Automaty i maszyna Turinga

Automat skończony (determ. i niedet.)

Automaty = (S, ∑, T, Q, A),

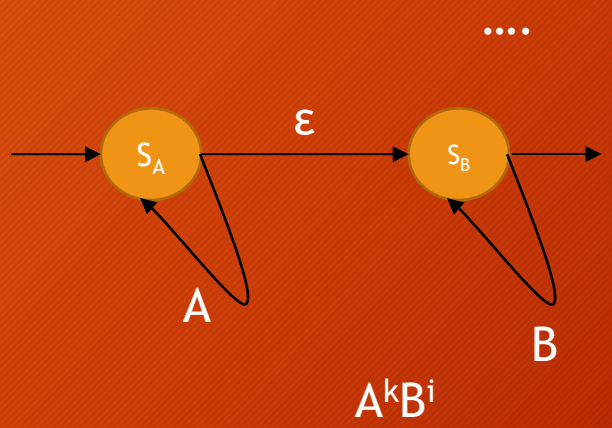
- S – stany
- ∑ - alfabet
- T - funkcja przejść S x ∑ → S (determin.) lub S x ∑ → P(S), gdzie [P(S) – zbiór wszystkich możliwych podzbiorów stanów zbioru S (niedetermin.)
- Q – zbiór stan początkowy
- A – zbiór stanów akceptujących

Automat ze stosem

Ang. Push-down automaton

Wstęp

- Automaty skończone rozpoznają tylko języki zwierające słowa (zdania) o powtarzalnej strukturze – vide lemat o pompowaniu.
- Ich pamięć jest ograniczona liczbą stanów
- W efekcie automat skończony nie rozpozna słów języka A^kB^k (np. AB, AAB,AABB, AAAABBBB, etc.)



Automat ze stosem (niedeterministyczny)

- Nieformalnie, automat różni się jedynie funkcją (relacją) przejścia, w której następny stan(y) zależy(ą) od:
  - Aktualnego stanu, wartości na wejściu i wartości na wierzchu stosu
  - Podczas przejścia można zapisać na stosie określony znak

Automat ze stosem – formalna definicja

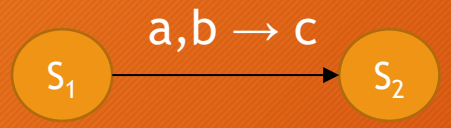
- Automat ze stosem PDA to szóstka uporządkowana: $(Q, \Sigma, \Gamma, \delta, q_0, F)$, gdzie
- Q, Σ, Γ i F zbiory skończone
- Q zbiór stanów,
- Σ alfabet wejściowy,
- Γ alfabet stosu (może być różny do wejściowego),
- δ funkcja (relacja) przejścia,
  - δ: $Q \times (\Sigma \cup \varepsilon) \times (\Gamma \cup \varepsilon) \rightarrow$ wszystkie podzbiory $Q \times (\Gamma \cup \varepsilon)$.
- $q_0 \in Q$ stan początkowy
- $F \subseteq Q$ zbiór stanów akceptujących

Rozumienie przejść

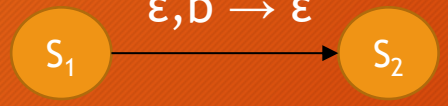
Dana jest wartość f-cji przejścia: $\delta(q_1, a, 1) = \{ (q_2, 0), (q_4, \varepsilon) \}$

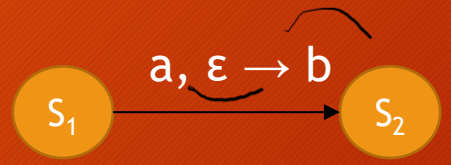
- Powyższe oznacza:
  - W stanie q_1, przeczytawszy symbol a, na wejściu i pobrawszy znak 1 ze stosu, idź do stanu q_2 zapisując zero na stosie lub q_4 nie zapisując nic na stosie.

Rozumienie przejść automatu ze stosem



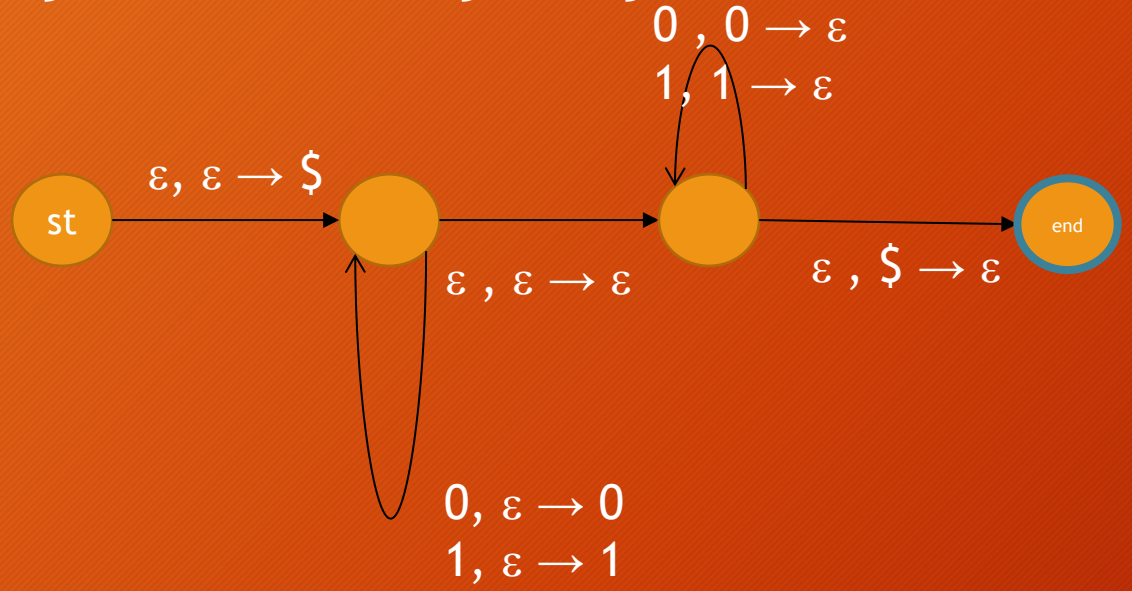
- W stanie s_1 jeśli przeczytasz z wejścia symbol „a” i podniesiesz ze stosu symbol „b” przejdź do stanu s_2 kładąc na stos symbol „c”
- ε – oznacza, że nie odczytujemy wejścia / nie pobieramy ze stosu lub nie kładziemy nic na stos





Automat ze stosem rozpoznający język złożony z palindromów o parzystej długości

- L={w w^R: w∈{0,1}*}, w^R – odwrócony łańcuch *w*
- Jak to ma działać – wrzucamy litery w na stos, a potem ściągamy je w odwrotnej kolejności



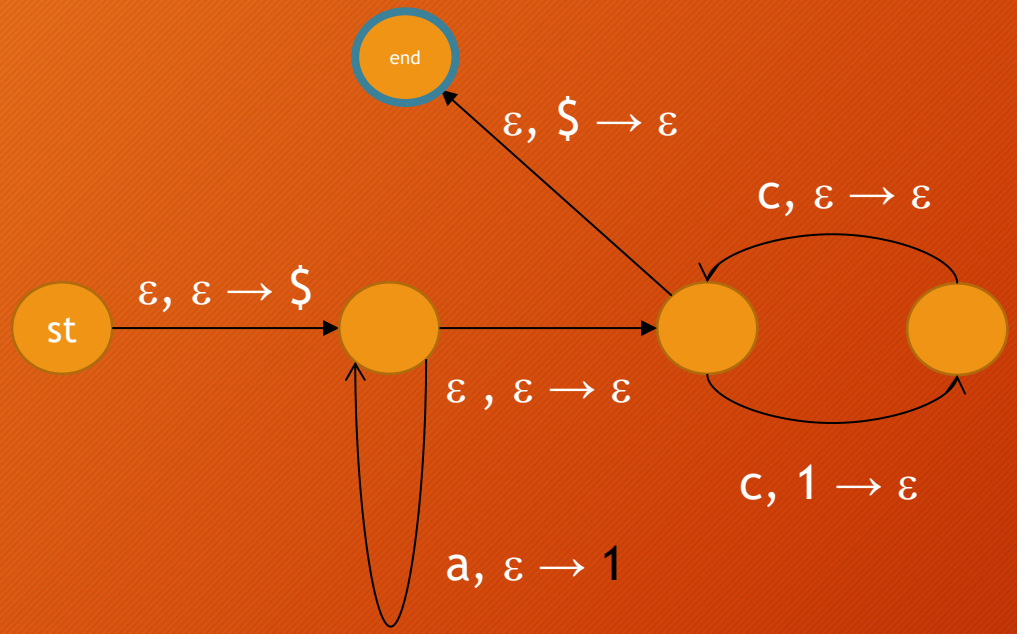
$ - specjalny symbol, znacznik końca

1010 0001

~~0~~
1
0
1

Automat ze stosem – dalszy przykład

- Automat ze stosem rozpoznający język $\{a^n c^{2n}\}$



Maszyna Turinga

Schemat działania:

1. Input = zapis w aktualnej pozycji głowicy
2. Wykonaj funkcję przejść δ, tj. ustal czy i co zapisać na taśmę, przesuń głowicę w lewo lub w prawo

Jednostka sterująca – automat

głowica

0	1	0	1	0	1	1	0	1	1

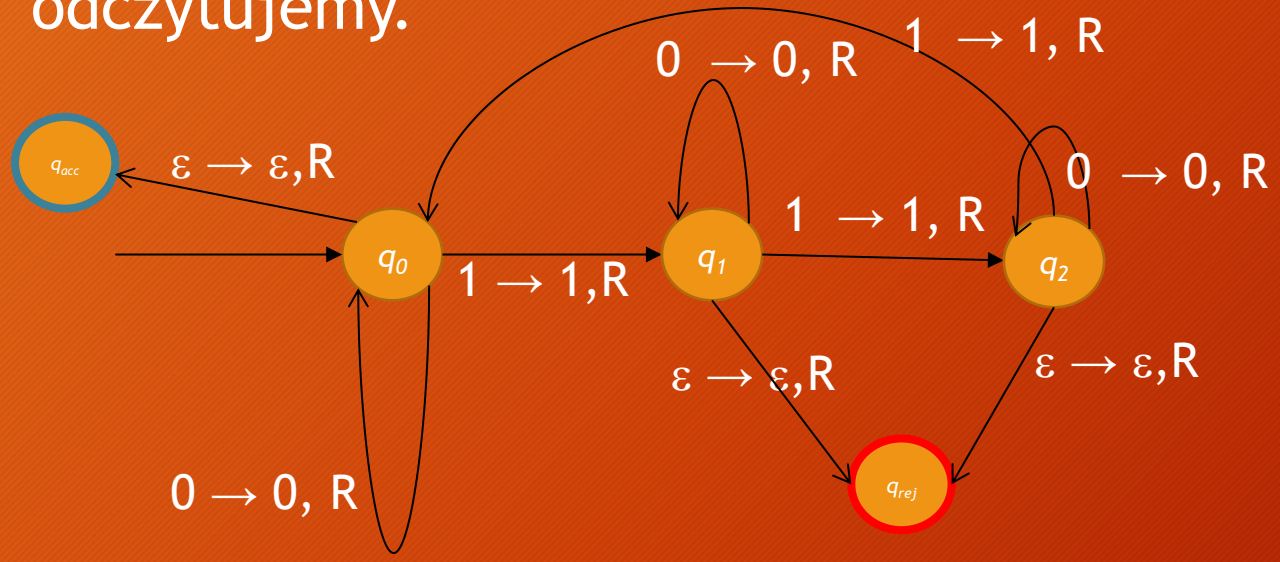
Taśma – pamięć, dane wejściowe i wyjściowe (po zakończeniu działania – rozwiązanie)

Maszyna Turinga - formalnie

- Q – zbiór stanów (automat sterujący)
- Σ - *alfabet wejściowy*
- Γ - *alfabet taśmowy* $\Sigma \subseteq \Gamma$ - ε
- δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ ***- logika (program) maszyny***
  - *Dla danego stanu maszyny i znaku odczytanego z taśmy, przejdź do pewnego stanu maszyny, zapisz określony wartością* δ *znak i przesuń głowicę w lewo lub w prawo*
  - *Np. (1, 0)* $\rightarrow$ *(2, 1, lewo) - oznacza, jeśli w stanie 1 odczytałeś z taśmy zero, przejdź do stanu 2, zapisz na taśmie 1 przejdź w lewo.*
- *Dodatkowo:*
  - ε *- znak/symbol pusty*
  - q_0 *stan początkowy,*
  - q_{acc} *- stan akceptujący*
  - q_{rej} *- stan odrzcający (różny od* q_{acc}*)*
- *{L, R}* – instrukcja dla głowicy (left/right)

Programujemy maszynę Turinga (1)

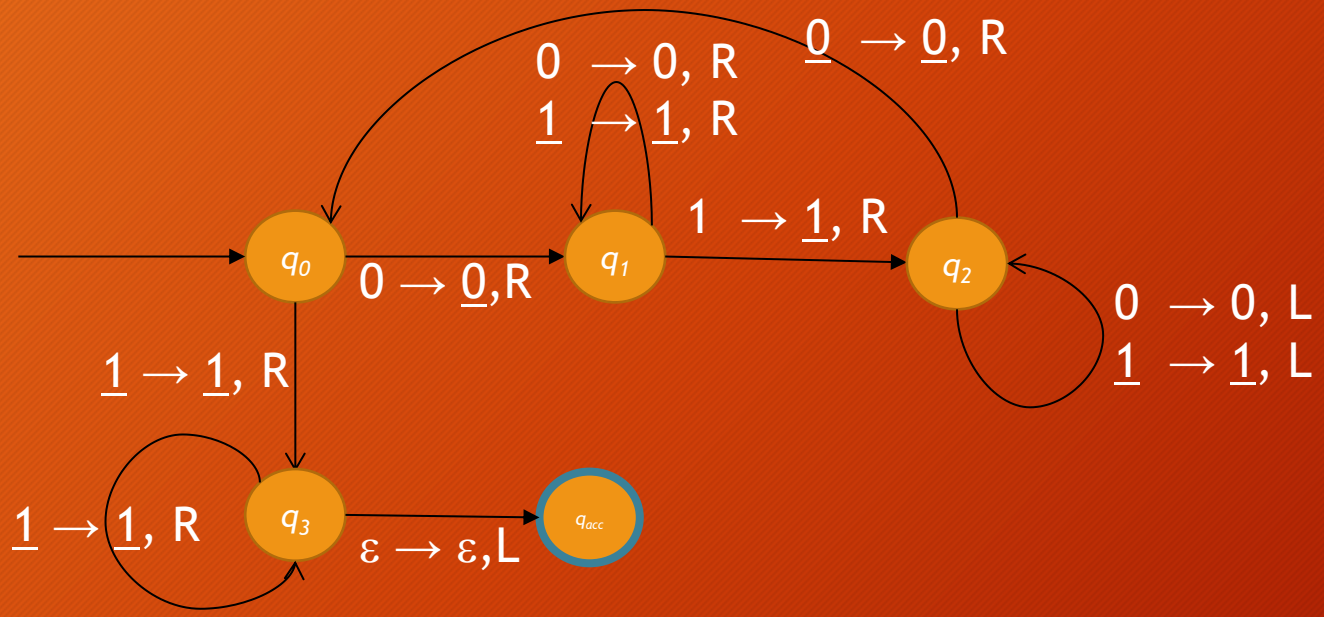
- Maszyna ma rozpoznawać język L={w $\in${0, 1}* : liczba 1-ek jest krotnością 3}
- Taśma zawiera słowo zapisane od lewej do prawej, które odczytujemy.



Taśma: 00000000011000000 1ε

Programujemy maszynę Turinga (2)

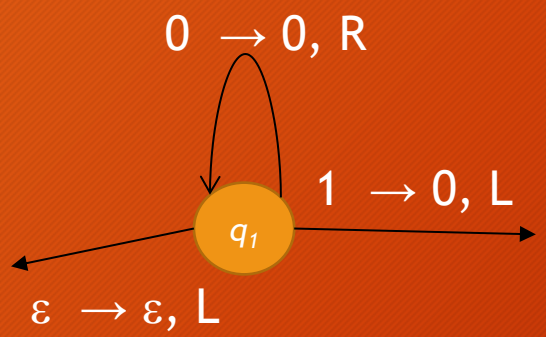
- Maszyna Turinga rozpoznająca język: L={0^n1^n: n>=0} Przykład „00001111”, „0000011111”



Wyjaśnienie – maszyna na taśmie Zaznacza w każdej iteracji kolejne 0 i kolejne 1, jeśli zaznaczy wszystkie Zera (i jedynki), to zmierza do stanu Akceptującego.

To zróbmy to nieco czytelniej…

- Przesuń w prawo „nad” zerami
- Jeśli przeczytasz „1”, nadpisz na niej „0” i przesuń w lewo
- Jeśli przeczytasz ε w lewo AKCEPT

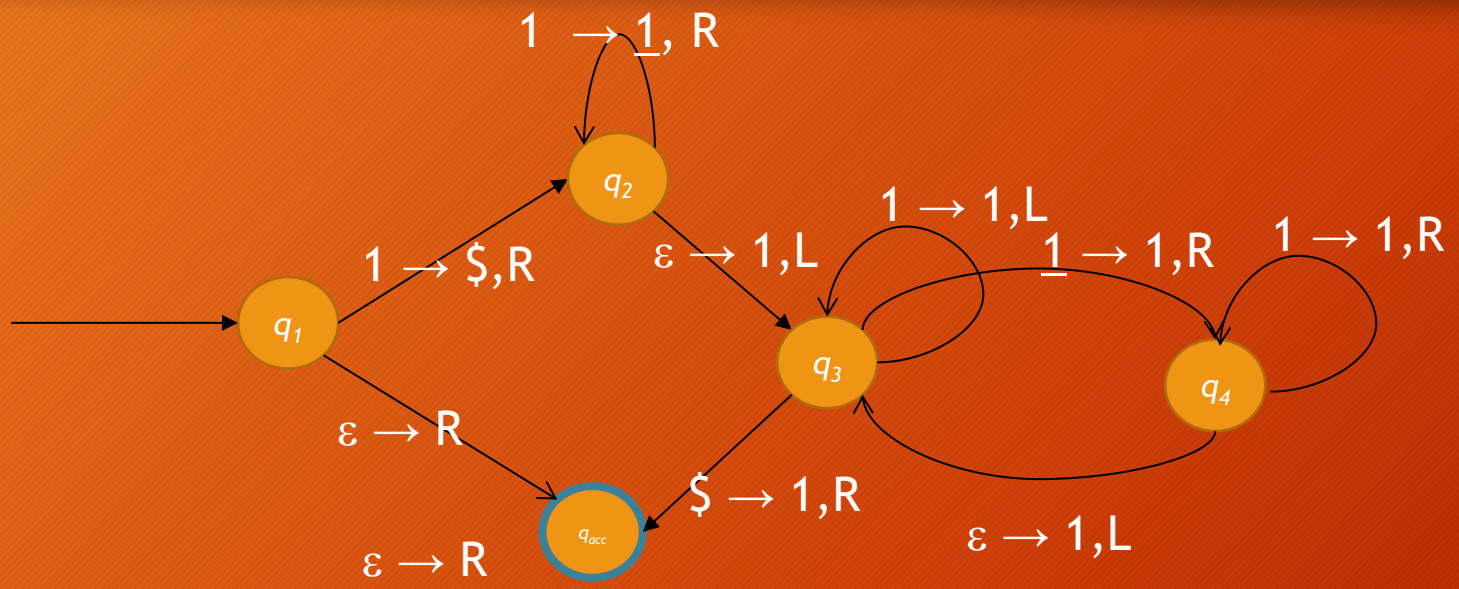


Programujemy dalej…

Dany jest ciąg wejściowy $\{1\}^*$, budujemy maszynę, która podwaja liczbę symboli w ciągu wejściowym

1. Jeśli pierwszy symbol to ε, to AKCEPT. Jeśli to 1, zamień na $ i przesuń w prawo.
2. Zastąp znak (1) znakiem (1) aż do napotkania ε (R), który nadpisz 1 (L)
3. Przesuń się w lewo nad 1 aż spotkasz pierwszą jedynkę oznaczoną (1). Znieś zaznaczenie - zapis (1'). Jeśli spotkasz $, zamień na 1, AKCEPT.
4. W prawo nad 1-kami aż do ε. Zapisz 1. Idź do 3.

Programowanie TM cd.



Wyjaśnienie:

- Na każdą jedynkę zaznaczoną dodaje jedną niezaznaczoną
- Jak zabraknie zaznaczonych, to koniec.

Jeszcze jeden przykład

Zbudować TM rozpoznającą język $L=\{a^n b^{n*n}: n>=1\}$. Rozwiązanie.

1. Przeleć z lewej do prawej. Jeśli nie ma a lub b lub są w niewłaściwej kolejności REJECT. //zapewnia, że na taśmie jest ciąg aaaa…bbbbbbb…
2. Wróć na początek. Szukaj „a”.
   jeśli znalezione -> zaznacz (a → a)
   przejedź głowicą na koniec. Napisz na taśmie „c”. Idź do 2.
   //Napisz na taśmie tyle razy c ile razy występuje a. ____aaaabbbbbbbbb**cc
3. Przesuń głowicę na początek. Szukaj „c”.
   Jeśli znalazłeś usuń, idź do 4.
   w przeciwnym razie
   Przesuń głowicę na początek
   Szukaj „b”.
   jeśli znaleziono REJECT, w przeciwnym razie ACCEPT.
4. Odznacz wszystkie a
5. Przesuń głowicę na początek. Szukaj „a”.
   Jeśli znalezione
   zaznacz, szukaj „b”, jeśli znalezione, zaznacz idź do 5 (z powrotem) w przeciwnym razie REJECT.
   w przeciwnym razie idź do 3.

Konkluzje

1. Maszyna Turinga rozpoznaje języki bardziej skomplikowane niż bezkontekstowe – ma większą moc wyrazu niż automat ze stosem (faktycznie rozpoznaje również języki kontekstowe i te jeszcze bardziej skomplikowane)
2. Stara obserwacja: im prostszy język zapisu algorytmu tym trudniej zrozumieć, jak on działa

Warianty maszyny Turinga

- Z dwoma taśmami
- Z wieloma taśmami
- Dwuwymiarowa taśma
- Niedeterministyczna maszyna Turinga
- Uniwersalna maszyna Turinga

Języki formalne. Hierarchia Chomsky’ego

Język formalny

- $\sum$ - alfabet
- Język L – zbiór słów zbudowanych ze znaków alfabetu $\sum$, tj. $L \subseteq \sum^*$
- $\sum^*$ - zbiór wszystkich możliwych ciągów liter (kolekcji) złożonych ze znaków alfabetu $\sum$
  - $A^* = \{w_1 w_2 \ldots w_k, k \geq 0$ i każde z $w_i \in A\}$ //tzw. gwiazdka Klenee'go
- Słowa = zdania

Gramatyka

Gramatyka - skończony zestaw reguł, zgodnie z którymi są generowane słowa (zdania)

Formalne G=(T, N, S, P)

- T - alfabet symboli terminalnych [faktycznych symboli/słów]
- N - alfabet symboli nieterminalnych, $S \cap T = \varnothing$ [jednostek składniowych]
- $S \in N$ - symbol startowy
- P - zbiór tzw. produkcji (reguł tworzenia słów/zdań)

  $P \subseteq \{a \rightarrow b$: a, b $\in (N \cup T)^*$ i a $\notin T^*\}$
- Produkcja - relacja (podstawienie) między ciągiem symboli nieterminalnych i terminalnych a ciągiem symboli terminalnych i/lub nieterminalnych
- Warunek: a $\notin T^*$ oznacza, że nie możemy dokonywać podstawień w miejsce samych symboli terminalnych (bo nie byłyby one terminalne)
- Produkcje - generator słów/ zdań

Gramatyka – interpretacja intuicyjna

- Od gramatyki zależy złożoność języka
- W istocie zależy ona od dopuszczalnych w danej klasie języków podstawień czyli produkcji
- Klasy języków formalnych wyróżnione są gramatyką tych języków

Przykład gramatyki

- N – symbole nieterminalne

{Instrukcja, instrukcja-if, instrukcja-while, instrukcja-złożona, warunek}

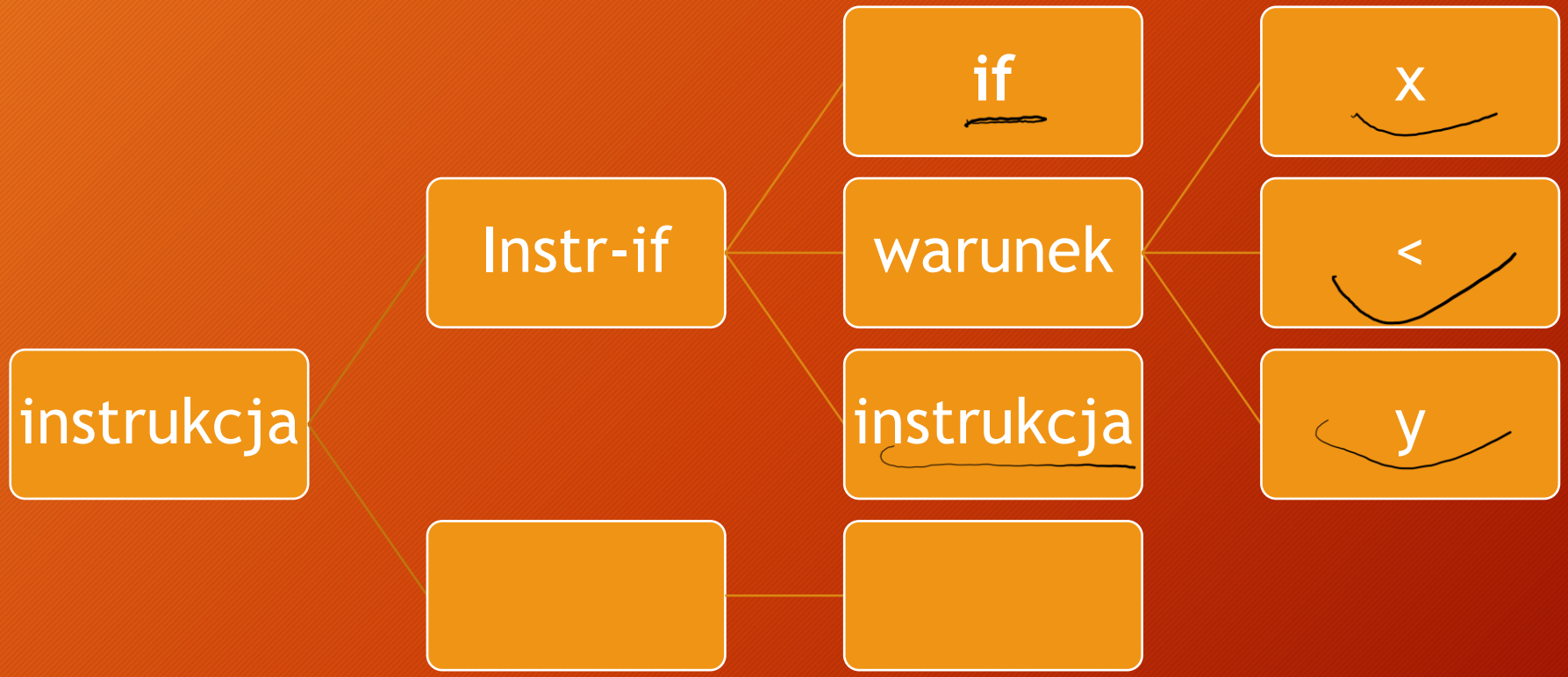
- T – symbole terminalne

{if, while, nazwa-zmiennej, `<`, `{`, `}` , liczba-całkowita }

Produkcje

- Instrukcja → instrukcja-if
- Instrukcja → instrukcja-while
- Instrukcja → instrukcja-złożona
- Instrukcja-złożona → { instrukcja }
- Instrukcja-if → if warunek instrukcja
- Warunek → nazwa-zmiennej < nazwa-zmiennej
- Warunek → nazwa-zmiennej < liczba-całkowita

Drzewo podstawień (przykład), generowanie języka



Język generowany przez gramatykę

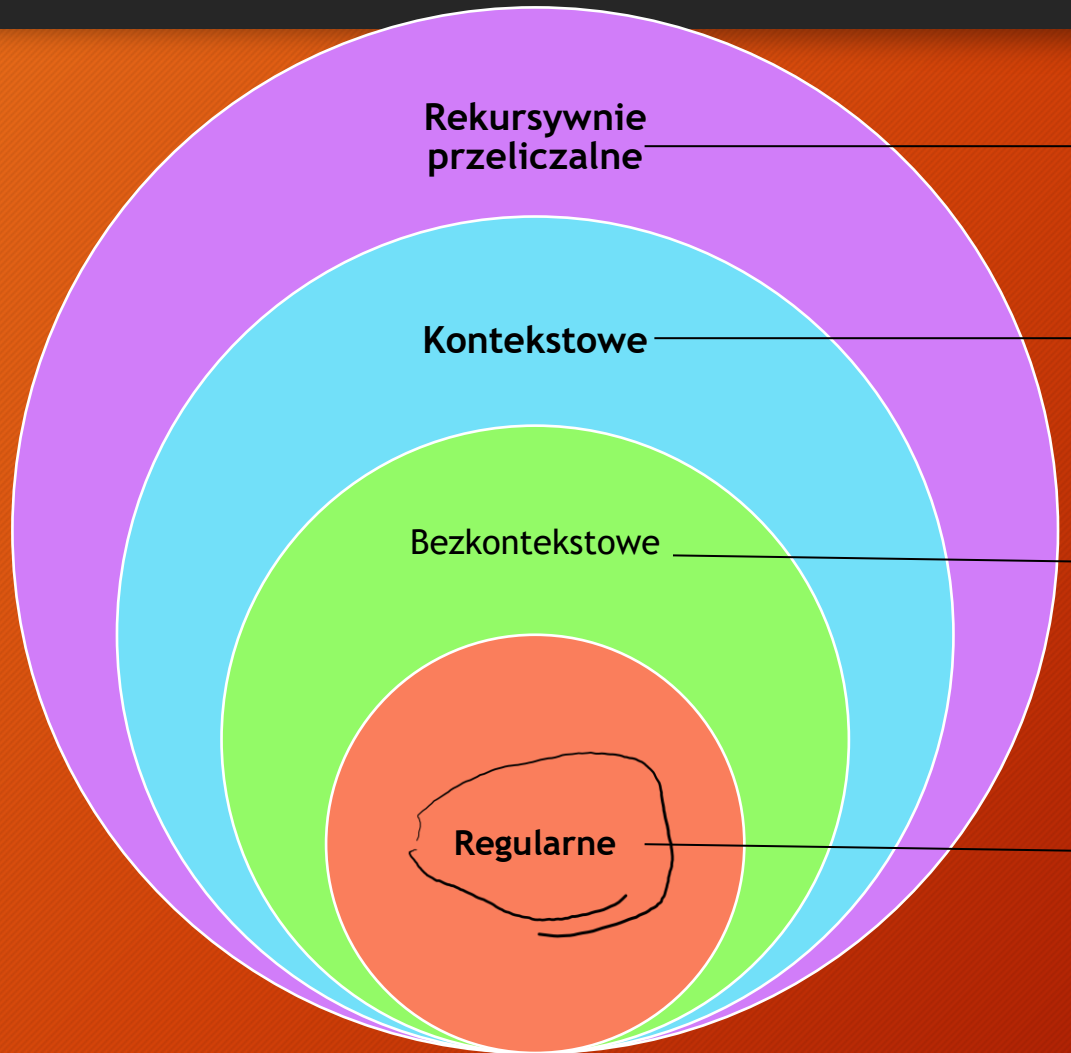
- Dana jest gramatyka G=(T, N, S, P)
- Zbiór wszystkich słów $w \in \Sigma^*$, które powstają przez zastosowanie produkcji z gramatyki zaczynając od symbolu startowego
- Oznaczenie: L(G)

Operacje na językach

- A i B - języki
- Suma mnogościowa słów: $A \cup B = \{w: w \in A \text{ or } w \in B\}$
- Konkatenacja (słów): $A \circ B = \{ vw: v \in A \text{ and } w \in B\}$ /zwykle pomijamy $\circ$/.
- Tzw. gwiazdka Kleen'ego: $A^* = \{w_1 w_2 \ldots w_k, k \geq 0$ i każde z $w_i \in A\}$.
  - Przykład
    - $B = \{ab\} \Rightarrow B^* = \{\varepsilon, ab, abab, ababab,...\}$
    - $C = \{a, bb\} \Rightarrow C^* = \{\varepsilon, a, bb, aa, abb, bba, bbbb, aaa,...\}$
    - $A = \{a,b\} \Rightarrow A^* = \{\varepsilon, a, ab, aa, bb, abb, aabb, \ldots\}$
  - Wszystkie możliwe łańcuchy złożone ze znaków / ciągów znaków należących do A

Klasy języków wg Chomsky’ego

Hierarchia Chomsky’ego



Maszyna Turinga

Liniowo ograniczony automat

Niedeterministyczny automat ze stosem (ang. pushdown automaton)

Automat skończony

- $(AB)^n$
- A^NB^N

Języki regularne

- Języki regularne – języki generowane przez gramatyki, z produkcjami postaci:
  - $A \rightarrow b\ B$ lub $A \rightarrow b$ gdzie $A, B \in N$, a $b \in T^*$

  lub
  - $A \rightarrow B\ b$ or $A \rightarrow ß$ gdzie $A, B \in N$ and $b \in T^*$

- {a, b}
- $S \rightarrow a\ B$
- $B \rightarrow b\ A \mid b$
- $A \rightarrow a\ B \mid a$

$a\ B \rightarrow a\ b$

$a\ \underline{B} \rightarrow a\ b\ \underline{A} \rightarrow a\ b\ a\ \underline{B} \rightarrow a\ b\ a\ b$

$\rightarrow a\ b\ a\ b\ A \rightarrow a\ b\ a\ b\ a\ b$

Języki regularne i automaty

- Tw. Kleeny’ego. Każdy język akceptowany przez automat skończony jest językiem regularnym, dla każdego języka regularnego istnieje akceptujący go automat skończony.

Lemat o pompowaniu (ang. pumping lemma)

Dla **każdego języka regularnego** L, **istnieje (!!!)** taka liczba naturalna *k*, że jeśli $s \in L$ i $|s| > k$, to s można podzielić na trzy części:

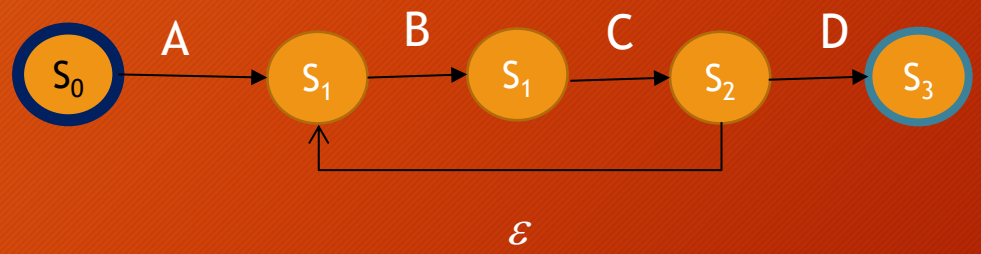
s=xyz (konkatenacja łańcuchów), że

- $|xy| \leq k$
- $|y| > 1$
- dla każdego $m \geq 0$ $xy^m z \in L$

- $x\ y^1\ z$ …. $x\ y^2\ z$

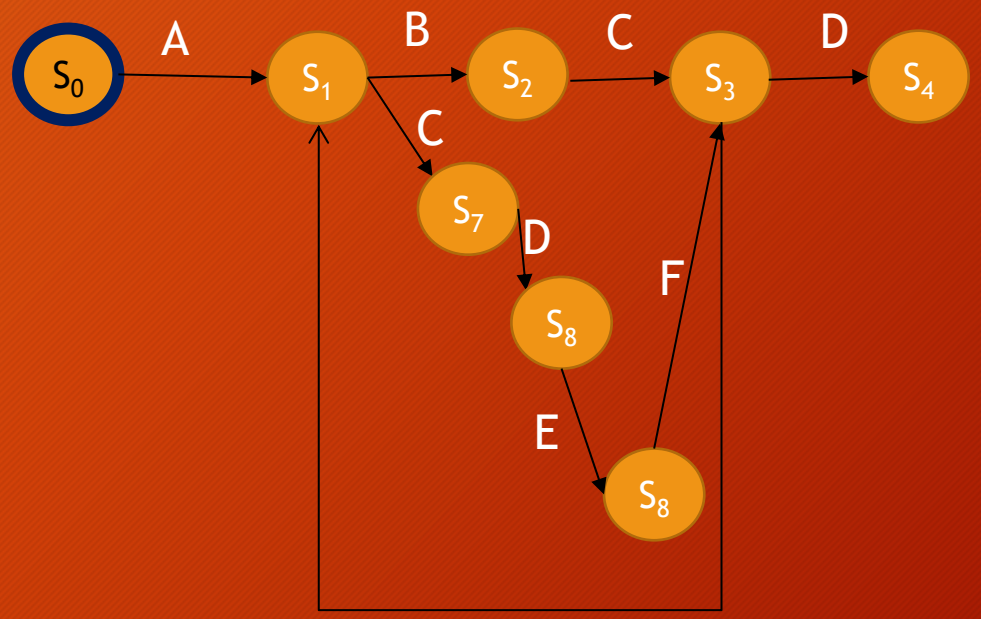
Ilustracja (1)

- A $(BC)^i$ D
- A BC D → A $(BC)^1$ D
- A BC BC D → A $(BC)^2$ D
-
- A BC BC BC D
- k=4
- X=A
- Y=BC
- Z=D



Ilustracja (2)

- A BC CDEF D
- x y z
- A $(BCCDEF)^k$ D



ε

Wyrażenia regularne

- R jest wyrażeniem regularnym, jeśli R jest
- 1. $\emptyset$ - wyrażeniem pustym.
- 2. ε.
- 3. $a \in \Sigma$.
- 4. $X \cup Y$, gdzie X i Y są wyrażeniami regularnymi
- 5. X Y, (konkatenacja), gdzie X i Y są wyrażeniami regularnymi
- 6. X^*, gdzie X jest wyrażeniem regularnym

Przykłady wyrażeń regularnych

- A={a b}
- R=$(ab)^*b$ → L(R)={abb, ababb, …}
- R=$(a^*) \cup (b^*)$ → L(R)={a, aa, aaa, …, b, bb, bbb}

Zamiana wyrażenia regularnego w niedeterm. automat skończony

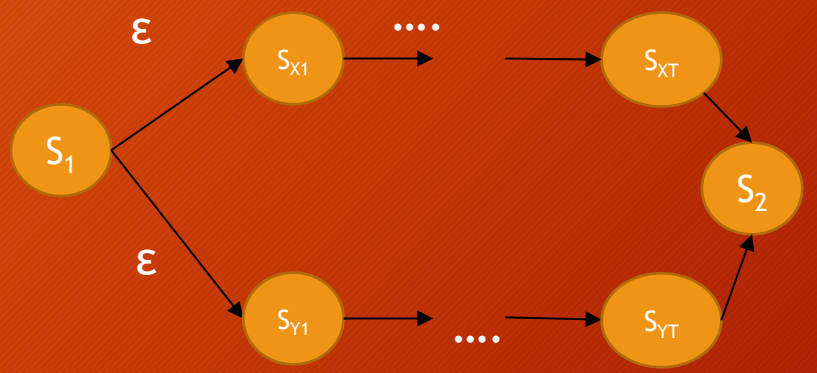
Niech dane będą automaty niedet. odpowiadające wyrażeniom X i Y

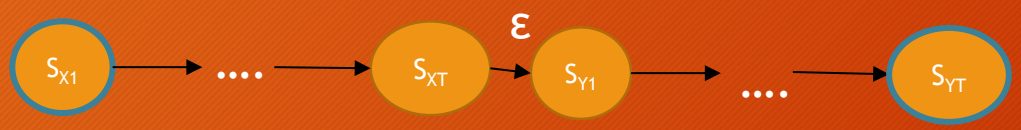
…

- 4. X ∪ Y, gdzie X i Y są wyrażeniami regularnymi
- 5. X Y, (konkatenacja), gdzie X i Y są wyrażeniami regularnymi
- 6. X^*, gdzie X jest wyrażeniem regularnym

Automaty rozpoznające wyrażenia regularne

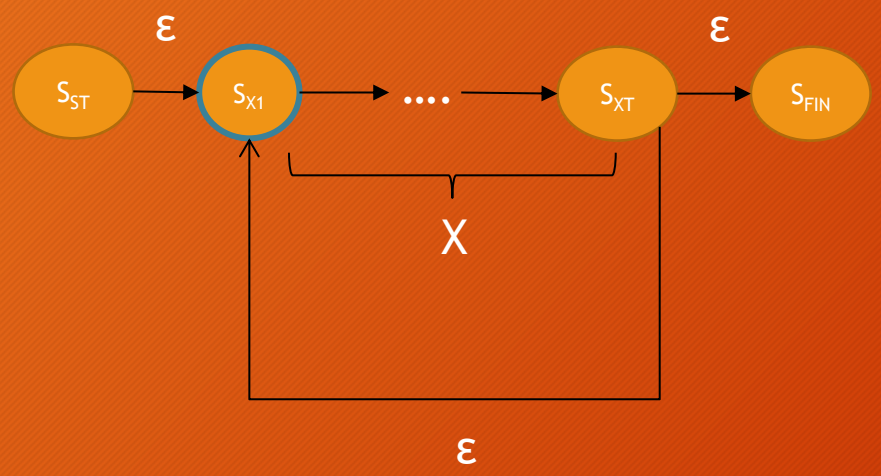
- X ∪ Y
- X Y



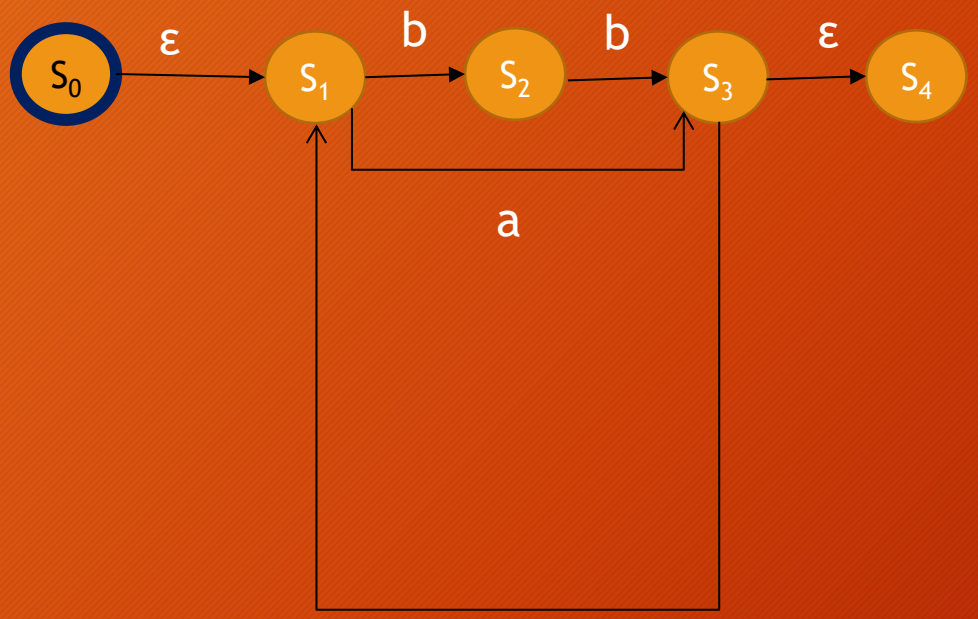


Automaty rozpoznające wyrażenia regularne

- X^*



Przykład – zbudować automat rozpoznający (a ∪ bb)*



Języki i wyrażenia regularne

- Język *L* jest regularny, jeśli opisuje go jakieś wyrażenie regularne.

Konkluzja finalna

- Automat skończony
- Wyrażenie regularne
- Gramatyka regularna

Mogą być alternatywnymi sposobami reprezentacji tego samego języka regularnego

Języki bezkontekstowe

Gramatyka bezkontekstowa

Gramatyka bezkontekstowa (ang. Context-Free Grammar, CFG) $G=(V, \Sigma, R, S)$, gdzie

- V (pierwotne: N) – zbiór symboli nieterminalnych (zwanych niekiedy zmiennymi)
- Σ – zbiór symboli terminalnych (rozłączny z V),
- R – zbiór produkcji postaci $V \rightarrow (V \cup \Sigma)^*$
- S – zmienna (symbol) początkowy $S \in V$.

Przykłady

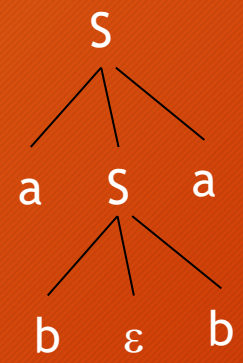
- Gramatyka G, która generuje palindromy o parzystej długości złożone z liter {a, b}
- $S \rightarrow aSa \mid bSb \mid \varepsilon$
- Gramatyka G, która generuje jw. Ale o nieparzystej długości
- $S \rightarrow aSa \mid bSb \mid a \mid b$

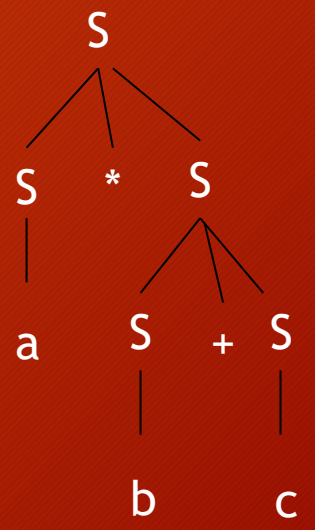
Drzewo parsowania (parse tree) [wyprowadzania]

- Dla danej gramatyki CFG G=(V, Σ, R, S), drzewem parsowania nazywamy takie drzewo, dla którego
  - Korzeniem drzewa jest S – symbol startowy
  - Wewnętrzne wierzchołki są oznaczone symbolami nieterminalnymi (zmiennymi)
  - Liście są oznaczone symbolami terminalnymi lub ε
  - Liście oznaczone ε muszą być jedynymi dziećmi swojego rodzica
  - Jeśli wierzchołek jest oznaczony jako A, a jego dzieci symbolami B_1, B_2 , …, B_n, to produkcja $A \rightarrow B_1, B_2 , \ldots, B_n$ należy do R (zbioru produkcji gramatyki G)

Przykład

- $S \rightarrow aSa \mid bSb \mid \varepsilon$





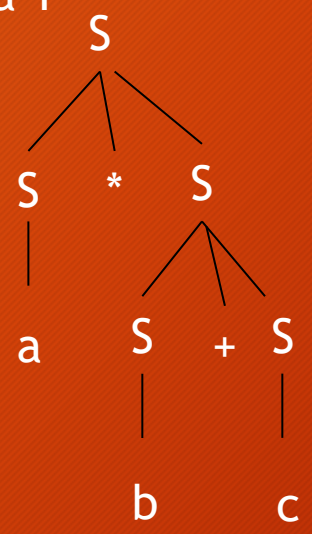
Przeszukać drzewo in-order by odczytać słowo / zdanie

Niejednoznaczność składniowa

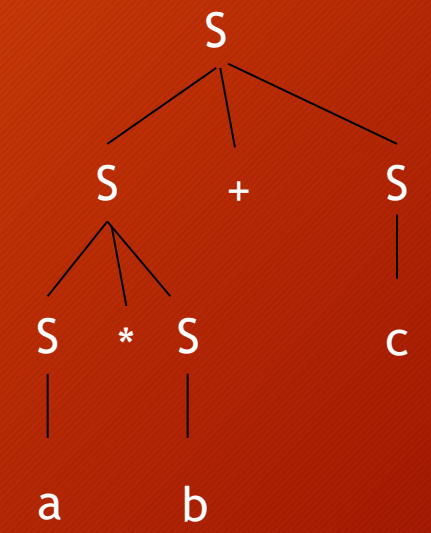
Przykład – produkcje pewnej gramatyki

- $S \rightarrow S * S$
- $S \rightarrow S + S$
- $S \rightarrow (S) \mid a \mid b \mid c$

Wersja 1



Wersja 2



Czyli ten sam łańcuch symboli można wygenerować stosując różne ciągi produkcji

a * b + c

Niejednoznaczność c.d.

Dane są produkcje

- $S \rightarrow AB$
- $A \rightarrow 00A \mid \varepsilon$
- $B \rightarrow 1B \mid \varepsilon$

Zauważmy, że łańcuch 0011 można wygenerować na dwa sposoby:

1) $S \rightarrow \underline{A}B \rightarrow \underline{00A}B \rightarrow 00B \rightarrow 001B \rightarrow 0011B \rightarrow 0011$ (skrajnie lewe podstawienia)
2) $S \rightarrow A\underline{B} \rightarrow A1\underline{B} \rightarrow A11\underline{B} \rightarrow \underline{A}11 \rightarrow 00\underline{A}11 \rightarrow 0011$ (skrajnie prawe podstawienia)

Niejednoznaczność CGF

CFG jest niejednoznaczna, jeśli istnieje

1) łańcuch, który ma dwie „lewo-skrajne” (ang. leftmost) ciągi wyprowadzeń
2) a w efekcie dwa różne drzewa podstawień (ang. parse tree)

Uproszczenia

Gramatyka nr 1

- $S \rightarrow 00S0 \mid D \mid B$
- $A \rightarrow 1A \mid 1$
- $B \rightarrow 0B \mid BB$
- $C \rightarrow 0D \mid B1$
- $D \rightarrow A$

Gramatyka nr 2

- $S \rightarrow 00S0 \mid A$
- $A \rightarrow 1A \mid 1$

Obie generują ten sam język
Język: $\{0^{2n}\ 1^{m}\ 0^{n}: n > 0, m > 1\}$

Symbole bezużyteczne:
1) Nieosiągalne ze startowego
2) Niegenerujące – takie, których podstawianie nie skończy się symbolami terminalnym

Eliminacja produkcji jednostkowych

- Ang. Unit productions
- $A \rightarrow B$, gdzie A i B to zmienne
- Jeśli mamy w gramatyce produkcję $A \rightarrow B$ oraz $B \rightarrow X$, to możemy zamiast nich wprowadzić produkcje $A \rightarrow X$

Eliminacja produkcji „zerowalnych” (nullable)

Przykład 1.

- S → 0S0 | 1A1
- A → 000 | ε

Po wyeliminowaniu: A → ε przez podstawienie

- S → 0S0 | 1A1 |11
- A → 000

Przykład 2.

S → 0S0 | 1A0A1

A → 000 | ε

Po wyeliminowaniu

- S → 0S0 | 1A0A1 | 10A1 |1A01 | 101
- A → 000

Postać normalna Chomsky’ego gramatyki bezkontekstowej

Gramatyka bezkontektowa jest w normalnej postaci Chomsky’ego (ang. skr. CNF), jeśli produkcje mają postać:

- $A \rightarrow BC$,
- $A \rightarrow a$, gdzie
- A, B, C – zmienne (symbole nieterminalne), B i C nie są symbolami startowymi
- a – symbol(e) terminalne

Dodatkowo jest zawsze określona produkcja $S \rightarrow \varepsilon$ (pusty symbol).

Czyli w uproszczeniu

$A \rightarrow BC \mid a \mid \varepsilon$

CNF - przykłady

Przykład 1

- $S \rightarrow AB \mid AA$
- $A \rightarrow a$
- $B \rightarrow BC \mid CC \mid b$
- $C \rightarrow a \mid c$

Przykład 2

- $S \rightarrow TU \mid BT \mid \varepsilon$
- $T \rightarrow AB \mid AT \mid c$
- $U \rightarrow BB \mid BU \mid c$
- $A \rightarrow a$
- $B \rightarrow b$

Drzewo podstawień dla CFG w CNF jest binarne

Zaleta CNF

- Wyprowadzenia każdego słowa o długości *n* symboli wymaga 2n – 1 podstawień

CNF a języki bezkontekstowe

Twierdzenie. Dowolny język bezkontekstowy może zostać wygenerowany przez gramatyka bezkontekstowa w postaci CNF (każdy język bezkontekstowy ma swoją gramatykę w CNF).

$S \rightarrow a\ B\ c\ A$
$\Rightarrow S \rightarrow D\ B\ E\ A \Rightarrow X\ Y$

Dowód: polega na pokazaniu, że każdą gramatykę ze zbiorem produkcji postaci bezkontekstowej można sprowadzić do CNF przez:

- Eliminację produkcji jednostkowych,
- Eliminację produkcji epsilonowych (puste podstawienia),
- Eliminując symbole terminalnych z podstawień, tam gdzie są podstawiane łącznie ze zmiennymi
- Eliminując nadmiar symboli przypisywanych (prawa strona) wprowadzając w razie dodatkowe zmienne

$S \rightarrow XY$
$X \rightarrow D\ B$
$Y \rightarrow E\ A$
$D \rightarrow a$
$E \rightarrow c$

Przykład przekształcenia CFG w CNF

Dana jest gramatyka bezkontekstowa ze zbiorem produkcji:

- S → 0A0 | 1B
- A → 1 | BB
- B → 111 | A0

1) Eliminujemy znaki terminalne z produkcji zawierających w przypisaniu zmienne

S → ZAZ | WB
A → 1 | BB
B → WWW | AZ
W → 1
Z → 0

2) Skracamy podstawienia 3 symboli wprowadzając nowe zmienne:

S → ZC | WB

A → 1 | BB

B → WD AZ

C → AZ

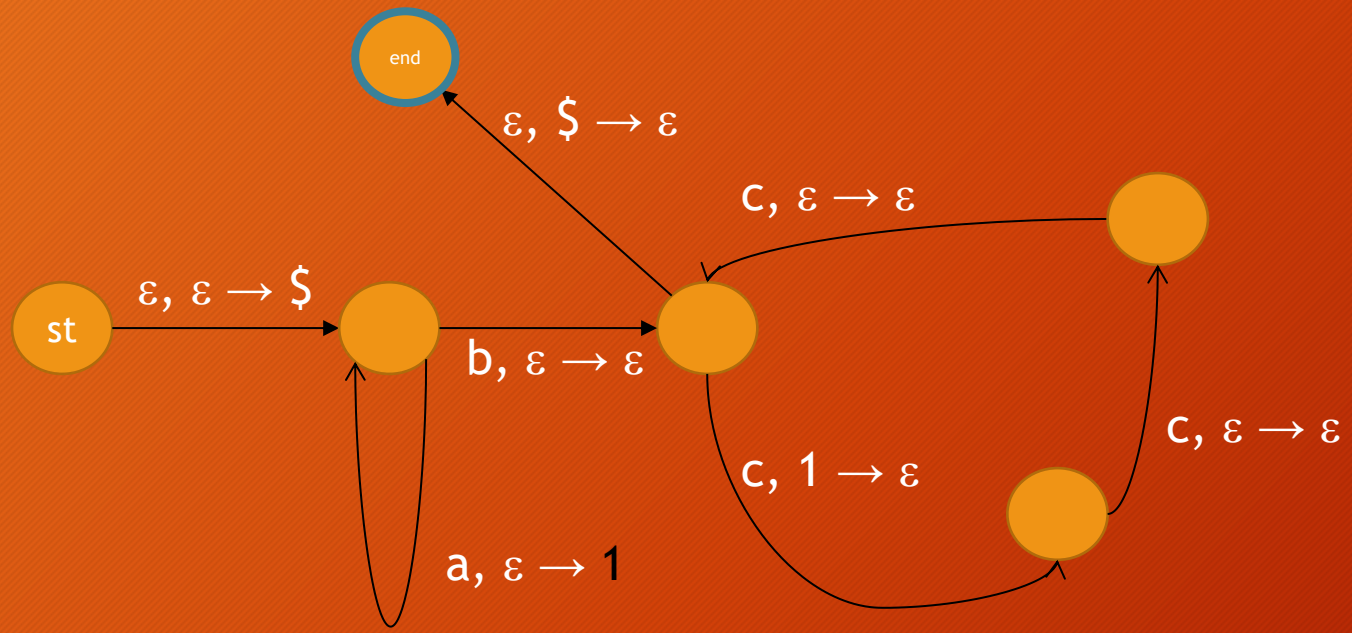
D → WW

W → 1

Z → 0

Jaki język rozpoznaje automat ze stosem

- $a^n b\ c^{3n}$



Gramatyki bezkontekstowe a automaty ze stosem

Twierdzenie. Język L jest bezkontekstowy wtedy i tylko wtedy, gdy istnieje automat ze stosem rozpoznający ten język.

Oznacza to, że:

- Dla danej gramatyki bezkontekstowej G możemy znaleźć automat ze stosem rozpoznającą język L(G)
- Dla danego automatu ze stosem rozpoznającego język L’ możemy znaleźć odpowiadającą mu gramatykę G, taką że L’=L(G)

Wniosek dotyczący języków regularnych

Twierdzenie. Każdy język regularny jest językiem bezkontekstowym.

Uzasadnienie: każdy automat skończony jest przypadkiem szczególnym automatu ze stosem nie korzystającym ze stosu, tj. o funkcjach przejścia o wartościach postaci $x, \varepsilon \rightarrow \varepsilon$, gdzie x jest symbolem z alfabetu wejściowego

Ciekawe pytanie

- Czy język $\{a^n b^n c^n : n>0\}$ jest bezkontekstowy?
- Nie, problem jednego stosu!

Lemat o pompowaniu dla języków kontekstowych

Jeśli *L* jest językiem bezkontektowym, to istnieje taka liczba p (długość pomowania – ang. pumping length), że jeżeli $s \in L$ i $|s| > p$,

to można podzielić s na 5 części s = uvxyz spełniających warunki:

1. $uv^kxy^k z \in L$ dla każdego $k \geq 0$
2. $|vy| > 0$ */v i y nie mogą być puste* ε*/*
3. $|vxy| \leq p$

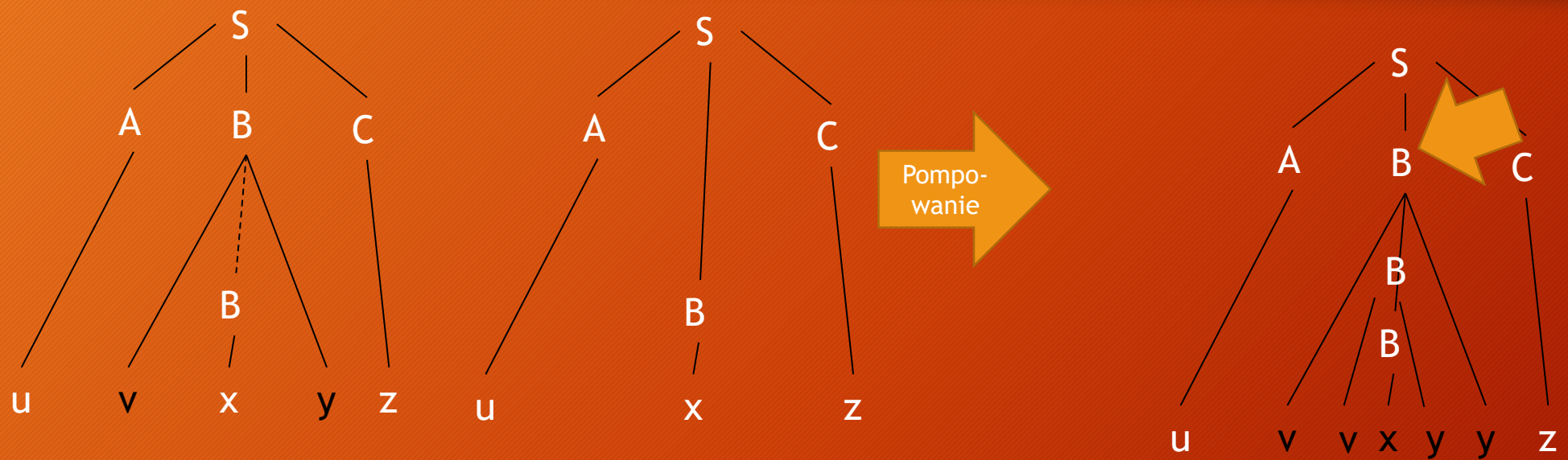
Pompowalność słów/wyrażeń jest warunkiem koniecznym, ale nie wystarczającym dla bycia językiem bezkontektowym

Przykład – ilustrujący lemat o pompowaniu dla CFL

- Język palindromów o nieparzystej długości

s	u	v	x	y	z
101	ε	1	0	1	ε
11011	1	1	0	1	1
011010110	011	0	1	0	110

Pompowanie CFL



Wykorzystanie lematu o pompowaniu dla CFL

- Dla języka $L=\{a^n b^n c^n : n \geq 0\}$ wykazać, że nie jest językiem bezkontekstowym
- Zadanie typu otwartego: załóżmy pewne p, jako długość pompowania, rozważmy łańcuch testowy $a^p b^p c^p$, które musimy podzielić na 5 części, w tym 2 „pompowalne”:
- Czyli $a^p b^p c^p = uvxyz$
  - Załóżmy, że v i y zawierają więcej niż jeden symbol, wtedy
    - $u\, v^2\, xy^2\, z \notin a^n b^n c^n$
  - Jeśli założymy, że *v*=a i *y*=b zawierają dokładnie jeden symbol
    - $u\, v^2\, xy^2\, z = a^{n+1} b^{n+1} c^n \notin a^n b^n c^n$

Uwaga!

- Ze spełnienia warunku pompowalności nie wynika, że język jest bezkontekstowy!
- Aby to sprawdzić trzeba, albo znaleźć gramatykę, albo automat ze stosem

Własność domkniętości języków bezkontekstowych

Analogicznie jak dla języków bezkontektowych

Jeśli A i B są językami regularnymi, to:

- $A \cup B$
- $A \circ B$ (konkatenacja)
- A^*

Też są językami kontekstowymi.

Gramatyki konktekstowe

(informacja)

Gramatyka kontekstowa

Gramatyka bezkontekstowa (ang. Context-Sensitive Grammar, CSG) G=(V, Σ, R, S), gdzie

- V (pierwotnie: N) – zbiór symboli nieterminalnych (zwanych niekiedy zmiennymi)
- Σ – zbiór symboli terminalnych (rozłączny z V),
- R – zbiór produkcji postaci $aUb \rightarrow a\ y\ b$, gdzie $x, y, z \in (V \cup \Sigma)^*$, z czego y musi być niepusty, a, b – ciągi symboli terminalnych lub nie terminalnych (w tym ε!), czyli aby dokonać podstawienia zmienna U musi być „w kontekście znaków *a* i *b*”
- S – zmienna (symbol) początkowy $S \in V$.

Przykład języka kontekstowego

$L=\{a^i b^i c^i : i>0\}$

1. S → aBC
2. S → aSBC
3. CB → CZ
4. CZ → WZ
5. WZ → WC
6. WC → BC
7. aB → ab
8. bB → bb
9. bC → bc
10. cC → cc

- Trudne przeparsowanie (nie każde jest udane, niektóre „zdychają”), generalnie działają
- S
- $\rightarrow_2$ **aSBC** $\rightarrow_2$ *aa**SBC**BC* $\rightarrow_1$ *aa**aBCBC**BC*
- $\rightarrow_3$ *aaaB**CZ**CBC* $\rightarrow_4$ *aaaB**WZ**CBC*
- $\rightarrow_5$ *aaaB**WC**CBC* $\rightarrow_6$ *aaaB**BC**CBC*
- $\rightarrow_3$ *aaaBBC**CZ**C*
- $\rightarrow_4$ *aaaBBC**WZ**C*
- $\rightarrow_5$ *aaaBBC**WC**C*
- $\rightarrow_6$ *aaaBBC**BC**C*
- $\rightarrow_3$ *aaaBB**CZ**CC*
- $\rightarrow_4$ *aaaBB**WZ**CC*
- $\rightarrow_5$ *aaaBB**WC**CC*
- $\rightarrow_6$ *aaaBB**BC**CC*
- $\rightarrow_7$ *aa**ab**BBCCC*
- $\rightarrow_8$ *aaa**bb**BCCC*
- $\rightarrow_8$ *aaab**bb**CCC*
- $\rightarrow_9$ *aaabb**bc**CC*
- $\rightarrow_{10}$ *aaabbb**cc**C*
- $\rightarrow_{10}$ *aaabbbc**cc***

Koniec

Algorytmy i struktury danych

Odc. 12. Maszyny Turinga. Nieobliczalność.

Maszyna Turinga

Schemat działania:

1. Input = zapis w aktualnej pozycji głowicy
2. Wykonaj funkcję przejść δ, tj. ustal czy i co zapisać na taśmę, przesuń głowicę w lewo lub w prawo

Jednostka sterująca – automat

głowica

0	1	0	1	0	1	1	0	1	1

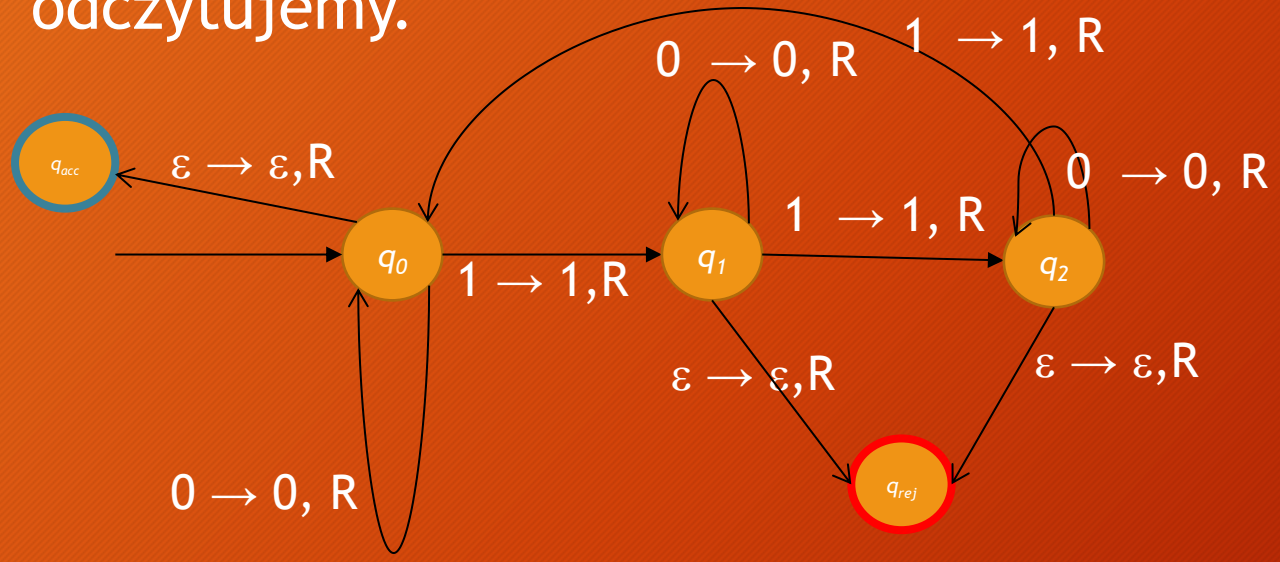
Taśma – pamięć, dane wejściowe i wyjściowe (po zakończeniu działania – rozwiązanie)

Maszyna Turinga - formalnie

- Q – zbiór stanów (automat sterujący)
- Σ - *alfabet wejściowy*
- Γ - *alfabet taśmowy* $\Sigma \subseteq \Gamma$ - ε
- δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ ***- logika (program) maszyny***
  - *Dla danego stanu maszyny i znaku odczytanego z taśmy, przejdź do pewnego stanu maszyny, zapisz określony wartością* δ *znak i przesuń głowicę w lewo lub w prawo*
  - *Np. (1, 0)* $\rightarrow$ *(2, 1, lewo) - oznacza, jeśli w stanie 1 odczytałeś z taśmy zero, przejdź do stanu 2, zapisz na taśmie 1 przejdź w lewo.*
- *Dodatkowo:*
  - ε *- znak/symbol pusty*
  - q_0 *stan początkowy,*
  - q_{acc} *- stan akceptujący*
  - q_{rej} *- stan odrzcający (różny od* q_{acc}*)*
- *{L, R}* – instrukcja dla głowicy (left/right)

Programujemy maszynę Turinga (1)

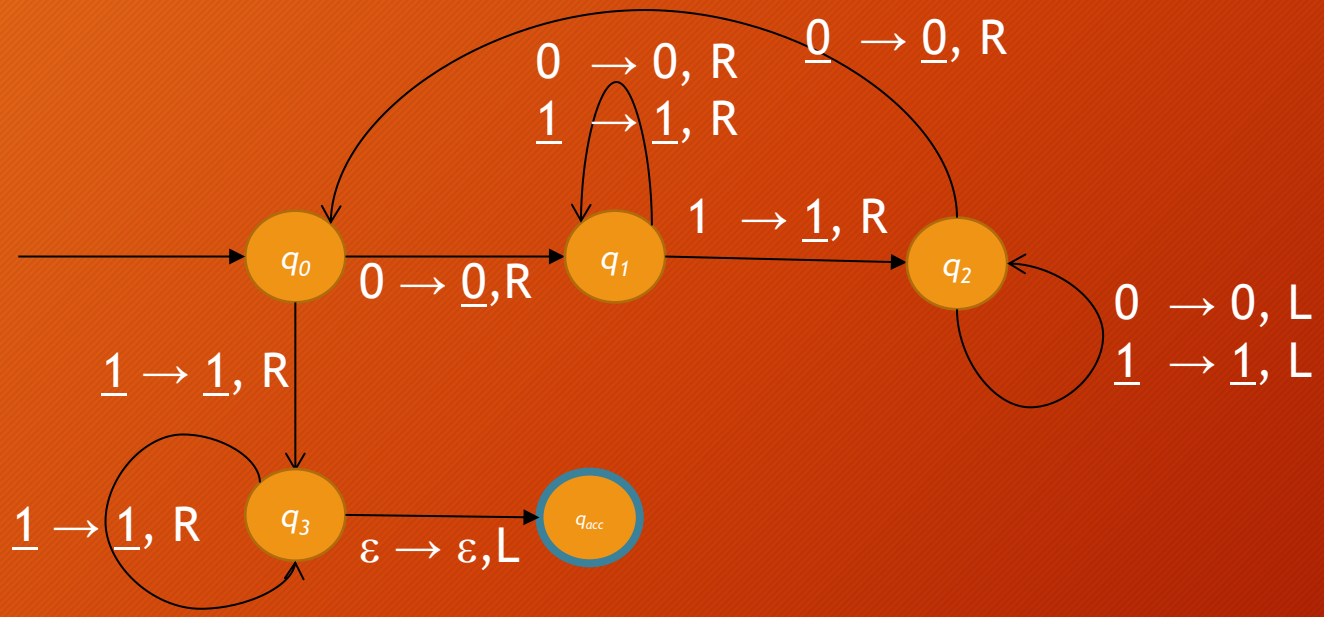
- Maszyna ma rozpoznawać język L={w $\in${0, 1}* : liczba 1-ek jest krotnością 3}
- Taśma zawiera słowo zapisane od lewej do prawej, które odczytujemy.



Taśma: 00000000011000000 1ε

Programujemy maszynę Turinga (2)

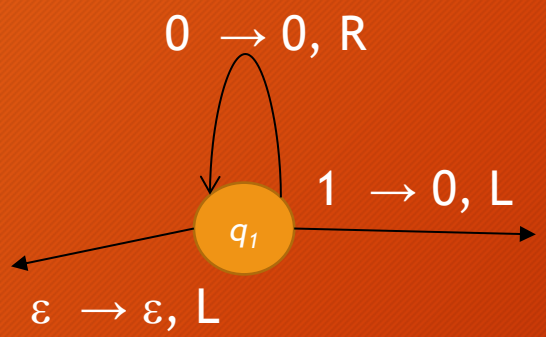
- Maszyna Turinga rozpoznająca język: L={0^n1^n: n>=0} Przykład „00001111”, „0000011111”



Wyjaśnienie – maszyna na taśmie Zaznacza w każdej iteracji kolejne 0 i kolejne 1, jeśli zaznaczy wszystkie Zera (i jedynki), to zmierza do stanu Akceptującego.

To zróbmy to nieco czytelniej…

- Przesuń w prawo „nad” zerami
- Jeśli przeczytasz „1”, nadpisz na niej „0” i przesuń w lewo
- Jeśli przeczytasz ε w lewo AKCEPT

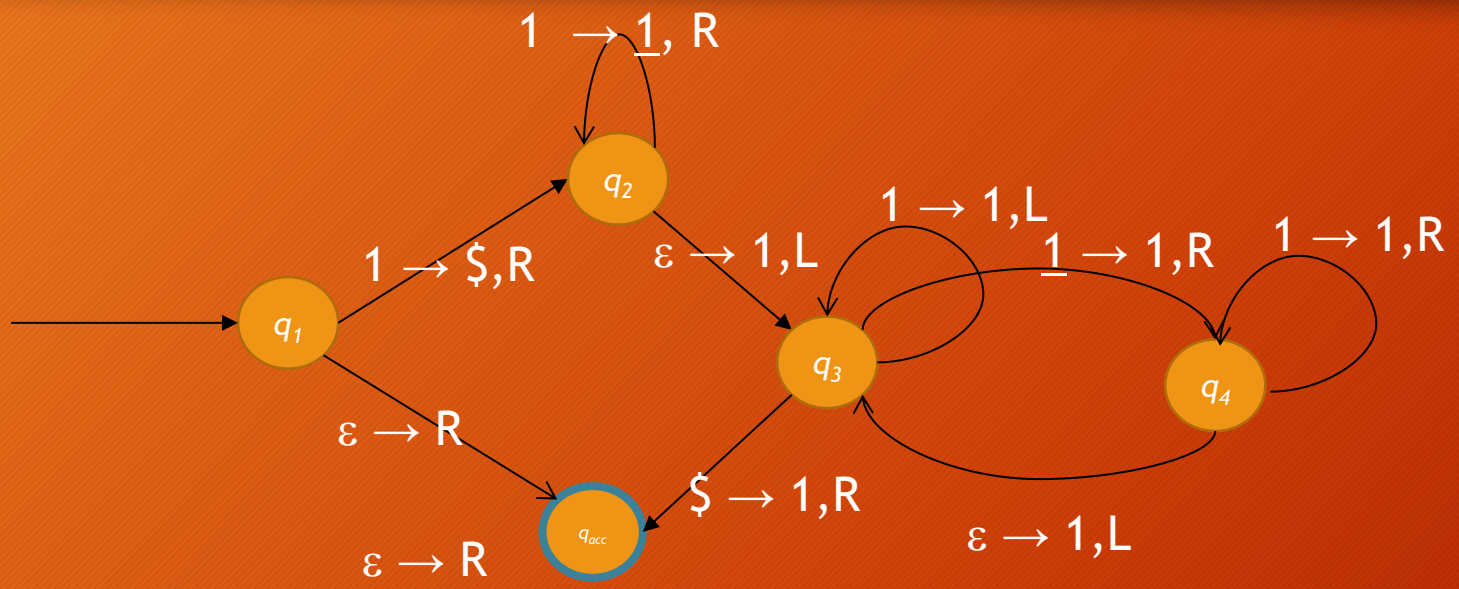


Programujemy dalej…

Dany jest ciąg wejściowy $\{1\}^*$, budujemy maszynę, która podwaja liczbę symboli w ciągu wejściowym

1. Jeśli pierwszy symbol to ε, to AKCEPT. Jeśli to 1, zamień na $ i przesuń w prawo.
2. Zastąp znak (1) znakiem (1) aż do napotkania ε (R), który nadpisz 1 (L)
3. Przesuń się w lewo nad 1 aż spotkasz pierwszą jedynkę oznaczoną (1). Znieś zaznaczenie - zapis (1'). Jeśli spotkasz $, zamień na 1, AKCEPT.
4. W prawo nad 1-kami aż do ε. Zapisz 1. Idź do 3.

Programowanie TM cd.



Wyjaśnienie:

- Na każdą jedynkę zaznaczoną dodaje jedną niezaznaczoną
- Jak zabraknie zaznaczonych, to koniec.

Jeszcze jeden przykład

Zbudować TM rozpoznającą język $L=\{a^n b^{n*n}: n>=1\}$. Rozwiązanie.

1. Przeleć z lewej do prawej. Jeśli nie ma a lub b lub są w niewłaściwej kolejności REJECT. //zapewnia, że na taśmie jest ciąg aaaa…bbbbbbb…
2. Wróć na początek. Szukaj „a”.
   jeśli znalezione -> zaznacz (a → a)
   przejedź głowicą na koniec. Napisz na taśmie „c”. Idź do 2.
   //Napisz na taśmie tyle razy c ile razy występuje a. ____aaaabbbbbbbbb**cc
3. Przesuń głowicę na początek. Szukaj „c”.
   Jeśli znalazłeś usuń, idź do 4.
   w przeciwnym razie
   Przesuń głowicę na początek
   Szukaj „b”.
   jeśli znaleziono REJECT, w przeciwnym razie ACCEPT.
4. Odznacz wszystkie a
5. Przesuń głowicę na początek. Szukaj „a”.
   Jeśli znalezione
   zaznacz, szukaj „b”, jeśli znalezione, zaznacz idź do 5 (z powrotem) w przeciwnym razie REJECT.
   w przeciwnym razie idź do 3.

Konkluzje

1. Maszyna Turinga rozpoznaje języki bardziej skomplikowane niż bezkontekstowe – ma większą moc wyrazu niż automat ze stosem (faktycznie rozpoznaje również języki kontekstowe i te jeszcze bardziej skomplikowane)
2. Stara obserwacja: im prostszy język zapisu algorytmu tym trudniej zrozumieć, jak on działa

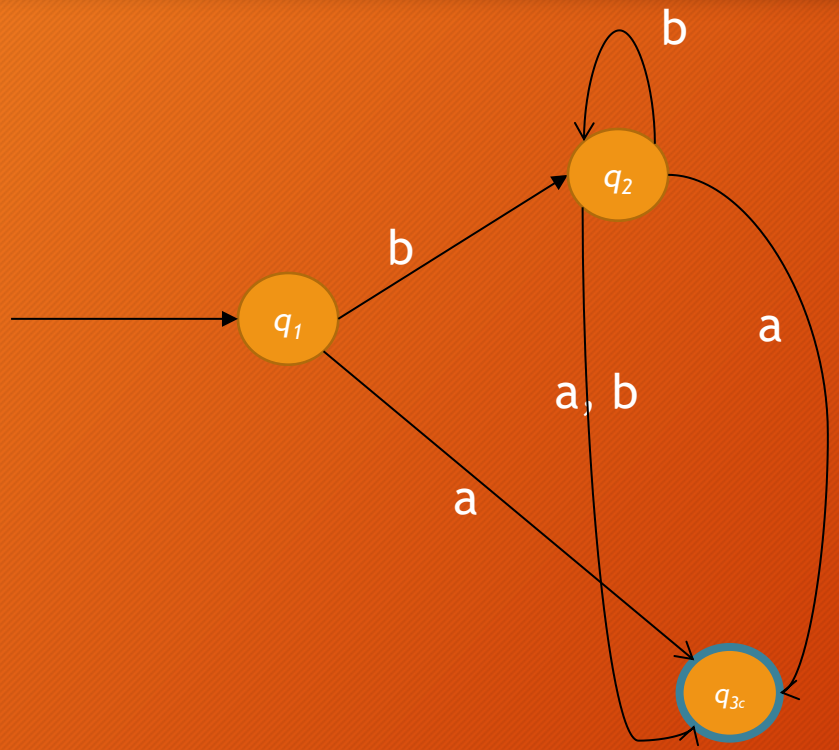
Warianty maszyny Turinga

- Z dwoma taśmami
- Z wieloma taśmami
- Dwuwymiarowa taśma
- Niedeterministyczna maszyna Turinga
- Uniwersalna maszyna Turinga

Kodowanie programu

- W komputerze używamy tylko 0 i 1. Można ponumerować stany jako 1, 11, 111, wykorzystując zera jako separatory komórek, dodać specjalne kody dla każdego z symboli A, R, a, b, oraz 00 jako separator #etc. Wtedy program maszyny można zapisać jako:
- #q1Raq3bq2 → 001010101110110 11
- #q3Aaq2bq2 → 0011101101011011011

Zakodowany automat determin



- #q1Raq3bq2#q2Rbq2aq3…
- A to możemy przekodować na zera i jedynki

- #q1Raq3bq2
- #=00 q1=1 R=01 a=10 b=11, 0 - sparator
- 00 1 0 01 0 10 0 111 0 10 0 11

Konkluzja (oczywista)

- Każdy z automatów, nawet logikę maszyny Turinga możemy zapisać w postaci ciągów zer i jedynek
- To jest dokładnie ta droga, która została pokonana w rozwoju komputerów – pierwszy i istotny problem – binarne kodowanie (zapis) danych
- Uwaga lingwistyczna
  - Kodowanie – nie mylić z szyfrowaniem, zakodowany to nie zaszyfrowany

Uniwersalna maszyna Turinga (UTM)

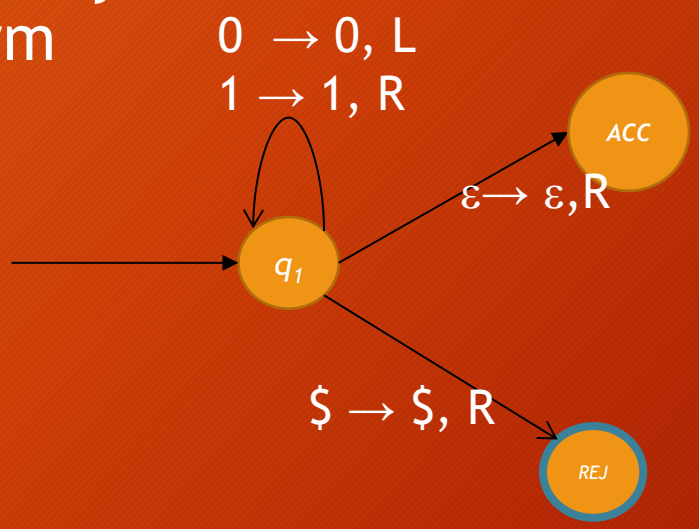
- Maszyna z 3 taśmami:

1. Jedna dla maszyny Turinga (zakodowany program), którą UTM symuluje (stała)
2. Jedna dla danych wejściowych (zmienna – zapis danych wyjściowych – tu jest zapisywane wyjście UTM i wynik obliczeń)
3. Jedna jako pamięć (zapamiętuje stan UTM) (zmienna - zapis stanu symulowanej maszyny)

Języki rozstrzygalne (ang. decidable)

- Maszyna Turinga nie zawsze kończy działanie w stanie akceptującym lub odrzucającym

- Przykład
  - 1111ε - akceptuje
  - $0011 – odrzuca
  - 1100 – krąży w nieskończoność!



Języki rozstrzygalne maszyną Turinga

Definicja. Maszyna Turinga, która niezależnie od tego, co znajduje się na taśmie zatrzymuje się – określana jest jako „maszyna rozstrzygająca” (ang. decider).

Język *L* jest rozstrzygalny w sensie Turinga (Turing-decidable), jeśli istnieje maszyna Turinga M, która rozstrzyga o przynależności do L.

Rozstrzyga, to znaczy, że:

- jeśli $w \in L$ => M kończy w stanie akceptującym
- Jeśli $w \notin L$ => M kończy w stanie odrzucającym

Język rozpoznawalny przez MT

- Maszyna M rozpoznaje *(nie rozstrzyga!!!)* język L, jeśli
  - Dla każdego słowa z *L* kończy w stanie akceptującym
  - Dla każdego słowa spoza L kończy w stanie odrzucającym lub ulega zapętleniu
- Inaczej jeśli rozstrzyga lub zapętla się → otwiera to pytanie, czy jeśli zaczekamy wystarczający długo, to maszyna się zatrzyma?
- Zbiór języków rozpoznawalnych $\supset$ zbiór języków rozstrzygalnych

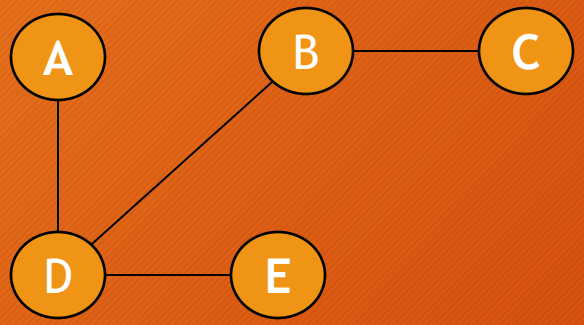
Przykłady (wysokopoziomowe) – nr 1

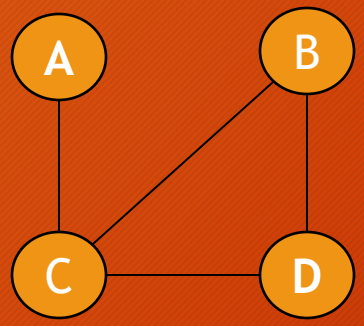
- Mamy dwa programy komputerowe. Każdy dostaje na wejściu jedną stronę tekstu (*N* znaków). Maszyna Turinga może dać każdemu z tych programów jakiekolwiek dane. Maszyna akceptuje, jeśli dla tych samych danych programy te dają ten sam wynik, inaczej odrzuca.
- Czy taka maszyna T jest maszyną rozstrzygającą?

Odpowiedź:

TAK. Maszyna może sprawdzić wszystkie możliwe wartość wejścia. Jeśli alfabet liczy sobie k znaków, musiałaby sprawdzić N^k różnych wejść (tzw. wariacja z powtórzeniami)

Przykłady wysokopoziomowe





- Drzewa możemy zapisać w postaci incydencji #A-D#D-ABE#B-DC#E-D (i jakkolwiek przekodować binarnie).
- Czy istnieje maszyna Touringa rozpoznająca język zawierający same drzewa?
  - Tak. Sprawdzamy czy z każdego węzła są połączenia z innymi węzłami, sprawdzamy cykle i w zależności odrzucamy lub akceptujemy

Rozstrzyganie cd.

- A_{DFA} = {<D, w> : D jest det. aut. skończ. akceptującym słowo *w*}
- Czy istnieje maszyna Turinga, która rozpoznaje taki język

Rozwiązanie: Tak. Symulowanie automatu deterministycznego przez maszynę Turinga jest możliwe (zapis automatu D i słowa mogą być w oddzielnych częściach taśmy lub na oddzielnych taśmach).

Symulacja automatu przez odczytanie wszystkich znaków tekstu i sprawdzenie stanu automatu – rozstrzyga czy dana para automat i słowo należy do języka lub do niego nie należy.

Rozstrzyganie

- Język L={<G,w>: G – gramatyka bezkontekstowa generująca w}
- Czy istnieje maszyna Turinga rozstrzygająca problem przynależności do L

Odp. TAK. Rozumowanie podobne, jak wcześniej.

Nierozstrzygalność (zaznaczenie)

- A_{TM}={<M,w>:M jest maszyną Turinga akceptującą słowo w}
- Czy A_{TM} jest rozstrzygalny, tj. czy mając daną maszynę Turinga, łańcuch w, uniwersalna maszyna Turinga może rozstrzygnąć w skończonym czasie czy *w* jest akceptowane przez M.
- Niestety nie wiadomo. Maszyny Turinga jak wiemy z przykładu mogą wpadać w nieskończone pętle

Zbiory przeliczalne i nieprzeliczalne

Definicja. Zbiór jest przeliczalny – zbiór równoliczny ze zbiorem liczb naturalnych

Równoliczny – taki, którego każdemu elementowi możemy przypisać inną liczbę naturalną

Przeliczalne są zbiory liczb całkowitych, iloczyn Z x Z

Zbiór liczb rzeczywistych jest nieprzeliczalny

- Argument diagonali Cantora
- *1 2 3 4 5…*
- a_1 3 6 5 4 1
- a_2 8 7 5 2 3
- a_3 6 1 1 5 2
- a_4 8 2 6 9 4
- …

- Dobierając ciąg różnych wartości od danych z diagonali konstruujemy coś czego nie ma w zbiorze. Zawsze możemy stworzyć coś czego nie wyliczyliśmy

Czy zbiór funkcji f: $Z^+ \rightarrow Z^+$ przeliczalny?

- Odp. NIE.
- | | f1 | f2 | f3 | f4 | … |
- | 1 | 19 | 834 | 11 | 1 | … |
- | 2 | 5 | 7 | 2 | 10 | … |
- | 3 | 127 | 9 | 3 | 33 | … |
- | 4 | 58 | 13 | 8 | 49 | |
- 5

czyli f(1)<>19, f(2)<>7…

Nierozstrzygalność

- Czy zbiór wszystkich języków nad alfabetem {0, 1} jest przeliczalny?
- Nie! Argument diagonalny Cantora: wiersze języki, kolumny słowa, na przecięciu 0 – należy do języka, 1 nie należy → na podstawie elementów na diagonali konstruujemy język nie należący do tego wykazu.
- A zatem musi istnieć język, który nie jest rozpoznawalny przez maszynę Turinga.

Uwaga końcowa

- Wiadomość zła: istnieją więc problemy, których nie da się mechanicznie rozwiązać
- Wiadomość dobra: bardzo wiele problemów da się jednak rozwiązać mechanicznie

KONIEC

„Trudne”

- Trudne – takie, dla których nie ma algorytmu (deterministycznego) rozwiązującego problem w czasie wielomianowym

Elementy teorii złożoności obliczeniowej

P, NP, NP-complete

Klasy problemów (wstęp)

Klasa	Czas rozwiązania	Czas sprawdzenia rozwiązania
P	Wielomianowe	Wielomianowe
NP (non-deterministic polynomial)	W praktyce: wykładniczy (przynajmniej na razie ;-)) – „brute force”. W teorii: wykładniczy wielomianowy algorytm niedeterministyczny	Wielomianowy
NP-Complete	Taki problem z NP, do którego można sprowadzić każdy problem tej klasy	Wielomianowy

Złożoność rozwiązania > złożoności sprawdzenia rozwiązania

Problem SAT (boolean satisfiability)

Dane jest wyrażenie logiczne w postaci CNF

- $\Psi = \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_n$
  - „$\wedge$” - koniunkcja
- *Np.* $\Psi_i = (x_1 \vee x_2 \vee x_3')$ – alternatywa maks. K zmiennych i/lub ich negacji ($'$)
- Czyli mamy $\Psi(x_1, x_2, \ldots, x_K) = \Psi_1(X) \wedge \Psi_2(X) \wedge \ldots \wedge \Psi_n(X)$, gdzie $X=(x_1, \ldots, x_K)$

Zadanie: Znaleźć wektor wartości logicznych X, dla których wyrażenie *Ψ jest prawdziwe*

Najprostszy algorytm

- Wiadomość dobra: problem jest rozstrzygalny (zawsze można znaleźć rozwiązanie!), bo jest algorytm znajdujący rozwiązanie (lub stwierdzający, że wyrażenie jest sprzeczne):
- **Podstawienie w $X=(x_1,\ldots, x_K)$ wszystkich możliwych kombinacji wartości logicznych**
- Złożoność 2^K

Twierdzenie Cook’a-Levin’a

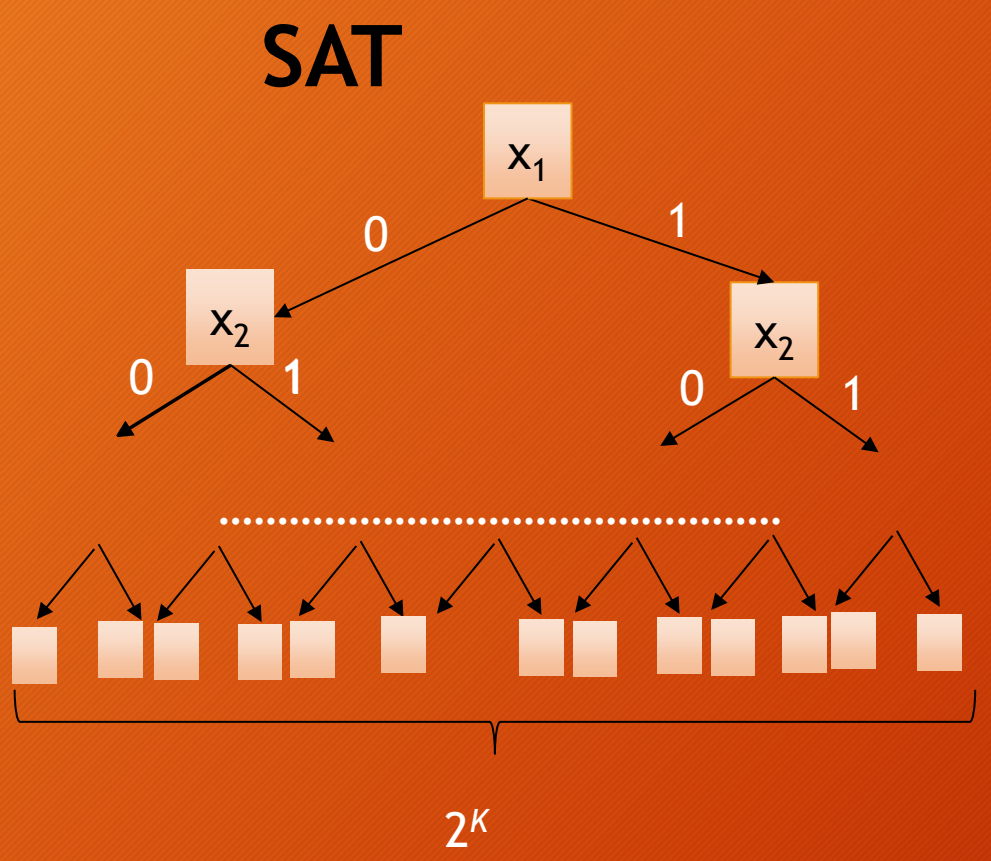
Problem SAT jest NP-zupełny.

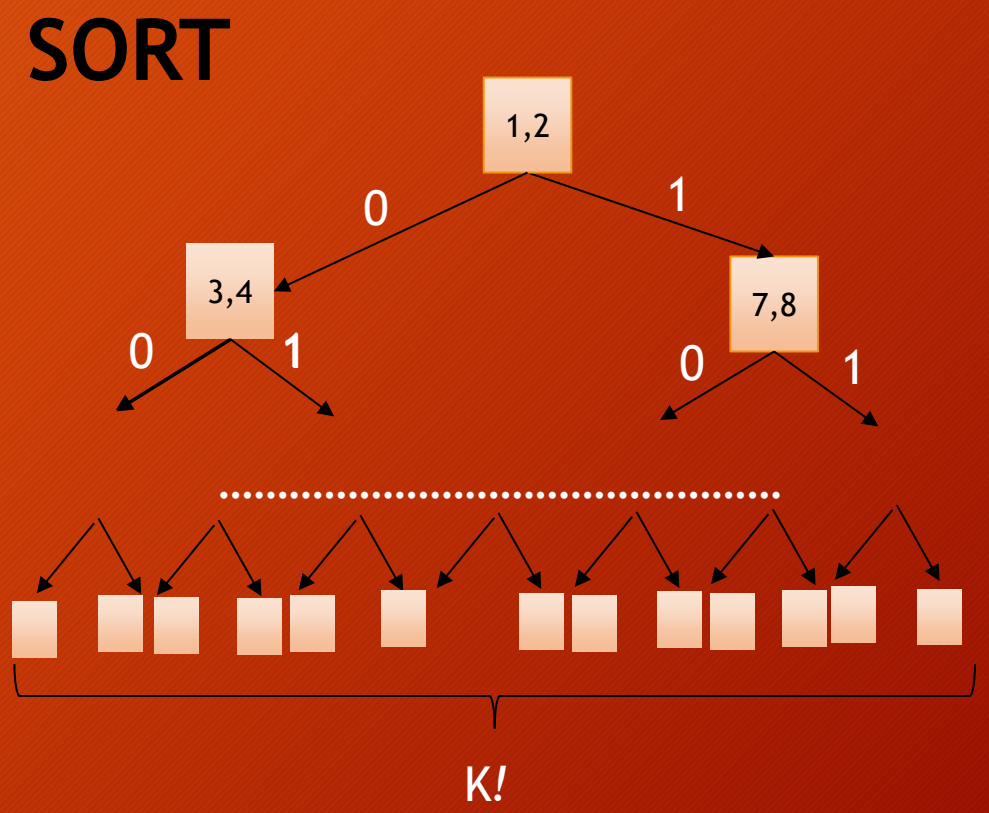
Czyli każdy problem klasy NP można sprowadzić do tego problemu!

Na czym polega trudność problemu SAT?

- Powiązania wyrażeń składowych (te same zmienne)
- Nie wiadomo, jak ukierunkować poszukiwanie rozwiązania – nie wynika to z samych zapisów badanego wyrażenia

Problem SAT a problem sortowania





Cykl i ścieżka Hamiltona

Definicja

Ścieżka Hamiltona – ścieżka przechodząca przez każdy z wierzchołków grafu dokładnie raz

Cykl Hamiltona – cykliczna ścieżka Hamiltona

Własność

- Należy do kategorii problemów „trudnych” – i do klasy problemów NP-zupełnych (ang. NP-complete).
- Nie jest znany algorytm wielomianowy, który znajduje (lub stwierdza nie istnienie) ścieżkę lub cykl Hamiltona dla dowolnego grafu

Algorytmy (1)

- Sprawdzenie wszystkich permutacji węzłów, jeśli dla każdej pary kolejnych węzłów istnieje łącząca krawędź, to mamy cykl Hamiltona
  - Czas: prop. ~N!

Algorytmy (2)

Zmodyfikowane przeszukiwanie w głąb

Zaczynamy od dowolnego wierzchołka v_0, który zapamiętujemy. Odpalamy: **DFS-HAM**(v)

DFS-HAM(v) – procedura rekurencyjna

1) Połóż-na-stosie(v)

2) Jeśli *h* (wysokość stosu) = liczba węzłów grafu i $v_0 \in$ następnik(*v*) to mamy cykl Hamiltona (na stosie), możemy zapisać *w przeciwnym razie*

3) Dla wszystkich *v'* $\in$ następnik(*v*) uruchom *DFS-HAM*(*v'*)

4) Zdejmij *v* ze stosu

SAT solver

1) Wyrażamy w postaci wyrażenia logicznego
2) Rozwiązujem problem spełnialności tej formuły (SAT) – specjalnym narzędziem

Konwersja HAM do problemu SAT

Klucz do sukcesu(!): p_{ij} – prawda, jeśli i jest j-tym węzłem na ścieżce Hamiltona, czyli

ścieżka Hamiltona składa się z tych węzłów *i*, dla których p_{ij} jest prawdą, w kolejności wyznaczonej przez *j*

Struktura grafu dana jest funkcją *N*(*i*) – zbiór następników

Konwersja HAM -> SAT

$$H(n,N) = \mathcal{P}(n) \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n-1} \left(\bigvee_{v \in N(i)} p_{v,j+1} \right)$$

- Szukamy takich p_{ij}, że H(n, N) jest prawdziwe
  - *P*(*n*) – warunek, że p_{ij} obejmujące wszystkie węzły, ale każdy tylko raz
  - Druga część koniunkcji oznacza, że ścieżka przebiega krawędziami grafu (nie jest tylko ciągiem n-węzłów grafu)
- Jeśli ścieżka złożona z krawędzi, dla których (p_{ij} , $p_{i,j+1}$)=prawda (problem SAT)

Warunek prawidłowego zapisu w „systemie” p_{ij}

$$\mathcal{P}(n) = \bigwedge_{j=1}^{n} \left(\bigvee_{i=1}^{n} p_{i,j} \wedge \bigwedge_{k \in \{1..n\} \setminus \{i\}} \neg p_{k,j} \right) \wedge \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{n} p_{i,j} \wedge \bigwedge_{k \in \{1..n\} \setminus \{j\}} \neg p_{i,k} \right)$$

- *Przypomnienie*: ***i*** – pozycja na ścieżce, ***j*** nr węzła na *i*-tej pozycji
- Pierwszy człon głównej koniunkcji…
- Drugi człon głównej koniunkcji…

1 2 3 4 – pozycja na ścieżce H.
1 0 0 0 1
2 0 1 0 0
3 0 0 1 0
4 1 0 0 0

Teraz...

- SAT(*H*(n, N))

Podobieństwo – tzw. programowanie binarne

Dane:

- $F(X)$: $\{0, 1\}^N$ -> R, gdzie X – wektor N zer i jedynek
- $G(X) = 0$ – ograniczenia np. $AX + B = 0$ (A macierz N x N)

Zadanie:

- Znaleźć takie X, że $F(X)$ jest minimalne i spełnione są ograniczenia $G(X) = 0$

Inne problemy NP-zupełne

Problem plecakowy - sformułowanie

Dany jest:

- Pojemność (plecaka) – C_K.
- Zbiór n przedmiotów (o nr 1… n) o objętości c_i i wartości v_i

Zadanie:

- Wybrać takie przedmioty, które zmieszczą się w plecaku i których łączna wartość (suma) będzie maksymalna

Problem plecakowy - rozwiązanie

Niech x_i = 1 (prawda), gdy *i*-ty przedmiot jest w plecaku i 0 (fałsz) – zmienna decyzyjna, $X=(x_1,\ldots, x_n)$

- W wersji „przedmiotów” sypkich – działa strategia zachłanna – wsypujemy kruszywa od najdroższego do najtańszego – im drożej wypełnimy każdą jednostkę objętości tym lepiej.
- W binarnej wersji problemu musimy przejrzeć wszystkie wartości 2^n wartości wektora X. Co do zasady ograniczenia na pojemność plecaka można zapisać w postaci wyrażeń logicznych… wi

Programowanie binarne

Maksymalizuj $\Sigma_{i=1\ldots n}\ x_i * v_i$

- Przy ograniczeniu na pojemność plecaka: $x_i * c_i <= C_k$

N-hetmanów

- Jak ustawić *n* hetmanów na planszy o wymiarach $n \times n$ tak, by żaden nie był zagrożony biciem
- p_{ij} – prawda, tzn. hetman na polu szachownicy o współrz. (i, j)

Ustawienie hetmanów spełniające wyrażenie P(n) $\wedge$ D(n) jest rozwiązaniem problemu. *P*(*n*) jak w problemie ścieżki Hamiltona.

$$\mathcal{D}(n) = \bigwedge_{k=1}^{n} \bigwedge_{l=1}^{n} \Bigg(\bigwedge_{1 \le i \ne k < n} \; \bigwedge_{1 \le j \ne l \le n-i} \big((\neg p_{j,i+j} \vee \neg p_{l,k+l}) \wedge (\neg p_{i+j,j} \vee \neg p_{k+l,l}) \wedge$$

$$(\neg p_{j,n-1-(i+j)} \vee \neg p_{l,n-1-(k+l)}) \wedge (\neg p_{i+j,n-1-j} \vee \neg p_{k+l,n-1-l}) \big) \Bigg)$$

Kolorowanie grafu

Dany jest graf G=(V, E) i stała całkowita k.

Czy istnieje taka funkcja *c*: *V* -> {1,…, *k*}, że

Dla każdego $v \in V$, dla każdego $u \in N(v)$ $c(v) <> c(u)$

Jeśli przypiszemy kolory wierzchołkom to, koniec każdej krawędzi powinien być tego samego koloru.

Problem kliki

Dany jest graf nieskierowanych G=(V, E)

Def. $S \subseteq V$ nazywamy kliką, gdy każdy wierzchołek z S jest połączony z innym wierzchołkiem z S (każdy w klice jest w relacji z każdym).

Problem: Czy graf *G* zawiera klikę *S* o wielkości przynajmniej k, czyli taką, że $|S| \geq k$.

Algorytm wykładniczy dla problemu SAT, k-zmiennych

- Dla wszystkich możliwych kombinacji
  - X=(prawda, …, fałsz)

- Podstaw X do wyrażenia w CNF i sprawdź czy CNF(X)=prawda, wtedy STOP, w przeciwnym razie kontunuuj przeszukiwanie

Algorytmy deterministyczne w zmaganiach z problemami NP-zupełnymi

- W podobny sposób można konstruować algorytmy deterministyczne dla każdego z problemów omówionych do tej pory i opisanych jako NP-zupełne (nie mylić z NP!!!)
- I faktycznie mają one niewielomianową złożoność obliczeniową

Obserwacja P i NP

- Z przykładów widać, że w NP są problemy trudniejsze, niż te, które znamy z klasy P, tj. te, które umiemy rozwiązać tylko brutalną siłą przeglądania wszystkich wariantów rozwiązań!!!

$P \subseteq NP$, ale czy przypadkiem nie P = NP

- $P \subseteq NP$ – algorytmy detetrministyczne będą szczególnym przypadkiem niedeterministycznych, więc jeśli mamy „normalny” deterministyczny algorytm rozwiązujący problem w czasie wielomianowym, to mieści się on też w klasie NP.
- Jest jedno, ale: nikt nie wykazał, że dla tych problemów z NP, dla których nie znamy deterministycznego algorytmu wielomianowego, taki algorytm nie istnieje.
- Tego nie wiemy! I stąd słynny problem,
  - czy P=NP!!!

Problemy NP-zupełne (NPC) – definicja

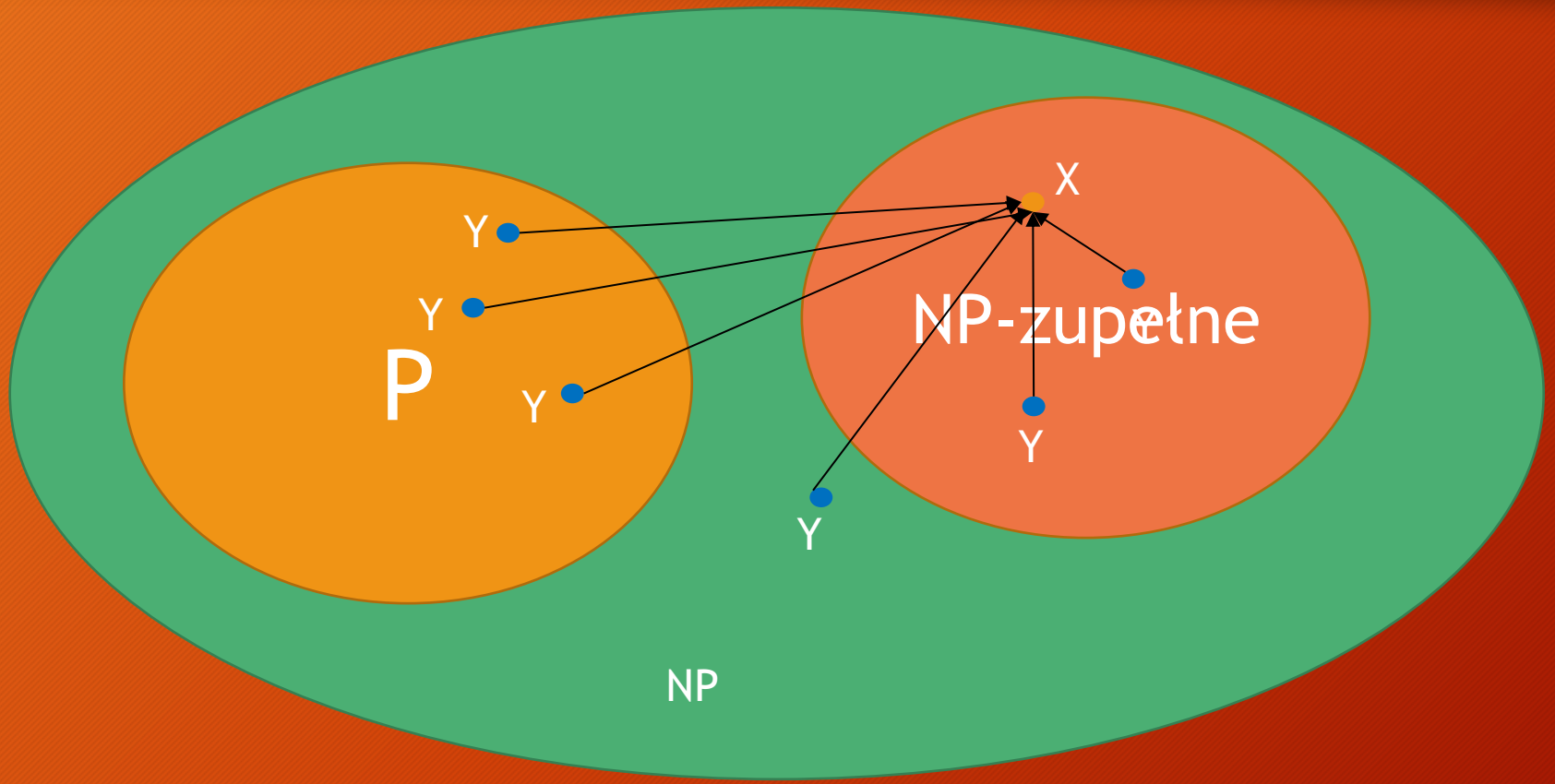
Def. Problem X należy do problemów NP-zupełnych, jeśli:

Warunek 1. $X \in NPC$.

Warunek 2. Każdy problem Y z NP można (sprowadzić) zredukować w czasie wielomianowym do X

Uwaga – problem łatwiejszy redukujemy do trudniejszego

Ilustracja warunku 2 NP-zupełności



NP-zupełność – interpretacja, wyjaśnienia

- Definicja NPC postuluje, że są problemy „reprezentatywne” czy też o „sile wyrazu” pozwalającej wyrazić wszystkie inne problemy rozwiązywalne niedeterministycznym algorytmem w czasie wielomianowym
- NPC (NP-zupełne) to te problemy, w języku których / w postaci których można wyrazić wszelkie (!!!) inne problemy klasy NP, **w tym problemy klasy P!!!!**

NPC - ilustracje

- Wiemy (choć tego nie pokazaliśmy!), że do NPC należą problemy opisane już na poprzednim wykładzie – kolorawnie grafu, cykl Hamiltona, klika, SAT
- Oznacza to, że w postaci problemu cyklu Hamiltona czy SAT można wyrazić problem sortowania, najkrótszej ścieżki (i każdy inny z P i z NP)

Sortowanie i problem cyklu Hamiltona

- Zauważmy, że problem sortowania polega na znalezieniu takie ciągu indeksów, że wartości w tablicy / liście pod tymi indeksami są uporządkowane rosnąco
- (idx_1,…, idx_n), że (T[idx_1], T[idx_2], …, T[idx_n]) są uprządkowane

- Na tym samym polega problem cyklu Hamiltona… nawet sprawdzenie jest takie samo (sprawdzenie czy pary wierzchołków są elementami relacji sąsiedztwa)
- Co nie znaczy, że to jest dobry sposób na sortowanie ;-) , ale na pewno problem sortowania można sprowadzić do problemu Hamiltona, więc (jak pamiętamy i do SAT)

Koniec

Automaty, języki formalne

Gramatyki, automaty, hierarchia Chomsky’ego

Automat skończony (determ. i niedet.)

Automaty = (S, ∑, T, Q, A),

- S – stany
- ∑ - alfabet
- T - funkcja przejść S x ∑ → S (determin.) lub S x ∑ → P(S), gdzie [P(S) – zbiór wszystkich możliwych podzbiorów stanów zbioru S (niedetermin.)
- Q – zbiór stan początkowy
- A – zbiór stanów akceptujących

Języki formalne. Hierarchia Chomsky’ego

Język formalny

- $\sum$ - alfabet
- Język L – zbiór słów zbudowanych ze znaków alfabetu $\sum$, tj. $L \subseteq \sum^*$
- $\sum^*$ - zbiór wszystkich możliwych ciągów liter (kolekcji) złożonych ze znaków alfabetu $\sum$
  - $A^* = \{w_1 w_2 \ldots w_k, k \geq 0$ i każde z $w_i \in A\}$ //tzw. gwiazdka Klenee'go
- Słowa = zdania

Gramatyka

Gramatyka - skończony zestaw reguł, zgodnie z którymi są generowane słowa (zdania)

Formalne G=(T, N, S, P)

- T - alfabet symboli terminalnych [faktycznych symboli/słów]
- N - alfabet symboli nieterminalnych, $S \cap T = \varnothing$ [jednostek składniowych]
- $S \in N$ - symbol startowy
- P - zbiór tzw. produkcji (reguł tworzenia słów/zdań)

  $P \subseteq \{a \rightarrow b$: a, b $\in (N \cup T)^*$ i a $\notin T^*\}$
- Produkcja - relacja (podstawienie) między ciągiem symboli nieterminalnych i terminalnych a ciągiem symboli terminalnych i/lub nieterminalnych
- Warunek: a $\notin T^*$ oznacza, że nie możemy dokonywać podstawień w miejsce samych symboli terminalnych (bo nie byłyby one terminalne)
- Produkcje - generator słów/ zdań

Gramatyka – interpretacja intuicyjna

- Od gramatyki zależy złożoność języka
- W istocie zależy ona od dopuszczalnych w danej klasie języków podstawień czyli produkcji
- Klasy języków formalnych wyróżnione są gramatyką tych języków

Przykład gramatyki

- N – symbole nieterminalne

{Instrukcja, instrukcja-if, instrukcja-while, instrukcja-złożona, warunek}

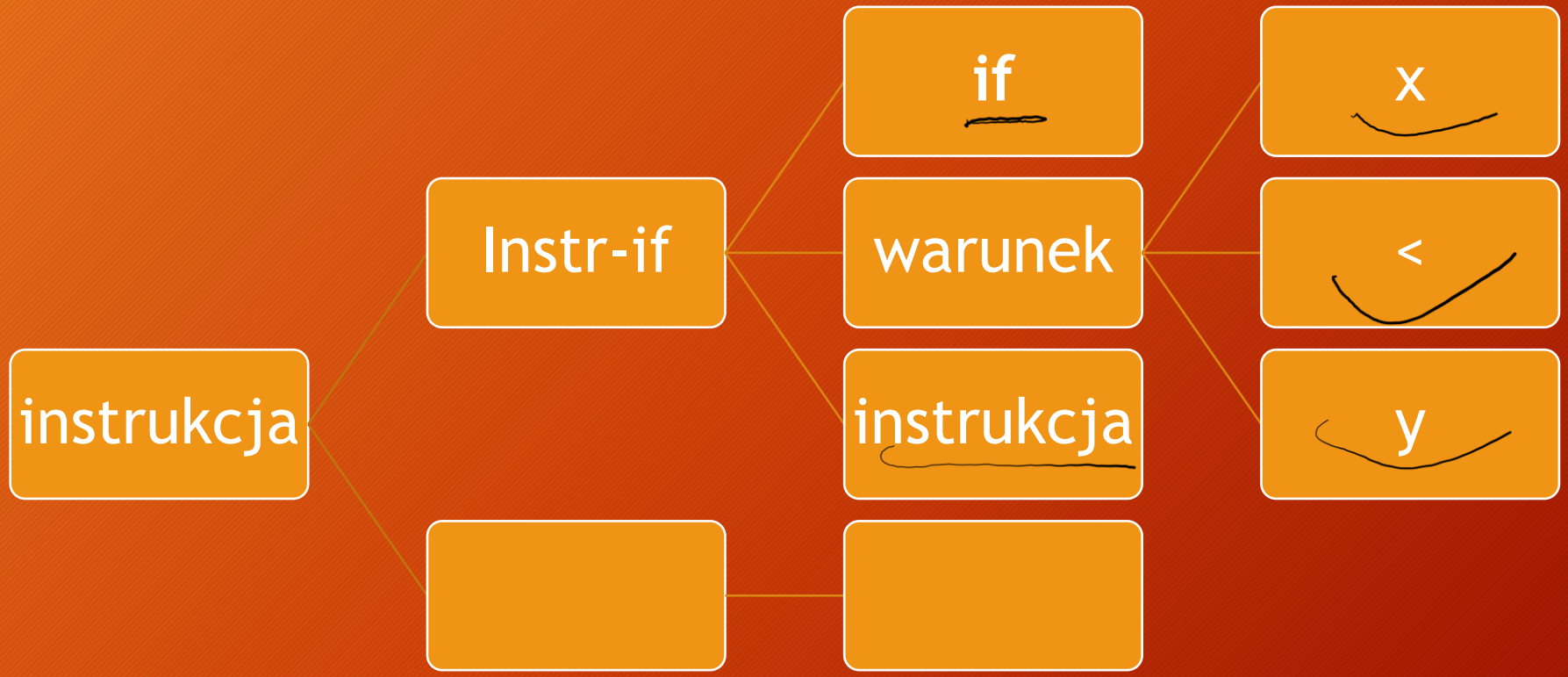
- T – symbole terminalne

{if, while, nazwa-zmiennej, `<`, `{`, `}` , liczba-całkowita }

Produkcje

- Instrukcja → instrukcja-if
- Instrukcja → instrukcja-while
- Instrukcja → instrukcja-złożona
- Instrukcja-złożona → { instrukcja }
- Instrukcja-if → if warunek instrukcja
- Warunek → nazwa-zmiennej < nazwa-zmiennej
- Warunek → nazwa-zmiennej < liczba-całkowita

Drzewo podstawień (przykład), generowanie języka



Język generowany przez gramatykę

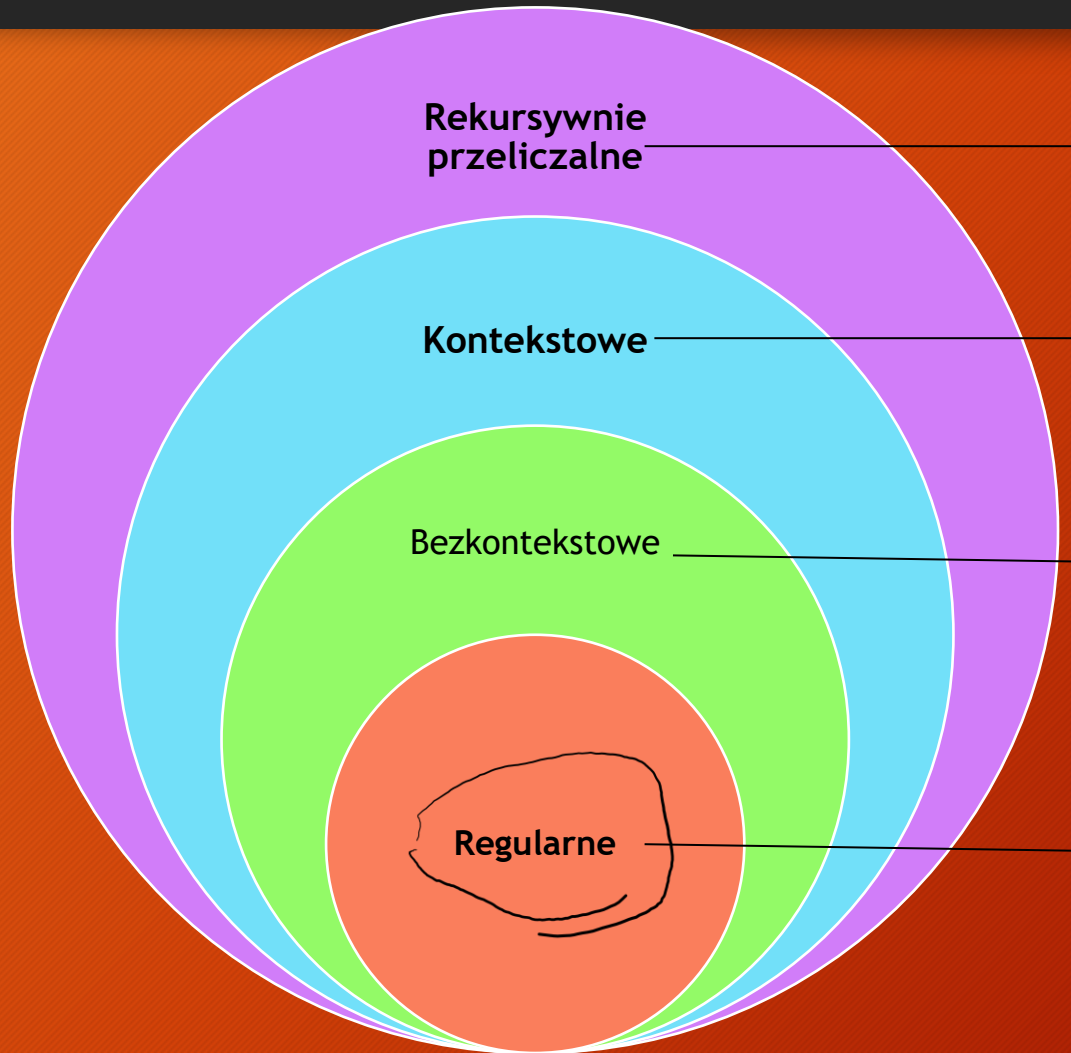
- Dana jest gramatyka G=(T, N, S, P)
- Zbiór wszystkich słów $w \in \Sigma^*$, które powstają przez zastosowanie produkcji z gramatyki zaczynając od symbolu startowego
- Oznaczenie: L(G)

Operacje na językach

- A i B - języki
- Suma mnogościowa słów: $A \cup B = \{w: w \in A \text{ or } w \in B\}$
- Konkatenacja (słów): $A \circ B = \{ vw: v \in A \text{ and } w \in B\}$ /zwykle pomijamy $\circ$/.
- Tzw. gwiazdka Kleen'ego: $A^* = \{w_1 w_2 \ldots w_k, k \geq 0$ i każde z $w_i \in A\}$.
  - Przykład
    - $B = \{ab\} \Rightarrow B^* = \{\varepsilon, ab, abab, ababab,...\}$
    - $C = \{a, bb\} \Rightarrow C^* = \{\varepsilon, a, bb, aa, abb, bba, bbbb, aaa,...\}$
    - $A = \{a,b\} \Rightarrow A^* = \{\varepsilon, a, ab, aa, bb, abb, aabb, \ldots\}$
  - Wszystkie możliwe łańcuchy złożone ze znaków / ciągów znaków należących do A

Klasy języków wg Chomsky’ego

Hierarchia Chomsky’ego



Maszyna Turinga

Liniowo ograniczony automat

Niedeterministyczny automat ze stosem (ang. pushdown automaton)

Automat skończony

- $(AB)^n$
- A^NB^N

Języki regularne

- Języki regularne – języki generowane przez gramatyki, z produkcjami postaci:
  - $A \rightarrow b\ B$ lub $A \rightarrow b$ gdzie $A, B \in N$, a $b \in T^*$

  lub
  - $A \rightarrow B\ b$ or $A \rightarrow ß$ gdzie $A, B \in N$ and $b \in T^*$

- {a, b}
- $S \rightarrow a\ B$
- $B \rightarrow b\ A \mid b$
- $A \rightarrow a\ B \mid a$

$a\ B \rightarrow a\ b$

$a\ \underline{B} \rightarrow a\ b\ \underline{A} \rightarrow a\ b\ a\ \underline{B} \rightarrow a\ b\ a\ b$

$\rightarrow a\ b\ a\ b\ A \rightarrow a\ b\ a\ b\ a\ b$

Języki regularne i automaty

- Tw. Kleeny’ego. Każdy język akceptowany przez automat skończony jest językiem regularnym, dla każdego języka regularnego istnieje akceptujący go automat skończony.

Lemat o pompowaniu (ang. pumping lemma)

Dla **każdego języka regularnego** L, **istnieje (!!!)** taka liczba naturalna *k*, że jeśli $s \in L$ i $|s| > k$, to s można podzielić na trzy części:

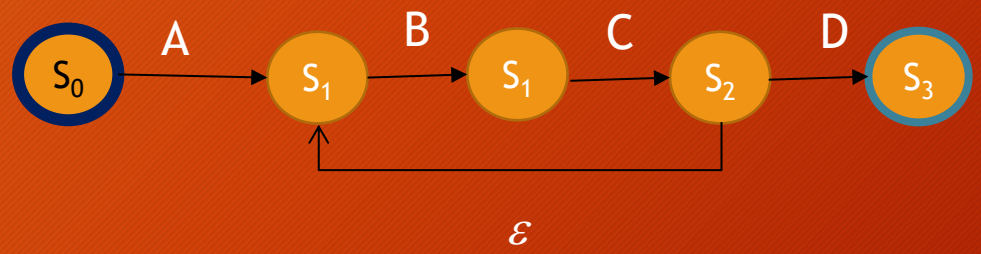
s=xyz (konkatenacja łańcuchów), że

- $|xy| \leq k$
- $|y| > 1$
- dla każdego $m \geq 0$ $xy^m z \in L$

- $x\ y^1\ z$ …. $x\ y^2\ z$

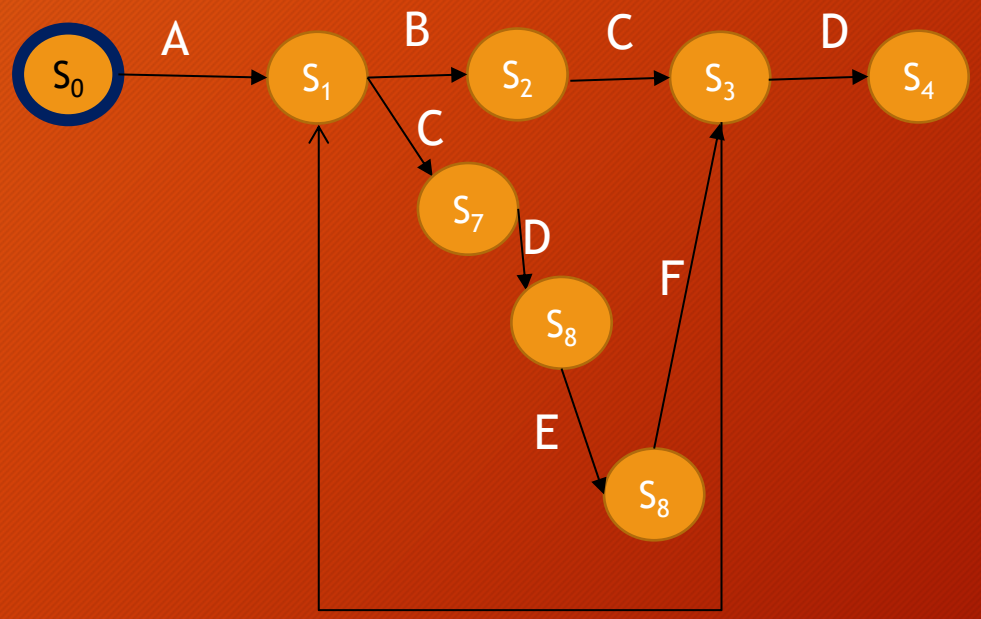
Ilustracja (1)

- A $(BC)^i$ D
- A BC D → A $(BC)^1$ D
- A BC BC D → A $(BC)^2$ D
-
- A BC BC BC D
- k=4
- X=A
- Y=BC
- Z=D



Ilustracja (2)

- A BC CDEF D
- x y z
- A $(BCCDEF)^k$ D



ε

Wyrażenia regularne

- R jest wyrażeniem regularnym, jeśli R jest
- 1. $\emptyset$ - wyrażeniem pustym.
- 2. ε.
- 3. $a \in \Sigma$.
- 4. $X \cup Y$, gdzie X i Y są wyrażeniami regularnymi
- 5. X Y, (konkatenacja), gdzie X i Y są wyrażeniami regularnymi
- 6. X^*, gdzie X jest wyrażeniem regularnym

Przykłady wyrażeń regularnych

- A={a b}
- R=$(ab)^*b$ → L(R)={abb, ababb, …}
- R=$(a^*) \cup (b^*)$ → L(R)={a, aa, aaa, …, b, bb, bbb}

Zamiana wyrażenia regularnego w niedeterm. automat skończony

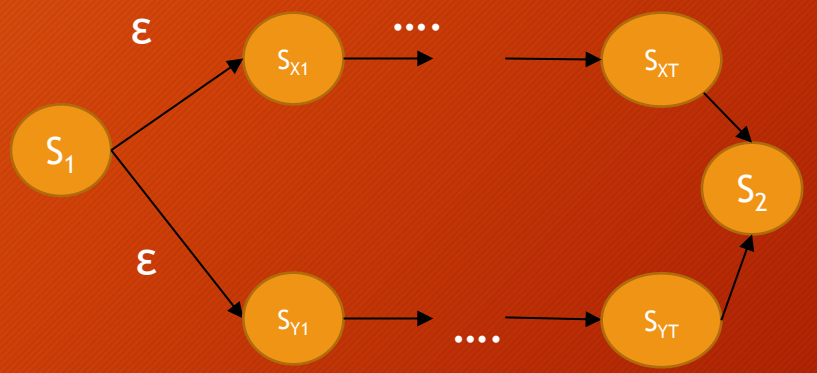
Niech dane będą automaty niedet. odpowiadające wyrażeniom X i Y

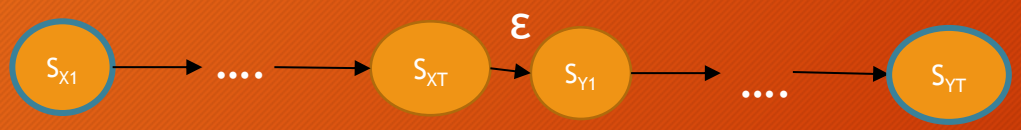
…

- 4. X ∪ Y, gdzie X i Y są wyrażeniami regularnymi
- 5. X Y, (konkatenacja), gdzie X i Y są wyrażeniami regularnymi
- 6. X^*, gdzie X jest wyrażeniem regularnym

Automaty rozpoznające wyrażenia regularne

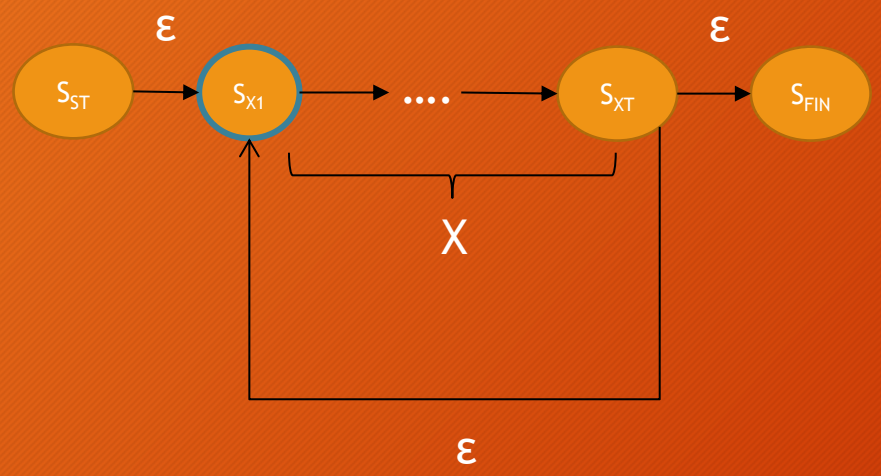
- X ∪ Y
- X Y



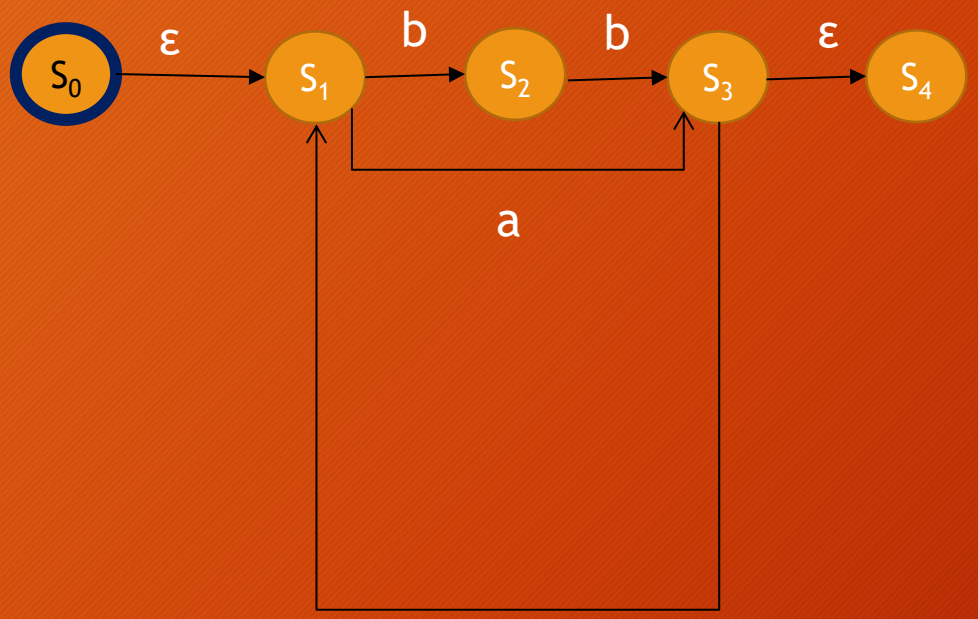


Automaty rozpoznające wyrażenia regularne

- X^*



Przykład – zbudować automat rozpoznający (a ∪ bb)*



Języki i wyrażenia regularne

- Język *L* jest regularny, jeśli opisuje go jakieś wyrażenie regularne.

Konkluzja finalna

- Automat skończony
- Wyrażenie regularne
- Gramatyka regularna

Mogą być alternatywnymi sposobami reprezentacji tego samego języka regularnego

Języki bezkontekstowe

Gramatyka bezkontekstowa

Gramatyka bezkontekstowa (ang. Context-Free Grammar, CFG) $G=(V, \Sigma, R, S)$, gdzie

- V (pierwotne: N) – zbiór symboli nieterminalnych (zwanych niekiedy zmiennymi)
- Σ – zbiór symboli terminalnych (rozłączny z V),
- R – zbiór produkcji postaci $V \rightarrow (V \cup \Sigma)^*$
- S – zmienna (symbol) początkowy $S \in V$.

Przykłady

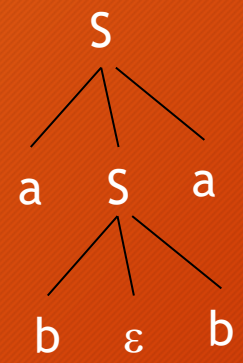
- Gramatyka G, która generuje palindromy o parzystej długości złożone z liter {a, b}
- $S \rightarrow aSa \mid bSb \mid \varepsilon$
- Gramatyka G, która generuje jw. Ale o nieparzystej długości
- $S \rightarrow aSa \mid bSb \mid a \mid b$

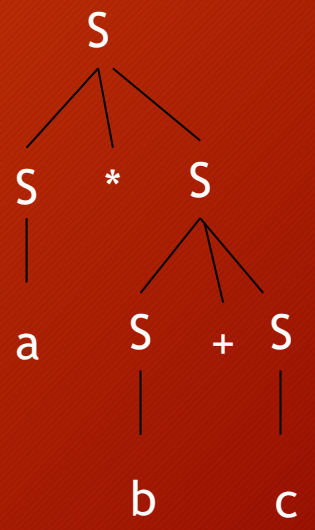
Drzewo parsowania (parse tree) [wyprowadzania]

- Dla danej gramatyki CFG G=(V, Σ, R, S), drzewem parsowania nazywamy takie drzewo, dla którego
  - Korzeniem drzewa jest S – symbol startowy
  - Wewnętrzne wierzchołki są oznaczone symbolami nieterminalnymi (zmiennymi)
  - Liście są oznaczone symbolami terminalnymi lub ε
  - Liście oznaczone ε muszą być jedynymi dziećmi swojego rodzica
  - Jeśli wierzchołek jest oznaczony jako A, a jego dzieci symbolami B_1, B_2 , …, B_n, to produkcja $A \rightarrow B_1, B_2 , \ldots, B_n$ należy do R (zbioru produkcji gramatyki G)

Przykład

- $S \rightarrow aSa \mid bSb \mid \varepsilon$





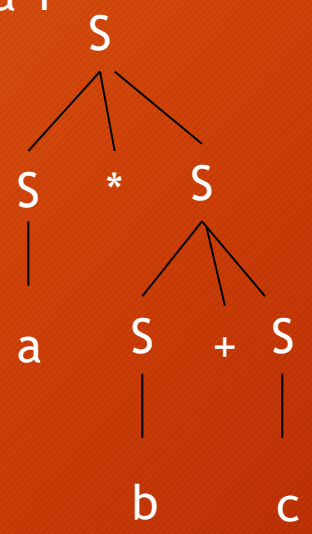
Przeszukać drzewo in-order by odczytać słowo / zdanie

Niejednoznaczność składniowa

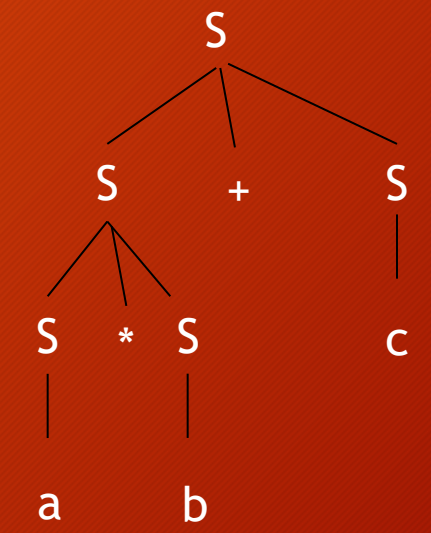
Przykład – produkcje pewnej gramatyki

- $S \rightarrow S * S$
- $S \rightarrow S + S$
- $S \rightarrow (S) \mid a \mid b \mid c$

Wersja 1



Wersja 2



Czyli ten sam łańcuch symboli można wygenerować stosując różne ciągi produkcji

a * b + c

Niejednoznaczność c.d.

Dane są produkcje

- $S \rightarrow AB$
- $A \rightarrow 00A \mid \varepsilon$
- $B \rightarrow 1B \mid \varepsilon$

Zauważmy, że łańcuch 0011 można wygenerować na dwa sposoby:

1) $S \rightarrow \underline{A}B \rightarrow \underline{00A}B \rightarrow 00B \rightarrow 001B \rightarrow 0011B \rightarrow 0011$ (skrajnie lewe podstawienia)
2) $S \rightarrow A\underline{B} \rightarrow A1\underline{B} \rightarrow A11\underline{B} \rightarrow \underline{A}11 \rightarrow 00\underline{A}11 \rightarrow 0011$ (skrajnie prawe podstawienia)

Niejednoznaczność CGF

CFG jest niejednoznaczna, jeśli istnieje

1) łańcuch, który ma dwie „lewo-skrajne” (ang. leftmost) ciągi wyprowadzeń
2) a w efekcie dwa różne drzewa podstawień (ang. parse tree)

Uproszczenia

Gramatyka nr 1

- $S \rightarrow 00S0 \mid D \mid B$
- $A \rightarrow 1A \mid 1$
- $B \rightarrow 0B \mid BB$
- $C \rightarrow 0D \mid B1$
- $D \rightarrow A$

Gramatyka nr 2

- $S \rightarrow 00S0 \mid A$
- $A \rightarrow 1A \mid 1$

Obie generują ten sam język
Język: $\{0^{2n}\ 1^{m}\ 0^{n}: n > 0, m > 1\}$

Symbole bezużyteczne:
1) Nieosiągalne ze startowego
2) Niegenerujące – takie, których podstawianie nie skończy się symbolami terminalnym

Eliminacja produkcji jednostkowych

- Ang. Unit productions
- $A \rightarrow B$, gdzie A i B to zmienne
- Jeśli mamy w gramatyce produkcję $A \rightarrow B$ oraz $B \rightarrow X$, to możemy zamiast nich wprowadzić produkcje $A \rightarrow X$

Eliminacja produkcji „zerowalnych” (nullable)

Przykład 1.

- S → 0S0 | 1A1
- A → 000 | ε

Po wyeliminowaniu: A → ε przez podstawienie

- S → 0S0 | 1A1 |11
- A → 000

Przykład 2.

S → 0S0 | 1A0A1

A → 000 | ε

Po wyeliminowaniu

- S → 0S0 | 1A0A1 | 10A1 |1A01 | 101
- A → 000

Postać normalna Chomsky’ego gramatyki bezkontekstowej

Gramatyka bezkontektowa jest w normalnej postaci Chomsky’ego (ang. skr. CNF), jeśli produkcje mają postać:

- $A \rightarrow BC$,
- $A \rightarrow a$, gdzie
- A, B, C – zmienne (symbole nieterminalne), B i C nie są symbolami startowymi
- a – symbol(e) terminalne

Dodatkowo jest zawsze określona produkcja $S \rightarrow \varepsilon$ (pusty symbol).

Czyli w uproszczeniu

$A \rightarrow BC \mid a \mid \varepsilon$

CNF - przykłady

Przykład 1

- $S \rightarrow AB \mid AA$
- $A \rightarrow a$
- $B \rightarrow BC \mid CC \mid b$
- $C \rightarrow a \mid c$

Przykład 2

- $S \rightarrow TU \mid BT \mid \varepsilon$
- $T \rightarrow AB \mid AT \mid c$
- $U \rightarrow BB \mid BU \mid c$
- $A \rightarrow a$
- $B \rightarrow b$

Drzewo podstawień dla CFG w CNF jest binarne

Zaleta CNF

- Wyprowadzenia każdego słowa o długości *n* symboli wymaga 2n – 1 podstawień

CNF a języki bezkontekstowe

Twierdzenie. Dowolny język bezkontekstowy może zostać wygenerowany przez gramatyka bezkontekstowa w postaci CNF (każdy język bezkontekstowy ma swoją gramatykę w CNF).

$S \rightarrow a\ B\ c\ A$
$\Rightarrow S \rightarrow D\ B\ E\ A \Rightarrow X\ Y$

Dowód: polega na pokazaniu, że każdą gramatykę ze zbiorem produkcji postaci bezkontekstowej można sprowadzić do CNF przez:

- Eliminację produkcji jednostkowych,
- Eliminację produkcji epsilonowych (puste podstawienia),
- Eliminując symbole terminalnych z podstawień, tam gdzie są podstawiane łącznie ze zmiennymi
- Eliminując nadmiar symboli przypisywanych (prawa strona) wprowadzając w razie dodatkowe zmienne

$S \rightarrow XY$
$X \rightarrow D\ B$
$Y \rightarrow E\ A$
$D \rightarrow a$
$E \rightarrow c$

Przykład przekształcenia CFG w CNF

Dana jest gramatyka bezkontekstowa ze zbiorem produkcji:

- S → 0A0 | 1B
- A → 1 | BB
- B → 111 | A0

1) Eliminujemy znaki terminalne z produkcji zawierających w przypisaniu zmienne

S → ZAZ | WB
A → 1 | BB
B → WWW | AZ
W → 1
Z → 0

2) Skracamy podstawienia 3 symboli wprowadzając nowe zmienne:

S → ZC | WB

A → 1 | BB

B → WD AZ

C → AZ

D → WW

W → 1

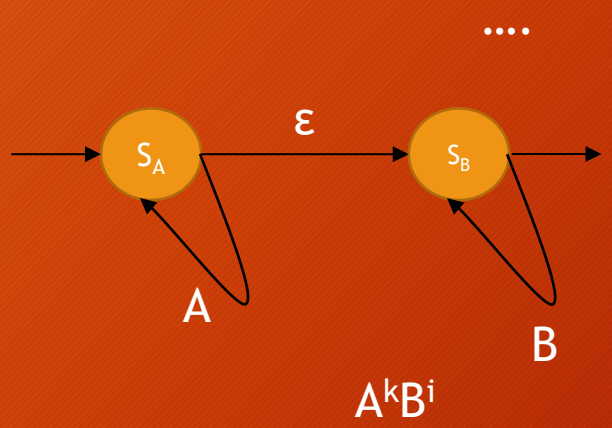
Z → 0

Automat ze stosem

Ang. Push-down automaton

Wstęp

- Automaty skończone rozpoznają tylko języki zwierające słowa (zdania) o powtarzalnej strukturze – vide lemat o pompowaniu.
- Ich pamięć jest ograniczona liczbą stanów
- W efekcie automat skończony nie rozpozna słów języka A^kB^k (np. AB, AAB,AABB, AAAABBBB, etc.)



Automat ze stosem (niedeterministyczny)

- Nieformalnie, automat różni się jedynie funkcją (relacją) przejścia, w której następny stan(y) zależy(ą) od:
  - Aktualnego stanu, wartości na wejściu i wartości na wierzchu stosu
  - Podczas przejścia można zapisać na stosie określony znak

Automat ze stosem – formalna definicja

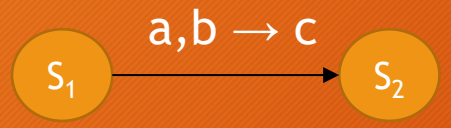
- Automat ze stosem PDA to szóstka uporządkowana: $(Q, \Sigma, \Gamma, \delta, q_0, F)$, gdzie
- Q, Σ, Γ i F zbiory skończone
- Q zbiór stanów,
- Σ alfabet wejściowy,
- Γ alfabet stosu (może być różny do wejściowego),
- δ funkcja (relacja) przejścia,
  - δ: $Q \times (\Sigma \cup \varepsilon) \times (\Gamma \cup \varepsilon) \rightarrow$ wszystkie podzbiory $Q \times (\Gamma \cup \varepsilon)$.
- $q_0 \in Q$ stan początkowy
- $F \subseteq Q$ zbiór stanów akceptujących

Rozumienie przejść

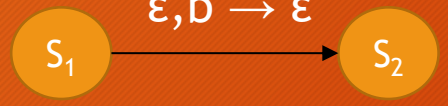
Dana jest wartość f-cji przejścia: $\delta(q_1, a, 1) = \{ (q_2, 0), (q_4, \varepsilon) \}$

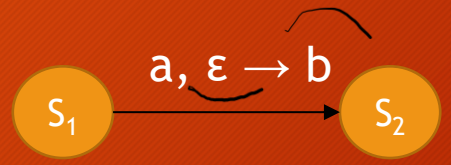
- Powyższe oznacza:
  - W stanie q_1, przeczytawszy symbol a, na wejściu i pobrawszy znak 1 ze stosu, idź do stanu q_2 zapisując zero na stosie lub q_4 nie zapisując nic na stosie.

Rozumienie przejść automatu ze stosem



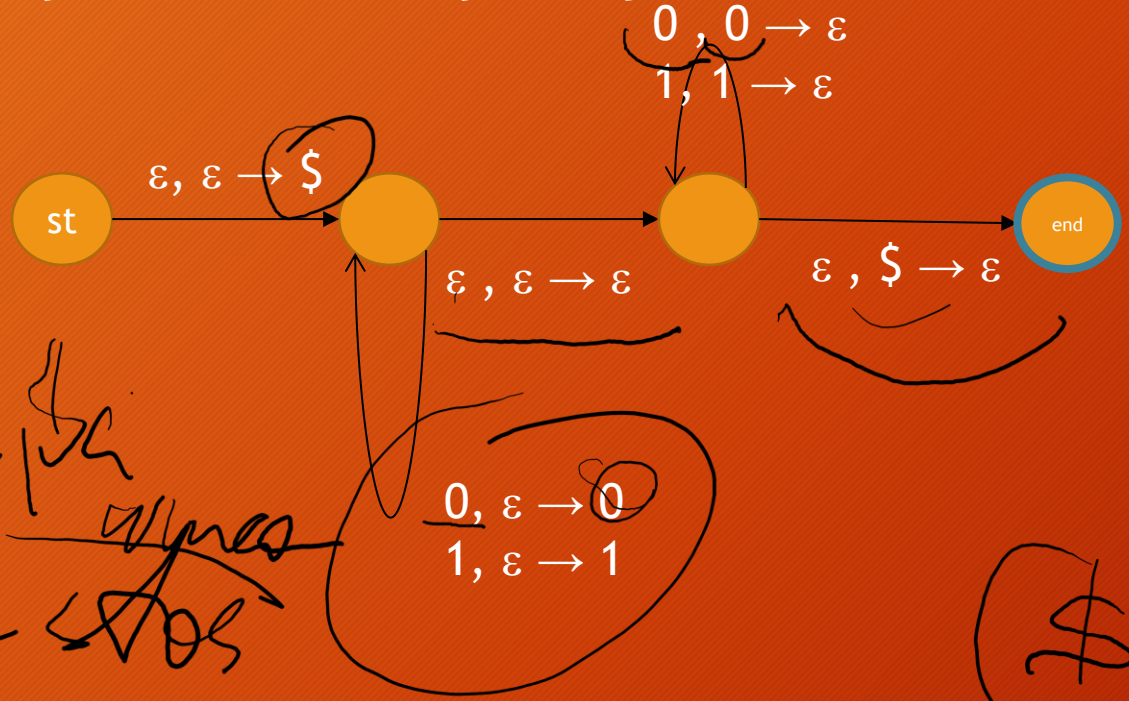
- W stanie s_1 jeśli przeczytasz z wejścia symbol „a” i podniesiesz ze stosu symbol „b” przejdź do stanu s_2 kładąc na stos symbol „c”
- ε – oznacza, że nie odczytujemy wejścia / nie pobieramy ze stosu lub nie kładziemy nic na stos





Automat ze stosem rozpoznający język złożony z palindromów o parzystej długości

- L={w w^R: w∈{0,1}*}, w^R – odwrócony łańcuch *w*
- Jak to ma działać – wrzucamy litery w na stos, a potem ściągamy je w odwrotnej kolejności



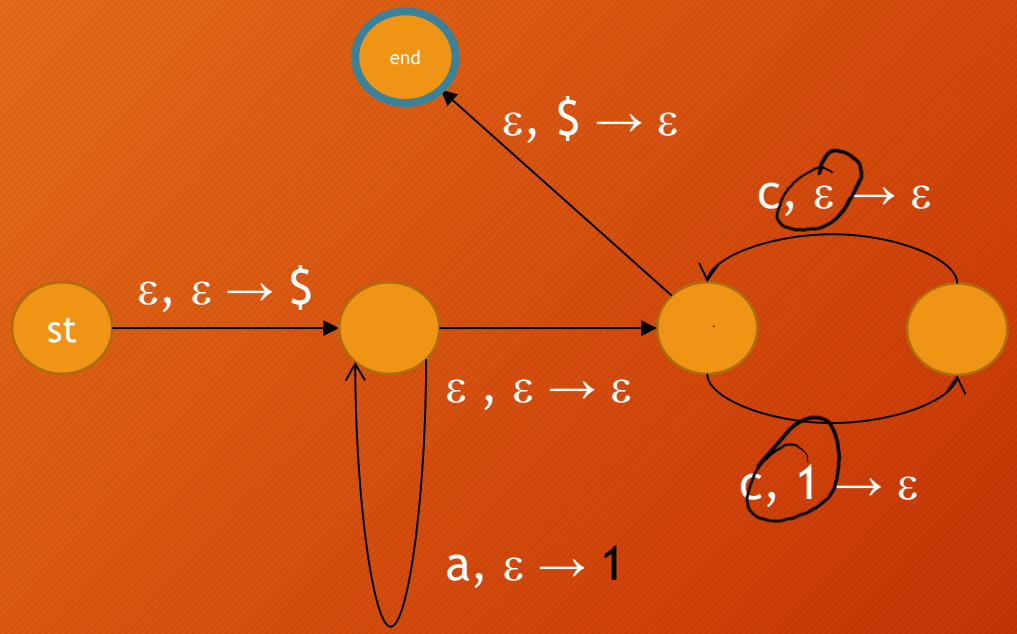
$ - specjalny symbol, znacznik końca

1010 0001

~~0~~
1
0
1

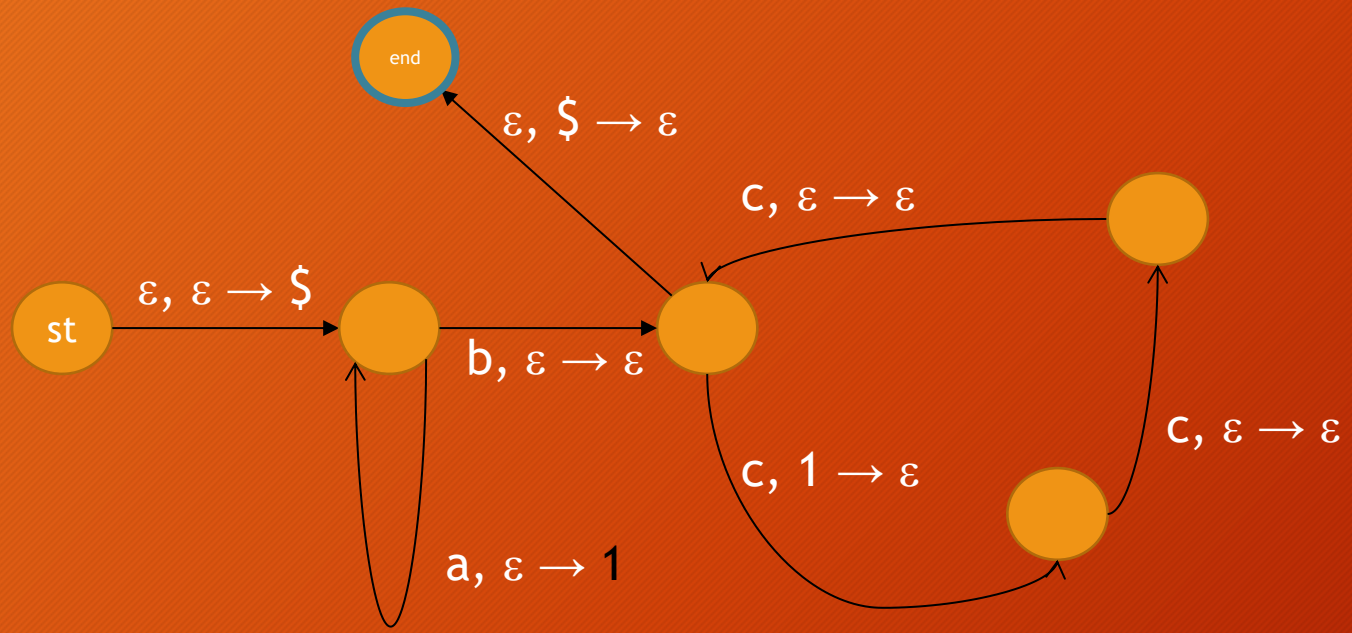
Automat ze stosem – dalszy przykład

- Automat ze stosem rozpoznający język $\{a^n c^{2n}\}$



Jaki język rozpoznaje automat ze stosem

- $a^n b\ c^{3n}$



Gramatyki bezkontekstowe a automaty ze stosem

Twierdzenie. Język L jest bezkontekstowy wtedy i tylko wtedy, gdy istnieje automat ze stosem rozpoznający ten język.

Oznacza to, że:

- Dla danej gramatyki bezkontekstowej G możemy znaleźć automat ze stosem rozpoznającą język L(G)
- Dla danego automatu ze stosem rozpoznającego język L’ możemy znaleźć odpowiadającą mu gramatykę G, taką że L’=L(G)

Wniosek dotyczący języków regularnych

Twierdzenie. Każdy język regularny jest językiem bezkontekstowym.

Uzasadnienie: każdy automat skończony jest przypadkiem szczególnym automatu ze stosem nie korzystającym ze stosu, tj. o funkcjach przejścia o wartościach postaci $x, \varepsilon \rightarrow \varepsilon$, gdzie x jest symbolem z alfabetu wejściowego

Ciekawe pytanie

- Czy język $\{a^n b^n c^n : n>0\}$ jest bezkontekstowy?
- Nie, problem jednego stosu!

Lemat o pompowaniu dla języków bezkontekstowych

Jeśli *L* jest językiem bezkontektowym, to istnieje taka liczba p (długość pomowania – ang. pumping length), że jeżeli $s \in L$ i $|s| > p$,

to można podzielić s na 5 części s = uvxyz spełniających warunki:

1. $uv^kxy^k z \in L$ dla każdego $k \geq 0$
2. $|vy| > 0$ */v i y nie mogą być puste* ε*/*
3. $|vxy| \leq p$

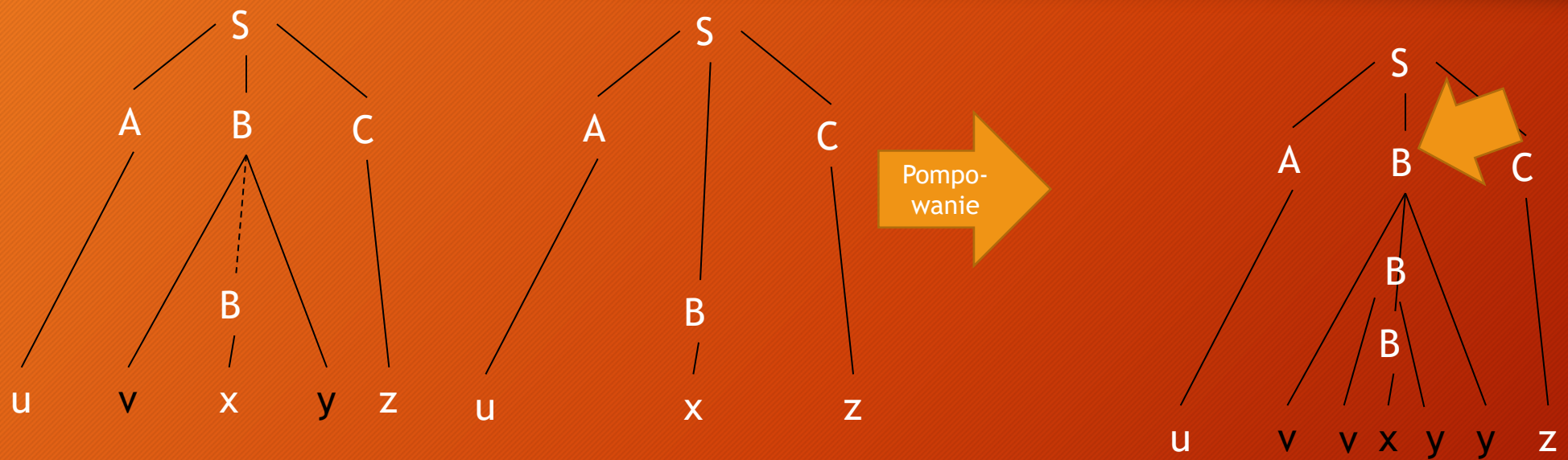
Pompowalność słów/wyrażeń jest warunkiem koniecznym, ale nie wystarczającym dla bycia językiem bezkontektowym

Przykład – ilustrujący lemat o pompowaniu dla CFL

- Język palindromów o nieparzystej długości

s	u	v	x	y	z
101	ε	1	0	1	ε
11011	1	1	0	1	1
011010110	011	0	1	0	110

Pompowanie CFL



Wykorzystanie lematu o pompowaniu dla CFL

- Dla języka $L=\{a^n b^n c^n : n \geq 0\}$ wykazać, że nie jest językiem bezkontekstowym
- Zadanie typu otwartego: załóżmy pewne p, jako długość pompowania, rozważmy łańcuch testowy $a^p b^p c^p$, które musimy podzielić na 5 części, w tym 2 „pompowalne”:
- Czyli $a^p b^p c^p = uvxyz$
  - Załóżmy, że v i y zawierają więcej niż jeden symbol, wtedy
    - $u\, v^2\, xy^2\, z \notin a^n b^n c^n$
  - Jeśli założymy, że *v*=a i *y*=b zawierają dokładnie jeden symbol
    - $u\, v^2\, xy^2\, z = a^{n+1} b^{n+1} c^n \notin a^n b^n c^n$

Uwaga!

- Ze spełnienia warunku pompowalności nie wynika, że język jest bezkontekstowy!
- Aby to sprawdzić trzeba, albo znaleźć gramatykę, albo automat ze stosem

Własność domkniętości języków bezkontekstowych

Analogicznie jak dla języków bezkontektowych

Jeśli A i B są językami regularnymi, to:

- $A \cup B$
- $A \circ B$ (konkatenacja)
- A^*

Też są językami kontekstowymi.

Gramatyki konktekstowe

(informacja)

Gramatyka kontekstowa

Gramatyka bezkontekstowa (ang. Context-Sensitive Grammar, CSG) G=(V, Σ, R, S), gdzie

- V (pierwotnie: N) – zbiór symboli nieterminalnych (zwanych niekiedy zmiennymi)
- Σ – zbiór symboli terminalnych (rozłączny z V),
- R – zbiór produkcji postaci $aUb \rightarrow x\ y\ z$, gdzie $x, y, z \in (V \cup \Sigma)^*$, z czego y musi być niepusty, a, b – symbole terminalne (w tym ε!), czyli aby dokonać podstawienia zmienna U musi być „w kontekście znaków *a* i *b*”, za
- S – zmienna (symbol) początkowy $S \in V$.

Przykład języka kontekstowego

$L=\{a^i b^i c^i : i>0\}$

1. S → aBC
2. S → aSBC
3. CB → CZ
4. CZ → WZ
5. WZ → WC
6. WC → BC
7. aB → ab
8. bB → bb
9. bC → bc
10. cC → cc

- Trudne przeparsowanie (nie każde jest udane, niektóre „zdychają”), generalnie działają
- S
- $\rightarrow_2$ **aSBC** $\rightarrow_2$ *aa**SBC**BC* $\rightarrow_1$ *aa**aBCBC**BC*
- $\rightarrow_3$ *aaaB**CZ**CBC* $\rightarrow_4$ *aaaB**WZ**CBC*
- $\rightarrow_5$ *aaaB**WC**CBC* $\rightarrow_6$ *aaaB**BC**CBC*
- $\rightarrow_3$ *aaaBBC**CZ**C*
- $\rightarrow_4$ *aaaBBC**WZ**C*
- $\rightarrow_5$ *aaaBBC**WC**C*
- $\rightarrow_6$ *aaaBBC**BC**C*
- $\rightarrow_3$ *aaaBB**CZ**CC*
- $\rightarrow_4$ *aaaBB**WZ**CC*
- $\rightarrow_5$ *aaaBB**WC**CC*
- $\rightarrow_6$ *aaaBB**BC**CC*
- $\rightarrow_7$ *aa**ab**BBCCC*
- $\rightarrow_8$ *aaa**bb**BCCC*
- $\rightarrow_8$ *aaab**bb**CCC*
- $\rightarrow_9$ *aaabb**bc**CC*
- $\rightarrow_{10}$ *aaabbb**cc**C*
- $\rightarrow_{10}$ *aaabbbc**cc***

Koniec

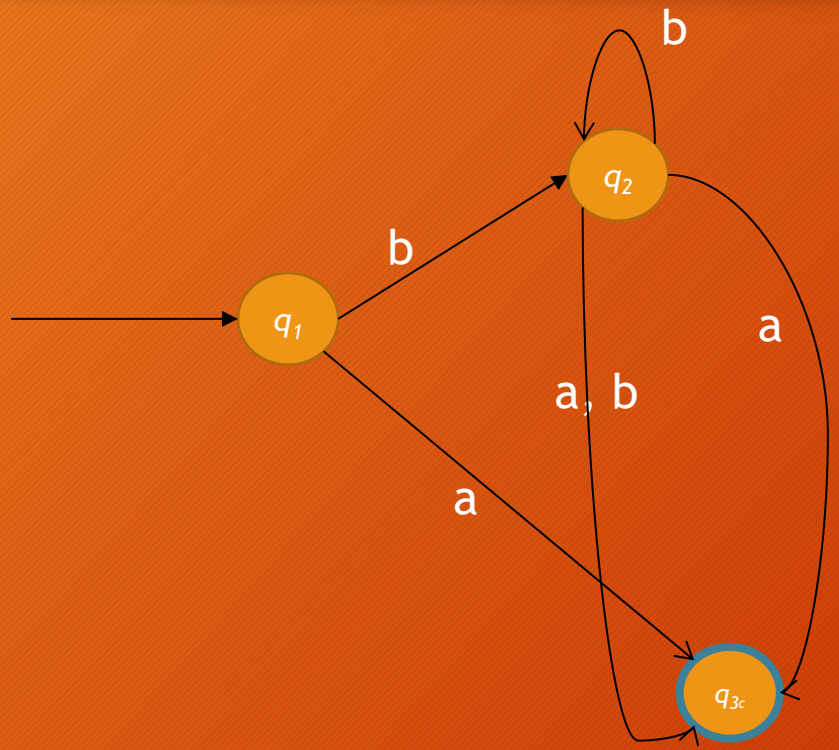
Algorytmy i struktury danych

Odc. 12. Maszyny Turinga. Nieobliczalność.

Kodowanie programu

- W komputerze używamy tylko 0 i 1. Można ponumerować stany jako 1, 11, 111, wykorzystując zera jako separatory komórek, dodać specjalne kody dla każdego z symboli A, R, a, b, oraz 00 jako separator #etc. Wtedy program maszyny można zapisać jako:
- #q1Raq3bq2 → 001010101110110 11
- #q3Aaq2bq2 → 0011101101011011011

Zakodowany automat determin



- #q1Raq3bq2#q2Rbq2aq3…
- A to możemy przekodować na zera i jedynki

- #q1Raq3bq2
- #=00 q1=1 R=01 a=10 b=11, 0 - sparator
- 00 1 0 01 0 10 0 111 0 10 0 11

Konkluzja (oczywista)

- Każdy z automatów, nawet logikę maszyny Turinga możemy zapisać w postaci ciągów zer i jedynek
- To jest dokładnie ta droga, która została pokonana w rozwoju komputerów – pierwszy i istotny problem – binarne kodowanie (zapis) danych
- Uwaga lingwistyczna
  - Kodowanie – nie mylić z szyfrowaniem, zakodowany to nie zaszyfrowany

Uniwersalna maszyna Turinga (UTM)

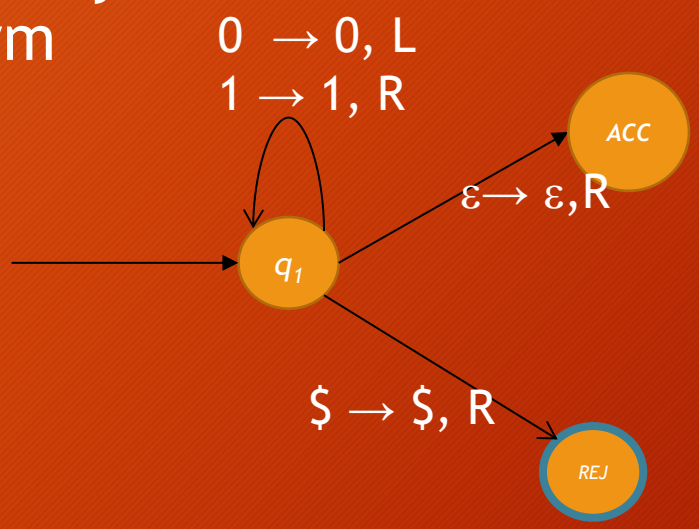
- Maszyna z 3 taśmami:

1. Jedna dla maszyny Turinga (zakodowany program), którą UTM symuluje (stała)
2. Jedna dla danych wejściowych (zmienna – zapis danych wyjściowych – tu jest zapisywane wyjście UTM i wynik obliczeń)
3. Jedna jako pamięć (zapamiętuje stan UTM)(zmienna – zapis stanu symulowanej maszyny)

Języki rozstrzygalne (ang. decidable)

- Maszyna Turinga nie zawsze kończy działanie w stanie akceptującym lub odrzucającym

- Przykład
  - 1111ε - akceptuje
  - $0011 – odrzuca
  - 1100 – krąży w nieskończoność!



Języki rozstrzygalne maszyną Turinga

Definicja. Maszyna Turinga, która niezależnie od tego, co znajduje się na taśmie zatrzymuje się – określana jest jako „maszyna rozstrzygająca” (ang. decider).

Język *L* jest rozstrzygalny w sensie Turinga (Turing-decidable), jeśli istnieje maszyna Turinga M, która rozstrzyga o przynależności do L.

Rozstrzyga, to znaczy, że:

- jeśli $w \in L$ => M kończy w stanie akceptującym
- Jeśli $w \notin L$ => M kończy w stanie odrzucającym

Język rozpoznawalny przez MT

- Maszyna M rozpoznaje *(nie rozstrzyga!!!)* język L, jeśli
  - Dla każdego słowa z *L* kończy w stanie akceptującym
  - Dla każdego słowa spoza L kończy w stanie odrzucającym lub ulega zapętleniu
- Inaczej jeśli rozstrzyga lub zapętla się → otwiera to pytanie, czy jeśli zaczekamy wystarczający długo, to maszyna się zatrzyma?
- Zbiór języków rozpoznawalnych $\supset$ zbiór języków rozstrzygalnych

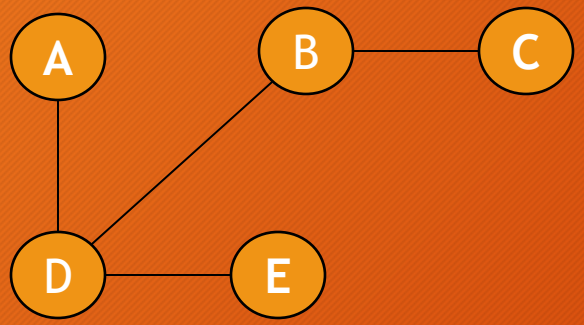
Przykłady (wysokopoziomowe) – nr 1

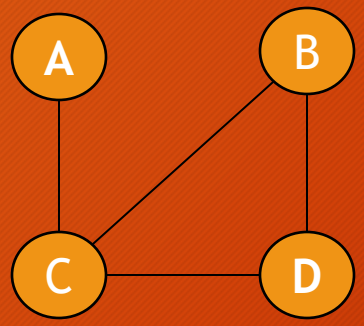
- Mamy dwa programy komputerowe. Każdy dostaje na wejściu jedną stronę tekstu (*N* znaków). Maszyna Turinga może dać każdemu z tych programów jakiekolwiek dane. Maszyna akceptuje, jeśli dla tych samych danych programy te dają ten sam wynik, inaczej odrzuca.
- Czy taka maszyna T jest maszyną rozstrzygającą?

Odpowiedź:

TAK. Maszyna może sprawdzić wszystkie możliwe wartość wejścia. Jeśli alfabet liczy sobie k znaków, musiałaby sprawdzić N^k różnych wejść (tzw. wariacja z powtórzeniami)

Przykłady wysokopoziomowe





- Drzewa możemy zapisać w postaci incydencji #A-D#D-ABE#B-DC#E-D (i jakkolwiek przekodować binarnie).
- Czy istnieje maszyna Touringa rozpoznająca język zawierający same drzewa?
  - Tak. Sprawdzamy czy z każdego węzła są połączenia z innymi węzłami, sprawdzamy cykle i w zależności odrzucamy lub akceptujemy

Rozstrzyganie cd.

- A_{DFA} = {<D, w> : D jest det. aut. skończ. akceptującym słowo *w*}
- Czy istnieje maszyna Turinga, która rozpoznaje taki język

Rozwiązanie: Tak. Symulowanie automatu deterministycznego przez maszynę Turinga jest możliwe (zapis automatu D i słowa mogą być w oddzielnych częściach taśmy lub na oddzielnych taśmach).

Symulacja automatu przez odczytanie wszystkich znaków tekstu i sprawdzenie stanu automatu – rozstrzyga czy dana para automat i słowo należy do języka lub do niego nie należy.

Rozstrzyganie

- Język L={<G,w>: G – gramatyka bezkontekstowa generująca w}
- Czy istnieje maszyna Turinga rozstrzygająca problem przynależności do L

Odp. TAK. Rozumowanie podobne, jak wcześniej.

Nierozstrzygalność (zaznaczenie)

- A_{TM}={<M,w>:M jest maszyną Turinga akceptującą słowo w}
- Czy A_{TM} jest rozstrzygalny, tj. czy mając daną maszynę Turinga, łańcuch w, uniwersalna maszyna Turinga może rozstrzygnąć w skończonym czasie czy *w* jest akceptowane przez M.
- Niestety nie wiadomo. Maszyny Turinga jak wiemy z przykładu mogą wpadać w nieskończone pętle

Zbiory przeliczalne i nieprzeliczalne

Definicja. Zbiór jest przeliczalny – zbiór równoliczny ze zbiorem liczb naturalnych

Równoliczny – taki, którego każdemu elementowi możemy przypisać inną liczbę naturalną

Przeliczalne są zbiory liczb całkowitych, iloczyn Z x Z

Zbiór liczb rzeczywistych jest nieprzeliczalny

- Argument diagonali Cantora
- *1 2 3 4 5…*
- a_1 3 6 5 4 1
- a_2 8 7 5 2 3
- a_3 6 1 1 5 2
- a_4 8 2 6 9 4
- …

- Dobierając ciąg różnych wartości od danych z diagonali konstruujemy coś czego nie ma w zbiorze. Zawsze możemy stworzyć coś czego nie wyliczyliśmy

Czy zbiór funkcji f: $Z^+ \rightarrow Z^+$ przeliczalny?

- Odp. NIE.
- | | f1 | f2 | f3 | f4 | … |
- | 1 | 19 | 834 | 11 | 1 | … |
- | 2 | 5 | 7 | 2 | 10 | … |
- | 3 | 127 | 9 | 3 | 33 | … |
- | 4 | 58 | 13 | 8 | 49 | |
- 5

czyli f(1)<>19, f(2)<>7…

Nierozstrzygalność

- Czy zbiór wszystkich języków nad alfabetem {0, 1} jest przeliczalny?
- Nie! Argument diagonalny Cantora: wiersze języki, kolumny słowa, na przecięciu 0 – należy do języka, 1 nie należy → na podstawie elementów na diagonali konstruujemy język nie należący do tego wykazu.
- A zatem musi istnieć język, który nie jest rozpoznawalny przez maszynę Turinga.

Uwaga końcowa

- Wiadomość zła: istnieją więc problemy, których nie da się mechanicznie rozwiązać
- Wiadomość dobra: bardzo wiele problemów da się jednak rozwiązać mechanicznie

KONIEC

Klasy problemów (wstęp)

Klasa	Czas rozwiązania	Czas sprawdzenia rozwiązania
P	Wielomianowe	Wielomianowe
NP (non-deterministic polynomial)	W praktyce: wykładniczy (przynajmniej na razie ;-)) – „brute force”. W teorii: wykładniczy wielomianowy algorytm niedeterministyczny	Wielomianowy
NP-Complete	Taki problem z NP, do którego można sprowadzić każdy problem klasy	Wielomianowy

Złożoność rozwiązania > złożoności sprawdzenia rozwiązania

Problem SAT (boolean satisfiability)

Dane jest wyrażenie logiczne w postaci CNF

- $\Psi = \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_n$
  - „$\wedge$” - koniunkcja
- *Np.* $\Psi_i = (x_1 \vee x_2 \vee x_3')$ – alternatywa maks. K zmiennych i/lub ich negacji ($'$)
- Czyli mamy $\Psi(x_1, x_2, \ldots, x_K) = R_1(X) \wedge R_2(X) \wedge \ldots \wedge R_n(X)$, gdzie $X=(x_1, \ldots, x_K)$

Zadanie: Znaleźć wektor wartości logicznych X, dla których wyrażenie Ψ *jest prawdziwe*

Najprostszy algorytm

- Wiadomość dobra: problem jest rozstrzygalny (zawsze można znaleźć rozwiązanie!), bo jest algorytm znajdujący rozwiązanie (lub stwierdzający, że wyrażenie jest sprzeczne):
- **Podstawienie w $X=(x_1,\ldots, x_K)$ wszystkich możliwych kombinacji wartości logicznych**
- Złożoność 2^K

Twierdzenie Cook’a-Levin’a

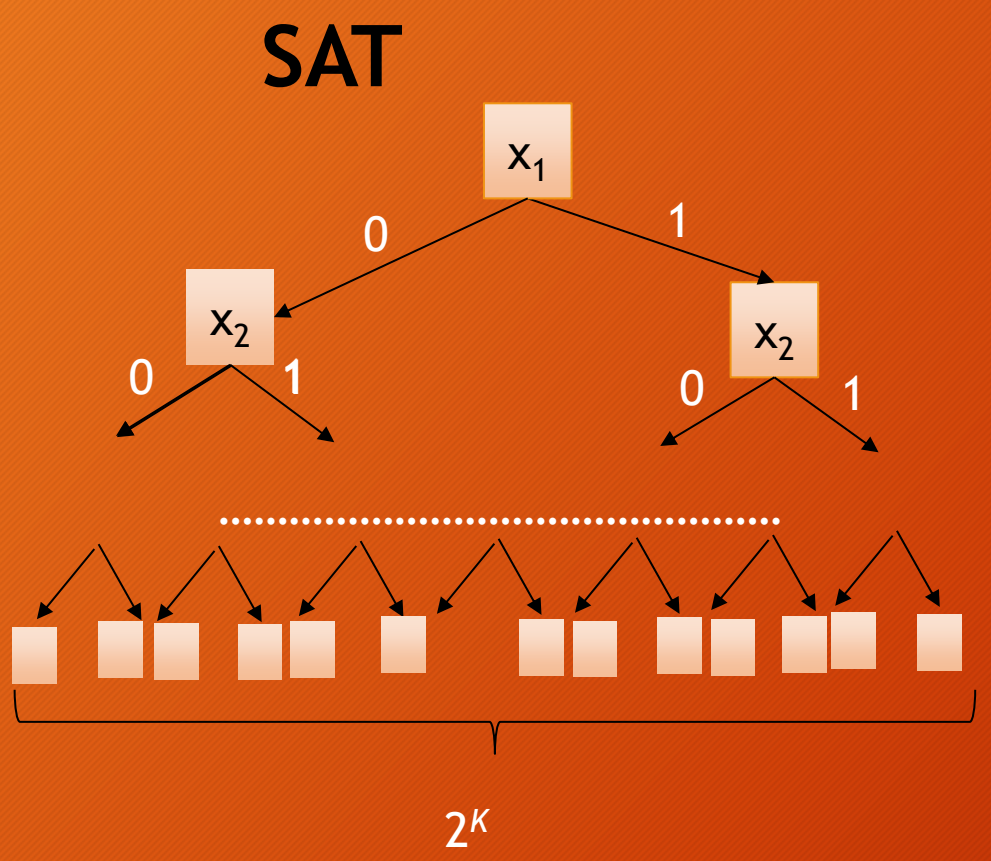
Problem SAT jest NP-zupełny.

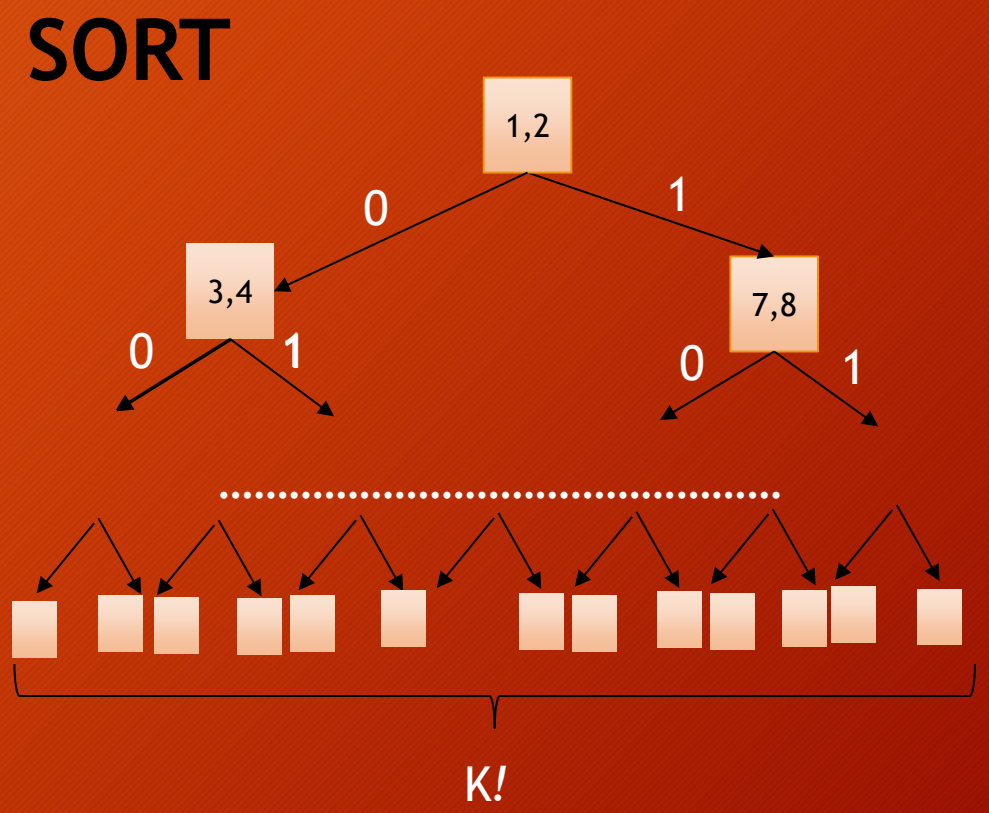
Czyli każdy problem klasy NP można sprowadzić do tego problemu!

Na czym polega trudność problemu SAT?

- Powiązania wyrażeń składowych (te same zmienne)
- Nie wiadomo, jak ukierunkować poszukiwanie rozwiązania – nie wynika to z samych zapisów badanego wyrażenia

Problem SAT a problem sortowania





Cykl i ścieżka Hamiltona

Definicja

Ścieżka Hamiltona – ścieżka przechodząca przez każdy z wierzchołków grafu dokładnie raz

Cykl Hamiltona – cykliczna ścieżka Hamiltona

Własność

- Należy do kategorii problemów „trudnych” – i do klasy problemów NP-zupełnych (ang. NP-complete).
- Nie jest znany algorytm wielomianowy, który znajduje (lub stwierdza nie istnienie) ścieżkę lub cykl Hamiltona dla dowolnego grafu

Algorytmy (1)

- Sprawdzenie wszystkich permutacji węzłów, jeśli dla każdej pary kolejnych węzłów istnieje łącząca krawędź, to mamy cykl Hamiltona
  - Czas: prop. ~N!

Algorytmy (2)

Zmodyfikowane przeszukiwanie w głąb

Zaczynamy od dowolnego wierzchołka v_0, który zapamiętujemy. Odpalamy: **DFS-HAM**(v)

DFS-HAM(v) – procedura rekurencyjna

1) Połóż-na-stosie(v)

2) Jeśli *h* (wysokość stosu) = liczba węzłów grafu i $v_0 \in$ następnik(*v*) to mamy cykl Hamiltona (na stosie), możemy zapisać *w przeciwnym razie*

3) Dla wszystkich *v'* $\in$ następnik(*v*) uruchom *DFS-HAM*(*v'*)

4) Zdejmij *v* ze stosu

SAT solver

1) Wyrażamy w postaci wyrażenia logicznego
2) Rozwiązujem problem spełnialności tej formuły (SAT) – specjalnym narzędziem

Konwersja HAM do problemu SAT

Klucz do sukcesu(!): p_{ij} – prawda, jeśli i jest j-tym węzłem na ścieżce Hamiltona, czyli

ścieżka Hamiltona składa się z tych węzłów *i*, dla których p_{ij} jest prawdą, w kolejności wyznaczonej przez *j*

Struktura grafu dana jest funkcją *N*(*i*) – zbiór następników

Konwersja HAM -> SAT

$$H(n,N) = \mathcal{P}(n) \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n-1} \left(\bigvee_{v \in N(i)} p_{v,j+1} \right)$$

- Szukamy takich p_{ij}, że H(n, N) jest prawdziwe
  - *P*(*n*) – warunek, że p_{ij} obejmujące wszystkie węzły, ale każdy tylko raz
  - Druga część koniunkcji oznacza, że ścieżka przebiega krawędziami grafu (nie jest tylko ciągiem n-węzłów grafu)
- Jeśli ścieżka złożona z krawędzi, dla których (p_{ij} , $p_{i,j+1}$)=prawda (problem SAT)

Warunek prawidłowego zapisu w „systemie” p_{ij}

$$\mathcal{P}(n) = \bigwedge_{j=1}^{n} \left(\bigvee_{i=1}^{n} p_{i,j} \wedge \bigwedge_{k \in \{1..n\} \setminus \{i\}} \neg p_{k,j} \right) \wedge \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{n} p_{i,j} \wedge \bigwedge_{k \in \{1..n\} \setminus \{j\}} \neg p_{i,k} \right)$$

- *Przypomnienie*: ***i*** – pozycja na ścieżce, ***j*** nr węzła na *i*-tej pozycji
- Pierwszy człon głównej koniunkcji…
- Drugi człon głównej koniunkcji…

1 2 3 4 – pozycja na ścieżce H.
1 0 0 0 1
2 0 1 0 0
3 0 0 1 0
4 1 0 0 0

Teraz...

- SAT(*H*(n, N))

Podobieństwo – tzw. programowanie binarne

Dane:

- $F(X)$: $\{0, 1\}^N$ -> R, gdzie X – wektor N zer i jedynek
- $G(X) = 0$ – ograniczenia np. $AX + B = 0$ (A macierz N x N)

Zadanie:

- Znaleźć takie X, że $F(X)$ jest minimalne i spełnione są ograniczenia $G(X) = 0$

Inne problemy NP-zupełne

Problem plecakowy - sformułowanie

Dany jest:

- Pojemność (plecaka) – C_K.
- Zbiór n przedmiotów (o nr 1… n) o objętości c_i i wartości v_i

Zadanie:

- Wybrać takie przedmioty, które zmieszczą się w plecaku i których łączna wartość (suma) będzie maksymalna

Problem plecakowy - rozwiązanie

Niech x_i = 1 (prawda), gdy *i*-ty przedmiot jest w plecaku i 0 (fałsz) – zmienna decyzyjna, $X=(x_1,\ldots, x_n)$

- W wersji „przedmiotów” sypkich – działa strategia zachłanna – wsypujemy kruszywa od najdroższego do najtańszego – im drożej wypełnimy każdą jednostkę objętości tym lepiej.
- W binarnej wersji problemu musimy przejrzeć wszystkie wartości 2^n wartości wektora X. Co do zasady ograniczenia na pojemność plecaka można zapisać w postaci wyrażeń logicznych… wi

Programowanie binarne

Maksymalizuj $\Sigma_{i=1\ldots n}\ x_i * v_i$

- Przy ograniczeniu na pojemność plecaka: $x_i * c_i <= C_k$

N-hetmanów

- Jak ustawić *n* hetmanów na planszy o wymiarach $n \times n$ tak, by żaden nie był zagrożony biciem
- p_{ij} – prawda, tzn. hetman na polu szachownicy o współrz. (i, j)

Ustawienie hetmanów spełniające wyrażenie P(n) $\wedge$ D(n) jest rozwiązaniem problemu. *P*(*n*) jak w problemie ścieżki Hamiltona.

$$\mathcal{D}(n) = \bigwedge_{k=1}^{n} \bigwedge_{l=1}^{n} \Bigg(\bigwedge_{1 \le i \ne k < n} \; \bigwedge_{1 \le j \ne l \le n-i} \big((\neg p_{j,i+j} \vee \neg p_{l,k+l}) \wedge (\neg p_{i+j,j} \vee \neg p_{k+l,l}) \wedge$$

$$(\neg p_{j,n-1-(i+j)} \vee \neg p_{l,n-1-(k+l)}) \wedge (\neg p_{i+j,n-1-j} \vee \neg p_{k+l,n-1-l}) \big) \Bigg)$$

Kolorowanie grafu

Dany jest graf G=(V, E) i stała całkowita k.

Czy istnieje taka funkcja *c*: *V* -> {1,…, *k*}, że

Dla każdego $v \in V$, dla każdego $u \in N(v)$ $c(v) <> c(u)$

Jeśli przypiszemy kolory wierzchołkom to, koniec każdej krawędzi powinien być tego samego koloru.

Problem kliki

Dany jest graf nieskierowanych G=(V, E)

Def. $S \subseteq V$ nazywamy kliką, gdy każdy wierzchołek z S jest połączony z innym wierzchołkiem z S (każdy w klice jest w relacji z każdym).

Problem: Czy graf *G* zawiera klikę *S* o wielkości przynajmniej k, czyli taką, że $|S| \geq k$.

Algorytm wykładniczy dla problemu SAT, k-zmiennych

- Dla wszystkich możliwych kombinacji
  - X=(prawda, …, fałsz)

- Podstaw X do wyrażenia w CNF i sprawdź czy CNF(X)=prawda, wtedy STOP, w przeciwnym razie kontunuuj przeszukiwanie

Algorytmy deterministyczne w zmaganiach z problemami NP-zupełnymi

- W podobny sposób można konstruować algorytmy deterministyczne dla każdego z problemów omówionych do tej pory i opisanych jako NP-zupełne (nie mylić z NP!!!)
- I faktycznie mają one niewielomianową złożoność obliczeniową

Obserwacja P i NP

- Z przykładów widać, że w NP są problemy trudniejsze, niż te, które znamy z klasy P, tj. te, które umiemy rozwiązać tylko brutalną siłą przeglądania wszystkich wariantów rozwiązań!!!

$P \subseteq NP$, ale czy przypadkiem nie P = NP

- $P \subseteq NP$ – algorytmy detetrministyczne będą szczególnym przypadkiem niedeterministycznych, więc jeśli mamy „normalny” deterministyczny algorytm rozwiązujący problem w czasie wielomianowym, to mieści się on też w klasie NP.
- Jest jedno, ale: nikt nie wykazał, że dla tych problemów z NP, dla których nie znamy deterministycznego algorytmu wielomianowego, taki algorytm nie istnieje.
- Tego nie wiemy! I stąd słynny problem,
  - czy P=NP!!!

Problemy NP-zupełne (NPC) – definicja

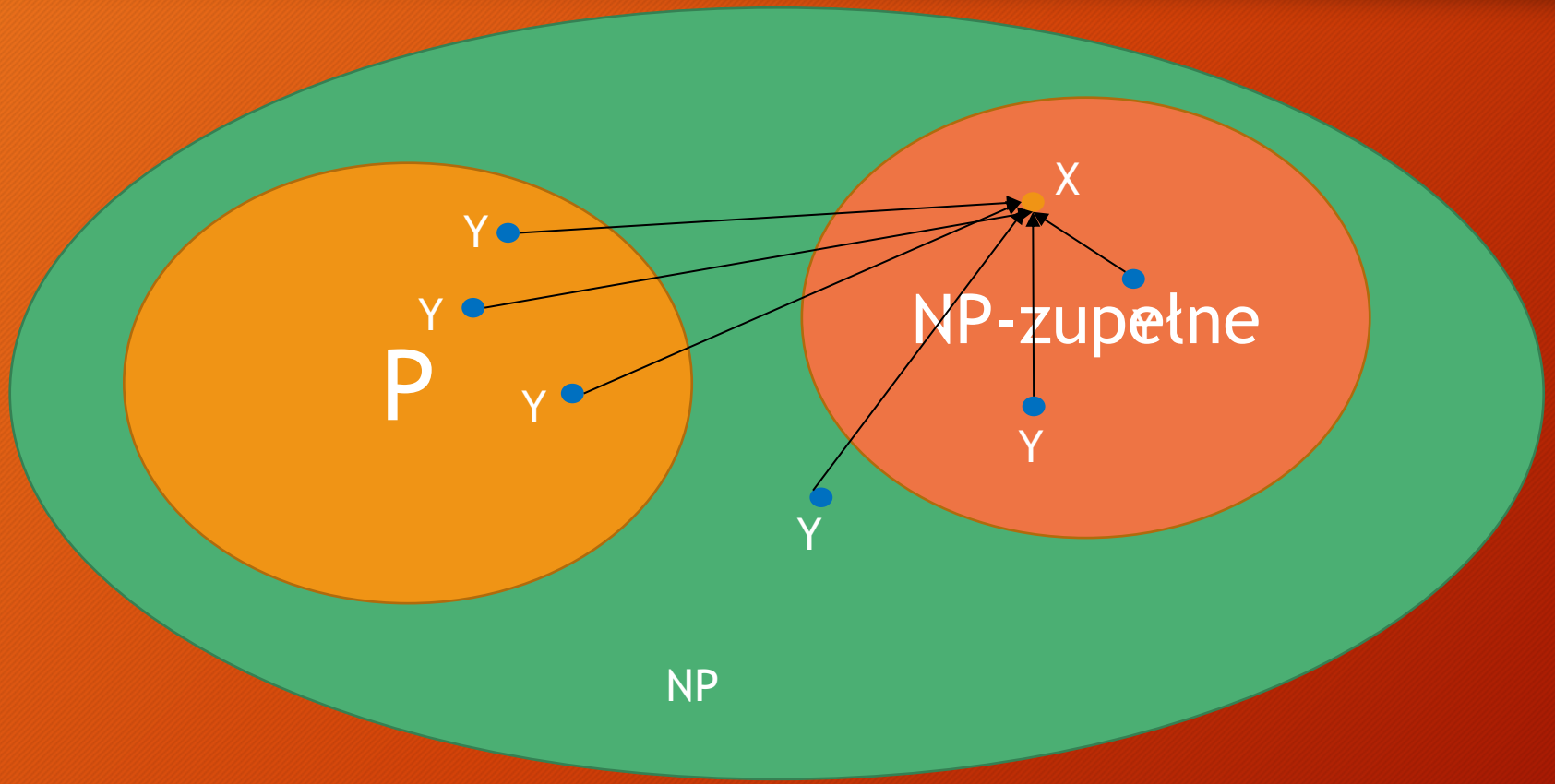
Def. Problem X należy do problemów NP-zupełnych, jeśli:

Warunek 1. $X \in NPC$.

Warunek 2. Każdy problem Y z NP można (sprowadzić) zredukować w czasie wielomianowym do X

Uwaga – problem łatwiejszy redukujemy do trudniejszego

Ilustracja warunku 2 NP-zupełności



NP-zupełność – interpretacja, wyjaśnienia

- Definicja NPC postuluje, że są problemy „reprezentatywne” czy też o „sile wyrazu” pozwalającej wyrazić wszystkie inne problemy rozwiązywalne niedeterministycznym algorytmem w czasie wielomianowym
- NPC (NP-zupełne) to te problemy, w języku których / w postaci których można wyrazić wszelkie (!!!) inne problemy klasy NP, **w tym problemy klasy P!!!!**

NPC - ilustracje

- Wiemy (choć tego nie pokazaliśmy!), że do NPC należą problemy opisane już na poprzednim wykładzie – kolorawnie grafu, cykl Hamiltona, klika, SAT
- Oznacza to, że w postaci problemu cyklu Hamiltona czy SAT można wyrazić problem sortowania, najkrótszej ścieżki (i każdy inny z P i z NP)

Sortowanie i problem cyklu Hamiltona

- Zauważmy, że problem sortowania polega na znalezieniu takie ciągu indeksów, że wartości w tablicy / liście pod tymi indeksami są uporządkowane rosnąco
- (idx_1,…, idx_n), że (T[idx_1], T[idx_2], …, T[idx_n]) są uprządkowane

- Na tym samym polega problem cyklu Hamiltona… nawet sprawdzenie jest takie samo (sprawdzenie czy pary wierzchołków są elementami relacji sąsiedztwa)
- Co nie znaczy, że to jest dobry sposób na sortowanie ;-) , ale na pewno problem sortowania można sprowadzić do problemu Hamiltona, więc (jak pamiętamy i do SAT)

Koniec