Home Page
Title Page
Page 1 of 38
Go Back
Full Screen
Close
Quit

Wykład 2: Wprowadzenie do teorii obliczeń

Nguyen Hung Son
son@mimuw.edu.pl

Streszczenie

Na tym wykładzie omówione będą podstawowe pojęcia teorii obliczeń takie, jak gramatyka, język, problem decyzyjny, oraz takie podstawowe modele obliczeń, jak automat skończony, maszyna Turinga, itp.

1. Teoria języków formalnych

- *Alfabet* = dowolny zbiór symboli użytych w danym języku. Np.
  - $\mathbf{B} = \{0, 1\}$ – alfabet dla systemu dwójkowego;
  - $\mathbf{C} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ – alfabet systemu dziesiętnego;
  - $\mathbf{A} = \{a, b, ..., z\}$ – alfabet łaciński (wspólny dla wielu języków);
- *Słowo* nad alfabetem Σ = dowolny ciąg znaków ze zbioru Σ.
- Σ^n = zbiór słów nad alfabetem Σ o długości n
- $\Sigma^0 = \{\epsilon\}$;
- Σ^* = zbiór wszystkich skończonych słów nad alfabetem Σ

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

Pytanie: Jaką moc ma zbiór Σ^*?

- *Językiem* nad Σ nazywamy dowolny zbiór słów $\mathbf{L} \subset \Sigma^*$.
- Każdy język może być definiowane przez podania zbioru słów, lub przez podania reguł tworzenia słów do niego należących. Taki zbiór reguł nazywa się *gramatyką*.
- Problem rozpoznawania języka: "*Dany jest język* $\mathbf{L}$ *i słowo* $w \in \Sigma^*$, *sprawdź czy* $w \in \mathbf{L}$?"
- Problem decyzyjny: *"czy dany język jest rozpoznawalny?"*

  Dane: język $\mathbf{L}$ nad alfabetem Σ,

  Szukane: algorytm obliczający funkcję $f_{\mathbf{L}} : \Sigma^* \to \{0, 1\}$ taką, że $f_{\mathbf{L}}(w) = 1 \Leftrightarrow w \in \mathbf{L}$.

  Innym słowem, znaleźć algorytm sprawdzający, czy dowolnie wybrane słowo w należy do $\mathbf{L}$".
- UWAGA: nie każdy problem decyzyjny jest tak samo trudny. Np. sprawdź czy liczba (pisana w systemie dziesiętnym) jest
  - podzielna przez 2?
  - podzielna przez 13?
  - liczbą pierwszą?

1.1. Tematy badań nad problemami decyzyjnymi

- Teoria rozwiązywalności problemów decyzyjnych
  - problem decyzyjny można traktować jako problem obiczenia wartości jej funkcji charkterystycznych.
  - klasa funkcji rekurencyjnych $\supseteq$ klasa rozwiązalnych problemów decyzyjnych.
  - istnieją problemy nierozwiązalne zwane "problemami nieroztrzygalnymi"
- Modele standardowe:
  - Automaty skończone
  - Maszyny Turinga
  - RAM (Random Access Machines)
  - ... komputery równoległe, ...

- Teoria złożoności obliczeniowej w modelach standardowych:
  - Klasa problemów łatwych (o złożoności wielomianowej)
  - Klasa problemów niebeznadziejnych (o złożoności co najwyżej wykładniczej)
  - Klasa problemów beznadziejnych (o złożoności co najmniej wykładniczej)
  - Algorytmy aproksymacyjne.
- Modele niestandardowe (brak realizacji)
  - Algorytmy symulowanego wyrzażania (maszyny Bolzmana)
  - Algorytmy genetyczne i programowanie ewolucyjne
  - Sieci neuronowe (perceptron, Hopfield, Kohonen, ...)
  - DNA computing, komputery molekularne

2. Plan wykładu monograficznego:

1. Wprowadzenie do teorii obliczeń, automat skończony;
2. Maszyny Turinga, model RAM;
3. Złożoności obliczeń;
4. Algorytmy symulowanego wyrzażania;
5. Maszyny Bolzmana;
6. Algorytmy genetyczne;
7. Rozwiązywania problemów za pomocą AG
8. Programowanie ewolucyjne;
9. Modele sieci neuronowych: perceptron;
10. Sieć neuronowa ze wsteczną propagacją błędu;
11. Sieć Hopfielda;
12. Sieć Kohonena;
13. Komputery molekularne;
14. Rozwiązywania problemów za pomocą DNA
15. Podsumowanie

Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 7 of 38
Go Back
Full Screen
Close
Quit

3. Automat skończony (FSA)

To jest najbardziej pierwotny model obliczeń. Automatem skończonym nazywamy czwórkę $\mathbb{A} = (\Sigma, Q, q_0, T)$, gdzie

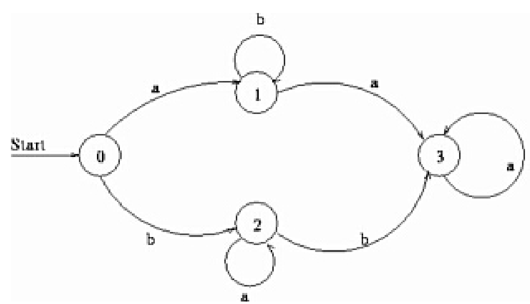
- Σ jest skończonym zbiorem liter zwanym *alfabetem*;
- Q jest skończonym zbiorem stanów;
- $q_0 \in Q$ jest stanem początkowym;
- $T \subset (\Sigma \times Q) \times Q$ jest zwane *relacją przejścia*

Jeżeli relacja przejścia T okaże się funkcją, to automat nazywamy "*deterministycznym*", w przeciwnym przypadku nazywamy "*niedeterministycznym*"

Przykład *Niech* $\Sigma = \{a, b\}$, $Q = \{0, 1, 2, 3\}$, $q_0 = 0$ *i*

$T =$

	a	b
0	1	2
1	3	1
2	2	3
3	3	3



Działanie automatu:

- Zakładamy, że
- Automat skończony działa następująco:
  - automat skończony otrzyma pewien strumień danych wejściowych (np. za pomocą czytnika do odbierania kolejnego znaku z alfabetu).
  - ustawia się w stanie początkowym q_0
  - odczytuje pojedynczy znak, następnie zmienia stan zgodnie z relacją przejścia.

Na przykład, na wejściu jest słowo: abbaaba

$$Q = \{0, 1, 2, 3\}$$

$$T = \begin{array}{c|cc} & a & b \\ \hline 0 & 1 & 2 \\ 1 & 3 & 1 \\ 2 & 2 & 3 \\ 3 & 3 & 3 \end{array}$$

Read	Go to
–	0
a	1
b	1
b	1
a	3
a	3
b	3
a	3

Wyniki obliczeń

- Niektóre stany możemy zaznaczyć jako "akceptujące"!
- Po zakończeniu obliczeń (koniec słowa wejściowego) sprawdźmy, w którym stanie znajduje się automat.
  - Jeśli ten stan jest akceptujący, to mówimy, że "automat akceptuje słowo"
  - w przeciwnym przypadku mówimy, że "automat nie akceptuje słowa" (odrzuca)
- Automat dzieli zbiór Σ^* na akceptowane i nieakceptowane.
- Zbiór słów akceptowanych przez jakiś automat nazywamy językiem przez niego akceptowanym.

Pytanie: Które języki mogą akceptować automaty skończone?
Odp.: Języki regularne !
Pytanie: Czy $L = \{a^i b^i\}$ jest akceptowany przez automaty skończone?
Odp.: Nie.

Home Page
Title Page
Page 10 of 38
Go Back
Full Screen
Close
Quit

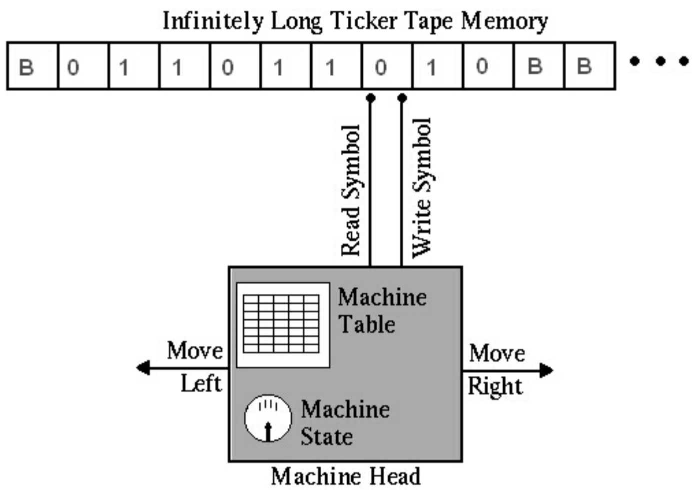
4. Maszyny Turinga

Model Maszyny Turinga składa się z

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

gdzie

- Q = skończony zbiór stanów zawierający q_0;
- Σ = skończony alfabet symboli wejściowych $B \notin \Sigma$;
- Γ = skończony alfabet symboli taśmowych zawierających Σ i B;
- $delta$ = funkcja przejścia z $Q \times \Gamma$ do $Q \times \{L, R\} \times \Gamma$;
- q_0 = stan początkowy;
- B = symbol "blank" (początkowo znajduje się wszędzie na taśmie, poza miejscami zajmowanymi przez słowo wejściowe);
- F = zbiór stanów akceptujących.

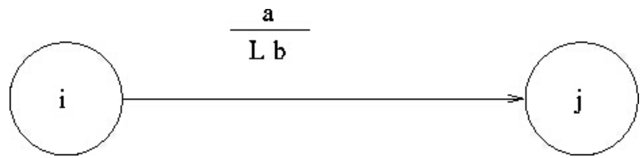


Symbol Read From Tape

Current Machine State	B	0	1
1	Write 1 to tape. Adopt State 6	Write B to tape. Adopt State 2	Move 1 cell left. Adopt State 1
2	Move 1 cell left. Adopt State 2	Write B to tape. Adopt State 3	Eh?
3	Move 1 cell left. Adopt State 3	Write B to tape. Adopt State 4	Write B to tape. Adopt State 5
4	Move 1 cell right. Adopt State 4	Eh?	Move 1 cell left. Adopt state 6
5	Move 1 cell right. Adopt State 5	Eh?	Move 1 cell left. Adopt State 1
6	Write 0 to tape. Adopt State 6	Stop	Move 1 cell left. Adopt State 3

4.1. Opis maszyny Turinga

Następujący rysunek:



oznacza, że będąc w stanie i i widząc bieżący symbol a, maszyna zamienia a na b, przesuwa głowicę w lewo i przechodzi do stanu j. Możemy ten akt pisać w postaci instrukcji przejścia:

$$\langle i, a, b, L, j \rangle$$

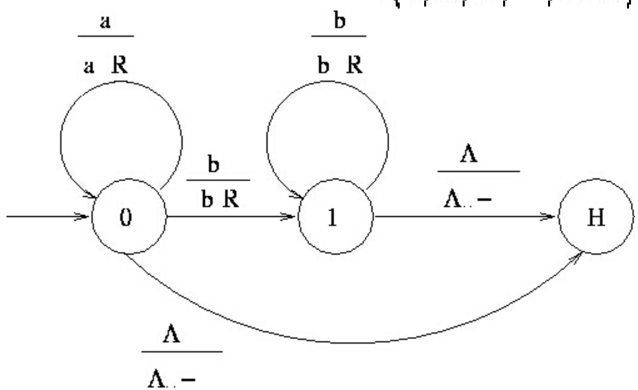
Maszyny Turinga mogą być zadane w postaci grafu lub w postaci listy takich instrukcji.

4.2. Rozpoznawanie języka przez TM

- wstawiamy słowo na taśmie, "zainicjujemy" maszynę ;
- twierdzimy, że słowo jest akceptowane, jeśli maszyna osiąga stan akceptujący.
- słowo nie jest odrzucany, jeśli zapętla się maszyna lub osiąga taką sytuację, dla której nie może już realizować żadnej instrukcji.

4.3. Przykład

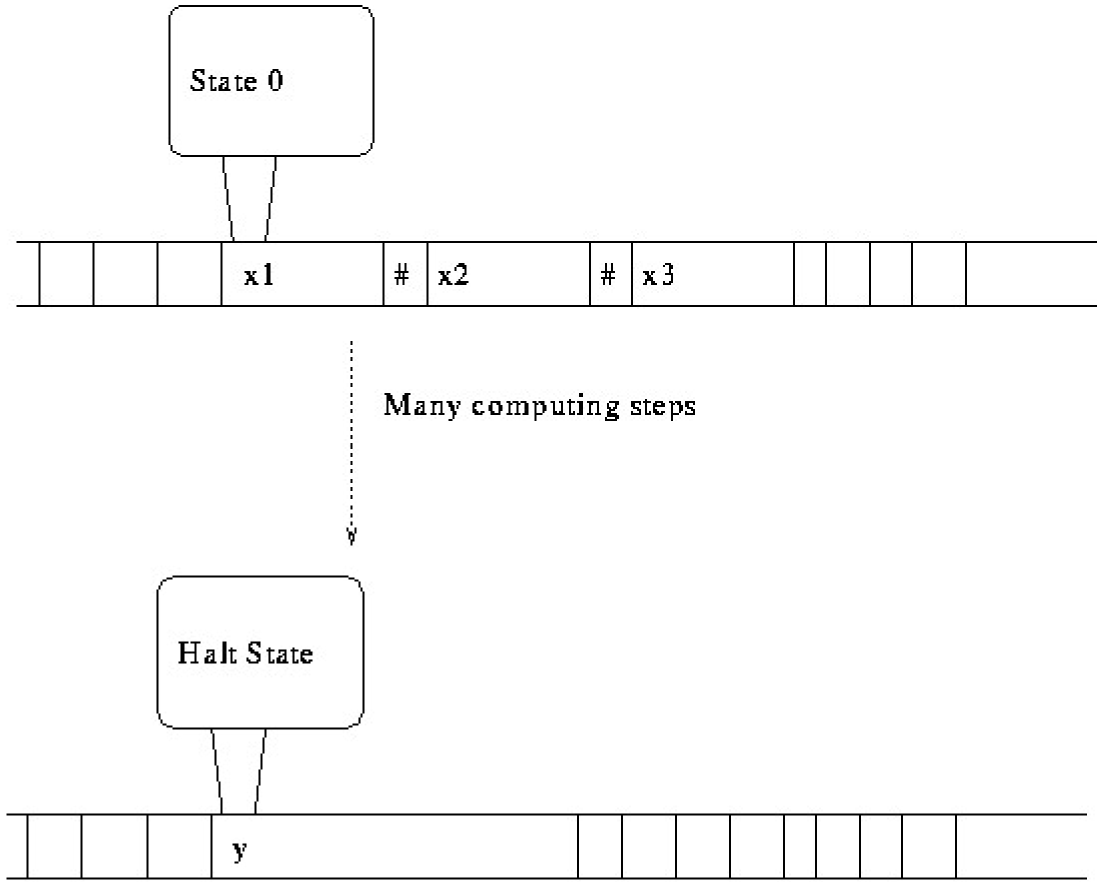
$$\langle 0, \Lambda, \Lambda, -, \mathrm{Halt}\rangle$$
$$\langle 0, a, a, R, 0\rangle$$
$$\langle 0, b, b, R, 1\rangle$$
$$\langle 1, b, b, R, 1\rangle$$
$$\langle 1, \Lambda, \Lambda, -, \mathrm{Halt}\rangle$$



$$\langle 0, a, X, R, 1\rangle$$
$$\langle 0, Y, Y, R, 0\rangle$$
$$\langle 0, Z, Z, R, 4\rangle$$
$$\langle 0, \Lambda, \Lambda, -, \mathrm{Halt}\rangle$$
$$\langle 1, a, a, R, 1\rangle$$
$$\langle 1, b, Y, R, 2\rangle$$
$$\langle 1, Y, Y, R, 1\rangle$$
$$\langle 2, c, Z, L, 3\rangle$$
$$\langle 2, b, b, R, 2\rangle$$
$$\langle 2, Z, Z, R, 2\rangle$$
$$\langle 3, a, a, L, 3\rangle$$
$$\langle 3, b, b, L, 3\rangle$$
$$\langle 3, X, X, R, 0\rangle$$
$$\langle 3, Y, Y, L, 3\rangle$$
$$\langle 3, Z, Z, L, 3\rangle$$
$$\langle 4, Z, Z, R, 4\rangle$$
$$\langle 4, \Lambda, \Lambda, -, \mathrm{Halt}\rangle$$

5. Obliczenie funkcji maszyną Turinga

- Można traktować maszynę Turinga jako maszyna obliczająca funkcję określoną na słowach wejściowych.
- Zazwyczaj zakładamy, że słowo wejściowe znajduje się na taśmie po prawej stronie głowicy aż do pierwszego symbolu blanku, a słowo wyjściowe będzie umieszczone w tym samym miejscu.
- Często będziemy traktowali słowa wejściowe i wyjściowe jako liczby, używając symboli 1, B, ewentualnie 0 i znak separatora ".".
- Liczba n może być przedstawiona jako ciąg unarny, binarny, dziesiętny lub inne.



A TM computes that $f(x1,x2,x3)=y$

6. Maszyny Turinga

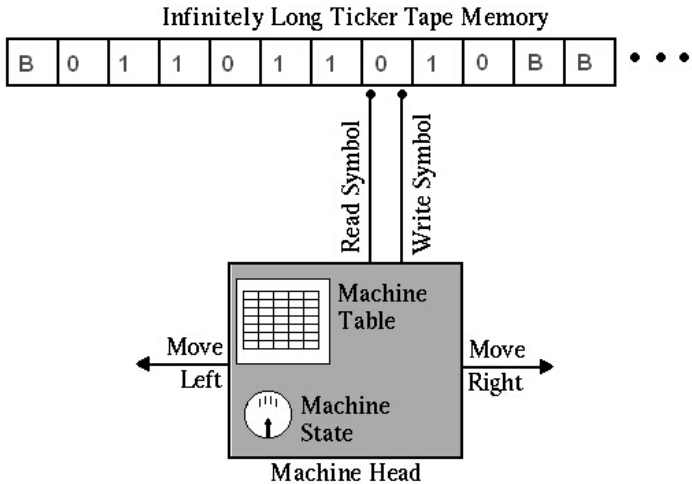
Model Maszyny Turinga składa się z

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

gdzie

- Q = skończony zbiór stanów zawierający q_0;
- Σ = skończony alfabet symboli wejściowych $B \notin \Sigma$;
- Γ = skończony alfabet symboli taśmowych zawierających Σ i B;
- $delta$ = funkcja przejścia z $Q \times \Gamma$ do $Q \times \{L, R\} \times \Gamma$;
- q_0 = stan początkowy;
- B = symbol “blank” (początkowo znajduje się wszędzie na taśmie, poza miejscami zajmowanymi przez słowo wejściowe);
- F = zbiór stanów akceptujących.

Teoria języków formalnych
Plan wykładu . . .
Automat skończony (FSA)
Maszyny Turinga
Obliczenie funkcji . . .
Maszyny Turinga
Obliczenie funkcji . . .
Uniwersalna maszyna . . .
Model RAM
Równoważność modeli
Church's Thesis
Home Page
Title Page
Page 17 of 38
Go Back
Full Screen
Close
Quit



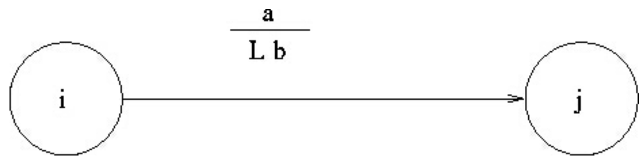
Symbol Read From Tape

Current Machine State	B	0	1
1	Write 1 to tape. Adopt State 6	Write B to tape. Adopt State 2	Move 1 cell left. Adopt State 1
2	Move 1 cell left. Adopt State 2	Write B to tape. Adopt State 3	Eh?
3	Move 1 cell left. Adopt State 3	Write B to tape. Adopt State 4	Write B to tape. Adopt State 5
4	Move 1 cell right. Adopt State 4	Eh?	Move 1 cell left. Adopt state 6
5	Move 1 cell right. Adopt State 5	Eh?	Move 1 cell left. Adopt State 1
6	Write 0 to tape. Adopt State 6	Stop	Move 1 cell left. Adopt State 3

Home Page
Title Page
Page 18 of 38
Go Back
Full Screen
Close
Quit

6.1. Opis maszyny Turinga

Następujący rysunek:



oznacza, że będąc w stanie i i widząc bieżący symbol a, maszyna zamienia a na b, przesuwa głowicę w lewo i przechodzi do stanu j. Możemy ten akt pisać w postaci instrukcji przejścia:

$$\langle i, a, b, L, j \rangle$$

Maszyny Turinga mogą być zadane w postaci grafu lub w postaci listy takich instrukcji.

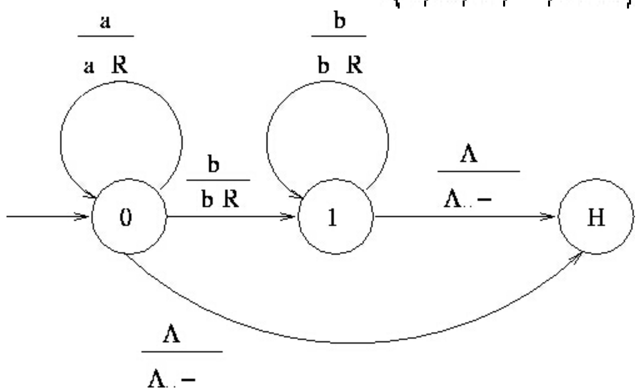
6.2. Rozpoznawanie języka przez TM

- wstawiamy słowo na taśmie, "zainicjujemy" maszynę ;
- twierdzimy, że słowo jest akceptowane, jeśli maszyna osiąga stan akceptujący.
- słowo nie jest odrzucany, jeśli zapętla się maszyna lub osiąga taką sytuację, dla której nie może już realizować żadnej instrukcji.

Home Page
Title Page
Page 19 of 38
Go Back
Full Screen
Close
Quit

6.3. Przykład

$$\langle 0, \Lambda, \Lambda, -, \mathrm{Halt}\rangle$$
$$\langle 0, a, a, R, 0\rangle$$
$$\langle 0, b, b, R, 1\rangle$$
$$\langle 1, b, b, R, 1\rangle$$
$$\langle 1, \Lambda, \Lambda, -, \mathrm{Halt}\rangle$$

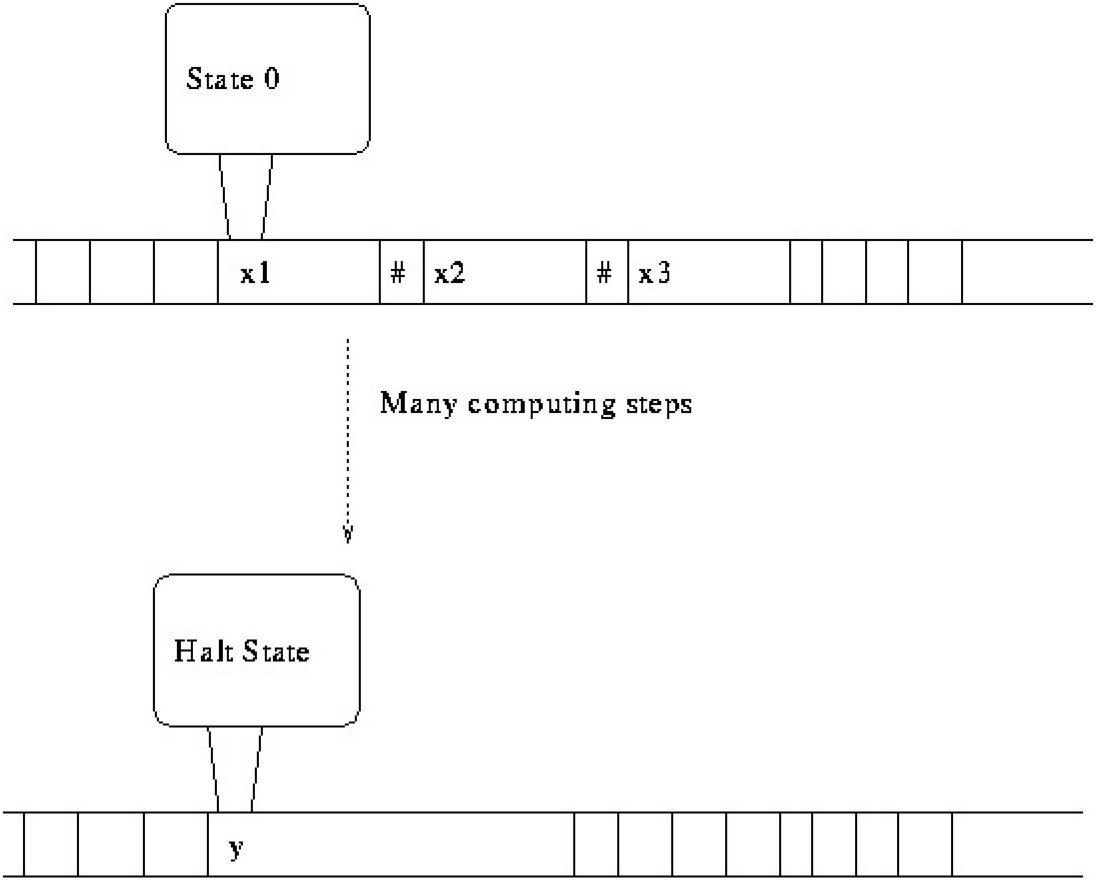


$$\langle 0, a, X, R, 1\rangle$$
$$\langle 0, Y, Y, R, 0\rangle$$
$$\langle 0, Z, Z, R, 4\rangle$$
$$\langle 0, \Lambda, \Lambda, -, \mathrm{Halt}\rangle$$
$$\langle 1, a, a, R, 1\rangle$$
$$\langle 1, b, Y, R, 2\rangle$$
$$\langle 1, Y, Y, R, 1\rangle$$
$$\langle 2, c, Z, L, 3\rangle$$
$$\langle 2, b, b, R, 2\rangle$$
$$\langle 2, Z, Z, R, 2\rangle$$
$$\langle 3, a, a, L, 3\rangle$$
$$\langle 3, b, b, L, 3\rangle$$
$$\langle 3, X, X, R, 0\rangle$$
$$\langle 3, Y, Y, L, 3\rangle$$
$$\langle 3, Z, Z, L, 3\rangle$$
$$\langle 4, Z, Z, R, 4\rangle$$
$$\langle 4, \Lambda, \Lambda, -, \mathrm{Halt}\rangle$$

7. Obliczenie funkcji maszyną Turinga

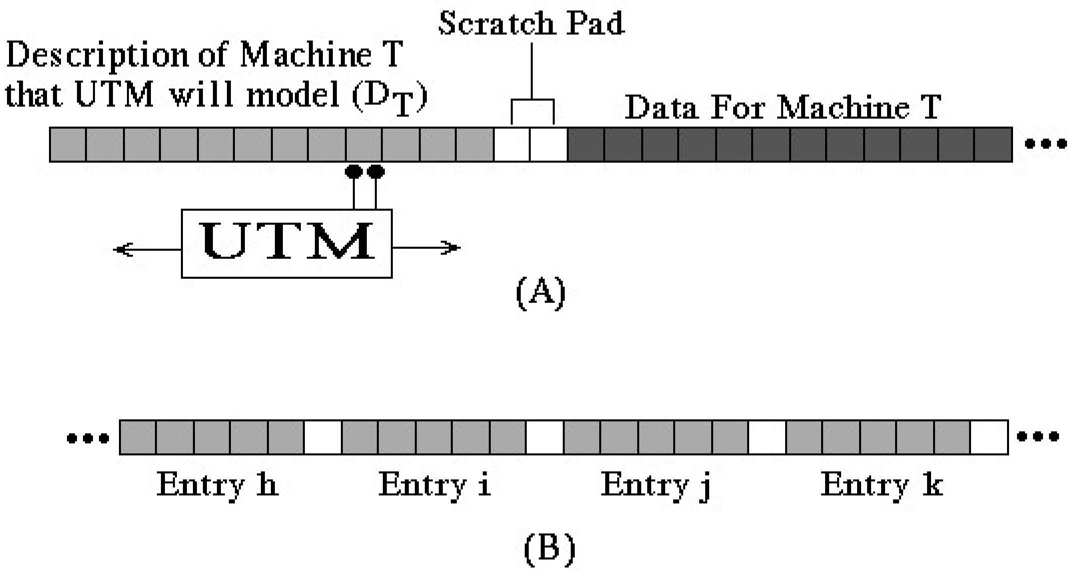
- Można traktować maszynę Turinga jako maszyna obliczająca funkcję określoną na słowach wejściowych.
- Zazwyczaj zakładamy, że słowo wejściowe znajduje się na taśmie po prawej stronie głowicy aż do pierwszego symbolu blanku, a słowo wyjściowe będzie umieszczone w tym samym miejscu.
- Często będziemy traktowali słowa wejściowe i wyjściowe jako liczby, używając symboli 1, B, ewentualnie 0 i znak separatora ".".
- Liczba n może być przedstawiona jako ciąg unarny, binarny, dziesiętny lub inne.

Teoria języków formalnych
Plan wykładu . . .
Automat skończony (FSA)
Maszyny Turinga
Obliczenie funkcji . . .
Maszyny Turinga
Obliczenie funkcji . . .
Uniwersalna maszyna . . .
Model RAM
Równoważność modeli
Church's Thesis
Home Page
Title Page
Page 21 of 38
Go Back
Full Screen
Close
Quit



A TM computes that $f(x1,x2,x3)=y$

8. Uniwersalna maszyna Turinga



9. Model RAM

RAM zawiera:

- Taśma wejściowa, z której maszyna wczytuje liczby będące danymi wejściowymi.
- Taśma wyjściowa, na której maszyna wypisuje liczby wyjściowe.
- nieskończenie wiele rejestrów: $R_0, R_1, \ldots$, które mogą zapamietać dowolnie duże liczby całkowite.
- oznaczmy przez r_i wartość trzymaną w rejestrze R_i.

9.1. Program w modelu RAM

- Programy w RAM są ciągi etykietowanych instrukcji z następującej listy:
  - LOAD op
  - STORE op
  - ADD op
  - SUB op
  - MULT op
  - DIV op
  - READ
  - WRITE
  - JUMP label
  - JGTZ label
  - JZERO label
  - HALT

- argumentem op w instrukcjach może być:
  - liczby całkowite;
  - zawartość rejestru r_i; lub
  - wartość rejestru pod danym adresem r_{r_i};
- Przykład:
  1. ADD r_1
  2. MULT -2
  3. LOAD r_{r_3}
  4. STORE r_7

9.2. Złożoność obliczeń dla modelu RAM

- Model RAM jest nierealistyczny przy założeniu, że może on zmagać z dowolnie dużą liczbą całkowitą.
- W k krokach możemy obliczyć wartość 2^{2^k} podczas, gdy ta liczba zajmuje 2^k bitów na zwykłej maszynie (taśma maszyny Turinga), czyli niemożliwe jest obliczenie tej liczby szybciej niż $O(2^k)$.
- Możemy rozwiązać ten problem wprowadzając koszty obliczeń, zależne od wielkości liczb, dla kolejnych kroków.

ADD r_1	$\log(r_0) + \log(r_1)$
MULT 2	$\log(r_0) + 1$
LOAD r_{r_3}	$\log(r_3) + \log(r_{r_3})$
STORE r_7	$\log(r_0) + 3$

- W ten sposób możemy definiować pojęcie czasu wykonywania dla RAM'owych obliczeń.
- Analogicznie, złożoność pamięciowa programu definiujemy przez największą wartość sumy:

$$\sum_i \log(|r_i|)$$

we wszystkich krokach obliczeń.

9.3. Przykład

Funkcja $f(n) = n^n$ (dla $n > 0$):

READ r_1;

IF $r_1 \leq 0$ THEN WRITE 0;

ELSE

  $r_2 := r_1;$

  $r_3 := r_1 - 1;$

  WHILE $r_3 > 0$ DO

  $r_2 := r_2 * r_1;$

  $r_3 := r_3 - 1;$

  OD

  WRITE r_2;

FI;

- Czas $O(n^2 \log n)$
- pamięć $O(n \log n)$

Home Page
Title Page
Page 28 of 38
Go Back
Full Screen
Close
Quit

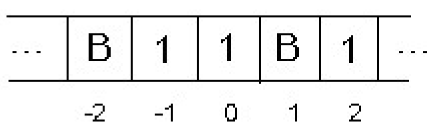
10. Równoważność modeli

Pokażemy, że RAM może symulować maszynę Turinga z 2 symbolami, a 3-taśmowa maszyna Turinga z 4 symbolami może symulować RAM.

10.1. Symulacja maszyny Turinga na modelu RAM

- Zarezerwujmy kilka (np. 10) rejestrów do pracy, a reszty używamy do zapamiętania zawartości taśmy (0 dla blanku, 1 dla symbolu 1)
- Numerujemy komórki taśmy: początek ma numer 0, po lewej stronie ujemne liczby, po prawej stronie – dodatnie.
- Pamiętamy zawartość komórki o numerze i w rejestrze n_i, gdzie:

$$n_i = \begin{cases} 10 + 2i & \text{Jeśli } i \geq 0, \\ 9 - 2i & \text{gdy } i < 0 \end{cases}$$



Rejestr	Zawartość
10	1
11	1
12	0
13	0
14	1

10.1.1. Symulacja maszyny Turinga na modelu RAM – program

- Rejestr R_1 służy do zapamiętania numer bieżącej komórki;
- Stany i instrukcje stanowią "procedury" (bloki) w programie RAM;
- Kod dla każdego stanu s jest następujący:
  - Wczytać zawartość R_1 (LOAD r_1);
  - Zamienić ten numer (bieżącej komórki) na numer rejestru i zapamiętać w R_2;
  - Wczytać zawartość a bieżącej komórki (LOAD r_{r_2});
  - Instrukcja warunkowa (w zależności wyniku: 0 czy 1):
  - W każdym przypadku (instrukcja maszyny Turinga: $(s, a, b, move, t)$):
    * Wpisać nową wartość (a) do R_{r_2};
    * Poprawić zawartość R_1 w zależności od kierunku ruchu ($move$);
    * wykonać JUMP do bloku dla następnego stanu t.

10.1.2. Złożoność symulacji maszyny Turinga

- Załóżmy, że mamy obliczenie na maszynie Turinga, które ma T kroków przy wejściu o rozmiarze n.
- Czyli głowica nigdy nie była dalej niż komórka o numerze $T+n$;
- Największa liczba, jaką widzimy podczas obliczeń jest $O(T+n)$.
- Stąd koszt symulacji jednej instrukcji maszyny Turinga wynosi $O(\log(T+n))$, a koszt całego programu wynosi $O(T\log(T+n))$
- Oszacowanie złożoności pamięciowej jest jeszcze lepsze!

Home Page
Title Page
Page 31 of 38
Go Back
Full Screen
Close
Quit

10.2. Symulacja programu RAM'owego na maszynie Turinga

- Stosujemy 3-taśmową maszynę Turinga z alfabetem $\{0, 1, B, \#\}$;
- Taśma 1 zawiera zawartość wszystkich rejestrów (z wyjątkiem akumulatora). Każdy rejestr jest zapamiętany jako sekwencja $\#\#i\#r_i$, gdzie i i r_i są zapamiętane jako ciągi binarne.

  Na przykład, gdy $r_2 = 17$ mamy

...	#	#	1	0	#	1	0	0	0	1	...

- Taśma 2 zawiera zawartość akumulatora r_0.
- Taśma 3 jest używana jako dodatkowa pamięć.
- Każda instrukcja w programie RAM jest symulowana przez pewien klaster stanów
- Rozpatrujemy kilka przykładów, pozostałe instrukcje są podobne.

10.2.1. Przykład

instrukcja: ADD $r_{r_{20}}$

TM "pseudo-code":

- Znajdź na taśmie 1 wzorca $\#\#10100\#$;
- Jeśli nie występuje, zakończ z błędem;
- Kopiuj dane znajdujące się za tym wzorcem (r_{20}) do taśmy 3;
- Znajdź na taśmie 1 wzorca $\#\#x$, gdzie x jest zawartością tasmy 3;
- Jeśli nie występuje napisz 0 na taśmie 3; w przeciwnym przypadku, przekopiuj dane znajdujące się za tym wzorcem $(r_{r_{20}})$ do taśmy 3;
- Dodaj binarnie liczb na taśmie 2 i 3.

instrukcja: STORE r_{30}

TM "pseudo-code":

- Szukaj wzorca $\#\#11110$ na taśmie 1;
- Jeśli nie występuje, dodaj $\#\#11110\#$ na końcu taśmy i dopisz dane z taśmy 2;
- W przeciwnym przypadku, przekopiuj wszystkie dane na taśmie 1 za znakiem $\#$ wzorca do taśmy 3;
- Przekopiuj dane z taśmy 2 do końca taśmy 1;
- Dopisz zawartość taśmy 3 na końcu taśmy 1;

10.2.2. Wniosek

- Można wykazać, że czas symulacji przez maszynę Turinga wynosi co najwyżej $O(T^2)$, gdzie T jest złożonością programu w RAM.
- Wszystkie obliczenie wykonane na maszynie Turinga w czasie wielomianowym może być realizowane na RAM w czasie wielomianowym (i *vice versa*)

11. Church's Thesis

- We have seen various notions of computability:
  - RAM-computability
  - Partial recursiveness
  - Turing-computability
- We have outlined proofs that they are all the same.
- They can also be shown to match definitions of computability coming from:
  - λ–definability and
  - Systems of Post and Markov based on productions of strings similar to those used in writing down formal grammars for programming languages
- We have worked with computable functions

$$f : \mathbb{N}^k \longrightarrow \mathbb{N}$$

but other things can also be coded as natural numbers.

All this led Church, Turing and Markov to claim:

Church's Thesis: *The intuitively and informally defined class of "effectively computable functions" is exactly the same as the class of URM-computable functions*

- This is not a theorem; it cannot be proved. It deals with informal and intuitive ideas.
- It is also sometimes called the Church-Turing Thesis.

11.1. Status of Church-Turing Thesis

Although unprovable, the thesis is, supported by a great deal of evidence:

- Many independently proposed, and very different, formal notions of computability can be proved to coincide;
- A vast range of functions that seem naturally "computable" have been shown to be TM computable;
- Since a RAM-program is essentially an algorithm, all RAM-computable functions are "computable";
- No one has ever found anything that seems intuitively "computable" which cannot be computed on a RAM or a TM.

Church's Thesis is widely accepted, but is more like a scientific result than a mathematical one – some day it might be disproved by new evidence.

11.2. Use of Church's Thesis

- It is common practice in the light of the evidence, to use Church's thesis to prove that a function is (TM-)computable by simply giving an algorithm for computing it, in the usual careful, but relatively informal, language used for algorithms.
- This is called proof by Church's thesis.
- Two considerations must be born in mind:
  - it is still necessary to prove rigourously that the algorithm:
    - computes the right function;
    - halts after finite time when the function is defined and
    - can be represented as a finite length program (no infinite look-up tables)
  - when you use such a proof you are essentially offering, if challenged, to construct (or prove the existence of) a RAM program, Turing machine or partial recursive function that carries out your algorithm.

We will use proof by Church's thesis in what follows, except for a few key results where we will actually check the details.

11.3. Przykład dowodu za pomocą hipotezy Church'a

- Suppose we have two computable (partial) functions f and g such that, if $f(x)$ and $g(x)$ are both defined, $f(x) = g(x)$.
- Define a function h, such that $h(x)$ is equal to whichever of $f(x)$ and $g(x)$ is defined (or to both)
- Then we can "intuitively" compute $h(x)$ by starting the computations for $f(x)$ and $g(x)$ is parallel, and returning the value returned by whichever process finishes first
- This is a proof "using Church's thesis" that h is computable.
- Actually describing how to build a TM for h from those for f and g would be tedious to do, and even harder to check.