*Leszek Dubiel**

# GENEROWANIE MASZYN TURINGA POPRZEZ ZASTOSOWANIE NOWYCH MODELI OBLICZENIOWYCH

*Dla każdego problemu obliczalnego istnieje algorytm jego rozwiązania, który może być przedstawiony w postaci konkretnej maszyny Turinga. Ze względu na prostotę tej maszyny programy są bardzo skomplikowane i nieprzejrzyste. Dlatego tworzy się inne modele obliczenia, na których można szybko i łatwo zapisywać algorytmy związane z danym typem problemu, a następnie symuluje się działanie tych modeli na maszynie Turinga. Niniejszy artykuł przedstawia metodologię znajdowania modelu obliczeniowego pasującego do danego typu problemów i przekształcenie go w odpowiednią maszynę Turinga na przykładzie obliczeń na liczbach naturalnych.*

**Słowa kluczowe:** *maszyna Turinga, automaty, teoria obliczeń*

# GENERATING TURING MACHINES BY USE OF OTHER COMPUTATION MODELS

*For each problem that can be solved there exists algorithm, which can be described with a program of Turing machine. Because this is very simple model programs tend to be very complicated and hard to analyse by human. The best practice to solve given type of problems is to define a new model of computation that allows for quick and easy programming, and then to emulate its operation with Turing machine. This article shows how to define most suitable model for computation on natural numbers and defines Turing machine that emulates its operation.*

**Keywords:** *Turing machine, automatons, theory of computation*

## 1. Wprowadzenie

### 1.1. Maszyna Turinga

W 1936 roku angielski matematyk Alan Turing opublikował pracę [1, 5, 8], w której formalnie zdefiniował pojęcie algorytmu. Opisał mechaniczne urządzenie wykonujące bardzo proste czynności na nieskończenie długiej taśmie. W tym czasie sformułowano również hipotezę Turinga–Churcha [5], która orzekała, że procesy, które są algorytmiczne w sensie intuicyjnym, to dokładnie te procesy, które mogą być zrealizowane na maszynie Turinga.

*Katedra Informatyki, Akademia Górniczo-Hutnicza, Leszek.Dubiel@dubielvitrum.pl

Początkowo większość naukowców była sceptycznie nastawiona do modelu Turinga [1]. Z jednej strony jego prostota wskazywała na możliwość istnienia algorytmów, które można zrealizować w sensie intuicyjnym, ale nie są możliwe do wykonania na maszynie Turinga. Z drugiej strony wszystkie znane algorytmy można było zapisać w postaci programu maszyny Turinga.

### 1.2. Cel rozważań

Prostota maszyny Turinga sprawia, że jej programowanie nie jest intuicyjne, i model ten funkcjonuje jedynie jako instrument teorii obliczeń, a nie jako narzędzie do obliczania złożoności programów w pracach inżynierskich. Celem artykułu jest pokazanie metody budowania maszyn Turinga. Można to pokazać, opierając się na nowych modelach obliczeniowych, które są lepiej przystosowane do wybranej dziedziny. Metodologia ta jest omówiona na przykładzie obliczeń arytmetycznych i **maszyny stertowej**.

Najpierw pokazana jest analiza obliczenia wykonywanego na papierze, która prowadzi do definicji maszyny działającej na napisach, czyli maszyny Turinga. Potem analogiczna analiza jest przeprowadzona na podstawie obserwacji działań na liczbach, czego wynikiem jest definicja nowego modelu. Na końcu rozważań określono transformację pomiędzy tymi modelami.

Mając problem liczbowy, można go łatwo zapisać na maszynie stertowej, a tę – automatycznie przekształcić na maszynę Turinga. Dzięki temu programowanie maszyny Turinga staje się znacznie prostsze.

## 2. Analiza obliczenia

### 2.1. Operacje na znakach

Turing rozpoczął rozważania od najbardziej elementarnych pytań – czym jest algorytm i jak pojmujemy intuicyjnie proces jego wykonania [8, 1]. Wyszedł od obserwacji człowieka realizującego dowolne obliczenie algorytmiczne [9], takie jak na przykład pisemne dodawanie dwóch liczb dziesiętnych. Zauważył, że rachmistrz najpierw zapisuje argumenty na kartce papieru, a następnie wykonuje szereg czynności, które doprowadzą go do wypisania wyniku obliczenia. Reguły, według których postępował rachmistrz, mogły być skomplikowane, ale raz ustalone – pozwalały na mechaniczne wykonywanie algorytmu. Turing dokonał analizy akcji składających się na obliczenie i stwierdził, że każda z nich jest albo czynnością syntaktyczną polegającą na zapisaniu lub odczytaniu znaku z kartki, albo powoduje przeniesienie uwagi rachmistrza na inne miejsce kartki. Dalsze rozważania pozwoliły na uproszczenie całego procesu. Turing zastąpił papier taśmą podzieloną na komórki, z których każda była pusta lub zawierała dokładnie jeden symbol. Nad nią umieścił głowicę, która działała krokowo i za każdym razem mogła zapisać symbol do komórki, nad którą się znajdowała, oraz przesunąć się w lewo lub w prawo o jedno pole. Turing uznał, że są to czynności ko-

nieczne i wystarczające do zapisania każdego algorytmu wykonywanego na papierze, więc dokonał formalizacji stworzonego modelu i opublikował koncepcję.

## 2.2. Działania na liczbach

Aby stworzyć arytmetyczny model obliczenia, trzeba przeprowadzić dokładną analizę działań, jakie wykonuje człowiek, analogicznie jak postąpił Turing, ale w odniesieniu do działań na liczbach.

Człowiek pierwotny oznaczał liczbę posiadanych przedmiotów, zawiązując węzełki na sznurze lub nacinając kości zwierzęce [7]. Okazuje się, że mógł to uczynić, nie będąc nawet świadomym pojęcia liczby. Przykładowo gdy chciał oznaczyć dużą liczbę zwierząt, która znajdowała się w pewnej zagrodzie, to najpierw przygotowywał sznur bez węzełków. Potem dzielił zagrodę na dwie części – w pierwszej z nich umieszczał wszystkie zwierzęta z zagrody, a drugą pozostawiał pustą. Następnie przenosił zwierzęta z pierwszej części do drugiej i za każdym razem zawiązywał węzełek na sznurze. Postępował w ten sposób, dopóki były zwierzęta w pierwszej części zagrody. Po skończeniu obliczenia liczba węzełków na sznurze była równa liczbie posiadanych zwierząt nawet wtedy, gdy początkowo w zagrodzie nie było zwierząt.

Algorytm wykonywany przez człowieka pierwotnego można opisać bardziej formalnie. Niech do zbioru $Z$ należą wszystkie zwierzęta, które trzeba policzyć, a zbór $W$ jest kolekcją węzełków zawiązanych na sznurze. Zbiór $Z$ jest dany na początku obliczenia jako argument, a zbiór $W$ jest wynikiem obliczenia i jego stworzenie musi być opisane w algorytmie. Wówczas można przykładowe obliczenie zapisać w postaci:

1. Stwórz zbiór pusty $W$.
2. Jeśli zbiór $Z$ jest pusty, to idź do kroku 6.
3. Usuń element ze zbioru $Z$.
4. Dodaj element do zbioru $W$.
5. Idź do kroku 2.
6. Usuń zbiór $Z$.
7. Koniec obliczenia.

## 2.3. Czynności elementarne

Do wykonania powyższego algorytmu potrzebne są następujące czynności elementarne:

1. Stwórz zbiór pusty $X$.
2. Usuń zbiór $X$.
3. Dodaj element do zbioru $X$.
4. Usuń element ze zbioru $X$.
5. Jeśli zbiór jest pusty, to idź do kroku $k$.
6. Idź do kroku $k$.
7. Koniec obliczenia.

Wykonanie pierwszych czterech czynności wiąże się z wybraniem zbioru $X$ spośród wszystkich dostępnych zbiorów, jakie biorą udział w obliczeniu, zaś piąta i szósta czynność wymaga przeglądania programu w celu znalezienia kroku $k$, który zostanie wykonany jako następny. Są to czynności, które w algorytmie występują niejawnie. Jednak nie zostaną one włączone do zbioru czynności elementarnych, gdyż spowodowałoby to konieczność wprowadzenia dalszych niejawnych czynności. Są one interpretowane jako sprzężenie modułu sterującego z danymi i programem tworzonego modelu. W algorytmie liczenia zwierząt procesorem jest człowiek pierwotny, który intuicyjnie wykonuje program, czyli bezwiednie odnajduje w nim odpowiednie kroki, natomiast danymi są zbiory, które również bezbłędnie są przez tego człowieka rozpoznawane.

## 2.4. Kodowanie liczb

Rozważania dotyczą obliczeń na liczbach, więc tworzony model obliczenia musi kodować liczby w swojej pamięci. Kodem liczby naturalnej jest kolekcja obiektów, a zatem w celu zachowania jednolitości kodowania należałoby ustalić, jakiego typu obiekty będą rozpatrywane. Na przykład liczbę naturalną $n$ można kodować jako kolekcję $n$ kamieni. Kodowanie kojarzy się przede wszystkim z pisaniem symboli na kartce papieru, warto więc pozostać przy tej koncepcji. Zatem najlepszym rozwiązaniem jest kodowanie liczb naturalnych w postaci zbiorów symboli zapisanych na kartce papieru.

Każda liczba będzie kodowana na osobnej **kartce** papieru, czyli innymi słowy – każda kartka papieru będzie kodem dokładnie jednej liczby naturalnej [3]. Dla ustalenia uwagi przyjmujemy, że kartka papieru jest biała, i możemy na niej umieścić dowolną liczbę czarnych **kropek**. Kartka jest formalnie interpretowana jako zbiór kropek, więc zamiast mówić, że na kartce napisano $n$ kropek, można mówić, że do tej kartki należy $n$ kropek. Kartka koduje liczbę naturalną $n$ wtedy i tylko wtedy, gdy zapisano na niej dokładnie $n$ czarnych kropek [8]. Zgodnie z tą konwencją pusta kartka jest kodem zera. Kartka, na której napisano jedną kropkę, jest kodem liczby jeden, kartka zawierająca dwie kropki jest kodem liczby dwa, natomiast kartka, na której umieszczono trzy kropki – koduje liczbę trzy. Ogólnie mówiąc, jeśli pewna kartka jest kodem liczby naturalnej $n$ oraz zostanie do niej dopisana jeszcze jedna kropka, to wtedy kartka ta staje się kodem następnika liczby $n$. Nie ma żadnych ograniczeń na wielkość liczby $n$, więc do każdej kartki można dopisać jeszcze jedną kropkę, czyli kartka może zawierać dowolną liczbę czarnych kropek.

Przedmiotem obliczenia jest zazwyczaj wiele liczb, a każda z nich musi być zapisana w pamięci maszyny w określonym porządku. Kartki można układać jedna **na** drugiej, co pozwala kodować całe wektory liczb naturalnych. Taka struktura będzie nazywana **stertą** liczb naturalnych [3]. Sterta pusta koduje zbiór pusty. Jedna kartka, która koduje liczbę naturalną $x_1$, jest stertą, która koduje wektor $(x_1)$. Jeśli na tej kartce położy się kartkę, która jest kodem liczby $x_2$, to wówczas tak utworzona sterta koduje wektor postaci $(x_1, x_2)$. Jeśli na tej stercie położy się kolejną kartkę, która koduje liczbę $x_3$, to sterta będzie kodem wektora $(x_1, x_2, x_3)$. Ogólnie mówiąc, jeśli

na stercie, która koduje wektor $(x_1, x_2, \ldots, x_m)$, położy się kartkę kodującą liczbę $x_n$, to ta sterta stanie się kodem wektora $(x_1, x_2, \ldots, x_m, x_n)$.

Jeśli sterta koduje wektor liczb naturalnych postaci $(x_1, x_2, \ldots, x_n)$, to znaczy, że jej **głębokość** wynosi $n$, a kartka kodująca liczbę $x_n$ znajduje się na głębokości zero. Jeśli na pewnej kartce sterty jest położona kartka, która znajduje się na głębokości $m$, to wtedy ta pierwsza kartka jest na głębokości będącej następnikiem liczby $m$. Kartkę, która znajduje się na głębokości zero, nazywa się **wierzchołkiem** sterty. Na przykład, jeśli sterta koduje wektor $(x_1, x_2, x_3)$, to kartka kodująca liczbę $x_3$ jest jej wierzchołkiem, gdyż znajduje się na głębokości zero, kartka kodująca liczbę $x_2$ jest na głębokości jeden, a kartka kodująca liczbę $x_1$ na głębokości dwa. Ponieważ nie ma żadnych ograniczeń na długość wektora, który może być kodowany na stercie, więc można na niej położyć dowolnie dużo kartek, czyli sterta może mieć dowolnie dużą głębokość. Kartka znajdująca się na głębokości $n$ pewnej sterty jest nazywana ***n*-tą kartką** tej sterty, lub **kartką *n***.

# 3. Maszyna stertowa

## 3.1. Operacje elementarne

Definicja modelu obliczeniowego zawiera zbiór operacji elementarnych wykonywanych przez procesor. Proces obliczenia jest ciągiem zdarzeń modyfikujących stertę. Modyfikacje te dotyczą zarówno układu kartek na stercie, jak również liczby kropek na poszczególnych kartkach. Można je opisać w postaci niżej podanego zestawu instrukcji.

1. Dodaj kartkę na głębokości $n$ sterty.
2. Usuń kartkę na głębokości $n$ sterty.
3. Dodaj kropkę do kartki na głębokości $n$ sterty.
4. Usuń kropkę z kartki na głębokości $n$ sterty.
5. Sprawdź, czy kartka na głębokości $n$ sterty jest pusta.

Każda z tych operacji może zostać wykonana na dowolnej głębokości sterty, która jest dozwolona. Oznacza to, że jeśli sterta ma głębokość $n$, to wszystkie operacje można wykonać na głębokości mniejszej niż $n$ i dodatkowo możemy dodać kartkę na głębokości równej $n$. Wynika stąd, że jeśli sterta jest pusta, to wówczas dodanie kartki jest jedyną instrukcją możliwą do wykonania. Operacja usunięcia kropki z pustej kartki jest niewykonalna.

Przy budowaniu algorytmów działających na liczbach powyższy zestaw operacji jest niewygodny, gdyż pewne często powtarzające się czynności muszą być kodowane za pomocą wielu instrukcji, a inne są zbyt uniwersalne i możne je uprościć. Zatem nowy model obliczenia będzie miał niżej wymieniony zbiór operacji elementarnych.

1. `ZER` – dodaj pustą kartkę na wierzchołek sterty.
2. `INC` – dodaj kropkę do kartki na wierzchołku sterty.
3. `DEC` – usuń kropkę z kartki na wierzchołku sterty.
4. `CPY` $n$ – skopiuj $n$-tą kartkę sterty na jej wierzchołek.

5. `DEL` $n$ – usuń $n$-tą kartkę ze sterty.
6. `JMZ` $k$ – jeśli kartka na wierzchołku jest pusta, to idź do kroku $k$.
7. `JMP` $k$ – idź do kroku $k$.
8. `STP` – koniec obliczenia.

Wszystkie instrukcje z wyjątkiem kopiowania kartek są identyczne z czynnościami, jakie przedstawiliśmy wcześniej, lub są ich szczególnymi przypadkami. Kopiowanie kartek wiąże się z wykonaniem pewnego algorytmu zbudowanego na bazie tamtych czynności. Zakładając, że dana jest sterta o głębokości nie mniejszej niż $n$ i należy skopiować $n$-tą kartkę tej sterty, w wyniku podjętych działań na wierzchołku sterty powinna zostać umieszczona kartka zawierająca tyle samo kropek, co kartka na głębokości $n$. Niech $n, n'$ oraz $n''$ są kolejnymi liczbami naturalnymi. Wówczas taką modyfikację sterty można otrzymać w wyniku wykonania poniższego algorytmu.

1. Dodaj pustą kartkę na głębokości 0.
2. Dodaj pustą kartkę na głębokości $n'$.
3. Jeśli kartka $n''$ jest pusta, to idź do kroku 8.
4. Usuń kropkę z kartki na głębokości $n''$.
5. Dodaj kropkę do kartki na głębokości $n'$.
6. Dodaj kropkę do kartki na głębokości 0.
7. Idź do kroku 3.
8. Usuń kartkę na głębokości $n''$.
9. Koniec obliczenia.

W pierwszym kroku na wierzchołek sterty wkładana jest pusta kartka, a zatem kartka, która ma być kopiowana, znajdzie się na głębokości $n'$. W drugim kroku dodaje się kolejną pustą kartkę, ale tym razem zostanie ona położona na kartce, która ma być kopiowana, czyli na głębokości równej $n'$. Zatem puste kartki, jakie zostały umieszczone na stercie, znajdują się na głębokości 0 i $n'$, natomiast kartka przeznaczona do kopiowania przesunęła się na głębokość $n''$. Algorytm polega na usuwaniu kropek z kartki $n''$ i dodawaniu ich do kartek 0 i $n'$. Obliczenie kończy się, jeśli kartka $n''$ jest pusta. Wtedy zostaje one usunięta i na stercie pozostają kartki 0 i $n'$ zawierające tyle samo kropek, ile miała kartka $n''$ na początku obliczenia. Porównując stertę przed i po realizacji algorytmu, można się przekonać, że efekt uzyskany w wyniku jego wykonania może być interpretowany jako skopiowanie $n$-tej kartki z głębi sterty na jej wierzchołek.

## 3.2. Definicja modelu

Maszyna stertowa jest modelem obliczeniowym, którego pamięć stanowi sterta liczb naturalnych, a procesorem jest rachmistrz wykonujący operacje według programu zapisanego przy użyciu instrukcji `ZER`, `INC`, `DEC`, `CPY`, `DEL`, `JMZ`, `JMP` oraz `STP`.

Obliczenie na tym modelu składa się z kilku etapów. Najpierw na stercie kodowane są argumenty obliczenia, a w programie wyróżnia się instrukcję, która będzie wykonana jako pierwsza. Następnie rachmistrz, postępując zgodnie z programem, wykonuje różne działania na stercie. Jeśli obliczenie kończy się sukcesem, to STP jest ostatnią instrukcją zrealizowaną przez rachmistrza i wówczas zawartość sterty interpretowana jest jako prawidłowy wynik obliczenia. W przeciwnym wypadku rachmistrz albo nigdy nie kończy pracy, albo zatrzymuje się na czynności niemożliwej do wykonania, co będzie określane jako zawieszenie pracy rachmistrza, albo kończy działanie po wykonaniu ostatniej instrukcji programu. W takiej sytuacji funkcja obliczana przez maszynę stertową nie jest określona dla zadanych argumentów.

## 4. Transformacja modeli

W rozdziale tym przedstawiona jest ogólna metoda konstruowania maszyny Turinga wykonującej identyczne obliczenie jak dana maszyna stertowa. W tym celu najpierw należy określić kodowanie wektorów liczb naturalnych na taśmie maszyny Turinga, a potem trzeba wygenerować odpowiedni program. W ostatniej części podany jest przykład dodawania liczb.

### 4.1. Generowanie pamięci

Liczbę naturalną $n$ można kodować na taśmie maszyny Turinga w postaci ciągu jedynek o długości równej następnikowi liczby $n$, który jest zakończony zerem. Wówczas 10 jest kodem liczby zero, 110 kodem liczby jeden, natomiast 1110 jest kodem liczby dwa. Wszystkie symbole kodu liczby zapisuje się w kolejnych komórkach taśmy, a na początku obliczenia głowicę umieszcza się nad komórką zawierającą ostatni symbol kodu. Zatem, aby w miejscu wskazanym przez głowicę zakodować liczbę $m$, należy wykonać program:

```
0     * 1 R 1        H
1       1 R 2        H
2       1 R 3        H
.....................
n       1 R end      H
end       H          H.
```

Wektor liczb naturalnych kodowany jest poprzez kolejne zapisanie kodów wszystkich jego współrzędnych. Na przykład ciąg:

```
10 110 11110 1111110
```

jest kodem wektora $(0, 1, 3, 5)$. Wektor można zapisać na taśmie poprzez kolejne wykonanie powyższego programu dla wszystkich współrzędnych tego wektora. Głowicę zawsze umieszcza się nad komórką zawierającą ostatni bit kodu wektora, czyli na zerze należącym do kodu jego ostatniej współrzędnej.

## 4.2. Generowanie programu

Konstrukcja programu polega na zamianie każdej instrukcji maszyny stertowej na sekwencję instrukcji maszyny Turinga w ten sposób, że jeśli sterta, która na początku obliczenia koduje wektor liczb naturalnych $\bar{x}$, jest przekształcana w wyniku wykonania programu maszyny stertowej do postaci kodującej wektor $\bar{y}$, to również taśma kodująca na początku obliczenia wektor $\bar{x}$ zostanie przekształcona w wyniku wykonania odpowiedniego programu maszyny Turinga do postaci kodującej wektor $\bar{y}$. Równoważność ta dotyczy także sytuacji, w której rachmistrz zawiesza swoje działanie, więc przyjmujemy, że w takim przypadku maszyna Turinga będzie zatrzymywała się w stanie oznaczonym `err`.

Budowę programu maszyny Turinga rozpoczyna się od zamiany każdej instrukcji maszyny stertowej na odpowiedni zestaw linii maszyny Turinga. Ponieważ każda z instrukcji `ZER`, `INC`, `DEC`, `CPY`, `DEL`, `JMZ`, `JMP` oraz `STP` ma zdecydowanie różne działanie, to poniżej rozważane są wszystkie te instrukcje po kolei.

1. Wykonanie instrukcji `ZER` powoduje położenie na wierzchołku sterty pustej kartki, a zatem do wektora zakodowanego na stercie dodawana jest jeszcze jedna współrzędna. Ponieważ głowica maszyny Turinga obserwuje ostatnią komórkę wektora zakodowanego na taśmie, więc najpierw przesunie się ona w prawo i do kolejnych komórek wpisze kod liczby zero, który – zgodnie z powyższym – jest sekwencją `10`. Odpowiedni program dla maszyny Turinga przybiera wówczas postać:

```
0       0 R 1       H
1       1 R end     H.
```

   Zakładając, że w chwili początkowej taśma koduje wektor $(3,2)$, możemy przeanalizować działanie tego programu, patrząc na tabelę 1. Prawa kolumna zawiera numer kroku, natomiast lewa postać taśmy przed jego wykonaniem. Gwiazdka jest ustawiona za komórką obserwowaną przez głowicę. Zgodnie z oczekiwaniami, na końcu obliczenia taśma koduje wektor $(3,2,0)$.

**Tabela 1**

| | |
|---|---|
| 1 1 1 1 0 1 1 1 0*0 0 | 0 |
| 1 1 1 1 0 1 1 1 0 0*0 | 1 |
| 1 1 1 1 0 1 1 1 0 1 0* | end |

2. Wykonując instrukcję `INC`, rachmistrz dopisuje jedną kropkę do kartki na wierzchołku sterty, więc maszyna Turinga powinna zapisać jedynkę w polu obserwowanym przez głowicę, a potem przesunąć głowicę w prawo. Niestety takie działanie daje fałszywy wynik w przypadku, kiedy sterta jest pusta, gdyż wtedy rachmistrz zawiesza swoje działanie, podczas gdy maszyna Turinga kończy je sukcesem.

W tej sytuacji musimy napisać bardziej skomplikowany program, który wówczas przyjmuje postać:

```
0       0 L 1          H
1       0 S err    1 R 2
2       1 R end        H.
```

Maszyna Turinga najpierw przesuwa głowicę w lewo i sprawdza zawartość obserwowanej komórki. Jeśli jest tam jeden, to modyfikuje taśmę i kończy obliczenie w stanie `end`. W przeciwnym wypadku sygnalizuje błąd, czyli zatrzymuje się w stanie `err`. Zakładając, że w chwili początkowej taśma jest kodem wektora $(3, 2)$, działanie programu przedstawia się jak w tabeli 2.

**Tabela 2**

| | |
|---|---|
| 1 1 1 1 0 1 1 1 0*0 | 0 |
| 1 1 1 1 0 1 1 1*0 0 | 1 |
| 1 1 1 1 0 1 1 1 0*0 | 2 |
| 1 1 1 1 0 1 1 1 1 0* | end |

3. Wykonanie instrukcji `DEC` jest bardziej skomplikowane, gdyż rachmistrz zawiesza swoje działanie nie tylko w przypadku, gdy sterta jest pusta, ale również wtedy, gdy kartka na jej wierzchołku nie zawiera żadnych kropek. Maszyna Turinga najpierw przesuwa głowicę w lewo i jeśli znajdzie się nad komórką zawierającą zero, to znaczy, że mamy do czynienia ze stertą pustą, czyli działanie kończy się w stanie `err`.

   W przeciwnym wypadku zamiast jedynki wpisywane jest zero i głowica powtórnie przesunie się w lewo. Jeśli tym razem obserwowana komórka jest pusta, to obliczenie kończy się błędem, ponieważ argumentem była liczba zero. Jeśli jednak w komórce będzie jedynka, to maszyna zatrzyma się w stanie `end`, a na taśmie zostanie zakodowany prawidłowy wynik. Takie działanie maszyny opisuje się przy użyciu programu:

```
0       0 L 1          H
1       0 S err    0 L 2
2       0 S err    1 R end,
```

którego działanie przedstawia tabela 3.

**Tabela 3**

| | |
|---|---|
| 1 1 1 1 0 1 1 1 0* | 0 |
| 1 1 1 1 0 1 1 1*0 | 1 |
| 1 1 1 1 0 1 1*0 0 | 2 |
| 1 1 1 1 0 1 1 0*0 | end |

4. Instrukcja `CPY` nie jest czynnością atomową, gdyż wcześniej została zdefiniowana jako pewien algorytm. Dlatego też odwzorowanie jej na maszynie Turinga jest bardziej skomplikowane niż w przypadku wcześniejszych operacji.

   Maszyna Turinga rozpoczyna obliczenie od znalezienia komórki zawierającej pierwszy bit kodu kopiowanej liczby. W tym celu przesuwa głowicę w lewo, badając równocześnie, czy nie znalazła się ona na dnie sterty, co oznaczałoby błąd obliczenia. Następnie wykonywana jest procedura iteracyjna, która kopiuje kolejne jedynki kodu znalezionej liczby na koniec kodu całego wektora. Każda jedynka jest chwilowo zamieniana na zero, aby jej pozycja była wyróżniona. Potem głowica przesuwa się na koniec kodu całego wektora, aby tam umieścić kopiowaną jedynkę. Następnie wraca do oznaczonej pozycji i odtwarza początkową zawartość komórki, zamieniając zero na jeden. W dalszej kolejności wykonuje ruch w prawo i jeśli okaże się, że obserwowana komórka jest pusta, to znaczy, że wszystkie jedynki zostały skopiowane. Wówczas głowica przesuwa się na koniec kodu wektora i obliczenie kończy się w stanie `end`. W przeciwnym wypadku kolejna jedynka zostanie oznaczana zerem i procedura będzie realizowana powtórnie.

   Dalej przedstawiony jest gotowy program maszyny Turinga będący odpowiednikiem instrukcji `CPY 1`. Jego wykonanie spowoduje skopiowanie kodu przedostatniej współrzędnej wektora zapisanego na taśmie.

```
0      0 L 1        H
1      0 S err    1 L 2
2      0 L 3      1 L 2
3      0 S err    1 L 4
4      0 R 5      1 L 4
5      0 R 12     0 R 6
6      0 R 7      1 R 6
7      0 R 8      1 R 7
8      1 L 9      1 R 8
9      0 L 10     1 L 9
10     0 L 11     1 L 10
11     1 R 5      1 L 11
12     0 R 13     1 R 12
13     0 S end    1 R 13.
```

   W tabeli 4 prezentujemy fragmenty przykładowego obliczenia, które zostało wykonane na taśmie kodującej wektor $(3, 2)$. Jako ciekawostkę można podać, że w rzeczywistości tak proste kopiowanie wymaga wykonania przez maszynę Turinga prawie stu kroków.

**Tabela 4**

| | |
|---|---|
| 0 1 1 1 1 0 1 1 1 0*0 0 0 0 0 | 0 |
| 0 1 1 1 1 0 1 1 1*0 0 0 0 0 0 | 1 |
| 0 1 1 1 1 0 1 1*1 0 0 0 0 0 0 | 2 |
| 0 1 1 1 1 0*1 1 1 0 0 0 0 0 0 | 2 |
| 0 1 1 1 1*0 1 1 1 0 0 0 0 0 0 | 3 |
| 0 1 1 1*1 0 1 1 1 0 0 0 0 0 0 | 4 |
| 0*1 1 1 1 0 1 1 1 0 0 0 0 0 0 | 4 |
| 0 1*1 1 1 0 1 1 1 0 0 0 0 0 0 | 5 |
| 0 0 1*1 1 0 1 1 1 0 0 0 0 0 0 | 6 |
| 0 0 1 1 1 0*1 1 1 0 0 0 0 0 0 | 6 |
| 0 0 1 1 1 0 1*1 1 0 0 0 0 0 0 | 7 |
| 0 0 1 1 1 0 1 1 1 0*0 0 0 0 0 | 7 |
| 0 0 1 1 1 0 1 1 1 0 0*0 0 0 0 | 8 |
| 0 0 1 1 1 0 1 1 1 0*1 0 0 0 0 | 9 |
| 0 0 1 1 1 0 1 1 1*0 1 0 0 0 0 | 10 |
| 0 0 1 1 1 0*1 1 1 0 1 0 0 0 0 | 10 |
| 0 0 1 1 1*0 1 1 1 0 1 0 0 0 0 | 11 |
| 0 0*1 1 1 0 1 1 1 0 1 0 0 0 0 | 11 |
| 0 1 1*1 1 0 1 1 1 0 1 0 0 0 0 | 5 |
| 0 1 0 1*1 0 1 1 1 0 1 0 0 0 0 | 6 |
| 0 1 0 1 1 0*1 1 1 0 1 0 0 0 0 | 6 |
| 0 1 0 1 1 0 1*1 1 0 1 0 0 0 0 | 7 |
| 0 1 0 1 1 0 1 1 1 0*1 0 0 0 0 | 7 |
| 0 1 0 1 1 0 1 1 1 0 1*0 0 0 0 | 8 |
| 0 1 0 1 1 0 1 1 1 0 1 0*0 0 0 | 8 |
| 0 1 0 1 1 0 1 1 1 0 1*1 0 0 0 | 9 |
| ..... | ... |
| 0 1 1 1 0*0 1 1 1 0 1 1 1 1 1 0 | 11 |
| 0 1 1 1 1 0*1 1 1 0 1 1 1 1 1 0 | 5 |
| 0 1 1 1 1 0 1*1 1 0 1 1 1 1 1 0 | 12 |
| 0 1 1 1 1 0 1 1 1 0*1 1 1 1 1 0 | 12 |
| 0 1 1 1 1 0 1 1 1 0 1*1 1 1 0 | 13 |
| 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0* | 13 |
| 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0* | end |

5. Wykonanie instrukcji `DEL` jest uzależnione od argumentu, gdyż jeśli jest nim zero, to operacja sprowadza się do trywialnego kasowania jedynek na lewo od głowicy, podczas gdy w przypadku przeciwnym wykonywany jest program znacznie bardziej skomplikowany. Poniżej osobno omówiono obie te sytuacje.

   Maszyna Turinga realizując instrukcję `DEL 0` najpierw sprawdza zawartość komórki na lewo od głowicy. Jeśli zawiera zero, to kończy działanie w stanie `err`,

gdyż w sytuacji analogicznej rachmistrz zawiesiłby swoje działanie. W przeciwnym wypadku głowica przesuwa się w lewo opróżniając wszystkie komórki zawierające kropkę. Takie obliczenie opisuje się w postaci programu:

```
0       0 L 1          H
1       0 S err     0 L 2
2       0 S end     0 L 2.
```

Analizując tabelę 5 można prześledzić jego działanie w przypadku, gdy na wejściu jest taśma kodująca wektor $(3,2)$.

**Tabela 5**

| | |
|---|---|
| 1 1 1 1 0 1 1 1 0* | 0 |
| 1 1 1 1 0 1 1 1*0 | 1 |
| 1 1 1 1 0 1 1*0 0 | 2 |
| 1 1 1 1 0 1*0 0 0 | 2 |
| 1 1 1 1 0*0 0 0 0 | 2 |
| 1 1 1 1 0*0 0 0 0 | end |

Idea kasowania kodów liczb, które nie są ostatnimi współrzędnymi wektora zapisanego na taśmie maszyny Turinga, polega na przesuwaniu w lewo zawartości komórek znajdujących się na prawo od kodu kasowanej liczby do momentu, aż kod ten zostanie usunięty z taśmy. Maszyna Turinga najpierw sprawdza, czy sterta ma wystarczającą głębokość, a następnie wykonuje procedurę iteracyjną, w której głowica krąży nad modyfikowanymi komórkami, za każdym razem przesuwając zawartość taśmy o jedno pole w lewo. Ponieważ jest to dość zawiłe postępowanie, można przykładowo rozważyć instrukcję `DEL 2`. Kasuje ona trzecią od końca współrzędną wektora kodowanego na taśmie, więc odpowiedni program przyjmuje postać:

```
0       0 L 1          H
1       0 S err     0 L 2
2       1 L 3       1 L 2
3       0 S err     0 L 4
4       1 L 5       1 L 4
5       0 S err     0 R 6
6       0 R 7       1 R 6
7       0 L 8       1 R 7
8         H         0 L 9
9       1 L 10      1 L 9
10        H         0 L 11
11      1 L 12      1 L 11
12      0 R 13      0 R 6
13      0 R 14      1 R 13
14      0 S end     1 R 14.
```

Zakładając, że na taśmie zakodowano wektor liczb naturalnych $(3, 2, 1)$, działanie programu przedstawia tabela 6, prezentująca najważniejsze punkty obliczenia.

**Tabela 6**

| | |
|---|---|
| 0 1 1 1 1 0 1 1 1 0 1 1 0* | 0 |
| 0 1 1 1 1 0 1 1 1 0 1 1*0 | 1 |
| 0 1 1 1 1 0 1 1 1 0 1*0 0 | 2 |
| 0 1 1 1 1 0 1 1 1 0*1 0 0 | 2 |
| 0 1 1 1 1 0 1 1 1*1 1 0 0 | 3 |
| 0 1 1 1 1 0 1 1*0 1 1 0 0 | 4 |
| 0 1 1 1 1 0*1 1 0 1 1 0 0 | 4 |
| 0 1 1 1 1*1 1 1 0 1 1 0 0 | 5 |
| 0 1 1 1 0 1*1 1 0 1 1 0 0 | 6 |
| 0 1 1 1 0 1 1 1 0*1 1 0 0 | 6 |
| 0 1 1 1 0 1 1 1 0 1*1 0 0 | 7 |
| 0 1 1 1 0 1 1 1 0 1 1 0*0 | 7 |
| 0 1 1 1 0 1 1 1 0 1 1*0 0 | 8 |
| 0 1 1 1 0 1 1 1 0 1*0 0 0 | 9 |
| 0 1 1 1 0 1 1 1 0*1 0 0 0 | 9 |
| 0 1 1 1 0 1 1 1*1 1 0 0 0 | 10 |
| 0 1 1 1 0 1 1*0 1 1 0 0 0 | 11 |
| 0 1 1 1 0*1 1 0 1 1 0 0 0 | 11 |
| 0 1 1 1*1 1 1 0 1 1 0 0 0 | 12 |
| 0 1 1 0 1*1 1 0 1 1 0 0 0 | 6 |
| 0 1 1 0 1 1 1 0*1 1 0 0 0 | 6 |
| 0 1 1 0 1 1 1 0 1*1 0 0 0 | 7 |
| 0 1 1 0 1 1 1 0 1 1 0*0 0 | 7 |
| ..... | ... |
| 0 0 1 1 1 0 1 1 0*0 0 0 0 | 7 |
| 0 0 1 1 1 0 1 1*0 0 0 0 0 | 8 |
| 0 0 1 1 1 0 1*0 0 0 0 0 0 | 9 |
| 0 0 1 1 1 0*1 0 0 0 0 0 0 | 9 |
| 0 0 1 1 1*1 1 0 0 0 0 0 0 | 10 |
| 0 0 1 1*0 1 1 0 0 0 0 0 0 | 11 |
| 0 0*1 1 0 1 1 0 0 0 0 0 0 | 11 |
| 0*1 1 1 0 1 1 0 0 0 0 0 0 | 12 |
| 0 1*1 1 0 1 1 0 0 0 0 0 0 | 13 |
| 0 1 1 1 0*1 1 0 0 0 0 0 0 | 13 |
| 0 1 1 1 0 1*1 0 0 0 0 0 0 | 14 |
| 0 1 1 1 0 1 1*0 0 0 0 0 0 | 14 |
| 0 1 1 1 0 1 1*0 0 0 0 0 0 | end |

Warto zauważyć, że głowica cały czas oscyluje wokół kodów drugiej i trzeciej współrzędnej, przesuwając je w każdej iteracji o jedno pole w lewo, co powoduje konsekwentne usuwanie kodu pierwszej współrzędnej wektora. Wydawać by się mogło, że przedstawiony algorytm jest sztucznie skomplikowany, jednak jest to rozwiązanie, które gwarantuje, iż pierwszy znak kodu wektora po wykonaniu operacji będzie znajdował się dokładnie w tym samym miejscu, co przed wykonaniem instrukcji.

6. Dla instrukcji JMZ zawsze powstaje taki sam program. Najpierw głowica przemieszcza się w lewo i automat sprawdza, czy w komórce znajduje się jeden. Jeśli jest zero, to maszyna kończy obliczenie w stanie `err`, gdyż rachmistrz w tej sytuacji zawiesiłby swoje działanie. W przeciwnym wypadku głowica po raz kolejny przesuwa się w lewo, po czym sprawdza zawartość obserwowanej komórki, aby rozstrzygnąć, czy badany kod symbolizuje liczbę zero, i skok musi być wykonany, czy też inną liczbę naturalną, co spowoduje kontynuację obliczenia. W pierwszym przypadku maszyna kończy działanie w stanie `tmp`, natomiast w sytuacji przeciwnej skok nie jest wykonywany, gdyż automat kończy pracę w stanie `end`. Odpowiedni program przyjmuje postać:

```
0      0 L 1        H
1      0 S err    1 L 2
2      0 R 3      1 R 4
3        H        1 R tmp
4        H        1 R end.
```

Tabele 7 i 8 obrazują dwie różne ścieżki postępowania w zależności od tego, jaka jest ostatnia współrzędna wektora kodowanego na taśmie.

**Tabela 7**

| | |
|---|---|
| 1 1 1 1 0 1 0* | 0 |
| 1 1 1 1 0 1*0 | 1 |
| 1 1 1 1 0*1 0 | 2 |
| 1 1 1 1 0 1*0 | 3 |
| 1 1 1 1 0 1 0* | tmp |

**Tabela 8**

| | |
|---|---|
| 1 0 1 1 1 1 0* | 0 |
| 1 0 1 1 1 1*0 | 1 |
| 1 0 1 1 1*1 0 | 2 |
| 1 0 1 1 1 1*0 | 4 |
| 1 0 1 1 1 1 0* | end |

7. Najprostszy program odpowiada instrukcji JMP, gdyż składa się on tylko z jednej linii postaci:

```
0      0 S tmp    1 S tmp.
```

8. Dla instrukcji STP również powstaje jedna linia programu maszyny Turinga, ale tym razem przyjmuje ona postać:

```
X        H        H,
```

gdzie X jest nazwą lub liczbą naturalną identyczną z oznaczeniem instrukcji STP w programie maszyny stertowej. Dzięki temu można ustalić zależności pomiędzy stanami końcowymi maszyny stertowej i maszyny Turinga. Na przykład, jeśli rachmistrz, mając na stercie kod pewnego wektora, kończy działanie po wykonaniu instrukcji oznaczonej symbolem **end**, to maszyna Turinga, mając na taśmie kod tego samego wektora, również kończy obliczenie w stanie **end**.

Po zamianie wszystkich instrukcji programu danej maszyny stertowej na odpowiednie zestawy linii maszyny Turinga wprowadza się unikatowe numery we wszystkich stworzonych instrukcjach. Z powodów wyjaśnionych powyżej nie zmienia się nazw **end** i **err**. Następnie gwiazdką oznacza się pierwszą linię programu maszyny Turinga, który został stworzony dla instrukcji maszyny stertowej, od której rachmistrz rozpoczyna działanie. Na końcu kolejno zapisuje się wszystkie zestawy linii, przy czym zamiast **end** wstawia się numer pierwszej linii programu, który będzie napisany następny. Jeśli w programie maszyny stertowej nie ma instrukcji **err**, to do programu maszyny Turinga dopisuje się linię postaci:

```
err        H          H.
```

Jeśli ostatnią instrukcją programu maszyny stertowej nie jest ani STP, ani JMP, dopisuje się instrukcję postaci:

```
X        0 S err    1 S err,
```

przy czym zamiast X trzeba wstawić unikatowy numer.

## 4.3. Przykład transformacji

Poniższy program na maszynę stertową opisuje dodawanie dwóch liczb. Na początku obliczenia na stercie umieszczane są argumenty. Rachmistrz sprawdza, czy liczba na wierzchołku sterty jest zerem. Jeśli tak, to usuwa ją ze sterty i kończy działanie. W przeciwnym wypadku zwiększa wartość pierwszego argumentu i zmniejsza wartość drugiego, korzystając z kopiowania i kasowania liczb na stercie.

```
0     * JMZ 3
        CPY 1
        INC
        DEL 2
        CPY 1
        DEC
        DEL 2
        JMP 0
3       DEL 0
end     STP.
```

Wykonując opisaną wcześniej procedurę przekształcania programu maszyny stertowej na maszynę Turinga otrzymuje się program przedstawiony poniżej.

1. instrukcja `0 * JMZ 3`

```
0     * 0 L 1          H
1       0 S err    1 L 2
2       0 R 3      1 R 4
3         H        1 R 78
4         H        1 R 6,
```

2. instrukcja `CPY 1`

```
6       0 L 7          H
7       0 S err    1 L 8
8       0 L 9      1 L 8
9       0 S err    1 L 10
10      0 R 11     1 L 10
11      0 R 18     0 R 12
12      0 R 13     1 R 12
13      0 R 14     1 R 13
14      1 L 15     1 R 14
15      0 L 16     1 L 15
16      0 L 17     1 L 16
17      1 R 11     1 L 17
18      0 R 19     1 R 18
19      0 S 21     1 R 19,
```

3. instrukcja `INC`

```
21      0 L 22         H
22      0 S err    1 R 23
23      1 R 25         H,
```

4. instrukcja `DEL 2`

```
25      0 L 26         H
26      0 S err    0 L 27
27      1 L 28     1 L 27
28      0 S err    0 L 29
29      1 L 30     1 L 29
30      0 S err    0 R 31
31      0 R 32     1 R 31
32      0 L 33     1 R 32
33        H        0 L 34
34      1 L 35     1 L 34
35        H        0 L 36
36      1 L 37     1 L 36
37      0 R 38     0 R 31
38      0 R 39     1 R 38
39      0 S 41     1 R 39,
```

5. instrukcja CPY 1

```
41    0 L 42     H
42    0 S err    1 L 43
43    0 L 44     1 L 43
44    0 S err    1 L 45
45    0 R 46     1 L 45
46    0 R 53     0 R 47
47    0 R 48     1 R 47
48    0 R 49     1 R 48
49    1 L 50     1 R 49
50    0 L 51     1 L 50
51    0 L 52     1 L 51
52    1 R 46     1 L 52
53    0 R 54     1 R 53
54    0 S 56     1 R 54,
```

6. instrukcja DEC

```
56    0 L 57     H
57    0 S err    0 L 58
58    0 S err    1 R 60,
```

7. instrukcja DEL 2

```
60    0 L 61     H
61    0 S err    0 L 62,
62    1 L 63     1 L 62
63    0 S err    0 L 64
64    1 L 65     1 L 64
65    0 S err    0 R 66
66    0 R 67     1 R 66
67    0 L 68     1 R 67
68      H        0 L 69
69    1 L 70     1 L 69
70      H        0 L 71
71    1 L 72     1 L 71
72    0 R 73     0 R 66
73    0 R 74     1 R 73
74    0 S 76     1 R 74,
```

8. instrukcja JMP 0

```
76    0 S 0      1 S 0,
```

9. instrukcja 3 DEL 0

```
78    0 L 79     H
79    0 S err    0 L 80
80    0 S end    0 L 80,
```

10. instrukcja end STP

```
end     H       H,
```

11. obsługa błędów

```
err     H       H.
```

## 5. Podsumowanie

Zapisywanie algorytmów w postaci maszyn Turinga jest bardzo kłopotliwe ze względu na prostą konstrukcję tego modelu. Z drugiej strony prostota maszyny Turinga ułatwia rozważania teoretyczne na temat algorytmów.

Okazuje się, że poprzez zastosowanie bardziej skomplikowanych modeli obliczeniowych można w sposób bardziej przystępny dla człowieka generować programy na maszynę Turinga. W tym celu należy najpierw zdefiniować wygodny model dla badanego problemu, a potem określić sposób przekształcania tego modelu na maszynę Turinga. Wówczas skonstruowanie maszyny Turinga, która będzie rozwiązywać dany problem, sprowadza się do zapisania programu dla stworzonego modelu oraz do wykonania transformacji tego modelu na maszynę Turinga.

Zaletą omawianego rozwiązania jest możliwość przyspieszenia generowania programów na model Turinga, generowanie programów o regularnej strukturze i duża szybkość wprowadzania zmian. Mimo że programy wygenerowane automatycznie nie są tak optymalne jak stworzone bezpośrednio przez człowieka na maszynę Turinga, to wciąż nadają się do badań, gdyż są równoważne wielomianowo [6]. W rozważanym przykładzie każda z instrukcji maszyny stertowej jest zamieniana na program maszyny Turinga, którego złożoność jest liniowa względem argumentu.

## Literatura

[1] Brady J. M.: *Informatyka teoretyczna w ujęciu programistycznym.* Warszawa, WNT 1983

[2] Davis M. D., Weyuker E. J.: *Computability, complexity and languages – fundamentals of theoretical computer science.* Orlando, Academic Press, Inc. 1983

[3] Engeler E.: *Introduction to the Theory of Computation.* New Jork, Academic Press, Inc. 1973

[4] Gårding L.: *Spotkanie z matematyką.* Warszawa, Wydawnictwo Naukowe PWN 1993

[5] Harel D.: *Rzecz o istocie informatyki – algorytmika.* Warszawa, WNT 1992

[6] Kościelski A.: *Teoria obliczeń – wykłady z matematycznych podstaw informatyki.* Wrocław, Wydawnictwo Uniwersytetu Wrocławskiego 1997

[7] Kowal S.: *500 zagadek matematycznych.* Warszawa, Wiedza Powszechna 1975

[8] Penrose R.: *Nowy umysł cesarza – o komputerach, umyśle i prawach fizyki.* Warszawa, Wydawnictwo Naukowe PWN 1996

[9] Trachtenbrot B. A.: *Algorytmy i automatyczne rozwiązywanie zadań.* Warszawa, Państwowe Wydawnictwo Naukowe 1961

[10] Wirth N.: *Algorytmy + struktury danych = programy.* Warszawa, WNT 1999