Teoria obliczeń

czyli czego komputery zrobić nie mogą

Marek Zaionc
Uniwersytet Jagielloński

Materiały do wykładu:

P. Odifreddi, Classical Recursion Theory, North Holland 1989.

J.H. Hopcroft, J.D. Ullman "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley 1979

Marek Zaionc, materiały do wykładu Teoria programowania

W naszych rozważaniach będziemy abstrahować od języka programowania w jakim wyrażane będą algorytmy. Może to być

- język wysokiego poziomu, PASCAL, C++, ...
- język niskiego poziomu, assembler, ...
- abstrakcyjne urządzenie obliczające, maszyna Turinga, Posta, ...
- funkcje rekurencyjne

Teza Churcha:

Wszystkie modele obliczeń, które znamy mają taką samą jak maszyny Turinga siłę ekspresji.

Składnia każdego z formalizmów w którym wyrażane są algorytmy pozwala na konstrukcję przeliczalnego, nieskończonego zbioru algorytmów.

Przykładowo dla języka wysokiego poziomu, PASCAL, C++ możemy wygenerować przeliczalnie wiele programów.

Przykładowo dla abstrakcyjnych urządzeń obliczających możemy zbudować przeliczalnie wiele maszyn Turinga, lub maszyn Posta

Zbiór programów lub maszyn jest mocy przeliczalnej.

Zbiór funkcji jest mocy kontinuum.

$$2^{\mathbb{N}} \leqslant_m \mathbb{N}^{\mathbb{N}} \leqslant_m (2^{\mathbb{N}})^{\mathbb{N}} \sim_m 2^{\mathbb{N}\times\mathbb{N}} \sim_m 2^{\mathbb{N}}$$

Moc zbioru funkcji $f : \mathbb{N} \to \mathbb{N}$ jest istotnie większa od mocy zbioru programów lub od zbioru maszyn Turinga.

MORAŁ: Większość funkcji, asymptotycznie prawie wszystkie, nie mają programów do ich obliczania.

Istnieją funkcje nieobliczalne

Jak wiele jest podzbiorów liczb naturalnych dla których istnieją programy rozpoznające, które elementy należą do tych zbiorów?

MORAŁ: Większość zbiorów, które można reprezentować przez ich funkcje charakterystyczne, nie ma programów do rozstrzygania które elementy do tych zbiorów należą.

Istnieją zbiory nierekurencyjne

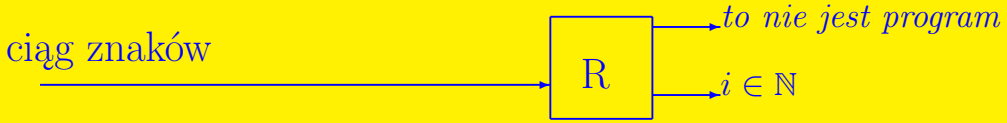
Istnieje program, który rozpoznaje co jest programem, a co nie jest programem.

Programy możemy ponumerować. Ponumerować trzeba najpierw wszystkie słowa, a następnie uruchomić program P który będzie rozpoznawał, czy ciąg znaków jest programem czy też nie.

Istnieje program, który mając na wejściu liczbę naturalną $i \in \mathbb{N}$ zwraca i-ty program.

Program i-ty będziemy nazywali M_i lub programem o indeksie i.

Istnieje program, który mając na wejściu ciąg znaków sprawdza czy jest ten ciąg znaków programem i wtedy zwraca jego indeks. W przypadku gdy nie jest to program zwraca komunikat *to nie jest program*.



Problemy decyzyjne

(1) Problemy dla których odpowiedziami są wyłącznie TAK lub NIE.
(2) Każda instancja takiego problemu ma skończony opis.

Przykłady problemów decyzyjnych:

- Problem stopu
- Problem semantycznej równoważności programów
- Problem syntaktycznej równoważności programów
- Dziesiąty problem Hilberta
- Dziesiąty problem Hilberta ograniczony
- Problem domina (Wang)
- Problem odpowiedniości Posta

Będziemy w tej części mówić jedynie o programach, (maszynach Turinga) które mając na wejściu ciąg znaków $w \in \Sigma^*$ odpowiadają *tak/nie*.

Językiem takiego programu nazywamy zbiór tych słów, które on akceptuje, czyli tych słów dla których odpowiedz brzmi *tak*. Jeżeli M jest takim programem a $\mathcal{S}$ językiem to zapisujemy to $L(M) = \mathcal{S}$.

Program lub maszyna decyzyjna ma własność STOPU gdy zatrzymuje się na każdym wejściu.

Zbiór $\mathcal{S}$ słów nad alfabetem Σ nazywamy *rekurencyjnie przeliczalnym,* jeżeli istnieje program (maszyna Turinga) nad Σ, który akceptuje każde słowo z $\mathcal{S}$, czyli $L(M) = \mathcal{S}$.

Zbiór $\mathcal{S}$ słów nad alfabetem Σ nazywamy *rekurencyjnym,* jeżeli istnieje program (maszyna Turinga) z własnością STOPU nad Σ, który akceptuje każde słowo z $\mathcal{S}$ czyli $L(M) = \mathcal{S}$.

Klasa zbiorów rekurencyjnych jest zamknięta na sumę mnogościową i uzupełnienia.

Zamknięcie na sumę pokazujemy przez sekwencyjne podłączenie maszyn (programów).

Zamknięcie na uzupełnienie pokazujemy przez zamianę styków YES/NO.

Klasa zbiorów rekurencyjnie przeliczalnych jest zamknięta na sumę mnogościową.

Dowód tego faktu pokazujemy przez równoległe podłączenie maszyn (programów).

Jeżeli zbiory $\mathcal{S}$ i $\setminus\mathcal{S}$ są rekurencyjnie przeliczalne to są rekurencyjne.

Dowód podajemy przez równoległe podłączenie maszyn rozpoznających odpowiednio $\mathcal{S}$ i $\backslash\mathcal{S}$.

Język diagonalny

Zbiór $L_{diag} = \{w_i \in \Sigma^* : w_i \notin L(M_i)\}$.
Twierdzenie: L_{diag} nie jest rekurencyjnie przeliczalny.

Dowód:

(1) Stawiamy hipotezę, że L_{diag} jest rekurencyjnie przeliczalny.
(2) Istnieje zatem maszyna M_{diag} rozpoznająca L_{diag} czyli $L(M_{diag}) = L_{diag}$
(3) Maszyna M_{diag} ma indeks (numer) i_0.
(4) Stawiamy pytanie czy M_{i_0} akceptuje słowo w_{i_0}.
(5) Pokazujemy
$w_{i_0} \in L(M_{i_0})$ wtedy i tylko wtedy $w_{i_0} \notin L(M_{i_0})$.

Język uniwersalny L_u

$$L_u = \{Mw \in \Sigma^* : M \text{ jest programem i } w \in L(M)\}$$

Język L_u jest rekurencyjnie przeliczalny.

Dowód: Konstrukcja uniwersalnego symulatora maszyn Turinga.

Twierdzenie: Język L_u nie jest rekurencyjny.

Dowód:
1. Hipoteza - L_u jest rekurencyjny.
2. Zatem $\backslash L_u$ też jest rekurencyjny.
3. Istnieje zatem maszyna $\overline{M}$ rozpoznająca $\backslash L_u$.
4. Budujemy nowy program decyzyjny (maszynę), który będzie używał $\overline{M}$ jako swojej procedury, który będzie rozpoznawał L_{diag} .

Oto program dla L_{diag}.

(1) wczytaj słowo w.
(2) policz indeks i_0 tego słowa.
(3) wygeneruj i_0-wą maszynę Turinga M_{i_0}. (instrukcje tą wykona procedura Q.)
(4) podaj na wejście maszyny $\overline{M}$ parę $M_{i_0}w_{i_0}$.
    - jeżeli $\overline{M}$ z wejściem $M_{i_0}w_{i_0}$ odpowie YES to odpowiedz YES
    - jeżeli $\overline{M}$ z wejściem $M_{i_0}w_{i_0}$ odpowie NO to odpowiedz NO

Naszą hipotezę doprowadziliśmy do sprzeczności.

Problemy rozstrzygalne i częściowo rozstrzygalne

Problem rozstrzygalny to taki, którego język kodujący jest rekurencyjny.

Problem częściowo rozstrzygalny to taki, którego język kodujący jest rekurencyjnie przeliczalny.

Uniwersalny problem decyzyjny: Dany jest program M i dane jest słowo $w \in \Sigma^*$.
Podejmij decyzję czy $w \in L(M)$.

Problem diagonalny nie jest nawet częściowo rozstrzygalny

Problem uniwersalny nie jest rozstrzygalny.

Problem uniwersalny jest częściowo rozstrzygalny

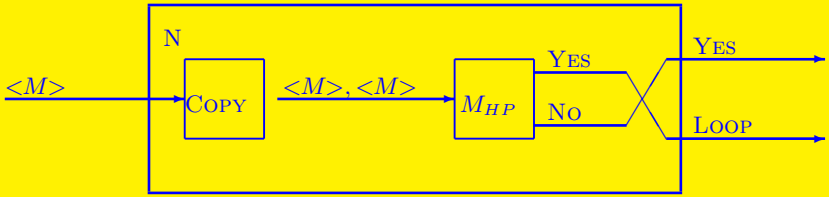
Problem stopu

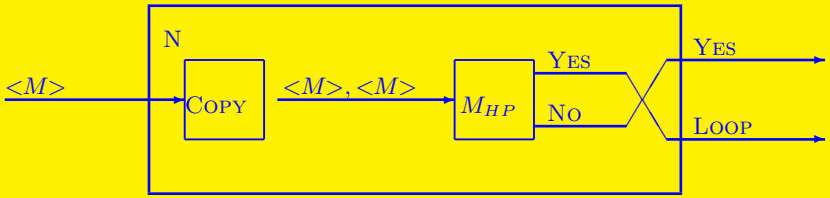
Rozważmy język

$$L_{HP} = \{M, x : \ M \text{ zatrzymuje się na } x\}$$

$$L_u \subsetneq L_{HP}$$

TWIERDZENIE: L_{HP} nie jest rekurencyjny (dowód pochodzi od Minsky'ego).
Hipoteza L_{HP} jest rekurencyjny. Niech M_{HP} będzie maszyną ze STOPEM rozpoznającą L_{HP}.
Budujemy nową maszynę o nazwie N, która analizuje kody innych maszyn.


N
<M>
Copy
<M>, <M>
M_{HP}
Yes
No
Yes
Loop



Specyfikacja maszyny N.
N akceptuje kod maszyny M wtw M pętli się na swoim własnym kodzie.

Problem *STOPU* jest nierozstrzygalny, ale jest częściowo rozstrzygalny. Problem dualny czyli problem *PĘTLENIA SIĘ* nie jest nawet częściowo rozstrzygalny.

Przykłady 1.

Pustość lub niepustość języka.

Rozważmy języki

$$L_{empty} = \{w \in \Sigma^* : \text{jeżeli } \exists_{maszyna\ M} \text{ taka że } w = M \text{ to } L(M) = \emptyset\}$$

$$L_{notempty} = \{M \in \Sigma^* : L(M) \neq \emptyset\}$$

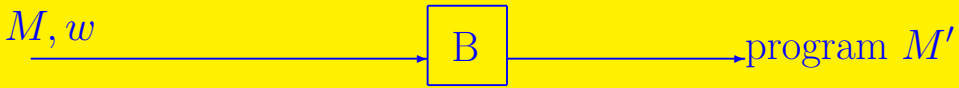
$$L_{empty} \cup L_{noempty} = \Sigma^*$$

Twierdzenie:

L_{empty} nie jest rekurencyjny.

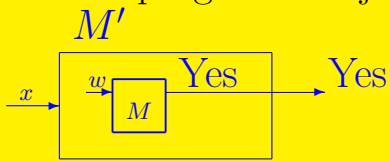
Dowód:
Stawiamy hipotezę: L_{empty} jest rekurencyjny.
Niech zatem M_{empty} będzie maszyną ze STOPEM rozpoznającą L_{empty}.
Dokonamy konstrukcji maszyny B budującej inne maszyny. Maszyna B jest rodzajem konstruktora maszyn i działa według następującego schematu:

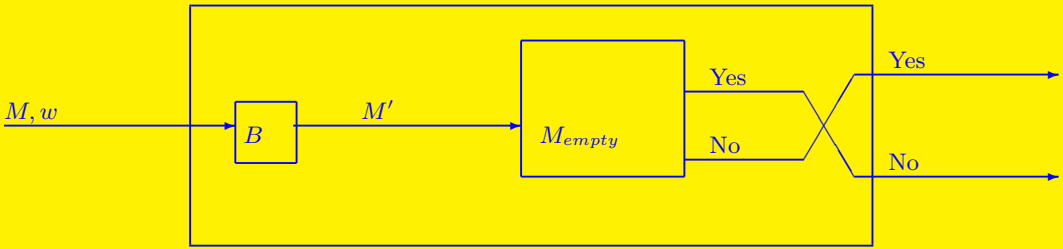
MASZYNA B ma na wejściu dowolny problem uniwersalny czyli parę M, w. Efektem jej działania jest wypisanie nowej maszyny M'



Maszyna B jest rodzajem pisarza programów. Napisany przez B nowy program będzie spełniał następującą specyfikacje:
Jeżeli M akceptuje słowo w to M' akceptuje Σ^*.
Jeżeli M nie akceptuje słowa w to M' akceptuje $\emptyset$.

Napisane przez konstruktora B program M' jest następujący:




M, w
B
M'
M_{empty}
Yes
No
Yes
No

$L_{noempty}$ nie jest rekurencyjny.

Napiszemy program rozpoznający $L_{noempty}$.
Niech $NEXT : \mathbb{N}^2 \to \mathbb{N}^2$ będzie funkcją, która punktowi na płaszczyźnie $\mathbb{N} \times \mathbb{N}$ przyporządkowuje następny punkt w porządku liniowym przy pomocy którego możemy liniowo przeglądać pary punktów. Łatwo napisać program, który zrealizuje funkcje $NEXT$.

(1) Wczytaj słowo w.
(2) Sprawdź czy w jest programem przy pomocy procedury P.
- Jeżeli w nie było programem to zakończ i wypisz NO
- Jeżeli w jest programem M to kontynuuj

(3) $(j, k) := (0, 0)$;
(4) Symuluj pierwsze k kroków programu M na j-tym słowie z Σ^*.
(5) Jeżeli M akceptuje w to odpowiedz YES i zatrzymaj program.
(6) Jeżeli M nie akceptuje w to $(j, k) := NEXT(j, k)$; i powtarzaj 4.

Program poprawnie rozpoznaje język $L_{noempty}$ aczkolwiek nie ma własności STOPU.

L_{empty} nie jest nawet rekurencyjnie przeliczalny.

Problemy decyzyjne związane z parą języków L_{empty} i $L_{noempty}$.

Problem niepustości języka jest częściowo rozstrzygalny ale nierozstrzygalny.

Problem pustości języka nie jest nawet częściowo rozstrzygalny

Przykład 2.

Rozważmy języki

$$L_{rek} = \{M \in \Sigma^* : L(M) \text{ jest rekurencyjny } \}$$
$$L_{norek} = \{M \in \Sigma^* : L(M) \text{ nie jest rekurencyjny } \}$$

$L_{empty} \cup L_{noempty} =$ Wszytkie programy

Przykład 3.

Równoważność programów.

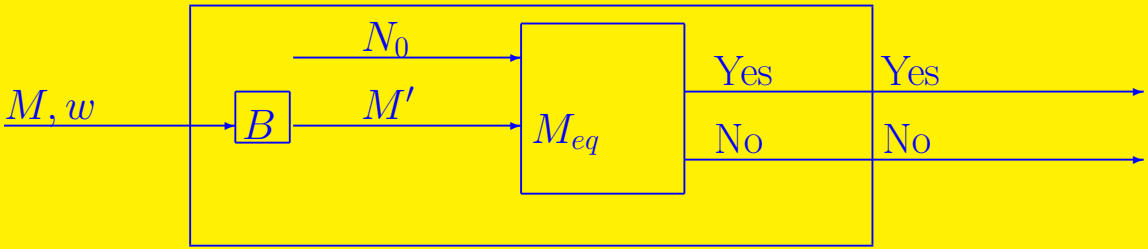
$$L_{eq} = \{MN \in \Sigma^* : L(M) = L(N)\}$$

Twierdzenie: L_{eq} nie jest nawet rekurencyjnie przeliczalny.

Hipoteza: Załóżmy że L_{eq} jest rekurencyjnie przeliczalny i rozpoznawany przez M_{eq}, czyli $L(M_{eq}) = L_{eq}$.

Ustalmy maszynę N_0, która nie akceptuje żadnego słowa. W konstrukcji posłużymy się również poznanym wcześniej konstruktorem programów B.

Przy pomocy maszyn N_0, B oraz hipotetycznej maszyny M_{eq} zbudujmy następującą maszynę:


M, w
B
N_0
M'
M_{eq}
Yes
No
Yes
No

Twierdzenie Rice'a

Klasę $\mathcal{S}$ języków rekurencyjnie przeliczalnych nazywamy trywialną gdy $\mathcal{S} = \emptyset$ lub $\mathcal{S} = \mathcal{REC}$.

Na klasę języków należy patrzeć jak na własność języków.

Klasa jest nietrywialna jeżeli nie jest trywialna.

Język $L_{\mathcal{S}}$ skojarzony z klasą języków $\mathcal{S}$ to język wszystkich kodów maszyn Turinga, których języki leżą w $\mathcal{S}$, czyli

$$L_{\mathcal{S}} = \{M : L(M) \in \mathcal{S}\}$$

Obserwacja: Dopełnienie nietrywialnej klasy jest klasą nietrywialną

Twierdzenie Rice'a:

Każda nietrywialna własność języków rekurencyjnie przeliczalnych jest nierozstrzygalna.

Dowód nie wprost: Niech $\mathcal{S}$ będzie nietrywialną klasą. Zakładamy, że pomimo nietrywialności $\mathcal{S}$ istnieje dla niej maszyna $M_{\mathcal{S}}$ ze STOPEM rozpoznająca $L_{\mathcal{S}}$.

Bez szkody dla ogólności mogę założyć, że język pusty nie należy do klasy $\mathcal{S}$, czyli

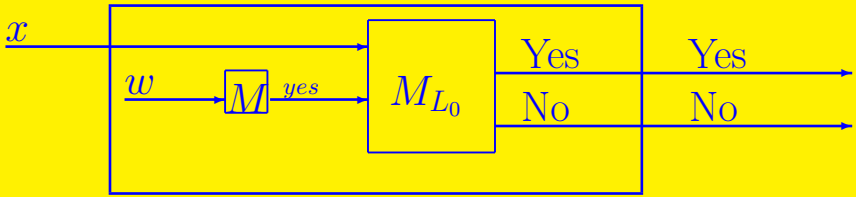
$$\emptyset \notin \mathcal{S}.$$

Jednocześnie z niepustości klasy $\mathcal{S}$ istnieje w niej język rekurencyjnie przeliczalny $L_0 \neq \emptyset$. Niech maszyna M_{L_0} rozpoznaje L_0 czyli $L(M_{L_0}) = L_0$.

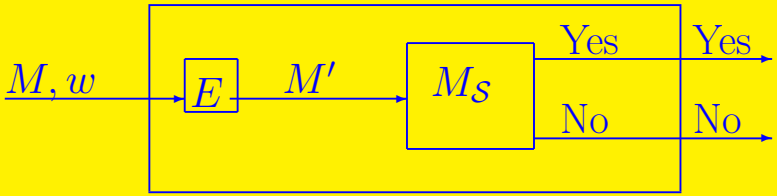
Budujemy maszynę E będącą rodzajem pisarza programów. Pisarz E akceptuje problem uniwersalny Mw i przerabia je na nowy programy M' zgodnie z następującą specyfikacją:

Jeżeli M akceptuje słowo w to M' akceptuje $L_0 \in \mathcal{S}$.
Jeżeli M nie akceptuje słowa w to M' akceptuje $\emptyset \notin \mathcal{S}$.

Konstrukcja programu M'



Przy pomocy programu E oraz rozpoznawacza klasy $M_{\mathcal{S}}$ napiszemy program ze stopem rozwiązujący problem uniwersalny. Oto on:



Twierdzenie Rice'a:

Wnioskowanie w stronę przeciwną jest proste.

Każda trywialna własność języków rekurencyjnie przeliczalnych jest rozstrzygalna.

Mamy dwie klasy trywialne, klasę pustą i klasę pełną.
Mamy:
$L_\emptyset = \{M : L(M) \in \emptyset\} = \emptyset$
$L_{REC} = \{M : L(M) \in REC\} =$ Maszyny Turinga

Oba te języki są rekurencyjne.

Przykłady zastosowań:
Szybkie dowody nierozstrzygalności semantycznych własności maszyn Turinga.

O zachowaniu się programów:

Każde nietrywialne zachowanie programów jest nierozstrzygalne.

Przykład 2.

Rozważmy języki

$$L_{rek} = \{M \in \Sigma^* : L(M) \text{ jest rekurencyjny } \}$$
$$L_{norek} = \{M \in \Sigma^* : L(M) \text{ nie jest rekurencyjny } \}$$

$L_{empty} \cup L_{noempty} =$ Wszytkie programy

Przykład 4.

$$L = \{MNK : L(M) \cup L(N) = L(K)\}$$

Czy język L jest rekurencyjny ?

Twierdzenie Rice'a dla języków rekurencyjnie przeliczalnych:

Jakie własności powinna mieć klasa języków $\mathcal{S}$ aby język kodujący tą klasę $L_{\mathcal{S}}$ był rekurencyjnie przeliczalny.

Innymi słowy: jaka powinna być klasa $\mathcal{S}$ aby problem przynależności do $L_{\mathcal{S}}$ był częściowo rozstrzygalny.

Warunkiem koniecznym i wystarczającym na to by problem przynależności do $L_{\mathcal{S}}$ był częściowo rozstrzygalny jest:

- Klasa $\mathcal{S}$ jest monotoniczna.
- Każdy język z $\mathcal{S}$ posiada podjęzyk skończony w $\mathcal{S}$.
- Klasa $\mathcal{S}$ posiada numerator języków skończonych należących do $\mathcal{S}$.