Języki formalne, automaty i teoria obliczeń
AUG/JAO/TO

Marcin Kubica

2004/2005

Niniejsze notatki do wykładów oparte są na podanej literaturze. Podane w nawiasach kwadratowych numery podrozdziałów, ćwiczeń itp. odnoszą się właśnie do wymienionych poniżej pozycji.

Wykłady oznaczone gwiazdką należy pominąć na zajęciach z JAO, a uwzględnić na wykładach z TO.

Wykład 1. Wprowadzenie

Celem niniejszego wykładu (w zależności od przedmiotu) jest:

- zrozumienie podstawowych pojęć dotyczących obliczalności, takich jak:
  - co to znaczy, że funkcja jest obliczalna?
  - czy są funkcje nieobliczalne?
  - w jakim stopniu to co możemy obliczyć zależy od dostępnych konstrukcji programistycznych?
- poznanie podstawowych środków opisu składni (wyrażenia regularne, gramatyki bezkontekstowe),
- poznanie narzędzi wspomagających przetwarzanie danych o określonej składni (flex, bison).

Nie są to proste pytania. Szukając na nie odpowiedzi napotkamy takie pojęcia, jak:

- stany,
- przejścia,
- niedeterminizm,
- sprowadzanie (redukcja),

- nieobliczalność.

W trakcie badań nad obliczalnością pojawiło się wiele modeli, które miały za zadanie ujęcie pojęcia *obliczalności*. Możemy je uszeregować wg. rosnącej siły wyrazu:

1. automaty skończone, wyrażenia regularne (stała skończona pamięć),
2. automaty stosowe (stała skończona pamięć + nieograniczony stos),
3. bez ograniczeń:
   - maszyny Turinga,
   - systemy Posta,
   - rachunek λ (Alonzo Church, Stephen C. Kleene), i inne.

Modele te powstały na długo przed tym zanim powstały komputery.

Równocześnie Noam Chomsky, lingwista, próbował sformalizować pojęcia *gramatyki* i *języka*. Stworzył on hierarchię języków o rosnącej złożoności:

1. gramatyki liniowe,
2. gramatyki bezkontekstowe,
3. gramatyki kontekstowe,
4. gramatyki dowolne.

Hierarchia ta w zadziwiający sposób odpowiada wymienionym modelom obliczeń:

- gramatyki liniowe = automaty skończone = wyrażenia regularne,
- gramatyki bezkontekstowe = automaty stosowe,
- gramatyki kontekstowe = liniowo ograniczone maszyny Turinga,
- dowolne gramatyki = maszyny Turinga.

Czy może być to tylko przypadek?

Należy pamiętać, że wszystko to są *modele* interesujących nas pojęć. Umiejętność posługiwania się niektórymi z nich przydaje się w praktyce, a w przypadku innych najistotniejsze będą reprezentowane przez nie pojęcia.

1.1 Literatura

- [HU] John E. Hopcroft, Jeffrey D. Ullman, *Wprowadzenie do teorii automatów, języków i obliczeń*, PWN, 2003 r.
- [K] Dexter C. Kozen, *Autoamata and Computability*,
- A. V. Aho, R. Sethi, J. D. Ullman, Kompilatory, WNT, 2002,
- W. M. Waite, G. Goos, Konstrukcja kompilatorów, WNT, 1989.

1.2 Podstawowe pojęcia

We wprowadzeniu wspomnieliśmy o *obliczeniach* i *językach*. Gdy mówimy o obliczaniu, to zwykle mamy na myśli obliczenie wartości takiej czy innej funkcji. Gdy zaś mówimy o języku, to mamy na myśli (potencjalnie nieskończony) zbiór napisów złożonych ze znaków ustalonego (skończonego) alfabetu. Jak połączyć te dwa pojęcia?

Def. 1 *Problem decyzyjny* to funkcja, która możliwym danym wejściowym przyporządkowuje wartości logiczne tak/nie. □

Język nie jest niczym więcej niż problemem decyzyjnym określonym dla napisów nad ustalonym alfabetem.

Def. 2 Alfabet, napis itp.

- *Alfabet* to dowolny niepusty, skończony zbiór. Może to być np. zbiór cyfr dziesiętnych, zbiór znaków ASCII, czy zbiór bitów $\{0, 1\}$. Alfabet będziemy zwykle oznaczać przez Σ, a elementy alfabetu przez $a, b, c, \ldots$.
- *Napis* nad alfabetem Σ, to dowolny skończony ciąg elementów Σ. W szczególności, pusty ciąg jest napisem nad każdym alfabetem. Oznaczamy go przez ε. Napisy będziemy zwykle reprezentować przez zmienne $x, y, z, \ldots$.
- *Długość* napisu oznaczamy przez $|\cdot|$. Np. $|ala| = 3$, $|\varepsilon| = 0$.
- *Sklejanie* napisów zapisujemy pisząc po sobie sklejane składowe. Np., jeżeli $x = ala, y = ma, z = kota$, to $xyz = alamakota$. Sklejanie napisów jest łączne, tzn. $x(yz) = (xy)z$, a ε jest elementem neutralnym sklejania, tzn. $x\varepsilon = \varepsilon x = x$.
- Przez a^n oznaczamy n-krotne powtórzenie symbolu a. Np. $a^5 = aaaaa$. $a^0 = \varepsilon$, $a^{n+1} = a^n a$.
- Podobnie, przez x^n oznaczamy n-krotne powtórzenie napisu x. $x^0 = \varepsilon$, $x^{n+1} = x^n x$.
- Przez $\#_a(x)$ oznaczamy *liczbę wystąpień* symbolu a w napisie x, np. $\#_a(ala) = 2$.
- *Prefiksem* napisu nazywamy dowolny jego początkowy fragment, tzn. x jest prefiksem y wtw., gdy istnieje takie z, że $xz = y$. Jeżeli ponadto $x \neq \varepsilon$ i $x \neq y$, to mówimy, że x jest *właściwym* prefiksem y.

□

Def. 3 Zbiory (języki) będziemy zwykle oznaczać przez $A, B, C, \ldots$. Operacje na zbiorach (napisów):

- suma $\cup$, przecięcie $\cap$, odejmowanie $\setminus$,
- dopełnienie $\overline{A} = \Sigma^* \setminus A$,

- złączenie (konkatenacja) zbiorów $AB = \{xy : x \in A, y \in B\}$,
- potęgowanie zbiorów: $A^0 = \{\varepsilon\}$, $A^{n+1} = A^n A$,
- $A^* = \bigcup_{n \geq 0} A^n$; w szczególności Σ^* to *zbiór wszystkich napisów* nad alfabetem Σ; przyjmujemy, że $\emptyset^* = \{\varepsilon\}$,
- $A^+ = AA^*$.

□

Zwróćmy uwagę na pewne proste własności operacji na zbiorach:

- $\{\varepsilon\}A = A\{\varepsilon\} = A$,
- $\emptyset A = A\emptyset = \emptyset$,
- $A(B \cup C) = AB \cup AC$.

1.3 Języki jako problemy decyzyjne

Język, to podzbiór zbioru Σ^*. Możemy więc język traktować jak problem decyzyjny, dla zbioru danych Σ^*.

Czy dla każdego języka istnieje rozpoznający go automat skończony / automat stosowy / program komputerowy. Zdecydowanie nie!

Zbiór Σ^* jest przeliczalny, gdyż jesteśmy w stanie ponumerować wszystkie słowa: puste, jednoliterowe, dwuliterowe, itd. Czyli $|\Sigma^*| = \aleph_0$. Język to podzbiór Σ^*. Czyli wszystkich możliwych języków jest $2^{\aleph_0}$. Z drugiej strony każdy mechanizm rozpoznający można opisać za pomocą słowa — program komputerowy to po prostu słowo. Tak więc języków jest ilościowo więcej niż mechanizmów rozpoznających. Stąd, tylko niektóre języki można rozpoznawać automatycznie, niezależnie od wybranego modelu obliczeń.

1.4 Powtórzenie pojęć dotyczących relacji

Kiedy relacja jest:

- zwrotna,
- przechodnia,
- symetryczna,
- antysymetryczna.

Grafowa interpretacja powyższych pojęć. Pojęcie domknięcia relacji:

- zwrotniego,

- przechodniego,
- symetrycznego

i ich interpretacja grafowa.

1.5 Praca domowa

- Jak jest różnica między $\emptyset$ i ε? Ile słów zawierają te języki?
- Podaj wszystkie słowa z języka $\{aba, ba, a\}\{ba, baba\}$.
- Podaj które prefiksy słowa *ababbbabab* są równocześnie jego sufiksami.

Ćwiczenia

- Ćwiczenia na podstawy matematyczne (można wykorzystać przed pierwszym wykładem):
  - Operacje na zbiorach. Niech $A, B \subset X$. Narysować diagram przedstawiający te zbiory. Uszeregować poniższe zbiory zgodnie z zawieraniem. Które są sobie równe? Które nie są porównywalne? Rozwiązanie można przedstawić w postaci DAG-a.
    * $A \cup B$,
    * $A \cap B$,
    * X,
    * A,
    * B,
    * $\emptyset$,
    * $X \setminus A$,
    * $A \cap B \cap (X\ A)$,
    * $(A \cup B) \setminus (A \cap B)$,
    * $X \setminus (A \cup B)$,
    * $X \setminus (A \cap B)$,
    * $(A \setminus B) \cup (B \setminus A)$,
    * $(X \setminus A) \cup (X \setminus B)$,
    * $(X \setminus A) \cap (X \setminus B)$,
    * $X \setminus ((X \setminus A) \cup (X \setminus B))$,
    * $X \setminus ((X \setminus A) \cap (X \setminus B))$.
  - Pojęcia prefiksu, postfiksu, podciągu i podciągu spójnego (podsłowa). (W razie potrzeby wyjaśnić te pojęcia). Wybieramy dowolny numer telefonu i traktujemy go jak ciąg cyfr. Podać:
    * wszystkie prefiksy tego ciągu,
    * wszystkie postfiksy tego ciągu,
    * kilka podciągów (niekoniecznie spójnych); ile jest wszystkich?
    * kilka spójnych podciągów (podsłów); ile jest wszystkich?
  - Wybieramy zbiór 2–3 elementowy. Jakie są jego podzbiory? Ile podzbiorów ma zbiór n-elementowy?
- proste przećwiczenie wprowadzonych pojęć na przykładach, np.:
  - $\{a, ab, ba, baba\} \cap \{ab, ba\}^2$,
  - $\{a, ab, ba\} \setminus \{a\}^*\{bb\}^*\{a\}^*$.

- ćwiczenia na sprawdzanie różnych tożsamości (w przypadku prawdziwych uzasadnienie, w przypadku fałszywych kontrprzykład; jeśli zachodzi zawieranie, to w którą stronę?):
  - $(A \cup B) \cup C = A \cup (B \cup C)$,
  - $(A \cup B) \cap C = A \cup (B \cap C)$,
  - $(A \cup B) \cap C \subseteq A \cup (B \cap C)$,
  - $A(B \cup C) = AB \cup AC$,
  - $A(B \cap C) = AB \cap AC$,
  - $A(B \cap C) \subseteq AB \cap AC$,
  - $(AB)^* = A^*B^*$,
  - $(A \cup B)^* = (A^*B^*)^*$,
  - $A^{*+} = A^{+*}$,
  - $\overline{(A \cup B)} = \overline{A} \cap \overline{B}$.
- Co to za języki:
  - $(A \cap A^*)^*$,
  - $A^* \setminus A^+$,
  - $(A \cap B)^* \setminus (A^* \cup B^*)$,
  - $\{a, ba, bb, ab\}^*$,
  - $(\Sigma\Sigma)^*$, (słowa parzystej długości),
  - $\overline{(\Sigma\Sigma)^*}$, (słowa nieparzystej długości),
  - $\Sigma(\Sigma\Sigma)^*$ (też słowa nieparzystej długości),
  - ΠΙΣΣΑ (a to jest po grecku pizza :-).
- Używając symboli i operacji wprowadzonych na pierwszym wykładzie (czyli operacji na słowach, zbiorach, językach i definicji zbiorów) zdefiniować języki:
  - słowa zawierające tyle samo liter a i b,
  - słowa nad alfabetem $\{a, b\}$ (nie)parzystej długości,
  - alarmowe, skrócone i międzymiastowe numery telefoniczne,
  - inne ...
- [HU ćw.1.7] Podać domknięcie zwrotnio-przechodnie i domknięcie symetryczne relacji $\{(1, 2), (2, 3), (3, 4), (5, 4)\}$.
- Pokazać, że następujące relacje są relacjami równoważności:

  - słowa x i y są w relacji wtw., gdy $|x| = |y|$,
  - słowa x i y są anagramami,
  - języki A i B są w relacji wtw., gdy $A^* = B^*$.

- Pokazać, że następujące relacje są częściowymi porządkami:
  - relacja zawierania $\subseteq$ na językach nad ustalonym alfabetem,
  - relacja porządku leksykograficznego na słowach,
  - relacja bycia prefiksem.

Wykład 2. Deterministyczne automaty skończone i języki regularne

Stan i przejście — intuicje. Zakładamy, że przejścia są natychmiastowe — jest to matematyczna abstrakcja. Przykłady systemów ze stanami i przejściami:

- kostka Rubik'a,
- zegarek cyfrowy,
- sterownik windy,
- wszechświat (?).

Przykład: Konstrukcja automatu modelującego działanie video. Modelujemy podstawowe typy pracy i klawisze: play, stop, pause, fwd i rew.

Przykład: Problem misjonarzy i kanibali. Na brzegu znajduje się trzech misjonarzy i trzech kanibali. Mają do dyspozycji łódkę. W łódce mogą się zmieścić maksymalnie dwie osoby. Wiosłować umieją wszyscy kanibale i jeden misjonarz. Jeżeli na którymkolwiek brzegu zostanie więcej kanibali, niż misjonarzy, to misjonarze zostaną ... zjedzeni.

Obserwujemy sekwencje ruchów łódki wraz z jej pasażerami. Skonstruować diagram systemu akceptującego takie sekwencje ruchów, które prowadzą do przepłynięcia wszystkich na drugi brzeg.

Def. 4 Automat skończony (deterministyczny):

$$M = \langle Q, \Sigma, \delta, s, F \rangle$$

gdzie:

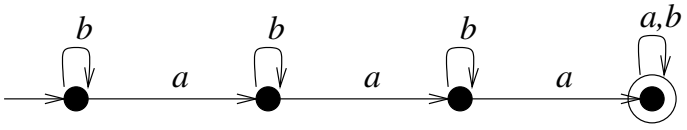
- Q to skończony zbiór *stanów*,
- Σ to skończony *alfabet wejściowy*,
- $\delta : Q \times \Sigma \to Q$ to *funkcja przejścia*; $\delta(q, a)$ to stan do którego przechodzimy ze stanu q na skutek wczytania znaku a,
- s to stan początkowy,
- $F \subseteq Q$ to zbiór stanów akceptujących.

□

Aby określić automat musimy podać informacje o wszystkich pięciu elementach. Możemy to zrobić formalnie, lub w postaci tabelki:

		a	b
$\rightarrow$	0	1	0
	1	2	1
	2	3	2
	$3F$	3	3

lub też diagramu:



Zaletą takich diagramów jest np. to, że nie ma konieczności nazywania stanów.

Dane dla automatu to dowolny napis $x \in \Sigma^*$. Opis działania: Zaczynamy w stanie początkowym i zmieniamy stany zgodnie ze znakami na wejściu. Po wczytaniu całego słowa jesteśmy w jakimś stanie. Słowo jest akceptowane, wtw., gdy stan końcowy jest akceptujący.

Przykład: Działanie powyższego automatu dla słów *baabbab* i *babbab*.

Def. 5

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, xa) = \delta(\delta * (q, x), a)$$

Automat *akceptuje słowo* x wtw., gdy $\delta^*(s, x) \in F$.
Język akceptowany przez automat, to:

$$L(M) \equiv \{x \in \Sigma^* : \delta^*(s, x) \in F\}$$

Powiemy, że język A jest *regularny*, jeśli istnieje taki automat skończony M, że $A = L(M)$. □

Przykład: Automat akceptujący $\{a, b\}^* aa \{a, b\}^*$. Stany reprezentują zliczane kolejne z rzędu literki a.

Przykład: Automat akceptujący $\{x \in \{0,1\}^* : x$ w układzie binarnym jest podzielne przez 3 $\}$. Stany reprezentują reszty modulo 3. Formalny dowód poprawności.

$$x0 = 2x \equiv (2(x \text{mod} 3)) \text{mod} 3$$

$$x1 = 2x + 1 \equiv (2(x \text{mod} 2) + 1) \text{mod} 3$$

2.1 Własności domknięcia klasy języków regularnych

Twierdzenie: Niech A i B będą językami regularnymi. Wówczas następujące języki są również regularne:

- $A \cup B$,
- $A \cap B$,
- $\overline{A}$,
- AB,
- A^*.

Dowód: Niech M_1 akceptuje A, a M_2 akceptuje B.

- Dopełnienie — przez dopełnienie F.
- Konstrukcja *automatu produktowego* M_3. Lemat: $\delta_3^*((p,q),x) = (\delta_1^*(p,x), \delta_2^*(q,x))$. Dowód indukcyjny. Z lematu mamy: $L(M_3) = L(M_1) \cap L(M_2)$.
- Z praw De Morgana mamy sumę języków.
- Sklejenie i gwiazdkę zostawiamy na potem.

Ćwiczenia

- [K Hw. 1.1] podaj deterministyczny automat skończony dla języków:
  - słowa nad alfabetem $\{4, 8, 1\}$ zawierające podsłowo 481,
  - słowa nad alfabetem $\{a\}$ o długości podzielnej przez 2 lub 7,
  - słowa nad alfabetem $\{0, 1\}$, które zawierają parzysta liczbę zer, a liczba jedynek jest podzielna przez 3,
  - słowa nad alfabetem $\{a, b\}$ zawierające przynajmniej 3 wystąpienia słowa bbb (być może zachodzące na siebie),
  - zapisy w układzie trójkowym liczb podzielnych przez 4.
- [HU ćw. 2,5] podaj deterministyczny automat skończony dla języków:
  - słowa zakończone na 00,
  - słowa zawierające trzy zera z rzędu,
  - słowa, w których co piąty symbol, licząc od początku, jest jedynką,
  - słowa, w których każde kolejne 5 symboli, licząc od początku, zawiera przynajmniej dwa zera.
- Skończony automat deterministyczny rozpoznający te słowa, w których na parzystych pozycjach występują 1-ki.
- Skończony automat deterministyczny rozpoznający te słowa, w których każde trzy kolejne znaki, licząc od początku słowa, zawierają choćjedną 1-kę.
- Automat akceptujący słowa:
  - zawierające podsłowa abb,
  - nie zawierające podsłowa aba,
  - zawierające dany numer telefonu, np. 625623, czyli *625623* (dobrze, żeby numer telefonu był trochę złośliwy i zawierał swoje własne prefiksy).
- [K Ex. 2] Podaj automaty akceptujące przecięcie/sumę języków akceptowanych przez poniższe automaty:

		a	b
$\rightarrow$	1	2	2
F	2	1	1

		a	b
$\rightarrow$	1	1	2
F	2	2	1

		a	b
$\rightarrow$	1	2	3
F	2	3	1
F	3	1	2

		a	b
$\rightarrow F$	1	3	2
	2	1	3
F	3	2	1

		a	b
$\rightarrow$	1	2	2
F	2	1	1

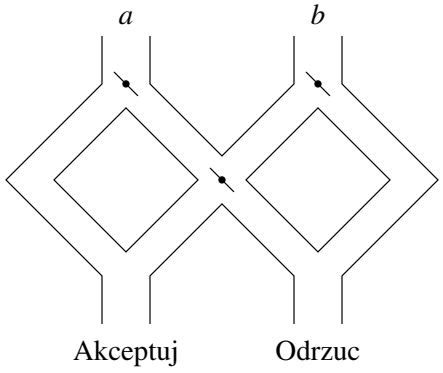
		a	b
$\rightarrow$	1	2	1
F	2	1	2

- [K Hw. 1.2] Podaj automaty akceptujące przecięcie/sumę języków akceptowanych przez poniższe automaty:

		a	b
$\rightarrow$	1	1	2
F	2	2	1

		a	b
$\rightarrow$	1	2	3
	2	3	1
F	3	1	2

- Automat akceptujący zapisy 10-tne liczb podzielnych przez 3.
- Automat akceptujący:
  - $\{ab\}^*$,
  - $\{ab\}^* \cup \{ba\}^*$
- [HU ćw. 2.3] Rozważamy mechanizm pokazany na rysunku. Stosownie do wczytywanych znaków, wrzucamy kulki w miejscach oznaczonych a i b. Kulki spadając wybierają kierunki zgodne z ustawieniem zapadek. Jednak kulka spadając powoduje zmianę ustawienia zapadki. Zamodeluj przedstawiony mechanizm jako automat skończony. Rysunek przedstawia stan początkowy. Jeśli kulka wypadnie w miejscu oznaczonym „akceptuj", to stan powinien być akceptujący.



Wykład 3. Automaty niedeterministyczne[JFA,JAO]

3.1 Intuicje

„Rozklekotany" automat deterministyczny. Nie wiadomo dokładnie jaki ruch wykona, może wykonać *niedeterministycznie* jeden z wielu ruchów. Symulacja: diagram to plansza, po której przesuwamy pionki. Jeśli ze stanu, w którym jest pionek wychodzi wiele krawędzi oznaczonych symbolem wyjściowym, to stawiamy pionki we wszystkich stanach docelowych; jeżeli z danego stanu nie wychodzi żadna taka krawędź, to pionek znika. Podobnie możemy mieć więcej niż jeden stan początkowy — na początku symulacji ustawiamy pionki we wszystkich stanach początkowych.

Automat akceptuje słowo jeśli po zakończeniu symulacji w którymkolwiek ze stanów akceptujących będzie znajdować się pionek. Innymi słowy, automat akceptuje słowo jeśli *może* się zdarzyć takie obliczenie, które prowadzi do jego zaakceptowania.

Można o tym pomyśleć jak o przeplatających się dwóch procesach: zgadywania rozwiązania i jego weryfikacji.

Przykład: Nieformalny. Automat akceptujący słowa, w których przedostatni znak to 1. Symulacja dla słowa 0110010.

3.2 Definicja automatu

Def. 6 Automat niedeterministyczny, to piątka $M = \langle Q, \Sigma, \delta, S, F \rangle$, gdzie:

- Q to skończony zbiór stanów,
- Σ to skończony alfabet wejściowy,
- δ to funkcja przejścia $\delta : Q \times \Sigma \to P(Q)$ ($\cong$ relacja $\delta \subseteq Q \times \Sigma \times Q$),
- $S \subseteq Q$ to zbiór stanów początkowych,
- $F \subseteq Q$ to zbiór stanów akceptujących.

Rozszerzenie funkcji przejścia $\delta^* : P(Q) \times \Sigma^* \to P(Q)$ definiujemy następująco:

$$\delta^*(A, \varepsilon) = A$$

$$\delta^*(A, ax) = \delta^*(\delta(A, a), x)$$

Automat M akceptuje słowo x wtw., gdy $\delta^*(S, x) \cap F \neq \emptyset$.
Język akceptowany przez automat, to:

$$L(M) = \{x \in \Sigma^* : \delta^*(S, x) \cap F \neq \emptyset\}$$

□

Automaty deterministyczne możemy traktować jako szczególny przypadek automatów niedeterministycznych.

Fakt: Niech $M = \langle Q, \Sigma, \delta, s, F \rangle$ będzie automatem deterministycznym. Wówczas automat niedeterministyczny $M' = \langle Q, \Sigma, \delta', \{s\}, F \rangle$, gdzie $\delta'(q, a) = \{\delta(q, a)\}$ akceptuje dokładnie ten sam język

$$L(M) = L(M')$$

Kilka własności funkcji δ^*:

Lemat: Dla dowolnych $x, y \in \Sigma^*$ mamy $\delta^*(A, xy) = \delta^*(\delta^*(A, x), y)$.

Dowód: Przez indukcję ze względu na $|y|$.

Lemat: Funkcja δ^* jest rozłączna ze względu na sumowanie zbiorów, tzn.:

$$\delta^*(\bigcup_i A_i, x) = \bigcup_i \delta^*(A_i, x)$$

Dowód: Przez indukcję ze względu na $|x|$.

3.3 Determinizacja automatu

Pokażemy, że siła wyrazu automatów niedeterministycznych jest dokładnie taka sama, jak deterministycznych. W tym celu musimy pokazać, że dla każdego automatu niedeterministycznego istnieje równoważny mu automat deterministyczny.

Zauważmy, że w trakcie symulacji działania automatu niedeterministycznego nie ma znaczenia, czy na danym polu znajdzie się jeden, czy więcej pionków. Istotny jest jedynie zbiór pól, na których w danym kroku znajdują się pionki. Nasza konstrukcja będzie polegała na zbudowaniu automatu deterministycznego, którego stanami będą *zbiory* stanów automatu niedeterministycznego.

Przykład: Nieformalny. Automat akceptujący słowa, w których przedostatni znak jest równy 1. Stany nieosiągalne nie mają znaczenia na działanie automatu, więc można ograniczyć się do 4 stanów osiągalnych, co dokładnie odpowiada automatowi zapamiętującemu dwa ostatnie znaki w słowie.

Przykład: Nieformalny. Automat akceptujący słowa o długości podzielnej przez 3 lub 5. Jedyny niedeterminizm polega na wybraniu stanu początkowego. Automat deterministyczny będzie miał 256 stanów, choć tylko 15 będzie osiągalnych.

3.3.1 Konstrukcja automatu potęgowego

Niech $M = \langle Q, \Sigma, \delta, S, F \rangle$ będzie automatem niedeterministycznym. Jego determinizacją będziemy nazywać automat deterministyczny $M' = \langle P(Q), \Sigma, \delta', S, F' \rangle$, gdzie:

$$\delta'(A, a) = \delta^*(A, a)$$

$$F' = \{A \subseteq Q : A \cap F \neq \emptyset\}$$

Pokażemy, że oba automaty akceptują te same języki.

Lemat: Dla dowolnych $A \subseteq Q$ i $x \in \Sigma^*$ zachodzi

$$\delta^*(A, x) = \delta'^*(A, x)$$

Dowód: Indukcja ze względu na $|x|$.

Korzystając z tego lematu, dowód, że $L(M) = L(M')$ jest natychmiastowy.

3.4 Automaty z ε-przejściami

Możemy jeszcze rozszerzyć pojęcie automatu niedeterministycznego, wprowadzając tzw. ε-przejścia. Są to przejścia, które automat może w każdej chwili wykonywać, nie wczytując nic z wejścia.

Przykład: Automat akceptujący $a^*b^*c^*$. Porównanie z automatem deterministycznym.

Def. 7 Automat z ε-przejściami, to taka piątka $M = \langle Q, \Sigma, \delta, S, F \rangle$, gdzie:

- Q jest skończonym zbiorem stanów,
- Σ to skończony alfabet,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$ to funkcja przejścia,
- $S \subseteq Q$ to zbiór stanów początkowych, oraz
- $F \subseteq Q$ to zbiór stanów akceptujących.

Przez $D(A)$ oznaczamy zbiór stanów, które można osiągnąć z A poprzez ε-przejścia. Rozszerzenie funkcji przejścia $\delta^* : P(Q) \times \Sigma \to P(Q)$ definiujemy następująco:

$$\delta^*(A, \varepsilon) = D(A)$$

$$\delta^*(A, ax) = \delta^*(\delta(D(A), a))$$

Język akceptowany przez automat z ε-przejściami to:

$$L(M) = \{x \in \Sigma^* : \delta^*(S, x) \cap F \neq \emptyset\}$$

□

Jak łatwo zauważyć, automaty bez ε-przejść są tylko szczególnym przypadkiem automatów z ε-przejściami. Pokażemy jednak, że ε-przejścia nie zwiększają siły wyrazu i że zawsze można je wyeliminować. Zauważmy, że z powyższej definicji δ^* wynika, że wystarczy w tym celu:

- zastąpić S przez $D(S)$,
- dla każdego $q \in Q$ i $a \in \Sigma$ zastąpić $\delta(q, a)$ przez $D(\delta(q, a))$.

Zauważmy, że:

$$D(\bigcup_i A_i) = \bigcup_i D(A_i)$$

czyli

$$\delta^*(A, xa) = \bigcup_{q \in \delta^*(A,x)} D(\delta(q, a))$$

Można pokazać, że

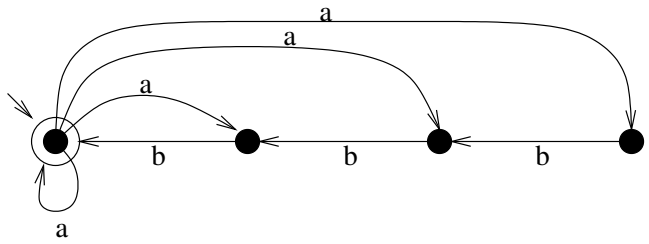
$$\delta^*(S, x) = \delta'^*(D(S), x)$$

3.5 Podsumowanie

Reasumując, siła wyrazu automatów deterministycznych, niedeterministycznych i z ε-przejściami jest dokładnie taka sama.

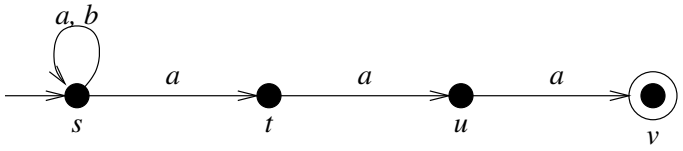
Ćwiczenia

- [K Ex. 3] Automat niedeterministyczny — jaki język akceptuje ten automat. Historyjka o rozmnażaniu się automatów, gdy możliwych jest wiele przejść i ginięciu kopii, gdy brak jest przejścia. Automat akceptuje słowo, gdy choć jedna jego kopia je zaakceptuje. Zdeterminizować poniższy automat.

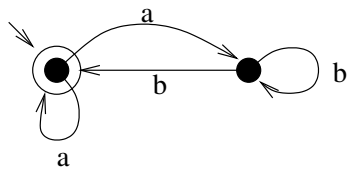


- Podaj automat niedeterministyczny akceptujący wszystkie słowa zawierające słowo *bababba*. Porównaj go z automatem deterministycznym stworzonym ręcznie i w wyniku determinizacji automatu niedeterministycznego.

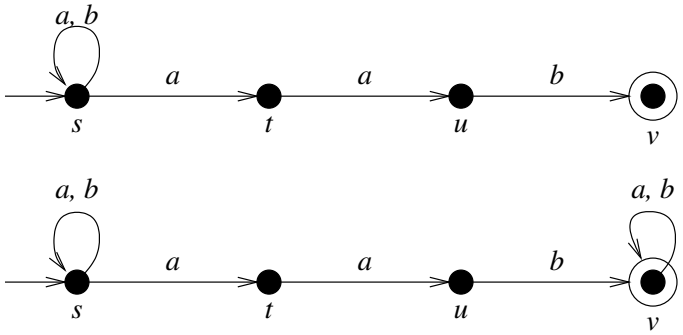
- [K Hw. 2.1] Determinizacja automatu.



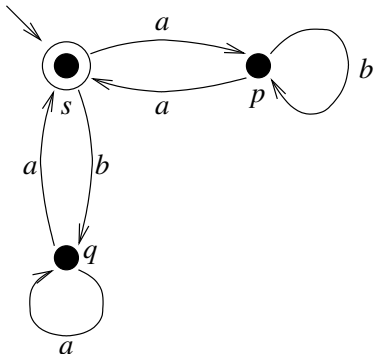
- [K Ex. 4.a] Determinizacja automatu. Pomiń w konstrukcji stany nieosiągalne.

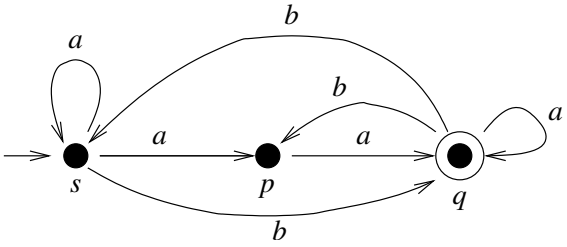


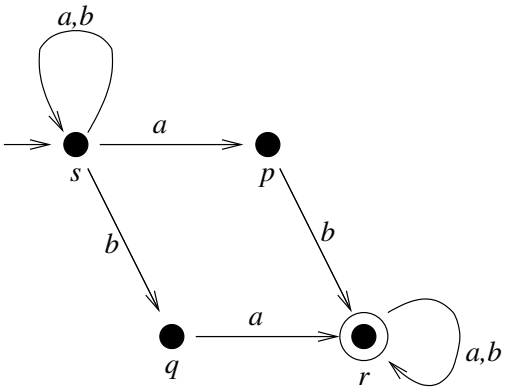
- [K Ex. 5] Determinizacja automatu. Pomiń w konstrukcji stany nieosiągalne. Podaj jakie zbiory stanów automatu niedeterministycznego odpowiadają stanom automatu deterministycznego.

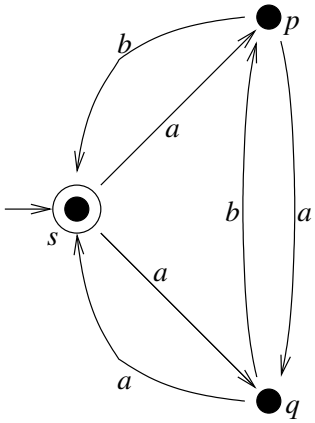


- Determinizacja automatu. Pomiń w konstrukcji stany nieosiągalne.









Wykład 4. Wyrażenia regularne i automaty

4.1 wprowadzenie

Opowiastka o wyrażeniach regularnych w Unixie. Tu rozpatrujemy dużo uboższą wersję, ale to dlatego, że nie interesuje nas praktyczna poręczność, ale teoretyczna siła wyrazu.

4.2 Wzorce

Def. 8 $\varepsilon, \emptyset, a, \alpha\beta, \alpha \cap \beta, \alpha \cup \beta, \alpha^*, \alpha^+, \overline{\alpha}$ □

Przykład:

- $(a^*bba^*) \cap (ab \cup ba)^*$,
- zapisy dziesiętne liczb naturalnych.

Definicja $L(\alpha)$.

- Jak sprawdzić, czy słowo pasuje do wzorca?
- Czy jakiekolwiek słowo pasuje do wzorca?
- Czy dla każdego języka istnieje opisujący go wzorzec? Nie.
- Czy wszystkie operacje są potrzebne? Nie.

Def. 9 Równoważność wzorców $\equiv$. □

Tożsamości dotyczące wzorców:

- $\alpha(\beta \cup \gamma) \equiv \alpha\beta \cup \alpha\gamma$,
- $(\beta \cup \gamma)\alpha \equiv \beta\alpha \cup \gamma\alpha$,
- $\emptyset\alpha \equiv \alpha\emptyset \equiv \emptyset$,
- $\alpha^+ \equiv \alpha \cup \alpha\alpha^+ \equiv \alpha \cup \alpha^+\alpha$,
- $(\alpha\beta)^*\alpha \equiv \alpha(\beta\alpha)^*$,
- $(\alpha^*)^* \equiv \alpha^*$,
- $(\alpha^*\beta^*)^* \equiv (\alpha \cup \beta)^*$.

4.3 Wyrażenia regularne

Definicja wyrażeń regularnych jako wzorców w których nie występują: $\cap$, $\overline{\alpha}$ i $^+$.

4.4 Równoważność wzorców, wyrażeń regularnych i automatów skończonych

Twierdzenie: Następujące stwierdzenia są równoważne:

a) A jest regularny, tzn. istnieje taki automat skończony M, że $A = L(M)$,

b) $A = L(\alpha)$ dla pewnego wzorca α,

c) $A = L(\alpha)$ dla pewnego wyrażenia regularnego α.

Dowód[JFA,JAO]**:** Pokażemy, że $c) \Rightarrow b)$, $b) \Rightarrow a)$ i $a) \Rightarrow c)$.

$c) \Rightarrow b)$ Oczywiste, bo każde wyrażenie regularne jest wzorcem.

$b) \Rightarrow a)$ Indukcyjnie po strukturze α budujemy automat z ε-przejściami, o jednym stanie początkowym i akceptującym, który akceptuje słowa pasujące do wzorca.

- ε, $\emptyset$, a — proste.
- Dla $\alpha = \overline{\beta}$ konstrukcja przebiega w kilku krokach:
  1. eliminacja ε-przejść i determinizacja automatu akceptującego β,
  2. dopełnienie zbioru stanów akceptujących,
  3. sprowadzenie zbioru stanów akceptujących do jednego stanu.
- $\alpha \cup \beta$, α^+, $\alpha\beta$ — rysunek,
- Dla $\alpha = \beta \cap \gamma$ i $\alpha = \beta^*$ wykorzystujemy tożsamości:

$$\beta \cap \gamma \equiv \overline{\overline{\beta} \cup \overline{\gamma}}$$

$$\beta^* \equiv \varepsilon \cup \beta^+$$

$a) \Rightarrow c)$ Budujemy wyrażenia regularne opisujące fragmenty automatu: $\alpha^X_{u,v}$ opisuje te słowa, za pomocą których można przejść z u do v zatrzymując się po drodze jedynie w stanach z X. Wyrażenie $\alpha^X_{u,v}$ definiujemy indukcyjnie ze względu na $|X|$.

1. Oznaczmy przez $a_1, \ldots, a_k$ wszystkie krawędzie prowadzące z u do v. Wówczas:

$$\alpha^{\emptyset}_{u,v} = \begin{cases} a_1 \cup \ldots \cup a_k \cup \varepsilon & ; \quad u = v \\ a_1 \cup \ldots \cup a_k & ; \quad u \neq v \end{cases}$$

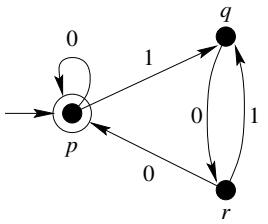
2. Niech $q \in X$.

$$\alpha^X_{u,v} = \alpha^{X-\{q\}}_{u,v} \cup \alpha^{X-\{q\}}_{u,q} (\alpha^{X-\{q\}}_{q,q})^* \alpha^{X-\{q\}}_{q,v}$$

Niech $F = \{f_1, \ldots, f_l\}$. Wówczas:

$$L(M) = L(\alpha^Q_{s,f_1} \cup \ldots \cup \alpha^Q_{s,f_l})$$

Przykład: [K 9.3/53] Przekształcenie automatu w równoważne wyrażenie regularne.



Język akceptowany przez automat jest opisany wyrażeniem.

$$\alpha_{p,p}^{\{p,q,r\}} = \alpha_{p,p}^{\{p,r\}} \cup \alpha_{p,q}^{\{p,r\}} (\alpha_{q,q}^{\{p,r\}})^* \alpha_{q,p}^{\{p,r\}}$$

Patrząc na automat możemy wywnioskować:

$$\alpha_{p,p}^{\{p,r\}} = 0^*$$

$$\alpha_{p,q}^{\{p,r\}} = 0^*1$$

$$\alpha_{q,q}^{\{p,r\}} = \varepsilon \cup 01 \cup 000^*1 = \varepsilon \cup 00^*1$$

$$\alpha_{q,p}^{\{p,q,r\}} = 000^*$$

Stąd:

$$\alpha_{p,p}^{\{p,q,r\}} = 0^* \cup 0^*1(\varepsilon \cup 00^*1)^*000^*$$

Dalej, wyrażenie to można uprościć do $\varepsilon \cup (0 \cup 10)^*0$.

4.5 Zastosowania automatów skończonych

- Analizatory leksykalne (BUK).
- Rozpoznawanie wzorców.
- Weryfikacja systemów — model checking.

4.5.1 Rozbudowana dygresja — BDD^{TO}

BDD (ang. *binary decision diagrams*) to przykład ciekawej struktury danych, która z jednej strony jest szczególnym rodzajem automatu skończonego, a z drugiej jest stosowana w weryfikacji modelowej do bardzo efektywnego reprezentowania dużych automatów.

Na BDD możemy patrzeć jak na reprezentację (semantyki) formuły zdaniowej. Powiedzmy, że w formule mogą występować zmienne zdaniowe $x_1, x_2, \ldots, x_n$. Wówczas, semantyką formuły zdaniowej jest funkcja przypisująca wartościowaniom zmiennych wartości prawda/fałsz. Każde wartościowanie zmiennych możemy reprezentować jako ciąg n wartości 0/1, czyli jako słowo nad alfabetem $\{0, 1\}$, długości n.

BDD możemy traktować jak deterministyczny automat skończony rozpoznający dokładnie te słowa długości n nad alfabetem $\{0,1\}$, które odpowiadają wartościowaniom spełniającym daną formułę. Stany BDD są ułożone „piętrami" — na najwyższym piętrze jest tylko jeden stan początkowy, a na najniższym piętrze dwa stany „prawda" i „fałsz". Z każdego stanu (poza stanami „prawda" i „fałsz") wychodzą dwa przejścia (dla 0 i 1) prowadzące o jedno piętro w dół.

Dla danej formuły zdaniowej może istnieć wiele (równoważnych) automatów reprezentujących ją. Od BDD wymagamy jednak, aby był to minimalny automat skończony. Minimalizacja BDD przebiega zgodnie z algorytmem opisanym w p. 6, choć dzięki „piętrowej" budowie BDD można go zaimplementować efektywniej. Minimalizujemy BDD od dołu, po jednym piętrze na raz.

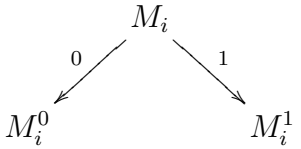
Dzięki minimalizacji, dla każdej formuły zdaniowej mamy dokładnie jeden BDD reprezentujący ją. Pozwala to na łatwe stwierdzenie, czy dwie formuły są równoważne (muszą mieć identyczne BDD), czy formuła jest tautologią (musi być równoważna prawdzie), lub czy jest spełnialna (stan „prawda" musi być osiągalny).

Konstrukcja BDD reprezentującego daną formułę zdaniową przebiega zgodnie z jej strukturą. Łatwo można zbudować BDD reprezentujące formuły *prawda*, *fałsz*, czy x_i. Jeśli mamy BDD reprezentujący formułę φ, to łatwo można z niego zrobić BDD reprezentujący $\neg\varphi$ — wystarczy zamienić rolą stany „prawda" i „fałsz" (a więc tak samo jak przy konstrukcji automatu akceptującego dopełnienie języka).

Konstrukcja BDD dla koniunkcji i alternatywy jest trudniejsza. Oczywiście można zbudować odpowiedni automat produktowy i zminimalizować go. Koszt takiego algorytmu jest dokładnie taki, jak iloczyn liczb stanów BDD reprezentujących człony alternatywy/koniunkcji. BDD wynikowy nie musi mieć jednak aż tyle stanów, może być mniejszy. Poniżej opiszemy krótko algorytm, który ma taką samą złożoność pesymistyczną, ale może też zachowywać się lepiej.

Niech $\otimes$ oznacza dowolny binarny operator boolowski (np. koniunkcję lub alternatywę). Niech M_1 i M_2 będą BDD reprezentującymi, odpowiednio, formuły φ_1 i φ_2. Szukamy BDD M' reprezentującego $\varphi_1 \otimes \varphi_2$.

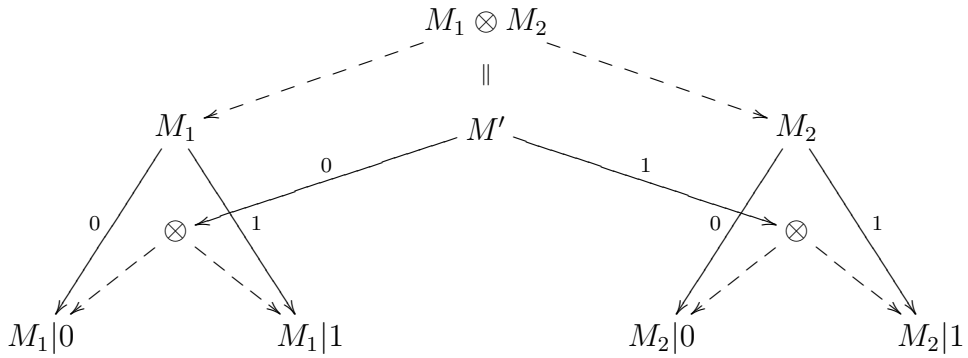
Jeśli M jest BDD, to przez $M|a$ oznaczymy BDD, którego stanem początkowym jest stan, do którego przechodzimy ze stanu początkowego M pod wpływem a. Inaczej mówiąc, $M|a$ to zawężenie M do tych wartościowań, w których $x_1 = a$. Oczywiście M_i^a ma jedno „piętro" mniej niż M_i.



Konstruując M' opieramy się na następującej tożsamości:

$$(\varphi_1 \otimes \varphi_2) \wedge (x_1 = a) \equiv (\varphi_1 \wedge (x_1 = a)) \otimes (\varphi_2 \wedge (x_1 = a))$$

czyli $M'|a = M_1|a \otimes M_2|a$. Tak więc M' można skonstruować w następujący sposób:



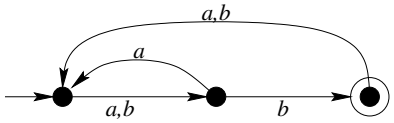
Oczywiście bezpośrednia implementacja tego pomysłu, jako procedury rekurencyjnej ma złożoność wykładniczą. Jeżeli oznaczymy przez $n_{i,k}$ liczbę wierzchołków w M_i na k-tym piętrze, to można uzyskać złożoność $\Theta(\sum_{j=1}^{n} n_{1,j} \cdot n_{2,j})$. (Patrz zadanie z *-ką.)

Ćwiczenia

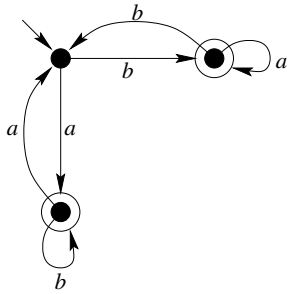
- [K Hw. 3.1] Dla poniższych języków podaj wyrażenia regularne:
  - $\{x | x$ zawiera parzystą liczbę symboli $a\}$,
  - $\{x | x$ zawiera nieparzystą liczbę symboli $b\}$,
  - $\{x | x$ zawiera parzystą liczbę symboli a lub nieparzystą liczbę symboli $b\}$,
- Podaj wyrażenia regularne opisujące język złożony ze słów:
  - nad alfabetem $\{a, b\}$, które nie zawierają podsłowa aa,
  - nad alfabetem $\{a, b, c\}$, które nie zawierają podsłowa aa,
  - nad alfabetem $\{a, b\}$, które zawierają podsłowo $bbab$,
  - nad alfabetem $\{a, b, c\}$ złożonych tylko z jednego rodzaju symboli,
  - nad alfabetem $\{a, b, c\}$ złożonych co najwyżej z dwóch rodzai symboli.
- [HU ćw.2.11, a) b)] Opisz języki reprezentowane przez następujące wyrażenia regularne:
  - $(11 \cup 0)^*(00 \cup 1)^*$,
  - $(1 \cup 01 \cup 001)^*(\varepsilon \cup 0 \cup 00)$,
- [K Ex. 11] Dla poniższych języków podaj wyrażenia regularne:
  - $\{x \in \{a, b\}^* | \, x$ nie zawiera podsłowa $a\}$,
  - $\{x \in \{a, b\}^* | \, x$ nie zawiera podsłowa $ab\}$,
- [K Ex. 12] Dopasuj automaty niedeterministyczne i wyrażenia regularne. ....
- [K Ex. 13] Dopasuj automaty niedeterministyczne i wyrażenia regularne. ....
- [K Ex. 14] Podaj automat niedeterministyczny o czterech stanach odpowiadający wyrażeniu regularnemu $(01 \cup 011 \cup 0111)^*$.
- [HU ćw.2.12] Podaj automaty skończone odpowiadające następującym wyrażeniom regularnym:
  - $10 \cup (0 \cup 11)0^*1$,
  - $01[((10)^* \cup 111)^* \cup 0]^*1$,
  - $((0 \cup 1)(0 \cup 1))^* \cup ((0 \cup 1)(0 \cup 1)(0 \cup 1))^*$.
- [K Ex. 16] Podaj automat deterministyczny odpowiadający następującym wyrażeniom regularnym:

- $(00 \cup 11)^*(01 \cup 10)(00 \cup 11)^*$,
- $(000)^* \cup (00)^*1$,
- $(0(01)^*(1 \cup 00) \cup 1(10)^*(0 \cup 11))^*$.

- [K Ex. 15] Podaj wyrażenie regularne odpowiadające automatowi niedeterministycznemu.



- [K Ex. 17] Podaj wyrażenie regularne odpowiadające automatowi deterministycznemu.



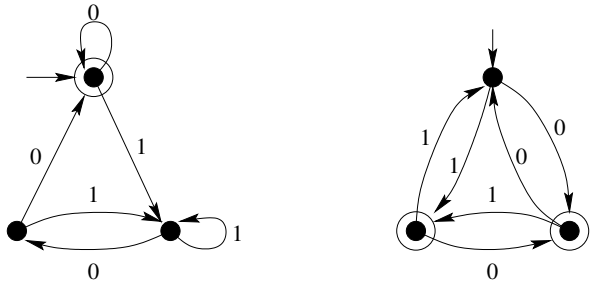
- [K Ex. 18] Porównaj wyrażenia regularne pod kątem równości/zawierania się odpowiadających im języków. Odpowiedzi uzasadnij. Udowodnij pomocnicze równanie $a(ba)^* = (ab)^*a$.

1)	$(0 \cup 1)^*$	$0^* \cup 1^*$,
2)	$0(120)^*12$	$01(201)^*2$,
3)	$\emptyset^*$	ε^*,
4)	$(0^*1^*)^*$	$(0^*1)^*$,
5)	$(01 \cup 0)^*0$	$0(10 \cup 0)^*$.

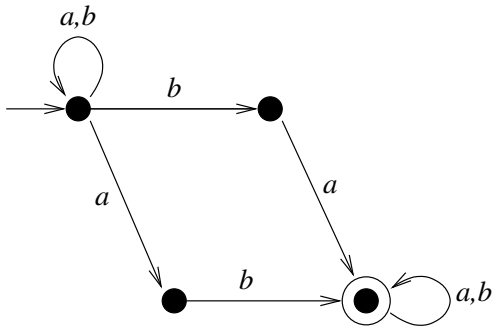
- [K Ex. 19] Oznaczmy $\alpha = (a \cup b)^*ab(a \cup b)^*$. Podaj wyrażenie regularne odpowiadające wzorcowi $\overline{\alpha}$, zakładając, że:

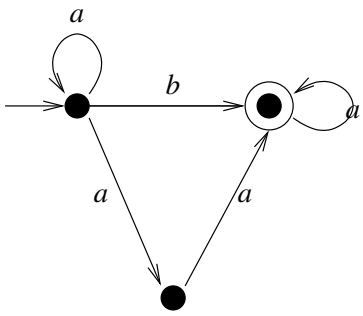
  1. $\Sigma = \{a, b\}$,
  2. $\Sigma = \{a, b, c\}$.

- [HU ćw. 2.13] Podaj wyrażenia regularne odpowiadające automatom:



- Podaj wyrażenia regularne odpowiadające automatom:





Wykład 5. Lemat o pompowaniu dla języków regularnych[JFA,JAO]

Nie każdy język jest regularny. Standardowy przykład to $A = \{a^n b^n : n \geq 0\}$.

Dowód (niewprost): Niech $M = (Q, \Sigma, \delta, s, F)$ będzie automatem akceptującym A i niech $n = |Q|$. Wówczas z zasady szufladkowej Dirichleta istnieje taki stan q, $u \geq 0$ oraz $v > 0$, że $n = u + v$, $\delta^*(s, a^u) = \delta^*(q, a^v) = q$. Wówczas tak samo jest akceptowane, lub nie, słowo $a^u b^n$ i $a^n b^n$. Sprzeczność.

Lemat o pompowaniu ($r \Rightarrow p$) Niech A będzie językiem regularnym. Wówczas istnieje takie k, że dla dowolnych słów x, y i z takich, że $xyz \in A$ oraz $|y| \geq k$ istnieją takie słowa u, v, w, że $y = uvw$, $v \neq \varepsilon$ i dla wszystkich $i \geq 0$ $xuv^i wz \in A$.

Dowód

Lemat o pompowaniu — sformułowanie odwrotne ($\neg p \Rightarrow \neg r$) Niech A będzie językiem. Przypuśćmy, że dla wszystkich $k \geq 0$ istnieją takie słowa x, y, z, że $xyz \in A$, $|y| \geq k$ oraz dla dowolnych u, v, w takich, że $y = uvw$, $v \neq \varepsilon$, istnieje takie $i \geq 0$, że $xuv^i wz \notin A$, wówczas A nie jest regularny.

Intuicja — gra ze złośliwym Demonem. Demon stara się pokazać, że język mimo wszystko jest regularny.

1. Demon wybiera k. (Jeśli A jest faktycznie regularny, to Demon może wybrać za k liczbę stanów automatu akceptującego A).
2. Wybierasz takie x, y, z, że $xyz \in A$ i $|y| \geq k$.
3. Demon dzieli y na takie u, v, w, że $y = uvw$, $v \neq \varepsilon$.
4. Wybierasz $i \geq 0$.

Jeśli $xuv^i wz \notin A$, to wygrałeś, wpp. wygrał Demon. Własność opisana w odwrotnym sformułowaniu lematu o pompowaniu odpowiada istnieniu strategii wygrywającej.

Zastosowanie lematu o pompowaniu do języka $A = \{a^n b^n : n \leq 0\}$.

Przykład: $C = \{a^{n!} : n \geq 0\}$. Demon wybiera k. Wybieramy $x = z = \varepsilon$ i $y = a^{k!}$. Powiedzmy, że Demon wybiera u, v, w długości odpowiednio j, m, n. Musimy teraz wybrać takie i, żeby $xuv^i wz \notin C$. Zauważmy, że $|xuv^i wz| = j + im + n = k! + (i-1)m$. Niech $i = (k+1)! + 1$. Wówczas $|xuv^i wz| = k! + (k+1)!m = k!(1 + (k+1)m)$. Gdyby dla pewnego p zachodziło $k!(1 + (k+1)m) = p!$, to $p(p-1)\cdots(k+1) = 1 + (k+1)m$. Jest to jednak niemożliwe, bo lewa strona jest podzielna przez $k+1$, a prawa nie. Tak więc wygraliśmy, czyli C nie jest regularny.

5.1 Jak sobie ułatwić dowodzenie

Jeśli chcemy pokazać, że jakiś język nie jest regularny, to możemy skorzystać z własności domknięcia klasy języków regularnych. Rozumujemy przy tym niewprost — gdyby język A był regularny, to B też, a że B nie jest, to i A nie może być.

Przykład: Rozważmy $D = \{x \in \{a,b\}^* : \#_a(x) = \#_b(x)\}$. Gdyby był on regularny, to $D \cap a^*b^* = \{a^n b^n : n \geq 0\}$ też byłby regularny, a nie jest.

Przykład: Język $E = \{a^n b^m : n \geq m\}$. Dowód przez przecięcie z rev E.

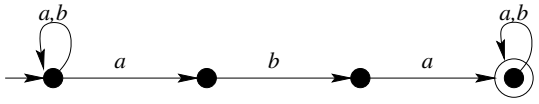
Ćwiczenia

- [K Hw. 4.1] Pokaż, że dane języki nie są regularne:
  1. $\{a^n b^m : n = 2m\}$,
  2. $\{x \in \{a, b, c\}^* : x$ jest palindromem, tzn. $x = \mathrm{rev}(x)\}$,
  3. $\{x \in \{a, b, c\}^* :$ długość słowa x jest kwadratem $\}$,
  4. poprawne wyrażenia nawiasowe.
- [K Ex. 35] Pokaż, że język $\{ww : w \in \{0, 1\}^*\}$ nie jest regularny.
- Przykład języka regularnego i jak lemat o pompowaniu jest dla niego spełniony $((0 \cup 1)^*)$.
- [K Ex. 37] Które z podanych języków są regularne, a które nie? Odpowiedzi uzasadnij.
  1. $\{a^n b^{2m} : n \geq 0 \wedge m \geq 0\}$,
  2. $\{a^n b^m : n \neq m\}$,
  3. $\{a^{p-1} : p$ jest liczbą pierwszą $\}$,
  4. $\{xcx : x \in \{a, b\}^*\}$,
  5. $\{xcy : x, y \in \{a, b\}^*\}$,
  6. $\{a^n b^{n+42} : n \geq 0\}$,
  7. $\{a^n b^m : n - m \leq 42\}$,
  8. $\{a^n b^m : n \geq m \wedge m \leq 42\}$,
  9. $\{a^n b^m : n \geq m \geq 42\}$,
  10. $L((a^*b)^*a^*)$,
  11. $\{a^n b^n c^n : n \geq 0\}$,
  12. składniowo poprawne programy w Pascalu.
- [K Ex. 38] Które z podanych języków są regularne, a które nie? Odpowiedzi uzasadnij.
  1. $\{x : \#_1(x) = 2 \cdot \#_0(x)\}$,
  2. $\{x : \#_1(x) - \#_0(x) < 42\}$,
  3. $\{x : \#_1(x) \cdot \#_0(x)$ jest liczbą parzystą $\}$.
- [HU ćw. 3.1] Które z podanych języków są regularne, a które nie? Odpowiedzi uzasadnij.
  1. $\{a^{2n} : n \geq 1\}$,
  2. $\{0^m 1^n 0^{m+n} : m, n \geq 1\}$,

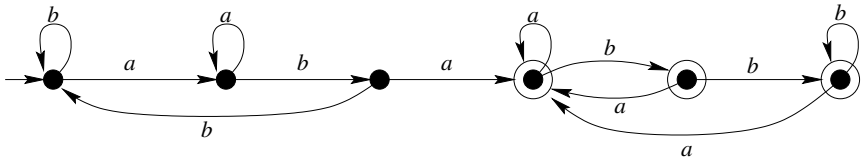
3. zbiór wszystkich napisów nad alfabetem $\{0,1\}$ nie zawierających trzech zer z rzędu,
4. $\{xw\text{rev}(x) : x, w \in \{0,1\}^*\}$.

Wykład 6. Minimalizacja deterministycznych automatów skończonych[JFA,TO]

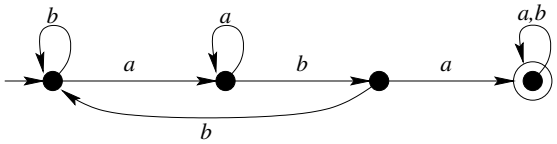
Przykład: [K str. 77] Determinizacja automatu akceptującego słowa zawierające *aba*.



Można usunąć stany nieosiągalne, pozostaje wówczas:



Dalej można skleić trzy stany akceptowalne, gdyż po ich osiągnięciu i tak słowo zostanie zaakceptowane.



W tym przypadku rozwiązanie jest oczywiste, a w ogólności?

6.1 Dowód istnienia automatu minimalnego

- Definicja języków ilorazowych $A_{/x}$.
- Związek między językami ilorazowymi, a stanami deterministycznego automatu skończonego akceptującego dany język, oraz relacją przejścia.
- Tw. Myhill-Neroda.
- Związek między liczbą języków ilorazowych, a liczbą stanów automatu akceptującego dany język.
- Konstrukcja minimalnego automatu akceptującego.
- Obserwacja: Automat minimalny można uzyskać przez usunięcie stanów nieosiągalnych i sklejenie stanów odopwiadających takim samym językom ilorazowym.

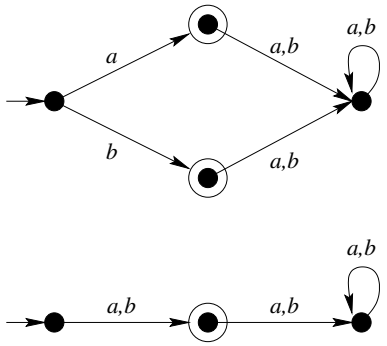
6.2 Konstrukcja automatu minimalnego

Algorytm minimalizacji składa się z dwóch faz:

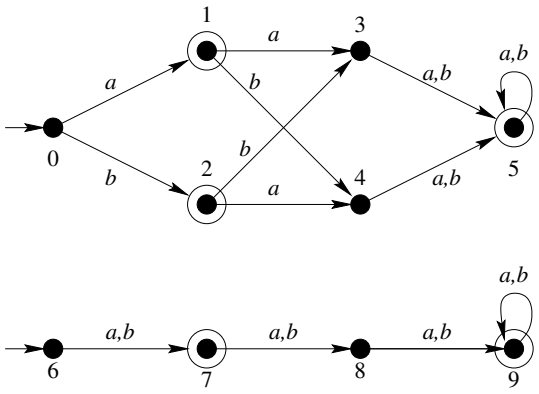
1. usuń stany nieosiągalne,
2. posklejaj ze sobą „równoważne" stany.

To, które stany są osiągalne jest oczywiste. Zobaczmy na przykładach, które stany należy sklejać.

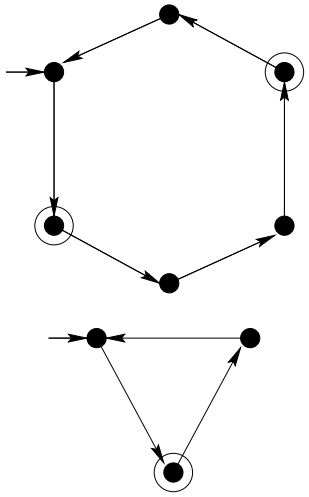
Przykład: [K Example 13.1] Sklejenie równoważnych stanów akceptujących.



Przykład: [K Example 13.2]



Przykład:



Kiedy możemy skleić stany? Jeśli istnieje rozróżniające je słowo, to nie, a jeśli dla każdego słowa wynik jest taki sam, to tak.

Def. 10 Niech $M = \langle Q, \Sigma, \delta, s, F \rangle$ będzie deterministycznym automatem skończonym. Na stanach M definiujemy relację $\approx \subseteq Q \times Q$:

$$p \approx q \text{ w.t.w. } \forall_{x \in \Sigma^*} \delta^*(p, x) \in F \Leftrightarrow \delta^*(q, x) \in F$$

□

Def. 11 Automat ilorazowy $M_{/\approx} = \langle Q_{/\approx}, \Sigma, \delta_{/\approx}, [s]_\approx, F_{/\approx} \rangle$. □

Dowód poprawności tej definicji.

Lemat $\delta'^*([p]_\approx, x) = [\delta^*(p, x)]_\approx$

Tw. $L(M_{/\approx}) = L(M)$

6.3 Algorytm minimalizacji

Algorytm minimalizacji automatu o wszystkich stanach osiągalnych. Badamy, których stanów nie możemy skleić ze sobą i gdy tylko znajdziemy powód ku temu, zaznaczamy to.

1. Tworzymy tabelę, początkowo pustą, wszystkich par stanów $\{p, q\}$.
2. Zaznaczamy wszystkie takie pary $\{p, q\}$, że $p \in F$ i $q \notin F$.
3. Dopóki można, to: jeśli taka niezaznaczona para $\{p, q\}$, że dla pewnego a mamy $\{\delta(p, a), \delta(q, a)\}$ jest zaznaczona, to zaznacz również $\{p, q\}$.

4. $p \approx q$ gdy para $\{p, q\}$ nie jest zaznaczona.

Szkic dowodu poprawności:

- Te stany, które zaznaczamy, że nie są równoważne, istotnie nie są sobie równoważne.
- Jeśli jakieś stany nie są sobie równoważne, to zostaną zaznaczone.

Przykład: [K Example 14.1] Minimalizacja automatu:

		a	b
$\rightarrow$	0	1	2
F	1	3	4
F	2	4	3
	3	5	5
	4	5	5
F	5	5	5

Przykład: [K Example 14.2] Minimalizacja automatu

		a
$\rightarrow$	0	1
F	1	2
	2	3
	3	4
F	4	5
	5	0

Ćwiczenia

- [K HW. 4.3] Zminimalizuj następujące automaty:

		a	b
$\rightarrow$	1	1	4
	2	3	1
F	3	4	2
F	4	3	5
	5	4	6
	6	6	3
	7	2	4
	8	3	1

		a	b
$\rightarrow F$	1	3	5
F	2	8	7
	3	7	2
	4	6	2
	5	1	8
	6	2	3
	7	1	4
	8	5	1

- [K Ex. 47] Zminimalizuj następujące automaty:

		a	b
$\rightarrow$	1	6	3
	2	5	6
F	3	4	5
F	4	3	2
	5	2	1
	6	1	4

		a	b
$\rightarrow$	1	2	3
	2	5	6
F	3	1	4
F	4	6	3
	5	2	1
	6	5	4

		a	b
F	0	3	2
F	1	3	5
	2	2	6
	3	2	1
	4	5	4
	5	5	3
	6	5	0

		a	b
$\rightarrow$	0	3	5
	1	2	4
	2	6	2
	3	6	6
F	4	0	2
F	5	1	6
	6	2	6

		a	b
$\rightarrow$	0	3	5
	1	6	3
	2	6	4
	3	6	6
F	4	0	5
F	5	2	4
	6	1	6

		a	b
$\rightarrow$	0	2	5
	1	6	2
	2	6	6
	3	6	4
F	4	5	0
F	5	4	3
	6	1	6

		a	b
$\rightarrow F$	1	6	4
F	2	7	5
	3	2	8
	4	1	8
	5	2	6
	6	3	1
	7	5	2
	8	4	2

		a	b
$\rightarrow$	1	6	2
	2	3	6
F	3	2	4
F	4	5	3
	5	4	1
	6	1	5
	7	1	8
	8	8	7

- [K Ex. 52] Zminimalizuj następujące automaty. Podaj dla nich równoważne wyrażenia regularne.

		a	b
$\rightarrow F$	1	2	5
F	2	1	4
	3	7	2
	4	5	7
	5	4	3
	6	3	6
	7	3	1

		a	b
$\rightarrow F$	1	2	6
F	2	1	7
	3	5	2
	4	2	3
	5	3	1
	6	7	3
	7	6	5

		a	b
$\rightarrow$	1	1	3
F	2	6	3
	3	5	7
F	4	6	1
	5	1	7
F	6	2	7
	7	5	3

		a	b
$\rightarrow F$	1	2	5
F	2	1	6
	3	4	3
	4	7	1
	5	6	7
	6	5	4
	7	4	2

Wykład 7. Języki bezkontekstowe

7.1 Intuicje

7.1.1 BNF

Opisując składnię języka programowania korzystamy czasem z notacji BNF. Opis notacji:

- ⟨`konstrukcja`⟩ ,
- ::=,
- `słowo kluczowe` ,
- $\ldots|\ldots$,
- $[\ldots]$,
- $\{\ldots\}^*$,
- $\{\ldots\}^+$,
- $\{\ldots\}$.

Przykład: Fragment składni Pascala, np. instrukcje.
Nieformalne wprowadzenie podstawowych pojęć:

- gramatyka bezkontekstowa,
- nieterminale,
- terminale,
- wyprowadzenie,
- drzewo wyprowadzenia.

7.1.2 Automaty stosowe

Opis, odpowiednik obliczeniowy gramatyk.

7.2 Gramatyki bezkontekstowe

Def. 12

- Gramatyka $G = \langle N, \Sigma, P, S \in N \rangle$.
- Produkcje i postać ich zapisu, | jako skrót.
- Oznaczenia A, B, C — nieterminale, a, b, c — terminale, α, β, γ — słowa nad $N \cup \Sigma$.
- Relacje $\rightarrow$ i $\rightarrow^*$.

□

Przykład: Gramatyka opisująca język: $\{a^n b^n : n \geq 0\}$.

Def. 13

- Wyprowadzenie. Fakt: α jest wyprowadzalna w gramatyce wtw., gdy $S \rightarrow^* \alpha$ i $\alpha \in \Sigma^*$.
- $L(G)$.

□

Przykład: Przykłady gramatyk:

- Przykłady wyprowadzenia i drzewa wyprowadzenia dla języka $\{a^n b^n : n \geq 0\}$.
- Palindromy nad alfabetem $\{a, b, c\}$.
- – Wyrażenia nawiasowe.
  – Co to są poprawne wyrażenia nawiasowe? $L(x) = \#_($(x), R(x) = \#_)(x)$, $L(x) = R(x)$ i dla każdego prefiksu y słowa x mamy $L(y) \geq R(y)$.
  – Dowód poprawności.
- Wyrażenia arytmetyczne i prosta gramatyka. Drzewo wyprowadzenia i niejednoznaczność.
- Wyrażenia arytmetyczne i gramatyka jednoznaczna.

Def. 14 Jednoznaczność gramatyki bezkontekstowej. □

7.3 Języki regularne a bezkontekstowe

Wszystkie regularne są bezkontekstowe, ale nie odwrotnie, np. $a^n b^n$, palindromy i wyrażenia nawiasowe tworzą języki, które nie są regularne, ale są bezkontekstowe.

Def. 15

- Gramatyka (prawostronnie) liniowa, to taka, w której wszystkie produkcje mają postać: $A \to \alpha B$ lub $A \to \alpha$, dla $\alpha \in \Sigma^*$.
- Gramatyka silnie (prawostronnie) linowa, to taka, w której wszystkie produkcje mają postać: $A \to aB$ lub $A \to \varepsilon$.

□

Zauważmy, że jeśli gramatyka jest liniowa lub silnie liniowa, to w wyprowadzeniu pojawia się na raz tylko jeden nieterminal, i to na samym końcu słowa.

Fakt Gramatyki (silnie) liniowe opisują dokładnie języki regularne.

Szkic dowodu:

1. Gramatykę liniową można sprowadzić do silnie liniowej.
2. Gramatykę silnie liniową można przerobić na automat niedeterministyczny.
3. Automat niedeterministyczny można przerobić na gramatykę silnie liniową.

Ćwiczenia

- Dla zadanego automatu/wyrażenia regularnego podaj gramatyki (prawostronnie) [silnie] liniowe opisujące odpowiedni język:
  - $a(a \cup b)^* aa$,
  - $(a(ab \cup ba)^*) \cup (b(bb \cup aa)^*)$,
  -

	0	1
$\rightarrow 1$	1	2
$F2$	1	3
$F3$	3	1

  -

	a	b
$\rightarrow 1$	2	–
$F2$	2	1

- [K Ex. 69] Dana jest gramatyka bezkontekstowa:

$$\begin{array}{lcl} S & \rightarrow & ABS \,|\, AB \\ A & \rightarrow & aA \,|\, a \\ B & \rightarrow & bA \end{array}$$

  Które ze słow: *aabaab*, *aaaaba*, *aabbaa*, *abaaba* należą do języka generowanego przez tę gramatykę? Podaj wyprowadzenia i drzewa wyprowadzenia.

- Dla gramatyki:

$$\begin{array}{lcl} S & \rightarrow & aB|Ab|SS \\ A & \rightarrow & a|aS \\ B & \rightarrow & b|Sb \end{array}$$

  i słowa *aabbab*:
  - podaj wyprowadzenie,
  - podaj drzewo wyprowadzenia,
  - czy jest to gramatyka jednoznaczna?

- Podaj gramatykę generującą język:
  - $\{a^n b^m : n = 2 * m\}$,
  - $\{a^n b^m : n \geq 2 * m\}$,
  - $\{a^i b^j c^i\}$,
  - $\{a^i b^j a^j b^i\}$.

- [K Hw. 5.3] Podaj gramatykę wyrażeń nawiasowych []({}), w której sparowane nawiasy są tego samego rodzaju. Wzmocnij ją tak, aby wymusić priorytety nawiasów (wewnątrz kwadratowych muszą być kwadratowe lub okrągłe, a wewnątrz wąsatych muszą być wąsate lub kwadratowe).
- [HU ćw. 4.1.f] Podaj gramatykę opisującą język: $\{a^i b^j c^k : i \neq j \vee j \neq k\}$
- [K Ex. 75, HU ćw. 4.1.d] Podaj gramatykę wyrażeń regularnych.
- Podaj gramatykę bezkontekstową generującą język:
  - $\{a^n b^{2n} : n \geq 1\}$,
  - $\{a^k b^n : k \geq 2n\}$,
  - $\{a^n b^k : 2n \geq 3k\}$.

Wykład 8. Postać normalna Chomsky'ego, algorytm rozpoznawania CKY i lemat o pompowaniu dla języków bezkontekstowych

8.1 Postać normalna Chomsky'ego

Def. 16 Gramatyka ma *postać normalną Chomsky'ego*, jeśli wszystkie produkcje w gramatyce są postaci: $A \to BC$ lub $A \to a$ (dla pewnych nieterminali A, B i C oraz terminala a). □

Fakt: Jeśli G jest w postaci normalnej Chomsky'ego, to $\varepsilon \notin L(G)$.

Fakt: Jeśli $\varepsilon \notin L(G)$, to gramatykę G można sprowadzić do postaci normalnej Chomsky'ego, tzn. istnieje taka gramatyka bezkontekstowa G' w postaci normalnej Chomsky'ego, że $L(G) = L(G')$.

Szkic dowodu: Dowód polega na skonstruowaniu gramatyki G'. Konstrukcja ta przebiega w kilku krokach:

1. Domykamy zbiór produkcji zgodnie z dwiema regułami:

   - jeśli $X \to \alpha Y \beta$ i $Y \to \varepsilon$, to $X \to \alpha\beta$,
   - jeśli $X \to Y$ i $Y \to \gamma$, to $X \to \gamma$.

   Tak powstały zbiór produkcji jest skończony, bo dodajemy produkcje, które nie są dłuższe od już istniejących.

2. Dla dowolnego słowa wyprowadzalnego w takiej gramatyce, jeżeli rozważymy jego najkrótsze wyprowadzenie, to w wyprowadzeniu tym nie będą się pojawiały produkcje postaci $X \to \varepsilon$ oraz $X \to Y$. Możemy więc, bez zmiany języka generowanego przez gramatykę, usunąć te produkcje. Wyjątek to $S \to \varepsilon$ — usunięcie tej produkcji spowoduje, że ε nie będzie wyprowadzalny.

3. Dla każdego terminala $(a, b, c, \ldots)$ dodajemy do gramatyki odpowiadający mu nieterminal $(A, B, C, \ldots)$ oraz produkcje: $A \to a$, $B \to b$, $C \to c$, ... We wszystkich produkcjach postaci $X \to \alpha$, gdzie $|\alpha| > 1$ zamieniamy wszystkie terminale na odpowiadające im nieterminale. W ten sposób mamy tylko dwie postaci produkcji w gramatyce: $X \to a$ gdzie a jest terminalem, oraz $X \to Y_1 Y_2 \ldots Y_k$ gdzie Y_i to nieterminale i $k > 1$.

4. Każdą produkcję postaci $X \to Y_1 Y_2 \ldots Y_k$ $(k > 2)$ zamieniamy na zestaw produkcji:

$$X \to Y_1 X_1$$

$$X_1 \to Y_2 X_2$$
$$\vdots$$
$$X_{k-2} \to Y_{k-1} Y_k$$

Tak uzyskana gramatyka jest w postaci normalnej Chomsky'ego.

Przykład:

- $a^n b^n$,
- wyrażenia nawiasowe.

8.2 Algorytm rozpoznawania Cocke-Kasami-Younger'a[JFA, TO]

Algorytm ten pozwala stwierdzić, czy dane słowo jest wyprowadzalne w danej gramatyce (w postaci normalnej Chomsky'ego). Jeśli gramatyka nie jest w postaci normalnej Chomsky'ego, to można ją sprowadzić do tej postaci zgodnie z opisaną wyżej konstrukcją.

Przypomnienie techniki programowania dynamicznego. Niech dana gramatyka to $G = \langle N, \Sigma, P, S \rangle$, a dane słowo to $a_1 a_2 \dots a_n$. Konstruujemy tablicę

$$T[i,j] = \{X \in N : X \to^* a_i \dots a_j\}$$

W jaki sposób możemy wypełniać tę tablicę?

- Zaczynamy od przekątnej:
$$T[i,i] = \{X : X \to a_i \in P\}$$
- Wypełniamy kolejne przyprzekątne. Zauważmy, że jeżeli $X \to^* a_i \dots a_j$, $i < j$, to wyprowadzenie to musi mieć postać: $X \to YZ$, $Y \to^* a_i \dots a_k$, $Z \to^* a_{k+1} \dots a_j$, dla $i \le k < j$. Obliczając $T[i,j]$ uwzględniamy wszystkie takie możliwe punkty podziału k:
$$T[i,j] = \bigcup_{i \le k < j} T[i,k] \otimes T[k+1,j]$$
gdzie:
$$Y \otimes Z = \{X : X \to BC \in P, B \in Y, C \in Z\}$$
Operacja $\otimes$ może być skomplikowana, ale jej koszt jest stały (zależny od gramatyki).
- Po wypełnieniu całej tablicy mamy:
$$T[1,n] = \{X \in N : X \to^* a_1 \dots a_n\}$$
czyli:
$$a_1 \dots a_n \in L(G) \Leftrightarrow S \in T[1,n]$$

Tablica ma rozmiar $\Theta(n^2)$ przy czym aby wypełnić jeden element tablicy trzeba wykonać liniową liczbę operacji $\otimes$, tak więc złożoność tego algorytmu to $\Theta(n^3)$.

Przykład: ....

8.3 Lemat o pompowaniu dla języków bezkontekstowych[JAO]

Lemat: Niech A będzie językiem bezkontekstowym. Istnieje wówczas taka stała $k \geq 0$, że dla każdego słowa $z \in A$ o długości $|z| \geq k$ można słowo z podzielić na pięć części: $z = uvwxy$ w taki sposób, że $vx \neq \varepsilon$, $|vwx| \leq k$ oraz dla każdego $i \geq 0$ mamy $uv^iwx^iy \in A$.

Szkic dowodu Jeśli język jest bezkontekstowy, to istnieje odpowiadająca mu gramatyka bezkontekstowa w postaci normalnej Chomsky'ego. Drzewa wyprowadzeń dla gramatyk w postaci normalnej Chomsky'ego mają postać regularnych drzew binarnych z „frędzelkami" długości 1 (produkcje, z których otrzymujemy terminale). Fakt: Jeśli drzewo wyprowadzenia ma wysokość h, to długość wyprowadzanego słowa nie przekracza 2^{h-2}. Stąd, jeżeli słowo ma długość większą niż 2^k, to drzewo wyprowadzenia ma wysokość przynajmniej $k+3$. W takim drzewie wyprowadzenia istnieje ścieżka, na której jest przynajmniej $k+2$ nieterminali. Jeśli $|N| = n$, to dla słów o długości $\geq 2^n$, w ich drzewach wyprowadzenia istnieją ścieżki zawierające przynajmniej $n+1$ nieterminali, co oznacza, że pewien nieterminal musi się powtarzać. Wybieramy pierwsze powtórzenie idąc od dołu ścieżki. Wyznacza ono szukany podział słowa.

Sformułowanie alternatywne Prowadzimy grę z Demonem. Demon stara się pokazać, że język jest regularny, a my że nie. Gra składa się z 4 ruchów:

- Demon wybiera stałą $k \geq 0$. Jeśli język jest bezkontekstowy, to może to być $2^{|N|}$.
- Wybieramy słowo $z \in A$, $|z| \geq k$.
- Demon dzieli je na pięć takich części $z = uvwxy$, że $vx \neq \varepsilon$, $|vwx| \leq k$. Może to być podział wyznaczony przez powtarzający się nieterminal na ścieżce do korzenia.
- Wybieramy $i \geq 0$. Jeśli $uv^iwx^iy \notin A$, to wygraliśmy, wpp. wygrał Demon.

Przykład: Dowód, że język $A = \{a^nb^nc^n\}$ nie jest bezkontekstowy.

Przykład: Dowód, że język $A = \{ww : w \in \{a,b\}^*\}$ nie jest bezkontekstowy. (Korzystamy z faktu, że przecięcie bezkontekstowego z regularnym jest bezkontekstowe.)

Ćwiczenia

- [K H.W. 6.4] Podaj gramatykę w postaci normalnej Chomsky'ego, opisującą niepuste wyrażenia nawiasowe, z trzema rodzajami nawiasów (), [] i {} (ale bez ε).
- [K Ex. 71] Sprowadź następującą gramatykę do postaci normalnej Chomsky'ego:

$$\begin{array}{lcl} S & \rightarrow & aSBB|T \\ t & \rightarrow & bTaa|S|\varepsilon \end{array}$$

- [K EX. 72] Podaj gramatyki w postaci normalnej Chomsky'ego dla następujących języków:
  - $\{a^n b^{2n} c^k : k, n \geq 1\}$,
  - $\{a^n b^k a^n : k, n \geq 1\}$,
  - $\{a^k b^m c^n : k, m, n \geq 1 \wedge 2k \geq n\}$,
  - palindromy nad alfabetem $\{a, b\}$.
- [K Ex. 91]^{JFA,TO} Zasymuluj algorytm CKY dla gramatyki:

$$\begin{array}{lcl} S & \rightarrow & AB \\ A & \rightarrow & a \\ B & \rightarrow & AB|b \end{array}$$

  i słowa aab.
- [K Ex. 92]^{JFA,TO} Jak rozszerzyć algorytm CKY tak, aby obliczał również liczbę różnych drzew wyprowadzeń. Zasymuluj tak rozszerzony algorytm CKY dla gramatyki:

$$\begin{array}{lcl} S & \rightarrow & ST|a \\ T & \rightarrow & BS \\ B & \rightarrow & + \end{array}$$

  i słowa $a + a + a + a$. Spodziewany wynik: 5.
- [K Ex. 84]^{JAO} Które z poniższych języków są bezkontekstowe, a które nie? Odpowiedź uzasadnij.
  - $\{a^n b^m c^k : n, m, k \geq 1 \wedge (2n = 3k \vee 5k = 7m)\}$,
  - $\{a^n b^m c^k : n, m, k \geq 1 \wedge (2n = 3k \wedge 5k = 7m)\}$,
  - $\{a^n b^m c^k : n, m, k \geq 1 \wedge (n \neq 3m \vee n \neq 5k)\}$,
  - $\{a^n b^m c^k : n, m, k \geq 1 \wedge (n \neq 3m \wedge n \neq 5k)\}$,
  - $\{a^n b^m c^k : n, m, k \geq 1 \wedge n + k = m\}$,
  - $\{a^i b^j c^k d^l : i, j, k, l \geq 1 \wedge i = j \wedge k = l\}$,

- $\{a^i b^j c^k d^l : i, j, k, l \geq 1 \wedge i = k \wedge j = l\}$,
- $\{a^i b^j c^k d^l : i, j, k, l \geq 1 \wedge i = l \wedge j = k\}$.

- [K Ex. 85]^{JAO} Dla każdego z poniższych języków określ, czy jest on: (i) regularny, (ii) bezkontekstowy, ale nie regularny, (iii) nie jest bezkontekstowy:
  - $\{x \in \{a, b, c\}^* : \#_a(x) = \#_b(x) = \#_c(x)\}$,
  - $\{a^j : j$ jest potęgą 2-ki$\}$,
  - $\{x \in \{0, 1\}^* : x$ jest zapisem binarnym potęgi 2-ki$\}$,
  - $L(a^* b^* c^*)$,
  - wyrażenia nawiasowe złożone ze sparowanych nawiasów trzech rodzajów: (), [] i {},
  - $\{a^n b^m : n \neq m\}$,
  - $\{a^n b^m c^k d^l : 2n = 3k \vee 5m = 7l\}$,
  - $\{a^n b^m c^k d^l : 2n = 3k \wedge 5m = 7l\}$,
  - $\{a^n b^m c^k d^l : 2n = 3m \wedge 5k = 7l\}$,
  - $\{a^n b^m c^k d^l : 2n = 3l \wedge 5k = 7m\}$,
  - $\{a^i b^j c^k : i, j, k \geq 0 \wedge i > j \wedge j > k\}$,
  - $\{a^i b^j c^k : i, j, k \geq 0 \wedge (i > j \vee j > k\}$,
  - $\{x \in \{a, b\}^* : \#_a(x) > \#_b(x)\}$,
  - $\{a^m b^n : m, n \geq 0 \wedge 5m + 3n = 24\}$,
  - $\{a^m b^n : m, n \geq 0 \wedge 5m - 3n = 24\}$.

Wykład 9. Automaty stosowe

9.1 Intuicje

Intuicyjne przedstawienie automatu stosowego.

- Konstrukcja automatu.
- Sposób działania. Konfiguracja.
- Akceptacja pustym stosem.

Przykład: Nieformalny. Tyle samo liter a, co b. Przykładowe obliczenie dla $aabbba$.

9.2 Automaty stosowe

Def. 17 Automat stosowy $M = \langle Q, \Sigma, \Gamma, \delta, s, \bot\ [, F]\rangle$, gdzie:

- Q — skończony zbiór stanów,
- Σ — alfabet wejściowy,
- Γ — alfabet stosowy,
- $\delta \subseteq (Q \times \Gamma \times (\Sigma \cup \{\varepsilon\})) \times (Q \times \Gamma^*)$ funkcja przejścia,
- $s \in Q$ stan początkowy,
- $\bot \in \Gamma$ symbol początkowy na stosie,
- $F \subseteq Q$ zbiór stanów akceptujących (opcjonalny).

□

Notacja: Zamiast pisać $((p, a, A), (q, \gamma)) \in \delta$ piszemy $(p, A) \xrightarrow{a} (q, \gamma)$ $[\in \delta]$.

Def. 18 Konfiguracje maszyny stosowej — elementy $Q \times \Sigma^* \times \Gamma^*$. Konfiguracja początkowa dla słowa x, to $\langle s, x, \bot\rangle$. □

Def. 19 Relacja przejścia między konfiguracjami $\rightarrow$:

- jeśli $(p, A) \xrightarrow{a} (q, \gamma) \in \delta$, to dla dowolnych $y \in \Sigma^*, \beta \in \Gamma^*$ mamy $(p, ay, A\beta) \rightarrow (q, y, \gamma\beta)$,
- jeśli $(p, A) \xrightarrow{\varepsilon} (q, \gamma) \in \delta$, to dla dowolnych $y \in \Sigma^*, \beta \in \Gamma^*$ mamy $(p, y, A\beta) \rightarrow (q, y, \gamma\beta)$.

Niech $\rightarrow^*$ będzie zwrotnio-przechodnim domknięciem relacji $\rightarrow$. Wówczas język akceptowany przez automat możemy określić następująco:

akceptacja stanem $L(M)$ składa się z takich słów x, że $(s, x, \bot) \rightarrow^* (q, \varepsilon, \gamma)$ dla pewnych $q \in F$ oraz $\gamma \in \Gamma^*$,

akceptacja pustym stosem $L(M)$ składa się z takich słów x, że $(s, x, \bot) \rightarrow^* (q, \varepsilon, \varepsilon)$ dla pewnego $q \in Q$. Taką konwencję właśnie przyjmiemy.

□

Przykład: Tyle samo liter a, co b — formalnie. Przykładowe obliczenie dla *aabbba*.

Przykład: Automat akceptujący pustym stosem wyrażenia nawiasowe [,]. Przykładowe obliczenie prowadzące do zaakceptowania [[][[]]].

Przykład: Automat akceptujący palindromy nad alfabetem $\{a, b\}$. Przykładowe obliczenie akceptujące słowa *abba* i *baabaab*.

Przykład: Automat akceptujący $\overline{\{ww : w \in \{a, b\}^*\}}$. Automat zgaduje najpierw, czy słowo jest parzystej, czy nieparzystej długości. Jeśli nieparzystej, to tylko to sprawdza. Jeśli parzystej, to musi podzielić to słowo na połowę i znaleźć niezgodność między pierwszą i drugą połową. Wówczas słowo ma postać $xaybz$ lub $xbyaz$, przy czym $|y| = |x| + |z|$. Niedeterministycznie zgadujemy, który przypadek ma miejsce. Następnie zliczamy znaki słowa x wkładając je na stos. Niedeterministycznie zgadujemy, kiedy jest koniec x i sprawdzamy, czy kolejny znak to a. Dalej zliczamy znaki y zdejmując elementy ze stosu. Po wczytaniu $|x|$ pierwszych znaków y odsłania się $\bot$ i wczytując dalsze znaki y zliczamy je wkładając elementy na stos. Zgadujemy niedeterministycznie kiedy kończy się y i sprawdzamy, czy dalej jest b. Do wczytania pozostało słowo z, które ma tyle znaków, ile elementów włożyliśmy na stos. Po jego wczytaniu powinien odsłonić się $\bot$, który zdejmujemy.

9.3 Równoważność gramatyk bezkontekstowych i automatów stosowych

9.3.1 Symulacja wyprowadzenia przez automat stosowy

Mamy daną gramatykę bezkontekstową $G = \langle N, \Sigma, P, S \rangle$. Zbudujemy równoważny jej automat. Automat ten będzie miał tylko jeden stan $Q = \{s\}$. Na stosie będzie przechowywał

symbole terminalne nieterminalne. $\Gamma = N \cup \Sigma$. Przyjmujemy, że $S = \perp$. Jeśli na wierzchołku stosu jest terminal, to oczekujemy, że jest on na wejściu, wczytujemy i zdejmujemy. Jeśli na wierzchołku jest nieterminal, to zdejmujemy go i wkładamy na stos prawą stronę jednej z produkcji dla tego symbolu. Akceptujemy pustym stosem.

Automat ten realizuje lewostronne wyprowadzenia, to znaczy takie, w których zawsze jest rozwijany skrajnie lewy nieterminal. Jeśli wczytywane słowo to x, $x = yz$, to konfiguracji (s, z, α) odpowiada w wyprowadzeniu słowo $y\alpha$. Osiągając konfigurację $(s, \varepsilon, \varepsilon)$ tym samym mamy w wyprowadzeniu słowo x.

Przykład: Przerobić tą metodą gramatykę wyrażeń nawiasowych na automat.

Przykład: Przerobić tą metodą gramatykę generującą słowa zawierające tyle samo a co b na automat.

9.3.2 Symulacja automatu z jednym stanem przez gramatykę

Powyższa konstrukcja da się z grubsza odwrócić. Załóżmy, że mamy dany automat stosowy z jednym stanem $M = \langle \{s\}, \Sigma, \Gamma, \delta, s, \perp \rangle$. Dla uproszczenia zakładamy, że $\Gamma \cap \Sigma = \emptyset$. Nasza gramatyka ma postać $\langle \Gamma, \Sigma, P, \perp \rangle$, przy czym P zawiera następujące produkcje:

- jeśli $(s, A) \xrightarrow{a} (s, \gamma) \in \delta$, to $A \to a\gamma$,
- jeśli $(s, A) \xrightarrow{\varepsilon} (s, \gamma)) \in \delta$, to $A \to \gamma$.

Odpowiedniość jest taka jak poprzednio.

Przykład: Przerobić tą metodą automat akceptujący wyrażenia nawiasowe na gramatykę.

Przykład: Przerobić tą metodą automat akceptujący słowa zawierające tyle samo a co b na gramatykę.

9.3.3 Redukcja zbioru stanów do jednego

Pozostało nam pokazać, że dla każdego automatu stosowego $M = \langle Q, \Sigma, \Gamma, \delta, s \rangle$ istnieje równoważny mu automat z jednym stanem. Bez zmniejszenia ogólności możemy założyć, że automat M w momencie opróżnienia stosu będzie zawsze w jednym stanie q.

Wszystkie istotne informacje będziemy pamiętać na stosie, $\Gamma' = Q \times \Gamma \times Q$. $M' = \langle \{s\}, \Sigma, \Gamma', \delta' s \rangle$. Automat M' będzie na początku miał na stosie trójkę (p, A, q) i akceptował słowo x wtw., gdy M będzie zaczynało w stanie p i z symbolem A na stosie, wczytywało

x i kończyło w stanie q z pustym stosem. Bez zmniejszenia ogólności możemy założyć, że M akceptuje pustym stosem w jednym tylko stanie. Ogólnie, jeżeli na wierzchołku stosu jest (p, A, q), to odpowiada to sytuacji, gdy na wierzchołku M jest A, M jest w stanie p i po zdjęciu symbolu A ze stosu przejdzie do stanu q. Stany pamiętane na stosie muszą się „zazębiać", tzn. pod stanem (p, A, q) musi być stan postaci (q, B, r) itd.

δ' jest określone następująco:

- jeśli $((p, a, A), (q, B_1 B_2 \dots B_k)) \in \delta$, to $((s, a, (pAq_k)), (a, (q_0 B_1 q_1)(q_1 B_2 q_2) \dots (q_{k-1} B_k q_k)) \in \delta$,
- jeśli $((p, a, A), (q, \varepsilon))$, to $((s, a, (p, A, q)), (s, \varepsilon)) \in \delta$,

Oznacza to automat wysoce niedeterministyczny. Niedeterminizm pozwala nam przewidywać przyszłość i dzięki temu przechowywać stany automatu na stosie.

Ćwiczenia

- Skonstruuj automat stosowy akceptujący poniższe języki. [JAO, TO:] Jeśli automat ma więcej niż jeden stan, to przerób go tak, aby miał tylko jeden stan. Następnie przekształć automat stosowy w gramatykę bezkontekstową.
  - sparowane nawiasy ()[]{},
  - palindromy nad alfabetem $\{a, b, c\}$,
  - $\{a^i b^{2i}\}$,
  - $\{a^i b^j a^j b^i\}$,
  - $\{w : \#_a(w) >= 2\#_b(w)\}$ (kilka możliwych rozwiązań),
  - $\{a^i b^j c^k : i \neq j \vee j \neq k\}$,
  - wyrażenia arytmetyczne w notacji polskiej (dla uproszczenia liczby mogą być tylko jednocyfrowe),
  - wyrażenia arytmetyczne w odwrotnej notacji polskiej (dla uproszczenia liczby mogą być tylko jednocyfrowe).

Wykład 10. Deterministyczne automaty stosowe[JFA, TO]

Uzupełnić ...

- Definicja — dochodzi znacznik końca wejścia i warunek, że w każdej konfiguracji i dla każdego możliwego symbolu na wejściu tylko jedno przejście jest możliwe.
- Tw. Dopełnienie deterministycznego języka bezkontekstowego, jest deterministyczne bezkontekstowe. Dowód:
  - duplikujemy stany, $q' = q$, ale jeśli był wczytany znacznik końca wejścia, przejścia odpowiednie, narazie akceptowany język jest ten sam,
  - dodajemy stan akceptujący f' i stany odrzucające r i r'; jeśli czyścimy stos przed końcem wejścia, to skaczemy do r, a jeżeli po to do f', po osiągnięciu r automat wczytuje wejście i dochodzi do r', stosu nie czyścimy, tylko automat zapętla się w f' lub r',
  - poprawimy automat tak, ze zawsze albo akceptuje, albo odrzuca,
  - w jaki sposób automat może się zapętlić?
  - bierzemy taki nieskończony podciąg konfiguracji, że zawartość stosu po danej chwili nigdy nie jest mniejsza, $i_0 < i_1 < \ldots$, taki ciąg można skonstruować zaczynając od $i_0 = 0$, bo stos nie jest nigdy pusty,
  - z tego ciągu wybieramy taki nieskończony podciąg $j_0 < j_1 < \ldots$, że w chwilach tych jest wykonywane to samo przejście, istnieje i musi to być ε-przejście, co więcej automat nie zagląda pod to co jest na wejściu, czyli po wejściu do takiego stanu i symbolu na wierzchołku stosu musi się zapętlić, niezależnie od wejścia,
  - takie przejście zastępujemy przejściem do r (lub r'),
  - postępujemy tak, tak długo, aż inne zapętlenie niż odrzucenie lub akceptacja nie będzie możliwe,
  - wówczas dopełnienie uzyskujemy zamieniając miejscami r' i f'.
- Fakt: Istnieją języki bezkontekstowe, które nie są deterministyczne, np. język postaci

$$\overline{\{ww : w \in \{a, b\}^*\}}$$

.

Ćwiczenia

- Skonstruuj **deterministyczny** automat stosowy akceptujący proste wyrażenia arytmetyczne Wskazówka: Opracuj najpierw jednoznaczną gramatykę bezkontekstową. Zapewnij, aby mnożenie i dzielenie wiązało silniej niż dodawanie i odejmowanie. Operacje arytmetyczne powinny być prawostronnie łączne. Dla uproszczenia przyjmij, że liczby muszą być jednocyfrowe.
- [K HW. 7.1] Skonstruuj **deterministyczny** automat stosowy akceptujący wyrażenia regularne (nad alfabetem $\{a, b\}$). Wskazówka: Opracuj najpierw jednoznaczną gramatykę bezkontekstową. Zapewnij, aby * wiązała silniej niż konkatenacja, a ta wiązała silniej niż suma. Operacje binarne powinny być prawostronnie łączne.

Wykład 11. Maszyny Turinga i obliczalność

11.1 Teza Churcha

Na czym polega pojęcie obliczalności. Problem obliczalności zajmował naukowców jeszcze przed powstaniem prawdziwych komputerów. Powstało wiele formalizmów opisujących to pojęcie:

- logika kombinatoryczna [Schönfinkel 1924, Curry 1929],
- rachunek λ [Church 1933, Kleene 1935],
- maszyny Turinga [1936],
- systemy (przepisywania termów) Posta [1936],
- gramatyki ogólne (typu 0) (szczególny przypadek systemów Posta),
- rachunek funkcji μ-rekurencyjnych [Gödel, Herbrand 1965],

Wszystkie one opisują pojęcie obliczalności wychodząc z innych podejść, ale okazuje się, że wszystkie one są sobie równoważne. Stąd teza Churcha: wszystkie te formalizmy opisują dokładnie to co kryje się pod intuicyjnym pojęciem obliczalności.

11.2 Uniwersalność

Wszystkie te formalizmy są wystarczająco silne, aby zapisać w nich uniwersalny mechanizm obliczający, tzn. taki mechanizm, który dostając „opis algorytmu" i dane wykona ten algorytm na tychże danych.

- w rachunku funkcji μ-rekurencyjnych istnieje numeracja funkcji i istnieje funkcja, która symuluje funkcję o podanym numerze,
- istnieje maszyna Turinga, która potrafi symulować każdą inną maszynę Turinga na podstawie jej opisu,
- w dowolnym języku programowania można napisać interpreter tegoż języka,
- Gödel udowodnił, że język teorii liczb (z arytmetyką Peano) jest wystarczająco silny aby mówić o istnieniu dowodów, np. można napisać formułę, która mówi, że inną formułę da się udowodnić. Dalej, da się napisać formułę, która oznacza „mnie nie da się udowodnić". Jest to formuła prawdziwa, ale taka, której nie da się udowodnić.

11.3 Maszyny Turinga

- Nieformalny opis maszyn Turinga. Taśma, głowica, zapis, odczyt, przesuwanie, sterownik, konfiguracja początkowa, lewy ogranicznik, akceptacja, odrzucenie, zapętlenie.
- Definicja MT: $M = \langle Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r \rangle$, gdzie:
  - Q, to skończony zbiór stanów,
  - $\Sigma \subseteq \Gamma$, to alfabet wejściowy,
  - Γ, to alfabet taśmowy,
  - $\vdash \in \Gamma$, to lewy ogranicznik,
  - $\sqcup \in \Gamma$, to symbol pusty, blank,
  - $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, P\}$, to relacja przejścia; nie można zamazać $\vdash$, ani przesunąć głowicy na lewo,
  - $s \in Q$, to stan początkowy,
  - $t \in Q$, to stan akceptujący,
  - $r \in Q, r \neq t$, to stan odrzucający.
- Definicja konfiguracji $\in Q \times \Gamma^* \times \mathcal{N}$, $\to$ i $\to^*$. Konfiguracja początkowa.
- Definicja języków obliczalnych i częściowo obliczalnych.
- Równoważna sformułowanie języków częściowo obliczalnych – istnieje program wypisujący wszystkie słowa z języka.

Przykład: Maszyna akceptująca język $\{a^n b^n c^n : n \geq 0\}$. Tak więc klasa języków obliczalnych jest szersza niż bezkontekstowych.

- Sprawdzamy czy słowo pasuje do $a^* b^* c^*$.
- Dostawiamy ogranicznik na końcu.
- Jeździmy w kółko wymazując za każdym razem jedno a, b i c.
- Jeśli na końcu wszystkie znaki zostały wymazane, to akceptujemy.

Przykład: Maszyna akceptująca język $\{ww : w \in \{a, b\}^*\}$.

11.4 Uniwersalna maszyna Turinga

Ustalamy alfabet wejściowy $\{0, 1\}$. Kodowanie maszyn Turinga oraz alfabetu wejściowego. Język akceptowany, to $\{M\$x : x \in L(M)\}$.

11.5 Problemy nieobliczalne

Intuicja — maszyny Turinga potrafią to samo, co programy komputerowe. Czy istnieje problem nieobliczalny? Tak.

11.5.1 Technika diagonalizacji

Przykład: Dowód tw. Cantora.

Przykład: Problem stopu $\{M\$x : M$ zatrzymuje się dla danych $x\}$. Załóżmy, że istnieje maszyna Turinga M', która rozstrzyga ten język. Wówczas niech M'' akceptuje $\{M : M$ nie zatrzymuje się dla kodu $M\}$, ale zamiast odrzucać zapętla się. Czyli M'' zatrzymuje się dla kodu M, wtw, gdy M nie zatrzymuje się dla własnego kodu. Czy M'' zatrzyma się dla własnego kodu?

Fakt: Jeśli A i $\overline{A}$ są częściowo obliczalne, to A są one też obliczalne.

Problem stopu jest częściowo obliczalny — vide maszyna uniwersalna. Czy istnieje jakiś problem, który nie jest nawet częściowo obliczalny? Tak, np. dopełnienie problemu stopu, bo gdyby był częściowo obliczalny, to byłby też obliczalny, a nie jest. (To jest przykład redukcji.)

11.6 Wariacje nt. MT

Wszystkie poniższe warianty maszyn Turinga mają taką samą siłę wyrazu, ale czasem łatwiej się nimi posługiwać:

- maszyna z wieloma taśmami i głowicami,
- maszyna z taśmą nieskończoną w obydwie strony,
- maszyna z płaszczyzną magnetyczną.

Ćwiczenia

- Opisz MT, która akceptuje język (nie ma to być formalna konstrukcja MT, ale ścisły opis sposobu jej działania):
  - [K Ex. 96] $\{a^{n^2}\}$ (wskazówka: $1 + 3 + 5 + \cdots + (2k - 1) = k^2$),
  - [K HW. 8.1] $\{a^{2^n}\}$,
  - [K Ex. 97] $\{a^i b^j\}$, a zatrzymując się ma na taśmie słowo postaci $a^{NWD(i,j)}$ (wskazówka: do obliczania NWD służy algorytm Euklidesa; jest kilka wersji tego algorytmu: przez odejmowanie, przez modulo 2, przez badanie parzystości — wybierz taką wersję, która najbardziej pasuje),

Wykład 12. Problemy obliczalne i nieobliczalne

W poprzednim punkcie pokazaliśmy, że problem stopu nie jest obliczalny. Zastanówmy się czy następujące programy są obliczalne. Zakładamy, że dana jest pewna maszyna Turinga M. [K, str. 235]

- 1. Czy M ma przynajmniej 42 stany?
  2. Czy M wykonuje więcej niż 42 kroki dla pustego wejścia?
  3. Czy M wykonuje więcej niż 42 kroki dla pewnego wejścia?
  4. Czy M wykonuje więcej niż 42 kroki dla każdego wejścia?
  5. Czy M dla pustego wejścia kiedykolwiek przesuwa głowicę dalej niż 42 pozycje na prawo od lewego ogranicznika?

  Te problemy są obliczalne, ponieważ liczba konfiguracji, które należy rozważyć jest skończona.

- 1. Czy M akceptuje puste słowo?
  2. Czy M akceptuje dowolne słowo?
  3. Czy M akceptuje każde słowo?
  4. Czy M akceptuje skończony język?

  Nieobliczalne, przez sprowadzenie problemu stopu do tych problemów. Mamy dane $M\#x$. Musimy stwierdzić, czy M akceptuje x. Generujemy M', która:

  – wymazuje dane z wejścia,
  – zapisuje na taśmie x,
  – symuluje M,
  – akceptuje jeśli M się zatrzymuje.

  Kod M' jest obliczalny na podstawie M i x — analogia z tłumaczeniem programów. M' należy do wybranego powyższego języka wtw. gdy $M\#x \in STOP$.

- 1. Czy M akceptuje język regularny?
  2. Czy M akceptuje język bezkontekstowy?
  3. Czy M akceptuje język obliczalny?

Nieobliczalne. Mamy dane $M\#x$. Musimy stwierdzić, czy M akceptuje x. Niech A będzie dowolnym językiem częściowo obliczalnym, ale nie obliczalnym (np. językiem STOP). Generujemy M'', która:

- przepisuje wejście y na dodatkową ścieżkę,
- zapisuje na ścieżce danych x,
- symuluje M dla x,
- jeśli M się zatrzyma, to symuluje maszynę akceptującą A dla y i tak samo akceptuje lub nie

$$L(M'') = \begin{cases} A & ; \quad M \text{ się zatrzymuje dla x} \\ \emptyset & ; \quad \text{wpp.} \end{cases}$$

12.1 Sprowadzanie

Def. 20 Sprowadzenie problemu/języka A do B to taka **obliczalna** funkcja σ, że $x \in A \Leftrightarrow \sigma(x) \in B$.

Jeśli istnieje sprowadzenie A do B, to piszemy $A \leq B$. □

Fakt: $\leq$ jest zwrotna i przechodnia.

Przykład: Sprowadzenia z przykładów f)–i) i j)–l).

Fakt: Jeśli $A \leq B$, to $\overline{A} \leq \overline{B}$.

Fakt: Jeśli $A \leq B$ i B jest (częściowo) obliczalne, to A też jest (częściowo) obliczalne.

Fakt: Jeśli $A \leq B$ i A nie jest (częściowo) obliczalne, to B też nie jest.

Przykład: Niech $FIN = \{M : L(M) \text{ jest skończony}\}$. Pokażemy, że ani FIN, ani $\overline{FIN}$ nie są częściowo obliczalne.

- $\overline{STOP} \leq FIN$: $\sigma(M\#x) = M'$, gdzie M':
  - wymazuje wejście,
  - zapisuje na wejściu x,
  - symuluje M,
  - akceptuje jeśli M się zatrzymuje.

M' akceptuje albo $\emptyset$, albo Σ^*. M zapętla się $\Leftrightarrow L(M')$ jest skończony, czyli $M\#x \in \overline{STOP} \Leftrightarrow \sigma(M\#x) \in FIN$.

- $STOP \leq FIN$: $\tau(M\#x) = M''$, gdzie M'':
  - zapisuje wejście y na dodatkowej ścieżce,
  - zapisuje na ścieżce wejścia x,
  - symuluje działanie M dla danych x przez co najwyżej $|y|$ kroków,
  - akceptuje jeśli w trakcie symulacji M nie zakończyła działania, wpp. odrzuca.

  M zatrzymuje się dla $x \Leftrightarrow M''$ akceptuje słowa o ograniczonej długości $\Leftrightarrow L(M'')$ jest skończony, czyli $M\#x \in STOP \Leftrightarrow \tau(M\#x) \in FIN$.

Ćwiczenia

- [K, Ex. 106] Czy obliczalny jest następujący problem: Dla danego $M\#x$, czy maszyna M dla wejścia x zapisze kiedykolwiek na taśmie inny symbol niż blank?
- [K, Ex. 108] Czy podane problemy są obliczalne?
  1. Mając daną maszynę Turinga M i słowo y, stwierdzić, czy M dla wejścia y kiedykolwiek zapisze na swojej taśmie symbol #?
  2. Mając daną gramatykę bezkontekstową G stwierdzić, czy G generuje jakiekolwiek słowo?
  3. Mając daną maszynę Turinga M, stwierdzić, czy istnieje nieskończenie wiele innych, równoważnych jej maszyn Turinga?
- Ex. 109. Czy podane problemy są częściowo obliczalne?
  1. $\{(M, N) : M$ wykonuje mniej kroków dla pustego słowa na wejściu, niż $n\}$,
  2. $\{M : M$ wykonuje mniej niż 42^42 kroków dla pewnego wejścia $\}$,
  3. $\{M : M$ wykonuje mniej niż 42^42 kroków dla przynajmniej 42^42 różnych słów na wejściu $\}$,
  4. $\{M :$ Dla każdego wejścia M wykonuje mniej niż 42^42 kroków $\}$.
- [K, Ex. 111] Który z podanych problemów jest częściowo obliczalny?
  1. $\{M : L(M)$ zawiera przynajmniej 42 słowa $\}$,
  2. $\{M : L(M)$ zawiera co najwyżej 42 słowa $\}$.

Wykład 13. Hierarchia Chomsky'ego[JFA]

13.1 Niedeterministyczne maszyny Turinga

- Definicja niedeterministycznej maszyny Turinga.
- Maszyna niedeterministyczna nie ma stanu odrzucającego.
- Interesuje nas tylko, czy może ona zaakceptować dane słowo.
- Czy maszyny niedeterministyczne i deterministyczne akceptują te same klasy języków?
- Tak, bo maszyna deterministyczna może symulować wszystkie możliwe obliczenia maszyny niedeterministycznej.
- Niedeterministyczne maszyny Turinga akceptują języki częściowo obliczalne.

13.2 Gramatyki ogólne (typu 0)

Gramatyki bezkontekstowe nie są najogólniejszą znaną formą gramatyk. Gramatyki ogólne (typu 0) dopuszczają produkcje postaci

$$\alpha \to \beta$$

dla $\alpha, \beta \in (\Sigma \cup N)^*$, $\alpha \neq \varepsilon$. Możemy więc określić, że w wyprowadzeniu można zamieniać dowolny niepusty napis α na β.

Twierdzenie: Gramatyki ogólne są równoważne (niedeterministycznym) maszynom Turinga.

Szkic dowodu:

- Sprowadzenie gramatyki ogólnej do maszyny Turinga. MT zapamiętuje słowo z wejścia, symuluje wyprowadzenie, wybierając niedeterministycznie produkcje. Jeżeli osiągnie słowo dane na wejściu, to akceptuje, wpp. zapętla się.
- Sprowadzenie MT do gramatyk ogólnych. Mając daną MT musimy podać gramatykę równoważną danej MT. konstrukcja opiera się na następujących obserwacjach:
  - Konfigurację MT możemy reprezentować za pomocą słowa złożonego z (niepustej) zawartości taśmy, z wstawionym w odpowiednim miejscu znacznikiem reprezentującym położenie głowicy i stan MT.
  - Każdy możliwy ruch MT można reprezentować jako produkcję gramatyki ogólnej. Można więc skonstruować gramatykę ogólną, której wyprowadzenia naśladują dokładnie możliwe obliczenia zadanej MT.

– Mając daną MT M konstruujemy maszynę M', która dla pustego wejścia działa tak:
  * zgaduje niedeterministycznie słowo x i tworzy dwie jego kopie (na dwóch ścieżkach),
  * jednej ze ścieżek symuluje działanie maszyny M,
  * gdy M akceptuje słowo x, jako zawartość taśmy przywracane jest słowo x i akceptuje.

  Tak więc dokładnie dla słów x akceptowanych przez M maszyna M' może rozpocząć działanie z pustą taśmą i zakończyć je ze słowem x zapisanym na taśmie.
– Gramatyka ogólna równoważna maszynie M, to gramatyka symulująca działanie maszyny M' z dodanymi produkcjami, które:
  * przekształcają aksjomat w reprezentację konfiguracji początkowej maszyny M' dla pustego wejścia,
  * pozwalają na usunięcie znacznika reprezentującego głowicę dla stanu akceptującego.

  W takiej gramatyce możemy wyprowadzić słowa, które są końcowymi zawartościami taśmy maszyny M', a więc dokładnie słowa akceptowane przez M.

Twierdzenie: Język generowany przez gramatykę ogólną może być nierozstrzygalny.

dowód Jest to natychmiastowy wniosek z powyższej konstrukcji, oraz z faktu, że MT akceptują języki częściowo obliczalne, ale nie koniecznie obliczalne.

13.3 Gramatyki kontekstowe

Gramatyki kontekstowe to takie gramatyki ogólne, w których produkcje są postaci $\alpha \to \beta$, gdzie $\alpha, \beta \in (N \cup \Sigma)^*$, $|\beta| \geq |\alpha|$ i $\alpha \neq \varepsilon$.

Fakt: Języki generowane przez gramatyki kontekstowe są obliczalne.

szkic dowodu: Dla gramatyki liniowej długość słowa w wyprowadzeniu nie może maleć. Można więc prześledzić wszystkie możliwe wyprowadzenia słów o zadanej długości (wykrywając ew. zapętlenia).

Fakt: Gramatykom kontekstowym odpowiadają MT z taśmą ograniczoną liniowo.

szkic dowodu: Jeśli rozmiar taśmy jest ograniczony liniowo (tzn. istnieje taka stała c, że dla dowolnego wejścia x MT wykorzysta co najwyżej $c \cdot |x|$ miejsc na taśmie), to można ograniczyć długość wykorzystywanej taśmy dowolnie wybraną mniejszą stałą c' — stosując podobną technikę do symulowania MT z wieloma ścieżkami. Następnie możemy zasymulować działanie MT ograniczonej liniowo za pomocą gramatyki kontekstowej w sposób podobny do symulacji podanej dla gramatyk ogólnych.

MT symulująca gramatykę ogólną w przypadku gramatyki kontekstowej da MT ograniczoną liniowo.

13.4 Hierarchia Chomsky'ego

Omawiane klasy języków można scharakteryzować za pomocą rodzajów gramatyk:

Nr	Gramatyki	Maszyny	Języki
0	ogólne	(niedet.) maszyny Turinga	częściowo obliczalne
1	kontekstowe	ograniczone liniowo MT	kontekstowe
2	bezkontekstowe	automaty stosowe	bezkontekstowe
3	liniowe	automaty skończone	regularne

Dodatkowo wśród języków częściowo obliczalnych wyróżniamy obliczalne, gdy dany język i jego dopełnienie są częściowo obliczalne.

13.5 Wyniki *-ek