

Promotor
dr hab. Przemysław Stpiczyński

Gliwice 2011

Dziękuję serdecznie doktorowi hab. Przemysławowi Stpiczyńskiemu za opiekę, pomoc i czas poświęcony w toku realizacji przewodu doktorskiego. Dziękuję również doktorowi Jerzemu Mycce za owocne dyskusje, cenne sugestie i uwagi oraz czas poświęcony na lekturę mojej pracy.

Spis treści

Wstęp

Obecne w badaniach naukowych poszukiwanie nowych alternatywnych podejść do obliczalności spowodowało zainteresowanie obliczeniami analogowymi, które mają pewne zalety, w odniesieniu do obliczeń dyskretnych. Po pierwsze są szybkie. Działanie maszyn analogowych polega na wykorzystaniu pewnych zjawisk fizycznych do realizacji procesów obliczeniowych (dyfuzja gazów lub cieczy, przewodzenie prądu elektrycznego). Takie procesy mają miejsce w różnych systemach fizycznych. Zmiany zachodzące w konkretnym systemie fizycznym stanowią obliczenie maszyny analogowej. Analogowe procesy fizyczne charakteryzują się tym, iż zmiany w nich zachodzą w sposób ciągły, co można interpretować jako przejście przez nieskończoną liczbę stanów. Dzięki obliczeniom analogowym dostajemy również inną jakość obliczeń, ponieważ ich natura pozwala na przetwarzanie informacji ciągłej. Dlatego w przypadku komputerów analogowych, możemy mówić o operowaniu na pełnych reprezentacjach liczb rzeczywistych w trakcie obliczeń (zamiast na ich przybliżeniach wymiernych).

Zważywszy na zalety obliczeń analogowych, warto zbadać relację zachodzącą pomiędzy tym co obliczalne w klasycznym rozumieniu przez maszyny cyfrowe, a tym co obliczalne za pomocą maszyn analogowych. Zbadanie tych relacji może wskazać nam nowy kierunek powszechnych dziś, poszukiwań niekonwencjonalnych modeli komputerów oraz ich możliwych do realizacji odpowiedników, które pozwoliłyby na zastąpienie, w niektórych konkretnych przypadkach zastosowań, komputerów cyfrowych, urządzeniami alternatywnymi, tańszymi w konstrukcji czy też szybszymi w działaniu. W klasycznej teorii obliczeń istnieje również wiele problemów złożonościowych, które od dłuższego czasu nie mogą znaleźć rozwiązania opartego na modelach dyskretnych. Przeniesienie ich na pole modeli obliczeń analogowych dostarcza nowych możliwości znalezienia odpowiedzi na takie zagadnienia.

Zagadnienie porównania obliczeń analogowych i obliczeń dyskretnych by-

ło do tej pory wielokrotnie i w różnych wersjach omawiane między innymi przez M. L. Campagnolo, C. Moore'a, P. Koirana i innych (prace [6, 17, 33]). Obecnie w literaturze możemy spotkać kilka modeli obliczeń analogowych podobnie jak to ma miejsce w przypadku modeli obliczeń dyskretnych. Najszerzej dyskutowane z nich to model GPAC (ang. *General Purpose Analog Computer*, GPAC) wprowadzony przez C. Shannona w 1941 roku [68], model EAC (ang. *Extendend Analog Computer*, EAC) wprowadzony przez L. A. Rubla w 1993 roku [67] oraz rekurencyjne funkcje rzeczywiste po raz pierwszy zdefiniowane przez C. Moore'a w 1996 roku [47]. Niestety, do tej pory nie znane są dokładne relacje między klasami funkcji obliczalnych w poszczególnych z tych modeli.

Niniejsza rozprawa dotyczy związków między podstawowymi, klasycznymi modelami obliczeń, a analogowymi systemami liczącymi. Rozważane klasyczne modele to maszyna Turinga oraz funkcje częściwo rekurencyjne. Z kolei analogowe systemy obliczeń, z których korzystano w prezentowanych badaniach to rozszerzony komputer analogowy - model EAC oraz rzeczywiste funkcje rekurencyjne.

Celem rozprawy była integracja różnych typów obliczeń: konwencjonalnych (dyskretnych) i niekonwencjonalnych (analogowych). Dlatego też skupiliśmy się na zbadaniu zależności między analogowymi systemami liczącymi, a tym co obliczalne w klasycznym rozumieniu. W szczególności naszym celem było zbadanie następujących związków między:

- rzeczywistymi funkcjami rekurencyjnymi, a funkcjami obliczalnymi przez maszynę Turinga,
- funkcjami analitycznymi zmiennej rzeczywistej, a funkcjami obliczalnymi przez maszynę Turinga,
- funkcjami obliczalnymi w modelu EAC, a funkcjami obliczalnymi przez maszynę Turinga,
- funkcjami obliczalnymi w modelu EAC, a funkcjami i zbiorami rekurencyjnymi w klasycznym rozumieniu.

Teza rozprawy

Klasyczny problem stopu jest równoważny zagadnieniu rozstrzygania pustości zbioru w modelu EAC.

W niniejszej pracy wyróżnić możemy następujące etapy realizacji celu nadrzędnego. Przedstawiony został przegląd obecnych w literaturze modeli obliczeń dyskretnych oraz modeli obliczeń analogowych. Zaprezentowane zostały definicje poszczególnych modeli oraz ich podstawowe własności. W szczególności podana została formalna, wyrażona w matematycznych terminach, definicja modelu EAC (definicja 2.3.2) zaprezentowanego po raz pierwszy przez Rubla w pracy [67]. Sformalizowanie definicji daje możliwość prowadzenia badań modelu EAC na gruncie matematycznym. W następnej kolejności zaprezentowane zostały w paragrafie 3.2 rekurencyjne funkcje rzeczywiste generujące kolejne kroki obliczeń maszyny Turinga (twierdzenia 3.6, 3.7, 3.8), co pozwala na przeniesienie rozważań dotyczących maszyn Turinga, w szczególności problemów nierozstrzygalnych dla maszyn Turinga, na pole rekurencyjnych funkcji rzeczywistych. Następnie rozszerzone zostały wyniki pracy [33] z przypadku deterministycznej maszyny Turinga, na przypadek niedeterministycznej maszyny Turinga (twierdzenie 3.11). Wynik ten dotyczy symulacji niedeterministycznej maszyny Turinga, za pomocą funkcji analitycznej zmiennej rzeczywistej. Biorąc pod uwagę, iż większość fizycznych systemów dynamicznych jest analityczna, wynik ten daje możliwość znalezienia urządzenia, bazującego na pewnym fizycznym systemie, zdolnego do takiej symulacji. Zaprezentowany został również wynik porównujący możliwości obliczeniowe maszyny Turinga i modelu EAC, paragraf 4.1. Dzięki przeprowadzonej w paragrafie 4.1 symulacji uzyskano twierdzącą odpowiedź na pytanie postawione przez Rubla w pracy [67], dotyczące kwestii zdolności EACa do generowania wyników dowolnej maszyny Turinga (twierdzenie 4.3). Ostatni badany w tej pracy aspekt porównujący obliczenia klasyczne i analogowe dotyczy analogicznego zagadnienia jak dyskutowane wcześniej, z tym że teraz porównane zostały ze sobą funkcje rekurencyjne zdefiniowane na liczbach naturalnych i model EAC. Dokładniej w paragrafie 4.2 przedstawiono wyniki pokazujące, iż EAC zdolny jest do generowania zbiorów rekurencyjnie przeliczalnych (twierdzenie 4.5), jak również do generowania wyniku dowolnej funkcji rekurencyjnej w zadanym punkcie (twierdzenie 4.6). Dodatkowo wyniki te prowadzą do ważnego wniosku, że klasyczny problem stopu możemy sprowadzić do zagadnienia pustości zbioru w modelu EAC. Wniosek ten pokazał nową drogę próby pozytywnego rozwiązania problemu stopu dla funkcji częściowo rekurencyjnych. W pracy podane zostały również przykłady konstrukcji zbiorów rekurencyjnie przeliczalnych w modelu EAC, paragraf 4.2.1.

Krótkie omówienie treści poszczególnych rozdziałów pracy. W pierwszym, wstępnym rozdziale niniejszej pracy przybliżona została, nawiązując do prac V. Busha, C. Shanona [7, 68, 50], historia obliczeń analogowych, jej główne idee oraz kierunki obecnych badań prowadzonych w tej dziedzinie [14, 8, 9, 21]. W rozdziale drugim zbudowana została teoretyczna baza do dyskusji o procesach obliczeniowych. W pierwszej cześć drugiego rozdziału przedstawiono podstawy teorii obliczeń dyskretnych na podstawie klasycznych monografii [52, 71]. Szczególna uwaga została zwrócona na podstawowe własności maszyny Turinga oraz klasy funkcji częściowo rekurencyjnych. W drugiej części tego rozdziału zaprezentowano istotne dla rozprawy pojęcia teorii obliczeń analogowych. Zaczęto od przybliżenia podstawowego modelu obliczeń analogowych, modelu GPAC, a następnie jego uogólnienia, jakim jest szerszy model, EAC. Na koniec zaprezentowano rekurencyjne funkcje rzeczywiste. W rozdziale trzecim zaprezentowano wybrane związki pomiędzy obliczeniami na liczbach rzeczywistych, a maszyną Turinga. Jako pierwsze, przytoczone zostały pewne klasyczne wyniki dotyczące obliczalności przy użyciu maszyn Turinga poza zbiorem liczb naturalnych. Część ta powstała w oparciu o prace [4, 10]. Następnie rozważony został problem odwrotny - jak funkcje zdefiniowane na liczbach rzeczywistych mogą generować wyniki obliczeń maszyny Turinga. W pierwszej kolejności rozważono rzeczywiste funkcje rekurencyje. Przytoczone zostały wyniki z pracy [45] oraz własne wyniki z pracy [57], które wskazują na to, iż rezultat obliczeń dowolnej deterministycznej maszyny Turinga możemy otrzymać przy pomocy rzeczywistej funkcji rekurencyjnej. Jako ostatni fragment tego rozdziału, przedstawione zostały wyniki z pracy [33] oraz własne wyniki z pracy [60] mówiące, że wynik obliczeń dowolnej deterministycznej jak i niedeterministycznej maszyny Turinga możemy otrzymać za pomocą pewnej analitycznej funkcji zmiennej rzeczywistej. Wynik ten jest interesujący, głównie z tego powodu, iż większość fizycznych systemów dynamicznych jest analityczna, przynajmniej w idealnych warunkach. W rozdziale czwartym zaprezentowano główne, własne wyniki dotyczące modelu EAC i jego możliwości symulacji maszyn Turinga oraz generowania funkcji częściowo rekurencyjnych i zbiorów rekurencyjnych. Wyniki zaczerpnięte zostały z własnych prac [59, 61]. Są one dowodem na to, iż EAC generuje dowolne rekurencyjnie przeliczalne zbiory liczb naturalnych, co za tym idzie możemy w tym modelu również otrzymać wartość dowolnej funkcji częściowo rekurencyjnej w zadanym punkcie n. Ponadto, zaprezentowany został ważny wynik przenoszący klasyczny problem stopu, na pole obliczeń w modelu

EAC. Praca zaopatrzona została także w zwykłe uzupełnienia, jakimi są rysunki, tabele, przypisy, oznaczenia i bibliografia, pozwalające na pogłębioną analizę jej treści.

Rozdział 1

Wprowadzenie

Wśród wszystkich maszyn liczących możemy rozróżnić dwie rodziny. Pierwsza z nich pochodzi od klasycznego liczydła. Urządzenia te używają cyfr do reprezentowania liczb i są nazywane komputerami cyfrowymi. Druga pochodzi od graficznych rozwiązań problemów uzyskanych przez starożytnych mierniczych, którzy odnaleźli analogie pomiędzy własnościami problemów matematycznych, a ciągłymi liniami rysowanymi na papierze. Maszyny wykorzystujące do obliczeń podobne własności nazywamy maszynami analogowymi. Termin ten pochodzi z greckiego „analogikos", co oznacza: „przez proporcję". We wcześniejszych latach używano wielu urządzeń analogowych, takich jak nomogram, planimetr czy suwak logarytmiczny. Urządzenia te zwykle realizowały tylko jedną funkcję. Jeśli pewne urządzenie tego typu mogłoby być „programowane" do wykonania więcej niż jednej funkcji w różnym czasie, wtedy można mówić o komputerze analogowym. W przypadku urządzeń analogowych proces obliczeń jest zastąpiony przez mierzenie i manipulacje pewną ciągłą własnością fizyczną, taką jak mechaniczne przesunięcia lub napięcie, stąd są one nazywane również komputerami ciągłymi.

Poniżej przedstawiamy krótko rozwój teorii komputerów analogowych oraz główne idee obliczeń analogowych, obecne badania prowadzone w tym kierunku oraz perspektywy przyszłego rozwoju badań nad obliczeniami w dziedzinie rzeczywistej.

1.1 Historia obliczeń analogowych

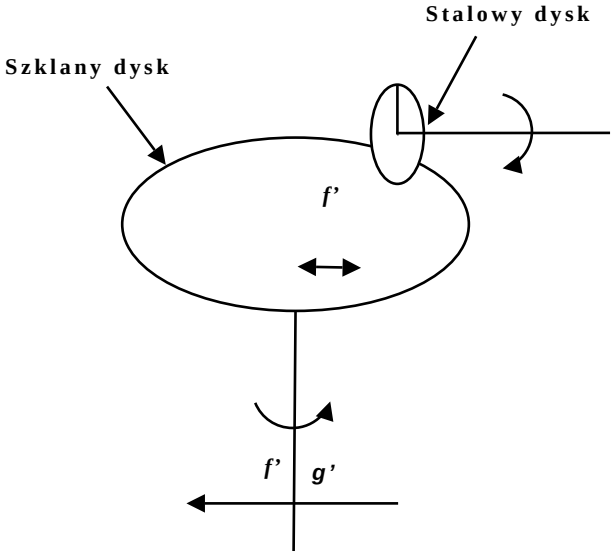
Przedstawimy teraz zarys historyczny rozwoju komputerów analogowych. Zaczniemy od czasów pierwszych nieprogramowalnych urządzeń liczących po czasy obecne. Od najdawniejszych lat ludzie próbowali ułatwiać sobie życie wymyślając różnego rodzaju urządzenia o samoczynnym działaniu ([30, 54]). Na przestrzeni wieków powstawało wiele różnego rodzaju mechanizmów wykonujących cały cykl swojej pracy bez udziału człowieka, głównie w Grecji (automatyczna lampa olejowa otwierająca drzwi w ateńskim Erachtejonie konstrukcji Herona (I wiek p.n.e.), opisane m. in. w książce J. Herlingera [28], automatyczne teatrzyki). W starożytnych Chinach skonstruowano wóz wskazujący południe, którego działanie, opierające się o zasadę na jakiej działa stosowany w samochodach mechanizm różnicowy, było zagadką do lat sześćdziesiątych XX wieku. Rekonstrukcja wozu wskazującego południe znajduje się w Muzeum Nauki w Londynie. Wzmianki o wozach wskazujących południe powtarzają się w czasach historycznych od XIII wieku p.n.e. do XII wieku n.e. Warto też wspomnieć o zegarze wodnym z 1090 roku, zbudowanym przez Chińczyka Su Sunga. Najdawniejszym takim samoczynnym urządzeniem o znanej zasadzie działania jest, zbudowany w III wieku p.n.e. przez Ktesibiosa z Aleksandrii, mechanizm zastosowany do regulacji przepływu wody w bardzo dokładnym zegarze wodnym. Jego konstruktor miał dodatkową trudność do pokonania, gdyż w jego czasach doba liczyła 24 godziny, ale dzień, liczony od wschodu do zachodu słońca, był dzielony na 12 godzin podobnie jak pozostała część doby - noc. Stąd godziny dnia i godziny nocy miały różny czasookres w zależności od pory roku. Rekonstrukcja takiego zegara znajduje się obecnie w Muzeum Niemieckim w Monachium.

Na początku XVII wieku szkocki matematyk John Napier opisał logarytm oraz opracował tablice mnożenia. Następnie w 1632 roku angielski matematyk William Oughtred, posługując się logarytmem Napiera, skonstruował suwak logarytmiczny, czyli prosty przyrząd ułatwiający obliczenia, który pozostawał w powszechnym użyciu przez ponad trzysta lat, aż do końca lat 80-tych XX wieku. Suwak logarytmiczny, poza mnożeniem i dzieleniem, umożliwia wiele innych działań, np:. logarytmowanie, potęgowanie, pierwiastkowanie. Spełnia też rolę tablic trygonometrycznych.

Pierwszą maszynę analogową zaprojektował w 1872 roku W. Thomson (lord Kelvin), brytyjski fizyk pochodzenia irlandzkiego. Był to syntezator harmoniczny. Około 4 lata później zaprojektował on kilka rodzajów maszyn

analogowych. Jego brat, J. Thomson, w tym samym czasie wynalazł integrator kulowy. Maszyny te działały na zasadzie wykorzystania analogii między opisem matematycznym rozwiązywanego problemu a wzorami opisującymi procesy zachodzące w maszynie.

W 1927 roku w Massachusetts Institute of Technology amerykański naukowiec Vannevar Bush, razem z dwoma współpracownikami, skonstruował pierwszy komputer analogowy, Product Intergraph, a następnie w 1931 roku zbudował on pierwszy mechaniczny analizator różniczkowy (Differential Analyzer), opisany w pracy [7]. Urządzenie to w założeniach przypominało automatyczny suwak logarytmiczny i opierało się o zasady podane przez lorda Kelvina w roku 1876 (praca [31]). Analizator różniczkowy był wykorzystywany do rozwiązywania prostych równań różniczkowych, na przykład postaci: $d^2y/dx^2 = -y$. Rozwiązuje on równania różniczkowe poprzez całkowanie. Typowy mechanizm analizatora różniczkowego składa się z dwu lub więcej zespołów integratora (zawierających jeden wzmacniacz momentu obrotowego w zespole), rysunek 1.1. Integrator jest w istocie mechanizmem zmian prędkości. Przyjmuje on formę dysku obracającego się poziomo, na którym opiera się okrągły stalowy dysk. Pionowa oś, poziomego dysku, jest tak ustawiana podczas ruchu wózka, na którym znajduje się ten dysk, aby odległość punktu styczności stalowego koła z centrum dysku zmieniała się.



Rysunek 1.1: Integrator $\int f'dg'$

Obrót poziomego dysku oraz ruch wózka stanowią dwa wejścia integratora. Jeśli integrator działa, dysk obraca się i jednocześnie dokonuje się jego przesunięcie równoległe. Dzięki tarciu napędy dyskowe obracają stalowe koło. Obrót koła reprezentuje wyjście integratora. Analizator różniczkowy Busha był olbrzymim osiągnięciem. Jego sukces spowodował duże zainteresowanie technikami obliczeń analogowych, a tym samym odciągnął uwagę naukowców na długi okres od badań nad rozwojem techniki cyfrowej. Wiele podobnych modeli analizatorów różniczkowych było budowanych pomiędzy rokiem 1934 a wczesnymi latami 50-tymi, aż w końcu zostały zastąpione przez komputery cyfrowe. Dziś można oglądać dwa takie zachowane urządzenia z tamtego okresu. Pierwsze z nich to część oryginalnego analizatora różniczkowego, zbudowanego przez Hartree'a i Portera w Manchesterze, w 1934 roku, aktualnie pokazywana w Muzeum Nauki w Londynie. Drugie znajduje się w Muzeum Transportu i Technologii (MOTAT) w Nowej Zelandii, Auckland i składa się z całego analizatora różniczkowego zbudowanego w 1935 przez J. B. Bratta z Uniwersytetu w Cambridge. W 1942 roku V. Bush zbudował też elektroniczny, programowalny analizator różniczkowy. W połowie lat czterdziestych, w miejsce mechanicznych, pojawiły się konstrukcje elektryczne i elektroniczne. Seryjna produkcja takich maszyn została podjęta w USA w 1949 roku i we Francji 1952 roku. W Polsce, pierwszą maszyną tego typu, był analizator równań różniczkowych (ARR) zbudowany w Zakładzie Aparatów Matematycznych PAN w 1954 roku. Później, prace nad podobnymi urządzeniami prowadzono również w Instytucie Podstawowych Problemów Techniki PAN. We wczesnych latach 40-tych dwudziestego wieku analizatory różniczkowe były wykorzystywane m. in. w Manchesterze, Filadelfii, Bostonie i Oslo, np. do rozwiązywania problemów teorii jądra atomu, w astrofizyce i balistyce. Po pojawieniu się komputerów cyfrowych w latach 50-tych i 60-tych, zaprzestano ich używania.

Kolejny, uznawany później za podstawowy, model komputera analogowego to wprowadzony przez Claude'a Shannona w 1941 roku, matematyczny model analizatora o nazwie General Purpose Analog Computer (GPAC) [68], dla którego fundamentalnymi realizowanymi operacjami są dodawanie, mnożenie i całkowanie. Wyniki w tym modelu są generowane na podstawie wartości wejściowych za pomocą zależności zdefiniowanej przez pewien graf skierowany, gdzie każdy węzeł jest jednym z następujących elementów:

– integrator - zespół dwóch wejść, jednego wyjścia, z ustawionymi warunkami początkowymi, jeżeli wejścia są jednoargumentowymi funkcjami

f, g, wówczas wyjście jest całką Riemanna-Stieltjesa;

– stały mnożnik - zespół jednego wejścia i jednego wyjścia, skojarzony z liczbą rzeczywistą k, jeżeli f jest na wejściu, wówczas na wyjściu dostajemy kf;

– sumator - zespół dwóch wejść i jednego wyjścia, jeżeli f i g są na wejściu wówczas na wyjściu dostajemy $f + g$;

– multiplikator - zespół dwóch wejść i jednego wyjścia, jeżeli f i g są na wejściu, wówczas na wyjściu dostajemy fg;

– stała - zespół bez wejść, z jednym wyjściem, na wyjściu zawsze dostajemy wartość 1.

Chociaż powyższe pojęcie GPACa wydaje się dość intuicyjne i naturalne, powszechnie akceptowaną definicję GPACa przedstawiła dopiero M. Pour-El w pracy [62]. Z uwagami, wprowadzonymi przez L. Lipshitza i L. A. Rubla [37], pokazała, że GPAC generuje funkcje algebraicznie różniczkowalne, czyli jednoznaczne rozwiązania wielomianowych równań różniczkowych z dowolnymi współczynnikami wymiernymi. Do tego zbioru funkcji należą proste funkcje takie jak: funkcja wykładnicza, funkcje trygonometryczne, jak również ich sumy, iloczyny i złożenia oraz rozwiązania równań różniczkowych budowanych na ich podstawie. W świetle tej definicji, GPAC potrafi również rozwiązywać proste zagadnienia brzegowe, natomiast nie potrafił rozwiązać, np. problemu Dirichleta [67]. Po pojawieniu się cyfrowych komputerów, hybrydowe komputery cyfrowo-analogowe rozwiązywały ponadto cząstkowe równania różniczkowe.

Badania dotyczące logiki Łukasiewicza wartościowanej w sposób ciągły spowodowały wprowadzenie jej do elektronicznych realizacji rozszerzonego modelu komputera analogowego (prace [41, 42, 44]). Logika Łukasiewicza ma przeliczalnie wiele wartości logicznych, dzięki czemu opisuje klasę idealnych układów analogowych, które mają nieskończoną maksymalną precyzję. Prawdziwe układy analogowe, które mają określoną maksymalną precyzję, są sklasyfikowane przez liczbę danych poszczególnych elementów obwodu. Algebra Boole'a jest jednym z modeli logik Łukasiewicza, która opisuje binarne układy cyfrowe. Przetwarzane, przez opisywane logiką Łukasiewicza, analogowe obwody, elementy to implikacje albo zaprzeczenia implikacji. Implikacja

Łukasiewicza, $b \Rightarrow a$, jest zdefiniowana przez funkcję $\min(1, 1 - a + b)$. Zaprzeczenie implikacji, $\neg(b \Rightarrow a)$, jest definiowane przez funkcję $\max(0, a - b)$. Układy logiczne Łukasiewicza przybliżają, w sposób wystarczająco dobry, obliczenia GPACa. Postawiono hipotezę, że klasa zwykłych algebraicznych równań różniczkowych, równań rozwiązywanych przez GPAC, jest aproksymowana dowolnie dokładnie przez zdania w logice Łukasiewicza (praca [41]).

1.2 Obecne badania

Po długiej przerwie spowodowanej rozwojem technologii cyfrowej, na przełomie lat osiemdziesiątych i dziewięćdziesiątych dwudziestego wieku ponownie pojawiło się zainteresowanie technologią analogową jako alternatywnym podejściem do realizacji obliczeń. Wówczas to pojawiły się dwie odrębne gałęzie rozwoju analogowych systemów liczących. Pierwsze z nich to modele, w których operacje wykonywane są na liczbach rzeczywistych, ale w czasie dyskretnym. Przykładem takich modeli są, np.: maszyny BSS [3] oraz Analog Recurrent Neural Network (ARNN) [69]. Mówiąc ogólnie, maszyna BSS przypomina w konstrukcji maszynę Turinga (paragraf 2.1.3) z tą istotną różnicą, że maszyna ta może wykonywać obliczenia w arytmetyce rzeczywistej. Model sieci Analog Recurrent Neural Network (ARNN) bazuje natomiast na pewnej metamorfozie biologicznej i umożliwia opis szerokiej klasy języków formalnych. Sieci składają się z wielorakich procesorów połączonych w skomplikowaną strukturę. Każdy bazowy procesor („neuron"), oblicza skalarną nieliniową funkcję wyjścia. Skalarna wartość obliczana przez neuron oddziałuje na inne neurony. Sieć jest analogowa, dlatego że jest zdefiniowana na ciągłej dziedzinie w przeciwieństwie do dyskretnych modeli komputerów cyfrowych. Tego typu analogowe modele obliczeń mają zastosowanie w analizie numerycznej [55].

Druga gałąź współczesnych nam obliczeń analogowych, to modele działające zarówno w ciągłej dziedzinie, jak i w ciągłym czasie. Tutaj pojawił się kolejny model komputera analogowego. L. Rubel zaproponował w 1993 roku teoretyczny model, Extended Analog Computer (EAC) [67], który w założeniach miał rozszerzać możliwości GPACa. Nowy model potrafi rozwiązywać problem brzegowych wartości dla cząstkowych równań różniczkowych. Rubel zaznaczył, że EAC jest to konstrukcja teoretyczna, której możliwość realizacji przez fizyczne urządzenia jest dotąd nieznana. Nie wiadomo również czy w modelu tym możemy otrzymać każdą funkcję analityczną, co dyskwalifi-

kowałoby go jako nazbyt obszerny.

Do dzisiejszego dnia trwają na uniwersytecie w Indianie prace nad zbudowaniem maszyny odpowiadającej modelowi EAC. Praca [43] z 2008 roku autorstwa J. Millsa (ucznia Rubla), opisuje pewien rzeczywisty model EACa, który udało się zbudować i z sukcesem wykorzystać w wielu dziedzinach, m. in. ekonomi, medycynie, balistyce. Jednak model ten nie realizuje wielu z założeń oryginalnego EACa Rubla. Nie wiadomo nawet czy taka stuprocentowa realizacja modelu Rubla jest możliwa.

Szeroką gałąź obecnych badań teoretycznych nad obliczeniami analogowymi zapoczątkowała praca C. Moore'a z 1996 roku [47]. W pracy tej Moore zaproponował definicję funkcji $\mathbb{R}$-rekurencyjnych, czyli funkcji określonych na liczbach rzeczywistych analogicznie do klasy funkcji rekurencyjnych w sensie Kleene'go (paragraf 2.1.1).

Definicja 1.1 [47] *Funkcję $f : \mathbb{R}^m \to \mathbb{R}^n$ nazywamy $\mathbb{R}$-rekurencyjną jeżeli możemy ją wygenerować ze stałych 0 i 1 za pomocą następujących operacji: jeżeli f i g są funkcjami $\mathbb{R}$-rekurencyjnymi, wówczas mamy*

- *składanie*: $h(\bar{x}) = f(g(\bar{x}))$;
- *rekursja różniczkowa lub całkowanie*:

  $$h(\bar{x}0) = f(\bar{x}), \quad \partial_y h(\bar{x}, y) = g(\bar{x}, y, h(\bar{x}, y)).$$

  Innymi słowy, niech $h = f$ dla $y = 0$ oraz niech pochodna h względem y zależy od $h(\bar{x}, y), y$ i $\bar{x}$. Wówczas piszemy

  $$h(\bar{x}, y) = f(\bar{x}) + \int_0^y g(\bar{x}, y', h(\bar{x}, y'))dy'$$

  lub w skrócie $h = f + \int g$;
- *μ-rekursja*: $h(\bar{x}) = \mu_y f(\bar{x}, y) = \inf\{y : f(\bar{x}, y) = 0\}$, *gdzie infimum zwraca y o najmniejszej wartości bezwzględnej oraz (z definicji) ujemną wartość, gdy istnieją dwa y o tej samej najmniejszej wartości bezwzględnej*;
- *wektor funkcji może być zdefiniowany poprzez zdefiniowanie jego składowych.*

Definicja wprowadzona przez Moora była interesująca, ponieważ wprowadzała klasę funkcji, w której można było wyodrębnić podklasy odpowiadające klasom z klasycznej teorii obliczeń, takie jak elementarne funkcje rekurencyjne (praca [10]). Moore, używając wprowadzonej przez siebie definicji, podał szereg własności klasy funkcji $\mathbb{R}$-rekurencyjnych. Niestety pojawiły się pewne problemy wynikające z przyjęcia takiej definicji, np. przy dowodzie tak ważnej własności jak istnienie analogowego rozwiązania problemu stopu dla funkcji częściowo rekurencyjnych. Główną przyczyną problemów z definicją Moore'a była pojawiająca się w niej operacja minimalizacji. Operacja ta wydaje się fizycznie nierealizowalna. Jest ona zaczerpnięta z teorii rekursji, lecz nie jest odpowiednia dla analitycznej rzeczywistości analogowych obliczeń. W związku z tym, w pracy [17] zaproponowana została przez J. F. Costę oraz J. Myckę, nowa definicja, bazująca na oryginalnej definicji Moore'a, definicja rekurencyjnych funkcji rzeczywistych (paragraf 2.4). Wprowadzono ją aby uniknąć problemów wynikających z pierwotnej definicji i aby ułatwić użycie narzędzi analizy matematycznej, zastępując operację minimalizacji, operacją granicy. Operacja granicy pozwala również wyodrębnić podklasy rekurencyjnych funkcji rzeczywistych odpowiadające poszczególnym poziomom hierarchii arytmetycznej (praca [17]).

Na tym polu różni autorzy prowadzą badania w bardzo wielu kierunkach. J. F. Costa, B. Loff, J. Mycka w swoich badaniach zaprezentowanych w pracy [15] szukają powiązań między teorią obliczalności (złożoności) a analizą matematyczną. Stawiają oni rekurencyjne funkcje rzeczywiste jako możliwą gałąź opisowej teorii mnogości. Zastanawiają się w jakim stopniu rekurencyjne funkcje rzeczywiste i opisowa teoria mnogości pokrywają się. Badają również związki między rekurencyjnymi funkcjami rzeczywistymi i analizą obliczeniową dowodząc, że można zawrzeć całą arytmetyczną hierarchię w hierarchii granic (η-hierarchia) ograniczając się do jej pewnego poziomu, gdzie zaczyna się hierarchia analityczna (patrz twierdzenie 2.3 z prac [15, 16]). Zastanawiają się również nad powiązaniami między analizą matematyczną a teorią obliczalności (złożoności). Ich celem jest przełożenia problemów występujących w teorii klasycznej obliczalności na pole analizy matematycznej z nadzieją znalezienia sposobu na rozwiązanie tych nierozstrzygalnych dotychczas, jak $P \neq NP$ (praca [20]). W pracy [15] dyskutowana jest również granica definiowalności w analizie, czyli np. czy można przy pomocy operacji granicy zdefiniować więcej funkcji niż definiując nowe funkcje przy pomocy rozwiązań równań różniczkowych I stopnia (rozwiązanie równania różniczkowego I stop-

nia możemy zawsze wyrazić przy pomocy pewnej granicy ale nie na odwrót). Drugi, rozważany aspekt definiowalności funkcji w analizie to czy operacja granicy zawsze dostarcza nam nowych funkcji, czy też może wystarczy pewna skończona liczba granic aby zdefiniować wszystkie możliwe funkcje. Autorzy pokazują, że nie ma indukcyjnej granicy definiowalności funkcji zmiennej rzeczywistej, gdy nowe funkcje definiujemy poprzez operacje składania, rozwiązania równania różniczkowego stopnia I oraz granice w nieskończoności (twierdzenie 3.1 z pracy [15]).

Jak widzimy obecnie w literaturze pojawia się kilka modeli obliczeń na liczbach rzeczywistych: GPAC, EAC, rekurencyjne funkcje rzeczywiste. Związki pomiędzy tymi modelami nie są zbyt dobrze znane, jak to ma miejsce w przypadku klasycznych modeli obliczeń. Tutaj mamy również kilka modeli, jak maszyna Turinga, funkcje rekurencyjne, λ-rachunek Churcha, itd. Wiemy o nich, iż są to modele równoważne pod względem klasy rozwiązywanych problemów. Co do modeli obliczeń w dziedzinie rzeczywistej, wiemy tylko o kilku zależnościach:

- problemy rozwiązywane w modelu GPAC znajdują rozwiązanie w modelu EAC [26];
- istnieje podklasa rzeczywistych funkcji rekurencyjnych $[\mathbb{R}, +_b, \times; SK, R]$, która jest równoważna z klasą funkcji obliczalnych przez FF-GPAC [47];
- funkcje generowane przez EAC na poziomie 0 są to rzeczywiste funkcje rekurencyjne klasy H_0 [58].

1.3 Motywacja

Teoria obliczeń analogowych, po około pięćdziesięcioletniej przerwie spowodowanej rozwojem technik cyfrowych, cieszy się ponownym zainteresowaniem. Możemy wymienić kilka przyczyn ponownego zainteresowania teorią obliczeń analogowych. Częściowo spowodowane jest to szeroko zakrojonym programem poszukiwań alternatywnych podejść do problemu obliczalności (na przykład takich jak sieci neuronowe [70]), chęcią udoskonalenia algorytmów numerycznych oraz użycia tej teorii do dokładniejszego sklasyfikowania możliwości obliczeniowych różnych ciągłych systemów dynamicznych, w celu wykorzystania ich do budowy wyspecjalizowanych urządzeń, które mogłyby

rozwiązywać pewne trudne z punktu widzenia obliczeń dyskretnych problemy w sposób ekonomiczniejszy pod względem kosztów bądź czasu. Przykłady tego typu rozwiązań, które znalazły miejsce w praktycznych zastosowaniach w takich dziedzinach jak ekonomia, medycyna, wojsko, możemy znaleźć w pracy [43].

Kolejnym powodem dla którego autorzy obecnie zajmują się problemem obliczeń analogowych są zagadnienia teoretyczne, które muszą być rozwiązane zanim pojawią się praktyczne aplikacje. Ze względu na swoją krótką historię, w obrębie zainteresowań badań naukowych, jest to bogaty obszar dla teoretycznych rozważań, które będą również przedmiotem naszej pracy. W pracy zestawiamy ze sobą dwie teorie: klasyczną teorię obliczeń oraz teorię obliczeń analogowych. Ma to na celu, między innymi, przeniesienie ważnych nierozstrzygalnych kwestii z klasycznej teorii obliczeń na grunt obliczeń analogowych, aby tam poszukać dla nich rozwiązania (rozdziały 3 i 4). Jak również wskazanie dróg alternatywy dla problemów, dla których istnieją już rozwiązań bazujących na obliczeniach dyskretnych. Rozwiązań, które byłyby efektywniejsze, np. przy problemach polegających na szybkim przetwarzaniu dużych ilości danych, tam gdzie rozwiązania dyskretne są szczególnie czasochłonne.

Do podjęcia badań nad obliczeniami analogowymi motywują słowa Richarda Karpa [2]:

The classical theory of computation does not deal adequately with computations that operate on real-valued data. Most computational problems in the physical sciences and engineering are of this type. [1]

[1]Klasyczna teoria obliczeń nie zajmuje się właściwie obliczeniami przetwarzającymi rzeczywiste wartości. Wiele obliczeniowych problemów w fizyce i inżynierii jest właśnie tego typu.

Rozdział 2

Podstawowe pojęcia teorii obliczalności dyskretnej i analogowej

Zaczniemy od wprowadzenia, potrzebnych nam w dalszej części pracy, pojęć dotyczących obliczeń na liczbach naturalnych oraz obliczeń na liczbach rzeczywistych.

Pierwsza część to krótki przegląd prezentowanych w literaturze klasycznych modeli obliczeń. Przedstawia on, w oparciu o monografie [52, 53], pojęcie zbióru funkcji rekurencyjnych w ujęciu Kleene'ego. Kolejno zaprezentowane zostały podstawowe definicje, wyróżniane podklasy, hierarchia oraz kilka interesujących własności tego zbioru funkcji. Następnie przedstawiony został model maszyny Turinga.

Kolejne części poświęcone zostały obliczeniom na liczbach rzeczywistych. W drugiej zaprezentowano podstawowy model operujący na liczbach rzeczywistych, wprowadzony przez Shannona w 1941 roku, nazywany General Purpose Analog Computer (GPAC) [68]. Podana została definicja oraz podstawowe własności tego modelu. Trzecia część dotyczy matematycznego modelu Extended Analog Computer (EAC), wprowadzonego przez Rubla w 1993 roku [67] z intencją poszerzenia możliwości obliczeniowych GPACa. Przedstawiona została koncepcja działania tego modelu oraz własna definicja, będąca matematycznym opisem pomysłu Rubla z pracy [67]. W ostatniej, czwartej części podalismy definicję oraz podstawowe własności matematycznego modelu rekurencyjnych funkcji rzeczywistych wprowadzonego przez Moore'a w 1996 roku, w pracy [47], jako uogólnienie modelu klasycznych funkcji re-

kurencyjnych na zbiór liczb rzeczywistych.

2.1 Klasyczne modele obliczeń

Do tej pory wprowadzono wiele dobrze znanych modeli efektywnych obliczeń takich jak funkcje rekurencyjne (1931) (paragraf 2.1.1), maszyny Turinga (1936)(paragraf 2.1.3), λ-rachunek Churcha (1936) [11], algorytmy normalne Markowa (1954) [38], maszyny o swobodnym dostępie Shepherdsona-Sturgisa (1963), maszyny URM (Unlimited Register Machine), itp. Modele te są wzajemnie równoważne ze względu na swoje możliwości obliczeniowe. Zwięzły ich przegląd oraz dowody ich równoważności znaleźć możemy w książce [75]. Źródłem podanych w niniejszym paragrafie definicji i twierdzeń jest monografia [52].

2.1.1 Funkcje rekurencyjne

Pojęcie funkcji częściowo rekurencyjnych odgrywa znaczącą rolę w klasycznej teorii obliczeń [52]. Są to funkcje przekształcające zbiór liczb naturalnych $\mathbb{N}$ w zbiór liczb naturalnych $\mathbb{N}$, o których możemy myśleć jako o odpowiedniku klasy funkcji obliczalnych w intuicyjnym sensie. Teoria obliczeń pokazuje, że funkcje częściowo rekurencyjne są to dokładnie te funkcje, które mogą być obliczone na maszynie Turinga, której definicję prezentujemy w paragrafie 2.1.3.

Algebra funkcji jest zbiorem funkcji określonych przez indukcyjne domknięcie dla pewnych operatorów i pewnego zbioru funkcji. Takie podejście jest często używane w teorii obliczeń, a ostatnio przy charakteryzowaniu klas złożoności.

Definicja 2.1 [63, 12] *Niech $\mathfrak{F}$ będzie klasą funkcji, $\mathcal{F} \subseteq \mathfrak{F}$ będzie podzbiorem funkcji oraz $\mathcal{O} \subseteq \bigcup_{k\in\mathbb{N}}\{O : \mathfrak{F}^k \to \mathfrak{F}\}$ zbiorem operatorów. Indukcyjnym domknięciem $\mathcal{F}$ dla $\mathcal{O}$, $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$, nazwiemy najmniejszy zbiór zawierający $\mathcal{F}$, taki że jeżeli $f_1, \ldots, f_k \in \mathcal{A}$ należą do dziedziny k-wymiarowego operatora $O \in \mathcal{O}$, wówczas $O(f_1, \ldots, f_k) \in \mathcal{A}$. Razem z $\mathcal{O}$ indukcyjne domknięcie $[\mathcal{F}; \mathcal{O}]$ nazywamy algebrą funkcji.*

Dla uproszczenia będziemy zapisywać $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$ w obydwu przypadkach: dla określenia indukcyjnego domknięcia $[\mathcal{F}; \mathcal{O}]$ oraz algebry funkcji

$([\mathcal{F};\mathcal{O}],\mathcal{O})$. W literaturze nie zawsze te pojęcia są rozróżniane [12]. Jednakże dobrze jest w niektórych sytuacjach wprowadzić rozróżnienie, gdyż np. pewne pojęcia (jak zawieranie się funkcji) odnoszą się do indukcyjnego domknięcia, jako zbioru, a inne (jak dziedzina funkcji) do algebry funkcji z jej specyficzną strukturą. Powyższa definicja jest dobrze określona co wynika z twierdzenia Kleene'go oraz twierdzenia Knastera-Tarskiego o punkcje stałym, szczegóły w książce [12].

Niech $\mathcal{F}$ będzie klasą funkcji, f i g funkcjami oraz $O_1,\ldots,O_n$ operatorami. Będziemy również stosować nierestrykcyjny zapis dla notacji w nawiasach kwadratowych:

$$[f,g,\mathcal{F};O_1,\ldots,O_n]=[f,g\cup\mathcal{F};O_1,\ldots,O_n].$$

Przygotujemy się teraz do przedstawienia definicji klasy funkcji częściowo rekurencyjnych. W klasie funkcji częściowo rekurencyjnych wyróżniamy następujące funkcje bazowe:

- funkcja zerowania, $\mathfrak{z}:\mathbb{N}\to\mathbb{N}$, zdefiniowana następująco: $\mathfrak{z}(n)=0$;
- funkcja następnika, $\mathfrak{s}:\mathbb{N}\to\mathbb{N}$, zdefiniowana następująco: $\mathfrak{s}(n)=n+1$;
- projekcja, dla każdego k, $\mathfrak{i}^k_1,\ldots,\mathfrak{i}^k_k$, gdzie $\mathfrak{i}^k_i:\mathbb{N}^k\to\mathbb{N}$, $i=1,\ldots,k$ jest zdefiniowana następująco: $\mathfrak{i}^k_i(n_1,\ldots,n_k)=n_i$;

Przyjmiemy oznaczenie $\mathfrak{i}=\{\mathfrak{i}^k_i:k,i\in\mathbb{N},1\leqslant i\leqslant k\}$.

W tym paragrafie rozważamy funkcje zdefiniowane na zbiorze liczb naturalnych $\mathbb{N}$. Również takie, które są zdefiniowane tylko dla pewnych liczb naturalnych a dla pozostałych nie mają określonej wartości. Będziemy stosować następujące oznaczenia:

- $f(n)\downarrow$ dla funkcji, której wartość w punkcie n jest określona;
- $f(n)\uparrow$ dla funkcji, której wartość w punkcie n jest nieokreślona.

Przejdziemy teraz do przedstawienia podstawowych operacji budujących klasę funkcji częściowo rekurencyjnych, są to:

- składanie (SK^l): niech $g_1,\ldots,g_l,l>0$ będą k-agumentowymi funkcjami oraz niech f będzie l-argumentową funkcją, wówczas operator składania zastosowany do tych funkcji generuje funkcję h k-argumentową, zdefiniowaną następująco:

$$h(\bar{n})=\mathrm{SK}^l(f,g_1,\ldots,g_l)(\bar{n})=f(g_1(\bar{n}),\ldots,g_l(\bar{n}));$$

– rekursja prosta (REK): niech f będzie k-argumentową funkcją oraz g $(k+2)$-argumentową funkcją, wówczas operator rekursji prostej zastosowany do tych funkcji generuje funkcję h $(k+1)$-argumentową, następująco:

$$h(\bar{n},0) = \mathrm{REK}(f,g)(\bar{n},0) = f(\bar{n}),$$

$$h(\bar{n},\mathfrak{s}(m)) = \mathrm{REK}(f,g)(\bar{n},\mathfrak{s}(m)) = g(\bar{n},m,h(\bar{n},m));$$

– minimalizacja (μ): niech f będzie $(k+1)$-argumentową funkcją, wówczas operator minimalizacji zastosowany do tej funkcji generuje k-argumentową funkcję h następująco:

$$h(\bar{n}) = \mu_m[(\forall j \leqslant m)(f(\bar{n},j)\downarrow) \wedge f(\bar{n},m) = 0],$$

gdzie $h(\bar{n})$ jest najmniejszym m, takim że $f(\bar{n},m) = 0$ oraz $h(\bar{n})$ jest niezdefiniowana jeżeli takie m nie istnieje.

Operacja składania jest zdefiniowana dla $h(\bar{n})$ jeżeli każda z wartości $g_1,\ldots,g_l$, f jest zdefiniowana w $\bar{n}$. Jeżeli co najmniej jedna z wartości $g_1,\ldots,g_l$, f jest niezdefiniowana w $\bar{n}$, wówczas $h(\bar{n})$ jest również niezdefiniowana w tym punkcie.
Operacja rekursji prostej definiuje $h(\bar{n},m)$, jeżeli $f(\bar{n})$ jest zdefiniowana oraz dla dowolnej liczby $i < m$, $h(\bar{n},i)$ i $g(\bar{n},i,h(\bar{n},i))$ są zdefiniowane. W przeciwnym razie wartość $h(\bar{n},m)$ jest niezdefiniowana dla danego m. Jeżeli funkcje $f(\bar{n})$ oraz $g(\bar{n},i,j)$ są wszędzie zdefiniowane, wówczas operacja rekursji prostej jest wszędzie zdefiniowana.
Operacja μ-rekursji jest zdefiniowana jeżeli $(\exists m)f(\bar{n},m) = 0$ oraz $(\forall j \leqslant m)f(\bar{n},j)$ jest zdefiniowana. Operacja μ-rekursji jest niezdefiniowana jeżeli $(\forall m)f(\bar{n},m) \neq 0$ lub $(\exists m)f(\bar{n},m) = 0$ lecz dla pewnego $j < m$, $f(\bar{n},j)$ jest niezdefiniowane. Jeżeli funkcja $f(\bar{n},m)$ jest regularna czyli wszędzie zdefiniowana oraz dla każdego $\bar{n}$ istnieje $m \in \mathbb{N}$, takie że $f(\bar{n},m) = 0$, wówczas operacja μ-rekursji jest również wszędzie zdefiniowana.

Zatem klasę funkcji częściowo rekurencyjnych, rozumianych jak w monografii [52], w terminach indukcyjnego domknięcia, definiujemy w następujący sposób:

Definicja 2.2 *Klasą funkcji częściowo rekurencyjnych* ($\mathcal{PRF}$) *zdefiniowanych na* $\mathbb{N}$ *nazwiemy* $\mathcal{PRF} = [\mathfrak{z},\mathfrak{s},\mathfrak{i};\mathrm{SK}^l,\mathrm{REK},\mu]$.

Jeżeli w definicji klasy funkcji częściowo rekurencyjnych dołożymy warunek regularności funkcji f przy operacji μ-rekursji, to otrzymamy klasę funkcji całkowicie rekurencyjnych (tj. zdefiniowanych i rekurencyjnych w każdym punkcie zbioru $\mathbb{N}$), dalej pisząc w skrócie funkcje rekurencyjne ($\mathcal{RF}$) będziemy mieli na myśli klasę funkcji całkowicie rekurencyjnych.

Przedstawimy teraz pojęcie zbiorów rekurencyjnych i rekurencyjnie przeliczalnych.

Definicja 2.3 [52] *Zbiór $A \subset \mathbb{N}^k$ nazywamy rekurencyjnym, jeżeli jego funkcja charakterystyczna jest rekurencyjna.*

Poprzez funkcję charakterystyczną zbioru rozumiemy funkcję postaci:

$$c_A(\bar{n}) = \begin{cases} 0 & \bar{n} \in A \\ 1 & \bar{n} \in \neg A \end{cases},$$

gdzie $\neg A$ oznacza dopełnienie zbioru A (czyli $\neg A \equiv \mathbb{N} \setminus A$).

Definicja 2.4 [52] *Zbiór $S \subset \mathbb{N}^k$ nazywamy rekurencyjnie przeliczalnym, jeżeli jest zbiorem pustym lub jest dziedziną funkcji częściowo rekurencyjnej.*

Definicja 2.5 [52] *Relację $R \subset \mathbb{N}^k$ nazywamy rekurencyjną, jeżeli jej funkcja charakterystyczna jest rekurencyjna.*

Poprzez funkcję charakterystyczną relacji rozumiemy funkcję postaci:

$$c_R(\bar{n}) = \begin{cases} 0 & R(\bar{n}) \\ 1 & \neg R(\bar{n}) \end{cases}.$$

Definicja 2.6 [52] *Relację $R \subset \mathbb{N}^k$ nazywamy rekurencyjnie przeliczalną, jeżeli istnieje funkcja $f : \mathbb{N}^k \to \mathbb{N}, f \in \mathcal{PRF}$, taka że:*

$$R(\bar{n}) \Leftrightarrow f(\bar{n}) \downarrow .$$

2.1.2 Podstawowe własności funkcji rekurencyjnych

W zbiorze funkcji częściowo rekurencyjnych wyróżniono kilka mniejszych klas funkcji [52, 53]. Jedną z nich jest już wspomniana w poprzednim paragrafie klasa funkcji całkowicie rekurencyjnych, w skrócie funkcji rekurencyjnych ($\mathcal{RF}$). A oto przykład kolejnej podklasy funkcji częściowo rekurencyjnych.

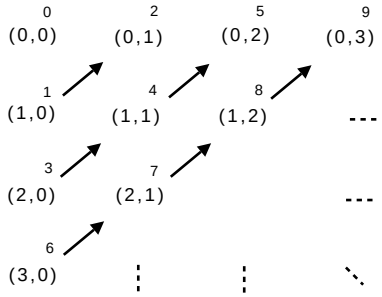
Definicja 2.7 *Klasą funkcji pierwotnie rekurencyjnych zdefiniowanych na* $\mathbb{N}$ *nazwiemy* $\mathcal{P} = [\mathfrak{z}, \mathfrak{s}, \mathfrak{i}; \mathrm{SK}^l, \mathrm{REK}]$.

Funkcje pierwotnie rekurencyjne w oczywisty sposób są funkcjami całkowitymi stąd mamy $\mathcal{P} \subseteq \mathcal{RF}$.

Przykład 2.1 Podstawowe funkcje takie jak $+, *, \dot{-}, |\cdot|, \uparrow, !, mod, div \in \mathcal{RF}$.

- dla $+(m,n) \equiv m+n$ mamy
  $+(m,0) = \mathfrak{i}^1_1(m)$, $+(m,\mathfrak{s}(n)) = \mathfrak{s}(\mathfrak{i}^3_3(m,n,+(m,n)))$;
- dla $*(m,n) \equiv m*n$ mamy
  $*(m,0) = \mathfrak{z}(m)$, $*(m,\mathfrak{s}(n)) = +(\mathfrak{i}^3_3(m,n,*(m,n)),m)$;
- dla $\dot{-}(m,n) = \begin{cases} m-n & m \geqslant n \\ 0 & m < n \end{cases}$ mamy
  $\dot{-}(m,0) = \mathfrak{i}^1_1(m)$, $\dot{-}(m,\mathfrak{s}(n)) = p(\mathcal{I}^3_3(m,n,\dot{-}(m,n)))$,
  gdzie $p(n) = \begin{cases} n-1 & n>0 \\ 0 & n=0 \end{cases}$ definiujemy następująco:
  $p(0)=0$, $p(\mathfrak{s}(n)) = \mathfrak{i}^2_2(n,p(n))$;
- dla $|m-n| = \begin{cases} m-n & m \geqslant n \\ n-m & m<n \end{cases}$ mamy
  $|m-n| = +(\dot{-}(m,n),\dot{-}(n,m))$;
- dla $\uparrow(m,n) \equiv m^n$ mamy
  $\uparrow(m,0) = \mathfrak{s}(\mathfrak{z}(m))$, $\uparrow(m,\mathfrak{s}(n)) = *(\mathfrak{i}^3_3(m,n,\uparrow(m,n)),m)$;
- dla $!(n) \equiv n!$ mamy
  $!(0)=1$, $!(\mathfrak{s}(n)) = *(\mathfrak{i}^2_2(n,!(n)),\mathfrak{s}(n))$;
- dla $mod(m,n) \equiv n \; mod \; m$ mamy
  $mod(m,0) = \mathfrak{z}(m)$,
  $mod(m,\mathfrak{s}(n)) = *(\mathfrak{i}^3_3(m,n,\mathfrak{s}(mod(m,n))), sg(|m-\mathfrak{s}(mod(m,n))|))$,
  gdzie $sg(n) = \begin{cases} 0 & n=0 \\ 1 & n>0 \end{cases}$ definiujemy następująco:
  $sg(0)=0$, $sg(\mathfrak{s}(n)) = \mathfrak{s}(\mathfrak{z}(\mathfrak{i}^2_2(n,sg(n)))) \equiv 1$;
- dla $div(m,n) \equiv n \; div \; m$ mamy
  $div(m,0) = \mathfrak{z}(m)$,
  $div(m,\mathfrak{s}(n)) = +(\mathfrak{i}^3_3(m,n,div(m,n)), \dot{-}(1,sg(|m-\mathfrak{s}(mod(m,n))|)))$;

Przykład 2.2 Funkcja Cantora kodowania par liczb, $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \overset{1-1}{\to} \mathbb{N}$ oraz funkcje dekodujące $\pi_1, \pi_2 : \mathbb{N} \to \mathbb{N}$, zwracające odpowiednio pierwszą i drugą składową pary należą do $\mathcal{P}$.



Rysunek 2.1: Przekątniowa numeracja par

Rysunek *2.1* przedstawia sposób numeracji par liczb wzdłuż przekątnych. Pary na przekątnych układają się następująco:

$$\underbrace{(0,0)}_{0}, \quad \underbrace{(1,0),(0,1)}_{1}, \quad \underbrace{(2,0),(1,1),(0,2)}_{2}, \quad \ldots$$

Mamy

$$\langle m, n \rangle = div(*(+(m,n), \mathfrak{s}(+(m,n))), 2) + n$$

oraz

$$\pi_2(p) = \dot{-}(p, \langle d(p), 0 \rangle), \;\; \pi_1(p) = \dot{-}(d(p), \pi_2(p)),$$

gdzie $p = \langle m, n \rangle$ oraz $d(p)$ przyjmuje wartość numeru przekątnej pary (m, n) i jest definiowana następująco:

$$d(0) = 0, \;\; d(\mathfrak{s}(p)) = +(\mathrm{i}_2^2(p, d(p)), \dot{-}(+(p,2), \langle \mathfrak{s}(d(p)), 0 \rangle)).$$

Kodowanie z powyższego przykładu możemy rozszerzyć na więcej liczb w następujący sposób: $\langle k, m, n \rangle = \langle k, \langle m, n \rangle \rangle$.

Jednak nie wszystkie funkcje klasy $\mathcal{RF}$ należą do klasy $\mathcal{P}$. Przykładem jest funkcja Ackermanna zdefiniowana następująco:

$$\begin{aligned} &A(0, m) = m + 1 \\ &A(\mathfrak{s}(n), 0) = A(n, 1) \\ &A(\mathfrak{s}(n), \mathfrak{s}(m)) = A(n, A(\mathfrak{s}(n), m)), \end{aligned}$$

która jest funkcją rekurencyjną całkowitą, ale nie jest funkcją pierwotnie rekurencyjną [52].

Wniosek 2.1 [52] *Zachodzą następujące zależności między klasami funkcji* $\mathcal{P} \subsetneq \mathcal{RF} \subsetneq \mathcal{PRF}$.

Wprowadzimy następujące oznaczenie dla pojęcia dziedziny funkcji f, $\mathrm{Dom}(f) = \{x : \exists_y f(x) = y\}$. Graf G_f funkcji f zdefiniujemy jako zbiór (relację) w następujący sposób: $G_f(\bar{n}, m) \Leftrightarrow \bar{n} \in \mathrm{Dom}(f)$ i $f(\bar{n}) = m$.

Przytoczymy teraz dwa ważne dla naszych rozważań twierdzenia.

Twierdzenie 2.1 [52] *Niech f i g będą funkcjami zdefiniowanymi na zbiorze liczb naturalnych o wartościach naturalnych. Wówczas:*

- *f jest funkcją częściowo rekurencyjną wtedy i tylko wtedy, gdy jej graf G_f jest rekurencyjnie przeliczalny,*
- *g jest funkcją całkowicie rekurencyjną wtedy i tylko wtedy, gdy jej graf G_g jest rekurencyjny.*

Twierdzenie 2.2 [52] *Zbiór A jest rekurencyjny wtedy i tylko wtedy, gdy jest rekurencyjnie przeliczalny oraz jego dopełnienie $(\neg A)$ jest rekurencyjnie przeliczalne.*

Niech $\mathcal{W}[n_1, n_2, \ldots, m_1, m_2, \ldots]$ [1] będzie pierścieniem wielomianów określonych na nieskończonym, przeliczalnym zbiorze niewiadomych o współczynnikach ze zbioru liczb całkowitych $\mathbb{Z}$.

Przedstawimy teraz fakt dotyczący zbiorów rekurencyjnie przeliczalnych [52].

Twierdzenie 2.3 [52] *Niech $p(n_1, \ldots, n_k, m_1, \ldots, m_l) \in \mathcal{W}$ będzie wielomianem o $(k+l)$ niewiadomych, $k, l > 0$, gdzie $n_1, \ldots, n_k, m_1, \ldots, m_l$ są liczbami naturalnymi. Zbiór D (zbiór diofantyczny)*

$$\langle n_1, \ldots, n_k \rangle \in D \Leftrightarrow (\exists (m_1, \ldots, m_l)) p(n_1, \ldots, n_k, m_1, \ldots, m_l) = 0$$

jest rekurencyjnie przeliczalny i odwrotnie każdy zbiór rekurencyjnie przeliczalny jest zbiorem diofantycznym.

[1] Czasem, dla skrócenia zapisu, będziemy pisać po prostu $\mathcal{W}$. Rozważamy tutaj wielomiany o nieskończonym, przeliczalnym zbiorze niewiadomych, z których część może być związana kwantyfikatorem, dla odróżnienia oznaczamy je przez $m_1, m_2, \ldots$.

Więcej informacji na temat powyższego wyniku możemy znaleźć w pracach [23], [39]. Warto wspomnieć, że twierdzenie 2.3 negatywnie rozstrzyga Dziesiąty Problem Hilberta [29].

Teoria rekursji wprowadziła w zbiorze relacji hierarchię zwaną hierarchią arytmetyczną [52]. Hierarchię tę definiujemy następująco:

Definicja 2.8 [52] *Relacja $R \subset \mathbb{N}^k$ należy do klasy Σ^0_{n+1}, jeżeli istnieje relacja $Q \subset \mathbb{N}^{k+1}$ klasy Π^0_n, taka że*

$$R(\bar{n}) \Leftrightarrow (\exists t)Q(t, \bar{n})$$

Relacja $R \subset \mathbb{N}^k$ należy do klasy Π^0_{n+1}, jeżeli istnieje relacja $Q \subset \mathbb{N}^{k+1}$ klasy Σ^0_n, taka że

$$R(\bar{n}) \Leftrightarrow (\forall t)Q(t, \bar{n})$$

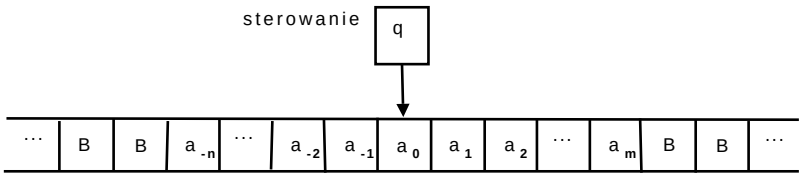
Relacja $R \subset \mathbb{N}^k$ należy do klasy Π^0_0, Σ^0_0, jeżeli jest relacją rekurencyjną.
Relacja $R \subset \mathbb{N}^k$ należy do klasy Δ^0_0, jeżeli należy do klasy Π^0_0 i należy do klasy Σ^0_0.

Lemat 2.1 [52] *Σ^0_1 to klasa relacji rekurencyjnie przeliczalnych.*

2.1.3 Maszyna Turinga

W kolejnym paragrafie przejdziemy do zaprezentowania innego ważnego modelu obliczeń, czyli maszyn Turinga. W latach trzydziestych XX wieku, a zatem przed pojawieniem się komputerów, A. Turing badał abstrakcyjną maszynę, która obecnie jest uważana za model klasycznego komputera cyfrowego. Maszyny Turinga są najbardziej użytecznym modelem jeśli chodzi o wyznaczanie klas złożoności problemów obliczeniowych. Oto podstawowa definicja maszyny Turinga [52].

Definicja 2.9 [52] *Maszynę Turinga* (MT) *będziemy rozumieć jako uporządkowaną piątkę $M = (Q, \Sigma, \delta, q_0, F)$, gdzie $Q = \{q_0, q_1, \ldots, q_k\}$, $k \geqslant 0$ jest skończonym zbiorem stanów, $\Sigma = \{s_0, s_1 \ldots, s_m\}$, $m \geqslant 0$ jest skończonym zbiorem symboli taśmowych (łącznie z wyróżnionym symbolem pustym B), $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, +1\}$ jest częściową funkcją nazywaną funkcją przejścia, -1 i $+1$ oznaczają odpowiednio ruch głowicy w lewo i ruch głowicy w prawo oraz 0 oznacza brak ruchu głowicy, q_0 i F oznaczają odpowiednio stan startowy i zbiór stanów finalnych.*



Rysunek 2.2: Model maszyny Turinga

Działanie maszyny Turinga możemy zdefiniować poprzez pojęcie konfiguracji. Konfigurację będziemy rozumieli jako aktualną zawartość taśmy maszyny Turinga wraz z określeniem aktualnego symbolu pod głowicą i stanu w jakim znajduje się maszyna Turinga (patrz rys. 2.2). Maszyna Turinga ma nieskończoną taśmę wypełnioną symbolami pustymi, oprócz być może skończonej liczby komórek, w których na starcie zapisane jest słowo wejściowe oraz komórek odwiedzonych przez głowicę maszyny w trakcie jej działania. Zatem słowo zapisane na taśmie zawiera zawsze nieskończony prefiks i nieskończony sufiks składający się z symboli pustych. W związku z tym, mówiąc o konfiguracji maszyny Turinga, będziemy uwzględniać tylko zawartość komórek zapisaną pomiędzy pierwszą od lewej komórką niepustą a pierwszą od prawej komórką niepustą. W szczególnym przypadku, kiedy głowica znajduje się nad jednym z pustych symboli w początkowej lub końcowej części taśmy, będziemy uwzględniać też pewną skończoną liczbę symboli pustych, odpowiednio na lewo lub na prawo od komórek niepustych. Formalnie konfiguracja M jest trójką (q, v, u), gdzie $q \in Q$, a v i u należą do Σ^* [2]. Słowo v jest zawartością niepustej części taśmy na lewo od głowicy (być może ze skończonym sufiksem składającym się z symboli pustych), u jest zawartością niepustej części taśmy na prawo od głowicy (być może ze skończonym prefiksem składającym się z symboli pustych) wraz z symbolem pod głowicą.

Definicja 2.10 [52] *Mówimy, że maszyna M z konfiguracji (q, v, u) przechodzi w jednym kroku do konfiguracji (q', v', u'), co oznaczamy $(q, v, u) \xrightarrow{M} (q', v', u')$, jeżeli zachodzi co następuje:*

- *a jest pierwszym symbolem u oraz $\delta(q, a) = (p, a', R)$, wówczas $q' = p$;*
- *jeżeli $R = +1$, wówczas v' jest równe v z dołączonym na końcu symbolem a', czyli dla $v = yb$ mamy $v' = yba'$, u' jest równe u z usuniętym pierwszym symbolem, czyli dla $u = ax$ mamy $u' = x$;*

[2] Poprzez Σ^* rozumiemy zbiór słów skończonych nad alfabetem Σ; Σ^i rozumiemy jako zbiór słów i-literowych nad alfabetem Σ.

– *jeżeli* $R = -1$*, wówczas* v' *jest równe* v *z usuniętym ostatnim symbolem, czyli dla* $v = yb$ *mamy* $v' = y$ *(jeżeli* v *było puste to* v' *jest również puste),* u' *jest równe* u *z pierwszym symbolem zastąpionym przez* a' *i dołączonym na początku ostatnim symbolem z* v*, czyli dla* $u = ax$ *mamy* $u' = ba'x$ *(jeżeli* v *jest słowom pustym to dołączamy symbol* B*);*

– *jeżeli* $R = 0$*, wówczas* v' *jest równe* v*,* u' *jest równe* u *z zastąpionym pierwszym symbolem przez* a'*, czyli dla* $u = ax$ *mamy* $u = a'x$*.*

Oczywiście maszyna Turinga w trakcie swoich obliczeń wykonuje wiele kroków zmieniających konfigurację. Jedna konfiguracja (q', v', u') może być otrzymana z drugiej (q, v, u) w k krokach, jeżeli istnieje ciąg konfiguracji (q_i, v_i, u_i), $i = 1, \ldots, k+1$, taki że:

$$(q, v, u) = (q_1, v_1, u_1),$$
$$(q', v', u') = (q_{k+1}, v_{k+1}, u_{k+1}),$$
$$\forall (i = 1, \ldots, k)(q_i, v_i, u_i) \xrightarrow{M} (q_{i+1}, v_{i+1}, u_{i+1}).$$

Takie k-krokowe przejście oznaczymy: $(q, v, u) \xrightarrow{M^k} (q', v', u')$, dla skończonego $k > 0$ będziemy również stosować zapis: $(q, v, u) \xrightarrow{M^*} (q', v', u')$.

Przyjmujemy, że maszyna Turinga rozpoczyna pracę dla następującej konfiguracji: taśma zawiera słowo wejściowe [3] $\boldsymbol{n}$, poza tym jest wypełniona symbolami pustymi, głowica maszyny wskazuje na pierwszy od lewej symbol słowa wejściowego oraz maszyna Turinga znajduje się w stanie startowym q_0, stąd też zapis konfiguracji startowej przyjmuje postać: $(q_0, \varepsilon, \boldsymbol{n})$. [4] Maszyna zmienia w trakcie działania swoją konfiguracje zgodnie z tym, co nakazuje funkcja przejścia δ oraz kończy swoje działanie, gdy przejdzie w któryś ze stanów finalnych q_f określonych w zbiorze F. Cała zawartość taśmy od głowicy do pierwszej od prawej komórki niepustej rozumiana jest jako wynik obliczeń maszyny Turinga. Zakładamy również, że na lewo od głowicy wówczas znajdują się tylko komórki puste. Zatem maszyna Turinga kończy działanie dla konfiguracji $(q_f, \varepsilon, \boldsymbol{m})$, gdzie $\boldsymbol{m}$ jest wynikiem obliczeń maszyny Turinga.

[3] W celu określenia związku między maszyną Turinga a funkcjami określonymi w zbiorze liczb naturalnych $\mathbb{N}$ słowa zapisame na taśmie maszyny Turinga będziemy rozumieć jako zapis liczb naturalnych w pewnym ustalonym systemie kodowania. W celu podkreślenie tego związku a zarazem uniknięcia niejednoznaczności przy zapisie słowa na taśmie maszyny Turinga będziemy używać czcionki pogrubionej $\boldsymbol{n}$, natomiast gdy będziemy mieli na myśli odpowiadającą temu słowu liczbę naturalną będziemy używać czcionki normalnej n.

[4] Symbol ε oznacza słowo puste.

Definicja 2.11 [52] *Obliczeniem maszyny Turinga* $M = (Q, \Sigma, \delta, q_0, F)$ *dla słowa* $\boldsymbol{n} \in (\Sigma \setminus \{B\})^*$ *nazywamy przejście* $(q_0, \varepsilon, \boldsymbol{n}) \overset{M^*}{\rightarrow} (q_f, \varepsilon, \boldsymbol{m})$, $\boldsymbol{n}, \boldsymbol{m} \in \Sigma^*$, *które oznaczymy* $M(\boldsymbol{n}) = \boldsymbol{m}$ *i będziemy mówić, że słowo* $\boldsymbol{n}$ *jest wejściem maszyny* M*, a słowo* $\boldsymbol{m}$ *wyjściem maszyny* M*. Jeżeli dla słowa* $\boldsymbol{n}$ *maszyna Turinga* M *nie osiąga stanu końcowego* q_f *to zapiszemy* $M(\boldsymbol{n}) \uparrow$.

Definicja 2.12 [52] *Funkcję naturalną* $f(n)$, $n \in \mathbb{N}$, *nazywamy funkcją obliczalną w sensie Turinga, jeżeli istnieje maszyna Turinga* M_f*, taka że dla dowolnej liczby* $n \in \mathbb{N}$*:*

$$M_f(\boldsymbol{n}) = \boldsymbol{m} \Leftrightarrow f(n) = m$$

lub

$$M_f(\boldsymbol{n}) \uparrow \Leftrightarrow f(n) \text{ jest niezdefiniowana.}$$

Aby zdefiniować jakąś maszynę Turinga musimy opisać jej wszystkie pięć elementów. Opisanie tych wszystkich elementów, poza najprostszymi przypadkami maszyn Turinaga, bywa kłopotliwe. Nie chcemy poświęcać za dużo czasu na definiowanie maszyn Turinga, dlatego w dalszym ciągu pracy będziemy opisywać maszyny Turinga na nieco wyższym poziomie, który będzie wystarczająco szczegółowy i przy okazji łatwiejszy do zrozumienia. Jednakże powinniśmy pamiętać, że każdy taki opis jest tylko skrótem opisu formalnego.

Aby przedstawić związek między formalnym opisem maszyny Turinga, a jej mniej formalnym opisem, podamy przykład obydwu sposobów opisu pewnej maszyny Turinga.

Przykład 2.3 Opiszemy maszynę Turinga $M_{\mathfrak{s}}$ wyznaczającą $\mathfrak{s}(n)$, gdzie liczba $n \in \mathbb{N}$ jest zapisana binarnie na taśmie maszyny Turinga.

$M_{\mathfrak{s}}(\boldsymbol{n})\{$

1. przesuń głowicę od lewej do prawej na ostatni symbol zapisany na taśmie;
2. jeśli pod głowicą znajduje się 0, zmień na 1 i zakończ obliczenia;
3. jeśli pod głowicą znajduje się 1, zmień na 0, przesuń głowicę na prawo i przejdź do kroku 2;

4. jeśli pod głowicą jest symbol B, zmień na 1 i zakończ obliczenia;

5. zwróć zawartość taśmy jako wynik obliczeń;

}

Krok 5 oznacza przesunięcie głowicy na skrajnie lewy niepusty symbol reprezentujący wynik obliczeń maszyny $M_{\mathfrak{s}}$, zaś wszystkie komórki taśmy poza wynikowymi są wypełnione symbolami pustymi.

Teraz przedstawimy formalny opis maszyny $M_{\mathfrak{s}} = (Q, \Sigma, \delta, q_0, F)$:
$Q = \{q_0, q_1, q_2, q_3\}$;
$\Sigma = \{0, 1, B\}$, gdzie B jest wyróżnionym symbolem pustym;
q_0 jest stanem startowym;
$F = \{q_3\}$;
funkcję przejścia δ opiszemy w Tabeli 2.1:

Tabela 2.1: Funkcja przejścia δ dla maszyny Turinga $M_{\mathfrak{s}}$

Lp.	(Q, Σ)	$(Q, \Sigma, \{+1, -1\})$
1	$(q_0, 1)$	$(q_0, 1, +1)$
2	$(q_0, 0)$	$(q_0, 0, +1)$
3	(q_0, B)	$(q_1, B, -1)$
4	$(q_1, 1)$	$(q_1, 0, -1)$
5	$(q_1, 0)$	$(q_2, 1, -1)$
6	(q_1, B)	$(q_2, 1, -1)$
7	$(q_2, 1)$	$(q_2, 1, -1)$
8	$(q_2, 0)$	$(q_2, 0, -1)$
9	(q_2, B)	$(q_3, B, +1)$

Zilustrujemy teraz jak zmieniają się konfiguracje maszyny $M_{\mathfrak{s}}$ w kolejnych krokach obliczenia dla $n = 11$.

$$\begin{array}{lllllllll} (q_0, \varepsilon, 1011) & \rightarrow & (q_0, 1, 011) & \rightarrow & (q_0, 10, 11) & \rightarrow & (q_0, 101, 1) & \rightarrow \\ (q_0, 1011, \varepsilon) & \rightarrow & (q_1, 101, 1) & \rightarrow & (q_1, 10, 10) & \rightarrow & (q_1, 1, 000) & \rightarrow \\ (q_2, \varepsilon, 1100) & \rightarrow & (q_2, \varepsilon, B1100) & \rightarrow & (q_3, \varepsilon, 1100) & & & \end{array}$$

Końcowa konfiguracja przedstawia nam taśmę z binarnym zapisem liczby 12.

W dalszej części pracy będziemy zakładać, że jeżeli wynikiem działania pewnej maszyny Turinga będzie więcej niż jedna liczba, wówczas na taśmie po

osiągnięciu przez tą maszynę stanu finalnego pozostaje słowo będące zapisem reprezentacji wszystkich wynikowych liczb oddzielonych od siebie pojedynczymi symbolami pustymi. Analogicznie w przypadku, gdy pewna maszyna Turinga ma przetwarzać więcej niż jedną liczbę, początkowy zapis taśmy stanowi słowo będące zapisem reprezentacji wszystkich tych liczb oddzielonych od siebie pojedynczymi symbolami pustymi.

W informatyce teoretycznej istotne miejsce zajmują rozważania dotyczące ilości czasu i miejsca potrzebnych do rozwiązania poszczególnych problemów algorytmicznych. Dla maszyn Turinga można w bardzo łatwy sposób badać potrzebną ilość czasu i pamięci.

Czasem pracy maszyny M dla słowa wejściowego $\boldsymbol{n}$ będziemy nazywali liczbę kroków, po których maszyna M się zatrzyma czyli długość ciągu obliczenia maszyny Turinga: $(q_0, \varepsilon, \mathbf{n}) \overset{M^*}{\rightarrow} (q_f, \varepsilon, \mathbf{m})$. W przykładzie 2.3 czas pracy maszyny $M_{\mathfrak{s}}$ dla $n = 11$ wynosi 10. Jeżeli M nie zatrzymuje się dla $\boldsymbol{n}$, to czas pracy dla $\boldsymbol{n}$ jest nieskończony. Czas pracy maszyny M dla słowa $\boldsymbol{n}$ oznaczymy przez $T(M, \boldsymbol{n})$

Definicja 2.13 [53] *Złożonością czasową maszyny M z własnością stopu* [5] *nazwiemy funkcję $f : \mathbb{N} \to \mathbb{N}$, taką że*

$$\forall(i \in \mathbb{N}) : f(i) = \max\{T(M, \boldsymbol{n}) : \boldsymbol{n} \in \Sigma^i\}.$$

Intuicyjnie $f(i)$ określa czas wystarczający do obliczenia na każdym i-literowym słowie wejściowym.

Ponieważ funkcja określająca dokładny czas działania maszyny Turinga często jest opisywana skomplikowanym wyrażeniem, przeważnie tylko ją szacujemy. Najbardziej interesuje nas określenie przybliżonego czasu działania maszyny dla dużych słów wejściowych. Dlatego też, uwzględniamy tylko składniki najwyższego rzędu w wyrażeniach opisujących czas działania maszyny Turinga, pomijając składniki niższych rzędów, ponieważ dla dużych wartości składniki wyższych rzędów dominują pozostałe. Na przykład funkcja $f(n) = 12n^4+5n^2-5n+35$ ma cztery składniki. Składnikiem najwyższego rzędu jest $12n^4$. Opuszczając 12, powiemy, że f jest asymptotycznie co najwyżej n^4, zapisujemy ten fakt następująco: $f(n) = O(n^4)$.

[5] Jeżeli maszyna Turinga zatrzymuje się dla wszystkich słów wejściowych to mówimy, że ma własność stopu.

Definicja 2.14 [53] *Niech $f, g : \mathbb{N} \to \mathbb{R}^+$. Mówimy, że $f(n) = O(g(n))$ jeżeli istnieją $c, n_0 \in \mathbb{N}^+$ takie, że dla każdego $n \geqslant n_0$:*

$$f(n) \leqslant cg(n).$$

Gdy $f(n) = O(g(n))$ mówimy, że $g(n)$ jest ograniczeniem górnym dla $f(n)$ lub dokładniej, że $g(n)$ jest asymptotycznym ograniczeniem górnym dla $f(n)$, w celu podkreślenia faktu, że opuszczamy stały mnożnik.

Pamięcią potrzebną do obliczenia maszynie M dla słowa wejściowego $\boldsymbol{n}$ będziemy nazywali liczbę komórek taśmy, które zostały odwiedzone przez głowicę w trakcie całego działania maszyny M dla $\boldsymbol{n}$. Jeżeli M nie zatrzymuje się dla $\boldsymbol{n}$ to pamięć potrzebna do obliczenia jest nieokreślona. Pamięć potrzebną do obliczenia maszynie M dla słowa wejściowego $\boldsymbol{n}$ oznaczymy przez $S(M, \boldsymbol{n})$.

Definicja 2.15 [53] *Złożonością pamięciową maszyny M z własnością stopu nazwiemy funkcję $f : \mathbb{N} \to \mathbb{N}$, taką że*

$$\forall(i \in \mathbb{N}) : f(i) = \max\{S(M, \boldsymbol{n}) : \boldsymbol{n} \in \Sigma^i\}$$

Intuicyjnie $f(i)$ określa pamięć wystarczającą do obliczenia na każdym i-literowym słowie wejściowym.

W analizie złożoności problemów kluczowa rolę odgrywa jednak nierealny model nazywany niedeterministyczną maszyną Turinga, będący rozszerzeniem wersji deterministycznej.

Definicja 2.16 [52] *Poprzez niedeterministyczną maszynę Turinga* (NMT) *będziemy rozumieć uporządkowaną piątkę: $M = (Q, \Sigma, \Delta, q_0, F)$, gdzie $Q = \{q_0, q_1, \ldots, q_k\}$, $k \geqslant 0$ jest skończonym zbiorem stanów, $\Sigma = \{s_0, s_1 \ldots, s_m\}$, $m \geqslant 0$ jest skończonym zbiorem symboli taśmowych (łącznie z wyróżnionym symbolem pustym B), $\Delta \subset (Q \times \Sigma) \times (Q \times \Sigma \times \{-1, 0, +1\})$ jest częściową relacją nazywaną relacją przejścia, -1 i $+1$ oznaczają odpowiednio ruch głowicy w lewo i ruch głowicy w prawo oraz 0 oznacza brak ruchu głowicy, q_0 i F oznaczają odpowiednio stan startowy i zbiór stanów finalnych.*

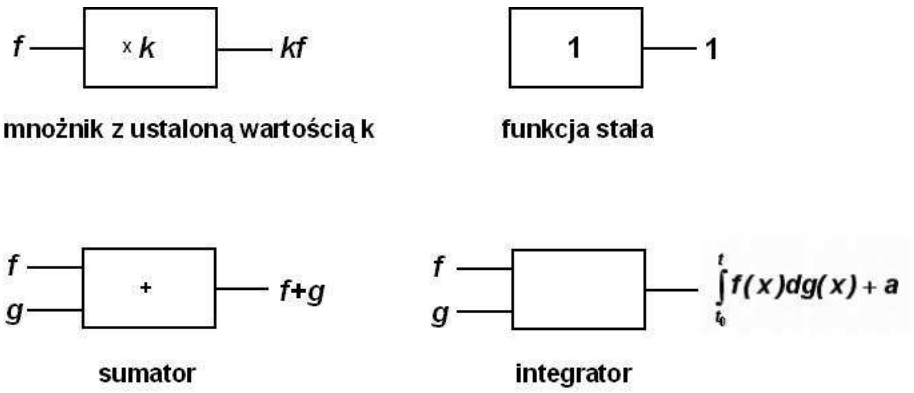
Niedeterministyczna maszyna Turinga jest podobna do zwykłej maszyny Turinga, elementem różniącym jest Δ, która oznacza, że niedeterministyczna maszyna Turinga nie ma jednoznacznie określonej kolejnej operacji do wykonania, ale ma możliwość wyboru między kilkoma alternatywnymi zachowaniami. Dla każdej niedeterministycznej maszyny Turinga istnieje jej deterministyczny odpowiednik [71].

2.2 GPAC

Przejdziemy teraz do przedstawienia kilku modeli obliczeń na liczbach rzeczywistych. Jako pierwszy, zaprezentujemy podstawowy model obliczeń analogowych, którego prototypem był analizator różniczkowy, mechaniczny komputer wymyślony przez V. Busha [7], przeznaczony do rozwiązywania prostych równań różniczkowych postaci $d^2y/dx^2 = -y$. Był on wykorzystywany od lat trzydziestych do wczesnych lat sześćdziesiątych przy rozwiązywaniu problemów numerycznych. W 1941 roku C. Shannon wprowadził matematyczny model wspomnianego analizatora różniczkowego i nazwał go General Purpose Analog Computer (GPAC) [68].

Działanie maszyn, które opisuje model GPAC bazuje na czterech analogowych jednostkach liczących łączonych ze sobą w obwody różnych kształtów. Są to:

- Integrator: jednostka z dwoma parametrami startowymi: stałe a i t_0; dwa wejścia - jednoargumentowe funkcje f, g; jedno wyjście - całka Riemanna-Stieltjesa: $\int_{t_0}^{t} f(x)dg(x) + a$.
- Sumator: jednostka, dwa wejścia - funkcje f, g; jedno wyjście - $f + g$.
- Mnożnik: jednostka, dwa wejścia - funkcje f, g; jedno wyjście - $f * g$.
- Funkcja stała: jednostka skojarzona z liczbą rzeczywista c, bez wejść; jedno wyjście - zawsze równe c.



Rysunek 2.3: Podstawowe analogowe jednostki liczące modelu GPAC

Model GPAC, wprowadzony przez Shannona, był później udoskonalany przez wielu autorów: Pour-El [62], Rubel [66], Lipshitz [37], Graça, Costa [26, 14]. Przyjrzyjmy się teraz kilku wynikom przedstawionym w literaturze [26, 14, 37, 62], dotyczącym modelu GPAC.

Definicja 2.17 [14] *Funkcja jednoargumentowa $f(x)$ jest różniczkowalnie algebraiczna na przedziale I, jeżeli istnieje liczba naturalna $n, (n > 0)$ i pewien $(n+2)$-wymiarowy niezerowy wielomian p o rzeczywistych współczynnikach, taki że*

$$p(x, f(x), f'(x), \ldots, f^{(n)}(x)) = 0$$

dla wszystkich $x \in I$.

Shannon w swoim artykule [68] stwierdza:

Twierdzenie 2.4 [68] *Funkcja jednoargumentowa może być wygenerowana w modelu GPAC wtedy i tylko wtedy, gdy jest różniczkowalnie algebraiczna.*

Wynik ten pokazuje, że duża klasa funkcji takich jak wielomiany, funkcje trygonometryczne, itp. są obliczalne w modelu GPAC. Z drugiej strony pewne funkcje, takie jak $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$ czy $\zeta(x) = \frac{1}{\Gamma(x)} \int_0^\infty \frac{z^{x-1}}{e^z-1}dz$, nie mogą być wyznaczone przez GPAC, ponieważ nie są algebraicznie różniczkowalne. Jednakże M. B. Pour-El w pracy [62] pokazała, że w dowodzie twierdzenia 2.4 były pewne luki. Aby pokazać, że relacja o której mowa w twierdzeniu 2.4 zachodzi Pour-El wprowadziła alternatywną definicję GPACa oznaczaną przez T-GPAC.

Definicja 2.18 [62] *Funkcja jednoargumentowa f jest generowana w modelu T-GPAC na przedziale I, jeżeli istnieje zbiór jednoargumentowych funkcji $f_1, \ldots, f_n$ oraz zbiór warunków początkowych $f_i(x_0) = f_i^*$, $i = 1, \ldots, n$, gdzie $x_0 \in I$, takich że:*

1. *$f = (f_1, \ldots, f_n)$ jest jednoznacznym rozwiązaniem na przedziale I układu równań ODE postaci:*

$$A(x, f)\frac{df}{dx} = b(x, f) \tag{2.1}$$

spełniających warunki początkowe, gdzie $A(x, f)$ oraz $b(x, f)$ są odpowiednio $n \times n$ oraz $n \times 1$ wymiarowymi macierzami. Dalej, każde wystąpienie A i b musi być liniowe względem $1, x, f_1, \ldots, f_n$.

2. *Dla pewnego* $i \in \{1, \ldots, n\}, f = f_i$ *na* I.

3. $(x_0, f_1^*, \ldots, f_n^*)$ *jest zdefiniowane w odniesieniu do równania (2.1), tj. istnieją zamknięte przedziały* $J_0, J_1, \ldots, J_n$ *(z niepustym wnętrzem), takie że* $(x_0, f_1^*, \ldots, f_n^*)$ *jest punktem wewnętrznym* $J_0 \times J_1 \times \ldots \times J_n$*, dalej przez* $(y_0, g_1^*, \ldots, g_n^*) \in J_0 \times J_1 \times \ldots \times J_n$ *będziemy rozumieli, iż istnieją funkcje jednoargumentowe* $g_1, \ldots, g_n$*, takie że:*

   (a) $g_i(y_0) = g_i^*$ *dla* $i = 1, \ldots, n$;

   (b) $(g_1, \ldots, g_n)$ *spełnia (2.1) na pewnym przedziale* I^* *o niepustym wnętrzu, takim że* $y_0 \in I^*$;

   (c) $(g_1, \ldots, g_n)$ *jest jednoznaczne na* I^*.

Punkt 3 definicji T-GPACa gwarantuje, że rozwiązanie równania (2.1) pozostaje jednoznaczne przy dostatecznie małych zmianach warunków początkowych. Pour-El dowiodła (z pewnymi poprawkami dodanymi przez Lipshitza i Rubla w pracy [37]) następujące twierdzenie.

Twierdzenie 2.5 [62] *Niech* f *będzie funkcją różniczkowalnie algebraiczną zdefiniowaną na* I*. Wówczas istnieje zamknięty przedział* $I' \subset I$ *o niepustym wnętrzu, taki że na* I'*,* f *może być wygenerowana przez T-GPAC.*

Mamy również odwrotne twierdzenie.

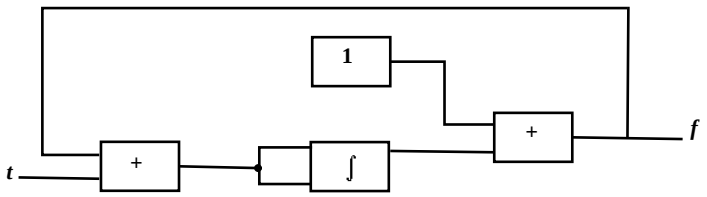
Twierdzenie 2.6 [62, 37] *Jeżeli* f *jest generowana na* I *przez T-GPAC, wówczas istnieje zamknięty przedział* $I' \subset I$ *o niepustym wnętrzu, taki że na* I'*,* f *jest funkcją różniczkowalnie algebraiczną.*

Model zaproponowany przez Pour-El jest tylko pozornie inny niż model Shannona, Pour-El przedstawiła również następujący wynik.

Twierdzenie 2.7 [62] *Jeżeli funkcja* f *jest generowana na* I *przez GPAC, jest również generowana przez T-GPAC na* I.

Niestety, dowód twierdzenia 2.7 zawierał pewne nieścisłości, co zauważono w pracy [37]. Nie wiadomo dokładnie jakie są relacje pomiędzy funkcjami generowanymi przez T-GPAC a funkcjami generowanymi przez istniejące komputery analogowe (analizator różniczkowy), których odpowiednikiem jest GPAC.

Na bazie modelu GPAC Shannona, D. S. Graça i J. F. Costa [14] zaproponowali bardziej użyteczną notację tego modelu nazwaną FF-GPAC. Udoskonalili oni definicję GPACa omijając problematyczne przypadki oraz pokazali, że model ten jest równoważny rozwiązaniom wielomianowych równań ODE. D. S. Graça i J. F. Costa zauważyli, że poza nieścisłościami w dowodzie twierdzenia 2.4, występuje jeszcze inny problem w modelu GPAC. Rozwiązania generowane przez GPACa mogą być niejednoznaczne. Jako przykład podali obwód przedstawiony na Rysunku 2.4.



Rysunek 2.4: Obwód GPAC generujący dwa różne wyniki

Wynik generowany w przedstawionym obwodzie GPACa, to funkcja zależna od czasu t, dana wzorem:

$$f(t) = 1 + \int_0^t (f(x) + x) d(f(x) + x).$$

Przypuśćmy, że $t_0 = 0$ oraz że warunek początkowy wyniku całkowania jest 0. Jeśli wystartujemy obliczenia otrzymamy dwa możliwe wyniki:

$$f_{\pm}(t) = 1 \pm \sqrt{-2t} - t.$$

Dlatego nie może istnieć fizyczna implementacja tego obwodu, czyli nie istnieje analizator różniczkowy symulujący taki obwód. Wprowadzony przez D. S. Gracę i J. F. Costę nowy model FF-GPAC, bazujący na modelu GPAC, nie generuje tego rodzaju problemów. Wprowadzili oni następującą definicję:

Definicja 2.19 [14] *Obwód liniowy jest to acykliczny GPAC zbudowany wyłącznie z sumatora, stałego mnożnika i funkcji stałej w następujący sposób:*

1. *Stały mnożnik jest obwodem liniowym o jednym wejściu i jednym wyjściu.*

2. *Sumator jest obwodem liniowym o dwóch wejściach i jednym wyjściu.*

3. *Funkcja stała jest obwodem liniowym bez wejść i jednym wyjściu.*

4. *Jeżeli A jest obwodem liniowym i jeżeli wyjście A podłączymy do stałego mnożnika, wynikowy GPAC jest obwodem liniowym. Jego wejścia to wejścia A a wyjście to wyjście stałego mnożnika;*

5. *Jeżeli A i B są obwodami liniowymi i jeżeli wyjścia A i B podłączymy do sumatora, wówczas wynikowy GPAC jest obwodem liniowym. Jego wejścia to wejścia A i B a wyjście to wyjcie sumatora.*

Mamy następujące twierdzenie.

Twierdzenie 2.8 [14] *Jeżeli $x_1, \ldots, x_n$ są wejściami obwodu liniowego, wówczas wynik tego obwodu przyjmuje postać $y = c_0 + c_1x_1 + \ldots + c_nx_n$, gdzie $c_0, c_1, \ldots, c_n$ są odpowiednimi stałymi. Odwrotnie, jeżeli $y = c_0 + c_1x_1 + \ldots + c_nx_n$, wówczas istnieje obwód liniowy o wejściach $x_1, \ldots, x_n$ i wyjściu y.*

Wprowadzono również nową jednostkę:

– Wejście: jednostka bez wejść; jedno wyjście.

Jednostkę tą możemy sobie wyobrazić jako rodzaj interfejsu wprowadzającego dane ze świata zewnętrznego. Przedstawimy teraz definicję modelu FF-GPAC zaproponowaną przez D. S. Gracę i J. F. Costę.

Definicja 2.20 [14] *Rozważmy GPAC U o n integratorach $U_1, U_2, \ldots, U_n$. Przypuśćmy, że każdy integrator U_i, $i = 1, 2, \ldots, n$ możemy połączyć z dwoma obwodami liniowymi A_i i B_i w ten sposób, że wejście wyrażenia całkowego i zmiennej całkowania integratora U_i są połączone odpowiednio z wyjściem A_i oraz B_i. Przypuśćmy również, że każde wejście obwodów liniowych A_i i B_i jest połączone z jednym z następujących elementów: wyjście integratora lub jednostka wejścia. Mówimy, że U jest to GPAC ze sprzężeniem wyprzedzającym (feed-forward GPAC, FF-GPAC) wtedy i tylko wtedy, gdy istnieje numeracja integratorów U, $U_1, U_2, \ldots, U_n$, taka że zmienna całkowania integratora k może być wyrażona w następujący sposób:*

$$c_k + \sum_{i=1}^{m} c_{ki}x_i + \sum_{i=1}^{k-1} \overline{c}_{ki}y_i, \quad dla\ wszystkich\ k = 1, \ldots, n,$$

gdzie y_i jest wyjściem U_i, dla $i = 1, 2, \ldots, n$, x_j jest wejściem skojarzonym z jednostką wejściową o numerze j oraz $c_k, c_{kj}, \overline{c}_{ki}$ są odpowiednimi stałymi dla wszystkich $k = 1, \ldots, n$, $j = 1, \ldots, m$ i $i = 1, \ldots, k-1$.

Ta definicja opisuje model GPACa z nałożonymi pewnymi ograniczeniami na dozwolone w obwodach GPACa sprzężenia wyprzedzające. Dzięki temu otrzymaliśmy model, który zachowuje sie lepiej w porównaniu z GPACiem. Ponadto, jeśli T-GPAC zastąpimy przez FF-GPAC, twierdzenia 2.5 oraz 2.6 pozostają w mocy. W pracy [14] pokazano także równoważność w sensie zbioru funkcji generowanych pomiędzy modelami T-GPAC a FF-GPAC.

Dalsze udoskonalenia notacji dotyczącej GPACa i GPAC obliczalności przedstawił Graça w pracy [26]. Wprowadził on pojęcie GPACa budowanego na bazie obwodów wielomianowych.

Definicja 2.21 [26] *Obwód wielomianowy jest to acykliczny GPAC zbudowany wyłącznie z sumatora, funkcji stałej i mnożnika w sposób analogiczny jak obwód liniowy.*

Ma miejsce również następujące twierdzenie.

Twierdzenie 2.9 [26] *Jeżeli $x_1, \ldots, x_n$ są wejściami obwodu wielomianowego, wówczas wynik tego obwodu przyjmuje postać $y = p(x_1, \ldots, x_n)$, gdzie p jest wielomianem o współczynnikach rzeczywistych. Odwrotnie, jeżeli $y = p(x_1, \ldots, x_n)$, wówczas istnieje obwód wielomianowy o wejściach $x_1, \ldots, x_n$ oraz wyjściu y.*

Oto formalna definicja GPACa budowanego na bazie obwodów wielomianowych.

Definicja 2.22 [26] *Rozważmy GPAC U o n integratorach $U_1, U_2, \ldots, U_n$ i jednym wejściu t. Przypuśćmy, że każdy integrator U_i, $i = 1, 2, \ldots, n$ możemy połączyć z obwodem wielomianowym A_i w ten sposób, że wejście wyrażenia całkowego jest połączone z wyjściem A_i. Przypuśćmy, że każde wejście obwodów wielomianowego A_i jest połączone z jednym z następujących elementów: wyjście integratora lub wejście t. Przypuśćmy również, że wejście zmiennej całkowania każdego integratora jest połączone z wejściem t. W takim przypadku mówimy, że U jest to wielomianowy GPAC (polynomial GPAC, P-GPAC) o wejściu t.*

Z pracy [26] pochodzi również następująca własność modelu P-GPAC.

Własność 2.1 [26] *Funkcja f jest generowana przez P-GPAC wtedy i tylko wtedy, gdy jest składową rozwiązania $f = (f_1, \ldots, f_n)$ równania $f' = p(f, t)$, gdzie p jest wektorem wielomianów.*

W pracy [26] Graça pokazał również, że funkcja Gamma Eulera jest obliczalna w modelu P-GPAC. Wydaje się to być sprzeczne z konsekwencjami twierdzeń 2.5 i 2.6 oraz faktem równoważności pomiędzy modelami T-GPAC i P-GPAC, ponieważ Γ nie jest funkcją algebraicznie różniczkowalną. Graça otrzymał obliczalność tej funkcji poprzez zmianę pojęcia obliczalności w modelu GPAC. Zwróćmy uwagę, że tradycyjnie zakłada się, że wyniki obliczeń w modelu GPAC otrzymujemy w czasie rzeczywistym, tzn. jeżeli na wejściu pojawi się wielkość t to na wyjściu obwodu w tym samym czasie pojawi się wynik $f(t)$, czyli samo obliczenie zajmuje czas 0. Natomiast w klasycznej analizie obliczeniowej [76] tak się nie dzieje. Podstawą w tej teorii jest to, iż funkcję $f(x)$ uznajemy za obliczalną jeżeli możemy ją aproksymować w efektywny sposób, być może z wykorzystaniem informacji zakodowanej w x. Zatem Graça wprowadził podobne pojęcie obliczalności w modelu GPAC. Niech $\|\cdot\|_\infty$ będzie normą supremum zdefiniowaną w $\mathbb{R}$ jak następuje: $\|(x_1, \ldots, x_n)\|_\infty = \max\{|x_1|, \ldots, |x_n|\}$.

Definicja 2.23 [26] *Funkcja $f : \mathbb{R}^n \to \mathbb{R}^k$ jest generowana przez aproksymację w modelu GPAC, jeżeli istnieje GPAC U z wejściem t, o co najmniej n integratorach z wartościami początkowymi $x_1, \ldots, x_n$, taki, że*

$$\|f(x_1, \ldots, x_n) - g(x_1, \ldots, x_n, t)\|_\infty \leqslant \varepsilon(x_1, \ldots, x_n, t),$$

gdzie $\lim_{t\to\infty} \varepsilon(x_1, \ldots, x_n, t) = 0$ oraz $g(x_1, \ldots, x_n, t)$ jest wektorem wyznaczonym przez k wyjść U, ε jest wyznaczone przez jedno wyjście U.

Definicja powyższa mówi głównie o tym, że możemy aproksymować z dowolną dokładnością wartość $f(x)$ za pomocą GPACa. Faktycznie możemy poczekać dostatecznie długi czas aby otrzymać żądaną dokładność δ. Ponadto dla każdego t mamy górne ograniczenie δ błędu aproksymacji dane przez wyjście ε. W podobny sposób możemy wprowadzić pojęcie obliczalności dla P-GPAC.

Zaznaczmy, że czas reprezentowany przez t nie musi odpowiadać fizycznemu czasowi. Ważne jest abyśmy zwrócili uwagę, że to podejście jest naturalnym z punktu widzenia dynamicznych systemów. Przedstawiamy pewne wejście jako zbiór wartości początkowych przez co określamy zachowanie systemu. Następnie rozważamy nowe wejście, czas, co pozwala nam obserwować ewolucję systemu (to jest proces obliczenia).

Ta zmiana w podejściu do GPAC obliczalności pozwoliła sformułować następujące interesujące twierdzenie.

Twierdzenie 2.10 [26] *Funkcja* Γ *jest generowana przez aproksymację w modelu GPAC, na przedziale* $(0, +\infty)$.

Dowód. Aby wykazać powyższe twierdzenie oprzemy się na własność 2.1. Chcemy wygenerować funkcję daną wzorem $f_x(t) = t^{x-1}e^{-t}$ celem obliczenia całki $\int_0^\infty t^{x-1}e^{-t}dt$. Zaznaczmy że ta całka jest zbieżna tylko na przedziale $(0, +\infty)$. Łatwo zauważyć, że f_x jest rozwiązaniem równania:

$$f' = \frac{(x-1)}{t}f - f, \quad f(1) = \frac{1}{e}. \tag{2.2}$$

Jednakże f nie jest zdefiniowana dla $t = 0$. Zatem, potrzebujemy kilku dalszych kroków w celu zdefiniowania funkcji Γ. Zapiszmy funkcję Γ następująco

$$\Gamma(x) = \int_0^1 t^{x-1}e^{-t}dt + \int_1^\infty t^{x-1}e^{-t}dt.$$

Podstawiając $w = \frac{1}{t}$ w pierwszej całce otrzymamy

$$\int_0^1 t^{x-1}e^{-t}dt = \int_1^\infty \left(\frac{1}{w}\right)^{x+1} e^{-\frac{1}{w}}dw,$$

stąd

$$\Gamma(x) = \int_1^\infty \left(\frac{1}{t}\right)^{x+1} e^{-\frac{1}{t}} + t^{x+1}e^{-t}dt.$$

Możemy przyjąć $t_0 = 1$ i przeprowadzić obliczenia dla $t \to \infty$. Postępujemy w następujący sposób, $\frac{1}{t}$ generujemy w P-GPAC jako rozwiązanie równania

$$y' = -y^2, \quad y(1) = 1.$$

Biorąc pod uwagę własność 2.1 oraz równanie (2.2) funkcja f_x może być wygenerowana przez P-GPAC, gdzie x jest dane przez jednostkę generującą stałą. Zatem możemy stwierdzić, że wyrażenie podcałkowe

$$\left(\frac{1}{t}\right)^{x+1} e^{-\frac{1}{t}} + t^{x+1}e^{-t} = \frac{f_x(t^{-1})}{t^2} + f_x(t)$$

jest wygenerowane przez pewien obwód P-GPAC. Używając wyjścia tego obwodu jako wejścia integratora oraz biorąc t jako drugie wejście integratora (zmienna całkowania) otrzymamy wyjście integratora zbieżne do

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt \quad \text{przy } t \to \infty.$$

Oznaczmy ten obwód jako U'.

Sprawdzimy teraz czy warunek z definicji 2.23 jest spełniony. Rozważmy szereg Taylora dla wyrażenia $z^{x-1}e^{-z}$ dla $z \geqslant 0$, wówczas mamy

$$z^{x-1}e^{-z} = z^{x-1}\left(\sum_{i=0}^{\infty}\frac{z^i}{i!}\right)^{-1} \leqslant z^{x-1}\left(\frac{z^k}{k!}\right)^{-1} \leqslant k!z^{-2},$$

gdy $k \geqslant x+1$ całkowite oraz $z \geqslant 1$. Biorąc $k = \lceil x+1 \rceil$ oraz $t \geqslant 1$, mamy

$$\left|\int_1^{\infty} z^{x-1}e^{-z} - \int_1^{t} z^{x-1}e^{-z}\right| = \int_t^{\infty} z^{x-1}e^{-z}dz \leqslant \frac{k!}{t} \leqslant \frac{(x+2)^{(x+2)}}{t}.$$

Ponadto, dla $z \geqslant 0, z^{x-1}e^{-z} \leqslant z^{x-1}$ i

$$\int_0^a z^{x-1}e^{-z}dz \leqslant \frac{a^x}{x}.$$

Stąd dla $t \geqslant 1$,

$$\left|\int_1^{\infty}\left(\frac{1}{w}\right)^{x+1}e^{-\frac{1}{w}}dw - \int_1^{t}\left(\frac{1}{w}\right)^{x+1}e^{-\frac{1}{w}}dw\right| = \int_t^{\infty}\left(\frac{1}{w}\right)^{x+1}e^{-\frac{1}{w}}dw =$$

$$= \int_0^{\frac{1}{t}} z^{x-1}e^{-z}dz \leqslant \frac{t^{-x}}{x}.$$

Dlatego w czasie $t \geqslant 1$ ($t_0 = 1$), wynik U': $\int_1^t \left[\left(\frac{1}{z}\right)^{x+1}e^{-\frac{1}{x}} + z^{x-1}e^{-z}\right]dz$ aproksymuje $\Gamma(x)$ z górnym ograniczeniem błędu równym:

$$\varepsilon(x,t) = \frac{t^{-x}}{x} + \frac{(x+2)^{(x+2)}}{t}.$$

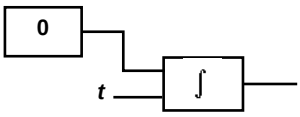
Funkcja $\varepsilon(x,t) \to 0$, gdy $t \to \infty$ oraz może być wygenerowana przez pewien P-GPAC U''. Konstruujemy obwód jak na rys. 2.5. Jego wynikiem jest x wejściowe ustawienie integratora. Zastępując jednostkę stałą generującą x w U' oraz U'' przez obwód rys. 2.5 oraz uruchamiając te obwody równolegle otrzymamy P-GPAC generujący funkcję Γ przez aproksymację.

□

Innym przykładem funkcji, która nie jest różniczkowalnie algebraiczna jest funkcja Zeta Riemmanna. Na osi rzeczywistej, dla $x > 1$ możemy zdefiniować ją następująco

$$\zeta(x) = \frac{1}{\Gamma(x)}\int_0^{\infty}\frac{z^{x-1}}{e^z - 1}dz.$$

Używając podobnych argumentów jak w przypadku funkcji Γ, możemy sformułować następujące twierdzenie.



Rysunek 2.5: Obwód GPAC, który dla parametru startowego integratora x daje wynik obwodu podczas całego obliczenia równy x

Twierdzenie 2.11 [26] *Funkcja ζ jest generowana przez aproksymację w modelu GPAC, na przedziale* $(1, +\infty)$.

W pracy [26] Graça stwierdza, że jeżeli funkcja jest obliczalna przez GPAC, to jest również obliczalna w modelu EAC, wprowadzonym przez Rubla i opisanym w kolejnym paragrafie, co sugeruje, że EAC może być modelem rozszerzającym możliwości GPACa.

2.3 EAC

Jako kolejny model obliczeń analogowych zaprezentujemy model rozszerzonego komputera analogowego (ang. *Extended Analog Computer*, EAC), który jest bardziej wszechstronnym modelem niż jego poprzednik GPAC. EAC jest matematycznym modelem wprowadzonym w 1993 roku przez L. A. Rubla w pracy [67], który do tej pory nie doczekał się pełnej fizycznej implementacji chociaż w MIT, J. Mils wraz z zespołem naukowców, prowadzi prace w tym kierunku [43].

2.3.1 Wprowadzenie

Model EAC, opisany w pracy [67], pracuje na poziomach tworzących hierarchię. W modelu tym nie mamy żadnych wejść, mamy tylko skończoną liczbę „ustawień" początkowych, które są dowolnymi liczbami rzeczywistymi (nie wymaga się aby były to liczby wymierne ani nawet w jakimkolwiek sensie obliczalne, np. przez maszynę Turinga). Na każdym poziomie znajdują się jednostki, które generują stałe rzeczywiste oraz niezależne zmienne $x_1, x_2, \ldots$. Wynik obliczeń na poziomie $n-1$ może być użyty jako dane wejściowe do obliczeń na poziomie n lub wyższych. Ogólnie, EAC na poziomie n generuje funkcje, które są budowane na bazie funkcji wygenerowanych na poziomach niższych poprzez następujące operacje: dodawanie, mnożenie, składanie, inwersję, różniczkowanie, znajdowanie analitycznego przedłużenie na

szerszy zbiór, rozwiązanie równania różniczkowego z warunkami brzegowymi czy znalezienie granicy po brzegu zbioru.

Wejściami i wynikami na poszczególnych poziomach są funkcje o skończonej liczbie niezależnych zmiennych $x_1, x_2, \ldots, x_n$, definiowane na pewnym zbiorze X. Zatem wynik obliczeń EACa będziemy zawsze zapisywać jako uporządkowana parę (f, X). Na najniższym poziomie 0, EAC generuje wielomiany o współczynnikach rzeczywistych i skończonej liczbie rzeczywistych zmiennych. Natomiast na poziomie 1 generowane są wszystkie funkcje różniczkowalnie algebraiczne.

Wszystkie funkcje generowane przez EAC są rzeczywistymi funkcjami analitycznymi. Klasę rzeczywistych funkcji analitycznych będziemy oznaczać przez C^ω i będziemy je rozumieć jako funkcje, które lokalnie możemy przedstawić w postaci sumy szeregu, który jest zbieżny. Zawsze, kiedy będziemy pisać $f \in C^\omega(X)$, będziemy mieli na myśli, że istnieje rozszerzenie funkcji $\tilde{f}$ na otwarty nadzbiór $\tilde{X}$ zbioru X, na którym $\tilde{f}$ jest rzeczywistą funkcją analityczną.

Poza funkcjami, EAC generuje również zbiory przestrzeni Euklidesowej. Zbiory są generowane na połówkach poziomów. Składają się one z elementów dziedziny pewnej funkcji f wygenerowanej na poprzednim poziomie, dających jej nieujemne lub dodatnie wartości. Ponadto na tym samym połówkowym poziomie EAC może wyznaczyć sumę i przecięcie tych zbiorów oraz ich rzuty na płaszczyznę wymiaru o jeden mniejszego.

2.3.2 Formalizm dla EAC

Przedstawimy teraz własny, matematyczny zapis definicji modelu EAC, podanej przez L. A. Rubla w pracy [67]. W tym celu zaprezentujemy uogólnioną definicję indukcyjnego domknięcia z paragrafu 2.1.1.

Definicja 2.24 [63, 12, 65] *Niech $\mathfrak{F}$ będzie rodziną klas elementów różnego typu, $\mathcal{F} \subseteq \mathfrak{F}$ rodziną zbiorów tych elementów, $\mathcal{O} \subseteq \bigcup_{k\in\mathbb{N}}\{O : \mathfrak{F}^k \to \mathfrak{F}\}$ zbiorem operatorów oraz funkcja $\sigma : \mathcal{O} \to \mathcal{S}^* \times \mathcal{S}; O \to (s_1, \ldots, s_n, s)$, gdzie $\mathcal{S}$ jest zbiorem nazw klas z rodziny $\mathfrak{F}$. Indukcyjnym domknięciem $\mathcal{F}$ dla $\mathcal{O}$, $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$, nazwiemy rodzinę najmniejszych zbiorów zawierających $\mathcal{F}_i, i \in \mathcal{S}$, taką że jeżeli $f_1, \ldots, f_k \in \mathcal{A}$ należą do dziedziny k-argumentowego operatora $O \in \mathcal{O}$, wówczas $O(f_1, \ldots, f_k) \in \mathcal{A}$. Razem z $\mathcal{O}$ indukcyjne domkniecie $[\mathcal{F}_1, \ldots, \mathcal{F}_n; \mathcal{O}]$ nazywamy algebrą wielosortową.*

Funkcja $\sigma(O) = (s_1, \ldots, s_n, s)$ podaje dla operatora O indeksy $s_1, \ldots, s_n$ klas, których elementy operator O przetwarza oraz jako ostatni indeks s klasy, której element operator O generuje.

Jeżeli w powyższej definicji rodzina $\mathfrak{F}$ będzie rodziną jednoelementową, to zakładając, że ten jedyny element $\mathfrak{F}$ jest klasą funkcji, otrzymamy przypadek indukcyjnego domknięcia dany definicją 2.1.

Rozważmy dwuelementową rodzinę $\mathfrak{F}$ gdzie $\mathfrak{F}_1$ będzie klasą funkcji $\mathbb{R}^n \to \mathbb{R}$, a $\mathfrak{F}_2$ klasą zbiorów z $\mathbb{R}^n$. Z terminem funkcji (w tym przypadku) będziemy kojarzyć zawsze parę (f, D_f), gdzie f jest funkcją zmiennej rzeczywistej klasy C^ω na zbiorze D_f, $D_f \subseteq \mathrm{Dom}(f)$, gdzie $\mathrm{Dom}(f)$ jest zbiorem tych argumentów $\bar{x} \in \mathbb{R}^n$, dla których wartość funkcji $f(\bar{x})$ jest określona. Zatem elementami klasy $\mathfrak{F}_1$ będą pary (funkcja, zbiór). Niech $\mathcal{F}_1 \subseteq \mathfrak{F}_1$ oraz $\mathcal{F}_2 \subseteq \mathfrak{F}_2$ będą zbiorami. Niech $X \neq \emptyset$ będzie skończonym zbiorem zmiennych $x_i : \mathbb{R} \to \mathbb{R}, i \in \{1, \ldots, k\}$, dokładniej zbiorem par $(x_i, \mathbb{R})$ i C skończonym zbiorem stałych rzeczywistych $c_i : \to \mathbb{R}, i \in \{1, \ldots, m\}$ (zbiór ten określa zachowanie modelu EAC), dokładniej zbiorem par $(c_i, \mathbb{R})$. Przyjmijmy $\mathcal{F}_1 = X \cup C$ oraz $\mathcal{F}_2 = \emptyset$.

Zdefiniujemy teraz zbiór dopuszczalnych w modelu EAC operatorów o dziedzinach w $\mathfrak{F}$. Wymagamy, aby wszystkie funkcje, które operator przyjmuje jako argument jak i wszystkie funkcje będące wynikiem działania operatora były klasy C^ω.

– dodawanie $(+)$: niech (f, D_f) i (g, D_g) będą k-argumentowymi funkcjami, wówczas operator dodawania zastosowany do tych funkcji generuje funkcję (h, D_h) k-argumentową, zdefiniowaną następująco:

$$h(\bar{x}) = +(f, g)(\bar{x}) = f(\bar{x}) + g(\bar{x}); \quad D_h = D_f \cap D_g;$$

$\sigma(+) = (1, 1, 1)$;

– mnożenie $(*)$: niech (f, D_f) i (g, D_g) będą k-argumentowymi funkcjami, wówczas operator mnożenia zastosowany do tych funkcji generuje funkcję h, D_h k-argumentową, zdefiniowaną następująco:

$$h(\bar{x}) = *(f, g)(\bar{x}) = f(\bar{x}) * g(\bar{x}); \quad D_h = D_f \cap D_g;$$

$\sigma(*) = (1, 1, 1)$;

– składanie (SK^l): niech $(g_1, D_{g_1}), (g_2, D_{g_2}), \ldots, (g_l, D_{g_l}), l > 0$ będą k-argumentowymi funkcjami oraz niech (f, D_f) będzie l-argumentową

funkcją, wówczas operator składania zastosowany do tych funkcji generuje funkcję (h, D_h) k-argumentową, zdefiniowaną następująco:

$$h(\bar{x}) = \mathrm{SK}^l(f, g_1, \ldots, g_l)(\bar{x}) = f(g_1(\bar{x}), \ldots, g_l(\bar{x}));$$

$D_h = \{\bar{x} \in D_{g_1} \cap \ldots \cap D_{g_l} : (g_1(\bar{x}), \ldots, g_l(\bar{x})) \in D_f\}$;
$\sigma(\mathrm{SK}^l) = (1, 1, \ldots, 1, 1)$;

– inwersja (INV): niech $(g_1, D_{g_1}), \ldots, (g_l, D_{g_l})$ będą $(l+k)$-argumentowymi funkcjami, wówczas operator inwersji zastosowany do tych funkcji generuje funkcję (h, D_h) k-argumentową, zdefiniowaną następująco:

$$h(\bar{x}) = \mathrm{INV}(g_1, \ldots, g_l)(\bar{x}, \bar{f}) = f_i(\bar{x}), \text{dla pewnego } i = 1, 2, \ldots, l,$$

gdzie $f_1(\bar{x}), \ldots, f_l(\bar{x})$[6] są rozwiązaniami układu równań:

$$\begin{cases} g_1(\bar{x}, f_1, f_2, \ldots, f_l) = 0 \\ g_2(\bar{x}, f_1, f_2, \ldots, f_l) = 0 \\ \vdots \\ g_l(\bar{x}, f_1, f_2, \ldots, f_l) = 0 \end{cases};$$

$\sigma(\mathrm{INV}) = (1, \ldots, 1, 1)$;

– pochodna cząstkowa (D): niech (g, D_g) będzie k-argumentową funkcją, wówczas operator pochodnej cząstkowej zastosowany do tej funkcji generuje funkcję (h, D_h) k-argumentową, zdefiniowaną następująco:

$$h(\bar{x}) = \mathrm{D}(g)(\bar{x}) = \frac{\partial^{\alpha_1+\alpha_2+\ldots+\alpha_k}}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \ldots \partial x_k^{\alpha_k}} g(\bar{x}), \quad D_h = D_g;$$

$\sigma(\mathrm{D}) = (1, 1)$;

– obcięcie (|): niech (g, D_g) będzie k-argumentową funkcją i niech $X \subseteq D_g$, wówczas operator obcięcia zastosowany do tej funkcji i zbioru X generuje funkcję (h, D_h) k-argumentową, zdefiniowaną następująco:

$$h(\bar{x}) = |(g, X)(\bar{x}) = g(\bar{x}); \quad D_h = X;$$

$\sigma(|) = (1, 2, 1)$;

[6]Wymaga się, aby te funkcje były dobrze zdefiniowanymi na X funkcjami klasy C^ω. Na przykład równanie $xy - 1 = 0$ ma rozwiązanie postaci $y = \frac{1}{x}$, które nie jest dobrze zdefiniowane na $\mathbb{R}$ (ponieważ nie jest zdefiniowana dla $x = 0$) ale jest dobrze zdefiniowane na przedziałach $(-\infty, 0)$ oraz $(0, \infty)$. Zatem $y = \frac{1}{x}$ nie jest EAC obliczalna na $\mathbb{R}$ ale jest EAC obliczalna na przedziale$(-\infty, 0)$ lub na przedziale $(0, \infty)$.

– przedłużenie ($\sim$): niech (g, D_g) będzie k-argumentową funkcją oraz niech X będzie zbiorem takim, że $X \cap D_g \neq \emptyset$, wówczas operator przedłużenia zastosowany do tej funkcji i zbioru X generuje funkcję (h, D_h), k-argumentową w taki sposób, że jest ona analitycznym przedłużeniem funkcji g na zbiór X, tj. funkcja $h(\bar{x}) = \sim (g, X)(\bar{x})$ jest klasy C^ω na X oraz $|(h, X \cap D_g) = |(g, X \cap D_g)$; $D_h = X$; $\sigma(\sim) = (1, 2, 1)$;

– równanie różniczkowe (RR): niech $(F_i, D_{F_i}), i = 1, \ldots, k$ będą $(k+l+1)$-argumentowymi funkcjami, (f_0, D_{f_0}) k-argumentową funkcją oraz niech X będzie pewnym zbiorem, wówczas operator równanie różniczkowe zastosowany do funkcji F_i oraz zbioru X generuje funkcję (h, D_h) jako rozwiązanie układu równań:

$$F_i(\bar{x} : f, f^{(\beta_1)}, f^{(\beta_2)}, \ldots, f^{(\beta_l)}) = 0,$$

dla $i = 1, \ldots, k$ na zbiorze X z warunkami brzegowymi f_0 dla $\gamma_0 \subset \partial X$ (∂X, brzeg X) [7], gdzie $f^{(\beta_1)}, f^{(\beta_2)}, \ldots, f^{(\beta_l)}$ są pochodnymi cząstkowymi funkcji f;

$$h(\bar{x}) = \mathrm{R}(f_0, F_1, \ldots, F_k, \gamma_0, X)(\bar{x}); \quad D_h = X;$$

$\sigma(\mathrm{RR}) = (1, 1, \ldots, 1, 2, 2, 1)$;

– granica (L_∂): niech $(g(\bar{x}), D_g)$ będzie k-argumentową funkcją, wówczas operator granicy zastosowany do tej funkcji generuje funkcję $(h(\bar{x}), D_h)$ k-argumentową, zdefiniowaną następująco:

$$h(\bar{x}_0) = L_\partial(g)(\bar{x}_0) = \lim_{\substack{\bar{x} \to \bar{x}_0 \\ \bar{x} \in D_g}} g(\bar{x}); \quad D_h = \partial D_g \text{ (brzeg } D_g)$$

oraz

$$\frac{\partial^{\alpha_1+\alpha_2+\ldots+\alpha_k}}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \ldots \partial x_k^{\alpha_k}} h(\bar{x}_0) = \lim_{\substack{\bar{x} \to \bar{x}_0 \\ \bar{x} \in D_g}} \frac{\partial^{\alpha_1+\alpha_2+\ldots+\alpha_k}}{\partial x_1^{\alpha_1} \partial x_2^{\alpha_2} \ldots \partial x_k^{\alpha_k}} g(\bar{x});$$

$\sigma(\mathrm{L}_\partial) = (1, 1)$;

[7] Na przykład: $f = f_0$ na części γ_0 brzegu X. Jeżeli operator RR jest używany na poziomie n, wówczas funkcja f_0 musi być generowana przez EAC na poziomie $n - 1$.

– wartości nieujemne ($\mathrm{G}_{\geqslant}$): niech (f, D_f) będzie k-argumentową funkcją, wówczas operator wartości nieujemne zastosowany do tej funkcji generuje zbiór A, zdefiniowany następująco:

$$A = \mathrm{G}_{\geqslant}(f) = \{\bar{x} \in D_f : f(\bar{x}) \geqslant 0\};$$

$\sigma(\mathrm{G}_{\geqslant}) = (1, 2)$;

– wartości dodatnie ($\mathrm{G}_{>}$): niech (f, D_f) będzie k-argumentową funkcją, wówczas operator wartości dodatnie zastosowany do tej funkcji generuje zbiór A, zdefiniowany następująco:

$$A = \mathrm{G}_{>}(f) = \{\bar{x} \in D_f : f(\bar{x}) > 0\};$$

$\sigma(\mathrm{G}_{>}) = (1, 2)$;

– alternatywa ($\cup$): niech A i B będą dwoma zbiorami w $\mathbb{R}^n$, wówczas operator alternatywy zastosowany do nich generuje zbiór C, zdefiniowany następująco:

$$C = \cup(A, B) = A \cup B;$$

$\sigma(\cup) = (2, 2, 2)$;

– koniunkcja ($\cap$): niech A i B będą dwoma zbiorami w $\mathbb{R}^n$, wówczas operator alternatywy zastosowany do nich generuje zbiór C, zdefiniowany następująco:

$$C = \cap(A, B) = A \cap B;$$

$\sigma(\cap) = (2, 2, 2)$;

– projekcja (P): niech A będzie zbiorem w $\mathbb{R}^n$, wówczas operator projekcji zastosowany do niego generuje zbiór B zawarty w $\mathbb{R}^{n-1}$, zdefiniowany następująco :

$$B = \mathrm{P}(A) = \{\bar{x} : (\exists x \in \mathbb{R})(x, \bar{x}) \in A\};$$

$\sigma(\mathrm{P}) = (2, 2)$;

Oto własna definicja EACa, wprowadzonego w pracy [67] przez Rubla, w terminach indukcyjnego domknięcia.

Definicja 2.25 *Klasą funkcji i zbiorów* EAC *obliczalnych* $\mathcal{EAC}$ *nazwiemy algebrę*

$$\mathcal{EAC} = [\mathcal{F}_1, \mathcal{F}_2; +, *, \mathrm{SK}^l, \mathrm{INV}, \mathrm{D}, |, \sim, \mathrm{RR}, \mathrm{L}_\partial, \mathrm{G}_>, \mathrm{G}_\geqslant, \cup, \cap, \mathrm{P}].$$

Zaprezentujemy teraz definicję klas $\mathcal{EAC}_n$, funkcji generowanych na poszczególnych poziomach EAC. Przyjmijmy

$$\mathcal{O}_f = \{+, *, \mathrm{SK}^l, \mathrm{INV}, \mathrm{D}, |, \sim, \mathrm{RR}, \mathrm{L}_\partial\}$$

zbiór operatorów generujących wartości ze zbioru $\mathcal{F}_1$, czyli funkcje. Przyjmijmy

$$\mathcal{O}_S = \{\mathrm{G}_>, \mathrm{G}_\geqslant, \cup, \cap, \mathrm{P}\}$$

zbiór operatorów generujących wartości ze zbioru $\mathcal{F}_2$, czyli zbiory.

Przyjmijmy następującą definicję:

Definicja 2.26 [63, 12, 65] *Niech* $\mathfrak{F}$ *będzie rodziną klas elementów różnego typu,* $\mathcal{F} \subseteq \mathfrak{F}$ *rodziną zbiorów tych elementów,* $\mathcal{O} \subseteq \bigcup_{k\in\mathbb{N}}\{O : \mathfrak{F}^k \to \mathfrak{F}\}$ *zbiorem operatorów oraz funkcja* $\sigma : \mathcal{O} \to \mathcal{S}^* \times \mathcal{S}; O \to (s_1, \ldots, s_n, s)$*, gdzie* $\mathcal{S}$ *jest zbiorem nazw klas z rodziny* $\mathfrak{F}$*. Przez* $\mathcal{A} = [\mathcal{F}; \mathcal{O}^1], i \in \mathbb{N}_0$ *oznaczymy rodzinę najmniejszych zbiorów zawierających* $\mathcal{F}_i, i \in \mathcal{S}$*, taką że jeżeli* $f_1, \ldots, f_k \in \mathcal{F}$ *należą do dziedziny k-argumentowego operatora* $O \in \mathcal{O}$*, wówczas* $O(f_1, \ldots, f_k) \in \mathcal{A}$*.*

Własna definicja klas funkcji generowanych przez EAC na poziomie n.

Definicja 2.27 *Klasę funkcji* EAC *obliczalnych na poziomie* 0, $\mathcal{EAC}_0$*, definiujemy następująco:* $\mathcal{EAC}_0 = [\mathcal{F}_1; +, \times]$*;*
Klasę zbiorów EAC *obliczalnych na poziomie* $\frac{1}{2}$*,* $\mathcal{EAC}_{\frac{1}{2}}$*, definiujemy następująco:* $\mathcal{EAC}_{\frac{1}{2}} = [\mathcal{EAC}_0, \mathcal{F}_2; \mathcal{O}_S]$*;*
Klasę funkcji EAC *obliczalnych na poziomie* $n+1$*,* $\mathcal{EAC}_{n+1}$*, definiujemy następująco:* $\mathcal{EAC}_{n+1} = [\mathcal{EAC}_n, \mathcal{EAC}_{n+\frac{1}{2}}; \mathcal{O}_f^1]$*;*
Klasę zbiorów EAC *obliczalnych na poziomie* $n+\frac{1}{2}, n > 0$*,* $\mathcal{EAC}_{n+\frac{1}{2}}$*, definiujemy następująco:* $\mathcal{EAC}_{n+\frac{1}{2}} = [\mathcal{EAC}_n, \mathcal{EAC}_{n-\frac{1}{2}}; \mathcal{O}_S]$*;*

Rysunek 2.6 przedstawia przykład modelu EAC na poziomie $n+1$, przy czym funkcje $g_1, \ldots, g_l$ nie koniecznie są różne od funkcji $f_1, \ldots, f_m$. Na tym

rysunku widzimy, że otrzymujemy pewne funkcje na poziomie n, a następnie na podstawie tych funkcji możemy wygenerować zbiory na poziomie $n+1/2$ poprzez pewne operacje O_S. Następnie te funkcje oraz zbióry mogą stanowić wejścia do modelu EAC na poziomie $n+1$ gdzie poprzez pewną operację O_f otrzymamy nową funkcję f jako wyjście EACa na poziomie $n+1$.

L. A. Rubel nałożył na EACa dodatkowe wymaganie, EWP (ang. *extremely well-posed*), aby generowane wyniki na zwartym zbiorze były bliskie oryginalnym wynikom, w przypadku gdy dane wejściowe nieco się różnią od początkowych ustawień. Oto definicja warunku EWP, który spełniają wszystkie generowane w modelu EAC funkcje.

Definicja 2.28 (EWP*: extremely well-posed*)[67] *Niech (f, Y) będzie wynikiem generowanym na poziomie n dla danych wejściowych $J_1, \ldots, J_n$ pochodzących z poziomu $(n-1)$. Niech $(\widetilde{f}, \widetilde{Y})$ będzie wynikiem generowanym na poziomie n dla danych wejściowych $J_1 + \lambda_1\varepsilon_1, \ldots, J_n + \lambda_n\varepsilon_n$, gdzie ε_i, $i = 1, \ldots, n$, są to funkcje będące testami dopuszczalnego błędu, takie że*

$$(\forall \bar{x} \in \mathbb{R}^k)(\forall \alpha \in \mathbb{N})(\forall p \in \mathbb{N})(\varepsilon_i^{(\alpha)}(\bar{x})[1 + \sum_{i=1}^{k} |x_i|^p] = O(1))$$

oraz $x_1, \ldots, x_k$ są to wszystkie niezależne zmienne używane i występujące na poziomie n.
Przez EWP *rozumiemy następujący warunek:*

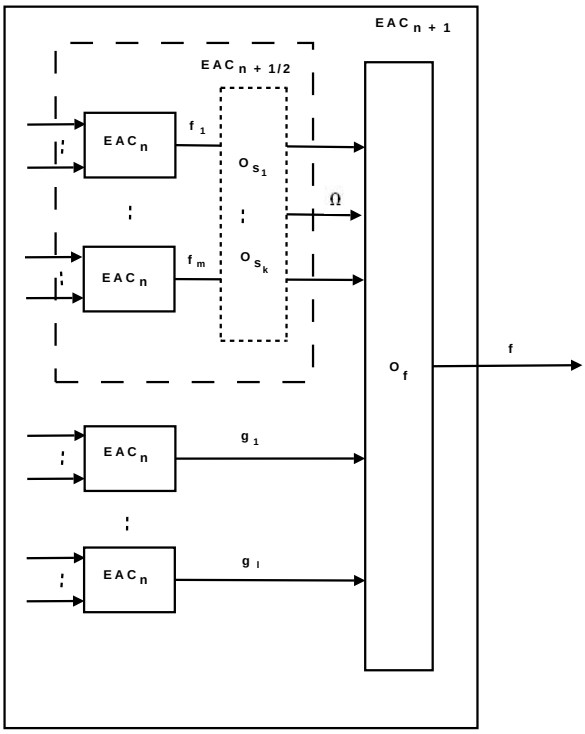
$$(\forall \varepsilon_1, \ldots, \varepsilon_n)(\forall K \subseteq Y)(\forall m \in \mathbb{N}_+)(\forall \delta > 0)(\exists \bar{\lambda}' \in \mathbb{R}_+^n)(\forall \bar{\lambda} \in \mathbb{R}^n)(|\lambda_i| \leqslant \lambda_i') \rightarrow$$

$$\rightarrow (\forall \bar{x} \in K)(\forall \bar{\beta} : \sum_{j=1}^{k} \beta_j \leqslant m, \bar{\beta} \in \mathbb{N}^k)|\widetilde{f}^{(\bar{\beta})}(\bar{x}) - f^{(\bar{\beta})}(\bar{x})| < \delta,$$

gdzie K jest zwartym podzbiorem Y[8].

Ten warunek implikuje $K \subseteq \widetilde{\Omega}$. To znaczy, że zawsze gdy J_i zależy tylko od podzbioru S_i liczb $\{1, 2, ..., k\}$, wówczas ε_i powinna zależeć tylko od zmiennych z S_i.

[8] $\varepsilon_i^{(\alpha)}$ oraz $y^{(\beta)}$ oznaczają odpowiednie pochodne cząstkowe.



Rysunek 2.6: Extended Analog Computer - model hierarchii poziomów

Zdefiniowany powyżej model EAC ma pewne interesujące własności. Zaprezentujemy teraz kilka pochodzących z pracy [67]. EAC potrafi wygenerować wszystkie funkcje, które generuje GPAC. Na poziomie 1 EAC generuje wszystkie funkcje algebraicznie różniczkowalne. Na wyższych poziomach EAC potrafi generować takie funkcje jak funkcja Gamma Eulera czy funkcja Zeta Riemmana. Wiemy również, że rozwiązanie problemu Dirichleta dla równania Laplace'a na kole jednostkowym może być otrzymane w modelu EAC.

Przykład 2.4 (funkcja Γ-Eulera) Przytoczymy definicję funkcji Gamma Eulera:

$$\Gamma(x) = \int_0^\infty t^x e^{-t} \frac{dt}{t}.$$

Aby pokazać, że EAC potrafi wygenerować tą funkcję wprowadzimy następującą funkcję:

$$\Gamma^*(x) = \int_1^\infty t^x e^{-t} \frac{dt}{t}.$$

Pokażemy poniżej, że EAC generuje funkcję $\Gamma^*(x)$. Pozostałą część $\int_0^1 t^x e^{-t}\frac{dt}{t}$ całki $\Gamma(x)$ możemy otrzymać w analogiczny sposób. Niech

$$f(x,y) = \int_{t=1}^{t=y} t^x e^{-t}\frac{dt}{t},$$

$$g(x,y) = f(x,\frac{1}{y}),$$

$$h(x) = \lim_{y\to 0} g(x,y).$$

Teraz pokażemy, że $h(x)$ jest EAC obliczalna. Ponieważ $t^{x-1}e^{-t}$ jest algebraicznie różniczkowalna zatem może być wygenerowana w modelu EAC. Dalej $g(x,y)$ jest rozwiązaniem następującego równania różniczkowego z warunkiem brzegowym:

$$g(x,1) = 0 \quad \text{dla wszystkich} \quad x > 0, \quad \partial_y g(x,y) = y^{x-1}e^{-y}(-\tfrac{1}{y^2}),$$

więc jest EAC obliczalną funkcją. Na podstawie definicji 2.25, możemy użyć operatora granicy L_∂ do wyznaczenia $h(x)$ w modelu EAC.

W podobny sposób można pokazać, że funkcja Zeta Riemmana może być wygenerowana w modelu EAC.

2.3.3 Dyskusja praktycznych zastosowań modelu typu EAC w odniesieniu do złożoności czasowej

W tej części zaprezentujemy krótko wyniki opublikowane przez J. W. Millsa, który na Uniwersytecie w Indianie, wraz z zespołem naukowców, prowadzi prace nad konstrukcją komputerów analogowych typu EAC. Wyniki te zaczerpnięte zostały z pracy [43]. Maszyna, którą przedstawił Mills w swoim artykule nie stanowi pełnej realizacji modelu EAC zaproponowanego przez Rubla, którego matematyczny opis został przedstawiony w poprzednim paragrafie. W tym paragrafie, dla rozróżnienia Rublowego EACa i EACa konstrukcji Millsa, ten pierwszy będziemy określać mianem modelu EAC, natomiast ten drugi, fizyczny model, po prostu skrótem EAC. Poza tym paragrafem wszędzie, gdzie będzie mowa o EACu, będziemy mieli na myśli model EAC zaproponowany przez Rubla. Obecnie EAC realizuje tylko niektóre operacje z założonych w teoretycznym modelu. Sam Rubel w swojej pracy [67]

stwierdził, że nie możemy jednoznacznie powiedzieć czy wszystkie z jego założeń będą kiedykolwiek mogły być zrealizowane. Wierzył on jednak, że jest to możliwe, ale potrzeba wielu lat prób i poszukiwań aby się o tym przekonać. Mills natomiast uważa, przekonany doświadczeniem, iż EAC już teraz można uznać za nowe osiągnięcie w dziedzinie obliczalności gdyż zmienia on zupełnie istotę obliczenia w porównaniu do klasycznego komputera. Na czym polega różnica pomiędzy fizyczną realizacją EACa a klasycznym komputerem? Odpowie na to pytanie dalsza część tego paragrafu wraz z przytoczonymi przykładami działania EACa.

W ogólności prezentowany przez Millsa EAC jest rodziną wyspecjalizowanych urządzeń, które obliczają określone funkcje poprzez analogię do naturalnych procesów zachodzących w przyrodzie (np. urządzenia przewodnictwa cieplnego, wibrujące struny i membrany, itp.).

Obliczenie EACa składa się z dwóch części:

– konfiguracji EACa, która wpływa na fizyczne własności operacji;

– interpretacji znaczenia przypisanego do ewolucji procesu fizycznego.

Obliczenie EACa bazuje na wykorzystaniu jakiegoś konkretnego procesu fizycznego, chemicznego, biologicznego, itp. Wyniki otrzymujemy zaś poprzez fakt analogii istniejącej pomiędzy rozwiązywanym przez nas problemem a danym procesem [9].

Przykład 2.5 (Morfogeneza skrzydła motyla) [43] EAC oblicza morfogenezę skrzydła motyla wykorzystując model równowagi pomiędzy aktywatorem i inhibitorem morfogenów (praca [51]). Dwie substancje chemiczne osiągają równowagę, gdy skrzydło rozwija się podczas morfogenezy. Matematyczny model morfogenezy jest opisany układem dwóch równań różniczkowych cząstkowych podanych przez Nijhout w pracy [51].

$$\begin{array}{c} \frac{\partial}{\partial t}a = \frac{ca^2}{i} - k_1 a + D_a \left(\frac{\partial^2}{\partial x^2}a + \frac{\partial^2}{\partial y^2}a\right) \\ \frac{\partial}{\partial t}i = ca^2 - k_2 i + D_i \left(\frac{\partial^2}{\partial x^2}i + \frac{\partial^2}{\partial y^2}i\right) \end{array} \tag{2.3}$$

Równania (2.3) definiują chemiczne współczynniki równowagi dwóch morfogenów: aktywator, jego stałą równowagi i tempo dyfuzji (a, k_1, D_a) oraz inhibitor, jego stałą równowagi i tempo dyfuzji (i, k_2, D_i). Gdy równowaga

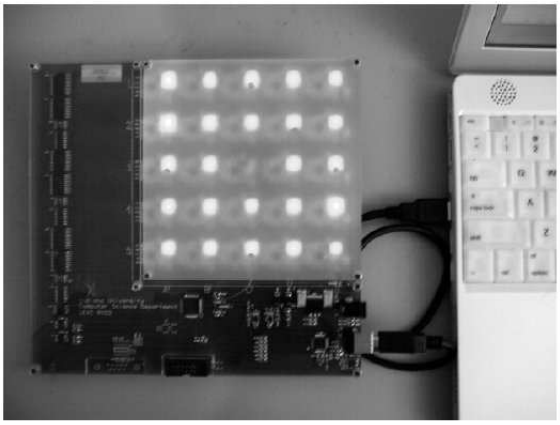
[9] Ten rodzaj obliczeń będziemy, krótko nazywać „obliczeniem przez analogię" lub jeszcze krócej „analogią".

zostanie osiągnięta w czasie morfogenezy, końcowa koncentracja aktywatora powoduje różną pigmentację rozwijających się łusek skrzydła, tworząc wzór na skrzydle motyla.

EAC w swoim działaniu nie wykorzystuje żadnego algorytmu rozwiązywania równań (2.3). Dwa morfogeny są implementowane przy pomocy elektronów, które występują w roli aktywatora a oraz dziur elektronowych, które pełnią rolę inhibitora chemicznego i. Stałe równowagi k_1, k_2 są implementowane dzięki prawu zachowania ładunku w arkuszu. Tempo dyfuzji poprzez ruch ładunku.

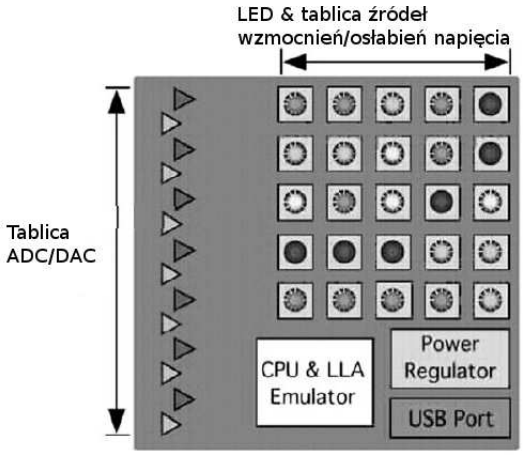
Zatem EAC jest urządzeniem, którego działanie nie opiera się o algorytm, jak w przypadku komputerów cyfrowych, ale polega na przeprowadzeniu pewnego procesu fizycznego, chemicznego, itp. Ważnym jest również aby zauważyć, iż jedna konfiguracja EACa może mieć wiele znaczeń. Na przykład ta sama konfiguracja EACa, niezmieniana, może obliczyć morfogenezą wzoru skrzydła motyla oraz pewien przypadek NP-zupełnego problemu cyklu Hamiltona.

Na ogół obliczenie przez analogię w EACu dobrze pasuje do wszystkich problemów szybkiego przetwarzania dużych ilości danych, rozpoznawania danych w czasie rzeczywistym lub tworzenia dużych modeli obliczeniowych, gdzie możemy tolerować pewną niedokładność obliczeń.



Rysunek 2.7: EAC R002 z pracy [43]

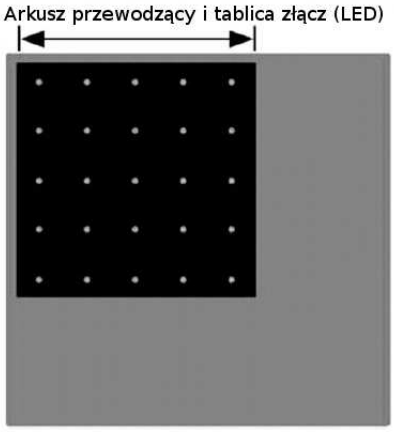
Przejdźmy do dokładniejszego omówienie przebiegu obliczenia w EACu. W tym celu, najpierw przybliżymy budowę maszyny EAC przedstawionej w pracy [43]. Wersja EAC R002 jest zbudowana z dwustronnego arkusza chronionego pleksą, realizującego proces obliczeniowy. Na jednej stronie arkusza liczącego umieszczona jest tablica 5×5 diod LED, stanowiących wejścia/wyjścia, są to programowalne źródła wzmocnień i osłabień natężenia prądu. Po lewej stronie arkusza znajdują się dwa konwertery, analogowo-cyfrowy (ADC) i cyfrowo-analogowy (DAC), na dole znajduje się mikroprocesor kontrolujący tablicę z diodami i USB łączącym z komputerem cyfrowym, rysunek 2.8. Pod tablicą z diodami po drugiej stronie arkusza znajduje się główny element liczący EACa, rysunek 2.9. Jest to arkusz elektrycznie przewodzącej pianki. Element ten wyznacza gradient napięcia rozwiązując problem o wiele rzędów szybciej niż metoda elementów skończonych rozwiązywania układów równań różniczkowych.



Rysunek 2.8: Wierzch arkusza liczącego EAC

Jak już wiemy, pierwszym etapem obliczeń EACa jest konfiguracja. Proces konfiguracji jest hybrydowy, odpowiada za niego algorytm ewolucyjny. Do EACa wysyłany jest ciąg poleceń z komputera cyfrowego ustalający konfigurację, punkty wejściowe i wynikowe (wszystkie punkty wynikowe połączone z arkuszem przewodzącym są stale mierzone). EAC zaczyna obliczenia w momencie, gdy zostanie włączony, w czasie gdy niesiony ładunek przepływa przez arkusz, licząc podczas procesu konfiguracji i kontynuując obliczenia

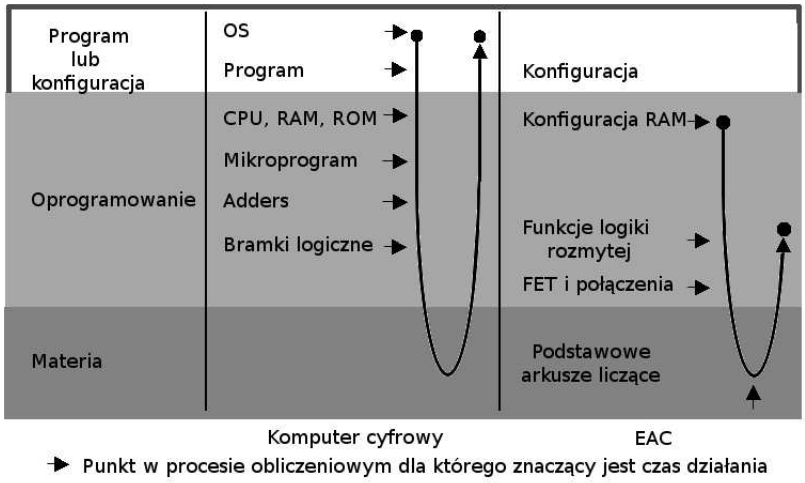
odpowiednio do określonej konfiguracji. Gdy już konfiguracja zostanie ustalona według preferencji użytkownika, określone obliczenie zajmuje milisekundy. Większość z tego czasu zajmuje komunikacja przez USB z komputerem cyfrowym, który wyświetla wyniki. Samo obliczenie EACa w arkuszu zajmuje mikrosekundy. Obliczenie kończy się gdy otrzymamy stabilną konfigurację. Po odczytaniu wyników, przekazywane one są do komputera cyfrowego, który modyfikuje bieżącą konfigurację. Na początku algorytm ustala losową konfigurację, wysyła ją do EACa, czyta wyjścia EACa i porównuje je z żądaną funkcją, następnie na podstawie błędu, który wykryje modyfikuje konfigurację EACa. Ten proces jest kontynuowany dopóki nie osiągniemy tolerowanego przez nas błędu. Ewolucyjna konfiguracja dla skomplikowanych problemów zajmuje tylko kilka sekund.



Rysunek 2.9: Spód arkusza liczącego EAC

Analogie wykorzystywane w procesie obliczeń EACa mają stosunkowo prostą ogólną strukturę. Może to pozwolić na rozwój klas złożoności, które byłyby zależne od modelu maszyny. Tym samym analogowe klasy złożoności moglibyśmy identyfikować z klasami problemów, które odwzorowują się na tą samą konfigurację EACa. Jest to podejście różniące się od istniejącej notacji złożoności, związanej z numeracją maszyn cyfrowych. W przypadku EACa musielibyśmy ponumerować analogie, które wykorzystywane są przez maszyny i są niezmienne względem szerokiej klasy problemów. Pojawiają się

pytania. Jak sklasyfikować analogie aby podać ich numerację? Czy analogowe klasy złożoności przypadków analogii są te same co klasy złożoności algorytmów?



Rysunek 2.10: Extended Analog Computer vs komputer cyfrowy

EAC, wykorzystujący pewną analogię do rozwiązania postawionego problemu, może w sposób niejawny mieć dostęp do dużej ilości obliczeń, na które ta analogia pozwala. Rozważania w tym temacie mogą dać wgląd w analogową złożoności. Jedyną widoczną formą specyfikowania analogii jest konfiguracja EACa, tj. ustalenie zbioru wejść i wyjść (bardziej złożone analogie mogą wymagać kilku EACów, gdy system okaże się zbyt skomplikowany aby „dopasować" go do jednego arkusza liczącego). Wydaje się, iż wiele z operacji modelu EAC, nie pojawia się fizycznie w EACu. Niektóre operacje, jak problem warunku brzegowego, możemy łatwo opisać przy pomocy widocznych komponentów EACa i jego arkusza liczącego. Natomiast próby znalezienia fizycznej analogii, np. dla analitycznego przedłużenia zakończyły się niepowodzeniem. Powodem tego jest różnica między „jawnymi" i „ukrytymi" funkcjami EACa.

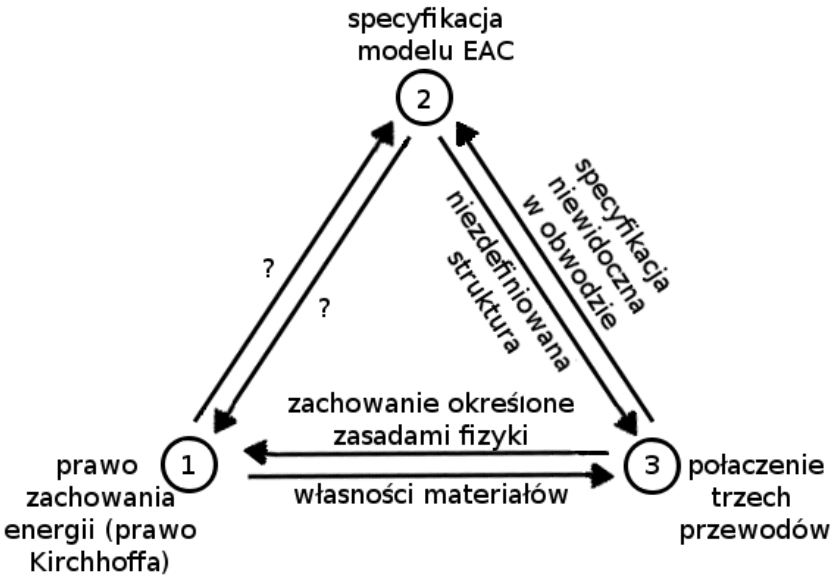
– Jawne funkcje są bezpośrednio implementowane poprzez dobranie odpowiedniej architektury, ich fizyczna struktura odpowiada ich definicji.

– Ukryte funkcje są z natury wyrażone we właściwościach i zasadach odnoszących się do materiału, z którego konstruowana jest architektura. Ich fizyczna struktura nie odpowiada strukturze ich definicji.

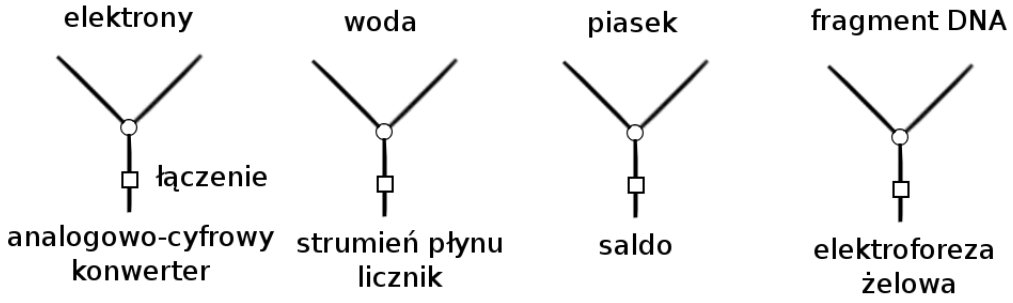
Przytoczymy dwa przykłady wyjaśniające tę różnicę, a tym samym wyjaśniające różnicę pomiędzy EACiem a komputerem cyfrowym. Najpierw przyjrzymy się konstrukcji jawnej funkcji dodawania dla komputerów cyfrowych. Jest ona specyfikowana przez wyrażenie logiki Boole'a. Definiuje ono fizyczną strukturę dodawania w terminach bramek logicznych. Na odwrót, z tej struktury możemy wywnioskować logiczną definicję dodawania. Dla porównania, dodawanie jest ukrytą funkcją w przypadku komputerów analogowych. Dodawanie może być zaimplementowane przy pomocy trzech połączonych przewodów, ale to nie odzwierciedla zapisu logiki Boole'a odpowiadającego tej funkcji. Jeśli pewne zdanie opisuje strukturę tego połączenia trzech przewodów, to nie opisuje ono fizyki tej operacji. Logika Boole'a jest zbyt odległa od fizyki. Operacja dodawania, bez binarnych bitów, jest otrzymywana przez prawo natury: prawo zachowania masy, energii, itp. W przypadku, gdy używamy do zdefiniowania dodawania transportu ładunków elektrycznych w przewodzie jest to prawo Kirchhoffa. Rysunek 2.11 wskazuje możliwość istnienia innych modeli połączonych trzech przewodów w roli dodawania (rysunek 2.12). Wiele takich ukrytych funkcji, wynikających z własności materiałów użytych do konstrukcji EACa, może być liczonych, pomimo że nie było to naszą intencją podczas konstrukcji maszyny. Parafrazując Richarda Feynmana [25], istnieje wiele tylnych pomieszczeń dla obliczeń EACa, o których nawet możemy nie wiedzieć.

Dla komputerów cyfrowych ważnym zagadnieniem w teorii złożoności są problemy NP-zupełne, czyli takie dla których nie znane jest wielomianowe rozwiązanie deterministyczne. Nie mamy dowodu, że P $\neq$ NP, ani że P = NP. Odniesienie złożoności obliczeniowej EACa do NP-zupełnych problemów nie było badane. Jednakże niektórzy badacze mają nadzieję, że analogowe komputery mogą dostarczyć odpowiedzi w tej kwestii [19, 5]. Ogólny pogląd utrzymuje, że P $\neq$ NP ale weźmy pod uwagę, że przez dekady wierzono, że nie ma komputera przekraczającego możliwości obliczeniowe maszyny Turinga. Dzisiejsze niekonwencjonalne modele, bazujące na pewnych dynamicznych układach fizycznych, wydają się w pewnych aspektach wykraczać poza model maszyny Turinga. (prace [19, 18, 43]). Ponieważ EAC liczy przez analogię, jest możliwe znalezienie nowej techniki zredukowania problemów NP-zupełnych do uniwersalnego NP-zupełnego problemu konfiguracji EACa.

Ta konfiguracja może okazać się wielomianowa dla wszystkich wystąpień tego NP-zupełnego problemu, wówczas otrzymamy dowód P = NP. Niestety, nic nam nie wiadomo o takim NP-zupełnym problemie, który mógłby być zastosowany przy tej technice. Mimo to, idea ta sugeruje pewną drogę zgłębienia wiedzy o złożoności obliczeniowej poprzez badanie przypadków analogii.



Rysunek 2.11: Ukryta funkcja dodawania (EAC)



Rysunek 2.12: Różne warianty sumatora (EAC)

Tymczasem użytecznym będzie przedstawić przegląd złożoności obliczenia EACa. Na początek zaznaczmy, że czas potrzebny na przygotowanie EACa, nawet dla dużych problemów, wynosi $O(n^2)$, gdzie n jest maksymalną liczbą punktów wejścia wzdłuż jednej krawędzi arkusza przewodzącego (lub arkuszy). Wymaga się aby w niezainicjowanym EACu wszystkie wejścia były najpierw rozłączone od arkusza piankowego procedurą o złożoności $O(n^2)$. Następnie zakładając, że wszystkie wyjścia są odczytywane, co jest najczęstszym przypadkiem, podobnie dostaniemy złożoność odczytania wyniku, $O(n^2)$. Jednak, użycie fizycznych procesów oraz proste odszyfrowania niekoniecznie prowadzą do rozwiązań o czasie wielomianowym problemów NP-zupełnych (chociaż EAC udowodnił niejednokrotnie, że może być wiele rzędów razy szybszy niż cyfrowy komputer). Powodem tego, że złożoność w takim przypadku może nie być wielomianowa, jest proces fizyczny potrzebujący wykładniczego czasu aby rozwiązać problem. Klasyczne fizyczne procesy (w przeciwieństwie do procesów kwantowych), niezależnie od przesłanek zasady minimalizacji energii (praca [1]), pozostają wykładnicze w czasie, wymuszając skalowanie obliczenia do dużego przypadku. Mimo to, ewolucyjne techniki użyte w EACu działają bardzo szybko dla małych lub umiarkowanie dużych problemów.

2.4 Rekurencyjne funkcje rzeczywiste

Kolejnym, zaprezentowanym przez nas modelem obliczeń będą rekurencyjne funkcje rzeczywiste. Klasa funkcji powstała jako rzeczywisty odpowiednik klasy funkcji rekurencyjnych dla liczb naturalnych. Z tym modelem obliczeń wiąże się zupełnie inny, niż w przypadku omówionych do tej pory modeli, rodzaj badań. Powodem tych badań jest, po pierwsze, chęć znalezienia matematycznego modelu obliczeń analogowych, po drugie, otrzymanie analogicznego, jak w przypadku liczb naturalnych, modelu charakteryzującego klasy złożoności. Jako pierwszy takie badania podjął C. Moore w 1996 roku. Podał on definicję funkcji $\mathbb{R}$-rekurencyjnych [47]. Funkcje te zostały określone na liczbach rzeczywistych, w sposób analogiczny jak klasa funkcji rekurencyjnych o dziedzinie w zbiorze liczb naturalnych, tak aby odpowiadały pojęciu analogowego komputera działającego w ciągłym czasie. Jest to nadspodziewanie duża klasa funkcji zawierająca wiele funkcji nieobliczalnych w tradycyjnym sensie (patrz paragraf 3.1).

2.4.1 Definicja

C. Moore w pracy [47] określił funkcję $f : \mathbb{R}^m \to \mathbb{R}^n$ jako $\mathbb{R}$-rekurencyjną, jeżeli jest to funkcja wygenerowana z rzeczywistych stałych 0 i 1 poprzez następujące operacje:

- złożenie: jeżeli f i g są funkcjami $\mathbb{R}$-rekurencyjnymi, wówczas $h(\bar{x}) = f(g(\bar{x}))$ jest funkcją $\mathbb{R}$-rekurencyjną;
- rekursja różniczkowa: jeżeli f i g są funkcjami $\mathbb{R}$-rekurencyjnymi, wówczas:
$$h(\bar{x}, 0) = f(\bar{x}),$$
$$h(\bar{x}, y) = g(\bar{x}, y, h(\bar{x}, y))$$
jest funkcją $\mathbb{R}$-rekurencyjną;
- μ-rekursja: jeżeli f jest funkcją $\mathbb{R}$-rekurencyjną, wówczas:
$$h(\bar{x}) = \mu_y f(\bar{x}, y) = \inf\{y : f(\bar{x}, y) = 0\}$$
jest funkcją $\mathbb{R}$-rekurencyjną;
- wektor funkcji jest zdefiniowany jako $\mathbb{R}$-rekurencyjny, jeżeli jego składowe są $\mathbb{R}$-rekurencyjne;

W μ-rekursji infimum wybiera takie y, dla którego funkcja f przyjmuje najmniejszą bezwzględną wartość. W stosunku do μ-rekursji określonej na liczbach naturalnych mamy dwie modyfikacje. Po pierwsze, oś rzeczywista została rozszerzona do dwóch wymiarów. Szukamy właściwego y zaczynając od zera na zewnątrz, szukając wśród y coraz bardziej oddalonych od 0. Jeżeli f przyjmuje tą samą wartość bezwzględną dla dwóch różnych wartości y o tej samej wartości bezwzględnej $|y|$, to wybierana jest umownie wartość ujemna. Po drugie, jeżeli powyżej y lub poniżej w przypadku wartości ujemnej y gromadzi się nieskończenie wiele zer funkcji f, to μ-rekursja zwraca y nawet jeżeli $f(\bar{x}, y) \neq 0$.

Ten sposób definiowania funkcji niesie ze sobą pewne problemy. Po pierwsze, rozwiązanie równania różniczkowego może być niejednoznaczne lub rozbieżne. Zatem będziemy mówić, że funkcja h jest zdefiniowana tylko wtedy, gdy istnieje skończone i jednoznaczne rozwiązanie równania różniczkowego. Skutkiem tego w klasie funkcji $\mathbb{R}$-rekurencyjnych mogą pojawić się funkcje częściowo niezdefiniowane.

Operacja rekursji różniczkowej może być realizowana przez analogowe urządzenia liczące, np. może być implementowana w modelu GPAC (paragraf 2.2). C. Moore, w pracy [47], identyfikuje podklasę funkcji $\mathbb{R}$-rekurencyjnych definiowanych bez μ-rekursji z klasą funkcji generowanych przez GPAC. Robi to jednak w sposób niewłaściwy. Zawodzi również podana przez C. Moora konstrukcja rozwiązania problemu stopu z klasycznej teorii obliczeń. Dzieje się tak dlatego, że operacja μ-rekursji zapożyczona z klasycznej teorii obliczeni nie pasuje do analitycznej rzeczywistości obliczeń analogowych. W pracy [17] zaproponowano w jej miejsce bardziej użyteczną operację z punktu widzenia analizy, operację granicy nieskończonej, w celu ominięcia problemów jakich dostarczała oryginalna definicja Moora.

Podamy teraz definicję rekurencyjnych funkcji rzeczywistych, wyprowadzoną z oryginalnej definicji Moora, pochodzącą z [16]. Zanim to zrobimy przytoczymy kilka definicji i twierdzeń pozwalających nam lepiej zdefiniować operację rekursji różniczkowej. Niech f będzie całkowitą funkcją działającą z $\mathbb{R}^{n+1}$ w $\mathbb{R}^n$ oraz niech I będzie (prawdopodobnie nieograniczonym) otwartym przedziałem (A, B) z $t_0 \in I$. Rozważmy problem Cauchy'ego postaci:

$$g(t_0) = g_0, \quad \partial_t g(t) = f(t, g(t)). \tag{2.4}$$

Rozwiązanie (2.4) na przedziale I jest funkcją ciągłą $g : \mathbb{R} \to \mathbb{R}^n$ przyjmującą wartość g_0 dla $t = t_0$ i spełniająca równanie różniczkowe dla wszystkich $t \in I$. Jeżeli funkcja f spełnia kilka prostych własności, wówczas mamy gwarancję, że rozwiązanie (2.4) istnieje i jest jednoznaczne.

Definicja 2.29 [24, 73] *Całkowita funkcja $f : \mathbb{R}^m \to \mathbb{R}^n$ spełnia lokalnie warunek Lipschitza, jeżeli dla każdego zwartego zbioru $C \subset \mathbb{R}^m$ istnieje stała K, taka że dla dowolnych $\bar{x}, \bar{y} \in C$ spełniony jest następujący warunek:*

$$\|f(\bar{x}) - f(\bar{y})\| \leqslant K\|\bar{x} - \bar{y}\| \tag{2.5}$$

Najmniejsze takie K nazywamy stałą Lipschitza funkcji f na C.

Przytoczymy teraz kilka ważnych dla nas własności funkcji spełniających lokalnie warunek Lipschitza.

Twierdzenie 2.12 [24, 73] (*Picard*) *Jeżeli f spełnia lokalnie warunek Lipschitza, wówczas w pewnym otoczeniu punktu t_0 istnieje rozwiązanie g równania* (2.4).

Twierdzenie 2.13 [24, 73] *Jeżeli f spełnia lokalnie warunek Lipschitza oraz g jest rozwiązaniem równania* (2.4) *na przedziale I, wówczas g jest jednoznacznym rozwiązaniem równania* (2.4) *na przedziale I.*

Z twierdzenia o jednoznaczności rozwiązania wynika następująca własność:

Własność 2.2 [24, 73] *Jeżeli f spełnia lokalnie warunek Lipschitza oraz g i $\tilde{g}$ sa dwoma rozwiązaniami równania* (2.4) *odpowiednio na przedziałach I i $\tilde{I}$, wówczas $g = \tilde{g}$ na przedziale $I \cap \tilde{I}$.*

Niech S będzie zbiorem rozwiązań równania (2.4) dla pewnej funkcji f spełniającej lokalnie warunek Lipschitza. Przyjmijmy A jako infimum z $\mathrm{Dom}(g)$ dla wszystkich $g \in S$ oraz B jako supremum z $\mathrm{Dom}(g)$ dla wszystkich $g \in S$. Wówczas powyższa własność prowadzi do następującej definicji.

Definicja 2.30 [24, 73] *Niech f spełnia lokalnie warunek Lipschitza. Maksymalnym rozwiązaniem równania* (2.4) *jest funkcja g zdefiniowana na przedziale (A, B), taka, że jeżeli $\tilde{g}$ jest rozwiązaniem równania* (2.4) *na pewnym przedziale (a, b), wówczas $g(t) = \tilde{g}(t)$ dla wszystkich $t \in (a, b)$. Przedział (A, B) nazywamy wówczas maksymalnym przedziałem równania* (2.4).

Następne twierdzenie mówi o tym, że takie rozwiązanie równania (2.4) jest dobrze zdefiniowane w A i B.

Twierdzenie 2.14 [24, 73] *Niech f spełnia lokalnie warunek Lipschitza oraz niech g będzie maksymalnym rozwiązaniem równania* (2.4) *zdefiniowanym na przedziale $J = (A, B)$. Wówczas $B < +\infty$ (lub $A > -\infty$) wtedy i tylko wtedy, gdy $\lim_{t \to \tilde{t}^-} \|g(t)\| = +\infty$ (odpowiednio $\lim_{t \to \tilde{t}^+} \|g(t)\| = +\infty$) dla pewnego $t \in \mathbb{R}$ w tym przypadku mamy $B = \tilde{t}$ (odpowiednio $A = \tilde{t}$).*

Przejdźmy teraz do ponownego zdefiniowania rekurencyjnych funkcji rzeczywistych. Oznaczenia dla funkcji stałych: $-1^n, 0^n, 1^n$, zdefiniowanych następująco: $-1^n(x_1, \ldots, x_n) = -1, 0^n(x_1, \ldots, x_n) = 0$ oraz $1^n(x_1, \ldots, x_n) = 1$. Projekcji rzeczywistej: $\mathfrak{i}_i^n$ dla wszystkich $n = 1, 2, \ldots$ oraz $1 \leqslant i \leqslant n$, zdefiniowanej analogicznie jak w przypadku funkcji określonych na zbiorze liczb naturalnych $\mathbb{N}$, $\mathfrak{i}_i^n(x_1, \ldots, x_m) = x_i$. Klasa rekurencyjnych funkcji rzeczywistych jest domknięta ze względu na następujący zbiór operacji $\mathcal{F}$:

– składanie (SK): niech $f : \mathbb{R}^k \to \mathbb{R}^n$ i $g : \mathbb{R}^m \to \mathbb{R}^k$, wówczas operator składania zastosowany do tych funkcji generuje funkcję $h : \mathbb{R}^m \to \mathbb{R}^n$ następująco:

$$h(\bar{x}) = \mathrm{SK}(f, g)(\bar{x}) = f(g(\bar{x})).$$

Dziedziną $\mathrm{SK}(f, g)$ jest $\mathrm{Dom}(\mathrm{SK}(f, g)) = \{\bar{x} \in \mathrm{Dom}(g) : g(\bar{x}) \in \mathrm{Dom}(f)\}$;

– rekursja różniczkowa (R): niech $f : \mathbb{R}^{n+1} \to \mathbb{R}$ będzie całkowitą funkcją spełniającą lokalnie warunek Lipschitza. Rozważmy dla ustalonego $\bar{x} \in \mathbb{R}^m$ problem Cauchy'ego

$$g(\bar{x}, 0) = \bar{x} \quad \partial_t g(\bar{x}, t) = f(t, g(\bar{x}, t)). \tag{2.6}$$

Operator rekursji różniczkowej zastosowany do funkcji f generuje funkcję $h : \mathbb{R}^{n+1} \to \mathbb{R}$ taką, że dla dowolnego ustalonego $\bar{x} \in \mathbb{R}^n$,

$$h(\bar{x}, t) = \mathrm{R}(f)(\bar{x}, t) = g(\bar{x}, t),$$

gdzie $g(\bar{x}, \cdot)$ jest maksymalnym rozwiązaniem (2.6). Dziedziną $\mathrm{R}(f)$ jest $\mathrm{Dom}(\mathrm{R}(f)) = \{(\bar{x}, t) : \bar{x} \in \mathbb{R}^n, A(\bar{x}) < t < B(\bar{x})\}$, gdzie A, B dają ekstrema maksymalnego przedziału;

– nieskończona granica górna (LS): niech $f : \mathbb{R}^{m+1} \to \mathbb{R}^n$, wówczas operator nieskończonej granicy górnej zastosowany do funkcji f generuje funkcję $h : \mathbb{R}^m \to \mathbb{R}^n$ określoną następująco: dla każdego $i = 1, \ldots, n$,

$$(h(\bar{x}))_i = (\mathrm{LS}(f)(\bar{x}))_i = \limsup_{y \to \infty} (f(\bar{x}, y))_i$$

lub w skrócie

$$h(\bar{x}) = \mathrm{LS}(f)(\bar{x}) = \limsup_{y \to \infty} f(\bar{x}, t).$$

Dziedziną $\mathrm{LS}(f)$ jest $\mathrm{Dom}(\mathrm{LS}(f)) = \{\bar{x} \in \mathbb{R}^m : \limsup_{y \to \infty} f(\bar{x}, y)$ istnieje$\}$;

– agregacja (V): niech $f : \mathbb{R}^m \to \mathbb{R}^k$ oraz $g : \mathbb{R}^m \to \mathbb{R}^n$, wówczas operacja agregacji zastosowana do funkcji f i g generuje funkcję $h : \mathbb{R}^m \to \mathbb{R}^{k+n}$ następująco:

$$h(\bar{x}) = \mathrm{V}(f, g)(\bar{x}) = (f(\bar{x}), g(\bar{x})).$$

Dziedziną $\mathrm{V}(f, g)$ jest $\mathrm{Dom}(\mathrm{V}(f, g)) = \mathrm{Dom}(f) \cap \mathrm{Dom}(g)$.

Oto zapowiadana definicja rekurencyjnych funkcji rzeczywistych.

Definicja 2.31 [16] *Klasą rekurencyjnych funkcji rzeczywistych* ($\mathcal{REC}(\mathbb{R})$) *zdefiniowanych na* $\mathbb{R}$ *nazwiemy algebrę*

$$\mathcal{REC}(\mathbb{R}) = [-1^n, 0^n, 1^n, \mathfrak{i}_i^n; \mathrm{SK}, \mathrm{R}, \mathrm{LS}, \mathrm{V}].$$

Operator rekursji różniczkowej jest próbą naśladowania zachowania integratora skonstruowanego w XIX wieku przez lorda Kelvina i używanego do implementacji znanego analizatora różniczkowego V. Busha. Jak przedstawiono powyżej, R(f) jest zdefiniowana tylko w przypadku gdy f spełnia lokalnie warunek Lipschitza. Z czego wynika, że gdy $f \in \mathrm{Dom(R)}$, wówczas rozwiązanie istnieje (twierdzenie 2.12), jest jednoznaczne (twierdzenie 2.13) oraz dobrze zdefiniowane w punktach ekstremalnych dziedziny określoności (twierdzenie 2.14). Można użyć słabszych warunków definiowalności operacji rekursji różniczkowej i otrzymać podobne własności rozwiązania lecz autorzy pracy [16] sugerują, że mogłyby się one okazać równie trudne technicznie i złożone. Pomocne w określaniu czy dana funkcja jest w dziedzinie operatora R jest następujące twierdzenie:

Twierdzenie 2.15 [24, 73] *Niech* $f : \mathbb{R}^m \to \mathbb{R}^n$ *będzie funkcją całkowitą.*

(a) Funkcja f spełnia lokalnie warunek Lipschitza wtedy i tylko wtedy, gdy każda domknięta kula satysfakcjonuje (2.5)*.*

(b) Funkcja f spełnia lokalnie warunek Lipschitza wtedy i tylko wtedy, gdy każdy domknięty m-wymiarowy sześcian satysfakcjonuje (2.5)*.*

(c) Jeżeli f jest wszędzie różniczkowalna oraz Df jest ograniczona na każdym zbiorze zwartym, wówczas f spełnia lokalnie warunek Lipschitza.

(d) Jeżeli $f \in C^1$*, wówczas f spełnia lokalnie warunek Lipschitza* [10]*.*

Definicja rozwiązania równania (2.4) zapewnia nam, że $R(f)$ zawsze będzie klasy C^1 ze względu na ostatnią zmienną, tj. $R(f)(\bar{x}, \cdot) \in C^1$ dla każdego ustalonego $\bar{x}$. Ponadto mamy następującą silniejszą własność.

[10] Przez C^1 rozumiemy klasę funkcji ciągłych, posiadających ciągłe pochodne w całej swojej dziedzinie.

Twierdzenie 2.16 [24, 73] *Dla każdej funkcji $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$ spełniającej lokalnie warunek Lipschitza, $g = \mathrm{R}(f)$ spełnia lokalnie warunek Lipschitza w swojej dziedzinie.*

Przez stwierdzenie, że funkcja g spełnia lokalnie warunek Lipschitza w swojej dziedzinie rozumiemy, iż każdy zbiór zwarty $C \in \mathrm{Dom}(g)$ musi spełniać warunek Lipschitza.

Również określenie, które funkcje należą do dziedziny operatora LS może sprawić kłopot. Przytoczymy tutaj kilka spostrzeżeń dotyczących tego zagadnienia pochodzących z pracy [16].

Uwaga 2.1 [16] *Weźmy ustalony punkt $\bar{x} \in \mathbb{R}^m$. Jeżeli $f(\bar{x}, y)$ jest niezdefiniowana dla pewnego dużego y, wówczas* $\mathrm{LS}(f)(\bar{x})$ *będzie niezdefiniowana, tj. $\bar{x} \notin \mathrm{Dom}(\mathrm{LS}(f))$.*

Uwaga 2.2 [16] *Weźmy ustalony punkt $\bar{x} \in \mathbb{R}^m$. Jeżeli co najmniej jeden komponent $f(\bar{x}, y)$ jest niezdefiniowana dla pewnego dużego y, wówczas* $\mathrm{LS}(f)(\bar{x})$ *będzie niezdefiniowana, tj. $\bar{x} \notin \mathrm{Dom}(\mathrm{LS}(f))$.*

Uwaga 2.3 [16] *Weźmy ustalony punkt $\bar{x} \in \mathbb{R}^m$. Jeżeli co najmniej jedna z $\limsup_{y\to\infty}(f(\bar{x}, y))_i$ jest niezdefiniowana dla pewnego dużego y, wówczas* $\mathrm{LS}(f)(\bar{x})$ *będzie niezdefiniowana, tj. $\bar{x} \notin \mathrm{Dom}(\mathrm{LS}(f))$.*

Poniższe dwa twierdzenia zaczerpnięte z pracy [16] podają przykłady funkcji z klasy $\mathcal{REC}(\mathbb{R})$.

Twierdzenie 2.17 [16] *Funkcje $+, \times, -, \exp, \sin, \cos, \ln, x^y, \operatorname{arc\,tg}$ są rekurencyjnymi funkcjami rzeczywistymi. Funkcje $\frac{1}{x}, \frac{x}{y}, \sqrt{x}$ są rekurencyjnymi funkcjami rzeczywistymi na przedziale $(0, +\infty)$.*

Dowód. Funkcje te możemy zdefiniować, używając wyłącznie operacji składania, rekursji różniczkowej i agregacji w następujący sposób: Dodawanie, odejmowanie, mnożenie, funkcję eksponencjalną, sinus, cosinus, logarytm, arc-tangens definiujemy przy pomocy operacji rekursji prostej, w następujący sposób:

– dodawanie

$$+(x, 0) = x, \quad \partial_y + (x, y) = 1^2(y, +(x, y)) = 1;$$

– odejmowanie

$$-(x,0) = x, \quad \partial_y - (x,y) = -1^2(y, +(x,y)) = -1;$$

– dla mnożenia mamy:

$$g(x_1,x_2,0) = (x_1,x_2), \quad \partial_y g(x_1,x_2,y) = \mathrm{V}(\mathfrak{i}_3^3, 0^3) = ((g(x_1,x_2,y))_2, 0),$$

wówczas $g(x_1,x_2,y) = (x_1 + x_2 y, x_2)$ jest rozwiązaniem, zatem

$$\times(x,y) = \mathrm{SK}(\mathfrak{i}_1^2, \mathrm{SK}(g, \mathrm{V}(0^2, \mathrm{V}(\mathfrak{i}_1^2, \mathfrak{i}_2^2))))(x,y) = (g(0,x,y))_1;$$

– funkcja eksponencjalna

$$\exp(0) = 1, \quad \partial_x \exp(x) = \exp(x),$$

stosując notację jak w (2.6) mamy: $g_0 = 1$ i $f(t,x) = x$;

– funkcje $\sin(x), \cos(x)$ możemy otrzymać jako rozwiązanie równania:

$$g(0) = (0,1), \quad \partial_y g(y) = (g_2(y), -g_1(y)).$$

W notacji (2.6) mamy: $g_0 = (0,1)$ i $f(t,\bar{z}) = ((\bar{z})_2, -(\bar{z})_1)$. Rozwiązaniem jest $g(x) = (\sin(x), \cos(x))$;

– funkcję $1/x$ definiujemy następująco:

$$\frac{1}{1} = 1, \quad \partial_x \frac{1}{x} = -1 \times \left(\frac{1}{x}\right) \times \left(\frac{1}{x}\right) = -\frac{1}{x^2};$$

– logarytm

$$\ln(1) = 1, \quad \partial_x \ln(x) = \frac{1}{x};$$

– arcus tangens

$$\operatorname{arc\,tg}(0) = 0, \quad \partial_x \operatorname{arc\,tg}(x) = \frac{1}{x^2+1};$$

Pozostałe funkcje możemy zdefiniować przy pomocy operacji składania:

$\frac{x}{y} = x \times \frac{1}{y}$;

$x^y = \exp(\ln(x) \times y)$ dla $x > 0$;

$\sqrt{x} = x^{\frac{1}{2}}$ dla $x \geqslant 0$;

□

Twierdzenie 2.18 [16] *Delta Kroneckera δ, funkcja Heavisidea Θ zdefiniowane następująco:*

$$\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases}, \quad \Theta(x) = \begin{cases} 1 & x \geqslant 0 \\ 0 & x < 0 \end{cases}$$

oraz funkcja podłogi[11] $\lfloor\rfloor$ *są rzeczywistymi funkcjami rekurencyjnymi.*

Dowód. Mamy:

$$\delta(x) = \limsup_{y\to\infty} \left(\frac{1}{x^2+1}\right)^y,$$

$$\Theta(x) = \left(\limsup_{y\to\infty} \frac{1}{1+2^{-xy}}\right) + \frac{1}{2}\delta(x)$$

oraz

$$\lfloor x \rfloor = x - r(-x),$$

gdzie $r(x) = s(x)\tilde{r}(x+1) + (1-s(x))\tilde{r}(x)$,
natomiast $\tilde{r}(x)$ jest funkcję definiowaną poprzez operator rekursji różniczkowej:

$$\tilde{r}(0) = 0, \quad \partial_x \tilde{r}(x) = 2\sin(\pi x)^2 s(x) - \frac{1}{2}$$

oraz $s(x) = \Theta(\sin(\pi x))$.

□

Twierdzenie 2.19 [16] *Funkcje charakterystyczne równości $c_=$, nierówności $c_\leqslant$ oraz nierówności właściwej $c_<$ są rzeczywistymi funkcjami rekurencyjnymi.*

Dowód. Mamy: $c_=(x,y) = \delta(y-x), c_\leqslant(x,y) = \Theta(y-x)$ oraz $c_<(x,y) = c_\leqslant(x,y) - c_=(x,y)$.

□

Dodajmy, że przez rekurencyjne liczby rzeczywiste obliczalne będziemy rozumieć wartości pewnych rekurencyjnych funkcji rzeczywistych wyznaczane dla argumentu 0. Oczywiście wartość obliczoną dla pewnego argumentu x_0 możemy uznać za obliczalną w tym sensie, składając daną funkcję z funkcją $-(x_0, x)$. Stąd e, π są rekurencyjnymi liczbami rzeczywistymi obliczalnymi: $e = \exp(1)$, $\pi = 4 \times \operatorname{arc\,tg}(1)$.

[11] $\lfloor x \rfloor = \max\{n \in \mathbb{Z} : n \leqslant x\}$

2.4.2 Podstawowe własności

Zaczniemy ten paragraf od przedstawienia notacji opisu (deskrypcji) rekurencyjnych funkcji rzeczywistych.

Definicja 2.32 [16] *Niech $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$ będzie przeliczalną algebrą funkcji dla zbioru $\mathcal{F} = \{f_1, f_2, \ldots\}$ funkcji i zbioru $\mathcal{O} = \{O_1, O_2, \ldots\}$ operatorów. Zbiór $D_{\mathcal{A}}$ deskrypcji algebry $\mathcal{A}$ jest najmniejszym zbiorem słów złożonych z symboli $\{fun, Op, \langle, \rangle, ,, 0, 1, \ldots, 9\}$, takich że:*

(i) $\langle fun, n\rangle \in D_{\mathcal{A}}$ *dla wszystkich* n *oraz*

(ii) jeżeli $d_1, \ldots, d_k \in D_{\mathcal{A}}$, *wówczas* $\langle Op, n, d_1, \ldots, d_k\rangle \in D_{\mathcal{A}}$.

Niech $\mathcal{D}_{\mathcal{A}}$ oznacza zbiór dobrych deskrypcji w $D_{\mathcal{A}}$. Poprzez dobrą deskrypcję rozumiemy deskrypcję d taką, że

(i) d *jest* $\langle fun, n\rangle$ *oraz* $f_n \in \mathcal{F}$ *(tj.* $\mathcal{F}$ *zawiera co najmniej* n *funkcji), w tym przypadku* d *jest deskrypcją* f_n *lub*

(ii) d *jest* $\langle Op, n, d_1, \ldots, d_k\rangle$ *dla pewnych dobrych deskrypcji* $d_1, \ldots, d_k$, *które opisują* $g_1, \ldots, g_k$ *z dziedziny operatora* $O_n \in \mathcal{O}$, *wówczas* d *jest deskrypcją* $O(g_1, \ldots, g_k)$.

Powyższa definicja ma zastosowanie do wszystkich klas funkcji definiowanych przy pomocy indukcyjnego domknięcia. Jeżeli przyjmiemy, że $\mathcal{H}$ jest algebrą funkcji dla $\mathcal{REC}(\mathbb{R})$ to $\mathcal{D}_{\mathcal{H}}$ będzie zbiorem dobrych deskrypcji dla rekurencyjnych funkcji rzeczywistych.

Rozważając algebry funkcji chcielibyśmy wiedzieć, które operatory dodane do naszej algebry faktycznie rozszerzą nam zbiór funkcji należących do algebry, a które nie. Innymi słowy, mając dany operator $O : \mathfrak{F}^k \to \mathfrak{F}$ chcielibyśmy wiedzieć czy dana algebra $\mathcal{A}$ jest zamknięta ze względu na operator O czy nie. Wiele ważnych, z punktu informatyki, problemów może być równoważnych z faktem czy pewna algebra jest zamknięta ze względu na dany operator. Przykładem może być ważny problem dotyczący klasy $\mathcal{PRF}$, $P = NP$ wtedy i tylko wtedy, gdy P jest zamknięta ze względy na operator minimalizacji μ ($P = [P; \mu]$). Przytoczymy teraz definicję efektywnego zamknięcia algebry funkcji ze względu na dany operator.

Definicja 2.33 [16] *Mówimy, że przeliczalna algebra funkcji $\mathcal{A}$ jest zamknięta ze względu na operator $O : \mathfrak{F}^k \to \mathfrak{F}$, jeżeli istnieje efektywna procedura, która dla danych dobrych deskrypcji funkcji $f_1, \ldots, f_k \in \mathcal{A}$ będących w dziedzinie operatora O otrzyma dobrą deskrypcje funkcji $O(f_1, \ldots, f_k)$.*

Podamy teraz przykład dwóch operatorów, dla których klasa $\mathcal{REC}(\mathbb{R})$ jest efektywnie zamknięta.

Definicja 2.34 [16] *Operator granicy dolnej nieskończonej* (LI) *oraz operator granicy nieskończonej* (L) *jest określony następująco:*

$$\mathrm{LI}(f) = \liminf_{y\to\infty} f(\bar{x}, y), \quad \mathrm{L}(f) = \lim_{y\to\infty} f(\bar{x}, y);$$

gdzie $f : \mathbb{R}^{m+1} \to \mathbb{R}^n$ należy do $\mathfrak{F}$.

Twierdzenie 2.20 ([49]) *$\mathcal{REC}(\mathbb{R})$ jest efektywnie zamknięta ze względu na* LI *i* L.

Dowód. Mamy $\mathrm{LI}(f)(\bar{x}) = -\limsup_{y\to\infty} -f(\bar{x}, y)$.
Wiemy, że $\lim_{y\to\infty} f(\bar{x}, y)$ jest zdefiniowana wtedy i tylko wtedy, gdy obie granice $\limsup_{y\to\infty} f(\bar{x}, y)$ i $\liminf_{y\to\infty} f(\bar{x}, y)$ są zdefiniowane i sobie równe. Wtedy mamy $\lim_{y\to\infty} f(\bar{x}) = \limsup_{y\to\infty} f(\bar{x}, y)$. Zatem możemy operator L zdefiniować następująco:

$$\mathrm{L}(f)(\bar{x}) = \frac{1}{c_=(\limsup_{y\to\infty} \sigma(f(\bar{x}, y)), \liminf_{y\to\infty} \sigma(f(\bar{x}, y)))} \limsup_{y\to\infty} f(\bar{x}, y),$$

gdzie $\sigma(x) = \frac{\exp(x)}{1+\exp x}$.

□

Przy definiowaniu nowych funkcji możemy również w pewnych przypadkach używać granicy w punkcie. Jest to możliwe dzięki wyrażeniu granicy w punkcie poprzez granicę nieskończoną. Na przykład $\lim_{y\to\frac{1}{\pi}} \sin xy$ możemy zapisać jako $\lim_{y\to\infty} \sin x(\operatorname{arc\,tg} y)$.

W klasie rekurencyjnych funkcji rzeczywistych wprowadzono hierarchię zwaną η-hierarchią, która wiąże się ze stopniem nieciągłości funkcji. W tym przypadku nieciągłość może pojawić się wszędzie tam gdzie pojawia się w definicji funkcji operacja granicy. Hierarchia ta opisuje stopnie bazujące na liczbie koniecznych granic do zdefiniowania danej funkcji.

Oto definicje stopnia funkcji ze względu na pewien zbiór operatorów oraz η-hierarchii pochodzące z pracy [16].

Definicja 2.35 [16] *Niech $\mathcal{A} = [\mathcal{F}; \mathcal{O}]$ będzie przeliczalną algebrą funkcji z $\mathcal{O} = \{O_1, O_2, \ldots\}$ oraz $\tilde{\mathcal{O}}$ będzie podzbiorem $\mathcal{O}$. Stopień dobrej deskrypcji $d \in \mathcal{D}_{\mathcal{A}}$ dla zbioru operatorów $\tilde{\mathcal{O}}$ ze względu na algebrę funkcji $\mathcal{A}$, $rk(d)$, definiujemy indukcyjnie w następujący sposób:*

(i) $rk(\langle fun, n\rangle) = 0$,

(ii) jeżeli $O_n \notin \tilde{\mathcal{O}}$ *to* $rk(\langle O_p, n, d_1, \ldots, d_k\rangle) = \max(rk(d_1), \ldots, rk(d_k))$ *oraz*

(iii) jeżeli $O_n \in \tilde{\mathcal{O}}$ *to* $rk(\langle O_p, n, d_1, \ldots, d_k\rangle) = \max(rk(d_1), \ldots, rk(d_k)) + 1$.

Stopień funkcji $f \in \mathcal{A}$ dla $\tilde{\mathcal{O}}$ ze względu na algebrę $\mathcal{A}$, $rk(f)$, jest dany następująco:

$$rk(f) = \min\{rk(d) : d \text{ jest dobrą deskrypcja opisującą } f\}.$$

Definicja 2.36 [16] *Niech $\mathcal{H}$ (eta) oznacza algebrę funkcji dla $\mathcal{REC}((R))$. Hierarchię stopni dla operatora granicy nieskończonej ze względu na algebrę $\mathcal{H}$ dla $\mathcal{REC}(\mathbb{R})$ nazywamy η-hierarchią, n-ty stopień tej hierarchii oznaczymy przez H_n.*

$$H_n = \{f \in \mathcal{REC}(\mathbb{R}) : rk(f) \leqslant n\}.$$

Wyraźniejszy obraz η-hierarchii przedstawia poniższy wniosek.

Wniosek 2.2 [16] *η-hierarchia jest dana indukcyjnie w następujący sposób:*

(i) $H_0 = [-1^n, 0^n, 1^n, \mathfrak{i}_i^n; \mathrm{SK}, \mathrm{R}, \mathrm{V}]$,

(ii) $\tilde{H}_n = H_n \cup \{\mathrm{LS}(f) : f : (R)^{m+1} \to \mathbb{R}^k \in H_n\}$ *oraz*

(iii) $H_{n+1} = [\tilde{H}_n; \mathrm{SK}, \mathrm{R}, \mathrm{V}]$.

O η-hierarchii możemy myśleć jako o mierze skomplikowania rekurencyjnej funkcji rzeczywistej. Jeżeli $f \in H_j$, wówczas j oznacza liczbę zagnieżdżonych (nie występujących równolegle) η koniecznych „poprawek" aby z funkcji f uczynić funkcję ciągłą. Zatem funkcje z twierdzenia 2.17 są klasy H_0, gdyż można je zdefiniować nie używając operacji granicy, natomiast funkcje z twierdzenia 2.18 i twierdzenia 2.19 są klasy H_1, gdyż do zdefiniowania każdej z nich użyto jednokrotnie operatora LS. Przy definicji funkcji podłogi $\lfloor\rfloor$ użyto kilkakrotnie funkcji s, która tak naprawdę jest funkcją Θ złożoną

z funkcją sin, a więc jest funkcją definiowaną przy pomocy operatora LS. Na szczęście wystąpienia funkcji s mimo iż kilkakrotne w obrębie definicji funkcji podłogi $\lfloor\rfloor$, są wystąpieniami równoległymi, więc nadal pozostajemy w klasie H_1.

Twierdzenie 2.21 [16] *Funkcje* Γ *Eulera oraz* ζ *Riemanna są rekurencyjnymi funkcjami rzeczywistymi klasy* H_1*.*

Dowód. Funkcja Gamma Eulera: $\Gamma(x) = \int_0^\infty t^{x-1}\exp(-t)dt$. Łatwo widać, że $\Gamma(x) = \lim_{s'\to\infty}\int_0^{s'} s^{x-1}\exp(-s)ds$. Ponieważ $s^{x-1}\exp(-s)$ jest rekurencyjną funkcją rzeczywistą oraz $\int_0^{s'} s^{x-1}\exp(-s)ds$ jest w H_0, to Γ jest w H_1.

Prawdziwa jest następująca równość: $\zeta(x) = \frac{1}{\Gamma(x)}\int_0^\infty \frac{t^{x-1}}{1-\exp(-t)}dt$ dla $x > 0$, gdzie prawa strona może być zdefiniowana jako rekurencyjna funkcja rzeczywista używając poprzedniej konstrukcji. Zatem podobnie jak funkcja Γ, funkcja ζ jest w H_1.

□

Dla lepszej analizy funkcji wprowadzono pewien operator, który pozwala na kontrolowanie dziedziny i punktów osobliwych funkcji.

Definicja 2.37 [16] *Dla pewnej funkcji* $f : \mathbb{R}^{n+1} \to \mathbb{R}$ *definiujemy:*

$$\eta_y f(\bar{x}, y) = \begin{cases} 1 & \text{jeżeli } \lim_{y\to\infty} f(\bar{x}, y) \text{ jest zdefiniowana} \\ 0 & \text{w przeciwnym razie} \end{cases}$$

$$\eta^i_y f(\bar{x}, y) = \begin{cases} 1 & \text{jeżeli } \liminf_{y\to\infty} f(\bar{x}, y) \text{ jest zdefiniowana} \\ 0 & \text{w przeciwnym razie} \end{cases}$$

$$\eta^s_y f(\bar{x}, y) = \begin{cases} 1 & \text{jeżeli } \limsup_{y\to\infty} f(\bar{x}, y) \text{ jest zdefiniowana} \\ 0 & \text{w przeciwnym razie} \end{cases}$$

Zdefiniowana w ten sposób funkcja $\eta_y f(\bar{x}, y)$ jest funkcją charakterystyczną zbioru tych punktów $\bar{x}$, dla których $\lim_{y\to\infty} f(\bar{x}, y)$ jest dobrze zdefiniowana (bez punktów osobliwych). Analogicznie $\eta^i_y f(\bar{x}, y)$, $\eta^s_y f(\bar{x}, y)$ odgrywa taką samą rolę odpowiednio dla $\liminf_{y\to\infty} f(\bar{x}, y)$, $\limsup_{y\to\infty} f(\bar{x}, y)$. Pozostaje tylko pytanie kiedy taki operator jest rekurencyjnym operatorem rzeczywistym. W pracy [17] mamy następujące twierdzenie uszczegółowione potem w pracy [16].

Twierdzenie 2.22 [16] *Jeżeli funkcja f jest całkowicie rekurencyjną funkcją rzeczywistą z H_i, wówczas $\eta(f)$, $\eta^s(f)$ i $\eta^i(f)$ są w H_{i+1}.*

Dowód. Niech $f : \mathbb{R}^m \to \mathbb{R}$. Funkcja $\eta^s(f)$ jest dana następującym rekurencyjnym wyrażeniem rzeczywistym:

$$\eta^s(f)(\bar{x}) = 1 - \chi_{=}\left(\limsup_{y\to\infty} \sigma(f(\bar{x}, y)), 1\right) - \chi_{=}\left(\limsup_{y\to\infty} \sigma(f(\bar{x}, y)), 0\right).$$

Wiemy też, że $\limsup_{y\to\infty} \sigma(f(\bar{x}, y))$ istnieje ponieważ $\sigma \circ f$ jest ograniczoną, rzeczywistą, całkowicie rekurencyjną funkcją oraz $\eta^s(f)(\bar{x}) = 0$ wtedy i tylko wtedy, gdy $\limsup_{y\to\infty} \sigma(f(\bar{x}, y)) \in \{0, 1\}$. Podobne wyrażenie daje nam η^i.

Ponadto wiemy, że $\lim_{y\to\infty} f(\bar{x}, y)$ istnieje wtedy i tylko wtedy, gdy granica górna i dolna f istnieją i są sobie równe. Granice górna i dolna f istnieją i są sobie równe wtedy i tylko wtedy, gdy granica górna i dolna $\sigma \circ f$ są sobie równe. Z tego powodu $\eta(f)$ możemy zapisać jako:

$$\eta(f)(\bar{x}) = \eta^s(f)(\bar{x}) \times \eta^i(f)(\bar{x}) \times \chi_{=}\left(\liminf_{y\to\infty} \sigma(f(\bar{x}, y)), \limsup_{y\to\infty} \sigma(f(\bar{x}, y))\right).$$

□

W ten sposób zakończyliśmy prezentację modeli obliczeń dyskretnych i analogowych, które są przedmiotem dyskusji w dalszych częściach pracy.

Rozdział 3

Obliczenia analogowe a maszyna Turinga

Wstępna część tego rozdziału poświęcona została wprowadzeniu pojęcia obliczalności przy użyciu maszyn Turinga, poza zbiorem liczb naturalnych. Powstała ona w głównej mierze w oparciu o monografię K. Weihraucha [76] oraz pracę K. Weihraucha i X. Zhenga [77].

Kolejne części dotyczą różnych modeli obliczeń w dziedzinie liczb rzeczywistych. Stanowią porównanie ich możliwości z klasycznym modelem obliczeń jakim jest maszyna Turinga. W tym kontekście pojawia się ważny problem symulowania maszyn Turinga w modelach obliczeń operujących na liczbach rzeczywistych. Ponieważ systemy funkcji, które mają możliwość takiej symulacji, są w stanie symulować również uniwersalną maszynę Turinga. Dlatego też te systemy mogą wykonywać uniwersalne obliczenia. Ponadto badanie rozstrzygalności problemów obliczeniowych może być analizowane tylko w modelach mających zdolność takiej symulacji.

Druga część rozdziału to porównanie rekurencyjnych funkcji rzeczywistych i maszyny Turinga w oparciu o własną pracę [57]. Zaprezentowany w niej został sposób symulowania obliczeń maszyny Turinga w modelu rekurencyjnych funkcji rzeczywistych.

Trzecia część powstała w oparciu o prace [33] oraz własną pracę [60]. Przedstawiono w niej symulację obliczeń maszyny Turinga poprzez analityczne funkcje zmiennej rzeczywistej.

3.1 Maszyna Turinga i obliczenia na liczbach rzeczywistych

Współcześnie, wśród aparatury obliczeniowej znajduje się wiele analogowych maszyn, które bezpośrednio generują funkcje ciągłe zmiennej rzeczywistej jako swoje wyniki. Te funkcje uznajemy za obliczalne na skutek faktu, iż istnieją urządzenia, które generują ich wartości. Możemy jednak spróbować zrozumieć naturę obliczalności takich funkcji przez pryzmat tradycyjnych komputerów. Pojęcie obliczalnej liczby rzeczywistej jako pierwszy wprowadził Alan Turing w pracy [74] z 1936 roku. Praca ta stanowi początek nowej dziedziny matematyki i informatyki, analizy obliczeniowej. Ponieważ maszyna Turinga liczy właściwie funkcje obliczalne zdefiniowane na liczbach naturalnych, tak naprawdę Turing, obliczalne liczby rzeczywiste definiuje jako liczby o obliczalnym rozwinięciu dziesiętnym. Liczba rzeczywista $x \in \mathbb{R}$ jest obliczalna jeżeli istnieje obliczalna funkcja $f : \mathbb{N} \to \{0, 1, \ldots, 9\}$, taka że $x = \Sigma_{n=0}^{\infty} f(n)10^{-(n+1)}$. Dalej przy wprowadzaniu pojęcia obliczalnej funkcji zmiennej rzeczywistej dużą rolę odegrali w latach sześćdziesiątych dwaj polscy matematycy S. Mazur [40] i A. Grzegorczyk [27]. Obecnie literatura dotycząca teorii rekursji przedstawia precyzyjne definicje funkcji zmiennej rzeczywistej obliczalnej w powyższym rozumieniu - Grzegorczyk [27], Lacombe [34], Weihrauch [76]. Przybliżymy je w bieżącym paragrafie.

Za model klasycznego komputera posłużyła nam maszyna Turinga. Zaprezentujemy teraz sposób połączenia obliczeń na liczbach rzeczywistych z obliczeniami maszyny Turinga. Aby zdefiniować obliczalną funkcję zmiennej rzeczywistej na maszynie Turinga, musimy w jakiś sposób aproksymować liczby rzeczywiste przez liczby naturalne. W matematyce pojawia się kilka różnych reprezentacji liczby rzeczywistej, które z matematycznego punktu widzenia są równoważne. Na ich podstawie możliwe jest wprowadzenie kilku różnych definicji pojęcia obliczalności w zbiorze liczb rzeczywistych.

Jedna z nich definiuje liczby rzeczywiste jako granice ciągów Cauchy'ego, czyli takich ciągów liczb wymiernych $\{r_n\}_{n\in\mathbb{N}}$, że dla każdego $\varepsilon > 0$ istnieje $m \in \mathbb{N}$, takie że dla wszystkich $k, l \geqslant m$, $|r_k - r_l| < \varepsilon$. Każdy ciąg Cauchy'ego $\{r_n\}_{n\in\mathbb{N}}$ reprezentuje pewną liczbę rzeczywistą x, taką że $x = \lim_{n\to\infty} r_n$. Taka reprezentacja jest nazywana naiwną reprezentacją Cauchy'ego liczby rzeczywistej. Bardziej użyteczna w analizie obliczeniowej jest reprezentacja, która zakłada efektywność zbieżności ciągu Cauchy'ego do x w takim sensie, że dla dowolnej liczby n, $|r_n - x| < 2^{-n}$. Ciąg $\{r_n\}_{n\in\mathbb{N}}$ możemy skojarzyć

z funkcją $f : \mathbb{N} \to \mathbb{Q}$, taką że dla dowolnego indeksu $n \in \mathbb{N}$, $f(n) = r_n$. Liczbę rzeczywistą x nazwiemy obliczalną na maszynie Turinga, jeżeli wspomniana funkcja $f : \mathbb{N} \to \mathbb{Q}$ jest obliczalna na maszynie Turinga. Tę reprezentację nazywamy po prostu reprezentacją Cauchy'ego liczby rzeczywistej i wykorzystaliśmy ją do wprowadzenie pojęcia obliczalności liczby rzeczywistej w tym paragrafie.

Innym sposobem definiowania liczb rzeczywistych w matematyce są tzw. przekroje Dedekinda. Są to podziały (C, D) liczb wymiernych, takie że $u < v$ dla wszystkich $u \in C$ oraz $v \in D$ i ponadto C nie zawiera elementu maksymalnego. Przekrój Dedekinda (C, D) reprezentuje liczbę rzeczywistą x, jeżeli x jest najmniejszym górnym ograniczeniem C. Przekrój Dedekinda jest jednoznacznie wyznaczony przez zbiór C, który definiujemy jako zbiór $C_x = \{r \in \mathbb{Q} : r < x\}$ liczb wymiernych oraz uważamy C_x jako reprezentację x. Zbiór C_x może być opisany poprzez funkcję charakterystyczną: $f : \mathbb{Z} \times \mathbb{N} \to \{0, 1\}$, taką że $f(n, m) = 1$ wtedy i tylko wtedy, gdy $\frac{n}{m} < x$. Wówczas dodatnią liczbę rzeczywistą x nazywamy obliczalna na maszynie Turinga jeżeli istnieje funkcja $f : \mathbb{N}^2 \to \{0, 1\}$ obliczalna na maszynie Turinga, taka że

$$f(n, m) = \begin{cases} 1 & n/m < x \\ 0 & n/m \geqslant x. \end{cases}$$

Najbardziej znaną reprezentacją liczby rzeczywistej jest reprezentacja dziesiętna. Jeżeli ograniczymy zbiór $\mathbb{R}$ do przedziału $[0, 1)$ to możemy reprezentować liczbę rzeczywistą x jako $x = 0.a_0a_1a_2\ldots$, gdzie cyfry $a_n \in \{0, 1, \ldots, 9\}$, stąd $x = \Sigma_{n=0}^{\infty} a_n 10^{-(n+1)}$. Ciąg $\{a_n\}_{n\in\mathbb{N}}$ możemy skojarzyć z funkcją $f : \mathbb{N} \to \{0, 1, \ldots, 9\}$, więc dziesiętna reprezentacja $x \in [0, 1)$ może być zdefiniowana poprzez funkcję $f : \mathbb{N} \to \{0, 1, \ldots, 9\}$, taką że $x = \Sigma_{n=0}^{\infty} f(n)10^{-(n+1)}$. Dziesiętna reprezentacja przedstawia liczby rzeczywiste przy podstawie 10. Ogólnie podstawę 10 możemy zastąpić dowolną inną liczbą naturalną $p > 1$ otrzymując reprezentację x w systemie pozycyjnym o podstawie p. Jeżeli przyjmiemy $p = 2$, dostaniemy binarną reprezentację liczby rzeczywistej, która w naturalny sposób zestawia liczby rzeczywiste z liczbami naturalnymi. Niech $f : \mathbb{N} \to \{0, 1\}$ będzie binarną reprezentacją liczby rzeczywistej x. Otrzymamy $x = \Sigma_{n=0}^{\infty} f(n)2^{-(n+1)} = \Sigma_{n\in A} 2^{-(n+1)}$, gdzie $A = \{n \in \mathbb{N} : f(n) = 1\}$. W ten sposób binarna reprezentacja x jest funkcją charakterystyczną zbioru A. W tym przypadku liczbę rzeczywista x nazwiemy obliczalną na maszynie Turinga jeżeli istnieje funkcja $f : \mathbb{N} \to \{0, 1, \ldots, (p - 1)\}$ obliczalna na maszynie Turinga, taka że

$x = \Sigma_{n=0}^{\infty} f(n) p^{-(n+1)}$.

Inną znaną w matematyce reprezentację liczby rzeczywistej stanowi ciąg $\{I_n\}_{n\in\mathbb{N}}$ przedziałów domkniętych o końcach wymiernych, takich że $I_{n+1} \subseteq I_n$. Ciąg $\{I_n\}_{n\in\mathbb{N}}$ reprezentuje liczbę rzeczywistą x jeżeli x jest jedynym wspólnym elementem wszystkich przedziałów I_n. Taki ciąg przedziałów definiują dwie funkcje. Dlatego przedziałową reprezentację x możemy zdefiniować jako parę funkcji: $f, g : \mathbb{N} \to \mathbb{Q}$, takich że dla wszystkich $n \in \mathbb{N}$, $f(n) \leqslant f(n+1) \leqslant x \leqslant g(n+1) \leqslant g(n)$ oraz $\lim_{n\to\infty}(g(n) - f(n)) = 0$. Dla tej reprezentacji liczbę rzeczywistą x nazwiemy obliczalną na maszynie Turinga, jeżeli wspomniane funkcje $f, g : \mathbb{N} \to \mathbb{Q}$ są obliczalne na maszynie Turinga.

Liczbę rzeczywistą możemy również reprezentować w postaci ułamka łańcuchowego. Każdą liczbę niewymierną dodatnią x możemy w sposób jednoznaczny zapisać w postaci ułamka łańcuchowego:

$$x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \ldots}},$$

gdzie $a_0 \in \mathbb{N}$ oraz $a_n \in \mathbb{N}$ dla $n \geqslant 1$. Zapisujemy $x = [a_0, a_1, a_2, \ldots]$, gdzie a_n nazywamy częściowym ilorazem stopnia n. Jeżeli x jest liczbą wymierną to jej ułamek łańcuchowy ma postać:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\vdots \cfrac{1}{a_n}}}}$$

dla pewnej liczby n co zapisujemy $x = [a_0, a_1, \ldots, a_n]$. Dla zachowania jednolitego zapisu możemy przedstawić liczbę wymierną w następujący sposób $x = [a_0, a_1, \ldots, a_n, 0, 0 \ldots]$. Zatem zarówno niewymierną jaki i wymierną liczbę x możemy zdefiniować poprzez funkcję $f : \mathbb{N} \to \mathbb{N}$, taką że $x = [f(0), f(1), f(2), \ldots]$. Jeżeli taka funkcja f jest obliczalna na maszynie Turinga, to liczbę x nazywamy obliczalną na maszynie Turinga.

Wszystkie wymienione reprezentacje liczby rzeczywistej są sobie równoważne z punktu widzenia matematyki, tzn. wszystkie definiują tą samą strukturę. Reprezentacje te używają do zdefiniowania liczby rzeczywistej funkcji przekształcających zbiór liczb naturalnych w zbiór liczby naturalnych lub funkcji przekształcających zbiór liczby naturalnych w zbiór liczb wymiernych. O ile wiemy jak jest zdefiniowana funkcja obliczalna na maszynie Turinga w pierwszym przypadku (definicja 2.12), to pojęcie funkcji obliczalnej

na maszynie Turinga z $\mathbb{N}$ w $\mathbb{Q}$ wymaga dodatkowego zdefiniowania, co ma miejsce w dalszej części tego paragrafu.

Mimo równoważności powyższych reprezentacji w sensie matematycznym (wszystkie opisują ten sam zbiór liczb rzeczywistych) nie mamy równoważności między wprowadzonymi na ich podstawie definicjami liczb rzeczywistych obliczalnych. Po dołożeniu warunku obliczalności zbiór liczb, jaki otrzymamy startując z jednej z wymienionych reprezentacji, może być inny niż gdybyśmy rozważyli inną reprezentację. Problem nierównoważności tych definicji oraz relacji zachodzących między nimi był rozważany między innymi w pracach Speakera [72], Péte'ego [56], Mostowskiego [48], Lehmana [36].

Przybliżymy teraz sposób definiowania liczb rzeczywistych oraz funkcji definiowanych na zbiorze liczb rzeczywistych, obliczalnych na maszynie Turinga, zaproponowany w pracy [27], opierający się na reprezentacji Cauchy'ego liczby rzeczywistej.

Na początek zakodujemy liczbę wymierną przy pomocy liczb naturalnych. Użyjemy do tego celu dobrze znanej funkcji Cantora do kodowania par liczb:

$$\langle i, j\rangle = \frac{(i+j)(i+j+1)}{2} + j$$

oraz funkcji dekodujących:

$$\pi_1(\langle i, j\rangle) = i, \pi_2(\langle i, j\rangle) = j.$$

Oczywiście możemy rozszerzyć powyższe kodowanie na więcej argumentów: $\langle i, j, k\rangle = \langle i, \langle j, k\rangle\rangle$. Możliwe jest również zdefiniowanie efektywnej numeracji liczb wymiernych $v_{\mathbb{Q}} : \mathbb{N} \to \mathbb{Q}$, następująco:

$$v_{\mathbb{Q}}(n) = \frac{\pi_1(\pi_1(n)) - \pi_1(\pi_2(n))}{\pi_2(n) + 1},$$

wówczas dla $n = \langle i, j, k\rangle$ mamy

$$v_{\mathbb{Q}}(n) = \frac{i-j}{k+1}.$$

Definicja 3.1 [76] *Funkcję $f : \mathbb{N}^k \to \mathbb{Q}$ nazywamy obliczalną jeżeli istnieje funkcja $g : \mathbb{N}^k \to \mathbb{N}$ obliczalna na maszynie Turinga, taka że*

$$\forall(n_1, \ldots, n_k \in \mathbb{N}) f(n_1, \ldots, n_k) = v_{\mathbb{Q}}(g(n_1, \ldots, n_k)).$$

Klasę funkcji obliczalnych $f : \mathbb{N}^k \to \mathbb{Q}$ oznaczymy przez $F_{\mathbb{Q}}$.

Wprowadzimy teraz pojęcie obliczalnego ciągu liczb rzeczywistych. Wiemy, że każda liczba rzeczywista jest granicą ciągu Cauchy'ego liczb wymiernych. Ciąg liczb wymiernych $\{r_n\}_{n\in\mathbb{N}}$ będziemy utożsamiać z funkcją $r : \mathbb{N} \to X$, $X \subseteq \mathbb{Q}$, taką że $\forall(k \in \mathbb{N})(r(k) = r_k)$, gdzie zbiór X stanowią elementy ciągu $\{r_n\}_{n\in\mathbb{N}}$. Naszą konstrukcję opieramy na obliczalności i efektywnej zbieżności ciągu $\{r_n\}_{n\in\mathbb{N}}$.

Definicja 3.2 [76] *Ciąg liczb wymiernych $\{r_n\}_{n\in\mathbb{N}}$ nazywamy obliczalnym, jeżeli istnieje obliczalna funkcja $r : \mathbb{N} \to \mathbb{Q}$, taka że $\forall(k \in \mathbb{N})(r(k) = r_k)$.*

Otrzymamy zatem następującą definicję liczby rzeczywistej obliczalnej.

Definicja 3.3 [76] *Liczbę rzeczywistą x nazywamy obliczalną, jeżeli istnieje obliczalny ciąg liczb wymiernych $\{r_n\}_{n\in\mathbb{N}}$ zbieżny do x efektywnie, to znaczy:*

$$\forall(n \in \mathbb{N})(|x - r_n| < 2^{-n}).$$

Z powyższych definicji wynika, że wszystkie liczby rzeczywiste wymierne są obliczalne. Również niektóre liczby niewymierne są obliczalne, np. pierwiastek drugiego stopnia z dowolnej liczby naturalnej. Z pracy [74] wynika, że wszystkie liczby rzeczywiste algebraiczne [1] są obliczalne oraz niektóre liczby rzeczywiste przestępne [2].

Przykład 3.1 Liczba e jest przestępną liczbą rzeczywistą obliczalną.

Mamy $e = \sum_{i=0}^{\infty} \frac{1}{i!}$. Zdefiniujmy obliczalny ciąg liczb wymiernych następująco:

$$r_n = \sum_{i=0}^{n} \frac{1}{i!}.$$

Dal dowolnej liczby $n \in \mathbb{N}$ mamy:

$$|r_n - e| = |\sum_{i=0}^{n} \frac{1}{i!} - \sum_{i=0}^{\infty} \frac{1}{i!}| = \sum_{i=n+1}^{\infty} \frac{1}{i!} < \sum_{i=n+1}^{\infty} \frac{1}{2^i} = 2^{-n}.$$

Stąd e jest liczbą rzeczywistą obliczalną.

[1] Liczba rzeczywista algebraiczna to liczba, która jest pierwiastkiem rzeczywistym pewnego wielomianu o współczynnikach wymiernych.

[2] Liczbę rzeczywistą, która nie jest algebraiczna, nazywamy liczbą rzeczywistą przestępną.

Definicja 3.4 [76] *Ciąg $(x_n)_{n\in\mathbb{N}}$ liczb rzeczywistych nazywamy obliczalnym, jeżeli istnieje obliczalny podwójny ciąg $(r_{n,k})_{n,k\in\mathbb{N}}$ liczb wymiernych, który jest efektywnie, jednostajnie zbieżny do $(x_n)_{n\in\mathbb{N}}$, tj. istnieje obliczalna na maszynie Turinga funkcja $e : \mathbb{N}^2 \to \mathbb{N}$, taka że*

$$(\forall n, m, k \in \mathbb{N})(k \geqslant e(n, m)) \to |r_{n,k} - x_n| < 2^{-m}.$$

Wyjaśnijmy, że powyższa definicja może być wyrażona w następujący nieformalny sposób: ciąg liczb rzeczywistych jest obliczalny, jeżeli istnieje taka maszyna Turinga, która dla danego indeksu n pewnego elementu ciągu i dla danej precyzji 2^{-m} wyznaczy kod takiej liczby wymiernej r, która różni się od x_n o mniej niż 2^{-m}.

Twierdzenie 3.1 [76] *Niech $\{x_n\}_{n\in\mathbb{N}}$ będzie zbieżnym, obliczalnym ciągiem liczb rzeczywistych, dla którego istnieje taka funkcja obliczalna $m : \mathbb{N} \to \mathbb{N}$, że:*

$$\forall(n, i, j \in \mathbb{N})(i, j > m(n) \to |x_i - x_j| < 2^{-n}).$$

Wówczas jego granica $x = \lim_{n\to\infty} x_n$ jest obliczalną liczbą rzeczywistą.

Dowód. Niech $\{x_n\}_{n\in\mathbb{N}}$ będzie zbieżnym, obliczalnym ciągiem liczb rzeczywistych oraz niech $x = \lim_{n\in\mathbb{N}} x_n$. Zatem, zgodnie z definicją 3.4, istnieje obliczalny, podwójny ciąg liczb wymiernych spełniający warunek:

$$(\forall n, m, k \in \mathbb{N})(k \geqslant e(n, m)) \to |r_{n,k} - x_n| < 2^{-m}.$$

Niech $m : \mathbb{N} \to \mathbb{N}$ będzie funkcją obliczalną spełniającą warunek:

$$\forall(n, i, j \in \mathbb{N})(i, j > m(n) \to |x_i - x_j| < 2^{-n}).$$

Zdefiniujmy obliczalny ciąg liczb wymiernych: $w_n = r_{m(n+1),e(m(n+1),n+1)}$. Wówczas dla dowolnej liczby $n \in \mathbb{N}$ mamy:

$$|w_n - x| = |w_n - x_{m(n+1)} + x_{m(n+1)} - x| \leqslant |w_n - x_{m(n+1)}| + |x_{m(n+1)} - x| \leqslant$$
$$\leqslant |r_{m(n+1),e(m(n+1),n+1)} - x_{m(n+1)}| + 2^{-(n+1)} < 2^{-(n+1)} + 2^{-(n+1)} = 2^{-n}.$$

Zatem, zgodnie z definicją 3.3, x jest liczbą rzeczywistą obliczalną.

□

Przykład 3.2 Liczba π jest przestępną liczbą rzeczywistą obliczalną.

Mamy $\pi = \sum_{i=0}^{\infty}(-1)^i \frac{1}{2i+1}$. Zdefiniujmy obliczalny ciąg liczb rzeczywistych następująco:

$$x_n = \sum_{i=0}^{n}(-1)^i \frac{1}{2i+1}.$$

Dla dowolnej liczby $i \leqslant j$ mamy:

$$0 \leqslant (-1)^i \sum_{k=i}^{j}(-1)^k \frac{1}{2k+1} \leqslant \frac{1}{2i+1} \rightarrow |\sum_{k=i}^{j}(-1)^k \frac{1}{2k+1}| \leqslant \frac{1}{2i+1}.$$

Dla dowolnych liczb $n, i, j \in \mathbb{N}$, takich że $j > i \geqslant m(n)$ mamy:

$$|x_i - x_j| = |\sum_{k=i+1}^{j}(-1)^k \frac{1}{2k+1}| \leqslant \frac{1}{2i+3} < \frac{1}{i} \leqslant 2^{-n}.$$

Stąd, na mocy twierdzenia 3.1, liczba $\frac{\pi}{4}$ jest obliczalna, zatem π jest również liczbą obliczalną.

Scharakteryzujemy teraz pewien rodzaj liczb rzeczywistych ze względu na ich związek z liczbami rzeczywistymi obliczalnymi.

Definicja 3.5 [76] *Niech $A \subseteq \mathbb{N}$. Binarną liczbą rzeczywistą zbioru A nazywamy liczbę*

$$x[A] = \sum_{i \in A} 2^{-i}.$$

Twierdzenie 3.2 [76] *Niech $A \subseteq \mathbb{N}$, $x[A]$ jest obliczalną liczbą rzeczywistą wtedy i tylko wtedy, gdy A jest zbiorem rekurencyjnym.*

Dowód. Załóżmy, że $x[A]$ jest liczbą obliczalną. Jeżeli A lub $\neg A$ jest skończone, to zbiór A jest rekurencyjny. Niech A i $\neg A$ będą zbiorami nieskończonymi. Ponieważ $x[A]$ jest liczbą obliczalną, to na mocy definicji 3.3 istnieje obliczalny ciąg liczb wymiernych $\{r_n\}_{n \in \mathbb{N}}$, taki że:

$$\forall (n \in \mathbb{N})(|x[A] - r_n| < 2^{-n}).$$

Niech

$$A_n = \{i \in A : i < n\} \text{ i } w_n = \sum_{i \in A_n} 2^{-i} + 2^{-n}$$

oraz

$$k_n = \min_{i \in \mathbb{N}}(w_n \notin [r_i - 2^{-i}, r_i + 2^{-i}]).$$

Takie k_n zawsze istnieje ponieważ x[A] nie jest skończoną, dwójkową liczbą wymierną (x[A] nie jest liczbą postaci: $\sum_{i=0,i\in A}^{n} 2^{-i}$). Wówczas mamy: jeśli $w_n < r_{k_n} - 2^{-k_n}$, to $n \in A$, jeśli $w_n > r_{k_n} + 2^{-k_n}$, to $n \notin A$, zatem zbiór A jest rekurencyjny.

Załóżmy, że A jest zbiorem rekurencyjnym. Wówczas jego funkcja charakterystyczna $c_A : \mathbb{N} \to \{0, 1\}$ jest obliczalna na maszynie Turinga. Niech

$$r_n = \sum_{i=0}^{n+1} (2^{-i}(1 - c_A(i))).$$

Ciąg $\{r_n\}_{n\in\mathbb{N}}$ jest obliczalnym ciągiem liczb wymiernych. Ponadto dla każdej liczby $n \in \mathbb{N}$ mamy:

$$|x[A] - r_n| = |\sum_{i\in A} 2^{-1} - \sum_{i=0}^{n} +1(2^{-i}(1 - c_A(i)))| \leqslant \sum_{i=n+2}^{\infty} 2^{-i} = 2^{-(n+1)} < 2^{-n}.$$

Zatem, na mocy definicji 3.3, $x[A]$ jest liczbą rzeczywistą obliczalną.

□

Twierdzenie 3.3 [76] *Zbiór liczb rzeczywistych obliczalnych jest zamknięty ze względu na operacje arytmetyczne: dodawania, odejmowania, mnożenia i dzielenia.*

Dowód. Wynika z definicji 3.3 oraz z własności wymienionych działań arytmetycznych dla zbieżnych ciągów liczb wymiernych.

□

Na koniec zdefiniujemy funkcję zmiennej rzeczywistej, która jest obliczalna na maszynie Turinga. Chcielibyśmy mieć następujące intuicyjne własności: jeżeli dany wymierny argument jest w określonym sąsiedztwie liczby x, wówczas maszyna Turinga generuje liczbę wymierną w określonym sąsiedztwie wartości $f(x)$. Ponadto powinniśmy mieć możliwość decydowania poprzez zadany parametr o tym jak małe jest to sąsiedztwo.

Zatem chcielibyśmy mieć dla funkcji f realizowanej przez maszynę Turinga M_f efektywną procedurę, która dla danej precyzji m oraz danego ciągu $r_{n,k}$ (generowanego przez inną maszynę Turinga) zbieżnego do x, znajduje takie n_0, k_0, że $M_f(r_{n_0 k_0}) = q$ oraz $|q - f(x)| < 2^{-m}$.

Zgodnie z tą intuicją możemy przedstawić następującą definicję obliczalnej funkcji zmiennej rzeczywistej, w sensie funkcji obliczalnej przez maszynę Turinga [77].

Definicja 3.6 [76] *Funkcję zmiennej rzeczywistej* $f : [a, b] \to \mathbb{R}$ *nazywamy obliczalną jeśli:*

1. *f odwzorowuje każdy obliczalny ciąg liczb rzeczywistych* $\{x_n\}_{n\in\mathbb{N}}$ *w obliczalny ciąg liczb rzeczywistych* $\{f(x_n)\}_{n\in\mathbb{N}}$ *(własność Banacha-Mazura);*

2. *f jest efektywnie, jednostajnie ciągła, to znaczy istnieje obliczalna funkcja* $d : \mathbb{N} \to \mathbb{N}$*, taka że:*

$$\forall(x, y \in [a, b])\forall(m \in \mathbb{N})(|x - y| < 2^{-d(m)} \Rightarrow |f(x) - f(y)| < 2^{-m}).$$

Taka definicja daje nam całkiem inne spojrzenie na problem obliczalności w dziedzinie rzeczywistej. Ograniczając zbiór funkcji zmiennej rzeczywistej w taki sposób, konstruujemy tylko funkcje, które mogą być efektywnie aproksymowane przez maszynę Turinga. W tym sensie są one fizycznie obliczalne przez komputery cyfrowe, ponieważ zawsze możemy jako wynik otrzymać liczbę wymierną bliską oryginalnej wartości funkcji z żądaną precyzją.

Przykład 3.3 Funkcja $f(x) = x^2$ dla $x \in \mathbb{R}$ jest obliczalna w sensie definicji 3.6.

1. Niech $\{x_n\}_{n\in\mathbb{N}}$ będzie obliczalnym ciągiem liczb rzeczywistych, takim że $x_n \in [a, b]$.

   Zatem, zgodnie z definicją 3.4, istnieje podwójny obliczalny ciąg liczb wymiernych $\{r_{n,k}\}_{n,k\in\mathbb{N}}$, $r_{n,k} \in [a, b]$ oraz maszyna Turinga M_e, taka że

$$\forall(n, m, k \in \mathbb{N})(k \geqslant M_e(n, m) \to |r_{n,k} - x_n| < 2^{-m}).$$

   Ciąg $\{r_{n,k}\}_{n,k\in\mathbb{N}}$ jest obliczalny, zatem istnieje maszyna Turinga M_r, taka że $M_r(n, k)$ wyznacza liczbę wymierną $r_{n,k}$ znajdując licznik $l_{n,k} \in \mathbb{N}$ i mianownik $m_{n,k} \in \mathbb{N}$ ułamka $r_{n,k} = \frac{l_{n,k}}{m_{n,k}}$.

   Pokażemy teraz, że ciąg $\{f(x_n)\}_{n\in\mathbb{N}}$ jest ciągiem obliczalnym.

   Niech $\{p_{n,k}\}_{n,k\in\mathbb{N}}$ będzie ciągiem zdefiniowanym następująco:

$$p_{n,k} = f(r_{n,k}).$$

   Zdefiniujmy maszynę Turinga M_p w następujący sposób:

$M_p(n,k)\{$

(a) uruchom maszynę $M_r(n,k)$;

(b) uruchom maszynę $M_*(l_{n,k}, l_{n,k})$;

(c) uruchom maszynę $M_*(m_{n,k}, m_{n,k})$;

(d) zwróć wynik obliczeń maszyn $M_*(l_{n,k}, l_{n,k})$ i $M_*(m_{n,k}, m_{n,k})$;

$\}$

W powyższym zapisie $l_{n,k}, m_{n,k}$ oznaczają wynik działania maszyny $M_r(n,k)$.

Maszyna $M_*(n,m)$ wyznacza liczbę naturalną równą $n * m$, więc maszyna $M_p(n,k)$ wyznacza liczbę wymierną będącą elementem ciągu $\{p_{n,k}\}_{n,k\in\mathbb{N}}$ o indeksie n,k. Zatem ciąg $\{p_{n,k}\}_{n,k\in\mathbb{N}}$ jest obliczalny.

Pokażemy teraz, że istnieje maszyna Turinga $M_{e_1}(n,m)$, taka że:

$$\forall(n,m,k\in\mathbb{N})(k \geqslant M_{e_1}(n,m) \rightarrow |p_{n,k} - f(x_n)| < 2^{-m}).$$

Mamy:

$$|p_{n,k} - f(x_n)| = |r_{n,k}^2 - x_n^2| = |r_{n,k} - x_n||r_{m,k} + x_{n,k}| \leqslant |r_{n,k} - x_n|2|b| <$$

$$< 2^{-m}2|b| < 2^{-m+2+\log_2\lfloor|b|\rfloor},$$

gdzie

$$\log_2(n) = \min_{i\leqslant n}[2^i > n] - 1$$

jest obliczalną na maszynie Turinga funkcją działającą z $\mathbb{N} \rightarrow \mathbb{N}$. Niech M_{log_2} będzie maszyną Turinga wyznaczającą $\log_2$ oraz $M_+, M_{||}, M_{\lfloor\rfloor}$ będą maszynami Turinga wyznaczającymi odpowiednio wartości sumy, wartości bezwzględnej liczby wymiernej, oraz obcięcia liczby wymiernej do jej części całkowitej.

Zdefiniujmy maszynę M_{e_1} następująco:

$M_{e_1}(n,m)\{$liczba wymierna b jest stałą znaną maszynie $M_{e_1}\}\{$

(a) uruchom maszynę $M_{||}(b)$;

(b) uruchom maszynę $M_{\lfloor\rfloor}(|b|)$;

(c) uruchom maszynę $M_{log_2}(\lfloor|b|\rfloor)$;

(d) uruchom maszynę $M_+(m, 2)$;

(e) uruchom maszynę $M_+(m+2, log_2\lfloor|b|\rfloor)$;

(f) uruchom maszynę $M_e(n, m+2+log_2\lfloor|b|\rfloor)$;

(g) zwróć wynik obliczeń maszyny $M_e(n, m+2+log_2\lfloor|b|\rfloor)$;

}

Maszyna M_{e_1} wyznaczy dla danego $n, m \in \mathbb{N}$ takie $i \in \mathbb{N}$, że $\forall(k \geqslant i)(|p_{n,k} - f(x_n)| < 2^{-m})$. Stąd istnieje obliczalna na maszynie Turinga funkcja $e_1(n, m)$, taka że:

$$\forall(n, m, k \in \mathbb{N})(k \geqslant e_1(n, m) \rightarrow |p_{n,k} - f(x_n)| < 2^{-m}).$$

Zatem, zgodnie z definicją 3.4, ciąg $\{f(x_n)\}_{n\in\mathbb{N}}$ jest obliczalnym ciągiem liczb rzeczywistych.

2. (Warunek ciągłości) Pokażemy, że istnieje obliczalna na maszynie Turinga funkcja $d : \mathbb{N} \rightarrow \mathbb{N}$, taka że:

$$\forall(x, y \in [a, b])\forall(m \in \mathbb{N})(|x - y| < 2^{-d(m)} \rightarrow |f(x) - f(y)| < 2^{-m}).$$

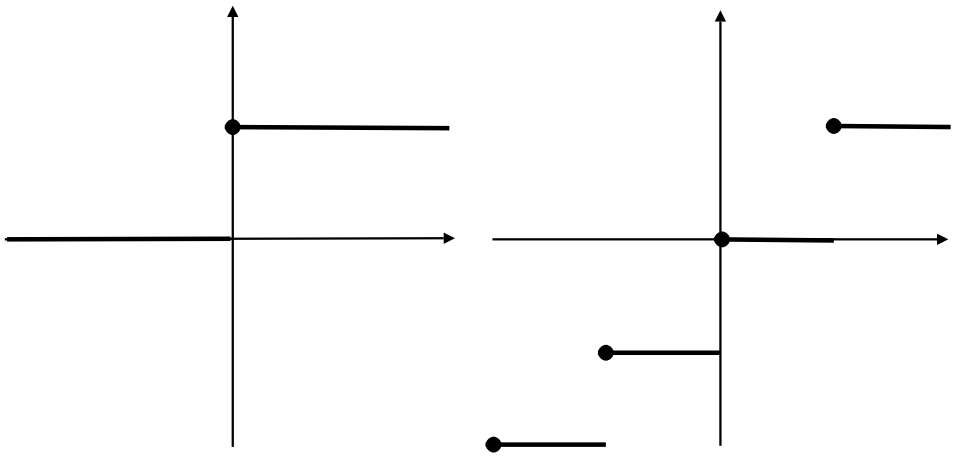
Przyjmijmy $d(m) = m + 2 + \log_2\lfloor|b|\rfloor$, otrzymamy wówczas:

$$|f(x) - f(y)| = |x^2 - y^2| = |x - y||x + y| < 2^{-(m+2+\log_2\lfloor|b|\rfloor)}2|b| < 2^{-m}.$$

Łatwo widać, że istnieje maszyna Turinga M_d wyznaczająca wartość funkcji d, zatem d jest szukaną, obliczalną na maszynie Turinga funkcją.

Funkcjami obliczalnymi, zdefiniowanymi na zmiennych rzeczywistych są między innymi: wielomiany o współczynnikach obliczalnych, funkcja wykładnicza, wartość bezwzględna, funkcje trygonometryczne jak i wiele innych (praca [76]).

Konsekwencją definicji 3.6 jest to, że proste funkcje takie jak funkcja skoku: $\text{sgn}(x) = \begin{cases} 0 & x > 0 \\ 1 & x \leqslant 0 \end{cases}$, funkcja podłogi (część całkowita z x) są nieobliczalne (rys. 3.1) gdyż są nieciągłe.



Rysunek 3.1: Przykłady nieobliczalnych funkcji zmiennej rzeczywistej

W pracy [77] pokazano, że w tak zdefiniowanej klasie obliczalnych liczb rzeczywistych można wprowadzić hierarchię arytmetyczną, analogiczną do hierarchii arytmetycznej podzbiorów liczb naturalnych.

Definicja 3.7 [76](Hierarchia arytmetyczna liczb rzeczywistych)

1. $\Sigma_0 = \Pi_0 = \Delta_0 = \{x \in \mathbb{R} : x \text{ jest bliczalne}\}$.

2. Dla $n \in \mathbb{N}$, $n > 0$:

$$\Sigma_n = \{x \in \mathbb{R} : (\exists f \in F_{\mathbb{Q}}) = \sup_{i_1 \in \mathbb{N}} \inf_{i_2 \in \mathbb{N}} \sup_{i_3 \in \mathbb{N}} \ldots \Theta_{i_n} f(i_1, i_2, \ldots, i_n)\};$$

$$\Pi_n = \{x \in \mathbb{R} : (\exists f \in F_{\mathbb{Q}}) = \inf_{i_1 \in \mathbb{N}} \sup_{i_2 \in \mathbb{N}} \inf_{i_3 \in \mathbb{N}} \ldots \bar{\Theta}_{i_n} f(i_1, i_2, \ldots, i_n)\};$$

$$\Delta_n = \Sigma_n \cap \Pi_n,$$

gdzie Θ_{i_n} *oznacza* $\sup_{i_n \in \mathbb{N}}$, *jeśli n jest nieparzyste i* $\inf_{i_n \in \mathbb{N}}$, *jeśli n parzyste oraz* $\bar{\Theta}_{i_n}$ *oznacza* $\inf_{i_n \in \mathbb{N}}$, *jeśli n jest nieparzyste i* $\sup_{i_n \in \mathbb{N}}$, *jeśli n parzyste.*

Aby bliżej scharakteryzować zaproponowaną hierarchię przytoczymy teraz kilka dodatkowych definicji oraz twierdzeń z pracy [77].

Lemat 3.1 [77] *Dla dowolnej liczby* $n \in \mathbb{N}$, $n > 0$ *i dowolnej liczby* $x \in \mathbb{R}$ *mamy:*

1. $x \in \Sigma_{n+1} \Leftrightarrow (\exists f \in F_{\mathbb{Q}}) x = \inf_{i \in \mathbb{N}} f(i)$;

2. $x \in \Pi_{n+1} \Leftrightarrow (\exists f \in F_{\mathbb{Q}}) x = \sup_{i \in \mathbb{N}} f(i)$;

3. $x \in \Delta_{n+1} \Leftrightarrow (\exists f \in F_{\mathbb{Q}})x = \lim_{i\to\infty} f(i)$ efektywnie*;*

4. $x \in \Delta_{n+1} \Leftrightarrow (\exists f \in F_{\mathbb{Q}})x = \lim_{i\to\infty} f(i)$*;*

Powyższy lemat wynika z twierdzeń zaprezentowanych w pracy [77].

Podamy teraz definicję prawostronnie i lewostronnie obliczalnej liczby rzeczywistej:

Definicja 3.8 [77] *Liczbę rzeczywistą x nazywamy lewostronnie (odp. prawostronnie) obliczalną, jeżeli istnieje rosnący (odp. malejący), obliczalny ciąg liczb wymiernych $\{r_n\}_{n\in\mathbb{N}}$ zbieżny do x.*

W świetle powyższej definicji można sformułować następujący wniosek:

Wniosek 3.1 [77] *Liczba rzeczywista x jest:*

- *lewostronnie obliczalna wtedy i tylko wtedy, gdy istnieje obliczalna funkcja $f : \mathbb{N} \to \mathbb{Q}$ taka, że $x = \sup_{i\in\mathbb{N}} f(i)$;*
- *prawostronnie obliczalna wtedy i tylko wtedy, gdy istnieje obliczalna funkcja $f : \mathbb{N} \to \mathbb{Q}$ taka, że $x = \inf_{i\in\mathbb{N}} f(i)$.*

Twierdzenie 3.4 [77] *Liczba rzeczywista x jest liczbą obliczalną wtedy i tylko wtedy, gdy jest liczbą lewostronnie i prawostronnie obliczalną.*

Dowód. Załóżmy, że $x \in \mathbb{R}$ jest liczbą rzeczywistą obliczalną. Zatem istnieje obliczalny ciąg liczb wymiernych, taki że

$$\forall(n \in \mathbb{N})(|x - r_n| < 2^{-n}).$$

Przyjmijmy:

$$l_n = r_n - 2^{-n+2} \text{ i } k_n = r_n + 2^{-n+2}.$$

Ciąg $\{l_n\}_{n\in\mathbb{N}}$ jest ciągiem rosnącym, a ciąg $\{k_n\}_{n\in\mathbb{N}}$ jest ciągiem malejącym, wynika z własności ciągu $\{r_n\}_{n\in\mathbb{N}}$, iż dla każdej liczby $n \in \mathbb{N}$, r_n należy do przedziału otwartego długości 2^{-n+1} o środku w punkcie x. Ponadto:

$$x = \lim_{n\to\infty} l_n = \lim_{n\to\infty} k_n.$$

Stąd x jest liczbą lewostronnie i prawostronnie obliczalną.

Załóżmy, że x jest liczbą rzeczywistą lewostronnie i prawostronnie obliczalną. Zatem istnieją obliczalne ciągi liczb wymiernych $\{l_n\}_{n\in\mathbb{N}}$ oraz $\{k_n\}_{n\in\mathbb{N}}$, takie że $\{l_n\}_{n\in\mathbb{N}}$ jest ciągiem rosnącym, a $\{k_n\}_{n\in\mathbb{N}}$ jest ciągiem malejący i

$$x = \lim_{n\to\infty} l_n \text{ oraz } x = \lim_{n\to\infty} k_n.$$

Przyjmijmy zatem

$$p_n = \min_{i\in\mathbb{N}}(k_i - l_i < 2^{-n}).$$

Takie p_n istnieje dla dowolnej liczby $n \in \mathbb{N}$, ponieważ $\lim_{n\to\infty}(k_n - l_n) = 0$. Ponadto dla dowolnej liczby $n \in \mathbb{N}$ $l_n < x$ i $k_n > x$. Zatem przyjmując $r_n = p_n$ otrzymamy:

$$\forall(n \in \mathbb{N})(|x - r_n| < 2^{-n}).$$

Stąd x jest liczbą rzeczywistą obliczalną.

□

Definicja 3.9 [77] *Liczbę rzeczywista x nazywamy obliczalnie aproksymowalną, jeżeli istnieje obliczalny ciąg liczb wymiernych $\{r_n\}_{n\in\mathbb{N}}$ zbieżny do x.*

Na podstawie definicji 3.7 zbiór liczb rzeczywistych obliczalnych jest tożsamy z klasami $\Delta_0, \Sigma_0, \Pi_0$. Wniosek 3.1 pokazuje, że zbiór liczb lewostronnie obliczalnych jest równy klasie Π_1, zaś zbiór liczb prawostronnie obliczalnych jest równy klasie Σ_1. Klasa $\Delta_1 = \Sigma_1 \cap \Pi_1$ i na podstawie twierdzenia 3.4 jest równa klasie liczb rzeczywistych obliczalnych oraz klasa Δ_2 na podstawie lematu 3.1 jest tożsama ze zbiorem liczb rzeczywistych obliczalnie aproksymowalnych. Wspomniane klasy zachowują również następujące własności:

$$\Delta_1 \subsetneq \Sigma_1 \cup \Pi_1 \subsetneq \Delta_2 \subsetneq \Sigma_2 \cup \Pi_2.$$

Twierdzenie 3.5 [77]

- *Dla wszystkich $n \in \mathbb{N}$, Δ_n jest zamknięta ze względu na operacje arytmetyczne takie jak dodawanie, odejmowanie, mnożenie i dzielenie.*
- *Jeżeli $f : \mathbb{R} \to \mathbb{R}$ jest obliczalną funkcją oraz $x \in \Delta_n$, wówczas $f(x) \in \Delta_n$.*

Powyższe twierdzenie można udowodnić poprzez indukcję względem n.

Rozważmy teraz rozstrzygalność podzbiorów liczb rzeczywistych. W zbiorze liczb naturalnych zbiór nazywamy rozstrzygalnym, jeżeli jego funkcja charakterystyczna $c_A : \mathbb{N} \to \mathbb{N}$ jest obliczalna. Przypuśćmy, że zbiór liczb rzeczywistych będziemy nazywać rozstrzygalnym wtedy i tylko wtedy, gdy jego funkcja charakterystyczna $c_A : \mathbb{R} \to \mathbb{R}$ będzie obliczalną funkcją zmiennej rzeczywistej. Funkcja charakterystyczna ze swej natury jest nieciągła, poza przypadkiem zbiorów $A = \emptyset, A = \mathbb{R}$, zatem zgodnie z definicją 3.6 nie może być obliczalną funkcją zmiennej rzeczywistej. Stąd taka definicja zbiorów rozstrzygalnych nie byłaby dobra.

Zdefiniujmy funkcję dystansu $d_A : \mathbb{R}^n \to \mathbb{R}$ w następujący sposób:

$$d_A(x) = \inf_{y \in A} |y - x|,$$

jako ogólniejszą postać funkcji charakterystycznej.

Definicja 3.10 [76] *Zbiór $A \subseteq \mathbb{R}^n$ nazywamy rozstrzygalnym, jeżeli jego funkcja dystansu $d_A : \mathbb{R}^n \to \mathbb{R}$ jest obliczalną funkcją zmiennej rzeczywistej.*

Takie proste zbiory jak przedziały $[a, b] \subseteq \mathbb{R}$ o obliczalnych końcach a i b lub zbiór $\{(x, y) \in \mathbb{R} : x \geqslant y\}$ są rozstrzygalne.

Jak wiadomo podzbiór $U \subseteq \mathbb{R}$ jest otwarty wtedy i tylko wtedy, gdy jest sumą otwartych przedziałów o wymiernych końcach. W celu wprowadzenia pojęcia zbioru rekurencyjnie przeliczalnego w zbiorze liczb rzeczywistych skojarzmy zbiór otwarty $U \subseteq \mathbb{R}$ z ciągiem $(I_0, I_1, \ldots)$ otwartych przedziałów o wymiernych końcach takich, że $U = I_0 \cup I_1 \cup \ldots$. Ciąg $(I_0, I_1, \ldots)$ będziemy określali jako deskrypcję zbioru U. W ogólnym przypadku, dla $U \subseteq \mathbb{R}^n$, każdy I_k rozumiemy jako iloczyn kartezjański n otwartych przedziałów o wymiernych końcach.

Definicja 3.11 [76] *Zbiór $U \subseteq \mathbb{R}^n$ nazywamy rekurencyjnie przeliczalnym, jeżeli istnieje jego obliczalna deskrypcja.*

Przykładami podzbiorów rekurencyjnie przeliczalnych zbioru liczb rzeczywistych są, np. przedziały otwarte $(a, b) \subseteq \mathbb{R}$ o obliczalnych końcach, zbiór $\{(x, y) \in \mathbb{R} : x < y\}$.

3.2 Maszyna Turinga a rekurencyjne funkcje rzeczywiste

Dotąd rozważaliśmy jak funkcje zmiennej rzeczywistej wyrazić w dyskretnej strukturze maszyny Turinga. Teraz przedstawimy analizę odwrotną, jak dyskretną maszynę Turinga wyrazić poprzez funkcje zmiennej rzeczywistej.

Przedmiotem dyskusji w tej części pracy będzie symulacja maszyn Turinga przy pomocy rekurencyjnych funkcji rzeczywistych. Do tego celu, jako narzędzie pośrednie, wykorzystane zostanie odwzorowanie przesuwne GS, wprowadzone przez C. Moore'a w pracy [45]. Przedyskutowane zostaną również relacje między wymiarem funkcji symulującej a klasami η-hierarchii dla rekurencyjnych funkcji rzeczywistych. Paragraf ten będzie oparty na własnej pracy [57].

Do celów prezentowanej symulacji przyjmiemy następujący opis konfiguracji maszyny Turinga: $\alpha_1 q \alpha_2$, gdzie $q \in Q$ jest bieżącym stanem maszyny Turinga, a $\alpha_1\alpha_2$ jest dwustronnie nieskończonym napisem nad alfabetem Σ zapisanym na taśmie. Alfabet Σ zawiera również symbol pusty, a napis $\alpha_1\alpha_2$ zawiera tylko skończoną ilość symboli różnych od symbolu pustego. Przyjmiemy również, że w jednym ruchu maszyna Turinga może przesuwać głowicę tylko w prawo $(+1)$ lub w lewo (-1), natomiast nie może jej pozostawić w miejscu, bez ruchu oraz ma tylko jeden stan finalny, kończący działanie maszyny. Takie ograniczenia oczywiście w niczym nie zmniejszą możliwości obliczeniowych maszyny Turinga. Przypuśćmy teraz, że głowica maszyny Turinga obserwuje pierwszy symbol łańcucha α_2. Nasza maszyna Turinga będzie działała w następujący sposób. Niech

$$\ldots x_{-n} \ldots x_{-1} q x_0 \ldots x_n \ldots$$

będzie bieżącym opisem maszyny Turinga. Dla $\delta(q, x_0) = (q', x_0', +1)$ mamy następującą zmianę w opisie naszej maszyny:

$$\ldots x_{-n} \ldots x_{-1} q x_0 \ldots x_n \ldots \rightarrow \ldots x_{-n} \ldots x_{-1} x_0' q' x_1 \ldots x_n \ldots,$$

natomiast dla $\delta(q, x_0) = (q', x_0', -1)$ mamy

$$\ldots x_{-n} \ldots x_{-1} q x_0 \ldots x_n \ldots \rightarrow \ldots x_{-n} \ldots x_{-2} q' x_{-1} x_0' \ldots x_n \ldots.$$

Tak działającą maszynę Turinga będziemy symulować używając zdefiniowanego przez C. Moore'a odwzorowania przesuwnego GS [45]. Jest to odwzorowanie działające na dwustronnie nieskończonych napisach nad alfabetem

Σ. Niech $a = \ldots a_{-2}a_{-1}.a_1a_2a_3\ldots$ będzie pewnym nieskończonym napisem nad alfabetem Σ. Definiujemy odwzorowanie przesuwne GS w następujący sposób:

$$\Phi : a \to \partial^{f(a)}(a + g(a)).$$

Dziedziną jak również zbiorem wartości odwzorowania Φ jest zbiór dwustronnie nieskończonych napisów nad alfabetem Σ, natomiast $g : \Sigma^* \to \mathbb{R}$ jest odwzorowaniem a na skończony napis oraz $f : \Sigma^* \to \{-1, +1\}$, jest odwzorowaniem a na liczbę całkowitą. Ponadto wymaga się aby f i g były zależne od skończonego ciągu znaków napisu a. Będziemy ten ciąg znaków nazywać dziedziną zależności (ang. *Domain of Dependence*, DOD). Zapis $a + g(a)$ będziemy rozumieć jako zastąpienie skończonego ciągu znaków (DOD) w napisie a przez ciąg $g(a)$. Operator ∂ oznacza przesunięcie ciągu znaków w lewo lub w prawo w zależności od wartości $f(a)$. Gdy $f(a) = +1$ przesuwamy znaki o jedną pozycję w lewo względem kropki, gdy $f(a) = -1$ przesuwamy znaki o jedna pozycję w prawo względem kropki.

Tak zdefiniowane odwzorowanie przesuwne GS może być wykorzystane do symulowania działania maszyny Turinga. Aby otrzymać odwzorowanie przesuwne GS symulujące maszynę Turinga musimy odpowiednio zdefiniować przekształcenie g. Wprowadźmy sposób kodowania napisów nad Σ, które są zapisywane na taśmie. Niech zbiór symboli zapisywanych na taśmie w tym symbol oznaczający komórkę pustą $\Sigma = \{s_0, s_1, \ldots, s_m\}$, zbiór stanów $Q = \{q_0, q_1 \ldots, q_k\}$. Symbole ze zbiorów Σ oraz Q będziemy kodować kolejnymi cyframi systemu pozycyjnego o podstawie $n = \max\{m, k\} + 1$. Zarówno symbole taśmowe jak i stany możemy kodować tymi samymi cyframi. Taśmę maszyny Turinga znajdującej się w stanie q zakodujemy jako:

$$a = \ldots a_{-2}a_{-1}.a_0a_1a_2\ldots,$$

gdzie a_0 odpowiada stanowi q oraz a_i dla $i \in \mathbb{Z}$, $i \neq 0$ symbolom z $\sum$ zapisanym na taśmie, odpowiednio na lewo i na prawo od głowicy, a_1 jest symbolem pod głowicą oraz a_{-1} pierwszym symbolem na lewo od głowicy. Niech

$$DOD(a) = a_{-1}.a_0a_1$$

będzie skończonym ciągiem z a, który będzie obszarem przekształceń dla Φ.

Definicja 3.12 *Niech g będzie zdefiniowane następująco:*

$$g(a) = \begin{cases} a_{-1}.a'_1a'_0, & \text{dla} \quad f(a) = +1 \\ a'_0.a_{-1}a'_1, & \text{dla} \quad f(a) = -1 \end{cases}, \tag{3.1}$$

gdzie a'_1 *(nowy symbol) oraz* a'_0 *(nowy stan) są określone odpowiednio przez funkcje przejścia maszyny Turinga* $\delta(a_0, a_1) \to (a'_0, a'_1, r)$ *zaś* $f(a)$ *– definiuje ruch głowicy po taśmie*: $+1$ *w prawą stronę,* -1 *w lewą stronę.*

Lemat 3.2 *Odwzorowanie przesuwne GS z* g *i* f *zdefiniowanymi jak powyżej, symuluje maszynę Turinga.*

Przedstawimy teraz przykład i kilka słów ilustrujących lemat 3.2.

Przykład 3.4 Konstrukcja odwzorowania przesuwnego GS dla problemu binarnego następnika.

Tabela 3.1: Funkcja przejścia δ dla maszyny Turinga z przykładu 3.4.

Lp.	(Q, Σ)	$(Q, \Sigma, \{+1, -1\})$
1	$(0, 1)$	$(0, 1, +1)$
2	$(0, 0)$	$(0, 0, +1)$
3	$(0, 2)$	$(1, 2, -1)$
4	$(1, 1)$	$(1, 0, -1)$
5	$(1, 0)$	$(2, 1, -1)$
6	$(1, 2)$	$(2, 1, -1)$
7	$(2, 1)$	$(2, 1, -1)$
8	$(2, 0)$	$(2, 0, -1)$
9	$(2, 2)$	$(3, 2, +1)$

Posłużymy się do tego celu maszyną $M_{\mathfrak{s}}$ z paragrafu 2.1.3. Niech $\Sigma = \{0, 1, 2\}$, 2 reprezentuje symbol pusty, $Q = \{0, 1, 2, 3\}$, 0 jest stanem startowym, 3 stanem finalnym a 1 i 2 stanami modyfikacji (maszyna Turinga czyta po kolei symbole od prawej do lewej i dokonuje potrzebnych zmian). Zbiory Σ i Q mają trzy identyczne elementy ale elementy te mają różne znaczenie.

Funkcja przejścia maszyny $M_{\mathfrak{s}}$, δ oraz odwzorowanie przesuwne GS pokazują odpowiednio tabele 3.1 oraz 3.2.

W naszym przykładzie na taśmie maszyny $M_{\mathfrak{s}}$ są zapisywane trzy różne symbole. W definicji przekształcenia g dodatkowo, oprócz stanu i symbolu pod głowicą, musimy wziąć pod uwagę symbol poprzedzający symbol obserwowany przez głowicę, dlatego możemy otrzymać trzy różne sytuacje odpowiadające jednemu przejściu funkcji δ. Jest tak dlatego, że dla jednego przejścia

funkcji δ może pojawić się, ze względu na to jaki mamy symbol na lewo od głowicy, konieczność zdefiniować g i f dla trzech różnych argumentów. Funkcja przejścia δ zależy od a_0 (bieżący stan) i a_1 (symbol pod głowicą)

Tabela 3.2: Odwzorowanie przesuwne GS dla δ z tabeli 3.1

	Lp.	$a_{-1}.a_0a_1$	$g(a)$	$f(a)$
I	1	2.01	2.10	+1
	2	1.01	1.10	+1
	3	0.01	0.10	+1
II	4	2.00	2.00	+1
	5	1.00	1.00	+1
	6	0.00	0.00	+1
III	7	1.02	1.12	−1
	8	0.02	1.02	−1
IV	9	2.11	1.20	−1
	10	1.11	1.10	−1
	11	0.11	1.00	−1
V	12	2.10	2.21	−1
	13	1.10	2.11	−1
	14	0.10	2.01	−1
VI	15	2.12	2.21	−1
VII	16	2.21	2.21	−1
	17	1.21	2.11	−1
	18	0.21	2.01	−1
VIII	19	2.20	2.20	−1
	20	1.20	2.10	−1
	21	0.20	2.00	−1
IX	22	2.22	2.23	+1

natomiast dla g i f, $DOD(a) = a_{-1}.a_0a_1$ czyli g i f zależą dodatkowo od symbolu leżącego na lewo od głowicy: a_{-1}. Kolejne instrukcje funkcji przejścia δ odpowiadają instrukcjom w poszczególnych sekcjach odwzorowania przesuwnego GS (tabela 3.2). Pewne sytuacje na taśmie są niemożliwe, dlatego w sekcji III, VI oraz IX mamy mniej niż trzy instrukcje. Sekcja III odpowiada trzeciemu przejściu z tabeli 3.1. Jest to przejście zdefiniowane dla

stanu 0 (maszyna Turinga mija symbole na taśmie od lewej do prawej bez ich modyfikacji) i symbolu pod głowicą 2. W tym przypadku pojawienie się 2 z lewej strony głowicy jest niemożliwe. Taka sytuacja oznaczałaby, że wejście maszyny Turinga było puste. Dlatego mamy tutaj tylko dwie instrukcje. Podobnie w sekcjach VI i IX mamy tylko jedną możliwą sytuację, są to momenty gdy głowica znajduje się na lewo od niepustych komórek taśmy, więc możliwy symbol na lewo od głowicy to tylko symbol pusty.

Zilustrujemy teraz jak odwzorowanie przesuwne GS symuluje działanie maszyny Turinga. Pojedynczy krok obliczeń będzie zapisywany następująco:

$$\ldots a_{-2}a_{-1}.a_0a_1a_2\ldots \xrightarrow{g} \ldots a_{-2}g(\ldots a_{-1}.a_0a_1\ldots)a_2\ldots \xrightarrow{f} \ldots a'_{-2}a'_{-1}.a'_0a'_1a'_2\ldots$$

Niech taśma wejściowa maszyny Turinga M wygląda następująco:

$$\ldots 222.1101222\ldots,$$

stan startowy to $q = 0$, wówczas startujemy z:

$$a = \ldots a_{-2}a_{-1}.qa_1a_2\ldots = \ldots 222.01101222\ldots,$$

$$DOD(a) = 2.01.$$

Kolejne kroki odwzorowania przesuwnego GS są następujące:

$$\begin{array}{lll}
\ldots 222.01101222\ldots & \xrightarrow{g(\ldots 2.01\ldots)} & \ldots 222.10101222\ldots \\
\ldots 2221.0101222\ldots & \xrightarrow{g(\ldots 1.01\ldots)} & \ldots 2221.1001222\ldots \\
\ldots 22211.001222\ldots & \xrightarrow{g(\ldots 1.00\ldots)} & \ldots 22211.001222\ldots \\
\ldots 222110.01222\ldots & \xrightarrow{g(\ldots 0.01\ldots)} & \ldots 222110.10222\ldots \\
\ldots 2221101.0222\ldots & \xrightarrow{g(\ldots 1.02\ldots)} & \ldots 2221101.1222\ldots \\
\ldots 222110.11222\ldots & \xrightarrow{g(\ldots 0.11\ldots)} & \ldots 222111.00222\ldots \\
\ldots 22211.100222\ldots & \xrightarrow{g(\ldots 1.10\ldots)} & \ldots 22212.110222\ldots \\
\ldots 2221.2110222\ldots & \xrightarrow{g(\ldots 1.21\ldots)} & \ldots 2222.1110222\ldots \\
\ldots 222.21110222\ldots & \xrightarrow{g(\ldots 2.21\ldots)} & \ldots 222.21110222\ldots \\
\ldots 222.221110222\ldots & \xrightarrow{g(\ldots 2.22\ldots)} & \ldots 222.231110222\ldots \\
\ldots 222.31110222\ldots & &
\end{array}$$

Dla liczby 1101 zapisanej na taśmie w momencie startu maszyny $M_{\mathfrak{s}}$ otrzymujemy, w momencie osiągnięcia przez nią stanu finalnego $q = 3$, następujący kod taśmy $\ldots 222.31110222\ldots$ będący wynikiem przekształcenie

przesuwnego GS, co oznacza, że na taśmie maszyny $M_{\mathfrak{s}}$ mamy następujący zapis $\ldots 22111022\ldots$. Zatem wynik jej działania to 1110.

Przedstawimy teraz trzy własne wyniki dotyczące powyższej symulacji maszyny Turinga z pracy [57]. Zacznijmy od zaprezentowania funkcji $\mathbb{R} \to \mathbb{R}$ symulującej maszynę Turinga. Przyjmijmy, że dwustronnie nieskończony łańcuch:

$$a = \ldots a_{-2}a_{-1}.a_0a_1a_2\ldots$$

o podstawie kodowania równej n, przekształcamy na jednostronnie nieskończony łańcuch x_a postaci:

$$0.a_0a_{-1}a_1a_{-2}a_2a_{-3}\ldots,$$

wówczas $x_a \in [0,1)$.

Lemat 3.3 *Niech Φ będzie odwzorowaniem przesuwnym GS z $DOD(a) = a_{-1}.a_0a_1$. Istnieje funkcja $f_{GS} : \mathbb{R} \to \mathbb{R}$, taka że*

$$\Phi(a) = b \equiv f_{GS}(x_a) = x_b.$$

Dowód. W tym dowodzie wszystkie liczby są kodowane w systemie pozycyjnym o podstawie n. Niech a_0 – oznacza bieżący stan maszyny Turinga, a_1 – oznacza symbol znajdujący się pod głowicą oraz a_{-1} – oznacza pierwszy na lewo symbol od głowicy. Zgodnie z definicją GS zastępujemy elementy a_{-1}, a_0 oraz a_1 przez elementy $g(a)$, a następnie przesuwamy wszystkie cyfry w łańcuchu a zgodnie z wartością f. Z definicji 3.12 mamy:

$$g(a) = \begin{cases} a_{-1}.a'_1a'_0, & \text{dla} \quad f(a) = +1 \\ a'_0.a_{-1}a'_1, & \text{dla} \quad f(a) = -1 \end{cases}.$$

Jeżeli n jest podstawą systemu kodowania zaś $x_a = 0.a_0a_{-1}a_1a_{-2}a_2\ldots$, wówczas:

$$\lfloor x_a n^2 \rfloor - \lfloor x_a n \rfloor n = a_{-1},$$

zatem dla $f(a) = +1$ mamy:

$$g(a) = a_{-1}.a'_1a'_0 = \lfloor x_a n^2 \rfloor - \lfloor x_a n \rfloor n + \frac{a'_1}{n} + \frac{a'_0}{n^2}.$$

W łańcuchu a zastępujemy $a_{-1}.a_0a_1$ nowym łańcuchem $g(a)$, więc x_a zmieni się w następujący sposób:

$$0.a_0a_{-1}a_1a_{-2}a_2a_{-3}\ldots \to 0.a'_1a_{-1}a'_0a_{-2}a_2a_{-3}\ldots,$$

a po przesunięciu otrzymamy

$$x_b = 0.a'_0a'_1a_2a_{-1}a_3a_{-2}\ldots.$$

Mamy $g(a)n^2 = a_{-1}a'_1a'_0$ oraz $\lfloor g(a)n\rfloor n = a_{-1}a'_10$, gdzie $a_{-1}a_10$ oznacza liczbę o cyfrach a_{-1}, a'_1, 0 w systemie pozycyjnym o podstawie n, stąd

$$\frac{g(a)n^2 - \lfloor g(a)n\rfloor n}{n} = 0.a'_0.$$

Kontynuując ten proces mamy $\lfloor g(a)n\rfloor = a_{-1}a'_1$ oraz $\lfloor g(a)\rfloor n = a_{-1}0$, analogicznie jak poprzednio dostaniemy

$$\frac{\lfloor g(a)n\rfloor - \lfloor g(a)\rfloor n}{n^2} = 0.0a'_1.$$

Po obliczeniu sumy powyższych łańcuchów otrzymamy pierwsze trzy elementy x_b, tj. $0.a'_0a'_1$. Kolejne elementy są przesunięte o dwie pozycje: wszystkie parzyste elementy w prawo, a wszystkie nieparzyste w lewo. Łańcuch $0.000a_{-1}0a_{-2}0a_{-3}\ldots$ możemy otrzymać jako sumę szeregu:

$$\sum_{k=1}^{\infty} \frac{\lfloor x_a n^{2k}\rfloor - \lfloor x_a n^{2k-1}\rfloor n}{n^{2k+2}}$$

oraz drugi łańcuch $0.00a_20a_30a_4\ldots$ jako

$$\sum_{k=1}^{\infty} \frac{\lfloor x_a n^{2k+3}\rfloor - \lfloor x_a n^{2k+2}\rfloor n}{n^{2k+1}}.$$

Dlatego ostatecznie dla $f(a) = +1$ otrzymamy formułę:

$$f_{GS}(x_a) = \frac{g(a)n^2 - \lfloor g(a)n\rfloor n}{n} + \frac{\lfloor g(a)n\rfloor - \lfloor g(a)\rfloor n}{n^2}$$

$$+ \lim_{m\to\infty} \sum_{k=1}^{m} \frac{\lfloor x_a n^{2k}\rfloor - \lfloor x_a n^{2k-1}\rfloor n}{n^{2k+2}} + \lim_{m\to\infty} \sum_{k=1}^{m} \frac{\lfloor x_a n^{2k+3}\rfloor - \lfloor x_a n^{2k+2}\rfloor n}{n^{2k+1}}.$$

Dla $f(a) = -1$ mamy:

$$g(a) = a'_0.a_{-1}a'_1 = a'_0 + \frac{\lfloor x_a n^2\rfloor - \lfloor x_a n\rfloor n}{n} + \frac{a'_1}{n^2}.$$

W łańcuchu a zastępujemy $a_{-1}.a_0a_1$ nowym łańcuchem $g(a)$, więc x_a zmieni się w następujący sposób:

$$0.a_0a_{-1}a_1a_{-2}a_2a_{-3}\ldots \to 0.a_{-1}a_0'a_1'a_{-2}a_2a_{-3}\ldots,$$

a po przesunięciu:

$$x_b = 0.a_0'a_{-2}a_{-1}a_{-3}a_1'a_{-4}a_2a_{-5}a_3\ldots.$$

Zatem analogicznie jak w poprzednim przypadku dla $f(a) = -1$ otrzymamy wzór:

$$f_{GS}(x_a) = \frac{\lfloor g(a)\rfloor}{n} + \frac{g(a)n^2 - \lfloor g(a)n\rfloor n}{n^5} + \frac{\lfloor x_a n^2\rfloor - \lfloor x_a n\rfloor n}{n^3}$$
$$+ \lim_{m\to\infty}\sum_{k+1}^{m}\frac{\lfloor x_a n^{2k+3}\rfloor - \lfloor x_a n^{2k+2}\rfloor n}{n^{2k+5}} + \lim_{m\to\infty}\sum_{k=1}^{m}\frac{\lfloor x_a n^{2k+2}\rfloor - \lfloor x_a n^{2k+1}\rfloor n}{n^{2k}}.$$

□

Dzięki powyższemu wynikowi możemy sformułować następujące twierdzenie.

Twierdzenie 3.6 *Niech M będzie maszyną Turinga ze stanem startowym q_0 i wejściem ω oraz niech x_a będzie kodem startowej konfiguracji M, istnieje wówczas rzeczywista funkcja rekurencyjna $f_{GS} : \mathbb{R} \to \mathbb{R}$, należąca do klasy H_2, która symuluje M, tj. maszyna M zatrzymuje się po t krokach dla wejścia ω wtedy i tylko wtedy, gdy $f_{GS}^t(x_a) = x_b$, gdzie x_b odpowiada konfiguracji wynikowej maszyny M.*

Dowód. Z paragrafu 2.4.2 wiemy, że dodawanie, odejmowanie, mnożenie oraz potęga należą do klasy H_0 podczas gdy $\lfloor\rfloor$ jest w klasie H_1. Definiując $f_{GS}(x_a) = x_b$ używamy złożenia dwóch operacji $\lfloor\rfloor$. Mamy $\lfloor g(a)\rfloor$ która jest z klasy H_2 ponieważ

$$g(a) = \begin{cases} \lfloor x_a n^2\rfloor - \lfloor x_a n\rfloor n + \frac{a_1'}{n} + \frac{a_0'}{n^2}, & \text{dla} \quad f(a) = +1 \\ a_0' + \frac{(\lfloor x_a n^2\rfloor - \lfloor x_a n\rfloor n)}{n} + \frac{a_1'}{n^2}, & \text{dla} \quad f(a) = -1 \end{cases}$$

jest w H_1. Użyte przy powyższej konstrukcji f_{GS} granice istnieją i są skończone oraz należą do klasy H_2. Rozważmy granicę:

$$\lim_{m\to\infty}\sum_{k=1}^{m}\frac{\lfloor x_a n^{2k}\rfloor - \lfloor x_a n^{2k-1}\rfloor n}{n^{2k+2}} = \sum_{k=1}^{\infty}\frac{\lfloor x_a n^{2k}\rfloor - \lfloor x_a n^{2k-1}\rfloor n}{n^{2k+2}}.$$

Szereg

$$\sum_{k=1}^{\infty} \frac{\lfloor x_a n^{2k} \rfloor - \lfloor x_a n^{2k-1} \rfloor n}{n^{2k+2}}$$

jest zbieżny ponieważ jest ograniczony przez szereg zbieżny $\sum_{k=1}^{\infty} \frac{n-1}{n^{2k+2}}$. Zatem na podstawie twierdzenia Weierstrassa rozważany szereg jest zbieżny. Stąd

$$\lim_{m\to\infty} \sum_{k=1}^{m} \frac{\lfloor x_a n^{2k} \rfloor - \lfloor x_a n^{2k-1} \rfloor n}{n^{2k+2}}$$

z definicji 2.36 jest w klasie H_2. Pozostałe granice zachowują się podobnie.

□

Zaprezentujemy teraz symulację maszyny Turinga posługując się dwuwymiarową rekurencyjną funkcją rzeczywistą $\mathbb{R}^2 \to \mathbb{R}^2$. W tym celu będziemy reprezentować dwustronnie nieskończony łańcuch

$$a = \ldots a_{-2}a_{-1}.a_0a_1a_2\ldots$$

przez parę nieskończonych łańcuchów (x_a, y_a) postaci:

$$(0.a_0a_1a_2\ldots, 0.a_{-1}a_{-2}a_{-3}\ldots),$$

$(x_a, y_a) \in [0,1) \times [0,1)$.

Lemat 3.4 *Niech Φ będzie odwzorowaniem przesuwanym GS z $DOD(a) = a_{-1}.a_0a_1$. Wówczas istnieje funkcja $f_{GS} : \mathbb{R}^2 \to \mathbb{R}^2$, taka że:*

$$\Phi(a) = b \equiv f_{GS}(x_a, y_a) = (x_b, y_b).$$

Dowód. Zgodnie z definicją GS, (x_b, y_b) otrzymujemy z (x_a, y_a) poprzez zastąpienie symboli a_{-1}, a_0 oraz a_1 przez elementy będące wynikiem $g(a)$, a następnie przez przesunięcie w zależności od wartości funkcji f. Z definicji 3.12 mamy:

$$g(a) = \begin{cases} a_{-1}.a'_1a'_0, & \text{dla} \quad f(a) = +1 \\ a'_0.a_{-1}a'_1, & \text{dla} \quad f(a) = -1 \end{cases}.$$

Jeżeli n jest podstawą systemu kodowania zaś

$$(x_a, y_a) = (0.a_0a_1a_1\ldots, 0.a_{-1}a_{-2}a_{-3}\ldots),$$

wówczas:

$$\lfloor y_a n \rfloor = a_{-1},$$

zatem dla $f(a) = +1$ mamy:

$$g(a) = a_{-1}.a'_1 a'_0 = \lfloor y_a n \rfloor + \frac{a'_1}{n} + \frac{a'_0}{n^2}.$$

Teraz w łańcuchu x_a elementy $a_0 a_1$ powinny być zastąpione przez $a'_1 a'_0$ podczas gdy w łańcuchu y_a symbol a_{-1} pozostaje taki sam. Zatem mamy:

$$(0.a_0 a_1 a_2 \ldots, 0.a_{-1} a_{-2} a_{-3} \ldots) \to (0.a'_1 a'_0 a_2 \ldots, 0.a_{-1} a_{-2} a_{-3} \ldots)$$

i po przesunięciu:

$$(x_b, y_b) = (0.a'_0 a_2 a_3 \ldots, 0.a'_1 a_{-1} a_{-2} \ldots).$$

Mnożąc $\lfloor g(a)n \rfloor = a_{-1} a'_1$ przez n i odejmując od $g(a)n^2 = a_{-1} a'_1 a'_0$ otrzymamy a'_0,

$$\frac{g(a^q)n^2 - \lfloor g(a)n \rfloor n}{n} = 0.a'_0.$$

Stąd mamy

$$x_b = \frac{g(a)n^2 - \lfloor g(a)n \rfloor n}{n} + \frac{x_a n^2 - \lfloor x_a n^2 \rfloor}{n}.$$

Kontynuując podobne rozważania mamy $\lfloor g(a)n \rfloor = a_{-1} a'_a$ oraz $\lfloor g(a) \rfloor = a_{-1}$. Poprzez $\frac{\lfloor g(a)n \rfloor - \lfloor g(a) \rfloor n}{n}$ otrzymamy $0.a'_1$. Otrzymujemy poszukiwany łańcuch y_b :

$$y_b = \frac{\lfloor g(a)n \rfloor - \lfloor g(a) \rfloor n}{n} + \frac{y_a}{n}.$$

Dla $f(a) = -1$ mamy:

$$g(a) = a'_0.a_{-1} a'_1 = a'_0 + \frac{\lfloor y_a n \rfloor}{n} + \frac{a'_1}{n^2}.$$

W łańcuchu a zastępujemy $a_{-1}.a_0 a_1$ nowym łańcuchem $g(a)$, więc (x_a, y_a) zmienią się w następujący sposób:

$$(0.a_0 a_1 a_2 \ldots, 0.a_{-1} a_{-2} a_{-3} \ldots) \to (0.a_{-1} a'_1 a_2 a_3 \ldots, 0.a'_0 a_{-2} a_{-3} \ldots),$$

a po przesunięciu:

$$(x_b, y_b) = (0.a'_0 a_{-1} a'_1 a_2 a_3 \ldots, 0.a_{-2} a_{-3} \ldots).$$

Zatem otrzymamy następujące wzory:

$$x_b = \frac{\lfloor g(a) \rfloor}{n} + \frac{\lfloor y_a n \rfloor}{n^2} + \frac{g(a)n^2 - \lfloor g(a)n \rfloor n}{n^3} + \frac{x_a n^2 - \lfloor x_a n^2 \rfloor}{n^4}$$

oraz

$$y_b = y_a n - \lfloor y_a n \rfloor.$$

□

Powyższy wynik pozwala nam na sformułowanie następującego twierdzenia.

Twierdzenie 3.7 *Niech M będzie maszyną Turinga ze stanem startowym q_0 i wejściem ω oraz niech (x_a, y_a) będzie kodem startowej konfiguracji M, istnieje wówczas rzeczywista funkcja rekurencyjna $f_{GS} : \mathbb{R}^2 \to \mathbb{R}^2$, należąca do klasy H_2, która symuluje M, tj. maszyna M zatrzymuje się po t krokach dla wejścia ω wtedy i tylko wtedy, gdy $f_{GS}^t(x_a, y_a) = (x_b, y_b)$, gdzie (x_b, y_b) odpowiada konfiguracji wynikowej maszyny M.*

Dowód. Definiując $f_{GS}(x_a, y_a) = (x_b, y_b)$ użyliśmy złożenia dwóch operacji $\lfloor\rfloor$. Mamy $\lfloor g(a) \rfloor$, gdzie

$$g(a) = \begin{cases} \lfloor y_a n \rfloor + \frac{a_1'}{n} + \frac{a_0'}{n^2}, & \text{dla} \quad f(a) = +1 \\ a_0' + \frac{\lfloor y_a n \rfloor}{n} + \frac{a_1'}{n^2}, & \text{dla} \quad f(a) = -1 \end{cases}.$$

Zatem f_{GS} jest w klasie H_2.

□

Widzimy więc, że mimo zwiększenia wymiaru funkcji symulującej z 1 na 2, klasa funkcji symulującej w η-hierarchii się nie zmienia.

W celu zmniejszenia klasy funkcji symulującej w η-hierarchii zaproponujemy zwiększenie stopnia funkcji symulującej do 3. Jest to możliwe dzięki uproszczeniu konstrukcji symulacji odwzorowania przesuwnego GS przez rekurencyjną funkcję rzeczywistą. Aby skonstruować funkcję symulująca $\mathbb{R}^3 \to \mathbb{R}^3$, wprowadzimy reprezentację dwustronnie nieskończonego napisu

$$a = \ldots a_{-2} a_{-1} . a_0 a_1 a_2 \ldots$$

jako trzy elementy (x_a, y_a, z_a)(gdzie x_a, y_a to nieskończone ciągi, zaś z_a to jedna cyfra - symbol znajdujący się na lewo od bieżącego położenia głowicy) w następujący sposób:

$$(0.a_0a_1a_2\ldots, 0.a_{-2}a_{-3}a_{-4}\ldots, a_{-1}).$$

Opis maszyny Turinga dany jak powyżej jako trzy oddzielne elementy, pozwoli nam na wyznaczenie wartości $g(a)$ bez użycia $\lfloor\rfloor$, co pozwoli na zmniejszenie klasy funkcji symulującej działanie maszyny Turinga w η-hierarchi. Niestety nie uda nam się jeszcze całkowicie wyeliminować funkcji $\lfloor\rfloor$ przy wyznaczaniu (x_b, y_b, z_b).

Lemat 3.5 *Niech Φ będzie odwzorowaniem przesuwnym GS z $DOD(a) = a_{-1}.a_0a_1$. Wówczas istnieje funkcja $f_{GS} : \mathbb{R}^3 \to \mathbb{R}^3$, taka że*

$$\Phi(a) = b \equiv f_{GS}(x_a, y_a, z_a) = (x_b, y_b, z_b).$$

Dowód. Podobnie jak w lematach 3.3 i 3.4 funkcja f_{GS} powinna zastąpić a_{-1}, a_0 oraz a_1 cyframi wyznaczonymi przez $g(a)$ a następnie przesunąć cyfry w łańcuchu $\ldots a_{-2}a_{-1}.a_0a_1a_2\ldots$ w prawo lub w lewo względem kropki, odpowiednio do wyniku funkcji f. Z definicji 3.12 mamy:

$$g(a) = \begin{cases} a_{-1}.a'_1a'_0, & \text{dla} \quad f(a) = +1 \\ a'_0.a_{-1}a'_1, & \text{dla} \quad f(a) = -1 \end{cases}.$$

Jeżeli n jest podstawą systemu kodowania zaś

$$(x_a, y_a, z_a) = (0.a_0a_1a_1\ldots, 0.a_{-1}a_{-2}a_{-3}\ldots, a_{-1}),$$

wówczas dla $f(a) = +1$ mamy:

$$g(a) = a_{-1}.a'_1a'_0 = z_a + \frac{a'_1}{n} + \frac{a'_0}{n^2}.$$

W łańcuchu a zastępujemy $a_{-1}.a_0a_1$ nowym łańcuchem $g(a)$, więc (x_a, y_a, z_a) zmieni się następująco:

$$(0.a_0a_1a_2\ldots, 0.a_{-2}a_{-3}a_{-4}\ldots, a_{-1}) \to (0.a'_0a'_1a_2\ldots, 0.a_{-2}a_{-3}a_{-4}\ldots, a_{-1}),$$

a po przesunięciu otrzymamy:

$$(x_b, y_b, z_b) = (0.a'_0a_2a_3\ldots, 0.a_{-1}a_{-2}a_{-3}\ldots, a'_1).$$

Zatem otrzymamy następujące wzory:

$$\begin{array}{rcl} x_b & = & \frac{g(a)n^2-\lfloor g(a)n\rfloor n}{n} + \frac{x_a-\lfloor x_a n^2\rfloor}{n^2}n, \\ y_b & = & \frac{z_a}{n} + \frac{y_a}{n}, \\ z_b & = & \lfloor g(a)n\rfloor - \lfloor g(a)\rfloor n. \end{array}$$

Zaś dla $f(a) = -1$ mamy:

$$g(a) = a'_0.a_{-1}a'_1 = a'_0 + \frac{z_a}{n} + \frac{a'_1}{n^2}.$$

W łańcuchu a zastępujemy $a_{-1}.a_0a_1$ nowym łańcuchem $g(a)$, więc (x_a, y_a, z_a) zmieni się następująco:

$$(0.a_0a_1a_2\ldots, 0.a_{-2}a_{-3}a_{-4}\ldots, a_{-1}) \to (0.a'_0a'_1a_2\ldots, 0.a_{-2}a_{-3}a_{-4}\ldots, a_{-1}),$$

a po przesunięciu otrzymamy:

$$(x_b, y_b, z_b) = (0.a'_0a_{-1}a_1a_2\ldots, 0.a_{-3}a_{-4}a_{-5}\ldots, a_{-2}).$$

Zatem otrzymamy następujące wzory:

$$\begin{array}{rcl} x_b & = & \frac{g(a)n^2-\lfloor g(a)n\rfloor n}{n} + \frac{z_a}{n^2} + \frac{x_a-\frac{\lfloor x_a n\rfloor}{n}}{n}, \\ y_b & = & (y_a - \frac{\lfloor y_a n\rfloor}{n}), \\ z_b & = & \lfloor y_a n\rfloor. \end{array}$$

□

Powyższy wynik pozwoli nam na sformułowanie następującego twierdzenia.

Twierdzenie 3.8 *Niech M będzie maszyną Turinga ze stanem startowym q_0 i wejściem ω oraz niech (x_a, y_a, z_a) będzie kodem startowej konfiguracji M, istnieje wówczas rzeczywista funkcja rekurencyjna $f_{GS} : \mathbb{R}^3 \to \mathbb{R}^3$, należąca do klasy H_1, która symuluje M, tj. maszyna M zatrzymuje się po t krokach dla wejścia ω wtedy i tylko wtedy, gdy $f^t_{GS}(x_a, y_a, z_a) = (x_b, y_b, z_b)$, gdzie (x_b, y_b, z_b) odpowiada konfiguracji wynikowej maszyny M.*

Dowód. Zauważmy, że

$$g(a) = \begin{cases} z_a + \frac{a'_1}{n} + \frac{a'_0}{n^2}, & \text{dla} \quad f(a) = +1 \\ a'_0 + \frac{z_a}{n} + \frac{a'_1}{n^2}, & \text{dla} \quad f(a) = -1 \end{cases}$$

jest w klasie H_0.

Zatem definiując $f_{GS}(x_a, y_a, z_a) = (x_b, y_b, z_b)$ używamy pojedynczych operacji $\lfloor\rfloor$ z klasy H_1, pozostałe operacje zaś są w klasie H_0, dlatego f_{GS} jest w klasie H_1.

□

W tej części przedstawiliśmy trzy własne metody symulacji maszyny Turinga za pomocą rekurencyjnych funkcji rzeczywistych. Wykorzystaliśmy do tego celu rozszerzoną wersję odwzorowania przesuwającego GS zaproponowanego przez C. Moore'a w pracy [45]. Określiliśmy również klasy η hierarchii, w których znajdują sie przedstawione symulacje. Jedno- i dwuwymiarowe symulacje są w klasie H_2, natomiast trójwymiarowa symulacja jest już w klasie H_1.

3.3 Inne klasy funkcji rzeczywistych a maszyna Turinga

Paragraf ten zajmuje się, podobnie jak poprzedni, symulacją maszyny Turinga w dziedzinie rzeczywistej. W pierwszej kolejności została przedstawiona symulacja uniwersalnej maszyny Turinga przez funkcję analityczną zdefiniowaną z funkcji elementarnych (dodawanie, mnożenie, funkcje trygonometryczne). Jest to wynik pochodzący z pracy P. Koirana, C. Moore'a [33]. W dalszej części tego paragrafu rozszerzono zaprezentowaną symulację na przypadek symulacji niedeterministycznej maszyny Turinga przez funkcję analityczną zdefiniowaną z funkcji elementarnych, własna praca [60]. Różni autorzy niezależnie pokazali, że skończenie wymiarowe, liniowe, sklejane odwzorowania mogą symulować maszyny Turinga. Takie metody są prezentowane m. in. w pracach [45, 57, 69], w niektórych przypadkach z dodatkowymi warunkami nakładanymi na funkcje symulujące. Wspomniane konstrukcje oparte sa na prostej zasadzie. Umieszczamy cyfry współrzędnych x i y punktu a odpowiednio na lewej i prawej, względem głowicy, części taśmy maszyny Turinga. Wówczas możemy przesuwać głowicę po taśmie poprzez dzielenie lub mnożenie x i y oraz pisać na taśmę poprzez dodawanie stałych do nich. Zatem dwa wymiary wystarczą dla samego odwzorowania, zaś trzy wymiary dla przeprowadzenia obliczeń ciągłych w czasie.

Jednakże funkcje częściowo liniowe nie są najlepsze z fizycznego punktu widzenia, mimo że takie odwzorowania mogą być gładkie ze względu na różniczkowanie (praca [45]), ponieważ wiele fizycznych systemów dynamicznych jest analitycznych (przynajmniej w świecie fizyki klasycznej). Satysfakcjonujące byłoby zatem znalezienie obliczeniowo uniwersalnej funkcji analitycznej, określonej na liczbach rzeczywistych, zbudowanej z funkcji elementarnych, która przekształcałaby liczby naturalne na zbiór $\{0, 1\}$. Taka funkcja została zaprezentowana w dalszej części tego paragrafu.

W pierwszej kolejności przedstawiamy jednowymiarowe funkcje analityczne zbudowane ze skończonej liczby wyrażeń trygonometrycznych, które mogą symulować maszynę Turinga z wykładniczym opóźnieniem, tzn. dla maszyny Turinga działającej w czasie t liczba iteracji funkcji symulującej potrzebnych do wyznaczenia wyniku działania maszyny Turinga rośnie wykładniczo względem t. Prezentowana metoda symulacji pochodzi z pracy P. Koirana, C. Moore'a [33] i dotyczy symulacji uniwersalnej maszyny Turinga.

Przytoczymy teraz definicję klasy funkcji analitycznych pochodzącą z pracy [33] i oznaczymy ją symbolem $\mathcal{U}_n$.

Definicja 3.13 *Niech $\mathcal{U}_n$ będzie klasą funkcji zdefiniowanych na $\mathbb{R}$, następująco $\mathcal{U}_n = [\mathbb{Q}, \pi, \mathrm{i}_i^n; +, -, \times, \sin]$.*

Następnie przedstawiamy konstrukcje dwuwymiarowej funkcji symulującej działanie maszyny Turinga. Wstępna wersja tego wyniku została przedstawiona w pracach [32, 46].

Na wstępie krótkie przypomnienie funkcji Collatza w klasycznej wersji problemu $3x + 1$, podanej w pracy [35].

Niech f będzie funkcją zdefiniowaną na liczbach naturalnych, następującej postaci:

$$f(x) = \begin{cases} x/2 & (x \text{ parzyste}) \\ 3x + 1 & (x \text{ nieparzyste}). \end{cases}$$

Możemy wówczas badać problem, czy dla konkretnego x istnieje takie t, że $f^t(x) = 1$? W terminach systemów dynamicznych, czy wszystkie liczby naturalne w trakcie kolejnych iteracji f wpadają w cykl $\{1, 4, 2\}$?

Możemy uogólnić ten problem następująco. Niech

$$f(x) = a_i x + b_i \quad \text{gdzie} \quad x = i \bmod p \tag{3.2}$$

dla pewnego p oraz stałych a_i, b_i dla $0 \leqslant i < p$. Każdą taką funkcję f nazywamy funkcją Collatza. J. H. Conway w pracy [13] pokazał, że w ogólności problem, czy dla pewnego t, $f^t(x) = 1$, jest nierozstrzygalny. Conway pokazał to przy pomocy maszyn Minsky'ego. Są to skończone automaty ze stosem (FSA), które mogą zmniejszać o jeden, zwiększać o jeden lub porównywać z zerem skończoną liczbę rejestrów.

Do naszej symulacji maszyny Turinga użyjemy takiego skończenie stanowego automatu o trzech rejestrach. Symulacja będzie przebiegała dwuetapowo. W pierwszym etapie zaprezentujemy jak taki automat może symulować maszynę Turinga natomiast w drugim przedstawimy funkcję, która symuluje jeden krok działania wspomnianego automatu. Niech L i R reprezentują lewą i prawą stronę taśmy (R zawiera symbol a_0, na którym znajduje się aktualnie głowica) oraz dodatkowo użyjemy trzeciego rejestru roboczego W. W szczególności jeżeli nasza maszyna Turinga ma k stanów oraz m symboli zapisywanych na taśmie to mamy

$$L = \sum_{i=1}^{\infty} m^{i-1} a_{-i} \text{ i } R = \sum_{i=0}^{\infty} m^i a_i.$$

Dla każdego stanu maszyny Turinga nasz automat FSA ma pętlę o m stanach, która zmniejsza za każdym obrotem R o jeden. W ostatnim z m obrotów zwiększa o jeden W. Gdy otrzymamy w pętli $R = 0$, wówczas stan FSA wskazuje nam na $a_0 = R \bmod m$ oraz wychodzimy z $W = \lfloor R/m \rfloor$.

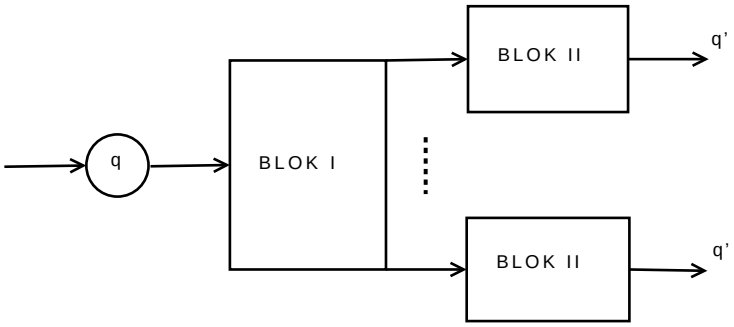
Jeżeli instrukcje maszyny Turinga przesuwają głowicę na prawo, wówczas z powrotem przepisujemy W do R poprzez powtarzanie $(\mathrm{dec}_W, \mathrm{inc}_R)$ dopóki nie otrzymamy $W = 0$, mnożymy L przez m w pętli $(\mathrm{dec}_L, m\mathrm{inc}_W)$ (gdzie $m\mathrm{inc}_W$ oznacza zwiększenie W, m razy) dopóki nie otrzymamy $L = 0$ i wówczas powtarzamy $(\mathrm{dec}_W, \mathrm{inc}_L)$ dopóki nie otrzymamy $W = 0$, po czym dodajemy nowy symbol a' do L poprzez $a'\mathrm{inc}_L$.

Jeżeli maszyna Turinga przesuwa głowice w lewo, najpierw ustalamy $R = m^2 W$ w pętli $(\mathrm{dec}_W, m^2\mathrm{inc}_R)$ dopóki nie otrzymamy $W = 0$, a następnie uruchamiamy pętlę o m stanach, która zmniejsza o jeden L. Opuszczamy pętlę z $W = \lfloor L/m \rfloor$ i znajomością $a_{-1} = L \bmod m$, więc dodajemy $a_{-1} + ma'$ do R. Na koniec przepisujemy W z powrotem do L powtarzając (dec_W, inc_L) dopóki nie otrzymamy $W = 0$.

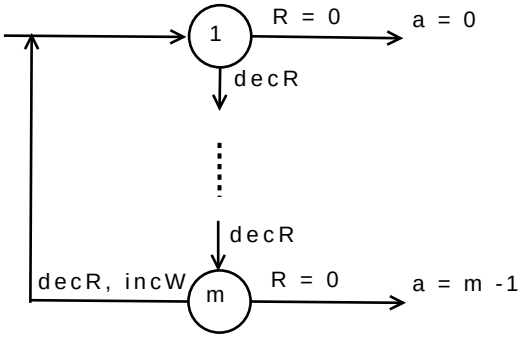
W obydwu przypadkach przechodzimy do części FSA odpowiadającej nowemu stanowi q'.

Wyjaśnimy teraz znaczenie poszczególnych bloków przedstawionego schematu FSA (rys. 3.2). Blok I (rys. 3.3) jest skonstruowany w celu wyznaczenia

symbolu znajdującego się aktualnie pod głowicą.



Rysunek 3.2: FSA symulujący działanie maszyny Turinga



Rysunek 3.3: Blok I. Wyznaczenie symbolu pod głowicą maszyny Turinga

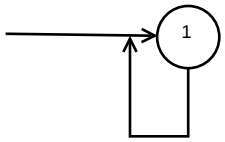
Rejestry na wyjściu Bloku I przyjmują następujące wartości $W = \lfloor R/m \rfloor$, $R = 0$. Ten blok ma możliwość skończenia działania w m różnych stanach, każdy z nich odpowiada innemu symbolowi zapisywanemu na taśmie. Blok kończy działanie w stanie odpowiadającemu symbolowi, który mamy aktualnie pod głowicą, a rejestry są przygotowane do zmiany symbolu pod głowicą.

W przypadku gdy funkcja przejścia maszyny Turinga ma postać $\delta(q, a) = (q', a', +1)$, co oznacza, że mamy zmienić symbol pod głowicą i wykonać ruch głowicy w prawo, kończymy działanie FSA Blokiem II (rys. 3.5).

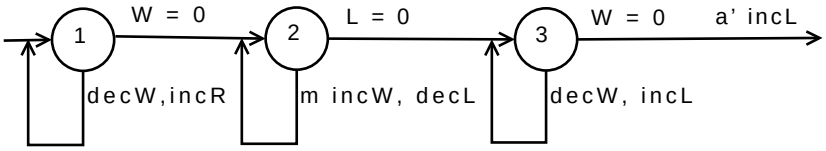
Na wyjściu Blok II zostawia rejestry równe odpowiednio (a' symbol dany przez funkcję przejścia dla symbolu pod głowicą a i stanu q): $L = mL + a'$, $R = \lfloor R/m \rfloor$, $W = 0$.

Jednakże działanie Bloku II w przypadku gdy funkcja przejścia maszyny Turinga ma postać $\delta(q, a) = (q', a', -1)$ jest bardziej skomplikowane. Przedstawimy je w dwóch częściach: Blok IIa (rys. 3.6) oraz Blok IIb (rys. 3.7).

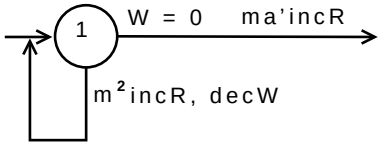
Na wyjściu Bloku IIa mamy rejestry równe odpowiednio: $R = m^2\lfloor R/m \rfloor + ma', W = 0$ oraz jesteśmy gotowi do ruchu głowicy w lewo. Zatem kończymy Blokiem IIb, który wyznacza symbol na lewo od głowicy i dodaje go do rejestru R. Ostatecznie otrzymujemy: $R = m^2\lfloor R/m \rfloor + ma' + i, L = \lfloor L/m \rfloor$, $W = 0$, gdzie i jest wyznaczonym, w Bloku IIb, symbolem. Blok II zarówno w przypadku ruchu głowicy w prawo jak i w lewo zmienia bieżący stan maszyny Turinga na odpowiadający nowemu stanowi maszyny Turinga q'.



Rysunek 3.4: Blok I w przypadku, gdy q jest stanem finalnym maszyny Turinga



Rysunek 3.5: Blok II. Realizacja ruchu głowicy w prawo i zmiana symbolu pod głowicą

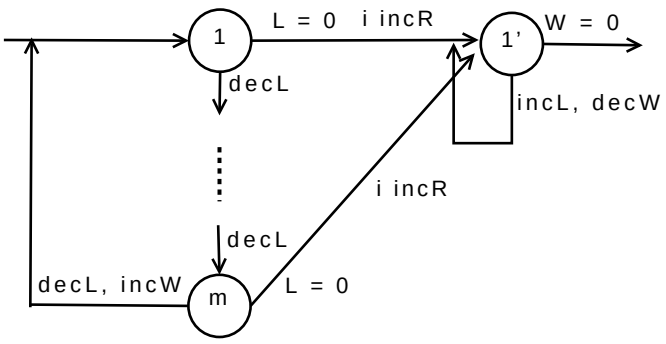


Rysunek 3.6: Blok IIa. Zmiany symbolu pod głowicą

W przypadku gdy stan q jest stanem finalnym maszyny Turinga Blok I przyjmuje postać jak na rys. 3.4 i automat zmienia stan na finalny czyli kończy swoje działanie.

Zatem potrzebujemy 4 stanów dla każdej pary stan/symbol, która przesuwa głowice w prawo, $m+3$ dla każdej pary, która przesuwa głowice w lewo i dwa dla par, które zatrzymują działanie maszyny Turinga.

Jeśli obliczenie maszyny Turinga zajmuje czas t, to może być ono symulowane przez FSA z trzema rejestrami w czasie $O(t(m^l))$. Każdy krok



Rysunek 3.7: Blok IIb. Ruch w lewo. Wyznaczenie symbolu na lewo od głowicy oraz zmiana rejestrów FSA odpowiednio dla ruchu głowicy w lewo

maszyny Turinga potrzebuje do symulacji $O(\max(L, R))$ kroków FSA, L i R są to wielkości rzędu $O(m^l)$, gdzie l jest długością taśmy wykorzystywanej przez maszynę Turinga.

Pokażemy teraz jak symulować 3-rejestrowy automat FSA poprzez funkcję analityczną zbudowaną z funkcji elementarnych klasy $\mathcal{U}_n$. Jeżeli FSA ma n stanów i znajduje się właśnie w stanie q, gdzie $0 \leqslant q < n$, definiujemy

$$x = 2^L 3^R 5^W n + q, \ x \in \mathbb{R}.$$

Wszystkie nasze operacje mogą być wyrażone poprzez proste przekształcenia na x. Na przykład, do zmniejszenia R, zwiększenia W m razy oraz zmiany stanu z q na q' wykonamy następujące przekształcenie x:

$$(\mathrm{dec}_R, m\mathrm{inc}_W) : f(x) = (5^m/3)(x - q) + q' = (5^m/3)x + (q' - (5^m/3)q).$$

Widzimy więc, że $f(x)$ ma postać funkcji Collatza dla $a = (5^m/3)$ oraz $b = q' - (5^m/3)q$. Ponadto możemy wyodrębnić q oraz sprawdzić równość z zerem rejestrów poprzez wartości jakie przyjmuje wyrażenie $x \bmod 30$, patrz tabela 3.3.

Tylko $19n$ wartości $x \bmod 30n$ potrzebujemy rozróżniać, pozostałe $11n$ nigdy nie wystąpią. Ponadto nie wszystkie kombinacje wartości niezerowych rejestrów L, R i W wystąpią w każdym stanie automatu FSA.

Funkcję Collatza zadaną wzorem (3.2) symulującą jeden krok działania automatu FSA możemy otrzymać z funkcji elementarnych klasy $\mathcal{U}_n$. Niech

$$h_p(x) = \left(\frac{\sin \pi x}{p \sin \frac{\pi x}{p}} \right)^2 = \begin{cases} 1 & x \bmod p = 0 \\ 0 & x \bmod p \neq 0 \end{cases}, \quad \text{dla całkowitych } x.$$

Tabela 3.3: Zestawienie wartości x mod $30n$ w zależności od wartości rejestrów FSA

rejestry	$i \quad (x \bmod 30n)$
$L>0, R>0, W>0$	q
$L=0, R>0, W>0$	$q+15n$
$L>0, R=0, W>0$	$q+10n,\ q+20n$
$L>0, R>0, W=0$	$q+6n,\ q+12n,\ q+18n,\ q+24n$
$L=0, R=0, W>0$	$q+5n,\ q+25n$
$L>0, R=0, W=0$	$q+2n,\ q+4n,\ q+8n,\ q+16n$
$L=0, R>0, W=0$	$q+3n,\ q+9n,\ q+21n,\ q+27n$
$L>0, R=0, W=0$	$q+n$

Ta funkcja jest analityczna na całym $\mathbb{R}$, może być ona rozwinięta w szereg. Przyjmując $y = \frac{\pi x}{p}$ możemy zapisać:

$$\frac{\sin py}{\sin y} = \frac{e^{ipy} - e^{-ipy}}{e^{iy} - e{-iy}} = \sum_{i=0}^{p-1} e^{iky} e^{-i(p-1-k)y}.$$

Zatem dzielenie nie jest nam potrzebne do konstrukcji funkcji h_p, możemy ją więc skonstruować w $\mathcal{U}_n$. Wówczas

$$f_{\mathcal{U}_n}(x) = \sum_{i=0}^{29} h_{30}(x-i)(a_i x + b_i)$$

odpowiada równaniu (3.2) dla liczb naturalnych. Ostatecznie, jeżeli nasz stan startowy to q_0, wejście dla maszyny Turinga zapiszemy tradycyjnie na taśmie po prawej stronie głowicy, to wejściu v maszyny Turinga odpowiada wartość $x = 3^v n + q_0$. Zatem mamy:

Twierdzenie 3.9 [33] *Dla dowolnej maszyny Turinga M o m symbolach taśmowych z pewnym wejściem v istnieje funkcja analityczna jednej zmiennej $f_{\mathcal{U}_n}$ oraz stałe n (liczba stanów FSA) i q_0 (stan startowy maszyny Turinga), takie że maszyna Turinga zatrzymuje się po t krokach z wynikiem y wtedy i tylko wtedy, gdy istnieje takie $t' \in \mathbb{N}$, że $f^{t'}_{\mathcal{U}_n}(x) = 2^{y_1} 3^{y_2} n + q_1$, gdzie $x = 3^v n + q_0$, $y = y_1 \times m^{\|y_2\|+y_2}$, q_1 odpowiada stanowi finalnemu maszyny Turinga oraz $t' = O(m^t)$.*

Na koniec przyjmijmy, że $q = 1$ jest stanem finalnym oraz określmy $a_i = 0$ i $b_i = 1$ dla wszystkich $i = 1 + ck$, wówczas koniec obliczeń maszyny Turinga będzie odpowiadał punktowi stałemu funkcji $f_{\mathcal{U}_n}$ dla $x = 1$. Dzięki temu możemy sprowadzić problem stopu maszyny Turinga do problemu iteracji funkcji Collatza i odwrotnie gdyż prawdziwe wówczas jest poniższe twierdzenie.

Twierdzenie 3.10 [33] *Dla dowolnej maszyny Turinga M o m stanach, wejściu v istnieje funkcja analityczna $f_{\mathcal{U}_n}$ jednej zmiennej oraz stałe n i q_0 takie, że M zatrzymuje się w czasie t wtedy i tylko wtedy, gdy $f^{t'}_{\mathcal{U}_n}(3^v n + q_0) = 1$ dla pewnego $t' = O(m^t)$.*

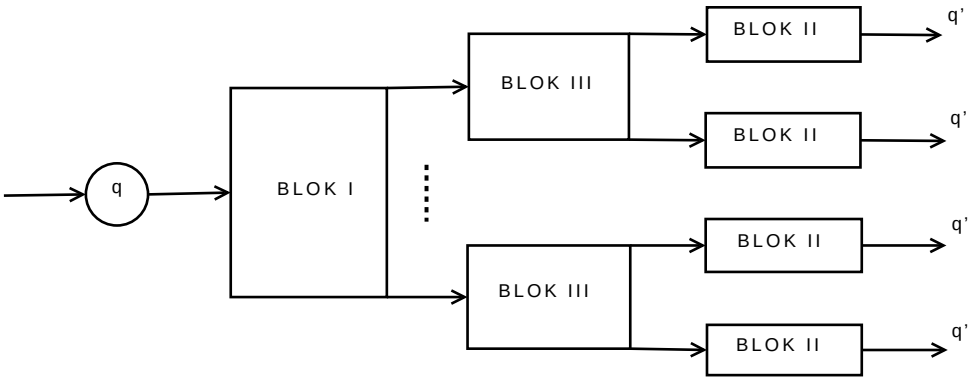
Przedstawimy teraz własny wynik, dotyczący podobnej symulacji, dla przypadku niedeterministycznej maszyny Turinga. Wariant takiej symulacji został zaprezentowany we własnej pracy [60]. Będziemy rozważać przypadek niedeterministycznej maszyny Turinga, dla której, w każdym kroku działania maszyny, co najwyżej dwie możliwości wyboru różnych wariantów obliczeń. Oznaczymy je przez 0 i 1. Takie ograniczenie nie ma istotnego wpływu na klasę problemów rozstrzygalnych przez niedeterministyczną maszynę Turinga [52]. Postępując podobnie jak w przypadku przedstawionym powyżej, użyjemy do symulacji niedeterministycznej maszyny Turinga o n stanach i m symbolach taśmowych, FSA z 4 rejestrami: L, R, G i W. Rejestry L, R oraz W będą miały identyczną funkcję jak poprzednio, natomiast rejestr G posłuży nam do przechowania wyborów niedeterministycznej maszyny Turinga w kolejnych krokach obliczeń.

$$G = \sum_{i=0}^{t-1} g_i 2^i, \;\; g_i \in \{0, 1\},$$

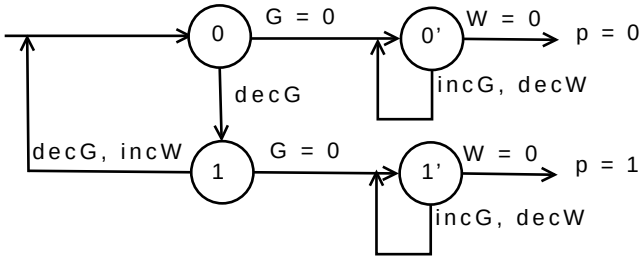
gdzie g_i oznacza wybór jednej z dwóch możliwych ścieżek obliczeń w $(i+1)$-szym kroku działania niedeterministycznej maszyny Turinga i przyjmuje wartość odpowiednio 0 lub 1. Zatem schemat FSA symulujący pewną niedeterministyczną maszynę Turinga przedstawia rys. 3.8.

Blok III (rys. 3.9) odpowiada za wyznaczenie wyboru jakiego w danym kroku dokonuje niedeterministyczna maszyna Turinga.

Zatem potrzebujemy 6 stanów dla każdej pary stan/symbol, która przesuwa głowicę w prawo, $m+5$ dla każdej pary, która przesuwa głowicę w lewo i dwa dla par, które zatrzymują działanie niedeterministycznej maszyny Turinga.



Rysunek 3.8: FSA symulujący działanie niedeterministycznej maszyny Turinga



Rysunek 3.9: Blok III. Wyznaczenie wyboru wariantu obliczeń niedeterministycznej maszyny Turinga

Jeśli obliczenie niedeterministycznej maszyny Turinga zajmuje czas t, to może być ono symulowane przez FSA z 4 rejestrami w czasie $O(t(m^l + 2^t))$. Każdy krok niedeterministycznej maszyny Turinga potrzebuje do symulacji $O(\max(L, R, G))$ kroków FSA, L i R są to wielkości rzędu $O(m^l)$, gdzie l jest długością taśmy wykorzystywanej przez niedeterministyczną maszynę Turinga, G jest wielkości 2^t.

Pokażemy teraz jak symulować 4-rejestrowy automat FSA przy pomocy funkcji klasy $\mathcal{U}_n$. Jeżeli FSA ma n stanów i znajduje się właśnie w stanie q, gdzie $0 \leqslant q < k$, definiujemy

$$x = 2^L 3^R 5^G 7^W n + q.$$

Wszystkie nasze operacje dokładnie tak jak poprzednio mogą być zakodowane w x. Również jak w poprzednim przypadku możemy wyodrębnić q oraz sprawdzić równość z zerem rejestrów poprzez wartości jakie przyjmuje wyrażenie $x \bmod p$, tym razem $p = 210n$ (patrz tabela 3.4).

Tabela 3.4: Zestawienie wartości $x \bmod 210n$ w zależności od wartości rejestrów FSA

rejestry	$i \quad (x \bmod 210n)$
$L>0, R>0, G>0, W>0$	q
$L=0, R>0, G>0, W>0$	$q+150n$
$L>0, R=0, G>0, W>0$	$q+70n,\ q+140n$
$L>0, R>0, G=0, W>0$	$q+42n,\ q+84n,\ q+126n,\ q+168n$
$L>0, R>0, G>0, W=0$	$q+30n,\ q+60n,\ q+90n,\ q+120n,$ $q+150n,\ q+180n$
$L=0, R=0, G>0, W>0$	$q+35n,\ q+175n$
$L=0, R>0, G=0, W>0$	$q+21n,\ q+63n,\ q+147n,\ q+189n$
$L=0, R>0, G>0, W=0$	$q+15n,\ q+45n,\ q+75n,\ q+135n,$ $q+165n,\ q+195n$
$L>0, R=0, G=0, W>0$	$q+14n,\ q+28n,\ q+56n,\ q+98n,\ q+112n,$ $q+154n,\ q+182n,\ q+196n$
$L>0, R=0, G>0, W=0$	$q+10n,\ q+20n,\ q+40n,\ q+50n,\ q+80n,$ $q+100n,\ q+110n,\ q+130n,\ q+160n,$ $q+170n,\ q+190n,\ q+200n$
$L>0, R>0, G=0, W=0$	$q+6kn\ ,\ q+12n,\ q+18n\ ,\ q+24n,\ q+36n,$ $q+48n,\ q+54n,\ q+66n,\ q+72n,\ q+78n,$ $q+96n,\ q+102n,\ q+108n,\ q+114n,$ $q+132n,\ q+138n,\ q+144n,\ q+156n,$ $q+162n,\ q+174n,\ q+186n,\ q+192n,$ $q+198n,\ q+204n$
$L=0, R=0, G=0, W>0$	$q+7n,\ q+49n,\ q+91n,\ q+133n$
$L=0, R=0, G>0, W=0$	$q+5n,\ q+25n,\ q+85n,\ q+125n,\ q+185n,$ $q+205n$
$L=0, R>0, G=0, W=0$	$q+3n,\ q+9n,\ q+27n,\ q+33n,\ q+39n,$ $q+51n,\ q+81n,\ q+87n,\ q+99n,\ q+117n,$ $q+141n,\ q+153n$
$L>0, R=0, G=0, W=0$	$q+2n,\ q+4n,\ q+8n,\ q+16n,\ q+32n,$ $q+46n,\ q+64n,\ q+92n,\ q+106n,$ $q+128n,\ q+158n,\ q+184n$

Potrzebujemy rozróżniać tylko $104n$ wartości x mod $210n$, pozostałe $106n$ wyników nigdy nie wystąpią. Jak w poprzednim przypadku nie wszystkie kombinacje wartości niezerowych rejestrów L, R, G i W wystąpią w każdym stanie automatu FSA.

Zatem otrzymamy jednowymiarową funkcję analityczną symulującą działanie niedeterministycznej maszyny Turinga. Jest to funkcja postaci:

$$f_{\mathcal{U}_n}(x) = \sum_{i=0}^{209} h_{210}(x-i)(a_i x + b_i),$$

która odpowiada równaniu (3.2) dla liczb naturalnych.

Jeżeli teraz przyjmiemy, że q_0 jest stanem startowym niedeterministycznej maszyny Turinga, g jest liczbą naturalną kodującą wybór niedeterministycznej maszyny Turinga w poszczególnych krokach oraz jeżeli wejście niedeterministycznej maszyny Turinga zapiszemy na taśmie na lewo od głowicy, wówczas wejściu v niedeterministycznej maszyny Turinga odpowiada wartość $x = 3^v 5^g n + q_0$. Możemy więc sformułować następujące twierdzenie, będące wynikiem powyższej konstrukcji.

Twierdzenie 3.11 *Dla dowolnej niedeterministycznej maszyny Turinga M o m symbolach taśmowych z pewnym wejściem v oraz pewnym wyborem g istnieje funkcja analityczna jednej zmiennej $f_{\mathcal{U}_n}$ oraz stałe n (liczba stanów FSA) i q_0 (stan startowy niedeterministycznej maszyny Turinga), takie że niedeterministyczna maszyna Turinga zatrzymuje się po t krokach z wynikiem y wtedy i tylko wtedy, gdy istnieje takie $t' \in \mathbb{N}$, że $f^{t'}_{\mathcal{U}_n}(x) = 2^{y_1} 3^{y_2} 5^g n + q_1$, gdzie $x = 3^v 5^g n + q_0$, $y = y_1 \times m^{\|y_2\|} + y_2$, q_1 odpowiada stanowi finalnemu niedeterministycznej maszyny Turinga oraz $t' = O(m^t + 2^t)$.*

Rozdział 4

EAC a klasyczne modele obliczeń

Kolejny rozdział będzie poświęcony modelowi EAC. Konkretnie zagadnieniu symulowania przez EAC obliczeń modeli klasycznych: maszyn Turinga i funkcji rekurencyjnych.

Definicja modelu EAC pojawiła się za sprawą L. Rubla w 1993 roku. Jest to model mało do tej pory przebadany pod kątem jego związków z innymi modelami obliczeń analogowych jak i związkami z klasycznymi modelami obliczeń. L. Rubel w swej pracy definiującej EACa [67] stawia dwa problemy dotyczące tego modelu. Mianowicie, czy EAC może być symulowany przez klasyczny komputer i czy klasyczny komputer może być symulowany przez EAC.

Pierwszy paragraf tego rozdziału odpowie nam na drugie pytanie i będzie poświęcony symulacji maszyny Turinga przez EAC, natomiast kolejny pokaże jak w modelu EAC generować zbiory i funkcje rekurencyjne.

4.1 Porównanie z maszyną Turinga

Jak już zapowiadaliśmy, w tej części pokażemy, że EAC może symulować maszynę Turinga, generując wyniki jej obliczeń z dowolną dokładnością. Zaprezentowana metoda symulacji pochodzi z własnej pracy [59] i opiera się na propozycji przedstawionej pierwotnie w pracy [6].

Na początku przytoczymy z [6] kilka lematów i funkcji, które posłużą naszej symulacji maszyny Turinga przy pomocy EACa. Dodatkowo zwrócimy

uwagę na fakt, że występujące w cytowanych wynikach funkcje i współczynniki są EAC obliczalne.

Lemat 4.1 [6] *Niech $n \in \mathbb{N}$ oraz niech $\varepsilon \in (0, 1/2)$. Wówczas istnieje współczynnik $\zeta_\varepsilon > 0$, taki że $\forall x \in [n - \zeta_\varepsilon, n + \zeta_\varepsilon] \Rightarrow |v(x) - n \bmod 10| \leqslant \varepsilon$, gdzie $v(x) = a_0 + a_5 \cos(\pi x) + \left(\sum_{j=1}^{4} a_j \cos\left(\frac{j\pi x}{5}\right) + b_j \sin\left(\frac{j\pi x}{5}\right)\right)$ jest jednostajnie ciągłą EAC obliczalną funkcją zdefiniowaną na $\mathbb{R}$ oraz $a_0, \ldots, a_5, b_1, \ldots, b_4$ są obliczalnymi współczynnikami, które możemy otrzymać jako rozwiązanie pewnego układu równań liniowych.*

Funkcja $v : \mathbb{R} \to \mathbb{R}$ z powyższego lematu jest analitycznym rozszerzeniem funkcji $g : \mathbb{N} \to \mathbb{N}$, zdefiniowanej następująco: $g(n) = n \bmod 10$, na zbiór liczb rzeczywistych. Z postaci funkcji v możemy łatwo stwierdzić, iż jest ona EAC obliczalna jako złożenie EAC obliczalnych funkcji: dodawania, mnożenia i funkcji trygonometrycznych. Współczynniki $a_0, \ldots, a_5, b_1, \ldots, b_4$ również możemy otrzymać w modelu EAC jako rozwiązanie pewnego układu równań liniowych.

Lemat 4.2 [6] *Niech $n \in \mathbb{Z}$ oraz niech $\varepsilon \in [0, 1/2)$. Wówczas istnieje współczynnik ściągający $\lambda_\varepsilon > 0$, taki że $(\forall \delta \in [-\varepsilon, \varepsilon])|\sigma(n + \delta) - n| \leqslant \lambda_\varepsilon \delta$, gdzie σ jest EAC obliczalną funkcją daną wzorem $\sigma(x) = x - 0.2 \sin(2\pi x)$.*[1]

Funkcja σ zmniejsza dystans pomiędzy liczbami całkowitymi, a liczbami z ich sąsiedztwa i jest potrzebna aby kontrolować błąd prezentowanej poniżej symulacji. Oczywiście jako złożenie różnicy, mnożenia i funkcji sinus, jest to funkcja EAC obliczalna.

Lemat 4.3 [6] *Niech $a \in \{0, 1\}$ oraz niech $\tilde{a}, y \in \mathbb{R}$ będą, takie że $|a - \tilde{a}| \leqslant \frac{1}{4}$ oraz $y > 0$. Wówczas istnieje EAC obliczalna funkcja $d_2 : \mathbb{R}^2 \to \mathbb{R}$, taka że $|a - d_2(\tilde{a}, y)| < \frac{1}{y}$, dana wzorem $d_2(x, y) = \frac{1}{\pi} \operatorname{arc\,tg}(4y(x - \frac{1}{2})) + \frac{1}{2}$.*

Lemat 4.4 [6] *Niech $a \in \{0, 1, 2\}$ oraz niech $\tilde{a}, y \in \mathbb{R}$ będą, takie że $|a - \tilde{a}| \leqslant \varepsilon$ oraz $y \geqslant 2$. Niech $p = \begin{cases} 0 & \varepsilon \leqslant \frac{1}{4} \\ \lceil -\frac{\log(4\varepsilon)}{\log \lambda_\varepsilon} \rceil & \varepsilon > \frac{1}{4} \end{cases}$. Wówczas istnieje EAC obliczalna funkcja $d_3 : \mathbb{R}^2 \to \mathbb{R}$, taka że $|a - d_3(\tilde{a}, y)| < \frac{1}{y}$, dana wzorem $d_3(x, y) = d_2((\sigma^{[p+1]}(x) - 1)^2, 3y)(2d_2(\frac{\sigma^{[p]}(x)}{2}, 3y) - 1) + 1$.*

[1] W dalszej części tego paragrafu będziemy zakładać, że $\varepsilon \in [0, 1/2)$ jest ustalone natomiast λ_ε jest odpowiadającym mu współczynnikiem ściągającym danym przez lemat 4.2.

Funkcje z powyższych dwóch lematów są kolejnymi, potrzebnymi nam do kontroli błędu symulacji maszyny Turinga, funkcjami zmniejszającymi dystans pomiędzy liczbami całkowitymi a liczbami z ich sąsiedztwa. Są to funkcje EAC obliczalne gdyż są one złożeniem podstawowych EAC obliczalnych funkcji analitycznych.

Podamy ostatni potrzebny nam lemat z pracy [6]:

Lemat 4.5 [6] *Jeżeli* $|a_i|, |\tilde{a}_i| \leqslant b$ *dla* $i = 1, \ldots, n$, *wówczas*

$$|a_1 \ldots a_n - \tilde{a}_1 \ldots \tilde{a}_n| \leqslant (|a_1 - \tilde{a}_1| + \ldots + |a_n - \tilde{a}_n|)b^{n-1}.$$

Mamy już wszystkie potrzebne nam funkcje aby móc kontrolować wielkość błędu jaki może nam się pojawić podczas generowania w modelu EAC wyniku obliczeń danej maszyny Turinga. Przejdziemy teraz do zaprezentowania schematu naszej symulacji [59].

Idea tej symulacji pochodzi od Graçy, Campagnolo i Bruescu, została ona przedstawiona w pracy [6].

Bieżącą konfigurację maszyny Turinga będziemy reprezentować jako trójkę $(x, y, z) \in \mathbb{N}^3$. Przyjmiemy również bez straty ogólności, iż maszyna Turinga używa tylko 10 symboli taśmowych, które będziemy kodować kolejnymi cyframi $1, 2, \ldots, 9$ oraz symbolu pustego $B = 0$. Łańcuch

$$\ldots BBBa_{-k}a_{-k+1} \ldots a_{-1}a_0a_1 \ldots a_nBBB \ldots$$

reprezentuje zawartość taśmy danej maszyny Turinga, gdzie $a_i \in \{0, 1, \ldots, 9\}$ oraz a_0 jest symbolem zapisanym pod głowicą. Przypomnijmy, że taśma maszyny Turinga zawiera zawsze tylko skończoną liczbę komórek niepustych otoczonych nieskończoną liczbą komórek zawierających symbole puste. Ponadto zakładamy, że maszyna Turinga używa m stanów, kodowanych kolejnymi liczbami $0, 1, \ldots, m$. W każdym przejściu głowica maszyny może wykonać ruch w lewo kodowany jako 0, ruch w prawo kodowany jako 2 lub pozostać w miejscu co będzie kodowane jako 1. Dla wygody przyjmiemy, że gdy maszyna Turinga osiągnie stan finalny, wówczas pozostanie w tej samej konfiguracji, czyli nie zmieni się jej stan, zawartość taśmy ani położenie głowicy. Przyjmijmy

$$y_1 = a_0 + a_1 10 + \ldots + a_n 10^n, y_2 = a_{-1} + a_{-2} 10 + \ldots + a_{-k} 10^{k-1}$$

oraz niech q będzie stanem skojarzonym z bieżącą konfiguracją maszyny Turinga. Trójka $(y_1, y_2, q) \in \mathbb{N}^3$ daje nam więc bieżącą konfigurację. Zatem funkcję przejścia $f_\delta : \mathbb{N}^3 \to \mathbb{N}^3$ będziemy rozumieć jako zestawienie następujących po sobie konfiguracji (y_1, y_2, q). Jako przykład przedstawionego powyżej sposobu opisu maszyny Turinga zaprezentujemy opis maszyny $M_{\mathfrak{s}}$, danej tabelą 2.1, z wejściem: 1001. Jak pamiętamy maszyna $M_{\mathfrak{s}}$ liczy następnik liczby zapisanej binarnie i wykorzystuje do tego celu 4 stany, zatem będziemy je kodować kolejnymi cyframi $0, 1, 2, 3$. Stan startowy zakodujemy cyfrą 0, a stan finalny cyfrą 3. Symbole zapisywane na taśmie to cyfry binarne 0 i 1 oraz symbol pusty B. Symbol pusty zawsze kodujemy cyfrą 0 zatem symbol 0 zakodujemy jako 1 oraz symbol 1 jako 2. Łańcuch reprezentujący zawartość taśmy maszyny $M_{\mathfrak{s}}$ z wejściem 1001 na starcie będzie wyglądał następująco:

$$\ldots 0002112000 \ldots .$$

Zatem konfiguracja startowa dla naszej maszyny to: $(2112, 0, 0)$, natomiast funkcję f_δ przedstawia tabela 4.1.

Tabela 4.1: Funkcja przejścia f_δ dla maszyny Turinga $M_{\mathfrak{s}}$

Lp.	(y_1, y_2, q)	(y_1', y_2', q')
1	$(2112, 0, 0)$	$(112, 2, 0)$
2	$(112, 2, 0)$	$(12, 12, 0)$
3	$(12, 12, 0)$	$(2, 112, 0)$
4	$(2, 112, 0)$	$(0, 2112, 0)$
5	$(0, 2112, 0)$	$(2, 112, 1)$
6	$(2, 112, 1)$	$(11, 12, 1)$
7	$(11, 12, 1)$	$(121, 2, 2)$
8	$(121, 2, 2)$	$(2121, 0, 2)$
9	$(2121, 0, 2)$	$(02121, 0, 2)$
9	$(02121, 0, 2)$	$(2121, 0, 3)$
9	$(2121, 0, 3)$	$(2121, 0, 3)$

Naszym celem będzie ustalenie, czy EAC potrafi symulować maszynę Turinga. Odpowiemy na to pytanie w dwóch krokach. Po pierwsze będziemy potrzebowali EACa symulującego jeden krok obliczeń maszyny Turinga, czyli EACa wyznaczającego z dowolnie zadaną dokładnością wartość $f_\delta(\bar{x})$ dla zadanego $\bar{x} \in \mathbb{R}^3$. Następnie potrzebna będzie EAC obliczalna funkcja, która

będzie iterowała symulację jednego kroku obliczeń, czyli EACa wyznaczającego $f_\delta^{[i]}(\bar{x})$ dla zadanego $\bar{x}$ oraz $i, \bar{x} \in \mathbb{R}^3, i \in \mathbb{N}$.

Zaczniemy od zacytowania twierdzenia z pracy [6].

Twierdzenie 4.1 [6] *Niech $f_\delta : \mathbb{N}^3 \to \mathbb{N}^3$ będzie funkcją przejścia dla maszyny Turinga M opartą na kodowaniu opisanym powyżej oraz niech $0 < \delta < \varepsilon < \frac{1}{2}$. Wówczas f_δ posiada analityczne rozszerzenie na zbiór liczb rzeczywistych $f_M : \mathbb{R}^3 \to \mathbb{R}^3$ odporne na błędy, w następującym sensie: dla wszystkich f, takich że $\|f - f_M\|_\infty \leqslant \delta$ oraz dla każdego $\tilde{\bar{x}}_0 \in \mathbb{R}^3$, takiego że $\|\tilde{\bar{x}}_0 - \bar{x}_0\| \leqslant \varepsilon$, gdzie $\bar{x}_0 \in \mathbb{N}^3$ reprezentuje konfigurację startową M, mamy*

$$\|f^{[j]}(\tilde{\bar{x}}_0) - f_\delta^{[j]}(\bar{x}_0)\|_\infty \leqslant \varepsilon.$$

Sformułujemy teraz twierdzenie prowadzące do głównego wyniku tego paragrafu.

Twierdzenie 4.2 *Funkcja f_M z twierdzenia 4.1 jest EAC obliczalna.*

Dowód. Pokażemy jak skonstruować f_M jako EAC obliczalną funkcję.

Niech (y_1, y_2, q) opisuje bieżącą konfigurację $\bar{x}_0$ maszyny M oraz niech $(\tilde{y_1}, \tilde{y_2}, \tilde{q})$ będzie przybliżeniem tej konfiguracji $\tilde{\bar{x}}_0$. Po pierwsze chcemy aby $\tilde{f}_M$ spełniała następujący warunek:

$$\|(y_1, y_2, q) - (\tilde{y_1}, \tilde{y_2}, \tilde{q})\|_\infty \leqslant \varepsilon \to \|f_\delta(y_1, y_2, q) - \tilde{f}_M(\tilde{y_1}, \tilde{y_2}, \tilde{q})\|_\infty \leqslant \varepsilon.$$

Niech a_0 będzie symbolem znajdującym się aktualnie pod głowicą maszyny M. Wówczas $v(y_1) = a_0$. Jeżeli $|y_1 - \tilde{y_1}| \leqslant \varepsilon$, wówczas $|a_0 - v \circ \sigma^{[l]}(\tilde{y_1})| \leqslant \varepsilon$ dla $l = \left\lceil \left| \frac{\log(\frac{\zeta_\varepsilon}{\varepsilon})}{\log(\lambda_\varepsilon)} \right| \right\rceil$, gdzie v i σ są odpowiednio funkcjami z lematu 4.1 oraz lematu 4.2. [2] Zatem $\tilde{y} = v \circ \sigma^{[l]}(\tilde{y_1})$ daje nam przybliżenie symbolu znajdującego się aktualnie pod głowicą. Podobnie $v \circ \sigma^{[l]}(\tilde{y_2})$ daje nam przybliżenie a_{-1} z błędem ograniczonym przez ε. Te przybliżenia mogą być wyznaczone dla danego l przez pewien EAC jako złożenie skończonej liczby funkcji EAC obliczalnych.

[2] $\lceil x \rceil = \min\{n \in \mathbb{Z} : x \leqslant n\}$.

Teraz wyznaczymy następny stan M. Przypomnijmy, że m oznacza liczbę stanów oraz $k = 10$ liczbę symboli taśmowych. Kolejny stan możemy wyznaczyć w następujący sposób:

$$\tilde{q}_{next} = \sum_{i=0}^{9}\sum_{j=1}^{m}\left(\prod_{r=0,r\neq i}^{9}\frac{\sigma^{[n]}(\tilde{y})-r}{i-r}\right)\left(\prod_{s=1,s\neq j}^{m}\frac{\sigma^{[n]}(\tilde{q})-s}{j-s}\right)q_{i,j}$$

dla $n = \left\lceil \frac{\log(10m^2b^{m+7}(m+8))}{-\log(\lambda_\varepsilon)} \right\rceil$, gdzie $b = \max\{9.5, m+\frac{1}{2}\}$ oraz $q_{i,j}$ jest następnym stanem określonym przez funkcję przejścia maszyny M dla symbolu pod głowicą i oraz stanu bieżącego j. Dla tak wybranego n mamy:

$$9|y-\sigma^{[n]}(\tilde{y})| + (m-1)|q-\sigma^{[n]}(\tilde{q})| \leqslant \frac{\varepsilon}{10m^2b^{m+7}}.$$

Stąd oraz z lematu 4.5 mamy:

$$|\tilde{q}_{next} - q_{next}| \leqslant \varepsilon.$$

Przekształcenie zwracające kolejny stan jest definiowane przez wielomianową interpolację z funkcją σ złożoną n razy, zatem może być wygenerowane przez EAC.

Aby wyznaczyć symbol, który ma być wpisany pod głowicą oraz ruch głowicy w danym przejściu użyjemy podobnych konstrukcji. Oznaczymy je jako $\tilde{s}_{next}$ oraz $\tilde{h}_{next}$.

Aby zaktualizować zawartość taśmy zdefiniujemy następujące wartości p_1, p_2, p_3, które będą aktualizować zawartość taśmy odpowiednio dla ruchu głowicy: w lewo, gdy zostaje w miejscu oraz dla ruchu w prawo. Niech $\tilde{\tilde{h}}$ będzie dodatkowym przybliżeniem $\tilde{h}_{next}$ określonym poniżej. Wówczas kolejna wartość dla y_1 może być wyznaczona następująco:

$$\tilde{y}_1^{next} = p_1\frac{1}{2}(1-\tilde{\tilde{h}})(2-\tilde{\tilde{h}}) + p_2\tilde{\tilde{h}}(2-\tilde{\tilde{h}}) + p_3\left(-\frac{1}{2}\right)\tilde{\tilde{h}}(1-\tilde{\tilde{h}}),$$

dla

$$\begin{aligned} p_1 &= 10(\sigma^{[p]}(\tilde{y}_1) + \sigma^{[p]}(\tilde{s}_{next}) - \sigma^{[p]}(\tilde{y})) + \sigma^{[p]} \circ v \circ \sigma^{[l]}(\tilde{y_2}), \\ p_2 &= \sigma^{[p]}(\tilde{y}_1) + \sigma^{[p]}(\tilde{s}_{next}) - \sigma^{[p]} \circ v \circ \sigma^{[l]}, \\ p_3 &= \tfrac{\sigma^{[p]}(\tilde{y}_1) - \sigma^{[p]}(\tilde{y})}{10}, \end{aligned}$$

gdzie $p \in \mathbb{N}$ jest dostatecznie duże. Przy wyznaczaniu p_1 problemem jest fakt, iż wartość ta zależy od $\tilde{y}_1$, która jest nieograniczona. Dlatego jeżeli przyjmujemy po prostu $\tilde{\tilde{h}} = \tilde{h}_{next}$, błąd wyrażenia $\frac{1}{2}(1-\tilde{\tilde{h}})(2-\tilde{\tilde{h}})$ jest wzmacniany

przy mnożeniu przez p_1. Aby wyznaczyć dostatecznie dobre $\tilde{\tilde{h}}$ proporcjonalne do y_1 korzystamy z funkcji d_3 i przyjmujemy:

$$\tilde{\tilde{h}} = d_3\left(\tilde{h}_{next}, 10000\left(\tilde{y}_1 + \frac{1}{2}\right) + 2\right).$$

Postępując podobnie przy wyznaczaniu p_2 i p_3 możemy podsumować, że $|\tilde{y}_1^{next} - y_1^{next}| < \varepsilon$. Ponadto p_1, p_2 i p_3 są wyznaczane jako złożenia EAC obliczalnych funkcji, zatem są również EAC obliczalne. Podobnie dla lewej części taśmy możemy zdefiniować $\tilde{y}_2^{next}$ w taki sposób, że $|\tilde{y}_2^{next} - y_2^{next}| < \varepsilon$.

Ostatecznie, $\tilde{f}_M : \mathbb{R}^3 \to \mathbb{R}^3$ definiujemy: $\tilde{f}_M(\tilde{y}_1, \tilde{y}_2, \tilde{q}) = (\tilde{y}_1^{next}, \tilde{y}_2^{next}, \tilde{q}_{next})$.

Niech $0 \leqslant \delta < \varepsilon$ oraz niech $i \in \mathbb{N}$ będą, takie że $\sigma^{[i]}(\varepsilon) \leqslant \varepsilon - \delta$. Zdefiniujmy $f_M = \sigma^{[i]} \circ \tilde{f}_M$. Wówczas, jeżeli $\bar{x}_0 \in \mathbb{N}^3$ jest konfiguracją startową maszyny M to mamy:

$$\|\tilde{\bar{x}}_0 - \bar{x}_0\|_\infty \leqslant \varepsilon \Rightarrow \|f_M(\tilde{\bar{x}}_0) - f_\delta(\bar{x}_0)\|_\infty \leqslant \varepsilon - \delta.$$

Zatem na mocy nierówności trójkąta, jeżeli $\|\tilde{\bar{x}}_0 - \bar{x}_0\| \leqslant \varepsilon$, wówczas dla $\|f(\tilde{\bar{x}}_0 - f_M(\bar{x}_0))\|_\infty \leqslant \delta$ mamy:

$$\|f(\tilde{\bar{x}}_0 - f_\delta(\bar{x}_0))\|_\infty \leqslant \|f(\tilde{\bar{x}}_0) - f_M(\tilde{\bar{x}}_0)\|_\infty + \|f_M(\tilde{\bar{x}}_0) - f_\delta(\bar{x}_0)\|_\infty \leqslant \delta + (\varepsilon - \delta) = \varepsilon.$$

Co kończy dowód dla $j = 1$ gdzie wszystkie elementy konstrukcyjne są EAC obliczalne. Dla $j > 1$ postępujemy indukcyjnie.

□

Powyższe twierdzenie dowodzi, że można w modelu EAC wygenerować z dowolną dokładnością jeden krok obliczeń maszyny Turinga.

A teraz przedstawimy główny, własny wynik tego paragrafu.

Twierdzenie 4.3 *Dla każdej maszyny Turinga M istnieje pewien EAC symulujący jej obliczenie z dowolnie małym błędem ε.*

Dowód. Dowód bazuje na konstrukcji pochodzącej z [6]. Niech $f_\delta : \mathbb{N}^3 \to \mathbb{N}^3$ będzie funkcją przejścia maszyny Turinga M. Do iteracji funkcji f_δ użyjemy pary funkcji kontrolujących ewolucję wartości dwóch zmiennych symulacyjnych z_1, z_2. Obie zmienne symulacyjne startują z wartościami bliskimi $\bar{x}_0$ dla $t = 0$.

Pierwsza zmienna jest iterowana podczas pierwszej połowy cyklu, w tym czasie wartość drugiej pozostaje w przybliżeniu taka sama, wartość jej pochodnej jest bliska zeru dzięki funkcji kontrolnej. Następnie, w drugiej połowie cyklu, pierwsza zmienna pozostaje utrzymana bez większych zmian podczas gdy wartość drugiej ulega zmianie tak, że jest bliska wartości pierwszej zmiennej w chwili $t = 1$, wówczas obie zmienne mają wartość bliską $f_\delta(\bar{x}_0)$.

Ten proces jest powtarzany dowolnie wiele razy, zachowując cały czas błąd obliczeń pod kontrolą dzięki temu, że istnieje analityczna funkcja odporna na błędy, która symuluje f_δ. Rozważmy następujący układ równań:

$$\begin{aligned} z_1' &= c_1(z_1 - f_M \circ \sigma^{[m]}(z_2))^3 f_1(t, z_1, z_2), \\ z_2' &= c_2(z_2 - \sigma^{[n]}(z_1))^3 f_2(t, z_1, z_2), \end{aligned}$$

z warunkami brzegowymi $z_1(0) = z_2(0) = \tilde{\bar{x}}_0$, gdzie

$$\begin{aligned} f_1(t, z_1, z_2) &= d_2\left(s(t), \tfrac{c_1}{\gamma}(z_1 - f_M \circ \sigma^{[m]}(z_2))^4 + \tfrac{c_1}{\gamma} + 10\right), \\ f_2(t, z_1, z_2) &= d_2\left(s(-t), \tfrac{c_2}{\gamma}(z_2 - \sigma^{[n]}(z_1))^4 + \tfrac{c_2}{\gamma} + 10\right) \end{aligned}$$

oraz c_1, c_2, γ, m i n są pewnymi stałymi; s jest EAC obliczalną funkcją postaci $s(t) = \frac{1}{2}(\sin^2(2\pi t) + \sin(2\pi t))$. Funkcja f_M jest EAC obliczalnym rozszerzeniem f_δ z twierdzenia 4.1. Funkcje σ i d_2 są funkcjami z powyższych lematów i sprawiają one, iż możemy kontrolować błąd wyniku. Powyższa konstrukcja gwarantuje, że z_2' jest dostatecznie mała na przedziale $\left[0, \frac{1}{2}\right]$, dlatego też mamy:

$$\left\| z_2\left(\frac{1}{2}\right) - \bar{x}_0 \right\|_\infty < \frac{\gamma + \delta}{2} + \varepsilon < \frac{1}{2}$$

oraz

$$\left\| z_1\left(\frac{1}{2}\right) - f_\delta(\bar{x}_0) \right\|_\infty < 2\gamma + \frac{\delta}{2} \leqslant \varepsilon.$$

Na przedziale $\left[\frac{1}{2}, 1\right]$ role zmiennych z_1 i z_2 się zamieniają. Proces ten może być powtarzany dla z_1 i z_2 na kolejnych przedziałach, ostatecznie dla $j \in \mathbb{N}$, $t \in \left[j, j + \frac{1}{2}\right]$ otrzymamy:

$$\|z_2(t) - f_\delta^{[j]}(\bar{x}_0)\|_\infty < \varepsilon.$$

Wszystkie użyte do tej pory funkcje są EAC obliczalne oraz funkcje z_1 i z_2 są EAC obliczalne jako analityczne rozwiązanie układu równań różniczkowych z warunkiem brzegowym $z_1(0) = z_2(0) = \tilde{\bar{x}}_0$, gdzie $\|\tilde{\bar{x}}_0 - \bar{x}_0\|_\infty \leqslant \varepsilon$ dla $0 < \varepsilon < \frac{1}{4}$.

Przypuśćmy teraz, że $z(t)$ jest funkcją o następującej własności:

$$\|z(t) - f_\delta^{[j]}(\bar{x}_0))\|_\infty < \varepsilon \quad \text{dla } j \in \mathbb{N}, t \in \left[j, j + \frac{1}{2}\right].$$

Zatem dzięki założeniu, że maszyna Turinga po osiągnięciu stanu finalnego przechodzi do tej samej konfiguracji, wystarczy wziąć $\lim_{t\to 0} z\left(\frac{1}{t}\right)$ aby otrzymać obwód EACa, który w sposób odporny na błędy generuje wynik obliczeń maszyny Turinga.

□

W ten sposób otrzymaliśmy pozytywną odpowiedź na pytanie „Czy EAC potrafi symulować komputer cyfrowy?". Wciąż pozostaje kwestią otwartą pytanie o istnienie symulacji cyfrowej modelu EAC oraz pytanie o to, czy zbiór funkcji EAC obliczalnych jest zamknięty ze względu na iterację.

4.2 Porównanie z funkcjami rekurencyjnymi

Rozważymy teraz podobny problem jak w poprzednim paragrafie, z tym że tym razem naszym modelem obliczeń dyskretnych będą funkcje rekurencyjne.

Zaczniemy od przytoczenia kilku wyników Richardsona z pracy [64] cytowanych przez N. C. A da Costę i F. A. Dorię w pracy [22]. Zaczniemy od definicji pewnej algebry funkcji zdefiniowanej w pracy [64]

Definicja 4.1 *Niech $\mathcal{T}$ będzie algebrą funkcji daną następująco*:

$$\mathcal{T} = ([\mathbb{R}, x_1, x_2, \ldots, \sin(x), \exp(x); +, \times, \mathrm{SK}], +, \times, \mathrm{SK}).^3$$

Zaprezentujemy teraz kilka potrzebnych nam wyników z pracy [64].

Lemat 4.6 [64] *$\mathcal{T}$ jest zamknięta ze względu na operację pochodnej cząstkowe.*

Dowód. Do dowodu użyjemy indukcji. Pierwszy krok indukcyjny dla funkcji stałej oraz pojedynczej zmiennej.

1. Jeżeli $f(x_1, \ldots, x_n) = c$,
   wówczas $\partial_i f(x_1, \ldots, x_n) = 0$ zatem jest w $\mathcal{T}$.

[3] $x_1, x_2, \ldots$ są to zmienne rzeczywiste.

2. Jeżeli $f(x_1, \ldots, x_n) = x_i$,
   wówczas $\partial_i f(x_1, \ldots, x_n) = 1$ zatem jest w $\mathcal{T}$.

Zaprezentujemy teraz drugi krok indukcyjny. Niech $f_a(x_1, \ldots, x_n)$ oraz $f_b(x_1, \ldots, x_n)$ będą funkcjami otrzymanymi w k lub mniej krokach jak powyżej:

$$f_a(x_1, \ldots, x_n) \in \mathcal{T} \quad \text{i} \quad \partial_i f_a(x_1, \ldots, x_n) \in \mathcal{T}$$

oraz

$$f_b(x_1, \ldots, x_n) \in \mathcal{T} \quad \text{i} \quad \partial_i f_b(x_1, \ldots, x_n) \in \mathcal{T}.$$

Wówczas

– jeżeli $f = f_a + f_b$, to $\partial_i f = \partial_i f_a + \partial_i f_b$;

– jeżeli $f = f_a f_b$, to $\partial_i f = f_b \partial_i f_a + f_a \partial_i f_b$; [4]

– jeżeli $f = f_a \circ f_b$, to $\partial_i f = \partial_i (f_a \circ f_b) + \partial_i f_b$;

– jeżeli $f = e^{f_a}$, to $\partial_i f = e^{f_a} \partial_i f_a$;

– jeżeli $f = \sin(f_a)$, to $\partial_i f = \sin(\frac{1}{2}\pi + f_a) \partial_i f_a$.

To kończy dowód.

□

Lemat 4.7 [22] *Jeżeli $f \in \mathcal{T}$, wówczas istnieje, taka $g \in \mathcal{T}$ że*:

$$(\forall \bar{x} \in \mathbb{R}^n) g(\bar{x}) > 1 \text{ oraz}$$

$$(\forall \bar{x} \in \mathbb{R}^n, \bar{\Delta} \in \mathbb{R}^n)(|\Delta_i| \leqslant 1) \rightarrow (g(\bar{x}) > |f(\bar{x} + \bar{\Delta})|).$$

Dowód. Do dowodu użyjemy indukcji. Pierwszy krok indukcyjny dla funkcji stałej oraz pojedynczej zmiennej.

1. Jeżeli $f(x_1, \ldots, x_n) = c$,
   wówczas mamy $g(x_1, \ldots, x_n) = |c| + 2$.

[4] Dla skrócenia zapisu będziemy opuszczać znak mnożenia $\times$.

2. Jeżeli $f(x_1, \ldots, x_n) = x_i$,
   wówczas mamy $g(x_1, \ldots, x_n)) = x_i^2 + 2$.
   Mamy $\forall((x_1, \ldots, x_n) \in \mathbb{R}^n) g(x_1, \ldots, x_n) > 1$ oraz $x_i^2 + 2 > |x_i + \Delta_i|$, $|\Delta_i| \leqslant 1$. Mamy $x^2 > x$ dla $x > 1$ oraz stałej 2 biorąc pod uwagę sytuację dla $0 < x \leqslant 1$.

Zaprezentujemy teraz drugi krok indukcji. Niech $f_a(x_1, \ldots, x_n)$ oraz $f_b(x_1, \ldots, x_n)$ będą funkcjami otrzymanymi w k lub mniej krokach jak powyżej oraz niech g i h będą funkcjami skonstruowanymi następująco:

$$g(x_1, \ldots, x_n) > |f_a(x_1 + \Delta_1, \ldots, x_n + \Delta_n)|;$$

$$g(x_1, \ldots, x_n) > 1$$

oraz

$$h(x_1, \ldots, x_n) > |f_b(x_1 + \Delta_1, \ldots, x_n + \Delta_n)|;$$

$$h(x_1, \ldots, x_n) > 1.$$

Wówczas,

- jeżeli $f = f_a + f_b$, bierzemy $k = g + h$;
- jeżeli $f = f_a f_b$, bierzemy $k = gh$;
- jeżeli $f = f_a \circ f_b$, bierzemy $k = g \circ h$;
- jeżeli $f = e^{f_a}$, bierzemy $k = e^g$;
- jeżeli $f = \sin(f_a)$, bierzemy $k = 2$.

To kończy dowód.

□

Jako wniosek z lematu 4.6 i 4.7 otrzymujemy następujący lemat.

Lemat 4.8 [64] *Istnieje konstrukcyjna procedura, dzięki której dla danego $p \in P$ możemy otrzymać funkcje $k_i \in \mathcal{T}$ spełniające następujący warunek: jeżeli $|\Delta_i| \leqslant 1, i = 1, \ldots, n$, wówczas*

$$k_i(\bar{m}, \bar{x}) > |\partial_i(p^2(\bar{m}, \bar{x} + \bar{\Delta}))|, \tag{4.1}$$

gdzie $\bar{m} = (m_1, m_2, \ldots, m_k) \in \mathbb{N}^k$ oraz $\bar{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n, \bar{\Delta} \in \mathbb{R}^n$, $k, n \in \mathbb{N}$.

Dowód. Niech $p(\bar{x}) \in P$, wówczas $p(\bar{x})$ jest w $\mathcal{T}$, więc $p^2(\bar{x})$ jest w $\mathcal{T}$. Z lematu 4.6 mamy, że $\partial_i(p^2(\bar{x}))$ jest w $\mathcal{T}$. Zatem z lematu 4.7 otrzymamy, że dla każdej $\partial_i(p^2(\bar{x}))$ istnieje funkcja $k_i(\bar{x}) \in \mathcal{T}$ spełniająca warunek (4.1).

□

Teraz przedstawimy kilka własnych obserwacji dotyczących własności algebry $\mathcal{T}$ w kontekście modelu EAC. Patrząc na definicję algebry $\mathcal{T}$ możemy sformułować następujący lemat łączący EACa i $\mathcal{T}$.

Lemat 4.9 EAC *może wygenerować dowolną funkcję należącą do* $\mathcal{T}$.

Dowód. Funkcje elementarne takie jak $\exp(x)$ oraz $\sin(x)$ są funkcjami algebraicznie różniczkowalnymi stąd są EAC obliczalne. Z definicji EAC jest zamknięty ze względu na dodawanie, mnożenie i składanie. Zatem EAC może wygenerować dowolną funkcję z $\mathcal{T}$.

□

Bezpośrednio z lematu 4.8 oraz lematu 4.9 otrzymamy następującą uwagę:

Uwaga 4.1 *Wszystkie funkcje* k_i *z lematu 4.8 są* EAC *obliczalne.*

Zacytujemy teraz dalsze użyteczne dla nas definicje i twierdzenia z pracy [64].

Definicja 4.2 *Dla danego* $p \in P$ *oraz* k_i *z lematu 4.8 definiujemy:*

$$f(\bar{m}, \bar{x}) = (n+1)^4[p^2(\bar{m}, \bar{x}) + \sum_{i=1}^{n}(\sin^2 \pi x_i)k_i^4(\bar{m}, \bar{x})], \tag{4.2}$$

gdzie $\bar{m} = (m_1, m_2, \ldots, m_k) \in \mathbb{N}^k$ *oraz* $\bar{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, $k, n \in \mathbb{N}$.

Możemy łatwo zauważyć, że funkcja $f(\bar{m}, \bar{x})$, jako złożenie EAC obliczalnych funkcji jest również EAC obliczalna. Następujący wynik został udowodniony w pracy [64].

Twierdzenie 4.4 [64] *Dla* p *oraz* f *zdefiniowanej wzorem* (4.2)*, następujące warunki są równoważne: dla każdego* $\bar{m} \in \mathbb{N}^k$:

1. *Istnieją liczby naturalne* $x_1, \ldots, x_n$, *takie że* $p(\bar{m}, x_1, \ldots, x_n) = 0$.

2. *Istnieją nieujemne liczby rzeczywiste* $x_1, \ldots, x_n$*, takie że* $f(\bar{m}, x_1, \ldots, x_n) = 0$.

3. *Istnieją nieujemne liczby rzeczywiste* $x_1, \ldots, x_n$*, takie że* $f(\bar{m}, x_1, \ldots, x_n) \leqslant 1$.

Wprowadzimy i dalej będziemy używać dodatkową funkcję:

$$\bar{f}(\bar{m}, x_1, \ldots, x_n) = f(\bar{m}, x_1^2, \ldots, x_n^2).$$

Oczywiście $\bar{f}$ jest również EAC obliczalna.

W celu otrzymania funkcji jedno-argumentowej w [64] i [22] użyto następującej konstrukcji, którą i my również zaadaptujemy.
Niech $r(y) = y\sin(y)$ oraz $s(y) = y\sin(y^3)$. Wówczas dla danej $\bar{f}(\bar{m}, x_1, \ldots, x_n)$, dokonujemy następującego podstawienia:

$$\begin{aligned}
&x_1 = r(y),\\
&x_2 = (r \circ s)(y),\\
&x_3 = (r \circ s \circ s)(y),\\
&\vdots\\
&x_{n-1} = (r \circ \underbrace{s \circ \ldots \circ s}_{n-2})(y),\\
&x_n = (\underbrace{s \circ s \circ \ldots \circ s}_{n})(y).
\end{aligned} \tag{4.3}$$

Ostatecznie otrzymujemy:

$$g(\bar{m}, y) = \bar{f}(\bar{m}, r(y), r(s(y)), \ldots, r(s(s(\ldots s(y))\ldots)), s(s(s(\ldots s(y))\ldots))),$$

gdzie $g : \mathbb{R}^{k+1} \to \mathbb{R}$ oraz $\bar{m} = (m_1, m_2, \ldots, m_k) \in \mathbb{N}^k$.

Powyższe funkcje $s(y)$ oraz $r(y)$ są EAC obliczalne, zatem powyższa konstrukcja może być wygenerowana przez pewien EAC.

Teraz jako konsekwencję twierdzenia 4.4 można udowodnić następującą własność.

Własność 4.1 [64] *Dla danej* $\bar{m} \in \mathbb{N}^k$ *następujące warunki są równoważne*:

1. *Istnieją liczby naturalne* $x_1, \ldots, x_n$*, takie że* $p(\bar{m}, x_1, \ldots, x_n) = 0$.

2. *Istnieje liczba rzeczywista* y*, taka że* $g(\bar{m}, y) \leqslant 1$.

Ponadto, p oraz g są EAC *obliczalne.*

Dowód. Na podstawie twierdzenia zaprezentowanego w [64] wiemy, że dla dowolnych liczb rzeczywistych $y_1, \ldots, y_n$ oraz dowolnej $\delta > 0$, istnieje liczba rzeczywista y, taka że:

$$\begin{aligned}
&|r(y) - y_1| < \delta \\
&|r(s(y)) - y_2| < \delta \\
&\vdots \\
&|r(\underbrace{s(s(\cdots s(y))\cdots)}_{n-2}) - y_{n-1}| < \delta \\
&\underbrace{s(s(s(\cdots s(y))\cdots))}_{n} = y_n.
\end{aligned}$$

W pierwszej kolejności udowodnimy drugi warunek naszej własności. Zatem na podstawie definicji

$$g(\bar{m}, y) = \bar{f}(\bar{m}, r(y), r(s(y))), \ldots, r(s(s(\ldots s(y))\ldots), s(s(s(\ldots s(y))\ldots)),$$

ten warunek zachodzi wtedy i tylko wtedy, gdy istnieją liczby rzeczywiste $y_1, \ldots, y_n$, dla których $\bar{f}(\bar{m}, y_1, \ldots, y_n) \leqslant 1$.
Dalej $\bar{f}(\bar{m}, y_1, \ldots, y_n)$ była zdefiniowana jako $f(\bar{m}, y_1^2, \ldots, y_n^2)$, zatem istnieje równoważny warunek: istnieją nieujemne liczby rzeczywiste $x_1, \ldots, x_n$, dla których $f(\bar{m}, x_1, \ldots, x_n) \leqslant 1$.
Następnie, na mocy twierdzenia 4.4, fakt, że istnieją nieujemne liczby rzeczywiste $x_1, \ldots, x_n$, dla których $f(\bar{m}, x_1, \ldots, x_n) \leqslant 1$, jest równoważny stwierdzeniu, że istnieją liczby naturalne $x_1, \ldots, x_n$, dla których $p(\bar{m}, x_1, \ldots, x_n) = 0$. Zatem ostatecznie otrzymujemy:

$$(\exists y \in \mathbb{R}) g(\bar{m}, y) \leqslant 1 \leftrightarrow (\exists (x_1, \ldots, x_n) \in \mathbb{N}^n) p(\bar{m}, x_1, \ldots, x_n) = 0.$$

Łatwo zaobserwować, że p oraz g są EAC obliczalne. Rzeczywiście, p jest wielomianem, a na podstawie konstrukcji g widzimy, że g jest również EAC obliczalna.

□

Przejdziemy teraz do zaprezentowania własnych, głównych dla tego paragrafu wyników.

Pokażemy, że EAC potrafi generować wartości dowolnej funkcji częściowo rekurencyjnej zdefiniowanej na $\mathbb{N}$ oraz zbiory rekurencyjnie przeliczalne liczb naturalnych.

Używając zaprezentowanej powyżej teorii możemy sformułować następujące własne twierdzenia:

Twierdzenie 4.5 *Każdy rekurencyjnie przeliczalny zbiór S może być wygenerowany przez pewien* EAC.

Dowód. Niech D będzie zbiorem rekurencyjnie przeliczalnym. Zatem D jest również zbiorem Diofantycznym. Zatem istnieje wielomian p, dla którego ma miejsce następująca własność:

$$x \in D \equiv (\exists z \in \mathbb{N}) p(x, z) = 0.$$

Z własności 4.1 istnieje EAC obliczalna funkcja $g(x, y)$, taka że

$$x \in S \equiv (\exists y \in \mathbb{R}) g(x, y) \leqslant 1,$$

gdzie zbiór $S \subseteq \mathbb{R}$ jest następująco powiązany ze zbiorem D:

$$(\forall n \in \mathbb{N})(n \in S \Leftrightarrow n \in D).$$

Zaprezentujemy teraz szczegóły konstrukcji S w modelu EAC. Niech

$$h(x, y) = 1 - g(x, y).$$

Wówczas z definicji 2.25, EAC może wygenerować zbiór

$$S' = \{(x, y) \in \mathbb{R} : h(x, y) \geqslant 0\}$$

oraz na tym samym poziomie zbiór $S = \{x : (\exists y \in \mathbb{R})(x, y) \in S'\}$.
Teraz skonstruujemy zbiór $\mathbb{N}$ liczb naturalnych jako przecięcie zbiorów N_1 oraz N_2, gdzie

$$N_1 = \{x \in \mathbb{R}_+ : \sin 2x\pi \geqslant 0\}, \quad N_2 = \{x \in \mathbb{R}_+ : -\sin 2x\pi \geqslant 0\}$$

oraz $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geqslant 0\}$. Funkcja $\sin 2x\pi$ jest EAC obliczalna. Zatem z definicji 2.25 zbiór $\mathbb{N}$ może być otrzymany w modelu EAC. Ostatecznie, zbiór D jest po prostu przecięciem $S \cap \mathbb{N}$.

□

Ponieważ każdy rekurencyjny zbiór jest również rekurencyjnie przeliczalny, mamy następujący wniosek.

Wniosek 4.1 *Każdy rekurencyjny zbiór S może być wygenerowany w modelu* EAC.

Twierdzenie 4.6 *Niech f będzie funkcją częściowo rekurencyjną zdefiniowaną na $\mathbb{N}$. Wówczas istnieje rodzina $(M_n)_{n\in\mathbb{N}}$ maszyn EAC, taka że dla każdego $n \in \mathbb{N}$, M_n generuje zbiór jednoelementowy $\{f(n)\}$, gdy $f(n)\downarrow$ lub $\emptyset$, gdy $f(n)\uparrow$.*

Dowód. Dla każdej funkcji częściowo rekurencyjnej, zbiór par $\{(n, f(n)) : n \in \mathbb{N}\}$ jest rekurencyjnie przeliczalny. Z dowodu twierdzenia 4.5 wynika, że EAC może wygenerować zbiór G_f, par liczb naturalnych, taki że dla każdej pary liczb naturalnych mamy: jeśli $(a, b) \in G_f$, wówczas $b = f(a)$ oraz $\{(n, f(n)) : n \in \mathbb{N}\} \subseteq G_f$.

Niech n będzie daną liczbą naturalną. Konstrukcja maszyny M_n, która generuje zbiór jednoelementowy $\{f(n)\}$ przedstawia się następująco. W pierwszej kolejności generujemy funkcje: $(x - n)$ oraz $(n - x)$, a następnie zbiór: $\Omega_n^1 \cap \Omega_n^2$, gdzie

$$\Omega_n^1 = \{(x, y) \in \mathbb{R}^2 : x - n \geqslant 0\}$$

oraz

$$\Omega_n^2 = \{(x, y) \in \mathbb{R}^2 : n - x \geqslant 0\}.$$

Wówczas na tym samym poziomie otrzymamy zbiór

$$\Omega_n = \{(x, y) \in \mathbb{R}^2 : x - n = 0\}$$

jako przecięcie $\Omega_n^1 \cap \Omega_n^2$. Ostatecznie jako przecięcie zbiorów G_f i Ω_n otrzymamy zbiór jednoelementowy $\{(n, f(n))\}$, gdy $f(n)\downarrow$ lub zbiór pusty, gdy $f(n)\uparrow$, następnie na tym samym poziomie, jako rzut $G_f \cap \Omega_n$, otrzymamy zbiór $\{f(n)\}$, gdy $f(n)\downarrow$ lub zbiór pusty, gdy $f(n)\uparrow$.

□

Powyższe twierdzenie pokazuje nam, że klasyczny problem stopu dla funkcji częściowo rekurencyjnych (tj. „Czy funkcja częściowo rekurencyjna jest zdefiniowana dla danego n, czy nie jest?") może być zredukowany do problemu rozstrzygnięcia w modelu EAC „Czy zbiór jest pusty?". Jednakże

powinniśmy pamiętać, że w twierdzeniu 4.6 nie rozważamy jednej maszyny EAC, a całą rodzinę maszyn.

Ponadto wiemy teraz, że dla dowolnej funkcji częściowo rekurencyjnej f zdefiniowanej na $\mathbb{N}$ oraz, dla każdego $n \in \mathbb{N}$ możemy otrzymać w modelu EAC zbiór jednoelementowy $\Lambda = \{f(n)\}$, gdy $f(n) \downarrow$ lub $\Lambda = \emptyset$, gdy $f(n) \uparrow$. Niech i będzie funkcją identyczności zdefiniowaną na $\mathbb{R}$. Zatem możemy wyznaczyć w modelu EAC funkcję $i_{|\Lambda}$ i w ten sposób otrzymać wartość $f(n)$, gdy $f(n) \downarrow$.

Podsumowując tą sekcję zaprezentowane w niej wyniki implikują fakt, iż EAC generuje dowolne rekurencyjnie przeliczalne zbiory liczb naturalnych, co za tym idzie, możemy również otrzymać wartość dowolnej funkcji częściowo rekurencyjnej w zadanym punkcie n. Wyniki powyższe nie rozstrzygają czy dla każdej funkcji częściowo rekurencyjnej f definiowanej na $\mathbb{N}$ istnieje EAC obliczalna funkcja $\tilde{f}$ definiowana na $\mathbb{R}$ o następującej własności $(\forall n \in \mathbb{N})\tilde{f}(n) \approx f(n)$, ale stanowią teoretyczną bazę do dalszych prac w tym kierunku. Ponadto mogliśmy zaobserwować ciekawy fakt, że klasyczny problem stopu jest równoważny odpowiedzi w modelu EAC na pytanie „Czy zbiór jest pusty?".

4.2.1 Przykłady generowania zbiorów rekurencyjnych przez EAC

Rozważmy zbiory rekurencyjne, których elementy są rozwiązaniami dwóch równań diofantycznych: równania Pella: $m_1^2 - dx_1^2 = 1$, gdzie d jest liczbą całkowitą dodatnią oraz równania Pitagorasa: $m_1^2 + x_1^2 = x_2^2$.

Przykład 4.1 Równanie Pella dla $d = 3$: $m_1^2 - 3x_1^2 = 1$.
Rozważmy wielomian: $p(m_1, x_1) = m_1^2 - 3x_1^2 - 1$ oraz zbiór

$$D_{Pell} = \{n \in \mathbb{N} : (\exists x_1 \in \mathbb{N})p(n, x_1) = 0\}.$$

Spróbujemy teraz wygenerować zbiór D_{Pell} w modelu EAC zgodnie z konstrukcją z dowodu twierdzenia 4.5. Zatem będzie nam potrzebna funkcja $g(m_1, y)$ z własności 4.1. Mamy

$$g(m_1, y) = \hat{f}(m_1, x_1) = f(m_1, x_1^2),$$

gdzie $x_1 = r(y) = y \sin y$.
Zgodnie z definicją 4.2:

$$f(m_1, x_1) = 2^4[(m_1^2 - 3x_1^2 - 1)^2 + (\sin^2 \pi x_1)k_1^4(m_1, x_1)].$$

Lemat 4.8 mówi nam, że funkcja k_1 ma następującą własność:

$$k_1(m_1, x_1) > |\partial_{x_1}(p^2(m_1, x_1 + \Delta))|, |\Delta| \leqslant 1$$

i zgodnie z konstrukcją zaprezentowaną w dowodzie lematu 4.7 przyjmuje postać:

$$k_1(m_1, x_1) = 14(m_1 + 2)^2(x_1^2 + 2) + 38(x_1^2 + 2)^3 + 14(x_1^2 + 2).$$

Po podstawieniu otrzymamy:

$$g(m_1, y) = 2^4[(m_1^2 - 3y^2 \sin^2 y - 1)^2 + \sin^2(\pi y \sin y)[14(m_1^2 + 2)(y^4 \sin^4 y + \\ +2) + 38(y^4 \sin^4 y + 2)^3 + 14(y^4 \sin^4 y + 2)]^4]$$

Wyznaczamy teraz zbiór

$$S = \{x : (\exists y \in \mathbb{R})(1 - g(m_1, y) \geqslant 0)\},$$

a następnie zbiór $\mathbb{N}$ jak w dowodzie twierdzenia 4.5. Ostatecznie wyznaczamy zbiór D_{Pell} jako $S \cap \mathbb{N}$.

Przykład 4.2 Równanie Pitagorasa: $m_1^2 + x_1^2 = x_2^2$.
Rozważmy wielomian: $p(m_1, x_1, x_2) = m_1^2 + x_1^2 - x_2^2$ oraz zbiór

$$D_{Pitgoras} = \{n \in \mathbb{N} : (\exists x_1 \in \mathbb{N}) p(n, x_1, x_2) = 0\}.$$

Spróbujemy teraz wygenerować zbiór $D_{Pitagoras}$ w modelu EAC zgodnie z konstrukcją z dowodu twierdzenia 4.5. Zatem będzie nam potrzebna funkcja $g(m_1, y)$ z własności 4.1. Mamy

$$g(m_1, y) = \hat{f}(m_1, x_1, x_2) = f(m_1, x_1^2, x_2^2),$$

gdzie $x_1 = r(y) = y \sin y, x_2 = r(s(y)) = y \sin y^3 \sin(y \sin y^3)$.
Zgodnie z definicją 4.2:

$$f(m_1, x_1, x_2) = 3^4[(m_1^2 + x_1^2 - x_2^2)^2 + (\sin^2 \pi x_1) k_1^4(m_1, x_1, x_2) + \\ +(\sin^2 \pi x_2) k_2^4(m_1, x_1, x_2)].$$

Lemat 4.8 mówi nam, że funkcje k_1, k_2 mają następującą własność:

$$k_1(m_1, x_1, x_2) > |\partial_{x_1}(p^2(m_1, x_1 + \Delta_1, x_2 + \Delta_2))|, |\Delta_1| \leqslant 1, |\Delta_2| \leqslant 1,$$

$$k_2(m_1, x_1, x_2) > |\partial_{x_2}(p^2(m_1, x_1 + \Delta_1, x_2 + \Delta_2))|, |\Delta_1| \leqslant 1, |\Delta_2| \leqslant 1$$

i zgodnie z konstrukcją zaprezentowaną w dowodzie lematu 4.7 przyjmują postać:

$$k_1(m_1, x_1, x_2) = 6(m_1^2+2)^2(x_1^4+2) + 6(x_1^4+2)^3 + 6(x_1^4+2)(x_2^4+2)^2,$$

$$k_2(m_1, x_1, x_2) = 6(m_1^2+2)^2(x_1^4+2) + 6(x_1^4+2)^2(x_2^4+2) + 6(x_2^4+2)^3.$$

Po podstawieniu otrzymamy:

$$\begin{aligned} &g(m_1, y) = 81[(m_1^2 + y^4 \sin^4 y - y^4 \sin^4 y^3 \sin(y \sin y^3))^2 + \\ &+ \sin^2(\pi y^2 \sin^2 y)(6(m_1^2+2)^2((y \sin y)^4+2) + 6((y \sin y)^4+2)^3 + \\ &+6((y \sin y)^4+2)((y \sin y^3 \sin(y \sin y^3))^4+2)^2)^4 + \\ &+ \sin^2(\pi y^2 \sin^2 y^3 \sin^2(y \sin y^3))(6(m_1^2+2)^2((y \sin y)^4+2) + 6((y \sin y)^4 + \\ &+2)^2((y \sin y^3 \sin(y \sin y^3))^4+2) + 6((y \sin y^3 \sin(y \sin y^3))^4+2)^3)^4]. \end{aligned}$$

Wyznaczamy teraz zbiór

$$S = \{x : (\exists y \in \mathbb{R})(1 - g(m_1, y) \geqslant 0)\},$$

a następnie zbiór $\mathbb{N}$ jak w dowodzie twierdzenia 4.5. Ostatecznie wyznaczamy zbiór $D_{Pitagoras}$ jako $S \cap \mathbb{N}$.

Rozdział 5

Podsumowanie

Podsumowując, dzięki zaprezentowanym różnym sposobom przeniesienia obliczeń modeli dyskretnych, na pole obliczeń w dziedzinie rzeczywistej, zrealizowane zostały wszystkie wymienione we wstępie niniejszej pracy cele. Jako pierwszą zaprezentowano symulację jednego z podstawowych, klasycznych modeli obliczeń, maszyny Turinga. Jest to model, który odgrywa dużą rolę w badaniach dotyczących złożoności obliczeń. Zaprezentowana symulacja, dotyczy generowania wyników dowolnej maszyny Turinga, poprzez funkcję klasy rzeczywistych funkcji rekurencyjnych.

Kolejny zaprezentowany wynik dotyczy, symulacji maszyny Turinga, przez funkcję analityczną zmiennej rzeczywistej. Większość procesów fizycznych jest analityczna (przynajmniej w fizyce klasycznej), dlatego też funkcje analityczne są szczególnie pożądane, gdy rozważamy obliczenia komputerów analogowych. Jeden z teoretycznych modeli obliczeń model EAC, wprost zakłada konieczność analityczności rozważanych w nim funkcji. Stąd prezentowana symulacja maszyny Turinga, której wyniki obliczeń są generowane za pomocą pewnej funkcji analitycznej. Jest to funkcja z rodziny funkcji Collatza. Za jej pomocą iterowane są kolejne kroki obliczeń maszyny Turinga. Symulacja ta daje możliwość sprowadzenia problemu stopu dla maszyn Tyringa do szeroko dyskutowanego w literaturze problemu, czy dla danej funkcji Collatza $f(x)$ zawsze któraś z kolei iteracja $f^t(x)$ da wartość 1, co doprowadzi do pojawienia się cyklu kolejnych iterowanych wartości $\{1, 4, 2\}$.

Dalej w pracy skupiono się na porównaniu możliwości modelu EAC, w stosunku do klasycznych komputerów cyfrowych. W pierwszej kolejności podany został sposób uzyskania wyniku dowolnej maszyny Turinga, z dowolną precyzją w modelu EAC, a następnie sposób uzyskania dowolnego zbioru rekuren-

cyjnie przeliczalnego oraz wyniku dowolnej funkcji rekurencyjnej w modelu EAC. Ostatni zaprezentowany wynik dowodzi słuszności postawionej w pracy tezy i daje nam możliwość sprowadzenia klasycznego problemu stopu, do problemu zbadania w modelu EAC pustości zbioru.

Zaprezentowane wyniki stanowią również dowód na to, iż przeniesienie obliczeń dyskretnych na dziedzinę rzeczywistą nie zmniejsza klasy rozwiązywanych problemów, gdyż wszystko co możemy otrzymać w klasycznych modelach obliczeń możemy również otrzymać w modelach analogowych, takich jak EAC, czy rzeczywiste funkcje rekurencyjne.

Uzyskane wyniki mogą służyć jako baza do dalszych rozważań w tym temacie oraz próby rozwiązania nasuwających się tutaj kilku otwartych problemów z zapytaniem o odwrotną relację, mianowicie:

1. Czy istnieje rekurencyjna funkcja rzeczywista, której wartości w punktach wymiernych nie mogą być wyznaczone przez maszynę Turinga z pewną dokładnością?

2. Czy istnieje funkcja zdefiniowana na zbiorze liczb rzeczywistych $\mathbb{R}$, obliczalna w modelu EAC, której wartości w punktach wymiernych nie mogą być wyznaczone przez maszynę Turinga z pewną dokładnością?

3. Czy EAC potrafi rozwiązać klasyczny problem stopu? Czyli, czy problem pustość zbioru jest problemem rozstrzygalnym w modelu EAC? Oznaczałoby to, że rozwiązuje on właściwie szerszą klasą problemów, niż komputery cyfrowe.

Przede wszystkim zaś zaprezentowane wyniki stanowią wkład do zintegrowania różnych typów obliczeń: konwencjonalnych (dyskretnych) i niekonwencjonalnych (analogowych) co było celem niniejszej rozprawy.

Spis tabel

Spis rysunków

Oznaczenia

x, y, z	liczby rzeczywiste $\mathbb{R}$
t	liczba rzeczywista – czas/liczba naturalna – krok
k, l, m, n	liczby naturalne $\mathbb{N}$
$\mathbf{k}, \mathbf{l}, \mathbf{m}, \mathbf{n}$	napisy nad pewnym alfabetem Σ kodujące liczby k, l, m, n
$A, C, D, S, X, Y,$	
Ω, Λ	zbiory
P	zbiór wielomianów n-zmiennych
I	przedział
c_A	funkcja charakterystyczna zbioru
c_R	funkcja charakterystyczna relacji
$\mathrm{Dom}(f)$	zbiór wszystkich argumentów, dla których funkcja f ma określoną wartość
$\int_{t_0}^{t} f(x)dg(x)$	całka Riemanna-Stieltjesa
$\bar{x}$	wektor $(x_1, x_2, \ldots, x_n)$
$\mathfrak{F}$	klasa funkcji
$\mathcal{F}$	zbiór funkcji należących do $\mathfrak{F}$
$\mathcal{O}$	zbiór operatorów $\mathfrak{F}^k \to \mathfrak{F}$
$[\mathcal{F}; \mathcal{O}]$	indukcyjne domkniecie/algebra funkcji
$\mathcal{A}$	oznaczenie indukcyjnego domknięcia/algebry funkcji
SK^l	operator l-argumentowej operacji składania, str. 17
REK	operator rekursji prostej, str. 17
μ	operator minimalizacji, str. 18
$\mathfrak{z}$	jedno-argumentowa funkcja zerowania zdefiniowana na $\mathbb{N}$, $\mathfrak{z}(n) = 0$
$\mathfrak{s}$	jedno-argumentowa funkcja następnika zdefiniowana na $\mathbb{N}$, $\mathfrak{s}(n) = n + 1$
$\mathfrak{i}_i^k$	k-argumentowa funkcja projekcji zdefiniowana na $\mathbb{N}$, $\mathfrak{i}_i^k(n_1, \ldots, n_k) = n_i$
$\mathfrak{i}$	$\mathfrak{i} = \{\mathfrak{i}_i^k : k, i \in \mathbb{N}, 1 \leqslant i \leqslant k\}$
$\downarrow, \uparrow$	str. 17
$\mathcal{PRF}$	klasa funkcji rekurencyjnych $\mathcal{PRF} = [\mathfrak{z}, \mathfrak{s}, \mathfrak{i}; \mathrm{SK}^l, \mathrm{REC}, \mu]$
$\mathcal{RF}$	klasa funkcji całkowicie rekurencyjnych

$\mathcal{P}$	klasa funkcji pierwotnie rekurencyjnych $\mathcal{P} = [\mathfrak{z}, \mathfrak{s}, \mathfrak{i}; \mathrm{SK}^l, \mathrm{REC}]$
$\langle \cdot, \cdot \rangle$	funkcja Cantora kodowania par
π_1, π_2	funkcje dekodujące dla $\langle \cdot, \cdot \rangle$ odpowiednio pierwszą i drugą składową pary
$\Delta_j^0, \Sigma_j^0, \Pi_j^0$	zbiory j-tego poziomu hierarchii arytmetycznej
δ	funkcja przejścia MT
Δ	relacja przejścia NTM
Σ	alfabet
B	symbol pusty, $B \in \Sigma$
α	napis nad alfabetem Σ
Q	zbiór stanów maszyny Turinga
F	zbiór stanów finalnych maszyny Turinga
$M_f(\mathbf{n})$	maszyna Turinga wyznaczająca wartość funkcji $f(n)$
$M_{\mathfrak{s}}(\mathbf{n})$	maszyna Turinga wyznaczająca wartość następnika liczby n, $\mathbf{n}$ jest binarnym zapisem liczby n
$f^t(\bar{x})$	funkcja f iterowana t-razy
$f^{(n)}(x)$	n-ta pochodna funkcji $f(x)$
$p(x_1, \ldots, x_n)$	wielomian zmiennych $x_1, \ldots, x_n$
$-1^n, 0^n, 1^n$	n-argumentowe funkcje stałe zdefiniowane na liczbach rzeczywistych
SK^l	operator składania, str. 40
INV	operator inwersji, str. 40
D	operator pochodnej cząstkowe, str. 41
$\mid$	operator obcięcia, str. 41
$\sim$	operator przedłużenia str. 41
RR	operator, wyznacza rozwiązanie równania różniczkowego, str. 41
L_∂	operator granicy, str. 42
$\mathrm{G}_>, \mathrm{G}_\geqslant$	operator wyznaczający zbiór argumentów, dla których funkcja przyjmuje odpowiednio wartości nieujemne i dodatnie, str. 42
P	operator projekcji $\{\bar{x} : (\exists x \in \mathbb{R})(x, \bar{x}) \in A\}$
SK	operator składania dwu funkcji, str. 57
R	operator rekursji różniczkowej, str. 57
L, LS, LI	operatory granicy nieskończonej, granicy nieskończonej górnej i granicy nieskończonej dolnej

V	operator agregacji $\mathrm{V}(f,g)(\bar{x}) = (f(\bar{x}), g(\bar{x})$
$\mathcal{RELC}(\mathbb{R})$	klasa rzeczywistych funkcji rekurencyjnych $\mathcal{RELC}(\mathbb{R}) = [-1^n, 0^n, 1^n, \mathfrak{i}_i^n; \mathrm{SK}, \mathrm{R}, \mathrm{LS}, \mathrm{V}]$
$\mathcal{H}$	algebra funkcji klasy $\mathcal{RELC}(\mathbb{R})$
$D_{\mathcal{A}}$	zbiór deskrypcji algebry $\mathcal{A}$
$\mathcal{D}_{\mathcal{A}}$	zbiór dobrych deskrypcji algebry $\mathcal{A}$
$rk(f)$	stopień funkcji f dla algebry $\mathcal{A}$ ze względu na zbiór operatorów $\tilde{\mathcal{O}}$
H_n	zbiór n-tego poziomu η-hierarchii, str. 64
$F_{\mathbb{Q}}$	klasa funkcji obliczalnych $f : \mathbb{N}^k \to \mathbb{Q}$
$\Delta_n, \Sigma_n, \Pi_n$	zbiory n-tego poziomu hierarchii arytmetycznej liczb rzeczywistych
$\partial^{f(a)}$	operator przesuwający ciąg znaków $a = \ldots a_{-2}a_{-1}.a_0a_1\ldots$ względem kropki w prawo lub w lewo w zależności od wartości $f(a)$
$\mathrm{DOD}(a)$	dla $a = \ldots a_{-2}a_{-1}.a_0a_1\ldots$, $\mathrm{DOD}(a) = a_{-1}.a_0a_1$
$\equiv$	symbol równoważności
$\lfloor\rfloor$	$\lfloor x \rfloor = \max\{y \in \mathbb{Z} : y \leqslant x\}$
$\mathcal{U}_n$	$\mathcal{U}_n = [\mathbb{Q}, \Pi, \mathfrak{i}_i^n; +, -, \times, \sin]$
L, R, G, W	rejestry skończenie-stanowego automatu
$\mathcal{T}$	$\mathcal{T} = [\mathbb{R}, x_1, x_2, \ldots, \sin(x), \exp(x); +, \times, \mathrm{SK}]$
G_f	graf funkcji f

Bibliografia

[1] S. Aaronson. Limits on efficient computation in the physical world. *Ph. D. Thesis, University of California, Berkeley*, 2004.

[2] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer.

[3] L. Blum, M. Shub, and S. Smale. On the theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21:1–46, 1989.

[4] M.S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 136:67–100, 1995.

[5] S. Bringsjord and J. Taylor. An argument for p = np. *Technical report*. www.rpi.edu$\tilde{}$brings.

[6] J. Buescu, M. L. Campagnolo, and D. S. Graça. Robust simulation of Turing machines with Analytic Maps and Flows. *Proceedings of CiE'05, New Computational Paradigms, Lecture Notes in Computer Science 3526*, pages 169–179, 2005. In B. Cooper, B. Loewe, L. Torenvliet, editors.

[7] V. Bush. The differential analyser. *Journal Franklin Institute*, 212:447–488, 1931.

[8] M. L. Campagnolo. The Complexity of Real Recursive Functions. *Lecture Notes in Computer Science*, 2509:1–14, 2002.

[9] M. L. Campagnolo, J. F. Costa, and C. Moore. Iteration, inequalities, and differentiability in analog computers. *Journal of Complexity*, 16 (4):642–660, 2000.

[10] M. L. Campagnolo, J. F. Costa, and C. Moore. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 18 (4):977–1000, 2002.

[11] A. Church. The calculi of lambda-conversion. *Annals of Mathematical Studies*, 6, 1941. Princeton Univ. Press, Princeton N.J.

[12] P. Clote. *Computation Models and Function Algebras in: Handbook of Computability Theory*, volume 140 of *Studies in Logic and the Foundations of Mathematics*, chapter 17 - Computation Models and Function Algebras, pages 608–609. Elsevier, 1999.

[13] J. H. Conway. Unpredictable iterations. *Proc. 1972 Number Theory University of Colorado*, pages 49–52, 1972.

[14] J. F. Costa and D.S. Graça. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19 (5):644–664, 2003.

[15] J. F. Costa, B. Loff, and J. Mycka. The new promise of analog computation. *Lecture Notes in Computer Science*, 30:189–195, 2007.

[16] J. F. Costa, B. Loff, and J. Mycka. A foundation for real recursive function theory. *Annals of Pure and Applied Logic*, 160:255–288, 2009.

[17] J. F. Costa and J. Mycka. Real recursive functions and their hierarchy. *Journal of Complexity*, 20:835–857, 2004.

[18] J. F. Costa and J. Mycka. The computational power of continuous dynamic sys. *Lecture Notes in Computer Science*, pages 164–175, 2005.

[19] J. F. Costa and J. Mycka. What lies beyond the mountains? computational systems beyond the turing limit. *Bulletin of the European Association for Theoretical Computer Science*, 2005.

[20] J. F. Costa and J. Mycka. An analytic condition for $P \subset NP$. *Journal of Complexity*, 2006.

[21] J. F. Costa and J. Mycka. Undecidability Over Continuous-Time. *Logic Journal of the IGPL, Oxford University Press*, 2006.

[22] N. C. A. da Costa and F.A. Doria. Undecidability and Incompleteness in Classical Mechanics. *International J. Theoret. Physics*, 4497:1041–1073, 1991.

[23] M. Davis, Y. Matijasevich, and J. Robinson. Hilbert's Tenth Problem. diophantine equations: Positive aspect of a negative solution. *Proc. Symp. Pure Math.*, 28:323–378, 1976.

[24] R. Driver. *Introduction to Ordinary Differential Equations*. Harper and Row, 1978.

[25] R. Feynman. Simulating physics with computers. *Int. J. Theoret. Phys.*, 21(6-7):467–488, 1982.

[26] D. S. Graça. Some recent developments on shannon's general purpose analog computer. *Math. Log. Quart.*, 50(4-5):473–485, 2004.

[27] A. Grzegorczyk. Computable functionals. *Fund. Math*, 42:168–202, 1955.

[28] J. J. Herlinger. *Niezwykłe perypetie odkryć i wynalazków*. Nasza Księgarnia, Warszawa, 1985.

[29] D. Hilbert. Mathematische probleme. *Proc. roy. Soc. Math.*, pages 58–114, 1990.

[30] I. Kaczmarczyk. Rozwój automatyki na przestrzeni wieków. http://free.of.pl/z/zst/pomoce/publikacje/automatyka.pdf.

[31] W. Thomson (Lord Kelvin). On an instrument for calculating the integral of the product of two given functions. *Proc. Royal Society of London*, 24:266–268, 1876.

[32] P. Koiran. Puissance de calcul des réseaux de neurones artificiels. *Ph.D.Thesis, Ecole Normale Supériere de Lyon*, 1993.

[33] P. Koiran and C. Moore. Closed-form analytic maps in one and two dimentions can simulate universal Turing machine. *Theoretical Computer Science*, (210):217–223, 1999.

[34] D. Lacombe. Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables reelles iii. *C. R. Acad. Sci. Paris.*, 241:151–153, 1955.

[35] J. C. Lagarias. The $3x+1$ problem and its generalizations. *Amer. Math. Montly*, 92:3–23, 1985.

[36] R. Lehman. On primitive recursive real numbers. *Fundamenta Mathematica*, 49:105–118, 1960/61.

[37] L. Lipshitz and L. A. Rubel. A differentialy algebraic replacement theorem, and analog computation. *Proceedings of the A.M.S.*, 99(2):367–372, 1987.

[38] A. Markow. Theory of algorithms. *Trudy Matiemat. Inst. V. A. Stiekłowa*, 42, 1954. przekład angielski: Israel Program for Scientific Translation, Jerusalem, 1961.

[39] Y. Matijasevich. Enumerable sets are diophantine. *Dokl. Acad. Nauk*, 191:279–282, 1970.

[40] S. Mazur. Computable analysis. *Rozprawy Matematyczne*, 33, 1963. Warszawa.

[41] J. Mills, M. G. Beavers, and C. Daffinger. Lukasiewicz logic arrays. *Proceedings of 2th Internationa Symposium on Multiple-Valued Logic, Charlotte*, pages 469–480, 1990.

[42] J. Mills and C. Daffinger. Cmos vlsi lukasiewicz logic arrays. *Proceedings 20th International Symposium on Multiple-valued Logic*, pages 4–10, 1990.

[43] J. W. Mills. The Nature of the Extended Analog Computer. *Physica D*, 237(9):1236–1256, 2008.

[44] R.A. Montante and J. Mills. Measuring information capacity in a vlsi analog logic circuit. *Technical Report 377, Computer Science Department, Indiana University*, March 1993.

[45] C. Moore. Unpredicitability and udecidability i dynamical systems. *Physical Review Letters*, 64(20):23–54, 1990.

[46] C. Moore. Smooth one-dimentional maps of the interval and the real line capable of universal computation. *Santa Fe Institute Working Paper 93*, pages 01–001, 1993.

[47] C. Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162:23–44, 1996.

[48] A. Mostowski. On computable sequences. *Fund. Math*, 44:37–51, 1957.

[49] J. Mycka. μ-recursion and infinite limits. *Theoretical Computer Science*, 302:123–133, 2003.

[50] J. Mycka and M. Piekarz. Przegląd zagadnień obliczalności analogowej. *Algorytmy, Metody i Programy Naukowe*, pages 125–132, 2004. materiały konferencyjne z LAFI 2004.

[51] F. Nijhout. Development and evolutio of butterfly wing patterns. *Smithsonian Inst. Press, Wash, DC*, 1991.

[52] P. Odifreddi. *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*. North Holland, 1989.

[53] P. Odifreddi. *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*, volume 2. North Holland, 1999.

[54] B. Orłowski, Z. Płochocki, and Z. Przyrowski. *Encyklopedia odkryć i wynalazków. Chemia, fizyka, medycyna, rolnictwo, technika.* Wiedza Powszechna, 1979. ISBN 83-214-0021-3.

[55] P. Orponen. A survey of continuous-time computational theory. *Advances in Algorithms, Languages and Complexity*, pages 209–224, 1997. Kluwer Academic Publishers.

[56] R. Péte. Recursive funktionen. *Akademischen Verlag*, 1951.

[57] M. Piekarz. Three simulations of Turing machine with the use of real recursive functions. *Annales UMCS Informatica AI 2*, pages 101–114, 2004.

[58] M. Piekarz. Extended Analog Computer and Real Recursive Functions. *Proceedings of Aplimat 2006*, pages 643–649, 2006.

[59] M. Piekarz. The Extended Analog Computer and Turing machine. *Annales UMCS Informatica AI*, 2:37–47, 2007.

[60] M. Piekarz. An introspect of simulation of nondeterministic Turing machine with a real-analytic function. *Journal of Applied Mathematics*, 2:248–254, 2009.

[61] M. Piekarz. The Extended Analog Computer and functions computable in digital sense. *Acta Cybernetica*, 19:749–764, 2010.

[62] M. B. Pour-El. Abstract computability and its relation to the general purpose analog computer. *Trans. Am. Math. Soc.*, 199:1–28, 1974.

[63] H. Rasiowa. *Wstęp do matematyki współczesnej*. PWN, 1984.

[64] D. Richardson. Some undecidable problems involving elementary functions of a real variable. *The Journal of Symbolic Logic*, 33(4):514–520, Decenber 1968.

[65] A. Romanowska. Uniwersalne metody algebry w informatyce. *Advances in Applied Mathematics*, 2009.

[66] L. A. Rubel. Some Mathematical Limitations of the General-Purpose Analog Computer. *Advances in Applied Mathematics*, 9:22–34, 1988.

[67] L. A. Rubel. The Extended Analog Computer. *Advances in Applied Mathematics*, 14:39–50, 1993.

[68] C. Shannon. Mathematical theory of the differential analyser. *Journal Mathematical Physics*, 20:337–354, 1941.

[69] H. Siegelmann and E. D. Sontag. On the computational power of neutral nets. *Journal of Comp. and Systems Sc.*, 50:132–150, 1995.

[70] H. T. Siegelmann. *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser, 1999.

[71] M. Sipser. *Introduction to the Theory of Computation*. Thomson, Course Tehnology, international Edition edition, 2006. Second Edition.

[72] E. Speaker. Nicht konstructiv beweisbare sätze er analysis. *Journal Symbolic Logic*, 14:145–158, 1949.

[73] G. Teschl. Ordinary differential equations and dynamical systems. 2007. Dostepne pod http://www.mat.univie.ac.at/∼gerald/ftp/book-ode/ode.pdf.

[74] A. M. Turing. On computable numbers, with an application to the "entscheidungsproblem". *Proccedings of the London Mathematical Society*, 42(2):230–265, 1936.

[75] P. van Emde Boas. *The Handbook of Theoretical Computer Science*, volume 1 of *Algoritm and Complexity*, chapter Machine Models and Simulation, pages 1–61. MIT Press, 1990. Cambridge, Massachusetts.

[76] K. Weihrauch. *Computable Analysis. An Introduction.* Springer edition, 2000.

[77] X. Zheng and K. Weihrauch. The arithmetical hierarchy of real numbers. *Math. Logic Quart.*, 47 (1):51–65, 2001.