Konwersja liczb naturalnych z systemu decymalnego na szesnastkowy za pomocą Maszyny Turinga

github@trash-hold

23 czerwca 2022

Spis treści

1 Wstęp

Sprawozdanie opisuje działanie Maszyny Turinga składającej się z **translatora** i **pliku definiującego**. Plik definiujący **`DecToHex.txt`** zawiera definicje stanów, alfabetu i taśmy właściwe dla zrealizowania algorytmu konwersji dec do hex. Translator **`turing.py`** został napisany w Pythonie (3.9.7), jego funkcją jest obsługa powyższego pliku tekstowego i interpretacja definicji stanów symulująca działanie Maszyny Turinga. Jednostka translatora jestw stanie zrealizować dowolny algorytm, który ma odpowiednią składnię definicji.

1.1 Maszyna Turinga

Jest abstrakcyjną maszyną składającą się z ruchomej głowicy i taśmy o nieskończonej ilości komórek wypełnionych pojedynczymi znakami (dopuszcza się też maszyny operujące na większej ilości par). Uproszczony model działania:

1. Odczytanie znaku z taśmy przez głowicę
2. Zapisanie nowego znaku w tej samej komórce
3. Przejście maszyny w następny stan
4. Przesunięcie głowicy o jedną komórkę w lewo/prawo

Formalną definicję można zapisać jako:

$$M = \Big\langle Q,\ q_0,\ F,\ \Gamma,\ \Sigma,\ B \Big\rangle \tag{1.1}$$

Q - skończony zbiór stanów
q_0 - stan początkowy
F - zbiór stanów końcowych
Γ - alfabet, czyli zbiór wszystkich akceptowanych znaków
$\Sigma \in \Gamma - \{B\}$ - zbiór dopuszczalnych znaków inputu
B - znak pusty

1.1.1 Definicje

Stan opisuje etap podczas pracy maszyny, zbiór wszystkich definicji zawierających dany stan jest równoważny do przebiegu funkcji/metody. Na przykład stan o nazwiee `CHANGE` mogłby być wykorzystany do inwersji binarnej.

Stan wejściowy ten, na którym obecnie operuje maszyna.

Stan wyjściowy ten, do którego przejdzie maszyna po odczytaniu danego znaku

Stan początkowy jest stanem wejściowym po uruchumieniu maszyny, w dalszej części oznaczany jako `QINIT`

Stan końcowy po odczytaniu go jako stan wejściowy maszyna przestaje pracować, zdefiniowany jako `FIN`

Definicja stanu to paczka danych definiująca zachowanie maszyny, zapis przyjęty w pracy to:

$$< \text{Stan wejściowy} >,\quad < \text{Input} >,\quad < \text{Stan wyjściowy} >,\quad < \text{Output} >,\quad < \text{Kierunek ruchu} > \tag{1.2}$$

1.1.2 Założenia

Zasady, które muszą być spełnione by maszyna działała:

- Taśma
  - Taśma musi zawierać znaki należące do alfabetu
- Stany
  - Nie mogą istnieć dwie definicje stanu wejściowego o tej samej nazwie i tym samym inpucie
  - Każdy stan wyjściowy musi być zdefiniowany jako stan wejściowy (oprócz `FIN`)
  - Musi istnieć chociaż jedna definicja stanu, w której stan wyjściowy to `FIN`

2 Translator

2.1 Podstawowe wytyczne i działanie

Translator ma trzy główne funkcjonalności:

- Kontrolna - weryfikacja składni pliku z inputem
- Sprawcza - symulacja działania Maszyny Turinga i zwrócenie końcowego stanu taśmy
- Informacyjna - zawiera plik **`readMe.txt`** zawierający kluczowe informacji o działaniu programu

Oprócz założeń wynikających bezpośrednio z definicji Maszyny, istnieją te dotyczyące samej jednostki translacyjnej a dokładniej jej składni:

- Alfabet - poniższe znaki nie mogą być zadeklarowane jako część alfabetu:
  - # - zarezerwowany jako pusty znak
  - / - zarezerwowany jako znak początka komentarza
- Taśma - nie może zawierać znaku spacji
- Stany:
  - Kolejne dane definicji stanu musza być rozdzielone sekwencją ", " (przecinek, spacja)
  - Kierunek jest określany jako: $<$ (lewo), $>$ (prawo)
  - Dopuszczalne są linijki zawierające wyłącznie znak nowej linii `\n`
- Input - musi zostać podany w kolejności:
  1. Alfabet
  2. Taśma
  3. Definicja stanów + komentarze

2.1.1 Budowa programu

Program operuje na klasach: `TuringMachine i State`. `State` pełni funkcje "kontenera" zawierającego paczkę 5 danych formułujących definicję stanu:

```
class State:
    def __init__(self, name, input, state_to, output, dir):
        self.__name__ = name
        self.__in__ = input
        self.__nstate__ = state_to
        self.__out__ = output
        self.__d__  = dir
```

Cała reszta funkcjonalności programu znajduje się wewnątrz klasy **TuringMachine**. Obiekt po utworzeniu wykonuje operacje zdefiniowane w pliku tekstowym, którego ścieżka/nazwa została podana jako argument `file`.

```
class TuringMachine:
    def __init__(self, file):
        f = open(file, "r")
        self.__alphabet__ = self.init_alphabet(f)
        self.__tape__ = self.init_tape(f)
        self.__states__ = self.init_states(f)
        f.close()
        self.__tape__ = self.execute()
```

Metody klasy **TuringMachine** wraz z ich skrótowym działaniem:

Inicjalizacja	
`init_alphabet(self, f)`	Zczytuje podany w inpucie alfabet, tworzy z niego zbiór i weryfikuje czy użytkownik nie podał zarezerwowanych znaków; Zwraca zbiór
`init_tape(self, f)`	Zczytuje podaną taśmę i sprawdza czy podane znaki są w alfabecie; Zwraca listę znaków
`init_states(self, f)`	Zczytuje definicje stanów i zapisuje je do napotkania EOF; weryfikuje składnie według wytycznych; Zwraca listę obiektów
Symulacja	
`execute(self)`	Metoda symulująca Maszynę, odczytuje dane z tablicy powstałej na podstawie taśmy, odszukuje stany wyjścia, zapisuje output i porusza "głowicą"
`show_output(self, clear_sym = None)`	Wyświetla stan taśmy w terminalu, możliwe jest pominięcie znaków zadeklarowanych pod `clear_sym`
Pomocnicze	
`readMe(self, file = "readMe.txt")`	Wypisuje zawartość readMe w terminalu
`create_dic(self)`	Tworzy słownik rodzin stanów; pozwala na szybsze znajdowanie definicji stanów wyjściowych

Przykładowa definicja pliku txt wraz z outputem:

```
//binaryNegation.txt

    1 0
    1000100101
    qinit, 1, qinit, 1, >
    qinit, 0, qinit, 0, >
    qinit, #, pass, #, <
    pass, 1, pass, 1, <
    pass, 0, pass, 0, <
    pass, #, neg, #, >
    neg, 1, neg, 0, >
    neg, 0, neg, 1, >
    neg, #, fin, #, >


Output: 0111011010
```

3 Definicja algorytmu

3.1 Idea

Celem algorytmu jest przekształcenie naturalnej liczby dziesiętnej na liczbę w systemie szesnastkowym. Alfabet, z którego korzysta Maszyna:

```
//alfabet
0 1 2 3 4 5 6 7 8 9 A B C D E F !
```

W celu skrócenia zapisu przejść między stanami w dalszej części sprawozdania wykorzystano oznaczenia zbiorów, do których należy rozważany znak:

- *bin* 0 lub 1
- *dec_num* cyfra dziesiętna
- *even* parzysta cyfra lub 0
- *odd* nieprzysta cyfra

3.1.1 Oznaczenia w schematach

Sześciokąt funkcja zbiorcza o znaczeniu ideowym

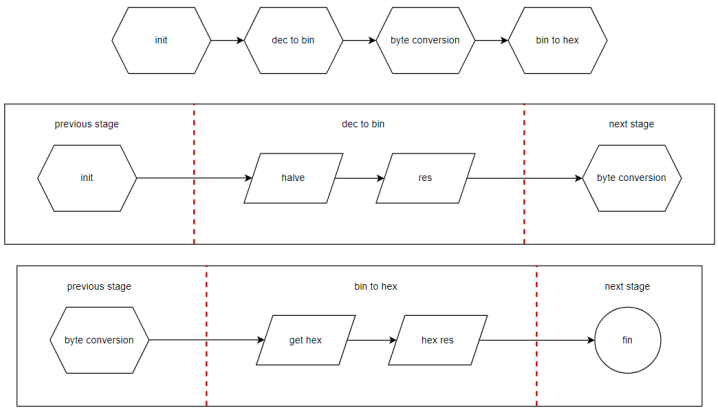
Równoległobok podfunkcja o niższym poziomie abstrakcji

Kwadrat rodzina definicji stanów pełniących tą samą funkcję

Koło oznaczenie stanu; Niebieski kolor stanu oznacza stan "początkowy" danego segmentu

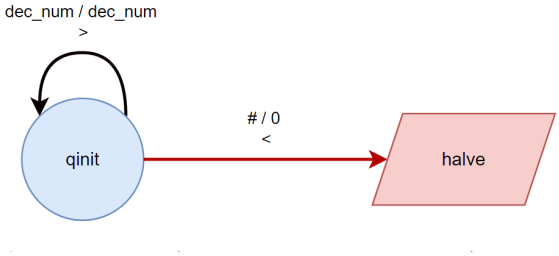
Kolor niebieski - stan początkowy segmentu; czerwony - przejście do innego segmentu; zielony - przejście do stanu końcowego

3.1.2 Schemat ideowy



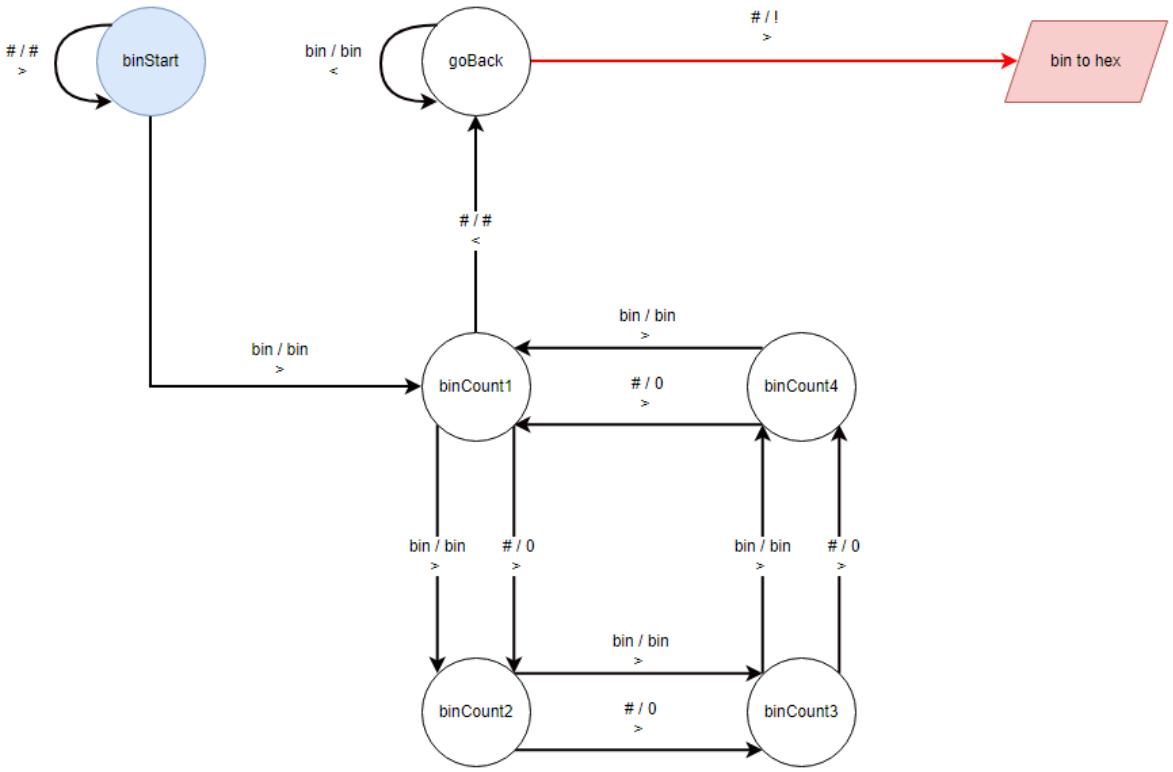
3.2 Skrótowy opis działania

init weryfikuje czy taśma składa się wyłącznie z cyfr.



dec to bin konwersja z systemu dziesiętnego do binarnego. Składa się z dwóch podfunkcji: *halve* dzielącej liczbę na pół i przenoszącej resztę z dzielenia; *res* przenoszącej wynik (0 lub 1) w odpowiednie miejsce taśmy.

byte conversion uzupełnia liczbę binarną zerami tak by składała się z pełnych bajtów.

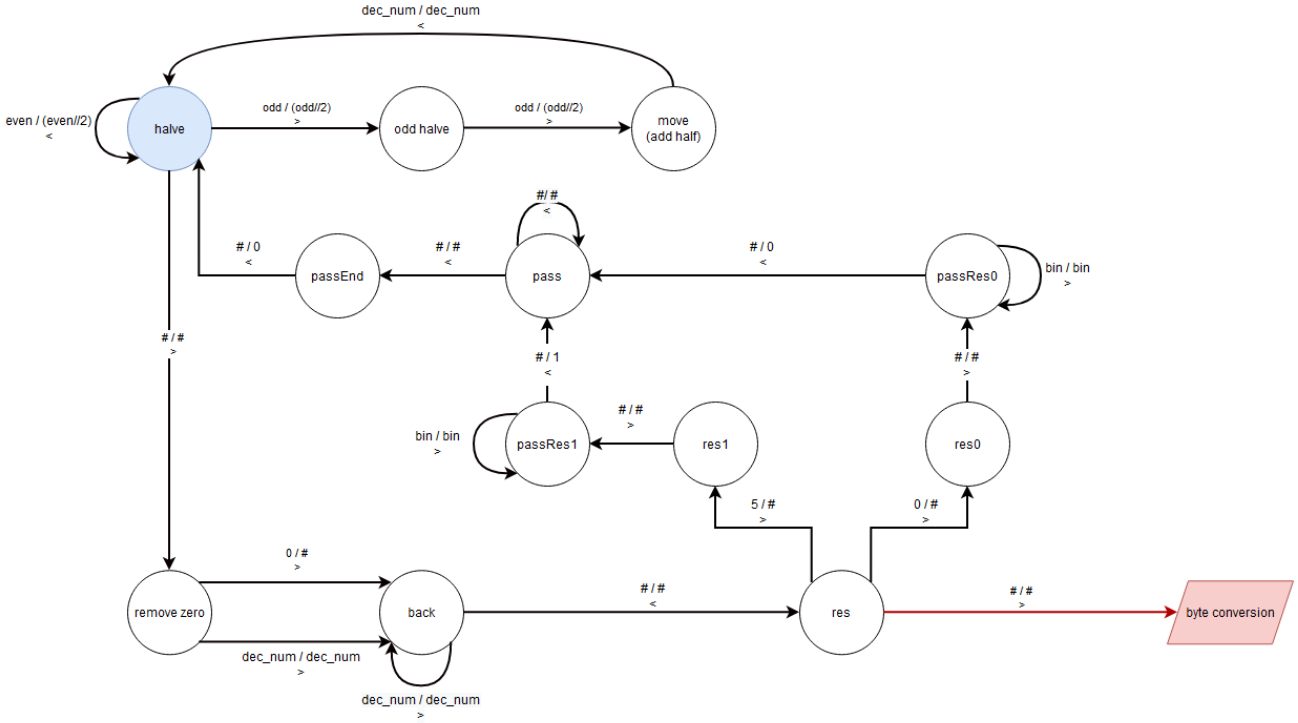


bin to hex przeprowadza konwersje z binarnego zapisu do szesnastkowego. Podfunkcje działają analogicznie do *dec to bin*

3.3 Funkcja *dec to bin*

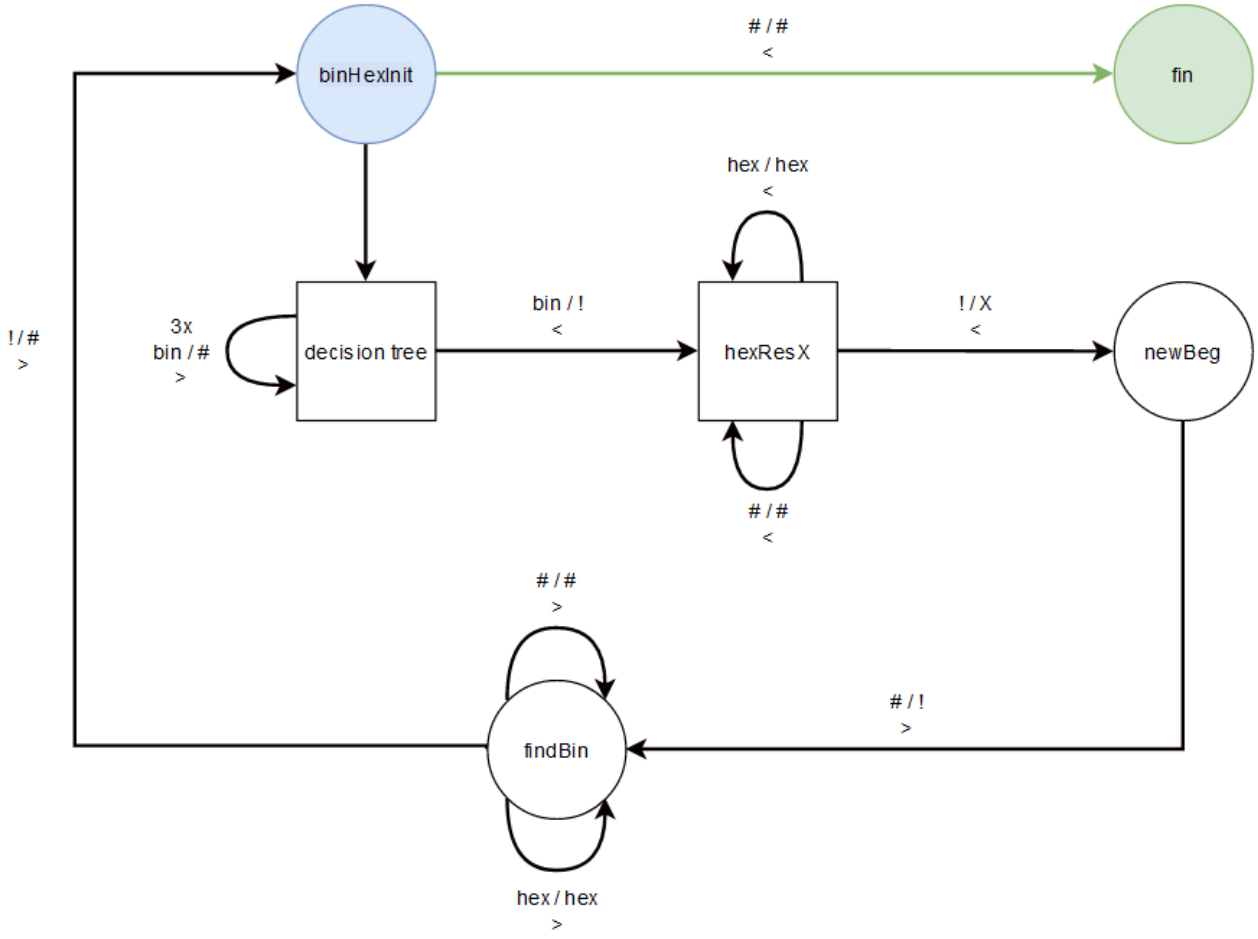
Algorytm konwersji został stworzony przez Maxa von Buelowa oraz kd3x i zaimplementowano go zgodnie z licencją CC BY-NC-SA 3.0. Przybliżona praca algorytmu:

1. Dzielenie liczby.
2. Zaprzestanie dzielenie po odczytaniu hasha
3. Przesuwanie się w prawo aż do znalezenia hasha
4. Wejście w jedną z dwóch rodzin stanów przenoszącą wynik (na podstawie odczytanej reszty z dzielenia) lub przejście do kolejnej funkcji
5. Przesuwanie się do znalezienia hasha
6. Zapisanie wyniku (0 lub 1)
7. Szukanie hasha przesuwając się w lewo
8. Zapisanie 0
9. Powrót do puntku 1



3.4 Funkcja *bin to hex*

Konwersja bin to hex jest oparta na drzewie decyzyjnym, które pełni rolę "pamięci". Każdy bit odczytywanego bajta może mieć wartość 1 lub 0, w związku z czym tworząc przejścia stanów można kontrlować sekwencje odczytanych znaków.



Kwadrat oznacza zbiór rodzin z licznymi stanami, ale które mają tę samą zasadę działania. *hexResX* ozanacza przenoszenie wyniku podobne do *dec to bin* z zastrzeżeniem, że X to znak określający wartość otrzymaną w drzewie decyzyjnym.

Symbol ! służy do zaznaczania "końca" hexów oraz początku ciągu bajtów binarnych. Został użyty by uniknąć przechodzenia Maszyny w nieskończoną pętlę

4 Podsumowanie

Translator działa zgodnie z założeniami i jest w stanie zrealizować dowolny program (zapisany poprawną składnia). Możliwe udoskonalenia do wprowadzenia: zabezpieczenia w przypadku nieistnienia plików, których ścieżkę podaje użytkownik; zabezpieczenie zmiennej `clear_sym`; dodanie testów. Pamięć może być potencjalnym problemem, niektóre algorytmy mogą generować dużo pustych znaków na taśmie, by uniknąć problemu można zaimplementować coś w rodzaju pamięci cache

Algorytm działa poprawnie również dla dużych liczb. Plik zawiera ponad 500 definicji stanów, szczególnie nieoptmalną metodą (pod względem objętościwoym) wydaje się konwersja bin to hex.