Modele Obliczeń

Wybrane Zagadnienia

"To nie jest cierpienie dla cierpienia, cierpimy w jakimś wyższym celu"

Popełnione przez

Bajtocjanin

Kraków

Anno Domini 2024

Uwaga, niniejszy Suffering spisywałem w Obsydianie i próbowałem go przekonwertować na PDF własnym konwerterem, dlatego część symboli może być nieczytelna lub błędna, zachowaj czujność, jego zaletą jest to, że obejmuje cały materiał kursu.

Spis treści

Rozdział 1

Języki Regularne

1.1 Języki

Alfabet to dowolny zbiór, którego elementy nazywane są *literami* bądź *symbolami*.

Słowo nad alfabetem to skończony ciąg liter z alfabetu A.

eg. 01010 jest słowem nad alfabetem 0,1

Długość słowa $w \in \Sigma^n$ to n oznaczane $|w|$, $\#w$, len w

Słowo puste to jedyny element Σ^0, oznaczane ϵ

Zbiór wszystkich słów nad alfabetem Σ to $\Sigma = \bigcup_{n \in \mathbb{N}} \Sigma^n$

Język nad alfabetem Σ to dowolny podzbiór $L \subseteq \Sigma^n$

Język prosty to $\emptyset \subseteq \Sigma^n$, *język pełny* nad Σ to Σ .

Formalnie słowo jednoliterowe a jest tym samym co litera

1.1.1 Oznaczenia

Alfabety oznaczamy je wielkimi literami greckimi np. Δ, Σ...

Litery oznaczamy początkowymi małymi literami alfabetu łacińskiego a, b..

Słowa oznaczamy końcowymi małymi literami alfabetu łacińskiego, w, u, x

1.1.2 Konkatenacja

Katenacja, konkatenacja, składanie słów to funkcja $\Sigma \times \Sigma = \Sigma$ zdefiniowana dla $u \in \Sigma^n, v \in \Sigma^m$ jako $uv \in \Sigma^{n+m}$

$$uv(i) = \begin{cases} u(i), & \text{dla } 0 \leqslant i \leqslant n \\ v(i-n), & \text{dla } n \leqslant i \leqslant n+m \end{cases} \tag{1.1}$$

Po prostu zlepiamy ze sobą słowa tworząc dłuższe słowo

Słowo x jest *prefiksem* słowa w, gdy istnieje słowo y takiem że w=xy.

Analogicznie słowo y jest sufiksem słowa w, gdy istnieje słowo x takie że w = xy

Prefiks lub sufix właściwy to taki, gdy $x \neq w$, ϵ jest sufiksem właściwym

1.1.2.1 Własności konkatenacji (dość naturalne)

- jest łączna $u(vw) = (uv)w$
- słowo puste jest elementem neutralnym
- $|uv| = |u| + |v|$

1.1.3 Monoid

Monoid to struktura $(M, \cdot, 1)$, w której działanie binarne $\cdot$ jest łączne i 1 jest jego elementem neutralnym

Dla dowolnego alfabetu Σ, struktura $(\Sigma, \cdot, \epsilon)$ jest więc monoidem

1.1.4 Potęga

Potęgę definiujemy następującą rekurencyjną relacją

1.1.4.1 Dla słów

$$w^n = \begin{cases} \epsilon & \text{dla } n = 0 \\ w^{n-1}w & \text{dla } n > 0 \end{cases} \tag{1.2}$$

1.1.4.2 Dla języków

$$L^n = \begin{cases} \{\epsilon\} & \text{dla } n = 0 \\ L^{n-1}L & \text{dla } n > 0 \end{cases} \tag{1.3}$$

1.1.5 Gwiazdka Kleene'go

$L = \bigcup_{n \in \mathbb{N}} L^n$

1.2 Operacje na językach

1.2.1 Operacje na językach

- suma
- przecięcie
- dopełnienie
- katenacja
- potęga
- gwiazdka kleenego

1.2.1.1 Własności operacji na językach

todo

- Obowiązują wszystkie teoriomnogościowe własności
- $(L_1L_2)L_3 = L_1(L_2L_3)$
- $\emptyset L = L\emptyset = 0$
- $\{\epsilon\}L = L\{\epsilon\} = L$
-

Języki nad alfabetem Σ tworzą zatem monoid.

1.3 Algebra Kleenego

Twierdzenie 1. *Temat spoza wymagań egzaminacyjnych*

Algebra $A = (A; +, \cdot, , 0, 1)$ jest **algebrą Kleenego** wtw. gdy zachodzą wszystkie aksjomaty:

1. $(a + b) + c = a + (b + c)$
2. $(a + b) = (b + a)$
3. $a + a = a$
4. $a + 0 = a$
5. $(ab)c = a(bc)$
6. $a \cdot 1 = 1 \cdot a = a$
7. $a \cdot 0 = 0 \cdot a = 0$
8. $a(b + c) = ab + ac$
9. $(a + b)c = ac + bc$
10. $1 + aa \leqslant a$
11. $1 + aa \leqslant a$
12. $ax + b \leqslant x \Rightarrow ab \leqslant x$
13. $xa + b \leqslant x \Rightarrow ba \leqslant x$

    gdzie zapis $a \leqslant b$ jest skrótem a+b=b

1.4 Definicja języków regularnych

Języki regularne definiujemy rekurencyjnie:

- $\emptyset$ jest językiem regularnym

- a jest językiem regularnym dla każdego $a \in \Sigma^1$
- jeśli L_1 i L_2 są językami reguralnymi to $L_1 \cup L_2$ też
- jeśli L_1 i L_2 są językami reguralnymi to $L_1 L_2$ też
- jeśli L jest językiem regularnym to L też
- żaden inny język nie jest regularny nad Σ

1.4.1 Wyrażenia regularne

Języki regularne można zdefiniować przez *wyrażenia regularne*

Wyrażenia regularne nad alfabetem Σ to język RE_Σ nad alfabetem $\Sigma' = \Sigma \cup \{+, , 0, 1, (,)\}$ zdefiniowany jako:

- 0 jest wyrażeniem regularnym
- 1 jest wyrażeniem regularnym
- a jest wyrażeniem regularnym dla każdego $a \in \Sigma$
- jeśli α_1 i α_2 są wyrażeniami regularnymi to $(\alpha_1 + \alpha_2)$ też
- jeśli α_1 i α_2 są wyrażeniami regularnymi to $(\alpha_1 \alpha_2)$ też
- jeśli α jest wyrażeniem regularnym to α też
- żadne inne słowo nad Σ' nie jest wyrażeniem regularnym nad Σ

można pominąć zbędne nawiasy pamiętając o priorytecie $+ < \cdot <$

1.4.1.1 Intuicja

- $+$ działa jak lub
- działa jako n potwórzeń wyrażania (może być 0)
- potęga $^d ziaajak$

Wyrażenia regularne opisują języki zgodnie z definicją:

- L(0) = $\emptyset$
- L(1) = $\{\epsilon\}$
- L(a) = $\{a\}$ dla $a \in \Sigma$
- L(α + β) = $L(\alpha) \cup L(\beta)$
- L($\alpha\beta$) = $L(\alpha)L(\beta)$
- L(a) = $L(\alpha)$

Każde wyrażenie regularne nad Σ opisuje języka regularny nad Σ i każdy taki język jest opisywany przez takie wyrażenie.

Gdy $L = L(\alpha)$ to mówimy, że α opisuje język L

Wyrażenia regularne α i β są równoważne jeśli $L(\alpha) = L(\beta)$

Rozdział 2

Automaty Skończone

2.1 Deterministyczne automaty skończone (DFA)

DFA to piątka $A = (Q, \Sigma, \delta, s, F)$, gdzie

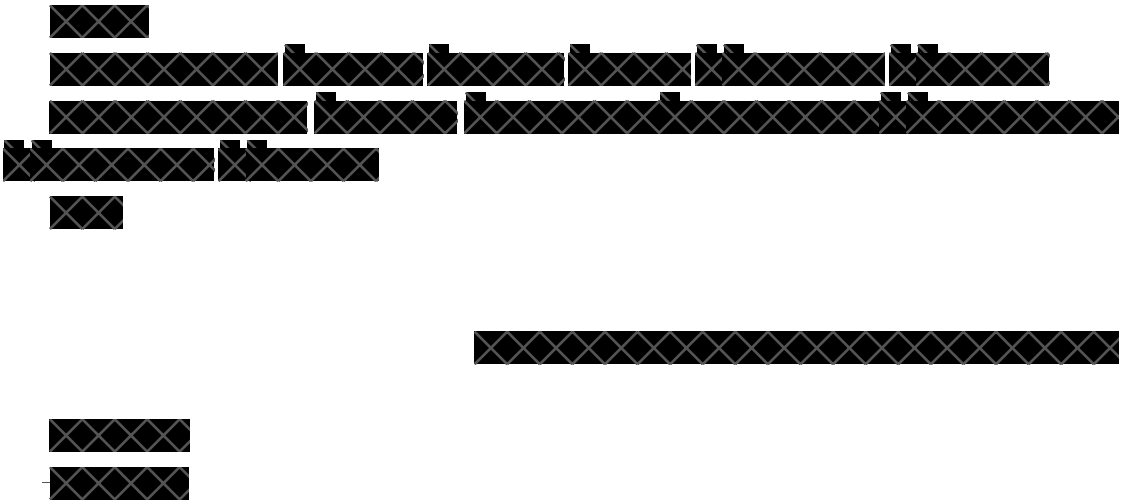
- Q to skończony zbiór stanów
- Σ to alfabet
- $\delta : Q \times \Sigma \to Q$ to funkcja przejścia
- $s \in Q$ to skończony zbiór stanów
- F $\subseteq Q$ to zbiór stanów końcowych

Rozszerzmy funkcje przejścia δ do funkcji $\hat{\delta}$:

$$\hat{\delta}(q, w) = \begin{cases} q & w = \epsilon \\ \delta(\hat{\delta}(q, x), a) & \text{dla } w = xa, \text{ gdzie } x \in \Sigma, a \in \Sigma \end{cases} \tag{2.1}$$

Język akceptowany przez automat A to $L(A) = \{w \in \Sigma : \hat{\delta}(s, w) \in F\}$
Lemat

$$\hat{\delta}(q, xy) = \hat{\delta}(\hat{\delta}(q, x), y) \text{ dla } x, y \in \Sigma \tag{2.2}$$



2.2 Niedeterministyczne automaty skończone (NFA)

NFA to piątka $A = (Q, \Sigma, \delta, S, F)$ analogicznie do DFA, co się zmieniło to że δ nie jest funkcją lecz relacją, relacja jest funkcją więc DFA zawiera się w NFA, a S jest zbiorem stanów startowych, może więc być teraz więcej niż jeden.

Intuicja na grafie NFA jest taka że w DFA wychodziła z każdego wierzchołka jedna strzałka dla jednej litery, a w NFA może wychodzić wiele strzałek.

2.2.1 L(A)

Język akceptowalny przez NFA to:

$L(A) = w \in \Sigma^{\cdot}\hat{\delta}(S, w) \cap F \neq 0$.

2.2.2 L(NFA) = L(DFA)

Klasy języków akceptowanych przez NFA i DFA są równe.

2.2.2.1 $L(DFA) \subseteq L(NFA)$

NFA może trywialnie symulować DFA, po prostu zamieniamy relację na funkcję, gdyż każda funkcja jest relacją więc OK.

2.2.2.2 $L(NFA) \subseteq L(DFA)$

W drugą stronę jest lekko trudniej ale też się da.

A_N jest NFA, $A = (Q, \Sigma, \delta, S, F)$

Korzystamy z faktu, że $\hat{\delta}$ jest funkcją i konstruujemy DFA A_D takie że $A_D = (\mathbb{P}(Q), \Sigma, \hat{\delta}, S, F')$

takie, że $F` = \{\beta \in \mathbb{P}(Q) | B \cap F \neq \emptyset\}$

Intuicyjnie tworzymy DFA, które w swoich stanach trzyma informację, w jakich stanach NFA mogłoby się znaleźć po przejściu tego samego prefiksu słowa.

2.3 - NFA

- NFA jest definiowane jako NFA, z tym że $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times Q$

Bardziej intuicyjnie – pozwalamy na przechodzenie krawędzią bez konsumowania obecnej litery – definicja funkcji $\hat{\delta}$ wygląda bardzo podobnie do tej ze „zwykłego" NFA.

$\delta(B, a) = \{q \in Q : \exists_{qb \in B}(q_B, q, q) \in \delta\}$

2.3.1 ϵ domknięcie

Epsilon domnięcie to

$$B^{A,\epsilon} \bigcup_{i=0}^{\infty} B_i^{A,\epsilon} \tag{2.4}$$

gdzie

$$B_i^{A,\epsilon} = \begin{cases} B, & \text{dla } i = 0 \\ \hat{\delta}(B_{i-1}^{A,\epsilon}, \epsilon) & \text{dla } i > 0 \end{cases} \tag{2.5}$$

intuicyjnie jest to zbiór wszystkich stanów, do których możemy dojść za pomocą samych epsilon przejść z jakiegoś innego zbioru stanów

$\mathbb{P}^{A,\epsilon}(Q)$ to zbiór wszystkich ϵ domkniętych zbiorów stanów, to znaczy

$$\mathbb{P}^{A,\epsilon}(Q) = \{B^{A,\epsilon} : B \in \mathbb{P}(Q)\} \tag{2.6}$$

2.3.2 Równoważność ϵ - NFA i DFA

Jedyne co się zmienia w stosunku do zwykłego NFA to to, że mamy do czynienia z epsilonami, nie jest to jednak problem, bo chodzimy sobie po epsilon domknięciach z użyciem funkcji Δ.

Mamy NFA A_N takie, że $A_N = (Q, \Sigma, \delta, S, F)$

Konstruujemy A_D, które jest DFA, takim, że

$$A_D = (\mathbb{P}^{\mathbb{A}_\mathbb{N},\epsilon}(\mathbb{Q}), \Sigma, \Delta, S, F^{'}) \tag{2.7}$$

gdzie,

$$F^{'} = \{\beta \in \mathbb{P}^{A_N,\epsilon}(Q) | \beta \cap F \neq \emptyset\} \tag{2.8}$$

2.4 Wybrane problemy decyzyjne dla automatów

Możemy założyć, że mamy do czynienia z - NFA, gdyż DFA i NFA to szczególne przypadki - NFA, a w drugą stronę możemy zrobić też konwersje.

2.4.1 Problem membership

Po prostu symulujemy działanie automatu na słowie, jeśli na samym końcu jesteśmy w stanie końcowym to słowo należy do języka.

2.4.2 Problem non-emptiness

Puszczamy DFS od stanu startowego, jeśli możemy dojść do jakiegoś stanu akceptującego to język jest niepusty

2.4.3 Problem finitness

Najpierw puszczamy DFS, dzięki któremu wyeliminujemy wszystkie wierzchołki nieosiągalne, jeśli przy okazji wyszło nam, że pusty to od razu zwracamy że skończony.

Następnie wyrzucamy stany, z których nie da się osiągnąć stanu akceptującego, także DFSem.

Zauważmy, że jeśli język jest nieskończony to musi istnieć jakiś cykl, bo liczba stanów jest skończona, ponadto ten cykl musi składać się nie tylko z krawędzi, bo musimy coś dodawać cały czas.

Nadajemy więc wagi krawędzią, 0 dla krawędzi i -1 dla zwykłych, po czym puszczamy Algorytm Bellmana-Forda, który pozwala sprawdzić czy istnieje cykl ujemny.

2.4.4 Minimalność liczby stanów

Po prostu puszczamy sobie Algorytm (z) D(upy) i sprawdzamy czy jakieś stany się zredukowały

2.4.5 Problem DFA Equivalence

Rozważymy tylko dla DFA, resztę można przekonwertować jak bardzo chcemy.

Najpierw redukujemy sobie wszystkie niepotrzebne rzeczy jak w finiteness, następnie minimalizujemy oba automaty poprzez Algorytm (z) D(upy), po czym sprawdzamy czy stany startowe są równoważne, jeśli są to początkowe automaty też.

2.5 Lemat o pompowaniu dla języków regularnych

Twierdzenie 2. *Lemat o pompowaniu*

Jeśli L jest językiem regularnym to:

$$\exists_{n>0} : \forall_{w:|w|\geqslant n} \exists_{xyz=w} : |xy| \leqslant n \wedge \forall_{i\in\mathbb{N}} : xy^i z \in L \tag{2.9}$$

2.5.1 Dowód

Jeśli L jest językiem regularnym to istnieje DFA A=(Q,Σ, δ , s, F) akceptujące L. Niech $n = |Q| \geqslant 1$ i weźmy dowolne słowo w, dla którego $m = |w| \geqslant n$.

Skoro słowo jest akceptowane to istnieje ścieżka $s = q_0, ..., q_m \in F$

Mamy więc $m + 1 > n$ stanów, czyli jakiś stan musi się powtarzać, wybieramy z nich q_i, którego pierwsze powtórzenie q_j jest najwcześniej.

Mamy zatem $x = a_0...a_{i-1}$, $y = a_i..a_{j-1}$, $z = a_j..a_{m-1}$

$xy \leqslant n$, bo inaczej q_i nie powtarzałby się najwcześniej.

Skoro $q_i = q_j$ to słowo y może wystąpić dowolną (również zerową) liczbę razy

Q.E.D.

2.5.1.1 A komu to potrzebne?

- Pamiętaj, że w lemacie o pompowaniu implikacja jest tylko w jedną stronę. Jest to warunek konieczny, aby język był regularny, lecz nie wystarczający.
- Zwykle używa się, żeby pokazać, że język NIE jest regularny

2.5.1.1.1 Przykład

T: $L = \{a^n b^n, n \in \mathbb{N}\}$ nie jest językiem regularnym

D: Załóżmy że jest regularny, więc musi wtedy spełniać lemat o pompowaniu. Dla w=$a^n b^n$ wybieramy sobie pierwsze powtórzenie. Dla $n \geqslant 1$ jest to a na pozycji

$0, 1$, więc z lematu o pompowaniu wyniałoby, że możemy zrobić dowolną liczbę powtórzen 1, więc w szczególności słowo $a^{2n}b^n \in L$ $x21af$.

2.6 Zamkniętość języków regularnych na operacje

Języki regularne są zamknięte na:

- sumę -¿po prostu $L_1 + L_2$
- przecięcie -¿ robimy sobie DFA, które chodzi po obu, i akceptuje tylko jeśli oba są w stanie akceptującym
- dopełnienie -¿ tworzymy DFA, którego $F' = Q/F$
- konkatenacja -¿ po prostu L_1L_2
- Gwiazdka kleenego -¿ po prostu a
- homomorfizm i odwrotny homomorfizm

2.7 Algorytm (z) D(upy)

Jest to algorytm służący do wyznaczania minimalnego DFA dla danego języka, na wejściu dostaje DFA, zwraca trójkątną macierz Boolowska M, w które $M_{i,j} = 1$ wtw. gdy e są rozróżnialne

2.7.1 A równoważność

Stany q_1 i g_2 zadanego DFA są **A-równoważne** jeśli:

$$\forall_{w\in\Sigma}\hat{\delta}(q_1, w) \in F \Leftrightarrow \hat{\delta}(q_2, w) \in F \tag{2.10}$$

i A rozróżnialne gdy:

$$\forall_{w\in\Sigma}\hat{\delta}(q_1, w) \in F \Leftrightarrow \hat{\delta}(q_2, w) \notin F \tag{2.11}$$

Twierdzenie 3. *A równoważność jest relacją równoważności*

Dowód poprzez sprawdzenie trywialnie warunków

2.7.2 Procedurato

- Wypełniamy M zerami.
- Jeśli $a_i \in F, a_j \notin F$ to $M_{i,j} = 1$
- Dopóki coś się dzieje to dla każdych dwóch stanów q_1 i q_2 oraz dla każdej litery a, jeśli $\hat{\delta}(q_1, a) = q_k$ i $\hat{\delta}(q_2, a) = g_l$ oraz $M_{k,l} = 1$ to $M_{i,j} = 1$. Czyli na chłopski rozum to jeśli każda litera prowadzi do stanu rozróżnialnego to też jest rozróżnialny

2.7.3 Kończąc

Jeśli dla jakiś stanów q_i, g_j $M_{i,j} = 0$ to te stany są A-równoważne. Collapsujemy więc wszystkie równoważne stany (według ich klas równoważności) w jeden wierzchołek i zredukujemy sobie liczbę stanów, a nawet zminimalizujemy.

2.7.4 Minimalizowanie

Twierdzenie 4. *Automat otrzymany w wyniku pracy algorytmu z Dupy jest najmniejszy spośród DFA rozpoznających dany język*

2.7.4.1 Dowód

Zdefiniujmy sobie automat C, taki, że $S_C = S_B$, $Q_C = Q_A \cup Q_B$, funkcje przejść także sumujemy.

Zauważmy, że ponieważ języki są równe to

$$\forall q_a \in Q_A : \exists q_b \in Q_B : q_a \equiv q_b \tag{2.12}$$

Ale to znaczy także, że w A mamy gdzieś jakieś dwa stany, które są równoważne z tym samym stanem w B, z przechodniości one też są wiec równoważne, więc algorytm nasz algorytm by je zredukował, więc A musi być minimalne.

2.8 Twierdzenie Myhille'a-Nerode'a

2.8.1 L-równoważność

Mówimy że relacja jest L-kongruentna dla pewnego języka $L \in \mathcal{P}(\Sigma)$ wtw. gdy:

1. $\forall_{v \in \Sigma} \equiv y \Rightarrow xv \equiv yv$
2. $x \equiv y \Rightarrow (x \in L \Leftrightarrow y \in L)$

2.8.2

Twierdzenie 5. *Twierdzenie Myhille'a-Nerode'a*

Język L jest regularny wtw. gdy istnieje relacja równoważności $\equiv \subseteq \Sigma \times \Sigma$ o skończenie wielu klasach równoważności, która jest L-kongruentna.

2.8.3 Intuicja

2.8.4 Dowód

gdziedowód

Rozdział 3

Języki bezkontekstowe

3.1 Definicja Gramatyki bezkontekstowej

Gramatyka bezkontekstowa to czwórka (N, Σ, P, S) gdzie:

- N - to skończony zbiór zmiennych (nieterminale)
- Σ - alfabet (terminale)
- P - produkcje $P \subseteq N \times (N \cup \Sigma)$
- $S \in N$ - symbol startowy

3.1.1 Forma zdaniowa

Forma zdaniowa to dowolne słowa nad $(N \cup \Sigma)$.

Intuicyjnie gramatyka bezkontekstowa definiuje zasady, na podstawie których możemy tworzyć nowe słowa nad alfabetem Σ, nieterminale N to stany pośrednie, które podmieniamy używając produkcji z P. Każda produkcja robi z nieterminala formę zdaniową.

3.1.2 Relacja $\rightarrow_G$

Dla gramatyki G definiumemy relację $\rightarrow_G$ na formach zdaniowych.

$$\alpha \rightarrow_G \beta \tag{3.1}$$

wtw.

$$\exists_{\alpha_1,\alpha_2,\gamma\in(N\cup\Sigma)} : \exists_{A\in N} : (A,\gamma) \in P \wedge \alpha = \alpha_1 A \alpha_2 \wedge \beta = \alpha_1\gamma\alpha 2 \tag{3.2}$$

Po ludzku wtw. możemy uzyskać formę zdaniową β w jednym kroku z formy zdaniowej α o ile w α jest jakieś wystąpienie nieterminala A, które możemy z produkcjami podmienić na formę zdaniową γ aby uzyskać β

3.1.3 Relacja $\rightarrow_G^*$

$\rightarrow_G$ to zwrotne i przechodnie domknięcie $\rightarrow_G$, czyli możemy z α otrzymać β w skończonym ciągu przekształceń $\rightarrow_G$.

3.1.4 Języki generowane

Języki generowane przez gramatykę G to

$$L(G) = \{w \in \Sigma | S \rightarrow_G^w\} \tag{3.3}$$

Podobnie, języki generowane przez terminal A to

$$L(G, A) = \{w \in \Sigma | A \rightarrow_G^w\} \tag{3.4}$$

Język jest bezkontekstowy jeśli jest generowany przez jakąś gramatykę bezkontekstową.

3.2 Klasy Gramatyk Bezkontekstowych

Gramatyka bezkontekstowa jest:

- **liniowa** jeśli po prawej stronie każdej produkcji występuje występuje co najwyżej jeden nieterminal.
- **prawoliniowa** jeśli wszystkie produkcje są postaci $A \to xB$ lub $A \to x$
- **silnie prawoliniowe** jeśli wszystkie produkcje są postaci $A \to aB$ lub $A \to \epsilon$

  Gdzie $A, B \in N$, $x \in \Sigma$, $a \in \Sigma$

3.2.1 Drzewa wywodów

Wywód to ciąg form zdaniowych $\alpha_0, .., \alpha_n$ takich, że $\alpha_i \to_G \alpha_{i+1}$, $\alpha_0 = S$, $\alpha_n = w \in \Sigma$

Wywód lewostronny to taki wywód, w którym jeśli $a_i = xA\beta_i$ to $a_{i+1} = x\gamma\beta_i$, czyli taki, w którym zawsze rozwijamy skrajnie lewy terminal

Drzewo wywodu to ukorzenione drzewo z porządkiem na dzieciach, w którym

- każdy wierzchołek ma etykietę z $\Sigma \cup N \cup \{\epsilon\}$
- etykieta korzenia to S
- Jeśli wierzchołek ma etykietę $A \in N$ a jego dzieci $X_1, ..., X_n$ to $(a, X_1, ..., X_{n)} \in P$

Wywód lewostronny otrzymujemy przechodząc DFSsem odwiedzając dzieci od lewej do prawej.

Gramatyka G jest jednoznaczna jeśli dla każdego słowa $w \in L(G)$ istnieje dokładnie jedno drzewo wywodu (dokładnie jeden wywód lewostronny)

3.3 Postać normalna Greibach

Twierdzenie 6. *Temat spoza listy wymagań egzaminacyjnych*

Gramatyka G jest w **postaci normalnej Greibach** jeśli produkcja jest w postaci

$$A \to \alpha B_1 ... B_n \tag{3.5}$$

gdzie $\alpha \in \Sigma, B_i \in N, n \in \mathbb{N}$

Dopuszczamy produkcję $S \to \epsilon$ jeśli $\epsilon \in L(G)$, ale wtedy S nie występuje po prawej stronie żadnej produkcji

3.4 Postać normalna Chomsky'ego

Symbol $A \in N \cup \Sigma$ jest **generujący** jeśli:

$$\exists_{w \in \Sigma} A \to_G w \tag{3.6}$$

Symbol $A \in N \cup \Sigma$ jest **osiągalny** jeśli:

$$\exists_{\alpha,\beta \in (N \cup \Sigma)} \to \alpha A \beta \tag{3.7}$$

Mówimy że symbol $A \in N \cup \Sigma$ jest **użyteczny** jeśli jest osiągalny i generujący

Gramatyka jest w **postaci normalnej Chomsky'ego** jeśli każda produkcja jest w jednej z następujących postaci

- $A \to BC$, gdzie $B, C \in N$
- $A \to a$, gdzie $A \in N$ oraz $a \in \Sigma$

Ponadto dopuszczamy, by dla stanu startowego S istniała produkcja $S \to \epsilon$ (jeśli gramatyka akceptuje słowo puste), ale wówczas S nie może znaleźć po prawej stronie żadnej produkcji.

Twierdzenie 7. *Dla każdej gramatyki bezkontekstowej G istnieje gramatyka bezkontekstowa słowa G' w postaci normalnej Chomsky'ego taka, że $L(G) = L(G')$*

3.4.1 Dowód

Po prostu pokaże algorytm konwersji

1. Jeśli nieterminal S jest gdzieś po prawej stronie to robimy sztuczny S' i przejście $S' \to S$

2. Usuwamy epsilonowe przejścia, jeśli mamy $A \to \epsilon$ to możemy po prostu we wszystkich produkcjach, które mają A po prawej stronie podmienić A na epsilon robiąc dodatkowe wersje tych produkcji.

3. usuwamy przejścia na jeden nieterminal $A \to B$, po prostu będziemy robić dodatkowe wersję każdej rzeczy, w której jest ten nieterminal z A zamiast B.

4. Zastępujemy każdą produkcję typu $A \to B_1 \dots B_n$ przez $A \to B_1 C$, i dodajemy $C \to B_2 \dots B_n$ i dalej rekurencyjnie póki zostało nam więcej niż 2

5. Jeśli mamy produkcje typu $A \to aB$ to zamieniamy sobie to na $A \to XB$ i $X \to A$.

3.5 Algorytm CYK

Algorytm CYK służy do sprawdzania czy słowo należy do Definicja Gramatyki bezkontekstowej—języka bezkontekstowego przedstawionego w Postać normalna Chomsky'ego—postaci normalnej Chomsky'ego. Działa w czasie $O(n^3 \cdot |G|)$,

3.5.1 Pseudokod

Tworzymy sobie tablice Ti,j,xdla i,j$\in 1, \dots, n przebiega wszystkie nieterminale$.

Dla każdego znaku a na pozycji i oraz dla każdego X takiego, że w gramatyce jest produkcja $X \to a$ ustawiamy w tablicy Ti,1,x= 1

```
Dla każdej długości i od 2 do n:
Dla każdej długości i od 2 do n:
```

```
Dla każdego początku j od 1 do n-i+1:
Dla każdego początku j od 1 do n-i+1:

Dla każdego podziału k od 1 do i-1:
Dla każdego podziału k od 1 do i-1:

Jeśli w tablicy są ustawione T[j,k,X] i T[j+k,i-k,Y], a w gramatyce mamy produkcję
Jeśli w tablicy są ustawione Tj,k,X i Tj+k,i-k,Y, a w gramatyce mamy produkcję

 Z → XY to ustawiamy T[j,i,Z]=1
 Z → XY to ustawiamy Tj,i,Z=1
```

Słowo należy do języka, jeśli T1,n,S= 1

3.6 Automat Büchiego

Twierdzenie 8. *Temat spoza zagadnień egzaminacyjnych*

Słowo nieskończone nad skończonym alfabetem Σ to funckja $\mathbb{N} \rightarrow \Sigma$ zapisana zwyczajowa jako nieskończony ciąg liter z Σ. Zbiór wszystkich słów nieskończonych nas Σ zwany również ω-słowami oznaczamy przez Σ^{ω}

Skończony **automat Büchiego** A dla słów nieskończonych to piątka $A = (Q, \Sigma, T, Q_0, F)$ gdzie:

- Q jest skończonym zbiorem stanów
- Σ skończonym alfabetem
- $\delta : Q \times \Sigma \rightarrow P(Q)$ jest funkcją przejścia
- $q_0 \in Q$ jest stanem startowym
- $F \subseteq Q$ zbiorem stanów akceptujących

Dla danego nieskończonego słowa $s = s_0 s_1 ...$ gdzie $s_i \in \Sigma$ dla każdego $i \in \mathbb{N}$ przebieg r BA A na s to nieskończony ciąg stanów $r = r_0 r_1 ...$ gdzie $R_0 \in Q_0$ i $r_{i+1} \in \delta(r_i, s_i)$ dla każdego $i \geqslant 0$. Dalej $\lim r = \{q | \{i | \{q = r_i\}|\} = \aleph_0\}$ czyli $\lim r$ zawiera te stany, które wustępują w przebiegu r nieskończenie wiele razy. Przebieg r jest akceptujący jeśli $\lim r \cap F \neq \emptyset$. BA A akceptuje slowo s jeśli istnieje akceptujący przebieg r automatu A na s.

3.7 Lemat o pompowaniu dla języków bezkontekstowych

Twierdzenie 9. *Jeżeli język L nad Σ jest bezkontekstowy to:*

$$\exists_{n_0 \in \mathbb{N}} \forall_{w \in L} \exists_{a,b,c,d,e \in \Sigma} : w = xuyvz \wedge |w| \geqslant n_0 \wedge |uyv| \leqslant n_0 \wedge |uv| \geqslant 1 : \forall_{i \in \mathbb{N}} x \cdot u^i \cdot y \cdot v^i \cdot z \in L \tag{3.8}$$

3.7.1 Dowód

Chcemy aby w derywacji naszego słowa w, które pompujemy dwa razy na jednej ścieżce pojawił się ten sam nieterminal A. Mamy wtedy $A \rightarrow_G c$ oraz $A \rightarrow_G bcd$. Możemy więc wstawić tą produkcję co doprowadza do bcd w miejsce tej co doprowadza do c, w ten sposób pompujemy *bcd* do *bbcdd*.

Niech $n = |N|$ to liczba dostępnych nieterminali oraz niech m będzie długością najdłuższej produkcji. Wtedy jeśli nasze słowo jest długości co najmniej $n_0 = m^{n+1} + 1$ to drzewo wywodu musi mieć wysokość co najmniej $n + 1$ a zatem jakiś nieterminal musi się powtórzyć na jakiejś ścieżce.

Aby zapewnić warunek $|bcd| \leqslant n_0$ z tezy bierzemy ten nieterminal, którego niższe wystąpienie jest najwyżej ze wszystkich, znajduje się na głębokości co najwyżej $n + 1$.

Może się zdarzyć tak, że wybrane tak $|bd| = 0$ to znaczy że przeszliśmy ścieżką, która nic nie produkuje. Szukamy wtedy innego spełniającego warunek nieterminal, musi taki być bo słowo jest długie.

3.7.2 Przykład

$L = \{a^n b^n c^n : n \in \mathbb{N}\}$ nie jest bezkontekstowy

- Bierzemy $n \geqslant 1$
- podajemy słowo $w = a^n b^n c^n \in L$
- Dostajemy podział $a^n b^n c^n = xuyvz$, w którym $|uyv| \leqslant n$ oraz $uv \neq \epsilon$ to znaczy, że uv jest niepuste i nie może mieć trzech różnych liter. Bez straty ogólności $\#a(uv) = 0$ oraz $\#_c uv \geqslant 1$
- Podajemy i=0, ponieważ: $\#_c(xyz) < \#(xuyvz) = \#_c w = \#_a w = \#_a w = \#_a(xuyvz) = \#_a(xyz)$ zatem $xu^i yv^i z \notin L$ dla $i = 0$

3.8 Lemat Ogdena

Lemat Ogdena to mocniejszy lemat o pompowaniu, który pozwala na wybranie miejsca, gdzie pompujemy,

Zaznaczenie słowa $w \in \Sigma$ to podzbiór $T \subseteq |w|$. Każde zaznaczenie słowa w indukuje zaznaczenie podsłowa słowa w. Dla zaznaczenia T słowa W oraz podsłowa v słowa w przez $|v|_t = |v|$ oznaczamy liczbę zaznaczonych pozycji w zaznaczeniu indukowanym przez T. W szczególności jeśli $T = |w|$ czyli wszystkie pozycje w w są zaznaczone to dla dowolnego podsłowa v słowa w mamy $|v|_T = |v|$.

Twierdzenie 10. *Lemat Ogdena*

$$\exists_{n_0 \in \mathbb{N}} \forall_{w \in L, T} \exists_{a,b,c,d,e \in \Sigma} : w = xuyvz \wedge |w|_T \geqslant n_0 \wedge |uyv|_T \leqslant n_0 \wedge |uv|_T \geqslant 1 : \forall_{i \in \mathbb{N}} x \cdot u^i \cdot y \cdot v^i \cdot z \in L \tag{3.9}$$

3.8.1 Dowód

Weźmy CFG - b w Postać normalna Chomsky'ego—postaci Chomsky'ego, że $L(b) = L$ oraz $w \in L, |w| \geqslant 2^{k+1} = p$,$k = \#$nieterminaliwb.

Oznaczmy sobie w dowolny sposób p liter w w.

Ponieważ mamy postać Chomsky'ego to za każdym razem dzielimy na dwa. Więc aby uzyskać 2^{k+1} długość słowa musieliśmy zrobić drzewo, które ma przynajmniej długość $k+1$. Zaczynamy na szczycie tego drzewa (w nieterminali startowym) i za każdym razem schodzimy w dół wybierając to dziecko (zawsze są dwa), które ma pod soba więcej zaznaczonych rzeczy, więc zawsze schodząc krok w dół zmniejszamy liczbę zaznaczonych pod sobą maksymalnie dwukrotnie, więc po $k+1$ przejściach mamy maksymalnie jeden pod sobą i ponieważ były $k+1$ przejścia, a mamy tylko k nieterminali to, któryś musiał się powtórzyć.

Musimy mieć na drodze jakąś liczbę takich wierzchołków, że oba dzieci mają pod sobą coś zaznaczonego, jeśli cały czas tylko jedne by miały to nigdy by się nie zmniejszyło i było by w końcu więcej niż elementów.

Weźmy więc dwa powtórzenia nieterminala A, najniższe, takie że jego dzieci oba mają coś zaznaczone.

Przypisujemy wszystkim na lewo od górnego powtórzenia A - x, na prawo, z, poniżej górnego, na lewo od dolnego u, na prawo v, pod dolnym y.

1. uv ma przynajmniej 1 zaznaczony element

   tylko z jednego dziecka prowadzi ścieżka do y, a drugie ma też coś zaznaczone więc pod tym drugim jest coś z u albo v.

2. $|uyv| \leqslant p$dzięki temu, ze drzewo jest długość $k+1$, to jego liści jest maksymalnie $2^{k+1} = p \leqslant p$

3.8.2 A komu to potrzebne?

Tym razem rozważamy tylko istotne wierzchołki drzewa wyprowadzenia, te z których wychodzą co najmniej dwie różne krawędzie będące początkami ścieżek do liścia wyróżnionego. Istnieje ścieżka, na której są dwa istotne wierzchoklki o tej samej etykiecie (nieterminalu). Teraz drzewo "pączkuje" tak jak poprzednio.

3.9 Definicja automatu ze stosem (PDA)

Automat ze stosem (pushdown automaton) - *PDA* to

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_{0,}F) \tag{3.10}$$

gdzie:

- Q - zbiór stanów
- Σ - skończony alfabet słów
- Γ - skończony alfabet stosu
- $\delta : (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^{)}$ - funkcja przejścia
- q_0 - stan startowy
- Z_0 - stan startowy stosu
- $F \subseteq Q$ - zbiór stanów akceptujących

δ dla każdego stanu, litery i symbolu na szczycie stosu oddaje zbiór nowych możliwych stanów, wraz z odpowiednimi symbolami, które mają zostać dodane na stos po takiej operacji. Symbol na szczycie stosu jest usuwany podczas operacji, wiele nowych symboli może być dodanych.

Jeśli opróżnimy stos to jest przypał, automat się zacina.

δ zwraca zbiór, a więc z natury *PDA jest niedeterministyczny*

3.9.1 Konfiguracja PDA

Konfiguracja PDA to trójka

$$(q, w, \gamma) \tag{3.11}$$

gdzie:

- q - aktualny stan

- w - część słowa pozostała do przeczytania
- γ - wszystkie symbole na stosie

3.9.1.1 Relacja $\vdash_P$

Dla konfiguracji PDA definiujemy relację $\vdash_P$

$$(q, aw, X\beta) \vdash_P (p, w, \alpha\beta) \Leftrightarrow (p, \alpha) \in \delta(q, a, X) \tag{3.12}$$

Intuicja $\vdash_P$ jest podobna jak $\rightarrow_G$ dla gramatyki bezkontekstowej. Analogicznie też definiujemy zwrotne i przechodnie domknięcie $\vdash_P$ opisujące, które konfigurację są osiągalne z których.

3.9.2

Twierdzenie 11. *Dla PDA P jeśli*

$$(q, x, a) \vdash_P (p, y, \beta) \tag{3.13}$$

to

$$(q, xw, a\gamma) \vdash_P (p, yw, \beta\gamma) \tag{3.14}$$

3.9.2.1 Dowód

gdziedowód

3.9.3 Akceptacja

3.9.3.1 Akceptacja stanem akceptującym

$$L(P) = \{w | (q_0, w, Z_{0)} \vdash_P (q_F, \varepsilon, \gamma) \wedge q_{f\in} F\} \tag{3.15}$$

3.9.3.2 Akceptacja pustym stosem

$$N(P) = \{w|(q_0, w, Z_0) \vdash_P (q, \varepsilon, \varepsilon)\} \tag{3.16}$$

3.9.4 Stos

Służy aby zmienne lokalne funkcji po prostu wrzucić do pamięci, to zwyczajny obszar w ramie. mamy operacje push i pop. call, wrzuca na stos adres następnej instrukcji i robi skok do funkcji, ret wyciąga tą zestosowaną instrukcje po zakończeniu funkcji

3.10 Równoważność wariantów (PDA)

Twierdzenie 12. *Dla języka L następujące twierdzenia są równoważne*
Istnieje P, taki że L(P)=L
Istnieje Q, taki że N(Q)=L

3.10.1 Dowód

3.10.1.1 Z L do N

3.11 Równoważność PDA i CFG

Niech L będzie językiem, następujące twierdzenia są równoważne

1. Istnieje gramatyka bezkontekstowa G, że $L(G) = L$
2. Istnieje automat ze stosem P, że $L(P) = L$
3. Istnieje automat ze stosem P', że $N(P') = L$

3.11.1 Dowód

3.11.1.1 Konwersja PDA do CFG

Definiujemy nieterminale jako

$$N = \{S\} \cup \{(q, z, p) : q, p \in Q, z \in \Gamma\} \tag{3.17}$$

Taki nieterminal symuluje słowa typu

$$(q, w, z\gamma) \vdash_P (p, \epsilon, \gamma) \tag{3.18}$$

Dla każdego $p \in Q$ dodajemy produkcje

$$S \to (q_0, Z_0, p) \tag{3.19}$$

Ponadto jeśli dla $a \in \Sigma$ lub $a = \epsilon$ mamy

$$(r, z_1, \ldots, z_n) \in \gamma(q, a, z) \tag{3.20}$$

to dodoajemu d;a wszystkich r_1,

Na początek wrzucamy na stos symbol końca stosu $\perp$, po czym przechodzi do stanu Q_{loop}, jeśli na szycie stostu

3.12 Zamkniętość CFL na operacje

todo

3.13 Wybrane problemy decyzyjne dla CFL

todo

Rozdział 4

Języki kontekstowe

4.1 Gramatyki Kontekstowe

4.1.1 Gramatyki typu 0

Gramatyki typu 0 to $G = (N, \Sigma, P, S)$, w których każda produkcja jest postaci

$$\alpha A \beta \rightarrow a \gamma \beta \tag{4.1}$$

gdzie $\alpha, \beta, \gamma \in (\Sigma \cup N), A \in N$

4.1.2 Gramatyki kontekstowe (CSG)

Gramatykę nazywamy kontekstową (ang. context sensitive grammar), jeśli każda produkcja w definicji gramatyki typu 0 spełnia dodatkowo $|\gamma| \geqslant 1$. Poza tym dopuszczamy produkcje postaci $S \rightarrow \epsilon$ jeśli S nie występuje po prawej stronie żadnej produkcji.

4.1.3 Gramatyki monotoniczne

Gramatyka monotoniczna to gramatyka, w której każda produkcja jest postaci

- $\alpha \rightarrow B$, gdzie $\alpha, \beta \in (\Sigma \cup N)$, $|\alpha| \leqslant |\beta|$ lub

- $S \to \epsilon$ pod warunkiem, że S nie jest po prawej stronie żadnej produkcji

4.2 Automat ograniczony liniowo (LBA)

Automat ograniczony liniowo to taka Maszyna Turinga, której taśma jest ograniczona przez długość inputu.

$$LBA = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, \vdash, \dashv, F) \tag{4.2}$$

gdzie

- Q - skończony zbiór stanów
- $\Sigma \subseteq \Gamma$ - skończony alfabet wejściowy
- Γ - skończony alfabet taśmowy
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 1\}$, przy czym nie możemy nigdy podmienić ograniczników taśmy
- q_0 -stan startowy
- $\sqcup \in F \backslash \Sigma$ - pusty symbol, domyślny symbol taśmy
- F - zbiór stanów akceptujących

  Ponadto $Q \cap \Gamma = \emptyset$

Rozdział 5

Maszyny Turinga

5.1 Teza Churcha

Funkcja matematyczna z liczb naturalnych może być policzona za pomocą efektywnej metody (czyli w pełni zaautomatyzowanej) wtedy i tylko wtedy, gdy może być obliczona przez maszynę Turinga.

W XXw. powstały 3 sformalizowane *modele obliczeń* do obliczania funkcji obliczalnych

1. Rachunek Lambda
2. Funkcje pierwotnie rekurencyjnie
3. Maszyna Turinga—Maszyny Turinga

Okazuje się, że modele te są równoważne, każda funkcja obliczalna jest zarówno λ-obliczalna jak i pierwotnie rekurencyjna jak i obliczalna przez maszynę Turinga.

5.2 Maszyna Turinga

5.2.1 Definicja

Maszyna Turinga to

$$MT = (Q, \Sigma, \Gamma, \delta, q + 0, \sqcup, F) \tag{5.1}$$

gdzie:

- Q - skończony zbiór stanów
- $\Sigma \subseteq \Gamma$ - skończony alfabet wejściowy
- Γ - skończony alfabet taśmowy
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{-1, +1\}$ - stan, litera -¿ nowy stan, zmiana litery, ruch głowicą na taśmie (lewo prawo)
- q_0 - stan startowy
- $\sqcup \in \Gamma \backslash \Sigma$ - pusty symbol / zero (domyślny symbol taśmy)
- F - zbiór stanów akceptujących

  Ponadto $Q \cap \Gamma = \emptyset$

5.2.2 Opis chwilowy (konfiguracja)

Konfiguracja Maszyny Turinga to słowa nad językiem $\Gamma^q \Gamma\Gamma$

Konfiguracja startowa Maszyny Turinga to słowo postaci $q_0 w \sqcup$

5.2.3 Relacja $\vdash_M$

todo

5.2.4 Język akceptowany przez Maszynę Turinga

$$L(MT) = \{w \in \Sigma | \exists_{q_f \in F} \exists_{\alpha,\beta \in \Gamma} q_{0w\sqcup} \vdash_{M^\alpha} q_f \beta\} \tag{5.2}$$

5.2.5 Twierdzenie Wrony

Maszyny Turinga nie wolno nazywać „automatem". Nazwanie Maszyny Turinga „automatem" skutkuje skreśleniem z listy studentów.

5.2.6 K-taśmowa maszyna

k-taśmowa maszyna Turinga jest równoważna zwykłej, gdyż możemy po prostu trzymać wszystkie taśmy na jednej taśmie.

5.2.7 Równoważność z PDA z k stosami

Jeżeli dodamy do PDA drugi stos to jest on równoważny Maszynie Turinga, pierwszego będziemy używać jako taśmy, drugiego jako bufora do czytania pierwszego

5.3 Niedeterministyczna Maszyna Turinga

Niedeterministyczną Maszynę Turinga definiujemy tak samo jak deterministyczną Maszyna Turinga—Maszynę Turinga, przy czym funkcja δ zostaje relacją; ciągnie to za sobą dosyć oczywiste konsekwencje w definiowaniu relacji $\vdash$.

Co ciekawe, definicja języka akceptowanego się nie zmienia; nadal zwyczajnie chcemy mieć stan akceptujący w domknięciu przechodnim.

5.3.1 Równoważność z deterministyczną maszyną Turinga

Dla dowolnego języka $L \subset \Sigma$ następujące twierdzenia są równoważne

1. Istnieje deterministyczna Maszyna Turinga DM taka, że $L(DM) = L$
2. Istnieje niedeterministyczna Maszyna Turinga NM taka, że $L(NM) = L$

5.3.1.1 Dowód

Aby stworzyć deterministyczną maszynę z niedeterministycznej chodzimy BFS-em po nieskończonym grafie konfiguracji maszyny N.

5.4 Uniwersalna Maszyna Turinga

Uniwersalna maszyna turinga to maszyna Turinga, która umie i może zasymulować inną maszynę Turinga, dostaje na wejściu zakodowaną maszynę i jej wejście i zwróćić wynik.

5.4.1 Kodowanie Maszyny Turinga

Pokaże jak zakodować $M = (Q, \Sigma, \gamma, \delta, q_0, \sqcup, F)$ nad alfabetem $\Sigma_\sqcup = \{0, 1\}$

- $q_i \in Q$ kodujemy unarnie jako ciąg i jedynek
- $\sqcup$ kodujemy jako pojedyńczą jedynkę
- $a_i \in \Sigma$ kodujemy jako ciąg $i + 1$ jedynek
- $z_i \in \Gamma$ kodujemy jako ciąg $i + |\Sigma + 1|$ jedynek, dla odróżnienia od Σ
- δ jest funkcją, a więc zbiorem par $((q_1, z_1), (q_2, z_2, \pm 1))$, każdy element z pary $((q_i z_j), (q_k, z_l, 1))$ kodujemy jako

$$1^i 0 1^j 0 1^k 0 1^l 0 1 \tag{5.3}$$

  a z pary $((q_i z_j), (q_k, z_l, 0))$ jako

$$1^i 0 1^j 0 1^k 0 1^l 0 1 1 \tag{5.4}$$

- Całą deltę kodujemy kodując wszystkie jej pary (w dowolnej kolejności) i oddzielając je dwoma zerami
- $F = \{q_{i_1}, ..., q_{i_n}\}$ kodujemy jako

$$1^{i_1} 0 ... 0 1^{i_n} \tag{5.5}$$

  Całą maszynę zapisujemy wpisując δ, q_0, F oddzielając je ciągiem 000

Słowo w=$a_{i_1}, .., a_{i_n}$ kodujemy jako

$$1^{i_1}0...01^{i_n} \tag{5.6}$$

Konfiguracje maszyny $X_1...X_k q_i X_{k+1}...X_n$ kodujemy jako

5.4.1.1

0 - kod X_1 - 0 - kod X_2 - 0 - kod X_k - 00 - kod X_{k+1} - 0 - kod X_n - 0

Zakładamy, że kodowanie M i aktualna konfiguracja są zapisane na osobnych taśmach.

5.4.2 Krok symulacji

1. Zlokalizuj głowicę M (szukamy 00)
2. Znajdź w kodowaniu δ wpis odpowiadający przejściu na $q_i X_k$
3. Wykonaj zadane przejście, możemy wpisać nową konfigurację na osobną tymczasową tame i przepisać ją z powrotem na taśmę operacyjną

5.5 Języki rekurencyjne i rekurencyjnie przeliczalne

5.5.1 Języki rekurencyjne przeliczalne (RE)

Twierdzenie 13. *Język jest* rekurencyjnie przeliczalny *jeśli istnieje Maszyna Turinga M, taka że* $L(M) = L$

5.5.2 Języki rekurencyjne (R)

Powyższa definicja działa zarówno w przypadku deterministycznym (w którym mamy jedną ścieżkę obliczeń) jak i niedeterministycznym (wtedy chcemy żeby każda ścieżka obliczeń miała taką własność). Niestety formalnie Maszyna Turinga się nie zatrzymuje, ale przyjmujemy że jak już wpadniemy w stan q_{accept} to akceptujemy

słowo i się zatrzymujemy a gdy wpadniemy w stan q_{reject} to odrzucamy słowo i się zatrzymujemy.

Twierdzenie 14. *Język L jest* rekurencyjny / rozsztrzygalny *jeśli istnieje Maszyna Turinga z własnością stopu M, taka że* $L(M) = L$

5.6 Funkcje częściowe

Graf funkcji f to zbiór napisów postaci: x, f (x)

$$, gdziexorazwartoffnax(czylif(x))zapisanesbinarnie.Mwimy, emaszynaTuringaobliczafunkcj(za \tag{5.7}$$

na taśmie zawsze kończy działanie i zatrzymując się na taśmie napisane jest jedynie $f(x)$.

Mówimy, że maszyna Turinga *oblicza funkcję częściową* f jeśli zaczynając z dowolnym słowem wejściowym x na taśmie kończy działanie jedynie z $f(x)$ zapisanym na taśmie zawsze jeśli wartość $f(x)$ jest określona przez f .

5.7 Zamkniętość R i RE na operacje

5.7.1 Konkatenacja

$$\exists A, B, M_A, M_B : L(M_A) = A \wedge L(M_B = B) \wedge A + B = w \tag{5.8}$$

możemy sobie po prostu podzielić na różne sposoby input na dwie części i symulować dwie maszyny na raz, wszystkie podziały na raz.

5.7.2 Gwiazda Kleenego

Symulujemy M_A na inpucie, jeśli A jest w stanie akceptującym to niedeterministycznie albo idziemy normalnie dalej, albo zwracamy ok.

5.7.3 Suma teoriomnogościowa

Po prostu symulujemy dwie maszyny na raz, zwracamy ok. jeśli którakolwiek w akceptującym.

5.7.4 Przekrój

Po prostu symulujemy dwie maszyny na raz, zwracamy ok. jeśli obie w akceptującym.

5.7.5 Dopełnienie

R jest zamknięte na dopełnienie, po prostu robimy sobie automat, który gdy poprzedni zwracał tak, zwraca nie i odwrotnie (mamy pewność, że tamten się skończył).

RE nie jest zamknięte na dopełnienie, jeśli by tak było moglibyśmy iść po obu na raz i zawsze by się skończyło.

5.7.6 Homomorfizm

Języki rozstrzygalne nie są zamknięte na homomorfizm, pokażę przykład

$$L = \{xy | x \in \{0,1\}, y \in \{a,b\}, x =< M, w >\} \tag{5.9}$$

y koduje liczbę naturalną, tak że M kończy się na w po n krokach.
L jest rozsztrzygalne, mogę po prostu zasymulować M przez n kroków
Zrobię homomorfizm $h : h(0) = 0, h(1) = 1, h(a) = h(b) = \epsilon$.
po homomorfizmie problem staje się po prostu LHALTem.

5.8 Enumeratory

Enumerator dla zbioru $S \subseteq N$, gdzie N to zbiór liczb naturalnych z zerem, to maszyna Turinga M , która nigdy nie kończy działania i (w dowolnej kolejności)

wypisuje na jednej z taśm: $10^{i_1}10^{i_2}$, gdzie $i_1, i_2, \cdots \in S$ i dla każdego $n \in S$ napis 0^n pojawia się w pewnym momencie działanie M na taśmie.

Twierdzenie 15. *Podzbiór* $\mathbb{N}$ *jest rekurencyjnie przeliczalny wtw. gdy istnieje dla niego enumerator*

Twierdzenie 16. *Każdy podzbiór* $\mathbb{N}$ *jest rekurencyjny wtw. gdy istnieje dla niego iterator wypisujący elementy w porządku leksykograficznym*

Używając enumeratora możemy po prostu sprawdzać po kolei czy słowo dane na wejściu zostało zwrócone przez iterator, w leksykograficznym możemy ponadto sprawdzać czy już nie jesteśmy dalej niż zadane słowo, wtedy zwracamy fałsz.

Rozdział 6

Rozstrzygalność

6.1 Język przekątniowy

Język przekątniowy jest przykładem języka, który nie jest rekurencyjnie przeliczalny.

$$L_d = \{w_i | w_i \notin L(M_i\} \tag{6.1}$$

czyli jest to język słów nie akceptowane przez żadną maszynę Turinga

6.1.1 Dowód nierozsztrzygalności

Analogiczny do Paradoks fryzjera—paradoksu fryzjera

Jeśli L_d jest rozsztrzygalny to istnieje M, że $L(M) = L_d$. Ma ona jakiś indeks k spośród przeliczalnie wielu maszyn turinga. Rozważmy co się dzieje ze słowem w_k

- jeśli $w_k \in L_d$ to $w_k \in L(M_k)$, ale to znaczy, że $w_k \notin L_d$ co prowadzi do sprzeczności
- jeśli $w_k \notin L_d$ to $w_k \notin L(M_k)$, więc $w_k \in L_d$ co prowadzi do sprzeczności

  Tak czy siak sprzeczność, więc $L_d \notin RE$

6.2 Język uniwersalny

Definiujemy **język uniwersalny** jako

$$L_u = \{(M, w) : w \in L(M\} \tag{6.2}$$

Nazwa się wzięła stąd, że zawiera wszystkie pary (maszyna,słowo), że maszyna akceptuje określone słowo

Twierdzenie 17.

$$L_u \in RE \wedge L_u \notin R \tag{6.3}$$

6.2.1 Dowód

Trywialnie $L_u \in RE$ bo możemy zasymulować maszynę M na słowie w na Uniwersalna Maszyna Turinga—Uniwersalnej Maszynie Turinga.

6.2.2 $L_u \notin R$

Załóżmy nie wprost, że mamy jakąś maszynę M z wlaśnością stopu, że $L(M) = L_u$
Konstruujemy maszynę M', ktora:

- wczytuje wejście x
- symuluje M na (A,x)
- neguje wejście M - możemy to zrobić bo M ma własność stopu

Dajemy maszynie M' na wejściu w, mamy teraz dwie opcje

- M' zaakceptowała w

$$w \in L(M') \Rightarrow (A, w) \notin L_u \Rightarrow w \notin L(A) \Rightarrow W \in L_d \tag{6.4}$$

- M' nie zaakceptowała w

$$w \notin L(M') \Rightarrow (A, w) \in L_u \Rightarrow w \in L(A) \Rightarrow W \notin L_d \tag{6.5}$$

Tak czy siak lekki przypał bo $L(M') = L_d \notin RE$ więc nie może istnieć maszyna M z wlasnością stopu, czyli $L_u \notin R$

6.3 Redukcja Turinga

Redukcja z języka L_1 do języka L_2 to funkcja $f : \Sigma \to \Sigma$ *obliczana przez maszynę Turinga,* taka że $x \in L_1 \Leftrightarrow L(f(x)) \in L_2$

Jeśli istnieje redukcja z L_1 do L_2 to zapisujemy to jako $L_1 \leqslant L_2$ i mówimy, że L_1 jest prostszy od L_2, L_1 redukuje się do L_2

6.4 Twierdzenie Rice'a

Nieformalnie **twierdzenie Rice** mówi, że

Twierdzenie 18. *Twierdzenie Rice'a Nieformalnie*
Własność P nietrywialna to taka, że $P \neq \emptyset \wedge P \neq RE$
Każda nietrywialna własność RE jest nierozstrzygalna

Twierdzenie 19. *Twierdzenie Rice'a Formalnie*

$$\forall_{P \subseteq RE}((P \neq \emptyset \wedge P \neq RE) \Rightarrow \{M : L(M) \in P\} \notin R) \tag{6.6}$$

6.4.1 Dowód

Niech P będzie dowolną nietrywialną własnością oraz niech $L_P = \{M : L(M) \in P\}$

Języki rekurencyjne są Zamkniętość R i RE na operacje—zamknięte na dopełnienie, więc weźmy sobie $\overline{L_P} \in R \Leftrightarrow L_P \in R$. Przyjmujemy ponadto, że $\emptyset \notin P$.

Zrobimy Redukcja Turinga z problemu stopu, weźmy N, $L(N) = L_P$, zakładamy nie wprost, że jest rekurencyjny, weźmy sobie także $(M, w) \in L_{HALT}$.

Zdefiniujmy sobie następującą maszynę M'

```
M'(M, N, w, x){
M'(M, N, w, x){

symuluje M na w
symuluje M na w

symuluje N na x i zwraca wynik
symuluje N na x i zwraca wynik

}
}
```

Trzeba teraz pokazać, że $(M, w) \in L_{HALT} \Leftrightarrow M^{'} \in L_P$.

- $\Rightarrow$

  Skoro $(M, w) \in L_{HALT}$ to skończy się symulacja w na M, więc skończy się symulacja N na x i zwróci się wynik tej operacji, więc ok.

- $\Leftarrow$

  Skoro M się nigdy nie kończy to N nigdy nie zaakceptuje, więc $L(M') = \emptyset$, ale założyliśmy, że $\emptyset \notin P$, więc $M' \notin L_P$.

Rozdział 7

Złożoność obliczeniowa

7.1 Podstawowe klasy złożoności obliczeniowej

7.1.1 Czas $T(N)$

Maszyna Turinga M działa w czasie $T : \mathbb{N} \to \mathbb{N}$ jeśli dostawszy input w o długości n zatrzymuje się po co najwyżej $T(n)$ przejściach w stanie akceptującym lub odrzucającym.

7.1.2 Klasa P

Problemy w klasie P - *polynomial* - są rozwiązywalne przez Deterministyczną maszynę Turinga. Funkcja $T(N)$ jest dla nich wielomianowa.

7.1.3 Klasa NP

Problemy w klasie NP - *non polynomial* są rozwiązywalne przez Niedeterministyczną maszynę Turinga

7.1.4 Redukcje wielomianowe

Jeśli możemy jakiś problem przekształcić w wielomianowym czasie w jakiś inny problem używając funkcji f to mówimy, że f jest *redukcją wielomianową (Karpa).*

Jeśli istnieje redukcja wielomianowa z języka L_1 do L_2 to zapisujemy $L_1 \leqslant_p L_2$

7.1.5 Problemy NP zupełne

Problemy NP Zupełne to takie, które są conajmniej tak trudne jak jakikolwiek inny problem w NP. Czyli są w NP, $L_1 \in NP$ i $\forall L_2 \in NP : L_2 \leqslant_p L_1$

7.1.6 P=NP?

Generalnie nadal nie mamy zielonego pojęcia czy $P < NP$, może się okazać, że $P = NP$. Ale w dowodach zwykle przyjmujemy, że jeśli udowodnimy, że jakiś problem jest NP zupełny to nie należy do P.

Co ciekawe, jeśli udałoby się udowodnić, że jakikolwiek problem NP zupełny jest w P to wszystkie problemy z NP są w P, a więc $P = NP$.

7.1.7 Problemy NP trudne

Jeśli umiemy udowodnić, że istnieje redukcja wielomianowa każdego problemu w NP w dany problem L, ale nie umiemy udowodnić, że problem $L \in NP$ to mówimy, że L jest NP trudny

7.1.8 Problemy co-NP

To problemy, których dopełnienie jest rozwiązywalne w czasie NP

7.1.9 Klasa PSPACE

Problemy wykorzystujące wielomianową przestrzeń

7.1.10 Klasa EXPTIME

Problemy wykorzystujące niewielomianową przestrzeń

7.2 Problem SAT i twierdzenie Cooka

Problem SAT jest problem Podstawowe klasy złożoności obliczeniowej—NP zupełnym, dostając dowolną Formuła Boolowska—formułę boolowską określa czy jest

ona spełnialna czy nie.

7.2.1 Twierdzenie Cooka-Levina

Twierdzenie 20. *Problem SAT jest NP zupełny*

Jest to turbo ważne twierdzenie, gdyż dzięki niemu SAT stał się pierwszym problemem, o którym wiemy, że jest NP zupełny. Dzięki temu możemy redukować wielomianowo inne języki do SATa, aby pokazać, że także one są NP zupełne.

7.2.2 Dowód

7.2.2.1 $SAT \in NP$

Ta część jest prosta, gdyż mając dobrane wartości zmiennych formuły boolowskiej można w czasie $O(n)$ sprawdzić czy jest spełniona. Żeby rozwiązać w NP, a więc na Niedeterministyczna Maszyna Turinga—Niedeterministycznej Maszynie Turinga możemy więc po prostu niedeterministycznie wybrać wartości zmiennych, które spełniają.

7.2.2.2 $\forall A \in NP : A \leqslant_p SAT$

Oj trudne się wylosowało...

No dobra,

Niech $A \in NP$ będzie rozstrzygalne przez NTM - M w czasie n^k, pokażemy redukcje z A do SAT.

M działa w czasie n^k, tworzymy więc tablicę rozmiaru $n^k \times n^k$. Tablica ta w każdym wierszu ma konfiguracje maszyny, a kolejne wiersze reprezentują kolejne konfiguracje w procesie dojścia do stanu akceptującego.

Skonstruujemy formułę logiczną, która korzystając z tak zdefiniowanej tabeli będzie opisywała poprawne przejścia maszyny.

Aby jakoś móc kodować tą tabelę to stworzymy sobie bardzo dużo zmiennych boolowskich typu x_{ija}, oznaczające, że w wierszu i, kolumnie j znajduje się symbol a

Aby nasza formuła ϕ była wartościowana na prawdę to spełnione muszą być 4 intuicyjne warunki, z których stworzymy sobie podformuły $\phi_1, \phi_2, \phi_3, \phi_4$, takie że

$$\phi = \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \tag{7.1}$$

1. *Wartości zmienne są spójne* - czyli w żadnej kratce tabeli nie ma dwóch zmiennych, czyli

   $$\neg\exists i, j, a, b : x_{ija} = 1 \wedge x_{ijb} = 1 \tag{7.2}$$

   oraz w każdej kratce tabeli jest jakiś symbol (blank też jest symbolem), czyli

   $$\forall i, j : \exists a : x_{ija} = 1 \tag{7.3}$$

   Zatem ϕ_1 definiujemy jako

   $$\phi_1 = \bigwedge_{i,j} A_{ij} \wedge B_{ij} \tag{7.4}$$

   gdzie

   $$A_{ij} = \bigvee_{a} x_{ija} \quad B_{ij} = \bigwedge_{a,b,a\neq b} \neg(x_{ija} \wedge x_{ijb}) \tag{7.5}$$

   Formuła A_{ij} wymaga aby coś było w każdej kratce, a B_{ij}, aby tylko jedno

2. *Konfiguracja startowa jest poprawnie określona.* Powinna być taka, że na pierwszym miejscu jest stan startowy, a potem mamy input, a za inputem aż do końca blanki. Także po prostu sprawdzamy sobie formułą ϕ_2

$$\phi_2 = x_{1,1,q_s} \wedge x_{1,2,w_1} \wedge x_{1,2,w_2} \wedge \cdots \wedge x_{1,|w|+1,\sqcup} \wedge x_{1,|w|+2,\sqcup} \wedge \cdots \wedge x_{1,n^k,\sqcup} \tag{7.6}$$

3. *NTM przeszła do stanu akceptującego*

   Jeśli NTM, akceptuje to, każde kolejne przejście pozostawia maszynę w stanie akceptującym, wystarczy więc sprawdzić czy coś w ostatnim wierszu jest stanem akceptującym, a więc ϕ_3 definiujemy jako

$$\phi_3 = \bigvee_{q_f \in F} \bigvee_{j} x_{n^k,j,q_f} \tag{7.7}$$

4. *Przejścia są poprawne*

   Tu trzeba wymachać

7.2.3 k-SAT

k-SAT to problem, który mając dane wyrażenie w CNF—k-CNF określa czy jest ono spełnialne.

Dla $k \geqslant 3$ k-SAT jest NP zupełny, dla $1SAT$ i $2SAT$ istnieją algorytmy liniowe (patrz ASD).

7.3 Problem Tautology

Problem Tautology dostając dowolną Formuła Boolowska—formułę boolowską odpowiada na pytanie, czy ta formuła jest tautologią.

Twierdzenie 21. *Problem Tautology jest coNP zupełny*

7.3.1 Dowód

Po prostu argument że wystarczy nam znaleźć jedną taką konfigurację, która nie spełnia, więc dopełnienie jest w NP więc problem jest w $coNP$.

- Tautology $\in coNPco$

  żeby pokazać, że Tautology jest $coNPco$ wystarczy pokazać, że $\overline{\text{Tautology}}$ jest NP zupełne. przeprowadzimy więc jego Redukcja Turinga—redukcję z Problem SAT i twierdzenie Cooka—SATa.

  Aby zredukować z SATa wystarczy po prostu zaprzeczyć formułę.

  Jeśli formuła jest spełnialna, to jej zaprzeczenie nie jest tautologią. Jeśli formuła nie jest tautologią to jej zaprzeczenie jest spełnialne.

Rozdział 8

Klasa PSPACE

8.1 Problemy wykorzystujące wielomianową przestrzeń

Klasa PSPACE definiuje problemy rozwiązywalne mając wielomianową przestrzeń, co oznacza, że rozwiązująca je maszyna Turinga zawsze odwiedzi wielomianową od rozmiaru inputu liczbę miejsc na taśmie.

Rozróżniamy $PSPACE$i $NPSPACE$w zależności czy rozważamy deterministyczną maszynę Turinga, czy nie, ale dowiedziemy że są równe.

8.1.1 Ograniczenie na liczbę ruchów maszyny przed akceptacją

Twierdzenie 22. *Jeśli M jest w PSPACE, z ograniczeniem $p(n)$ to istnieje stała c, że jeśli M akceptuje słowo w to, robi to w przeciągu $c^{1+p(n)}$ ruchów*

8.1.1.1 Dowód

8.1.2 Ograniczenie na liczbę ruchów maszyny akceptującej język

Twierdzenie 23. *Jeśli język jest w $PSPACE$ to L jest akceptowane przez jakąś maszynę Turinga o ograniczonej wielomianowo przestrzeni, która się zatrzymuje*

po maksymalnie $c^{q(n)}$ ruchach, gdzie q to jakiś wielomian, a c jakaś stała

8.1.2.1 Dowód

Możemy teraz złączyć dwie maszyny w maszynę M_3, która akeptuje jeśli M_1 akceptuje, a odrzuca jeśli M_1 lub M_2 odrzuca. M_3 zatem zatrzymuje się po maksymalnie $c^q(n)$ ruchach.

8.1.3 PSPACE kompletność

Analogicznie do NP kompletności, żeby problem był $PSPACE$ kompletny musi

1. być w $PSPACE$
2. wszystkie języki w $PSPACE$ redukują się w czasie wielomianowym do tego języka

8.2 Twierdzenie Savitcha

Co zaskakujące po męczarniach z rozróżnianiem między P a NP,

Twierdzenie 24. *Twierdzenie Savitcha*
$PSPACE$=$NPSPACE$

8.2.1 Dowód

Pokażemy zatem w drugą stronę, weźmy NTM - M w $NPSPACE$, ograniczone przez $p(n)$. Wiemy, że jeśli M akceptuje to maksymalnie w w maksymalnie $c^{1+p(|w)}$ ruchach. Stworzymy DTM - D, która akceptuje ten sam język jest w $PSPACE$.

Maszyna D korzysta z funkcji *reach*, która sprawdza czy N, może przejść między danymi konfiguracjami I oraz J w maksymalnie m ruchach. Sprawdza, rekursywnie czy w $m, \frac{m}{2}, \frac{m}{4}, \ldots$

```
Schemat funckji reach
```
```
Schemat funckji reach
```

```
reach(I,j,m):
reach(I,j,m):

if(m==1){
if(m==1){

if(i==J || I can become J in 1 move)return true
if(i==J || I can become J in 1 move)return true

else return false
else return false

}
}

else{
else{

for(K: all possible configurations){
for(K: all possible configurations){

if(reach(I,K,m/2) && reach(K,J,m/2))return true;
if(reach(I,K,m/2) && reach(K,J,m/2))return true;

return false
return false

}
}

}
}
```

To działa baaaaaaaaaardzo wolno, ale możemy pokazać, że ponieważ zawsze mamy odpalaną tylko jedną gałąź, to mamy odpalone co najwyżej $\log_2 m$ funckji, które zajmują oczywiście razem ograniczone wielomianowe miejsce.

A więc aby zasymulować N za pomocą D odpalamy po prostu po kolei $reach$ na wszystkich konfiguracjach startowych i końcowych, a za m dajemy $c^{1+p(n)}$. Każde wywołanie zajmuje $O(p(n))$ miejsca i każda konfiguracja zajmuje maksymalnie $O(p(n))$, więc maksymalnie zajmujemy $O(p(n)^2)$ czyli wielomianowo wiele.

8.3 QBF

QBF - Quantified boolean formulas to formuły boolowskie z dodanymi kwantyfikatorami $\forall$ i $\exists$. Aby uprościć rozumowania zakładamy, że żaden QBF nie ma dwóch wyrażeń, które mają dwóch kwalifikacji $\forall$ albo $\exists$ na tą samą zmienną x, nic to nie zmienia, ale upraszcza zapis i dowody. QBF ma wartość 1 gdy wyrażenie jest prawdziwe, 0 w.p.p.

Problem czy dany QBF jest prawdziwy jest $PSPACE$ zupełny.

Twierdzenie 25. *QBF jest w $PSPACE$*

8.3.1 Dowód

Aby ewaluować dany QBF - F zrobimy to rekurencyjnie

1. Jeśli mamy $F = F_1 + F_2$ to sprawdzamy najpierw jedno reukrencyjnie, jeśli zwróciło 1 to zwracamy 1, w.p.p. nadpisujemy używane przez poprzednie obliczenie miejsce i robimy to samo z drugim

2. Jeśli mamy $\exists x E$, to tworzymy sobie wyrażenie E_1 na podstawie E zamieniając każde wystąpienie x na 0 i rekurencyjnie ewaluujemy jeśli zwróciło 1 to zwracamy 1, w.p.p tworzymy E_2 analogicznie zamieniając x na 2.

3. Reszta w sumie analogicznie

   Znowu, nigdy się nie rozgałęziamy, więc koniec końców działamy w miejscu $O(n^2)$.

¿Twierdzenie

¿QBF jest $PSPACE$ zupełny

8.3.2 Dowód

xdddd

gdziedowód