Lingwistyka zadanka

Nie przestraszcie się że jest dużo stron, zadań jest mało, ale niektóre postanowiłem mocno rozpisać, w miarę krok po kroku i dla przejrzystości nie „nadziubdziać" tylko dać im trochę przestrzeni!

Aby ułatwić przyswojenie wiedzy z zakresu tego przedmiotu, postanowiłem zmontować mały skrypcik, który wspólnymi siłami panuję dokończyć jeszcze przed egzaminem. W zamyśle ma on zawierać rozwiązania zadań na cotygodniowe sprawdzianiki, ale ze względu na to że czasami trudne jest jednoznaczne określenie właściwej odpowiedzi, postaram się wykorzystać to przy próbie uczenia metodą „zderzeniową". Materiał jest udostępniony już teraz między innymi po to, że jeżeli gdzieś sam się pomyliłem i ktoś udowodni mi błąd, wstawię w to miejsce poprawioną wersję jak i ostrzeżenie na co należy uważać.

Dodatkowo na koniec każdej partii zadań umieszczę krótki „test". Test jest oczywiście dobrowolny, ale nie będzie polegał na wykonaniu jeszcze kilku zadań, ale spędzeniu około 5/10 minut na próbę zastanowienia się nad pytaniami lub zadankami tam umieszczonymi (sztandarowy przykład będzie w pierwszym rozdziale!). Proces ten jest jedną z metod szybkiego uczenia, którą chciałem w tym semestrze przetestować na sobie, a ze względu na egzamin teoria i zrozumienie tego przedmiotu może być dobrym polem do manewru.

Oczywiście umieszczam ostrzeżenie i wyzbywam się odpowiedzialności za niezdane kolokwium, jeżeli pojawią się jakieś błędy, których nikt mi nie zasygnalizuje, dlatego nie bierzcie wszystkiego co się tutaj znajduje bezkrytycznie i jak zawsze liczę na kontakt. Dobra, bo już się za dużo rozpisałem, a czas goni!

Spis treści

1 Ćwiczenia I

1.1 Rozgrzewka

Poznajemy semisystem Thue'ego, operację domknięcia (gwiazdki) Kleene'ego, pojęcie wyprowadzalności ale w zadaniach zajmiemy się głównie gramatykami kombinatorycznymi (złożonymi ze znanej nam czwórki) oraz generowanymi przez nie językami.
Zanim przejdziemy do zadań, spróbuj odpowiedzieć na pytanie z czym właściwie kojarzą Ci się pojęcia:

- Alfabet,
- Gramatyka,
- Zbiór,
- Język,
- Termin.

Powiedz na głos jedno zdanie związane z każdymi z tych słów, niekoniecznie ich definicje.

Co wiąże ze sobą te słowa? Może są od siebie niezależne?

Jesteś w stanie wykonać diagram, który je sensownie powiąże? Spróbuj!

1.2 Zadania z ćwiczeń

1.2.1 Wyznacz V^2 i V^3 dla podanych alfabetów

a) $V = \{0, 1\}$
Warto pamiętać o tym, że $V^0 = \{\epsilon\}$, gdzie ϵ - element neutralny operacji konkatenacji,
I działamy zgodnie ze wzorem: $V^{i+1} = \{wv : w \in V^i v \in V\}$ dla $i \in N_0$
czyli:

$$V^1 = \{wv : w \in V^0 v \in V\} = \{\epsilon 0, \epsilon 1\} = \{0, 1\}$$
$$V^2 = \{wv : w \in V^1 v \in V\} = \{00, 10, 01, 11\}$$
$$V^3 = \{wv : w \in V^2 v \in V\} = \{000, 001, 010, 011, 100, 101, 110, 111\}$$

b) $V = \{@, \#, \$\}$

Znaki troszkę inne, ale zasada ta sama ;)

$$V^1 = \{wv : w \in V^0 v \in V\} = \{\epsilon @, \epsilon \#, \epsilon \$\} = \{@, \#, \$\}$$
$$V^2 = \{wv : w \in V^1 v \in V\} = \{@@, @\#, @\$, \#@, \#\#, \#\$, \$@, \$\#, \$\$\}$$

$$V^3 = \left\{wv : w \in V^2 v \in V\right\} =$$

$\{@@@, @@\#, @@\$, @\#@, @\#\#, @\#\$, @\$@, @\$\#, @\$\$\#@@, \#@\#, \#@\$, \#\#@, \#\#\#, \#\#\$, \#\$@, \#\$\#, \#\$\$ \ldots\}$

c) $V = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$$V^1 = \left\{wv : w \in V^0 v \in V\right\} = \{\epsilon 1, \epsilon 2, \epsilon 3 \ldots\} = \{1, 2, 3 \ldots\}$$

I działamy analogicznie...

1.2.2 Ile elementów liczy zbiór V^n jeśli V jest k-elementowy

W każdym kroku do każdego elementu ze zbioru V^{i-1} dołączamy k elementów ze zbioru V, stąd

$$card(V^n) = k^n$$

.

1.2.3 Określ V^* dla zbioru $V = \{0, 1\}$

Przypomnijmy definicję: $V^* = \bigcup_{i=0}^{\infty} V^i$

A więc chodzi o wzięcie sumy mnogościowej wszystkich zbiorów, w ten sposób generujących domknięcia Kleene'ego.

$$V^* = \bigcup_{i=0}^{\infty} V^i = V^0 \cup V^1 \cup V^2 \cup \ldots == \{\epsilon\} \cup \{0, 1\} \cup \{00, 01, 10, 11\} \cup \ldots$$

Rozwiązaniem będą zatem zapisy wszystkich liczb systemu dwójkowego i element neutralny.

1.2.4 Mamy grę o następujących zasadach...

Musimy oznaczyć odpowiedni alfabet, z niego wybrać aksjomat systemu (wyróżnione niepuste słowo) i określić listę produkcji, a zatem:

$$T = (V, P, x)$$

Teraz czym będzie nasz alfabet? Oczywiście alfabet w systemie semisystemie Thuego składa się z symbolów (terminalnych i nieterminalnych, w gramatykach kombinatorycznych rozdzielimy to na dwa zbiory). W tym przypadku jedynymi symbolami są kule, białe i czarne, a więc:

$$V = \{c, b\}$$

Aksjomatem była początkowa zawartość urny (no to potrzebujemy jeszcze trochę z treści...), a więc: $x =$ początkowa zawartość urny.

Pozostaje lista produkcji... Skoro po wyjęciu dwóch kul jednego koloru wkładamy do środka czarną, no to mamy już dwie produkcje, no a różnego koloru wstawia białą, wystarczy tylko zerknąć!

$$P = \{(bb, c), (cc, c), (bc, b), (cb, b)\}$$

Ale czy kolejność wyciągania ma znaczenie? Wydaje mi się że nie, przynajmniej dla prostych przypadków działa.

1.2.5 Dana jest gramatyka kombinatoryczna ...

$$V = \{a, b, c, d\}$$

,

$$\Sigma = \{\sigma, A, B\}$$

Lista produkcji:

$$\sigma \to AaB, AB \to c, A \to AB\sigma B, B \to bb, B\sigma \to Ba$$

Przykładowe wyprowadzenie:

$$\sigma \Rightarrow AaB \Rightarrow AB\sigma BaB \Rightarrow ABaBaB \Rightarrow caBaB \Rightarrow cabbaB \Rightarrow cabbabb$$

1.2.6 Do jakiej klasy należy gramatyka $G = (V, T, P, S)$

a) $V = \{K, L\}, T = \{a, b, c\}, S = L, P = \{K \to KL, aK \to abK, L \to abc, cK \to cabc, bK \to bc, L \to abcKabc\}$

Ta gramatyka należy do klasy pierwszej czyli jest to gramatyka kontekstowa. Nie jest bezkontekstowa dlatego, że istnieje produkcja z więcej niż jednym symbolem po lewej stronie strzałki. Nie może zatem być regularna. Jest kontekstowa natomiast dlatego, że po lewej stronie strzałki zawsze mamy pojedynczy symbol nieterminalny, a jeżeli ma jakiś prefix, to po drugiej stronie strzałki też znajdziemy ten sam prefix (np. w produkcji $aK \to abK$ tym prefixem będzie a). Czyli po lewej stronie strzałki mamy jeden maksymalnie jeden symbol nieterminalny, a jeżeli ma on prefix lub sufix, to taki sam prefix lub sufix jesteśmy w stanie po drugiej stronie strzałki.

b) $V = \{M, Q, R\}$, $T = \{a, b\}$, $S = M$, $P = \{M \to aM, M \to bQ, Q \to aR, R \to bR, R \to b\}$

Ta gramatyka należy to klasy trzeciej, jest regularna prawostronnie, ponieważ każda z gramatyk jest postaci $A \to qA|q$, czyli symbol nieterminalny jest zamieniany na symbol terminalny sklejony z symbolem nieterminalnym, tak że terminalny jest z lewej strony lub po prostu zamieniany na symbol terminalny.

1.2.7 Gramatyka dla języka palindromów nad alfabetem $\{0, 1\}$

$$G = (V, \Sigma, P, S)$$

:

$$V = \{0, 1\}$$

$$\Sigma = \{M\}$$

$$S = M$$

$$P = \{M \to 1, M \to 0, M \to 11, M \to 00, M \to 1M1, M \to 0M0\}$$

Czyli jak coś generujemy to albo jedynkę albo zero albo dwie jedynki albo dwa zera albo coś, co będzie miało po obu stronach jedynki albo coś co będzie miało po obu stronach zera. Np. wyprowadźmy sobie 01010.

$$M \Rightarrow 0M0 \Rightarrow 01M10 \Rightarrow 01010$$

1.2.8 Podaj języki generowane przez gramatyki jej typ w hierarchi Chomsky'ego

a) $S \to aA$ i $A \to baA|aA|ba|a$

Po rozpisaniu sobie kilku przykładowych słów zauważamy, że możemy tworząc zdanie, w miejsce A możemy wstawić symbole ba lub a w dowolny sposób, a zatem językiem będą ciągi słów z elementami składowymi ba, a w dowolnym miejscu w dowolnej ilości, ale zaczynające się od litery a (a a później ile chcecie i w jakiejkolwiek konfiguracji ba lub a).

Patrząc na produkcje, zauważamy że mamy do czynienia z typem regularnym prawostronnie, ponieważ przed strzałką są pojedyncze symbole nieterminalne, a z prawej albo same symbole terminalne albo zbitek

symbole terminalne skonkatenowane z symbolem nieterminalnym.

b) $S \to aS|bS|aA$, $A \to aB|bB$ i $B \to \epsilon$

Zauważmy, że właściwie to ϵ to dużo nam nie zmienia... Tylko tyle że gramatyki tej nie możemy nazwać czystej pod względem ϵ produkcji.

Gramatyka będzie generować zbiór symboli a oraz b w dowolnej konfiguracji, zaczynając na oraz kończąc na dowolnym z nich, ale symbolem przedostatnim musi być a. Zwięźle: „Język nad alfabetem $\{a, b\}$ będący zbiorem wszystkich łańcuchów, w których przedostatnim symbolem jest litera a."

Należy do typu trzeciego, czyli jest gramatyką regularną prawostronnie, z tego samego powodu co w poprzednim podpunkcie.

c) $E \to E + E|E * E|(E)|2$

Gramatyka ta pozwala generować dowolne wyrażenia arytmetyczne związane z poprawnym dodawaniem oraz mnożeniem dwójek, również wykorzystując nawiasy. Jest to gramatyka typu drugiego, a więc bezkontekstowa, dlatego że posiada produkcje, w których symbole terminalne pojawiają się z obu stron nieterminalnego, ale każda produkcja ma przed strzałką tylko symbol nieterminalny.

Nie jest ona jednoznaczna, a to dlatego, że posiada przynajmniej jedno zdanie niejednoznaczne (np. $2 + 2 * 2$), możemy wskazać na zdanie, które posiada dwa możliwe drzewa syntaktyczne (drzewa wywodu), innymi słowy dwa różne wyprowadzenia danego słowa. Np:

$$E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow 2 * E + E \Rightarrow 2 * 2 + E \Rightarrow 2 + 2 * 2$$

$$E \Rightarrow E * E \Rightarrow E * E + E \Rightarrow \dots$$

d) $S \to Sa|Sb|\epsilon$

Gramatyka generuje dowolne ciągi symboli a lub b wraz z symbolem neutralnym. Jest to gramatyka typu trzeciego, regularna lewostronnie i z tak zwaną ϵ produkcją.

e) $S \to SS|(S)|\epsilon$

Mamy do czynienia z gramatyką generującą poprawne nawiasowanie (bez zawartości, ale to zawsze coś!). Jest to gramatyka bezkontekstowa, ponieważ posiada produkcje, która produkuje symbol nieterminalny z dwóch stron otoczony terminalnymi, ale wszystkie produkcje po lewej stronie strzałki mają pojedynczy symbol nieterminalny. Jest też gramatyką posiadającą ϵ produkcję.

1.2.9 Napisz gramatyki dla następujący języków

a) $a^n b^n, n \geqslant 1$

$$G = (V, \Sigma, P, S)$$

Alfabet terminalny jest raczej oczywisty:

$$V = \{a, b\}$$

Do alfabetu nieterminalnego wystarczy mi jeden symbol nieterminalny, oczywiście również startowy:

$$\Sigma = \{X\}$$

$$S = X$$

Produkcja również będzie dość intuicyjna... Albo z X tworzymy symbol ab albo coś co „wciśnie" się między a i b

$$P = \{X \to ab, X \to aXb\}$$

b) $a^n b^n c^k d^k, n > 1, k \geqslant 1$

$$G = (V, \Sigma, P, S)$$

Alfabet terminalny znów oczywisty:

$$V = \{a, b, c, d\}$$

Do alfabetu nieterminalnego teraz wybiorę trzy symbole, jeden z nich będzie służył do wystartowania całości, dwa pozostałe będą działały dość analogicznie do poprzedniego przypadku:

$$\Sigma = \{X, Y, Z\}$$

$$S = Z$$

Z racji, że literek $a^n b^n$ muszą być przynajmniej dwie pary (ponieważ $k > 1$) a literek $c^k d^k$ przynajmniej jedna para (ponieważ $k \geqslant 1$), to startując z symbolu startowego, od razu wrzucimy sobie przynajmniej jedną parę ab i możliwość produkcji innych par.

$$P = \{Z \to aXbY, X \to ab, X \to aXb, Y \to cd, Y \to cYd\}$$

c) $a^n b^n c^n, n \geqslant 1$

$$G = (V, \Sigma, P, S)$$

$$V = \{a, b, c\}$$

Alfabet nieterminalny znów będzie potrzebował aż trzech elementów, z czego jeden będzie potrzebny jako startowy, natomiast dwa pozostałe będą służyły do „przechodzenia"

$$\Sigma = \{X, Y, Z\}$$

$$S = Z$$

Dopóki mieliśmy dwa symbole, wszystko szło prosto, po prostu ładowaliśmy pomiędzy dwóch przedstawicieli parę nowych współlokatorów i wszystko się zgadzało. W tym przykładzie, jesteśmy jednak zmuszeni na użycie pewnej pośredniej rekursji. Kolejne produkcje w naszym ciągu „przechodzącym" będą doklejały za siebie odpowiedni symbol terminalny, tak aby zachować warunki języka. W ten sposób przejdą w prawo do samego końca, a później zawrócą na to samo miejsce (po przykładzie wszystko stanie się jasne!).

$$P = \{Z \to abc, Z \to aXbc, Xb \to bX, Xc \to Ybcc, bY \to Yb, aY \to aa, aY \to aaX\}$$

I mały przykład:

$$Z \Rightarrow aXbc \Rightarrow abXc \Rightarrow abYbcc \Rightarrow aYbbcc \Rightarrow aabbcc$$

1.2.10 Gramatyka dla poprawnego nawiasowania

Jedną z odpowiedzi znajdziecie w zadaniu 8 podpunkt e, ja natomiast tutaj pokażę inny fajny pomysł!

$$G = (V, \Sigma, P, S)$$

$$V = \emptyset$$

$$\Sigma = \{Z\}$$

$$S = Z$$

$$P = \{Z \to (Z)Z, Z \to \epsilon\}$$

1.3 Zadania dodatkowe z moodle'a

Czyli powtórzenie kilku bardziej sztandarowych z poprzedniego rozdziału.

1.3.1 Do jakiej klasy Chomsky'ego należy gramatyka $G = (V, \Sigma, P, \sigma)$, jeśli

a) $V = \{a, b, c, d\}$, $\Sigma = \{W\}$, $P = \{W \to abWc, bW \to cbW, Wc \to c\}$, $\sigma = W$

Zauważmy, że gramatyka ta posiada produkcję $bW \to cbW$, stąd wnioskuję że jest ona typu zerowego, czyli gramatyką struktur frazowych. (nie jest bezkontekstowa ponieważ po lewej stronie strzałki nie występują wyłącznie pojedyncze symbole nieterminalne ale nie jest też kontekstowa, ponieważ w wymienionej przeze mnie produkcji, przed symbol terminalny zostaje dopisany inny symbol terminalny, a więc jest to odmienne od postaci, którą potrzebują gramatyki kontekstowe).

A myśl drugiej definicji gramatyki kontekstowej (czyli że to przed strzałką nie może być dłuższe od tego za strzałką), bruździ nam produkcja $Wc \to c$, a więc tą drogą wnioskujemy że jest będzie do gramatyka klasy zerowej, czyli struktur frazowych (zwana też kombinatoryczną, tak żeby troszkę zamieszać :d).

b) $V = \{a, b\}$, $\Sigma = \{M, Q, R\}$, $P = \{M \to aM, M \to bQ, Q \to aR, R \to bR, R \to b\}$, $\sigma = M$

Mamy do czynienia z gramatyką regularną prawostronnie (prawostronna bo symbol nieterminalny z prawej strony się wyłania) (czyli klasy trzeciej), ponieważ każda produkcja po lewej stronie strzałki ma pojedynczy symbol nieterminalny, natomiast po prawej jest albo pojedynczy terminalny albo konkatenacja terminalnego z lewej i nieterminalnego z prawej.

1.3.2 Jaki język generuje gramatyka $G = (V, \Sigma, P, \sigma)$, jeśli

$V = \{x, y, z\}$, $\Sigma = \{A, B\}$, $P = \{A \to zyz, A \to Bz, B \to Bxy, B \to zy\}$, $\sigma = A$

Zauważmy, że musimy wystartować od A, a zatem jednym z wyprowadzeń na pewno będzie zyz. No ale możemy też z A zrobić Bz, a później w miejsce B doklejać albo Bxy dowolną ilość razy i lub zastosować produkcję $B \to zy$, która zakończy poprawne zdanie.

Stąd język generowany przez tę gramatykę:

$$J(G) = \{zy\,(xy)^n\,z, n \geqslant 0\}$$

1.3.3 Podaj gramatykę generującą język $J = \{a, b, ab, ba, aba, bab, abab, baba, \ldots\}$

Momentalnie rozpoznajemy symbole nieterminalne, a na nieterminalne decyduję się trzy. Właściwie możemy to rozpatrzeć jako dwie gramatyki. Dla treningu, jeżeli jeszcze nie widzicie, to zanim popatrzycie na odpowiedź spróbujcie znaleźć gramatykę dla języka $J = \{a, ab, aba, abab, ababa \ldots\}$.

Gramatyką dla tego konkretnego zadania będzie natomiast:

$$V = \{a, b\}$$

$$\Sigma = \{Z, X, Y\}$$

$$P = \{Z \to X, Z \to Y, X \to a, X \to aY, Y \to b, Y \to bX\}$$

$$\sigma = Z$$

1.3.4 Podaj gramatykę generującą język $J = \{a, ab, abc, abca, abcab, abcabc, abcabca, \ldots\}$

Tutaj będzie w sumie nawet prościej, bo przecież idąc w prawo będziemy sobie myśleć, aha no to albo kończę dodając ostatni znak albo dodam znak i coś co może mi wygenerować dalszą część słowa. Potrzebujemy zatem trzech symboli nieterminalnych dla możliwości skonstruowania każdego znaku terminalnego. A zatem:

$$V = \{a, b, c\}$$

$$\Sigma = \{X, Y, Z\}$$

$$P = \{X \to a, X \to aY, Y \to b, Y \to bZ, Z \to c, Z \to cX\}$$

$$\sigma = X$$

1.3.5 Dla gramatyki $G = (V, \Sigma, P, \sigma)$ znaleźć ciągi produkcji, które stanowią wywody:

$$V = \{a, b, c\}$$
$$\Sigma = \{X, Y, Z\}$$
$$P = \{Z \to cZc, Z \to acXcb, cX \to Xca, aXc \to aYc, Yca \to caY, Ycb \to cba\}$$
$$\sigma = Z$$

a) acacba

$$Z \Rightarrow acXcb \Rightarrow aXcacb \Rightarrow aYcacb \Rightarrow acaYcb \Rightarrow acacba$$

b) ccacacbacc

Pierwsze dwa kroki troszku inne, reszta żywce zdarta z poprzedniego podpunktu:

$$Z \Rightarrow cZc \Rightarrow ccZcc \Rightarrow ccacXcbcc \Rightarrow ccaXcacbcc \Rightarrow ccaYcacbcc \Rightarrow ccacaYcbcc \Rightarrow ccacacbacc$$

1.3.6 Do jakiej klasy należy gramatyka $G = \{V, P, p\}$, jeżeli:

a)

$$V = S \cup T, S = \{K, L\}, T = \{a, b, c\}$$
$$P = \{K \to KL, aK \to abK, L \to abc, cK \to cabc, bK \to bc, L \to abcKabc\}$$
$$p = L$$

Mamy do czynienia z gramatyką klasy pierwszej, czyli kontekstową. Nie jest regularna (np. przez pierwszą produkcję), nie jest bezkontekstowa (np. przez drugą produkcje, po lewej stronie strzałki mamy aK). Jest kontekstowa, bo jeżeli po lewej stronie strzałki mamy jakiś symbol nieterminalny poprzedzony symbolem terminalnym (np. w produkcji $aK \to abK$ symbol a), to po prawej stronie strzałki ten prefix, również jesteśmy w stanie znaleźć, po zmianie wspomnianego symbolu nieterminalnego.

b)

$$V = S \cup T, S = \{M, Q, R\}, T = \{a, b\}$$
$$P = \{M \to aM, M \to bQ, Q \to aR, R \to bR, R \to b\}$$
$$p = M$$

Jest to gramatyka klasy trzeciej, czyli regularna prawostronnie, dlatego że każda produkcja po lewej stronie ma pojedynczy symbol nieterminalny, a po prawej stronie wszędzie występuje albo ciąg (lub pojedynczy) symboli terminalnych, ewentualnie ciąg symboli terminalnych (lub pojedynczy) a po nich symbol nieterminalny.

c)

$$V = S \cup T, S = \{W\}, T = \{a, b, c, d\}$$
$$P = \{W \to abWc, bW \to cbW, Wc \to c\}$$
$$p = W$$

A to będzie (tak mi się wydaje) gramatyka klasy zerowej, czyli struktur frazowych. Nie jest regularna, ani bezkontekstowa (np. przez produkcję $bW \to cbW$), a kontekstowa nie jest dlatego, że występuje taka produkcja (moim zdaniem) $bW \to cbW$, dlatego że mamy to c... Żeby był kontekstowy, to prefix z symboli przed strzałką, po zamianie symbolu nieterminalnego powinien zostać zachowany (a nie zmieniany, czyli jak tam było b, to nagle nie możemy przemieć terminalnego b w terminalne cb, bo W nie ma takiego zasięgu).

A w myśl drugiej definicji (która mówi że gramatyka jest kontekstowa gdy to przed strzałką nie jest dłuższe niż to za strzałką) nie będzie ona kontekstowa ze względu na produkcję ($Wc \to c$).

1.3.7 Skonstruować gramatykę bezkontekstową, generującą język $J \subset \{a,b,c\}^*$,$J = \left\{(abc)^k(cba)^k\right\} : k > 1$

$$G = (V, \Sigma, P, \sigma)$$
$$V = \{a, b, c\}$$
$$\Sigma = \{X\}$$
$$P = \{X \to abcabccbacba, X \to abcXcba\}$$
$$\sigma = X$$

Tworzymy sobie takie ciągi *abcabccbacba*, *abcabcabccbacbacba* czyli k razy (k oczywiście większe od 1), wstawiamy sobie w środek taki dłuuuugi człon *abccba*. Druga produkcja jest po to, aby przypadkiem nie zrobić takiego zbyt krótkiego ciągu.

1.3.8 Skonstruować gramatykę bezkontekstową, generującą język $J \subset \{0,1\}^*$,$J = \left\{0^k1^{k+l}0^l\right\} : k > 0, l > 0$

$$G = (V, \Sigma, P, \sigma)$$
$$V = \{0, 1\}$$
$$\Sigma = \{X, Y, Z\}$$
$$P = \{Z \to 0X11Y0, X \to 0X1, X \to \epsilon, Y \to 1Y0, Y \to \epsilon\}$$
$$\sigma = Z$$

Najpierw zabezpieczymy się, aby spełnione były warunki $k > 0$ i $l > 0$, przez to że z symbolu startowego wypuścimy jedynie taki ciąg, który już zawiera przynajmniej po jednej parze 01 i 10, wraz z miejscami, w które możemy dopisywać kolejne pary lub symbol pusty, żeby sobie zakończyć.

1.4 Pytania, które dostaliśmy do przemyślenia

- Czy każda gramatyka klasy 2 jest jednocześnie gramatyką klasy 1?
- Co wytworzy domknięcie Kleene'ego gdy w alfabecie znajdzie się element neutralny?
- Czy elementy zbioru po operacji domknięcia Kleene'ego mogą się powtarzać? Rozważ alfabet $V = \{1, 11\}$,
- Czy istnieje alfabet dla którego zbiór wygenerowany poprzez domknięcie Kleene'ego jest skończona,

1.5 Test rozdziału

Spokojnie! To nie będzie nic strasznego, po prostu jeszcze pięć minutek zastanowienia się i próby przywołania czegoś z odmętów pamięci!

Spróbuj własnymi słowami wyjaśnić:

- Co produkuje domknięcie Kleene'ego?
- Czym wyróżniają się gramatyki regularne?
- Czy możliwe jest utworzenie gramatyki, dla której nie da się utworzyć zdania?

Spróbuj przypomnieć sobie gdzie wystąpiły pojęcia takie jak:

- Gramatyka bezkontekstowa,
- Symbol nieterminalny,
- Wyprowadzenie,
- Rekursja.

2 Ćwiczenia II

2.1 Rozgrzewka

Poznamy automaty deterministyczne (DAS) oraz niedeterministyczne z epsilon przejściami (ϵNAS). Będziemy badać łańcuchy akceptujące konkretne języki i poznamy dwie metody wygodnego przedstawiania takich automatów. Na koniec zajmiemy się kwestią konwersja ϵNAS do DAS.
Zanim przejdziemy do zadań, spróbuj odpowiedzieć na pytanie z czym właściwie kojarzą Ci się pojęcia:

- automatycznie,
- determinizm,
- akceptowanie,
- przejście.

Powiedz na głos jedno zdanie związane z każdymi z tych słów, niekoniecznie ich definicje.

Co wiąże ze sobą te słowa? Może są od siebie niezależne?

2.2 Zadania z ćwiczeń

2.2.1 Podaj DAS, który akceptuje język nad alfabetem $\{0,1\}$, który:

a) Jest zbiorem wszystkich łańcuchów zawierających podłańcuch $'101'$

No to jedziemy z tematem. Najpierw podrzucę tabelkę (postaram się też dorobić przynajmniej kilka grafów, generalnie dla mnie grafy są bardziej intuicyjne, ale niestety komputerowo ich przygotowanie zabiera więcej czasu).

	0	1
$> q_s$	q_s	q_1
q_1	q_{10}	q_1
q_{10}	q_s	q_{101}
$*q_{101}$	q_{101}	q_{101}

I teraz tabelkę czytamy tak. Pierwszy wiersz oznacza możliwe przeczytane symbole, a więc wypisujemy tam cały alfabet, pierwsza kolumna natomiast zawiera wszystkie możliwe stany automatu. Znaczek $>$ oznacza stan startowy, natomiast $*$ oznacza stan akceptujący (nie akceptowalny). I czytając łańcuch symbol po symbolu, skaczemy sobie po tabelce (po grafie wygodnie się skacze, wiem :/) i lodujemy w kolejnych stanach osiąganych przez dane słowo. Jeżeli zatrzymujemy się w stanie akceptującym to słowo to jest akceptowane przez automat, jeżeli nie, to nie.

Teraz tylko pytanie czy ten automat działa poprawnie? Sprawdźmy kiedy możemy się zatrzymać. Startujemy w q_s, każde przeczytane wtedy 0 sprowadza nas do tego samego stanu, ale jeśli wczytamy 1, to przeskoczymy do q_1.

Znajdując się w q_1 znów mamy dwie możliwości (w DAS musimy mieć tyle możliwości ile symboli alfabetu). Jeżeli wybierzemy 1 wracamy do q_1, co ma sens, no bo znów mamy szansę na wyłapanie potencjalnie podciągu $'101'$, a wprowadzenie symbolu 0 znacznie nas przybliża do osiągnięcia tego podciągu, dlatego w przypadku wczytania 0 przejdziemy do stanu q_{10}. (przy okazji sprawdza się powiedzenie Grzegorza Depczyńskiego „Dobre indeksowanie to skarb", nigdy bym się nie spodziewał, że sprawdzi się to na lingwistyce :d).

Stan q_{10} oznacza, że dwoma poprzednio wczytanymi symbolami były kolejno 1 i 0, a te które były wcześniejsze nas nie interesują (są to albo ciągi samych jedynek albo ciągi samych zer, przecinane ciągami jedyne, na pewno nie wystąpił jednak jeszcze szukany przez nas podciąg). Zauważmy, że jeżeli teraz wczytamy 1, to mamy już poszukiwany przez nas podciąg! W takim razie wczytanie 1 przeprowadza nas do stanu q_{101},

który może już wczytywać czego tylko chcemy i ile tylko chcemy i zawsze zaakceptuje słowo, w końcu szukany podciąg już wystąpił. A jeżeli w stanie q_{10} wczytamy 0, niestety, właściwie wracamy do punktu wyjścia, czyli do stanu q_s, no bo ostatnimi trzeba symbolami są kolejno 1, 0, i 0, (wiemy też że na pewno podciąg $'101'$ nie wystąpił wcześniej, bo byśmy go przecież złapali i wskoczyli do q_101), a skoro tak, to niestety podciągu $'101'$ musimy szukać od kolejnego wczytanego symbolu 1.

Będąc w stanie q_{101} oczywiście nie jesteśmy już w stanie przenieść się do żadnego innego stanu, co ma sens, ponieważ akceptuje on słowa z podciągiem $'101'$, a tylko takie są w stanie do niego „dotrzeć".

Rozpisałem się przy tym przykładzie dość mocno, dlatego myślę, że pozostałe powinny pójść znacznie łatwiej. Postaram się jedynie zaznaczać warte uwagi spostrzeżenia.

b) Jest zbiorem wszystkich łańcuchów zaczynających się od $'1'$ i kończących się na $'1'$

Idea jest prosta. Jeżeli wystartujemy z symbolu 0 (czyli jako pierwszy wczytamy symbol 0), to trafiamy do stanu martwego, czyli takiego z którego nie jesteśmy w stanie uciec, ani osiągnąć stanu akceptującego. Jak wczytaliśmy 1 jako pierwsze, to przeskakujemy do „podautomatu", który sprawdza ostatni symbol. Poskaczcie sobie z kilkoma symbolami i szybko da się ogarnąć.

	0	1
$> q_s$	q_m	q_{ok}
q_1	q_1	q_{ok}
$*q_{ok}$	q_1	q_{ok}
q_m	q_m	q_m

c) Jest zbiorem wszystkich łańcuchów zaczynających się od $'0'$ i kończących się na $'0'$

	0	1
$> q_s$	q_{ok}	q_m
q_1	q_{ok}	q_1
$*q_{ok}$	q_{ok}	q_1
q_m	q_m	q_m

Co się tutaj zmieniło? Zamieniliśmy kolumny 0 i 1 miejscami. Struktura grafu jest identyczna, tylko symbole na strzałkach grafu stały się przeciwne. (oczywiście dobrze to widać na grafach, jeżeli jeszcze nie zdążyłem ich wstawić spróbujcie przygotować i zauważyć to sami).

d) Jest zbiorem wszystkich łańcuchów zaczynających się od $'0'$ lub $'1'$ i kończących się na $'1'$

Czyli lekka kombinacja tych dwóch poprzednich, ale DAS będzie nawet łatwiejszy!

	0	1
$> q_s$	q_{tmp}	q_{ok}
q_{tmp}	q_{tmp}	q_{ok}
$*q_{ok}$	q_{tmp}	q_{ok}

Wychodzi na to, że DAS lubi dysjunkcje.

e) Jest zbiorem wszystkich łańcuchów zawierających podłańcuch $'001'$

Mocna analogia do podpunktu a), będzie jednak odrobinę inaczej!

	0	1
$> q_s$	q_0	q_s
q_0	q_{00}	q_s
q_{00}	q_{00}	q_{001}
$*q_{001}$	q_{001}	q_{001}

Przede wszystkim inaczej nazwaliśmy stany, jednak ich ilość jest identyczna (oczywiście od każdego musi być dwa możliwe przejścia, czyli tyle ile znaków w alfabecie), ale generalnie koncepcja jest baaardzo analogiczna.

f) Jest zbiorem wszystkich łańcuchów kończących się na $'00'$

Tutaj każda wczytany symbol 1 będzie nas jakby wracał na sam start, chyba że dostaniemy dwa symbole 0 z rzędu, wtedy trafiamy prosto w stan akceptujący.

	0	1
$> q_s$	q_0	q_s
q_0	q_{00}	q_s
$*q_{00}$	q_{00}	q_s

Ze stanu akceptującego q_{00} jest w stanie wyrzucić nas jedna marna 1 i musimy wszystko znów zaczynać od stanu q_s...

g) Jest zbiorem wszystkich łańcuchów zawierających w sobie $'000'$

Zadanie bardzo podobne do podpunkut a).

	0	1
$> q_s$	q_0	q_s
q_0	q_{00}	q_s
q_{00}	q_{000}	q_s
$*q_{000}$	q_{000}	q_{000}

Próbujemy złapać to 000, a więc każdy ciąg jedynek sprowadza nas do punktu startowego, chyba że zaczną się jakieś 0, no to łapiemy i przeskakujemy dalej, z tym że... Każde 1 sprowadza nas znów na punkt startowy, ponieważ szukamy tylko czystych rasowo ciągów 000. Kiedy takowy już złapiemy, zatrzymujemy się w stanie akceptującym i już go nie opuszczamy cokolwiek by się działo.

h) Jest zbiorem wszystkich łańcuchów posiadających $'1'$ na trzeciej pozycji od końca

No niezła mi kobyła tutaj wyszła... (Jeśli ktoś ma lepszy pomysł, to proszę o wsparcie). Ale zdecydowanie dało się to zrobić jeszcze bardziej rozbudowane, więc nie jest jeszcze tak źle, do dzieła.

	0	1
$> q_s$	q_s	q_1
q_1	q_{10}	q_{11}
q_{11}	q_{110}	q_{111}
q_{10}	q_{110}	q_{101}
$*q_{110}$	q_{100}	q_{101}
$*q_{111}$	q_{110}	q_{111}
$*q_{101}$	q_{10}	q_{11}
$*q_{100}$	q_s	q_1

No cóż, najlepiej jest rozrysować sobie graf i zobaczyć te plątaninę krawędzi skierowanych. Ale generalnie wystarczy rozpatrzeć przypadki: 0000,0100,0110,0101,0001, żeby zobaczyć dlaczego tak.

2.2.2 Opisać jaki język akceptują DAS:

a)

	0	1
$> q_1$	q_1	q_2
$*q_2$	q_1	q_2

Ze stanu startowego za sprawą symbolu 1 przeskakujemy do q_2, gdzie możemy „wciskać" ile chcemy jedynek,a jedno 0 sprowadza nas do stanu q_1. Jako że kończący jest stan q_2, to na pewno ostatnim symbolem będzie 1.

Język ten możemy opisać jako dowolny ciąg 0 i 1 kończący się na $'1'$.

Nie jestem pewien, ale chyba formalnie moglibyśmy zapisać to jako $\{x^n y : x \in \{'1', '0'\} \wedge y =' 1' \wedge n \geqslant 0\}$

b)

	0	1
$> *q_1$	q_1	q_2
q_2	q_1	q_2

Oczywiście znak $> *$ oznacza, że stan startowy jest również akceptujący.

A sytuacja jest bardzo podobna, z jedną zasadniczą różnicą. Możemy skończyć bez wczytania żadnego znaku, a więc znak pusty ϵ również będzie akceptowalny. Każda inna kombinacja 0 i 1 musi kończyć się symbolem 0.

Znów nie mam pewności, ale chyba moglibyśmy zapisać to jako $\{\epsilon, x^n y : x \in \{'1', '0'\} \wedge y =' 0' \wedge n \geqslant 0\}$

c)

	0	1	2	$< RESET >$
$> *q_0$	q_0	q_1	q_2	q_0
q_1	q_1	q_2	q_0	q_0
q_2	q_2	q_0	q_1	q_0

Śmiesznie to działa, bo właściwie przez tę kolumnę $< Reset >$ każde słowo kończące się symbolem $< Reset >$ będzie akceptujące. Nie biorąc pod uwagę $< Reset >$ opisujemy tutaj dodawanie modulo liczb 0,1,2, takie że suma takiego słowa jest równa 0 (Przez ten $< Reset >$ dopuszczalne są też wyniki 1 i 2, ale muszą się skończyć znakiem $< Reset >$). Czyli reasumując dowolne kombinacje '0', '1', '2' i '$< Reset >$', takie że po ostatnim resecie (jeżeli wystąpił), suma modulo 3 symboli musi wynosić 0.

2.2.3 Opisać jaki jest akceptowany przez ϵNAS:

dla ϵNAS:

	0	ϵ
$> q_s$	$\emptyset$	$\{q_{g1}, q_{d1}\}$
$*q_{g1}$	$\{q_{g2}\}$	$\emptyset$
q_{g2}	$\{q_{g1}\}$	$\emptyset$
$*q_{d1}$	$\{q_{d2}\}$	$\emptyset$
q_{d2}	$\{q_{d3}\}$	$\emptyset$
q_{d3}	$\{q_{d1}\}$	$\emptyset$

Spotykamy się tutaj z automatem niedeterministycznym, a więc analizując go dla jakiegoś łańcucha, w przypadku więcej niż jednej możliwości przeskoku, tworzymy rozgałęzienie wyprowadzenia. W związku z tym zauważmy, że dowolne słowo (czyli dowolny ciąg symboli 0) zaraz po wyjściu ze stanu startowego, rozdzieli się na dwa wyprowadzenia, które następnie będą sobie w takich jakby „pętlach" analizowane, niezależnie od siebie (dobre zadanie na współbiegi). Dodatkowo zauważmy, że stan początkowy jest też akceptujący, a więc ϵ również będzie akceptowany.

Język akceptowany przez ten automat to J = $\{\epsilon, 0^n : n > 1\,(n \mid 2 \vee n \mid 3)\}$. Czyli takie ciągi zer, żeby ich liczba była podzielna przez 2 lub przez 3, no i oczywiście słowo puste ϵ

2.2.4 Rozrysuj działanie ϵNAS dla łańcuchów:

ϵNAS:

	0	1	ϵ
$> q_1$	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
$*q_4$	$\{q_4\}$	$\{q_4\}$	$\emptyset$

Żeby łatwo było korygować ewentualne błędy, będę to robił na zasadzie wypunktowania (dla wielu rozgałęzień będzie cholernie nieczytelne, ale tutaj na szczęście nie będzie ich aż tak wiele).

Ważna uwaga! Jeżeli u mnie w przejściu pojawia się „$-\epsilon->$" to przy oficjalnym rozpisywaniu nie powinno się tego pisać (czyli wtedy gdy rozpisujemy to drzewkiem tak jak na zajęciach, wtedy po prostu od danego przejścia dokładamy też wszystkie ϵ przejścia). Ja zastosowałem to dopisanie, żeby lepiej było widać co skąd się bierze.
a) 010110

1. Ładujemy symbol 0: a więc mamy jedną ścieżkę:
   (a) $q_1 - 0- > q_1$
2. Ładujemy symbol 1: a więc dostajemy dwie ścieżki ze ścieżki „a)" i dodatkowo w związku z tym, że ze stanu q_2 mamy ϵ przejście, to tworzy się kolejna ścieżka, bez ładowania znaku:
   (a) $q_1 - 0- > q_1 - 1- > q_1$
   (b) $q_1 - 0- > q_1 - 1- > q_2$
   (c) $q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3$
3. Ładujemy symbol 0: a więc w ścieżce a) nic się nie zmienia, a w ścieżce b) zmieniamy stan, ścieżka c) umiera, dlatego że załadowany symbol ma zbiór pusty stanów do przejścia:
   (a) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1$
   (b) $q_1 - 0- > q_1 - 1- > q_2 - 0- > q_3$
   (c) $q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$
4. Ładujemy symbol 1: a więc ze ścieżki a) wydzielają się dwie nowe ścieżki, więc starą ścieżkę b) dajemy o dwa w dół i staje się ścieżką d), a starą martwą ścieżkę c) staje się ścieżką e):
   (a) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1$
   (b) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2$
   (c) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3$
   (d) $q_1 - 0- > q_1 - 1- > q_2 - 0- > q_3 - 1- > q_4$
   (e) $q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$
5. Ładujemy symbol 1: Noo, teraz to się troszkę pozmienia... Ze ścieżki a) zostają wydzielone dwie nowe ścieżki, ścieżka b) umiera, ścieżka c) przechodzi do nowego stanu, a ścieżka d) pozostaje w swoim stanie (oczywiście ścieżek martwych nie będziemy już modyfikować):
   (a) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 1- > q_1$

(b) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 1- > q_2$

(c) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 1- > q_2 - \epsilon- > q_3$

(d) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - 1- > dead$

(e) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3 - 1- > q_4$

(f) $q_1 - 0- > q_1 - 1- > q_2 - 0- > q_3 - 1- > q_4 - 1- > q_4$

(g) $q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

6. Ładujemy ostatni symbol - $'0'$. Teraz wystarczy aby tylko jedna ze ścieżek zakończyła się na stanie akceptującym i na szczęście zrobi to między innymi nasza ulubiona, ostatnia ścieżka (ulubiona bo się w poprzedniej iteracji ładnie zachowywała i się nie zmieniała):

   (a) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 1- > q_1 - 0- > q_1$

   (b) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 1- > q_2 - 0- > q_3$

   (c) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

   (d) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - 1- > dead$

   (e) $q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3 - 1- > q_4 - 0- > q_4 - -sukces$

   (f) $q_1 - 0- > q_1 - 1- > q_2 - 0- > q_3 - 1- > q_4 - 1- > q_4 - 0- > q_4 - -sukces$

   (g) $q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

b) 101001

1. Ładujemy symbol 1: a więc generujemy w sumie trzy ścieżki (patrz poprzedni podpunkt apropo ϵ przejść):

   (a) $q_1 - 1- > q_1$

   (b) $q_1 - 1- > q_2$

   (c) $q_1 - 1- > q_2 - \epsilon- > q_3$

2. Ładujemy symbol 0: ścieżka c) ginie, w pierszej nic się nie zmienia, a druga zmienia stan na q_3:

   (a) $q_1 - 1- > q_1 - 0- > q_1$

   (b) $q_1 - 1- > q_2 - 0- > q_3$

   (c) $q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

3. Ładujemy symbol 1: ścieżka a) generuje dwie nowe ścieżki, b) zmienia stan na q_4 i już nie będzie się zmieniać:

   (a) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1$

   (b) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2$

   (c) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3$

   (d) $q_1 - 1- > q_2 - 0- > q_3 - 1- > q_4$

   (e) $q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

4. Ładujemy symbol 0: ścieżka a) się nie zmienia, ścieżka b) zmienia stan na q_3, ścieżka c) ginie, no a ścieżka d), tak jak pisałem wcześniej już się nie będzie zmieniała:

   (a) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1$

   (b) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - 0- > q_3$

   (c) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

   (d) $q_1 - 1- > q_2 - 0- > q_3 - 1- > q_4 - 0- > q_4$

(e) $q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

5. Ładujemy symbol 0: ścieżka a) się nie zmienia, ścieżka b) ginie, a ścieżka d) już dawałaby nam odpowiedź czy słowo akceptowane czy nie (i to od dawna!):

   (a) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 0- > q_1$

   (b) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - 0- > q_3 - 0- > dead$

   (c) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

   (d) $q_1 - 1- > q_2 - 0- > q_3 - 1- > q_4 - 0- > q_4 - 0- > q_4$

   (e) $q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

6. Ładujemy symbol 1: ścieżka a) generuje dwie nowe ścieżki (ale to już ma niewielkie znaczenie w sumie), a ścieżka d)osiąga sukces:

   (a) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 0- > q_1 - 1- > q_1$

   (b) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 0- > q_1 - 1- > q_2$

   (c) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_1 - 0- > q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3$

   (d) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - 0- > q_3 - 0- > dead$

   (e) $q_1 - 1- > q_1 - 0- > q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

   (f) $q_1 - 1- > q_2 - 0- > q_3 - 1- > q_4 - 0- > q_4 - 0- > q_4 - 1- > q_4 - -sukces$

   (g) $q_1 - 1- > q_2 - \epsilon- > q_3 - 0- > dead$

c) 000111

1. Ładujemy symbol 0: mamy jedną ścieżkę i nic się nie zmienia (jako że mamy 000, to pominę dwa kolejne kroki i wstawię jako jedno przejście:

   (a) $q_1 - 000- > q_1$

2. Ładujemy symbol 1: generują się dwie nowe ścieżki ze ścieżki a):

   (a) $q_1 - 000- > q_1 - 1- > q_1$

   (b) $q_1 - 000- > q_1 - 1- > q_2$

   (c) $q_1 - 000- > q_1 - 1- > q_2 - \epsilon- > q_3$

3. Ładujemy symbol 1: ze ścieżki a) generują się dwie nowe ścieżki, ścieżka b) ginie, a ścieżka c) zmienia stan na akceptujący:

   (a) $q_1 - 000- > q_1 - 1- > q_1 - 1- > q_1$

   (b) $q_1 - 000- > q_1 - 1- > q_1 - 1- > q_2$

   (c) $q_1 - 000- > q_1 - 1- > q_1 - 1- > q_2 - \epsilon- > q_3$

   (d) $q_1 - 000- > q_1 - 1- > q_2 - 1- > dead$

   (e) $q_1 - 000- > q_1 - 1- > q_2 - \epsilon- > q_3 - 1- > q_4$

4. Ładujemy symbol 1: ze ścieżki a) generują się dwie nowe ścieżki, ścieżka b) ginie, a ścieżka c) zmienia stan na akceptujący, (a ścieżka e) to już dawno akceptuje!):

   (a) $q_1 - 000- > q_1 - 1- > q_1 - 1- > q_1 - 1- > q_1$

   (b) $q_1 - 000- > q_1 - 1- > q_1 - 1- > q_1 - 1- > q_2$

   (c) $q_1 - 000- > q_1 - 1- > q_1 - 1- > q_1 - 1- > q_2 - \epsilon- > q_3$

   (d) $q_1 - 000- > q_1 - 1- > q_1 - 1- > q_2 - 1- > dead$

(e) $q_1 - 000 -> q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4 - -sukces$

(f) $q_1 - 000 -> q_1 - 1 -> q_2 - 1 -> dead$

(g) $q_1 - 000 -> q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4 - 1 -> q_4 - -sukces$

d) 111000

1. Ładujemy symbol 1: Od razu ścieżka utworzy nam dwie nowe ścieżki:
   (a) $q_1 - 1 -> q_1$
   (b) $q_1 - 1 -> q_2$
   (c) $q_1 - 1 -> q_2 - \epsilon -> q_3$
2. Ładujemy symbol 1: Ze ścieżki a) dwie nowe ścieżki ...
   (a) $q_1 - 1 -> q_1 - 1 -> q_1$
   (b) $q_1 - 1 -> q_1 - 1 -> q_2$
   (c) $q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3$
   (d) $q_1 - 1 -> q_2 - 1 -> dead$
   (e) $q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4$
3. Ładujemy symbol 1: Ze ścieżki a) kolejne dwie ścieżki!
   (a) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_1$
   (b) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_2$
   (c) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3$
   (d) $q_1 - 1 -> q_1 - 1 -> q_2 - 1 -> dead$
   (e) $q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4$
   (f) $q_1 - 1 -> q_2 - 1 -> dead$
   (g) $q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4 - 1 -> q_4$
4. Ładujemy symbol 0: Nareszcie ścieżka a) się trochę uspokoiła... Mamy już akceptujące, ale dojedźmy z tym do końca!
   (a) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_1 - 0 -> q_1$
   (b) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_2 - 0 -> q_3$
   (c) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3 - 0 -> dead$
   (d) $q_1 - 1 -> q_1 - 1 -> q_2 - 1 -> dead$
   (e) $q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4 - 0 -> q_4$
   (f) $q_1 - 1 -> q_2 - 1 -> dead$
   (g) $q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4 - 1 -> q_4 - 0 -> q_4$
5. Ładujemy symbol 0: Teraz ścieżki a), e), g) raczej nie powinny już fikać
   (a) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_1 - 0 -> q_1 - 0 -> q_1$
   (b) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_2 - 0 -> q_3 - 0 -> dead$
   (c) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3 - 0 -> dead$
   (d) $q_1 - 1 -> q_1 - 1 -> q_2 - 1 -> dead$
   (e) $q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4 - 0 -> q_4 - 0 -> q_4$
   (f) $q_1 - 1 -> q_2 - 1 -> dead$

(g) $q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4 - 1 -> q_4 - 0 -> q_4 - 0 -> q_4$

6. Ładujemy symbol 0: I mamy dwie akceptujące

(a) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_1 - 0 -> q_1 - 0 -> q_1 - 0 -> q_1$
(b) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_2 - 0 -> q_3 - 0 -> dead$
(c) $q_1 - 1 -> q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3 - 0 -> dead$
(d) $q_1 - 1 -> q_1 - 1 -> q_2 - 1 -> dead$
(e) $q_1 - 1 -> q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4 - 0 -> q_4 - 0 -> q_4 - 0 -> q_4 - -sukces$
(f) $q_1 - 1 -> q_2 - 1 -> dead$
(g) $q_1 - 1 -> q_2 - \epsilon -> q_3 - 1 -> q_4 - 1 -> q_4 - 0 -> q_4 - 0 -> q_4 - 0 -> q_4 - -sukces$

2.2.5 Przekonwertować ϵNAS na DAS:

Właściwie przed przeczytaniem tego i następnego podrozdziału proponuję przejrzeć konwersje NAS do DAS z następnego rozdziału.
a) (z grafu z kartki do tabelki przerobiłem)

	a	b	ϵ
$> *q_1$	$\emptyset$	$\{q_2\}$	$\{q_3\}$
q_2	$\{q_2, q_3\}$	$\{q_3\}$	$\emptyset$
q_3	$\{q_1\}$	$\emptyset$	$\emptyset$

Jako, że jest to pierwszy przykład konwertowania, zrobię go tak dokładnie jak ten na zajęciach, czyli będziemy mieli trzy etapy, a końcowy to będzie już gotowy DAS (gdyby to było przejście NAS do DAS, to wystarczyłby w sumie dwa etapy, a właściwie nawet jeden).

Najpierw musimy zamienić kolumnę ϵ w kolumnę $\epsilon*$ (czyt. epsilon domknięcie), kolumna ta oznacza, gdzie se stanu w danym wierszu możemy dostać się przy pomocy zera, jednego, dwóch itp. ϵ przejść (Oczywistym jest, że zawsze stan z wiersza, który aktualnie rozpatrujemy się tam znajdzie). Tutaj ważna uwaga, abyśmy pamiętali o tym, żeby nie zatrzymać się na złapaniu jednego epsilon przejścia, te zbiory $\epsilon*$ lubią bardzo spuchnąć, bo czasami kilka stanów po przekonwertowaniu złapie nam się elegancko w jeden, dlatego pamiętajcie żeby złapać wszystkie te przejścia! Jest to szczególnie istotne, gdy poprzez dodanie któregoś wiersza w jego $\epsilon*$ znajdzie się jakiś stan, którego jeszcze wcześniej nie mieliśmy, trzeba pamiętać o dodaniu również jego $\epsilon*$. (Np. ostatnie zadanie treningowe).

	a	b	$\epsilon*$
$> *q_1$	$\emptyset$	$\{q_2\}$	$\{q_1, q_3\}$
q_2	$\{q_2, q_3\}$	$\{q_3\}$	$\{q_2\}$
q_3	$\{q_1\}$	$\emptyset$	$\{q_3\}$

Teraz najważniejszy moment (od razu wykonam tutaj wersję zoptymalizowaną, gdzie dodajemy stany osiągalne ze startowego), musimy scalić $\epsilon*$ z poszczególnymi słowami alfabetu (w tym przypadku a, b). Scalenie będzie odbywało się w ten sposób, bierzemy sobie jakiś wiersz, patrzymy na $\epsilon*$ w tym wierszu, następnie dla każdego stanu z podejrzanego $\epsilon*$ patrzymy gdzie przenosi nas dane słowo alfabetu (w sensie do których wierszy) i bierzemy sumę teoriomnogościową z $\epsilon*$-ek każdego z nich i to dopiero wpisujemy w odpowiednie miejsce wypełnianej tabelki. W kolejnych linijkach będziemy umieszczać takie zbiory stanów, które wystąpiły już w naszej tabelce. Aha, ważna rzecz, bo jeszcze startujemy nie ze stanu początkowego, ale z $\epsilon*$ stanu początkowego! Wszystko się wyjaśni naszym przykładem!

	a$\epsilon*$	b$\epsilon*$
$> *\{q_1, q_3\}$	$\{q_1, q_3\}$	$\{q_2\}$
$\{q_2\}$	$\{q_2, q_3\}$	$\{q_3\}$
$\{q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_3\}$
$*\{q_1, q_2, q_3\}$	$\{q_1, q_2, q_3\}$	$\{q_2, q_3\}$
$\{q_3\}$	$\{q_1, q_3\}$	$\emptyset$

Wypełnianie wygląda tak, zaczynam od domknięcia dla stanu początkowego i teraz równolegle dla q_1 i q_3 spojrzałem na ich domknięcia i ze stanów które tam przeczytałem, wziąłem sobie wszystkie stany, do których prowadziły ich kolejno przejścia a oraz b i z tych stanów wpisałem $\epsilon*$-cia w odpowiednie miejsce. Np. zobaczmy, że stan q_1 ma $\epsilon*$ ze stanami q_1 oraz q_3, w takim razie patrzymy jakie stany mamy w kolumnie $a\epsilon*$ dla stanów q_1 oraz q_3, okazuje się że jest tam zbiór pusty oraz stan q_1, a więc wpisujemy to, na co wskazuje $\epsilon*$ stanu q_1 czyli $\{q_1, q_3\}$, analogicznie dla kolumny b.

Teraz mamy moment optymalizacyjny, ponieważ nie przepisujemy wszystkich możliwych kombinacji, tylko rozpisujemy sobie jeden nowy wiersz, bo pojawiły się świeżutki zbiór stanów, więc pojawi się wiersz q_2. Rozpisujemy go analogicznie i zauważm, że wypełniając kolumnę $a\epsilon*$ dla q_2 pojawia się znowu nowy zbiór stanów q_2, q_3, a więc zapisujemy go jako kolejny wiersz i tak powtarzamy dopóki nie skończą nam się nowe, wcześniej niezbadane zbiory stanów, więc kończymy gdy niczego nowego nie jesteśmy w stanie dodać.

Teraz nadszedł czas na ostatni etap, czyli zamianę tego ϵ potworka na prawilnego DAS'a. Aby to zrobić musimy zmienić dwie rzeczy, po pierwsze dla każdego zbioru stanów tworzymy nowy stan, który będzie go wyrażał (można np. stan $q_{itutajwszystkiecyferkizezbioru}$, drugą sprawą, która akurat w tym przypadku wyszła, chociaż nie musi wychodzić zawsze jest to, że stan q_3 po załadowaniu b prowadzi do zbioru pustego, a więc musimy jeszcze na sam koniec dorzucić stan imitujący zbiór pusty (ja nazywam go sobie $\epsilon\phi$, nie da się z niego uciec, oraz nie jest akceptujący.

Po zmianach otrzymujemy:

	a	b
$> *q_{1,3}$	$q_{1,3}$	q_2
q_2	$q_{2,3}$	q_3
$q_{2,3}$	$q_{1,2,3}$	q_3
$*q_{1,2,3}$	$q_{1,2,3}$	$q_{2,3}$
q_3	$q_{1,3}$	q_ϕ
q_ϕ	q_ϕ	q_ϕ

Teraz jak ktoś ma ochotę, to można te trzy etapy sobie złączyć w dwa, tzn. najpierw wyznaczyć $\epsilon*$ gdzieś tam pomocniczo, a następnie na podstawie tabelki ϵ-NAS od razu stworzyć DAS.

b) (z zadania 4)

	0	1	ϵ
$> q_1$	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
$*q_4$	$\{q_4\}$	$\{q_4\}$	$\emptyset$

W tym przypadku spróbujemy wykonać wersję wspomnianą na końcu poprzedniego podpunktu (tradycyjna powinna dawać ten sam wynik). A zatem pomocniczo stworzymy sobie etap pierwszy, czyli wszystkie $\epsilon*$ wejściowego ϵNAS:

	0	1	$\epsilon*$
$> q_1$	$\{q_1\}$	$\{q_1, q_2\}$	$\{q_1\}$
q_2	$\{q_3\}$	$\emptyset$	$\{q_2, q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\{q_3\}$
$*q_4$	$\{q_4\}$	$\{q_4\}$	$\{q_4\}$

I teraz od razu będziemy sobie wypełniać tabelkę DAS. Zaczynamy od stanu q_1, dlatego że dla stanu q_1 w naszej tabelce w kolumnie $\epsilon*$ znajduje się właśnie stan q_1. q_1 przenosi nas poprzez 0 tylko do wiersza z q_1, a $\epsilon*$ w tym wierszu to nędzne q_1, a więc to wpisujemy w rozpatrywaną kratkę. Ciekawiej jest gdy dla q_1 rozpatrzymy przenoszenie poprzez 1, ponieważ weźmiemy pod uwagę wiersz z q_1 oraz wiersz z q_2 i chociaż $\epsilon*$ dla pierwszego z nich jest dość trywialna, to tego drugiego jest już nieco szersza, więc wynikowo wpiszemy sobie w kratkę stan $q_{1,2,3}$. I tak dalej, akceptujące będą oczywiście tylko te stany, które będą pochodziły od stanu q_4.

	0	1
$> q_1$	q_1	$q_{1,2,3}$
$q_{1,2,3}$	$q_{1,3}$	$q_{1,2,3,4}$
$q_{1,3}$	q_1	$q_{1,2,3,4}$
$*q_{1,2,3,4}$	$q_{1,3,4}$	$q_{1,2,3,4}$
$*q_{1,3,4}$	$q_{1,4}$	$q_{1,2,3,4}$
$*q_{1,4}$	$q_{1,4}$	$q_{1,2,3,4}$

Kolumna dla '1' wygląda dość... jednorodnie.

c) (z zadania 3)

	0	ϵ
$> q_s$	$\emptyset$	$\{q_{g1}, q_{d1}\}$
$*q_{g1}$	$\{q_{g2}\}$	$\emptyset$
q_{g2}	$\{q_{g1}\}$	$\emptyset$
$*q_{d1}$	$\{q_{d2}\}$	$\emptyset$
q_{d2}	$\{q_{d3}\}$	$\emptyset$
q_{d3}	$\{q_{d1}\}$	$\emptyset$

I znów spróbujemy metody przyśpieszonej, najpierw tworząc kolumnę $\epsilon*$, po to żeby później od razu przejść do DAS.

	0	$\epsilon*$
$> q_s$	$\emptyset$	$\{q_s, q_{g1}, q_{d1}\}$
$*q_{g1}$	$\{q_{g2}\}$	$\{q_{g1}\}$
q_{g2}	$\{q_{g1}\}$	$\{q_{g2}\}$
$*q_{d1}$	$\{q_{d2}\}$	$\{q_{d1}\}$
q_{d2}	$\{q_{d3}\}$	$\{q_{d2}\}$
q_{d3}	$\{q_{d1}\}$	$\{q_{d3}\}$

Właściwie ϵ ma jakieś znaczenie na początku, no ale właśnie na początku, powtórze się, ale co tam! Zaczynamy od stanu, który będzie oznaczał $\epsilon*$ stanu początkowego!

	0
$> *q_{s,g1,d1}$	$q_{g2,d2}$
$q_{g2,d2}$	$q_{g1,d3}$
$*q_{g1,d3}$	$q_{g2,d1}$
$*q_{g2,d1}$	$q_{g1,d2}$
$*q_{g1,d2}$	$q_{g2,d3}$
$q_{g2,d3}$	$q_{g1,d1}$
$*q_{g1,d1}$	$q_{g2,d2}$

Czyli właściwie powstał stan dla każdej z kombinacji tej całej podzielności przez 2 lub przez 3. Dodatkowo stan początkowy $q_{s,g1,d1}$ musi być akceptujący, ponieważ w swojej nazwie zawiera stany akceptujące (a że przyjęliśmy sobie taką a nie inną konwencję, to nazwa wynika bezpośrednio ze zbiorów stanów, więc ma to sens).

2.2.6 Zdefiniować ϵNAS dla języka $L_1 \cup L_2$ (suma teoriomnogościowa) i przekonwertować to na DAS, dla L_1 i L_2 jako:

a) L1 język określony w zadaniu 1b, L2 język określony w zadaniu 1c

Czyli język taki słów, które zaczynają się od '0' i kończą na '0' lub zaczynają się na '1' i kończy na '1'

Ogólnie strategia wyglądałaby tak, robimy sobie dwa *epsilon* przejścia do dwóch osobnych „podmaszyn", które sobie rozpoznają każdy z tych języków. Wtedy te *epsilon* przejścia ze stanu początkowego tworzą takie alternatywne (żeby nie powiedzieć współbieżne) rozpoznawania. Podmaszyny bierzemy sobie żywcem z zadania pierwszego, przy czym skoro tworzymy ϵNAS, możemy nie tworzyć specjalnych stanów, pułapek, tylko w odpowiednie miejsca wstawić prowadzenie do zbioru pustego.

	0	1	ϵ
$> q_s$	$\emptyset$	$\emptyset$	$\{q_{gs}, q_{ds}\}$
q_{gs}	$\emptyset$	$\{q_{gok}\}$	$\emptyset$
q_{g0}	$\{q_{g0}\}$	$\{q_{gok}\}$	$\emptyset$
$*g_{gok}$	$\{q_{g0}\}$	$\{q_{gok}\}$	$\emptyset$
q_{ds}	$\{q_{dok}\}$	$\emptyset$	$\emptyset$
q_{d1}	$\{q_{dok}\}$	$\{q_{d1}\}$	$\emptyset$
$*q_{dok}$	$\{q_{dok}\}$	$\{q_{d1}\}$	$\emptyset$

No i konwersja tego potwora do DAS. Nie będziemy nawet ustawiać tabelki ϵ przejść, bo ϵ przejście występuje tylko ze stanu początkowego, dlatego ustalmy prawidłowy stan początkowy (czyli $\epsilon*$ w stanie początkowym, gdybyśmy to rozpisywali) i jedziemy.

	0	1
$> *q_{s,gs,ds}$	q_{dok}	q_{gok}
$*q_{dok}$	q_{dok}	q_{d1}
$*q_{gok}$	q_{g0}	q_{gok}
q_{d1}	q_{dok}	q_{d1}
q_{g0}	q_{g0}	q_{gok}

W sumie spodziewałem się czegoś bardziej skomplikowane, ale wygląda na to, że ma sens.

b) L1 język określony w zadaniu 1a, L2 język określony w zadaniu 1f

Znów dwie alternatywne drogi wyboru od startu łapiące poszczególne podjęzyki, czyli L1 to zbiór łańcuchów zawierających podłańcuch '101', a L2 to tych kończących się na '00'.

	0	1	ϵ
$> q_s$	$\emptyset$	$\emptyset$	$\{q_{g1}, q_{d1}\}$
q_{g1}	$\{q_{g1}\}$	$\{q_{g2}\}$	$\emptyset$
q_{g2}	$\{q_{g3}\}$	$\{q_{g2}\}$	$\emptyset$
q_{g3}	$\{q_{g1}\}$	$\{q_{g4}\}$	$\emptyset$
$*q_{g4}$	$\{q_{g4}\}$	$\{q_{g4}\}$	$\emptyset$
q_{d1}	$\{q_{d2}\}$	$\{q_{d1}\}$	$\emptyset$
q_{d2}	$\{q_{d3}\}$	$\{q_{d1}\}$	$\emptyset$
$*q_{d3}$	$\{q_{d3}\}$	$\{q_{d1}\}$	$\emptyset$

Znów konwertując to do DAS, możemy nie tworzyć pomocniczej $\epsilon*$, tylko poprawnie wystartować i już będzie dobrze.

	0	1
$> q_{s,g1,d1}$	$q_{g1,d2}$	$q_{g2,d1}$
$q_{g1,d2}$	$q_{g1,d3}$	$q_{g2,d1}$
$q_{g2,d1}$	$q_{g3,d2}$	$q_{g2,d1}$
$*q_{g1,d3}$	$q_{g1,d3}$	$q_{g2,d1}$
$q_{g3,d2}$	$q_{g1,d3}$	$q_{g4,d1}$
$*q_{g4,d1}$	$q_{g4,d2}$	$q_{g4,d1}$
$*q_{g4,d2}$	$q_{g4,d3}$	$q_{g4,d1}$
$*q_{g4,d3}$	$q_{g4,d3}$	$q_{g4,d1}$

Aż się mienią w oczach te jedynki i dwójki, ale chyba działa.

2.3 Konwersje NAS do DAS z trzeciej kartki

Kolejność będzie inna niż na kartce, będzie za to taka jak na zajęciach.

2.3.1 Dany jest NAS podać język i przekonwertować na DAS

	0	1
$> q_1$	$\{q_1\}$	$\{q_1, q_2\}$
q_2	$\{q_3\}$	$\emptyset$
$*q_3$	$\emptyset$	$\emptyset$

a) Językiem są ciągi zer i jedynek, które kończą się na '10'

b) Konwersja do DAS.

Wykonamy sobie po raz pierwszy w tym skrypcie pełną konwersję, czyli taką, która bierze pod uwagę wszystkie możliwe podzbiory zbioru stanów. (w ten sposób bardzo często generowane jest mnóstwo stanów nieosiągalnych ze stanu początkowego, które są całkowicie zbędne). Raz to pokażmy i na dostatecznie małym przypadku, ten się nada.

Zrobimy tabelkę NAS, w której w pierwszej kolumnie znajdą się wszystkie podzbiory, zbioru stanów, (stąd właśnie nawiasy wąsiaste w pierwszej kolumnie). I oczywiście zbiory akceptujące to te, w których znajduje się przynajmniej jeden ze stanów akceptujących, w tym przypadku stan q_3. To do dzieła!

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$> \{q_1\}$	$\{q_1\}$	$\{q_1, q_2\}$
$\{q_2\}$	$\{q_3\}$	$\emptyset$
$* \{q_3\}$	$\emptyset$	$\emptyset$
$\{q_1, q_2\}$	$\{q_1, q_3\}$	$\{q_1, q_2\}$
$* \{q_1, q_3\}$	$\{q_1\}$	$\{q_1, q_2\}$
$* \{q_2, q_3\}$	$\{q_3\}$	$\emptyset$
$* \{q_1, q_2, q_3\}$	$\{q_1, q_3\}$	$\{q_1, q_2\}$

Możemy teraz zastosować algorytm usuwający, polega na tym, że patrzymy na wszystkie stany, do których nie da się wogóle dotrzeć i iterujemy tak długo, dopóki nie możemy niczego usunąć, a w następnym etapie usuwamy wszystkie podautomaty tworzące pętle, nieosiągalne z punktu widzenia stanu początkowego. Równie dobrze możęmy oczywiście skorzystać ze znanej nam już metody, czyli wychodząc z początkowego (przypomnienie odnośnie automatów ϵNAS, tam wychodziliśmy od ϵ domknięcia stanu początkowego, tutaj nie ma takich przejść, więc zbiór ten jest równoważny zbiorowi zawierającemu tylko stan początkowy, a więc tworząc nasz DAS możemy zacząć po prostu od stanu początkowego).

W takim razie zróbmy sobie jeszcze tabelkowego DAS, z powyższego NAS, od razu optymalizując:

	0	1
$> q_1$	q_1	$q_{1,2}$
$q_{1,2}$	$q_{1,3}$	$q_{1,2}$
$*q_{1,3}$	q_1	$q_{1,2}$

No, troszkę mniejsze wyszło.

2.3.2 Dany jest NAS podać język i przekonwertować na DAS

	0	1
$> q_1$	$\{q_1\}$	$\{q_1, q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\{q_4\}$	$\{q_4\}$
$*q_4$	$\emptyset$	$\emptyset$

a) Językiem będą takie ciągi zer i jedynek, że mają jedynkę na trzecim miejscu od końca (w podpunkcie h) z pierwszego podpunktu poprzedniego podrozdziału robiliśmy takiego DAS'a z głowy, teraz można sobie porównać co wydaje się trudniejsze do wymyślenia tako z głowy, tamten DAS, czy może taki NAS i przekonwertowanie go do DAS?

b) Konwersja do DAS.

Wystartujemy ze stanu q_1 i po prostu dopisujemy stany osiągalne (no powinno wyjść 8, a topologia ze wspomnianym podpunktem również powinna być taka sama).

	0	1
$> q_1$	q_1	$q_{1,2}$
$q_{1,2}$	$q_{1,3}$	$q_{1,2,3}$
$q_{1,3}$	$q_{1,4}$	$q_{1,2,4}$
$q_{1,2,3}$	$q_{1,3,4}$	$q_{1,2,3,4}$
$*q_{1,4}$	q_1	$q_{1,2}$
$*q_{1,2,4}$	$q_{1,3}$	$q_{1,2,3}$
$*q_{1,3,4}$	$q_{1,4}$	$q_{1,2,4}$
$*q_{1,2,3,4}$	$q_{1,2,3}$	$q_{1,2,3,4}$

2.3.3 Dany jest NAS go przekonwertować na DAS

	0	1
$> q_1$	$\{q_1\}$	$\{q_1, q_2\}$
q_2	$\{q_3, q_4\}$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$
$*q_4$	$\{q_1\}$	$\{q_2\}$

Na szczęście tutaj nie musimy mówić jaki to język, po prostu konwertujemy!

	0	1
$> q_1$	q_1	$q_{1,2}$
$q_{1,2}$	$q_{1,3,4}$	$q_{1,2,3}$
$*q_{1,3,4}$	q_1	$q_{1,2,4}$
$q_{1,2,3}$	$q_{1,3,4}$	$q_{1,2,3,4}$
$*q_{1,2,4}$	$q_{1,3,4}$	$q_{1,2,3}$
$*q_{1,2,3,4}$	$q_{1,3,4}$	$q_{1,2,3,4}$

Raczej od razu powinno już zaskoczyć.

2.4 Zadania treningowe

Zgadnijcie skąd wzięte!

2.4.1 ϵNAS do DAS

	0	1	ϵ
$> q_1$	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\{q_2\}$
$*q_4$	$\{q_4\}$	$\{q_4\}$	$\emptyset$

No to jedziemy naszą metodą, jeżeli chodzi o kolumnę z $\epsilon*$, to jak widać jest ona dosyć oczywista, rozszerza się troszkę tylko przy stanach q_2 oraz q_3, stanem początkowym będzie q_1, ponieważ nie ma z niego żadnego ϵ przejścia.

	0	1
$> q_1$	q_1	$q_{1,2,3}$
$q_{1,2,3}$	$q_{1,2,3}$	$q_{1,2,3,4}$
$*q_{1,2,3,4}$	$q_{1,2,3,4}$	$q_{1,2,3,4}$

2.4.2 ϵNAS do DAS

	0	1	ϵ
$> q_1$	$\{q_2\}$	$\emptyset$	$\{q_2, q_3\}$
$*q_2$	$\emptyset$	$\{q_1, q_2\}$	$\emptyset$
q_3	$\{q_1, q_3\}$	$\emptyset$	$\emptyset$

A to ciekawa bestia wyjdzie, tutaj wychodzi na jaw to o czym wspominam prawie od początku tego rozdziału, otóż startujemy nie od stanu początkowego, ale od $\epsilon*$ stanu początkowego, a tutaj zauważmy, że będą to wszystkie stany, czyli w naszym DAS'ie, będziemy zaczynać od stanu... $q_{1,2,3}$.

	0	1
$> *q_{1,2,3}$	$q_{1,2,3}$	$q_{1,2,3}$

Super język, który akceptuje wszystko co dostanie, nawet jeśli nic nie dostanie (chodzi mi o symbol pusty). No i wychodzi na to, że komentarz z wiki też ma jakiś sens.

2.4.3 ϵNAS do DAS

	0	1	ϵ
$> q_1$	$\{q_2\}$	$\emptyset$	$\{q_2, q_3\}$
q_2	$\emptyset$	$\{q_1, q_2\}$	$\emptyset$
q_3	$\{q_1, q_3\}$	$\emptyset$	$\{q_4\}$
$*q_4$	$\{q_4\}$	$\{q_4\}$	$\emptyset$

Zaczynamy oczywiście od pierwszego wiersza, a stan będzie określał $\epsilon*$ dla stanu q_1 z $\epsilon NAS'a$, no i tutaj zauważmy, że ta $\epsilon*$ mocno spuchnie, ponieważ ze stanu q_1 możemy po zero ϵ dostać się do q_1 (no oczywista sprawa), po jednym ϵ dostać się q_2 i q_3 (żywcem z tabelki), ale po dwóch ϵ dostaniemy się do q_4, ponieważ pierwszym z q_1 do q_3 można i drugim z q_3 do q_4, stąd stan początkowy będzie taki potężny w wynikowym DAS.

	0	1
$> *q_{1,2,3,4}$	$q_{1,2,3,4}$	$q_{1,2,3,4}$

Niby takie samo jak przykład zaraz wcześniej, ale jednak do wyniku doszliśmy w troszkę inny sposób.

2.5 Test rozdziału

Jeszcze pięć minutek zastanowienia się i próby przywołania czegoś z odmętów pamięci!

Spróbuj własnymi słowami wyjaśnić:

- Za co odpowiada człon „deterministyczny" w definicji DAS,
- Który z automatów wydaje się łatwiejszy do zdefiniowania dla człowieka,
- Który z automatów wydaje się szybszy do zaimplementowania,
- Dlaczego bierzemy pod uwagę ϵ przejścia w automatach niedeterministycznych.
- Po co tworzyliśmy $\epsilon*$ w niektórych zadaniach.

Spróbuj przypomnieć sobie gdzie wystąpiły pojęcia takie jak:

- Język,
- Łańcuch,
- Akceptowalność,
- Stan.

3 Ćwiczenia III

3.1 Rozgrzewka

Poznajemy kolejny sposób na opis gramatyk regularnych, tym razem mieliśmy właściwie z nim do czynienia w odległej przeszłości pierwszego i drugiego roku studiów, a więc z nowości przedstawimy tylko że da się to połączyć z dotychczasowymi sposobami!
Zanim przejdziemy do zadań, spróbuj odpowiedzieć na pytanie z czym właściwie kojarzą Ci się pojęcia, czego dotyczą?:

- regularność,
- req-exp,
- Pattern i Matcher,
- like/similar to.

Czy wszystkie oznaczają to samo? Może jesteś w stanie wskazać jakieś różnice?.

3.2 Zadania z ćwiczeń

Nie ma tutaj rozwiązań na sto procent ze wszystkich zadań, niektóre pozostają niestety do dyskusji lub znalezienia lepsze rozwiązania, a niektóre z tych z Hopcrofta z pełnymi rozwiązaniami to by się zwyczajnie nie zmieściły!

3.2.1 Hopcroft zadanie z urnami

a) Wersja przesunięcie po przejściu
...

b) Wersja przesunięcie przed przejściem
...

3.2.2 Hopcroft podaj DAS, który akceptuje języki nad alfabetem $\{0, 1\}$

Te zadania świetnie rozwiązuje się wyrażeniami regularnymi, ponieważ DAS'y są dosyć mozolne.
a) Zbiór łańcuchów, w których każdy blok pięciu kolejnych symboli zawiera przynajmniej dwa zera.

Zadanie to rozumiem w ten sposób, że łapiąc dowolną piątkę kolejnych symboli, dwa z nich muszą być zerem, będziemy zatem działać w ten sposób, że utworzymy sobie stan martwy, w którym automat znajdzie się po zbadaniu bloku niespełniającego warunku, a sieć stanów pozostałych musi być w taki sposób budowana, aby przeskakiwać do stanu akceptującego lub nie w zależności od ostatnio zbadanych pięciu symboli.

	0	1
$> q_s$	q_0	q_1
q_0	q_{00}	q_{01}
q_1	q_{10}	q_{11}
q_{00}	q_{000}	q_{001}
q_{01}	q_{010}	q_{011}
q_{10}	q_{100}	q_{101}
q_{11}	q_{110}	q_{111}
q_{000}	q_{0000}	q_{0001}
q_{001}	q_{0010}	q_{0011}
q_{010}	q_{0100}	q_{0101}
q_{011}	q_{0110}	q_{0111}
q_{100}	q_{1000}	q_{1001}
q_{101}	q_{1010}	q_{1011}
q_{110}	q_{1100}	q_{1101}
q_{111}	q_{1110}	$q_{\emptyset}$
q_{0000}	q_{00000}	q_{00001}
q_{0001}	q_{00010}	q_{00011}
q_{0010}	q_{00100}	q_{00101}
q_{0011}	q_{00110}	q_{00111}
q_{0100}	q_{01000}	q_{01001}
q_{0101}	q_{01010}	q_{01011}
q_{0110}	q_{01100}	q_{01101}
q_{0111}	q_{01110}	q_{01111}
q_{1000}	q_{10000}	q_{10001}
q_{1001}	q_{10010}	q_{10011}
q_{1010}	q_{10100}	q_{10101}
q_{1011}	q_{10110}	q_{10111}
q_{1100}	q_{11000}	q_{11001}
q_{1101}	q_{11010}	q_{11011}
q_{1110}	q_{11100}	$q_{\emptyset}$

Aż ciężko całość pomieścić, dlatego omówię teraz te część, która już istnieje, jest to podautomat odpowiedzialny za sprawdzenie pierwszych pięciu wczytywanych symboli, jak widać już tutaj może nastąpić sytuacja, gdy przy próbie wczytania czterech jedynek na pięć pierwszych symboli trafiamy do stanu martwego, ciąg dalszy tabelki poniżej:

	0	1
$*q_{00000}$	q_{00000}	q_{00001}
$*q_{00001}$	q_{00010}	q_{00011}
$*q_{00010}$	q_{00100}	q_{00101}
$*q_{00011}$	q_{00110}	q_{00111}
$*q_{00100}$	q_{01000}	q_{01001}
$*q_{00101}$	q_{01010}	q_{01011}
$*q_{00110}$	q_{01100}	q_{01101}
$*q_{00111}$	q_{01110}	$q_{\emptyset}$
$*q_{01000}$	q_{10000}	q_{10001}
$*q_{01001}$	q_{10010}	q_{10011}
$*q_{01010}$	q_{10100}	q_{10101}
$*q_{01011}$	q_{10110}	$q_{\emptyset}$
$*q_{01100}$	q_{11000}	q_{11001}
$*q_{01101}$	q_{11010}	$q_{\emptyset}$
$*q_{01110}$	q_{11100}	$q_{\emptyset}$
$*q_{10000}$	q_{00000}	q_{00001}
$*q_{10001}$	q_{00010}	q_{00011}
$*q_{10010}$	q_{00100}	q_{00101}
$*q_{10011}$	q_{00110}	q_{00111}
$*q_{10100}$	q_{01000}	q_{01001}
$*q_{10101}$	q_{01010}	q_{01011}
$*q_{10110}$	q_{01100}	q_{01101}
$*q_{10111}$	q_{01110}	$q_{\emptyset}$
$*q_{11000}$	q_{10000}	q_{10001}
$*q_{11001}$	q_{10010}	q_{10011}
$*q_{11010}$	q_{10010}	q_{10011}
$*q_{11011}$	q_{10110}	q_{10111}
$*q_{11100}$	q_{11000}	q_{11001}
$q_{\emptyset}$	$q_{\emptyset}$	$q_{\emptyset}$

No niezły automat z tego wyszedł... Drugą część tabelki można skomentować w taki sposób, że mamy tam stany akceptujące i możemy po nich przeskakiwać, przy czym istnieje kilka ślepych uliczek, o których musimy pamiętać, np. jeśli jesteśmy w stanie q_{10111} to wczytanie jedynki sprawi, że nasz cały łańcuch już nigdy nie spełni warunku, więc śmiało wpadamy w stan martwy.

b) Zbiór łańcuchów, których dziesiątym symbolem od końca jest 1.

No dobra, nie jestem aż tak pracowity człowiekiem, żeby klepać grafy albo wiersze niewiadomo jak wielkie... Jak do tej pory nie kazał nikomu tego zadanka na tablicy rozpisywać, jeżeli ktoś chce załapać zasadę, odsyłam do zadania z jedynką na trzecim miejscu od końca z poprzedniego rozdziału, natomiast powinniśmy określić ile stanów musiałbym mieć DAS, który coś takiego rozwiązuje (minimalnie!).

Pamiętamy, że dla jedynki na trzecim miejscu od prawej to trzeba było minimalnie osiem stanów, dlatego że musieliśmy zawsze pamiętać trzy ostatnio wczytane znaki i w zależności od tego co było wczytane wiedzieć gdzie przeskoczymy. Cztery stany były akceptujące. Zastanówmy się jakby to było gdybyśmy szukali jedynki na przedostatnim miejscu. Wystarczy wtedy cztery stany, ponieważ musimy pamiętać dwa ostatnio wczytane znaki. A jeżeli chodzi o automat, który rozpoznaje ostatni symbol jako jedynkę, no to potrzeba dwóch stanów. Szykuje nam się tutaj fajny ciąg. W myśl tej zasady taki automat, który złapie łańcuchy z jedynką na dziesiątym miejscu od końca potrzebuje minimalnie 2^{10} stanów.

c) Zbiór łańcuchów, które zaczynają się lub kończą sekwencją 01.

Wyrażenie regularne wydaje się dosyć oczywiste: $(01(0+1)*) + ((0+1)*01)$. A jak będzie wyglądał nasz automat? Rozpoznamy sobie sytuację, gdy słowo zacznie się od 01 i wprowadzimy go w pewny stan

akceptujący co by się tam dalej nie znalazło, natomiast każde odejście od tej zasady będzie sprowadzał się podautomatu, gdzie będziemy badać ostatnie dwa symbole.

	0	1
$> A$	B	C
B	E	D
$*D$	D	D
C	E	C
E	E	F
$*F$	E	C

3.2.3 Dane są języki $L = \{0, 10, 111, 001\}$, $M = \{\epsilon, 1, 01, 10\}$

a) Wyznacz L^0, L^1, L^2

$$L^0 = \{\epsilon\}$$

$$L^1 = \{0, 10, 111, 001\}$$

$$L^2 = \{00, 010, 0111, 0001, 100, 1010, 10111, 10001, 1110, 11110, 111111, 111001, 0010, 00110, 001111, 001001\}$$

b) Wyznacz M^0, M^1, M^2, M^3

$$M^0 = \{\epsilon\}$$

$$M^1 = \{\epsilon, 1, 01, 10\}$$

$$M^2 = \{\epsilon, 1, 01, 10, 11, 101, 110, 011, 0101, 0110, 1001, 1010\}$$

$$M^3 = \{\epsilon, 1, 01, 10, 11, 101, 110, 011, 0101, 0110, 1001, 1010, 1101, 1011, 10101, 10110, 11001, 11010, itd.\}$$

c) Wyznacz L^0M^0, L^1M^1, L^2M^2

$$L^0M^0 = \{\epsilon\}$$

$$L^1M^1 = \{0, 01, 001, 010, 10, 101, 1001, 1010, 111, 1111, 11101, 11110, 0011, 00101, 00110\}$$

$$L^2M^2 = \text{każdy z } L^2 \text{ z każdym z } M^2 \text{ ...}$$

3.2.4 Jaki językiem będzie L^* dla:

a) $L = \{a, b\}$

Będzie to dowolny ciąg a lub b o dowolnej długości i wyraz pusty.

b) $L = \{a, bb\}$

Będzie to dowolny ciąg złożony z dowolnych kombinacji podciągów bb oraz dowolnych kombinacji podciągów a, w dowolnym ustawieniu oraz symbol pusty.

3.2.5 Ile elementów posiada język L^i, jeśli

a) $L = \{a, b\}$

Nie mamy tutaj do czynienia ze „znikaniem" symboli ze względu na dziwne powtórki, w alfabecie nie ma też symbolu pustego, dlatego możemy korzystać ze wzoru $|L^k| = n^k$, gdzie n jest ilością symboli w alfabecie, więc ostatecznie mamy $|L^i| = 2^i$

b) $L = \{a, b, c\}$

Zasady jak wyżej, stąd: $|L^i| = 3^i$

c) $L = \{1, 2, ..., n\}$

I znów jak wyżej. $|L^i| = n^i$

3.2.6 Czy języki L^i z poprzedniego zadania są skończone? A L^*?

Języki L^i z poprzedniego zadania są skończone, ponieważ jesteśmy w stanie określić liczbę słów na podstawie wzoru, pod warunkiem, że znamy i. Jeżeli chodzi natomiast o L, to będzie on przeliczalny, dlatego że L^* stanowi sumę teoriomonogościową wszystkich L^i, dla $i = 0, 1, 2, ...$, a więc jest ich równoważnie liczb naturalnych, a więc i słów będzie równoważnie liczb naturalnych.

3.2.7 Dany jest język L będący zbiorem wszystkich łańcuchów złożonych z 'a' (w tym z łańcucha zawierającego zero symboli 'a'

a) Czy język ten jest skończony?

Ten język jest przeliczalny, czyli przeliczalność jego słów jest równoważna przeliczalności liczb naturalnych. Nie jest więc językiem skończonym.

b) Jakim językiem będzie L*?

Tym samym językiem? Jeżeli dobrze rozumiem treść, to zaczynamy traktować L, jakby był alfabetem, ale alfabet musi być skończony, więc albo bierzemy alfabet z języka L (czyli symbol a) i mamy wtedy domknięcie, które jest tym samym albo nie możemy wykonać zadania, ze względu na nieskończoność alfabetu. Ale nie mam stuprocentowej pewności.

3.2.8 Podaj przykład języka regularnego dla którego L^* nie jest nieskończone

A co powiecie np. na dość ubogi język $\{\epsilon\}$? Możemy je w kolejnych potęgach sklejać ze sobą, ale to nie będzie nam dawało nic nowego.

3.2.9 Zapisać wyrażenie regularne reprezentujące język:

a) będący zbiorem łańcuchów zaczynających się od '1' po którym następuje jedno lub więcej '0' i kończących się na '1'

$100 * 1$

b) będący zbiorem wszystkich łańcuchów w których na zmianę występuje '1' i '0'

$(01 + 101)(01) * +(10 + 010)(10)*$ to taka mocno rozbudowana wersja, bo nie pamiętam czy w końcu zgodziliśmy się, że powinno ich być parzyście czy nie musi być i czy dozwolone są pojedyncze symbole (ja

założyłem, że musi być przynajmniej jedno zero i przynajmniej jedna jedynka).

c) będący zbiorem łańcuchów stanowiących literały całkowitoliczbowe ósemkowe i szesnastkowe o dowolnej długości zapisywane zgodnie z konwencją języka Java.

$(0(1+2+3+4+5+6+7)(0+1+2+3+4+5+6+7)*)+(0x(0+1+2+3+4+5+6+7+8+9+A+B+C+D+E+F)(0+1+2+3+4+5+6+7+8+9+A+B+C+D+E+F)*)$

d) będący zbiorem łańcuchów przedstawiających numery NIP w formacie XXX-XXX-XX-XX lub XXX-XX-XX-XXX (gdzie X oznacza dowolną cyfrę 0-9).

$XXX-XX((X-XX-XX)+(-XX-XXX))$

e) będący zbiorem wszystkich łańcuchów nad alfabetem $\{a,b,c\}$ zawierającym co najmniej jedno a i jedno b.

$((a+b+c)*a(a+b+c)*b(a+b+c)*)+(a+b+c)*b(a+b+c)*a(a+b+c)*$

f) będący zbiorem wszystkich łańcuchów nad alfabetem $\{0,1\}$ w których jedynka jest na ósmej pozycji od prawej.

$(0+1)*1(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)$

3.2.10 Zakładając, że alfabet $\Sigma = \{0,1\}$ określ jakie języki reprezentują poszczególne wyrażenia regularne

Jeśli w wyrażeniu regularnym pojawia się znak Σ, rozumiemy go jako wystąpienie dowolnego ze znaków alfabetu.
a) 0^*10^*

Mamy do czynienia z ciągami zer i jedynek gdzie występuje tylko jedna jedynka.

b) $\Sigma^*1\Sigma^*$

Czyli mamy tutaj dowolny ciąg, który zawiera przynajmniej jedną jedynkę.

c) $\Sigma^*001\Sigma^*$

Czyli mamy tutaj jeden z naszych ulubionych języków, czyli język w którym występuje podciąg '001'.

d) $(\Sigma\Sigma)^*$

Dowolną ilość ciągów o parzystej ilości znaków wraz z symbolem ϵ.

e) $\Sigma\Sigma^*$

Ciąg o niezerowej długości, ale składający się totalnie dowolnie z zer i jedynek (też ciągi że samo zero i sama jedynka).

f) $\Sigma^*\Sigma^*$

Dowolny ciąg zer i jedynek i ϵ.

g) $01 + 10$

Językiem jest zbiór: $\{01, 10\}$.

h) $1 + \emptyset$ gdzie $\emptyset$ to zbiór pusty

Językiem jest zbiór: $\{1\}$.

i) $1\emptyset$

Językiem to $\emptyset$. No bo taki jest wynik każdego iloczynu kartezjańskiego ze zbiorem pustym, a przecież tym jest właśnie konkatenowanie.

j) $\emptyset^*$

Językiem będzie $\{\epsilon\}$. Bo „Domknięciem Kleene'ego zbioru pustego jest zbiór zawierający słowo puste".

3.2.11 Zamień wyrażenie regularne na ϵNAS

a) $01 * 0$

Ja budowanie zaczynam sobie od lewej do prawej, ale generalnie lepiej jest to robić z wewnątrz i dobudowywać. W tym przypadku zaczniemy sobie od $1*$ (przypominam że $*$ ma najwyższy priorytet), do niego dorzucimy z lewej strony mini automacik z 0 i z prawej mini automacik z 0, wszystko zlepiając tak jak wyrażenia regularne lubią najbardziej - ϵ'em.

Dla tego i następnego przykładu przygotuję kilka kroków, później będzie już właściwie treść, wynik.

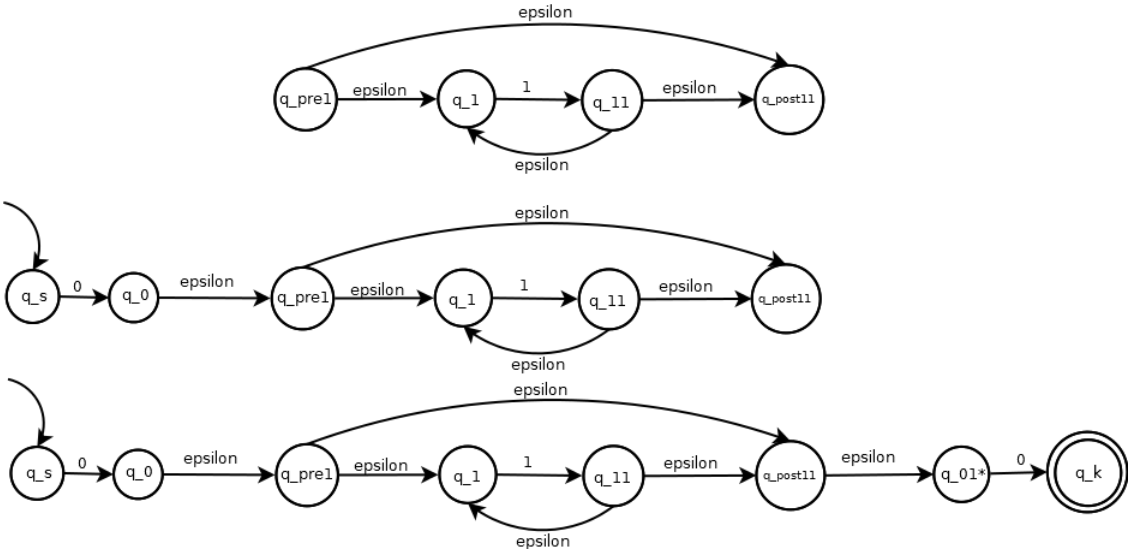
Zaczynamy więc od samego środka (to będzie ten automat na samej górze obrazka tego podpunktu), stosujemy standardowe podejście, a więc ze stanu q_1 możemy przejść do q_{11} poprzez załadowanie 1, teraz tworząc $*$ wydzielamy sobie z lewej i z prawej strony dodatkowy stan i łączymy je z odpowiednio q_{pre1} do q_1 po ϵ, natomiast q_{11} po ϵ do q_{post11}. Teraz przy pomocy ϵ łączymy sobie q_{pre1} z q_{post11} oczywiście używając ϵ. Teraz charakterystycznym punktem tej konstrukcji jest jeszcze połączenie końca mini automaciku w środku z jego początkiem (w tej kolejności), a zatem dokładamy ϵ przejście z q_{11} do q_1. To będzie koniec tego etapu, wynik to to pierwsze na obrazku.

Drugi i trzeci etap są w miarę podobne. Musimy dorzucić to zero z początku przed właśnie skonstruowany automacik, zarazem generując stan początkowy, a zatem dorzucamy stan q_s jako startowy, od niego przejście po 0 do stanu q_0. Teraz musimy sprzęgnąć dwa mikro automaciki, które już stworzyliśmy w całość, oczywiście użyjemy do tego celu ϵ, a więc łączymy nim stany q_0 oraz q_1. Trzeci etap jest bardzo analogiczny, z tym że zamiast początku zajmiemy się końcem i tak do stanu q_{01*} dostaniemy się poprzez ϵ ze stanu $q11$, natomiast z tamtego ładując 0 przechodzimy do q_k, który jest stanem akceptującym.

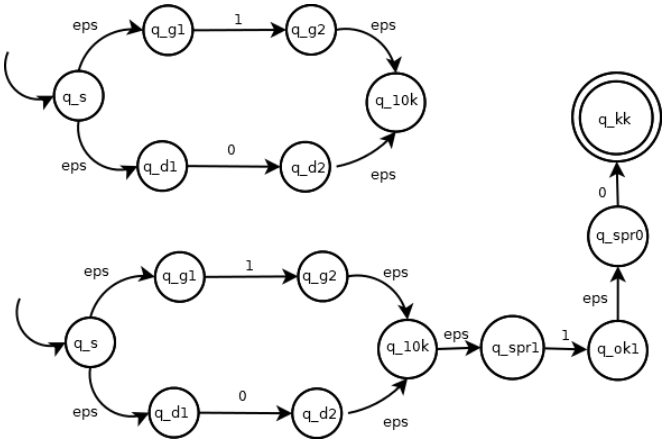
b) $(0 + 1)\,10$

W tym przypadku również zaczniemy od czegoś najbardziej zagłębionego, na szczęście tutaj będzie to też pierwsze coś od lewej. W pierwszym etapie stworzymy sobie zatem mini automacik rozpoznający 0 lub 1 w tym celu wydzielamy przed automaciki rozpoznające 0 (dam go wyżej) i ten rozpoznający 1 (dam go na dół) jeden wspólny stan q_s, od którego będziemy mogli tamte dosięgnąć ϵ przejściami. W tym przypadku będzie to też początkowy stan całego automatu. Podobny trik zastosujemy na końcu, tworząc jeden stan q_{10k}, do którego tamte obie gałęzie „spływają".

W drugim i zarazem końcowym etapie pamiętamy o tym, aby od stanu q_{10k} wyznaczyć sobie ϵ przejście do stanu q_{spr1} z niego poprzez 1 do stanu q_{ok1}, a na koniec sprzęgamy to z ostatnią cegiełką, a więc tworzymy ϵ przejście do q_{spr0} a z niego przejście poprzez 0 do q_{kk}, który będzie akceptujący.



Rysunek 1: Od góry wyniki po etapie pierwszy, drugim oraz końcowy automat po etapie trzecim dla wyrażenia $01 * 0$



Rysunek 2: Na górze wynik etapu pierwszego, poniżej końcowy automat dla wyrażenia $(0 + 1)\,10$

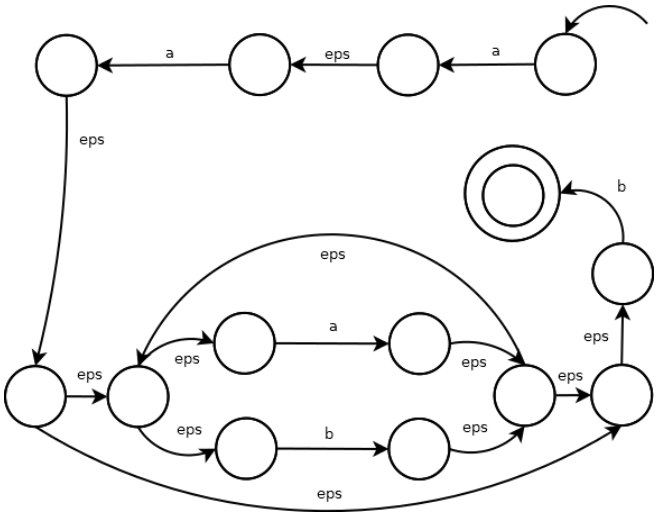
c) $aa\,(a + b) * b$

Może tym razem będę już darował sobie przydługawe wywody i po prostu umieszczę dobry graf jako odpowiedź. Od razu również zdecydowanie zrezygnuję z nazywania stanów (tworzy się ich zwyczajnie zbyt wiele...) i po prostu po samych przejściach będziemy domyślać się co, gdzie jak, oczywiście oficjalnie będziemy też musieli pamiętać o nazwach...

Jak widać najwięcej zachodu mamy oczywiście z tymi gwiazdeczkami, ale prawdziwe monstra powstają gdy łączy się je jeszcze z plusikami. Bardzo istotne jest, żebyśmy w trakcie tworzenia przedwcześnie nie zmazali na przykład ϵ przejść wiążących poszczególne mini automaciki, ponieważ takiego typu cuda robimy już po utworzeniu, co może się naprawdę przydać w kolejnym zadaniu.

3.2.12 Przekonwertować ϵNAS'y z poprzedniego zadania do DAS'ów

a) Ten jeszcze był w miarę sensowny, dlatego spróbujemy przekonwertować go takiego jaki jest teraz! Ale najpierw przetwórzmy sobie graf do tabelki.



Rysunek 3: Automat dla $aa\,(a+b)*b$

	0	1	ϵ
$>q_s$	$\{q_0\}$	$\emptyset$	$\emptyset$
q_0	$\emptyset$	$\emptyset$	$\{q_{pre1}\}$
q_{pre1}	$\emptyset$	$\emptyset$	$\{q_{post11}, q_1\}$
q_1	$\emptyset$	$\{q_{11}\}$	$\emptyset$
q_{11}	$\emptyset$	$\emptyset$	$\{q_1, q_{post11}\}$
q_{post11}	$\emptyset$	$\emptyset$	$\{q_{01*}\}$
q_{01*}	$\{q_k\}$	$\emptyset$	$\emptyset$
$*q_k$	$\emptyset$	$\emptyset$	$\emptyset$

I bierzmy się za konwersję, pamiętając o tym, żeby złapać wszystkie stany przejścia w kolumnę $\epsilon*$, zresztą możemy to dodać (jeżeli nie pamiętacie skąd bierze się taka ogromna $\epsilon*$ np. w wierszu drugim to wyjaśnienie znajduje się w poprzednim rozdziale):

	0	1	ϵ	$\epsilon*$
$>q_s$	$\{q_0\}$	$\emptyset$	$\emptyset$	$\{q_s\}$
q_0	$\emptyset$	$\emptyset$	$\{q_{pre1}\}$	$\{q_0, q_{pre1}, q_{post11}, q_1, q_{01*}\}$
q_{pre1}	$\emptyset$	$\emptyset$	$\{q_{post11}, q_1\}$	$\{q_{pre1}, q_{post11}, q_1, q_{01*}\}$
q_1	$\emptyset$	$\{q_{11}\}$	$\emptyset$	$\{q_1\}$
q_{11}	$\emptyset$	$\emptyset$	$\{q_1, q_{post11}\}$	$\{q_{11}, q_1, q_{post11}, q_{01*}\}$
q_{post11}	$\emptyset$	$\emptyset$	$\{q_{01*}\}$	$\{q_{post11}, q_{01*}\}$
q_{01*}	$\{q_k\}$	$\emptyset$	$\emptyset$	$\{q_{01*}\}$
$*q_k$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_k\}$

Na tej podstawie możemy już stworzyć automat DAS:

	0	1
$>q_s$	$q_{0,pre1,post11,1,01*}$	$q_\emptyset$
$q_{0,pre1,post11,1,01*}$	q_k	$q_{11,1,post11,01*}$
$q_{11,1,post11,01*}$	q_k	$q_{11,1,post11,01*}$
$*q_k$	$q_\emptyset$	$q_\emptyset$
$q_\emptyset$	$q_\emptyset$	$q_\emptyset$

Co jest dosyć oczywiste, w końcu jak tak popatrzymy na tego DAS'a, to widzimy, że wystartujemy dopiero gdy wejdzie jakieś 0, wtedy możemy albo ładując od razu albo po załadowaniu dowolnej ilości 1 wskoczyć poprzez jedno 0 do akceptującego, ale załadowanie jakiegokolwiek symbolu więcej wrzuca nas do stanu martwego.

b) Przykłady zaczynają się rozbudowywać, a więc i ja zacznę swoje brudne sztuczki. Już poprzednio dało się zauważyć, że niektóre ϵ przejścia niczego nie wnosiły a tylko tak to wszystko trochę zaszumiały... Dlatego możemy bez strachu odrzucić np. te przejścia, które łączą czyste konkatenacje, podobnie epsilon przejście łączące alternatywę z konkatenacją, jak również wyjściowy alternatywy, ale wejściowy alternatywy musimy pozostawić! Składając te wszystkie uproszczenia, dostaniemy automat w formie tabelki jako:

	0	1	ϵ
$> q_s$	$\emptyset$	$\emptyset$	$\{q_{g1}, q_{d1}\}$
q_{g1}	$\emptyset$	$\{q_{spr1}\}$	$\emptyset$
q_{d1}	$\{q_{spr1}\}$	$\emptyset$	$\emptyset$
q_{spr1}	$\emptyset$	$\{q_{ok1}\}$	$\emptyset$
q_{ok1}	$\{q_{kk}\}$	$\emptyset$	$\emptyset$
$*q_{kk}$	$\emptyset$	$\emptyset$	$\emptyset$

Przekonwertowanie do DAS będzie już bardzo oczywiste, po pokonaniu pierwszego stanu. Nie będę rozpisywał $\epsilon*$, bo generalnie wprowadza cokolwiek tylko tworząc pierwsze przejście. A co się tutaj stanie? Ano będziemy mieli stan startowy, który będą tworzyły stary startowy i sklejone q_{g1} i q_{d1}, od których będziemy mogli przejść do q_spr1 zarówno po 0 jak i 1. Reszta jest już raczej oczywista.

	0	1
$> q_{s,g1,d1}$	q_{spr1}	q_{spr1}
q_{spr1}	$q_\emptyset$	q_{ok1}
q_{ok1}	q_{kk}	$q_\emptyset$
$*q_{kk}$	$q_\emptyset$	$q_\emptyset$
$q_\emptyset$	$q_\emptyset$	$q_\emptyset$

No prawda jest taka, że takiego DAS'a, to by na piechotę można wymyśleć, nawet jest szybciej! No ale działamy!
c) No to teraz tego potwora... Skorzystamy ze wszystkich możliwych sztuczek. Numery stanów będą pojawiały się w takiej kolejności w jakiej występują na grafie licząc od początku.

	a	b	ϵ	$\epsilon*$
$> q_1$	$\{q_2\}$	$\emptyset$	$\emptyset$	$\{q_1\}$
q_2	$\{q_4\}$	$\emptyset$	$\emptyset$	$\{q_2\}$
q_4	$\{q_9\}$	$\{q_{10}\}$	$\{q_{13}\}$	$\{q_4, q_{13}\}$
q_9	$\emptyset$	$\emptyset$	$\{q_4, q_{13}\}$	$\{q_9, q_4, q_{13}\}$
q_{10}	$\emptyset$	$\emptyset$	$\{q_4, q_{13}\}$	$\{q_{10}, q_4, q_{13}\}$
q_{13}	$\emptyset$	$\{q_k\}$	$\emptyset$	$\{q_{13}\}$
$*q_k$	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_k\}$

Mam nadzieję, że mniej więcej da się zidentyfikować stany, q_9 oznacza stan po załadowaniu a w górnej odnodze alternatywy, q_10 to ten zaraz pod nim, stan q_{13}, to stan przedostatni, a q_k to q_{14}, ale wydaje mi się że q_k lepiej oddaje jego charakter. Ostatecznie wyszłoby coś takiego, chyba da się jeszcze mocniej zminimalizować, ale nie chciałbym żeby zatarło się całkowicie pochodzenie tego pięknego automatu.

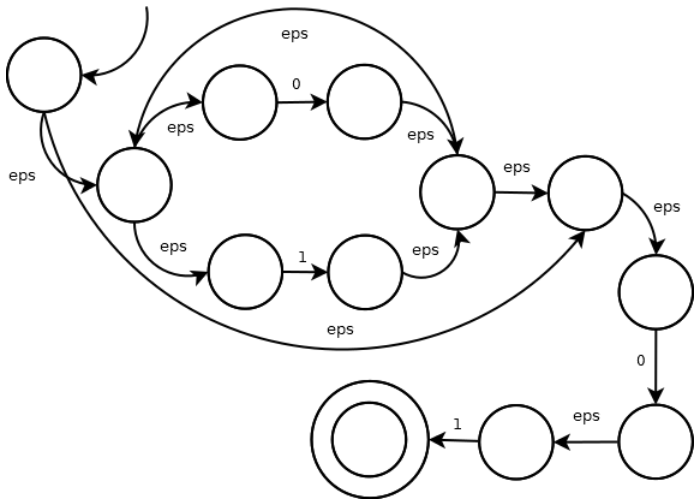
	a	b
$> q_1$	q_2	$q_\emptyset$
q_2	$q_{4,13}$	$q_\emptyset$
$q_{4,13}$	$q_{9,4,13}$	$q_{10,4,13,k}$
$q_{9,4,13}$	$q_{9,4,13}$	$q_{10,4,13,k}$
$*q_{10,4,13,k}$	$q_{9,4,13}$	$q_{10,4,13,k}$
$q_\emptyset$	$q_\emptyset$	$q_\emptyset$

3.3 Zadania treningowe

Przekonwertuj podane wyrażenia regularne do ϵNAS'ów w postaci grafów

3.3.1 $(0+1)*01$

Raczej powinno pójść gładko, musimy zacząć od stworzenia mini automaciku $0+1$, a więc już sześć stanów, następnie robimy z niego gwiazdkę, więc nowy stan przed (zarazem początkowy), nowy stan za, spinamy je ϵ przejściami, dodajemy między nowo dodanymi ϵ przejście do przodu i dodajemy cofanie na krańcach mini automaciku z plusem. Teraz na aktualnym wyjściu całości dopinamy przy użyciu ϵ przejścia automacik rozpoznający kolejno 0 i kolejny rozpoznający 1, będący jednocześnie stanem akceptującym.



Rysunek 4: Automat dla $(0+1)*01$

Raczej powinno się zgadzać, znów nie będę na diagramach nazywał stanów, tworzy to niepotrzebny w tym zadaniu szum, a wszystko jest teraz bardziej przejrzyste.

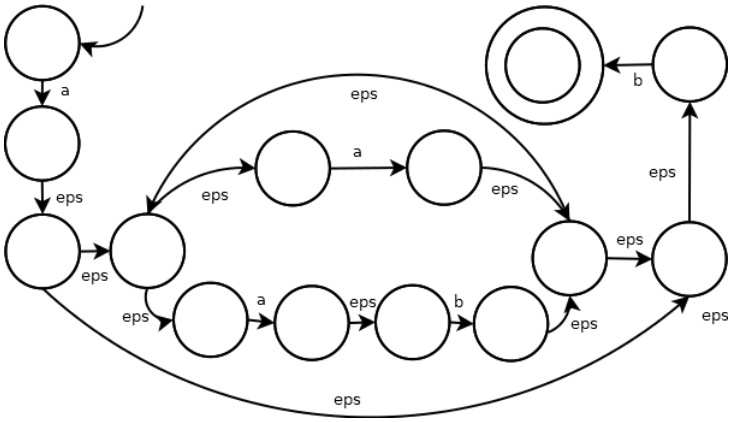
3.3.2 $a\,(ab+a)*b$

Mamy tutaj właściwie ten sam schemat co w zadaniu poprzednim, z tym że przed całość dorzucony zostanie jeszcze jeden mini automacik rozpoznający czy na początku było a, a w środku automaciku od plusa jedna gałąź zostanie nieco rozbudowana. (no oczywiście alfabet dla naszego automatu również trzeba będzie zmienić). Na osłodę zakończenie będzie nieco krótsze.

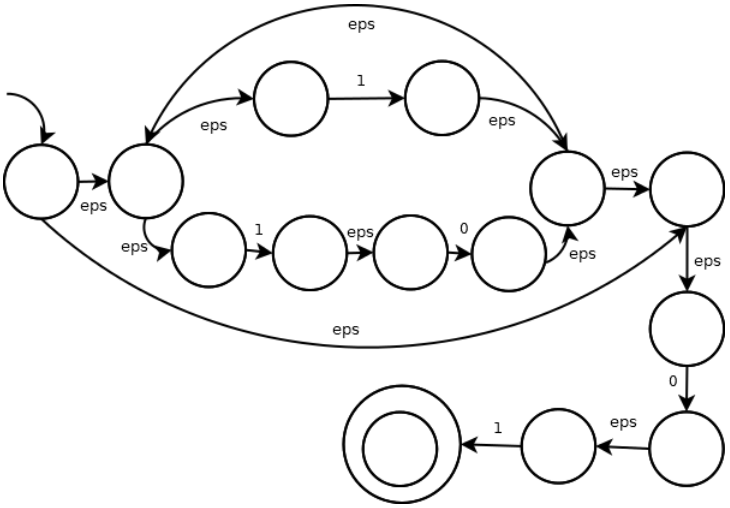
Generalnie wynikiem będzie nieco inaczej poindeksowany i troszkę zmieniony schemat z poprzedniego podpunktu.

3.3.3 $(10+1)*01$

Przykład dość podobny do poprzedniego, zresztą łatwo zauważyć że wszystkie podpunkty oprócz ostatniego będą się tutaj różnił w minimalnym stopniu.



Rysunek 5: Automat dla $a\,(ab + a) * b$



Rysunek 6: Automat dla $(10 + 1) * 01$

3.3.4 $(b + a) * ba$

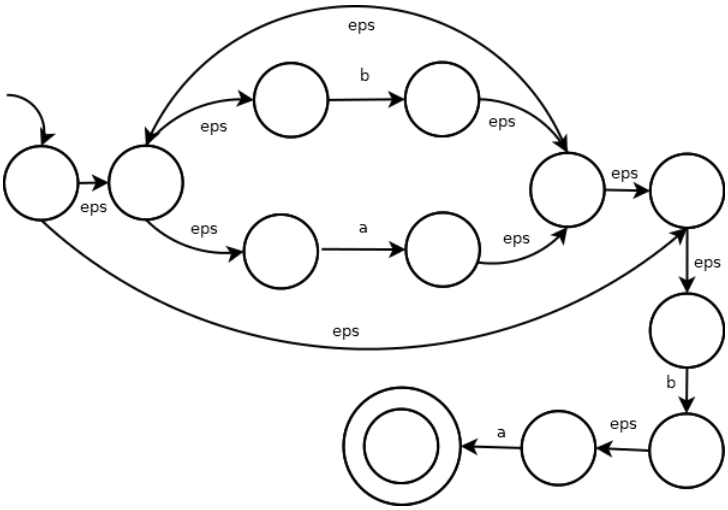
Można powiedzieć że mamy do czynienia z jeszcze łatwiejszym przykładem poprzednim. Różnicą jest tylko zawartość gałęzi w środku automatu z plusem oraz alfabet.

3.3.5 $(a + b) * ba*$

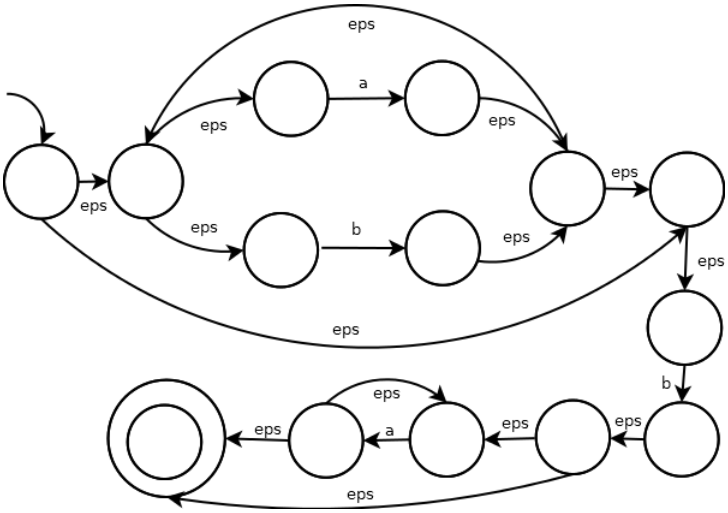
No, nareszcie coś się tutaj dzieje! Oczywiście nie spodziewajmy się fajerwerków, ale warto przypomnieć sobie w tym momencie priorytety operatorów i zastanowić się jakby to wyglądało np. beż żadnych nawiasów? (oczywiście rozwiązanie podrzucam do przykładu z nawiasem). Jedyną wartą uwagi rzeczą jest tutaj fakt, że stan akceptujący jest na końcu mini automaciku tworzącego gwiazdkę, co powinno być dość oczywiste, chociaż ja dałem się wcześniej złapać i stworzyłem nadmiarowy stan, więc warto uważać.

3.3.6 $(0 + 1) * 0 * 1$

Troszkę wariacja z poprzednim podpunktem, warto zwrócić uwagę na samą końcówkę, ponieważ początki w topologii mają te same.



Rysunek 7: Automat dla $(b + a) * ba$



Rysunek 8: Automat dla $(a + b) * ba*$

3.3.7 $(1 + 01) * (00 + 1 + 0)$

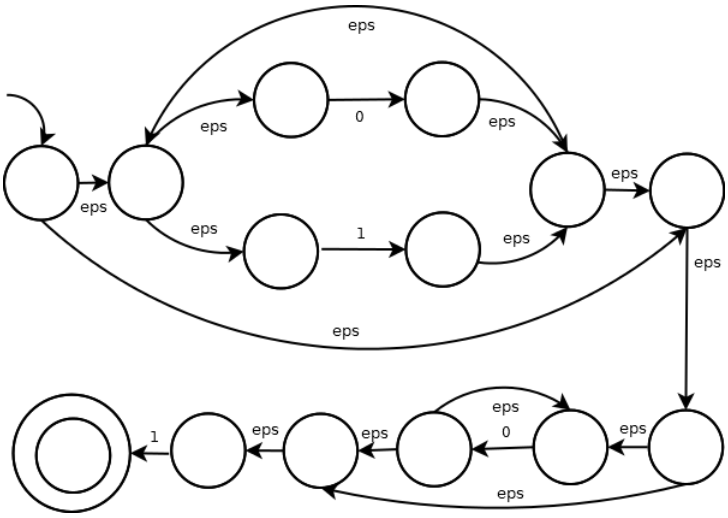
Jeżeli tworzymy zgodnie z procedurą z książki istotne jest, aby nie skrócić sobie i nie zrobić trzech ϵ przejść przy konstrukcji z dwoma plusikami obok siebie. Oczywiście działałoby tak samo, ale jeżeli mamy tworzyć zgodnie z procedurą, to musimy w myślach dodać dodatkowe nawiasowanie, także właściwie utworzyliśmy automat $(1 + 01) * ((00 + 1) + 0)$.

3.4 Test rozdziału

Czyli jeszcze dwie trzy minutki, żeby zastanowić się nad tym co właśnie robiliśmy.

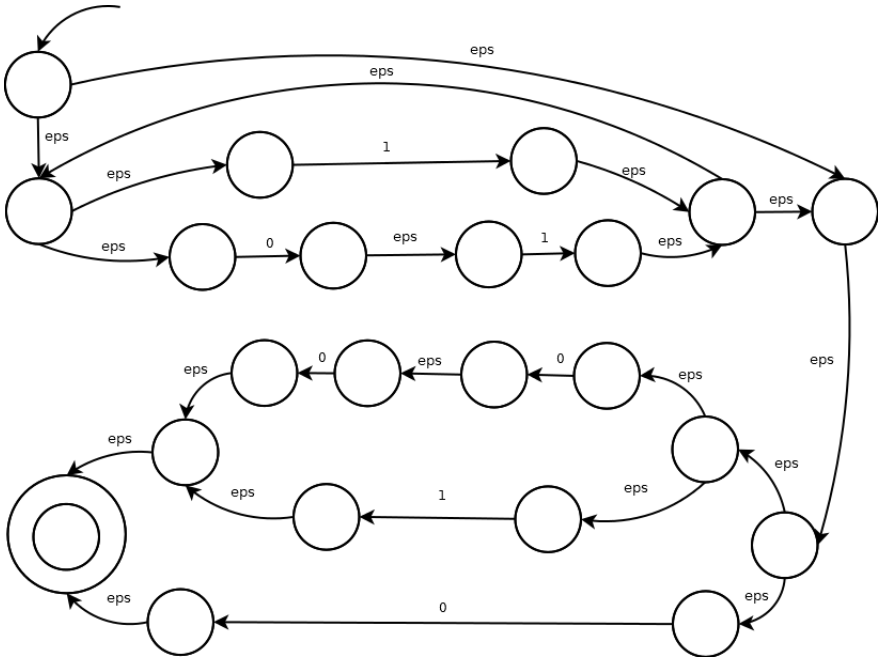
Zastanów się czy:

- Dla każdego języka, który opisaliśmy sobie wyrażeniem regularnym jesteśmy w stanie utworzyć automat deterministyczny?,
- Czy z każdego automatu deterministycznego, po wprowadzeniu kilku niezbędnych zmian bylibyśmy w stanie utworzyć wyrażenie regularne?,



Rysunek 9: Automat dla $(0 + 1) * 0 * 1$

- Które przejście okazuje się łatwiejsze dla człowieka, a które dla komputera,
- Czy w automacie utworzonym z wyrażenia regularnego zgodnie z procedurą z książki, do stanu startowego może wchodzić strzałka od innego stanu?
- Czy w automacie utworzonym z wyrażenia regularnego zgodnie z procedurą z książki, ze stanu końcowego może wychodzić jakaś strzałka?
- Trudniej przetwarza się w automat operację + czy $*$?



Rysunek 10: Automat dla $(1 + 01) * (00 + 1 + 0)$

4 Ćwiczenia IV

4.1 Rozgrzewka

Zanim przejdziemy do właściwych zadań zastanówmy się co umiemy już do tej pory, spróbuj odpowiedź umiem/nie umiem na poniższe stwierdzenia:

- Przekonwertować automat niedeterminstyczny na automat deterministyczny,
- Utworzyć minimalny automat deterministyczny, a jeżeli mam nieminimalny to do minimalnego sprowadzić,
- Powiązać automat skończony z wyrażeniem regularnym,
- Przejść z automatu skończonego na wyrażenie regularne,
- Przejść z gramatyki regularnej na wyrażenie regularne,
- Przejść z gramatyki regularnej na automat skończony i w drugą stronę.

A jeżeli jest czegoś nie umiemy, to rozwiązania w tym lub poprzednich rozdziałach czekają!

4.2 Zadania z ćwiczeń

4.2.1 Rozróżnialność i minimalizacja DAS

Mamy tutaj do czynienia z dość potężnym algorytmem, na szczęście jak już się go załapie i skorzysta z kilku sztuczek optymalizacyjnych, to okaże się że nie jest wcale taki trudny!

Generalnie najpierw od razu wyrzucimy sobie wszystkie stany nieosiągalne, po czym wypełniamy „tabelkę rozróżnialności", której warunkiem stopu jest odpowiednia ilość przejść lub zauważanie, że nic się w niej nie zmienia. Na jej podstawie wypiszemy następnie pary stanów nierozróżnialnych, z których utworzymy sobie klasy relacji rozróżnialności, aby ostatecznie utworzyć zredukowany DAS, biorąc za nowe stany właśnie te klasy. Brzmi strasznie? Na przykładzie powinno stać się oczywiste.

4.4.1)

Najpierw opiszmy automat tabelką:

	0	1
$> A$	B	A
B	A	C
C	D	B
$*D$	D	A
E	D	F
F	G	E
G	F	G
H	G	D

Wylistujemy sobie stany osiągalne ze stanu początkowego, z A możemy przejść do B lub A, z B możemy do A lub C, z C do D lub B a z D do D lub A. W związku z tym stany E, F, G oraz H są stanami nieosiągalnymi ze stanu początkowego, a zatem usuwamy je z automatu zminimalizowanego.

Teraz przechodzimy do drugiego punktu zadania czyli „tabelki rozróżnialności". W tej tabelce patrzymy sobie na pary stanów. Pierwszy będzie oznaczał kolumnę, drugi wiersz. W ten sposób jesteśmy w stanie znaleźć każdą parę. Jeżeli w którejś kratce znajdzie się znak 'X', oznacza to, że stany tworzące parę wskazującą na tę kratkę są rozróżnialne. Algorytm będzie chodził tak długo, dopóki w pełnej iteracji nie znajdzie nowych stanów rozróżnialnych. Wszystkie puste kratki będą wtedy oznaczały stany nierozróżnialne. Będziemy brali pod uwagę tylko stany znajdujące się pod przekątną, te na przekątnej są z automatu nierozróżnialne (Przykłady, w których z tego korzystamy znajdą się dopiero w zadaniach treningowych). Wróćmy do przykładu, tworzymy tabelkę tylko dla naszych 4 pozostałych stanów.

B			
C			
D			
	A	B	C

Wypełnianie tabelki będzie odbywało się iteracyjnie, najpierw wpisujemy znak 'X' w tych stanach, gdzie jeden jest akceptujący a drugi nie. Znak 'X' oznacza oczywiście rozróżnialność, w tym etapie wyznaczamy stany rozróżnialne po wczytaniu znaku ϵ. Czyli zaznaczamy 'X' w tych kratkach, gdzie jeden ze stanów z pary jest akceptujący, a drugi nie. W naszym przypadku będzie to cały najniższy wiersz.

B			
C			
D	X	X	X
	A	B	C

A teraz będziemy przechodzić po tej tabelce patrząc na pary odpowiednich stanów z wierszy. Patrzymy na wszystkie te pary, które pozostają nierozróżnialne, dlatego patrzymy sobie kolejno na (A, B),(A, C), (B, C).

Teraz gwóźdź programu. Weźmy kratkę (A, B), musimy popatrzeć do pierwotnej tabelki opisującej nasz automat, a właściwie na dwa wiersze, wiersz z przejściami ze stanu A oraz wiersz z przejściami ze stanu B. Teraz bierzemy sobie wszystkie pary pionowe (czyli będziemy sprawdzali (B, A) oraz (A, C) zliczając z lewego górnego rogu). Tutaj możecie się jeszcze pomylić bo i poziomo i pionowy wychodzi na to samo, ale zaraz zobaczymy różnicę. W każdym razie! Patrzymy sobie na kratki w tworzonej tabelce, jeżeli dla którejkolwiek z tych dwóch par wystąpi znak 'X', to znak 'X' wstawimy też w rozpatrywanej kratce, na podstawie której te pary wyciągnęliśmy. Zauważmy, że w tej iteracji ani (B, A), ani (A, C) nie prowadzą do 'X'.

Teraz wypełnijmy kratkę (A, C). Patrzymy na wiersze tabelki opisującej przejścia dla stanów A oraz C, no i mamy do sprawdzenia dwie pary z „tabelki rozróżnialności" są to: (A, D) oraz (A, B). (Przypominam o tym, że pary wykreślamy pionowo, jeżeli jeszcze jest to dla was nieoczywiste, zauważcie że ma to sens również dla automatu nad alfabetem jedno, jak i trzy, cztero itp. elementowym i algorytm będzie działał). Zauważmy, że kratka (A, D) w tablicy rozróżnialności ma już znak 'X', w takim razie śmiało wpisujemy znak 'X' w wypełnianą kratkę (A, C). Analogicznie dla kratki (B, C).

Po tej iteracji automat „tablica rozróżnialności" powinna wyglądać w ten sposób:

B			
C	X	X	
D	X	X	X
	A	B	C

Halo, halo, ale musimy przeprowadzić jeszcze jedną iterację! W końcu została nam jeszcze pusta kratka dla (A, B). No to sprawdzamy sobie znowu, pamiętamy już gdzie prowadziły nas A oraz B, a więc wiemy, że musimy sprawdzić kratki (B, A) oraz (A, C). O ile kratka (B, A) zostaje nadal pusta, to kratka (A, C) ma już 'X', a zatem 'X' wpisujemy teraz w kratkę (A, B). Teraz tabelka wypełniła się 'X'ami, a więc kolejna iteracja niczego nie zmieni, więc możemy zakończyć algorytm.

B	X		
C	X	X	
D	X	X	X
	A	B	C

Teraz mamy istotny etap tworzenia klas rozróżnialności, który tutaj, jest bardzo trywialny... Nie mamy tutaj par nierozróżnialnych, a więc wszystkie stany z „tabelki rozróżnialności" będą tworzyły nowy automat.

Polecam przerobić sobie ten przykład bez wykreślania stanów nieosiągalnych w pierwszym etapie i pozbycie się ich z gotowych klas relacji rozróżnialności, po tym jak przerobicie podpunkt 4.4.2.

4.4.2)

Wypełnianie tabelki się na szczęście nie zmieni. Dla odmiany w tym przykładzie nie ma stanów nieosiągalnych ze stanu początkowego, ale znajdziemy kilka fajnych stanów nierozróżnialnych, no ale do rzeczy!

	0	1
$> A$	B	E
B	C	F
$*C$	D	H
D	E	H
E	F	I
$*F$	G	B
G	H	B
H	I	C
$*I$	A	E

No to bierzmy się do roboty, najpierw rozrysujemy sobie tabelkę, pozwolę sobie narysować tabelkę już po iteracji z wypisaniem stanów rozróżnialnych przez ϵ przejścia, przypominam, że 'X' stawiamy teraz w tych kratkach, że jeden stan z pary jest akceptujący, a drugi nie (dlatego w kratkach (C, F),(C, I),(F, I) nie stawiamy znaku 'X').

B								
C	X	X						
D			X					
E			X					
F	X	X		X	X			
G			X			X		
H			X			X		
I	X	X		X	X		X	X
	A	B	C	D	E	F	G	H

Mamy to, oczywiście przypominam, że wypełniamy tylko przestrzeń pod przekątną. Teraz Patrzymy na każdą pustą kratkę zaczynając od pary (A, B) i patrzymy sobie na definicję naszego automatu. W kratkę (A, B) wstawimy 'X', ponieważ dla niego musimy sprawdzić dwie kratki: (C, B) oraz (E, F). Obie mają już znak 'X', a więc spokojnie wpisujemy 'X' w kratkę (A, B). Po sprawdzeniu wszystkich pozostałych jeszcze pustych kratek automat powinien nam wyjść taki:

B	X							
C	X	X						
D		X	X					
E	X		X	X				
F	X	X		X	X			
G		X	X		X	X		
H	X		X	X		X	X	
I	X	X		X	X		X	X
	A	B	C	D	E	F	G	H

Znowu musimy przelecieć po wszystkich stanach, ale nie zauważymy już zmiany. Oznacza to, że we wszystkich niewypełnionych kratkach wyłoniły się pary stanów nierozróżnialnych ze sobą. Teraz te pary: (A, D), (A, G), (B, E), (B, H), (C, F), (C, I), (D, G), (E, H), (F, I) musimy zebrać w klasy relacji rozróżnialności, czyli takie zbiory, które nie będą miały elementów wspólnych pomiędzy sobą, a będą w sumie zawierały wszystkie stany.

Zacznijmy tworzenie klasy od stanu A, weźmiemy pod uwagę wszystkie te pary, które zawierają stan A, następnie wszystkie te, które będą zawierały drugiego z pary dla tamtych A, następnie drugiego z pary dla każdego z drugiego z pary i tak aż nie będziemy w stanie dorzucić niczego, co już w klasie się nie znajdzie. Np. dla wspomnianego A powstaje kolejno klasa $[A, D]$ z pary pierwszej, ale patrząc na drugą możemy śmiało dorzucić tam stan G otrzymując klasę $[A, D, G]$. Zerkając na pozostałe pary nie zauważam, żebym mógł coś dodać (no para (D, G) rzuca się jeszcze w oczy, że mogłaby być, ale oba stany są już w klasie).

Analogicznie wyznaczę sobie pozostałe klasy, będą to: $[B, E, H]$ oraz $[C, F, I]$. Klasą akceptującą oznaczamy wszystkie te klasy, które zawierają przynajmniej jeden stan akceptujący, no w tym przypadku będzie to oczywiście klasa $[C, F, I]$.

Ten DAS, możemy w związku z tym zminimalizować do automatu, który będzie miał tylko trzy stany (po jednym dla każdej klasy) no i wszystko powinno dać się ładnie narysować. Polecam przetrenować przykłady z treningu lub zerknąć na ostatnie zdanie w poprzednim podpunkcie i powinno być dobrze.

4.2.2 Konwersja automatu na gramatykę regularną prawostronnie

Generalnie mamy tutaj kilka etapów. Załóżmy, że symbole nieterminalne gramatyki będziemy oznaczać tak samo jak stany automatu deterministycznego (terminalne to oczywiście możliwe przejścia pomiędzy stanami z alfabetu). W takim razie ustalamy symbol startowy jako stan startowy. Następnie dla każdego przejścia dopisujemy odpowiednią produkcję:

- Jeśli przejście ze stanu A do niego samego po symbolu a, to dopisujemy produkcję postaci $A \to aA$,
- Jeśli przejście ze stanu A do innego stanu B po symbolu a, to dopisujemy produkcję postaci $A \to aB$,
- Jeśli przejście ze stanu A do innego stanu B po symbolu ϵ, to dopisujemy produkcję postaci $A \to B$,
- Jeśli któryś stan A jest też stanem końcowym, to dopisujemy produkcję postaci $A \to \epsilon$.

No i chyba tyle!

4.2.3 Konwersja gramatyki regularnej prawostronnie na automat

Mocna analogia do poprzedniego zadania. Znów dla uproszczenia zakładam konwencję, że symbole nieterminalne gramatyki będziemy oznaczać tak samo jak stany automatu deterministycznego.

Znów symbol startowy magicznie stanie się stanem początkowym. I teraz każdą produkcję zamienimy na odpowiednie przejście lub uzakceptowanie automatu:

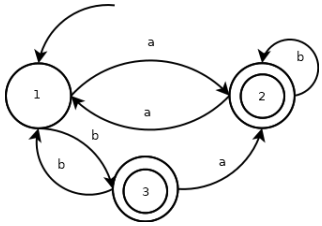
- Jeśli produkcja typu $A \to aA$, to tworzymy przejście ze stanu A do niego samego po symbolu a,
- Jeśli produkcja typu $A \to aB$, to tworzymy przejście ze stanu A do stanu B po symbolu a,
- Jeśli produkcja typu $A \to B$, to tworzymy przejście ze stanu A do stanu B po symbolu ϵ,

- Jeśli produkcja typu $A \to a$, to tworzymy przejście ze stanu A do stanu nowo stworzonego stanu *Aprim* po symbolu a, a stan *Aprim* oznaczamy jako akceptujący, ale wszystkie wyjścia ze stanu *Aprim* mają prowadzić do nieakceptującego stanu martwego,

Powinno siąść!

4.2.4 Przejście z AS do wyrażenia regularnego poprzez GNAS

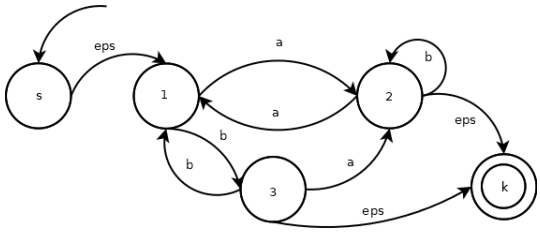
Pobawimy się na przykładzie zadania domowego.



Rysunek 11: Automat DAS z zadania domowego

Pierwszy etap jest dosyć standardowy i prosty, pamiętamy to zresztą z lekcji o wyrażeniach regularnych. Dodajemy dodatkowy stan początkowy, tak aby żaden stan do niego nie wchodził i poprowadź ϵ przejście do „starego" stanu początkowego. (UWAGA, gdybyśmy robili to super dokładnie, to musielibyśmy utworzyć i brać pod uwagę, aby do każdego stanu z każdego stanu (wyłączając wspomniane wchodzenie do początkowego) utworzyć dodatkowe przejście $\emptyset$, łącznie z pętlami (np. w naszym przypadku pętle do 1 i do 2, które nie zmieniają już kompletnie nic...) i te przejścia trzebaby brać pod uwagę na każdym etapie redukcji..., ale my podobno nie musimy tego robić (na szczęście)).

Teraz dodajemy jeszcze nowy stan końcowy, odbieramy akceptowalność, „starym" akceptującym i tworzymy ϵ przejścia do nowego końcowego.



Rysunek 12: Pierwszy etap tworzenia GNAS'a, dodanie stanu początkowego i końcowego, nazywamy go GNAS5, bo ma w tym momencie 5 stanów

Teraz, gdy mamy już porządnego GNAS'a początkowego (tego nazwiemy GNAS'em 5, ze względu na to, że mamy pięć stanów), musimy pamiętać, że na przejściach GNAS'a nie ma już symboli terminalnych, ale są wyrażenia regularne.

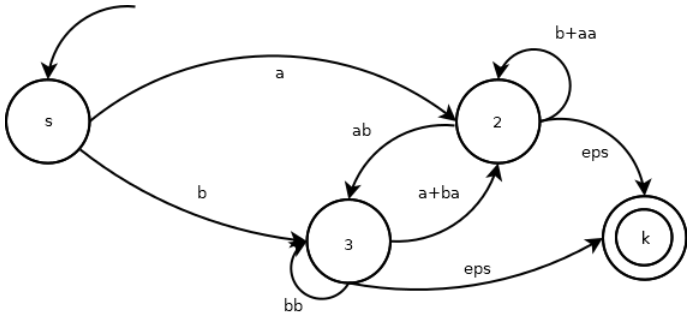
Pozbądźmy się stanu 1. Musimy dopisać do pętli które pozostaną wszystkie ścieżki, które istnieją i są nietrywialne przy istnieniu stanu 1. To oznacza, że np. utworzymy ze stanu S do stanu 2 ścieżkę z wyrażeniem a, ponieważ z wcześniej ze stanu 1 do stanu 2 można było przejść poprze a, a z S do 1 poprzez ϵ przejście.

Podobnie pojawia się ścieżka z b, znowu przed redukcją z S mogliśmy do 1 dostać się po ϵ przejściu, a z niego do 3 po przejściu b. Ciekawsze jest natomiast przejście z 2 do 3, które pojawia się jakby znikąd! Ale ma jednak logiczne uzasadnienie. Zwróćcie uwagę, że przed redukcją 1, istnieje możliwość przejścia z 2 do 1 po a, aby później przejść z 1 do 3 po b, a to jest przecież ścieżką ab, która prowadzi z 2 do 3. Dodatkowo jest jeszcze mała zmiana w przejściu z 3 do 2, ponieważ zniknęłaby ścieżka z 3 do 2 przez 1, tą też musimy zapamiętać, a zatem dodajemy po znaku + do już istniejącego przejścia a. Przypominam że na przejściach znajdują się teraz wyrażenia regularne, więc takie działanie jest dozwolone.

Pozostaje nam jeszcze dołożyć dwie dodatkowe pętle, które możemy wykonać. Do pętli stanu 2 dodajemy alternatywę aa, ponieważ możemy przejść do stanu 1 po a i spowrotem. Analogicznie dodajemy pętlę do

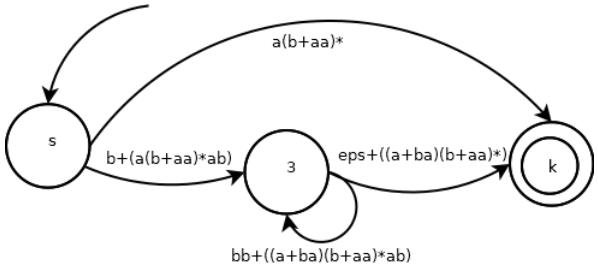
stanu 3. Uważajcie tylko, aby się nie zapędzić, ponieważ nie musimy dodawać pętli z 2 do 1 z 1 do 3 i z 3 do 2, ponieważ zostaje ona zachowana dzięki naszym nowym ścieżkom – wyrażeniom ab jeśli mamy ze stanu 3 do stanu 2 oraz $a + ba$, jeśli chodzi o przejście z 2 do 3.

Pozostałych ścieżek nie będziemy zmieniać, ponieważ nie dotyczą już redukowanego stanu 1, że tak powiem w sposób bezpośredni, tzn. one nie znikną wraz ze zniknięciem stanu 1, a to na takie patrzyliśmy w tej iteracji.



Rysunek 13: GNAS4, czyli po redukcji stanu 1

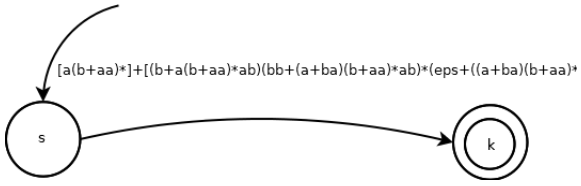
Zabawę kontynuujemy dla otrzymanego GNAS'a4. Teraz zredukujemy stan 2, a więc pojawi się bezpośrednia ścieżka z S do k, która będzie pochodziła od starej drogi z S do 2 i do k. Ważna uwaga, że jeżeli gdzieś konkatenujemy przechodząc przez stan w któym była nietrywialna pętla, to po wyrażeniu opisującym te nietrywialną pętle dajemy znak $*$. W pętli stanu 3 musimy teraz zapamiętać starą drogę z 3 do 2 do 3, a więc nieco się to rozbuduje. Dodatkowo ze stanu S do stanu 3 musimy dorzucić alternatywę, czyli starą drogę z S do 2 do 3 oraz analogicznie wzbogacić naszą ścieżkę 3 do k o składnik ze starej drogi 3 do 2 do k.



Rysunek 14: GNAS3, czyli po redukcji stanów 1 i 2

Przed nami ostatni etap. Właściwie w tym momencie możemy już nawet przestać myśleć o całym algorytmie, ponieważ gdy mamy już do czynienia z poprawny GNAS'em3, to zostaje już tylko skonkatenować ze sobą ścieżkę od S do 3 z ścieżką 3 do k, a wynik zrobić alternatywny z bezpośrednią ścieżką z S do k, która pojawiła nam się w poprzedniej iteracji, oczywiście pamiętając o tym, że pętle musimy traktować znakiem $*$.

To prowadzi nas do ostatniej postaci, czyli GNAS2, która zawiera rozwiązanie zadania, czyli odpowiadające wejściowemu automatowi skończonemu, wyrażenie regularne.



Rysunek 15: GNAS2, czyli po redukcji stanów 1,2 i 3, inaczej mówiąc wynikowy

Algorytm na pierwszy rzut oka wydaje się tajemniczy i rzeczywiście potrafi zmylić, szczególnie jak są te ścieżki w jedną i drugą stronę, ale generalnie pamiętajcie o zasadzie, że doklejamy te ścieżki, które na pewno

znikałyby wraz ze zniknięciem aktualnie redukowanego stanu. Pamiętajcie też o pierwszym etapie, który wydaje się trywialny, ale bez niego nie zajdziemy nigdzie i o tym, że czasami w grafie kryją się nietrywialne pętle, których nie widać na pierwszy rzut oka.

4.2.5 Lemat o pompowaniu

Być może pojawi się tutaj coś, jeżeli będę miał ogarnięte jakieś fajne i sensowne zadanko ogarnięte z tym tematem.

4.3 Zadania treningowe

4.3.1 Trening minimalizacji

a) Zminimalizować podany DAS:

	0	1
$> A$	B	C
$*B$	D	E
C	F	G
D	D	E
E	F	G
$*F$	D	E
G	F	G
H	D	E

Przykład jest dosyć prosty, porównajcie proszę wyniki czy wyszło Wam tak samo. Jedyną wartą uwagi rzeczą jest tutaj zachowanie się przy rozpatrywaniu kratek takich jak np. (E, G), no bo dla niego musimy sprawdzić co mamy w kratkach (F, F) oraz (G, G), leżących na przekątnej, których przecież nie wypełniamy... Tutaj warto zerknąć do wstępu do zadania 4.4.1, gdzie wspominałem, że wszystkie kratki na przekątnej od samego początku do samego końca traktujemy jako nierozróżnialne (w końcu każdy stan jest nierozróżnialny sam ze sobą). Z tą radą powinno wyjść spoko.

Stan H jest stanem nieosiągalnym, więc nie będę go rozpatrywał przy wypełnianiu tabelki. Tabelka po pierwszym przejściu.

B	X					
C		X				
D	X	X	X			
E		X		X		
F	X		X	X	X	
G		X		X		X
	A	B	C	D	E	F

Mamy aż sześć par stanów nierozróżnialnych (A, C), (A, E), (A, G), (B, F), (C, E), (C, G) i (E, G), stąd wyznaczamy sobie już łatwo klasy: $[A, C, E, G]$, $[B, F]$ i $[D]$, które określą stany w automacie zminimalizowanym.

b) Zminimalizować podany DAS:

	a	b
> 1	2	3
2	4	5
3	6	7
$*4$	4	5
5	6	7
$*6$	4	5
$*7$	6	7
8	4	5

Trochę mogą mylić oznaczenia, ale kto wie z czym przyjdzie zmierzyć się w prawdziwym boju? Standardowo wypełniamy sobie tabelkę i wypiszemy pary nierozróżnialne, zaczynając oczywiście od wyrzucenia stanu 8 jako nieosiągalnego ze stanu początkowego.

2	X					
3	X	X				
4	X	X	X			
5	X	X		X		
6	X	X	X		X	
7	X	X	X	X	X	X
	1	2	3	4	5	6

Mamy dwie pary stanów nierozróżnialnych $(3, 5)$ i $(4, 6)$, stąd wyznaczamy sobie już łatwo klasy: $[3, 5]$, $[4, 6]$, $[1]$, $[2]$ i $[7]$, które określą stany w automacie zminimalizowanym.

4.4 Test rozdziału

A więc takie pytanie podsumowujące wszystko do tej pory. Czyli bylibyśmy w stanie teraz stworzyć wyrażenie regularne opisujące jakiś język, przetworzyć to na automat skończony np. niedeterministyczny, ten na deterministyczny, zminimalizować go i wtedy hopla do gramatyki regularnej? A w druga stronę? Sprawdźcie z jakimiś prostymi językami!

5 Ćwiczenia V

Polecam znajdźcie i zmuście kogoś ogarniętego żeby ten CYK i EARLEY'a troche przygotował, chyba że wystarczą ogólnie dostępne źródła. Jeżeli ktoś chciałbym zrobić jakąś korekte tego co jest tutaj albo coś dorzucić, to napiszcie do mnie (Szymek_diks@interia.pl).

6 Dodatek A: Zadania z gramatyk

Proponuje jako formę treningu przed egzaminem/ostatnią kartkówką przemyśleć je sobie samodzielnie jako formę ćwiczenia umysłowego i porównać z tym co poniżej, oczywiście nie są to najlepsze możliwe rozwiązania, ale przykładowe (mam nadzieję poprawne).

6.1 Podaj gramatyki bezkontekstowe generujące:

6.1.1 Język $L = \left\{a^n b^m c^m d^{2n}\right\} : n \geqslant 0, m > 0$

Najpierw narobimy sobie n znaków a i $2n$ znaków d, przy czym może ich być również 0, po czym pomiędzy nie, będziemy wstawiać pary b c, przy czym tych przynajmniej musi być jedna parka.

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, Z\}$$
$$\Sigma = \{a, b, c, d\}$$
$$P = \{S \to aSdd, S \to Z, Z \to bc, Z \to bZc\}$$
$$\sigma = S$$

6.1.2 Język $L = \{a^n b^m\} : 0 \leqslant n \leqslant m \leqslant 2n$

Zastosujemy taką metodę, że na każde wygenerowane a będziemy generować jedno b i jedno potencjalne b jako nieterminalny, który może stać się b, ale nie musi, ale co ważne, musimy też uwzględnić ϵ.

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, Z\}$$
$$\Sigma = \{a, b\}$$
$$P = \{S \to aSbZ, S \to \epsilon, Z \to b, Z \to \epsilon\}$$
$$\sigma = S$$

6.1.3 Język L nad alfabetem $\Sigma = \{a, b, c\}$, zawierający wszystkie słowa, takie które zawierają co najmniej trzy symbole 'a'

Będziemy mieli cztery poziomy „ zagłębienia", tak żeby odpowiadały sytuacji, posiadania już na końcie zera, jednego, dwóch i trzech znaków 'a', przy czym tylko ostatnie mogą zakończyć

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, A, A', A'', X\}$$
$$\Sigma = \{a, b, c\}$$
$$P = \{S \to aA, S \to bS, S \to cS, A \to aA', A \to bA, A \to cA\} \cup$$
$$\{A' \to aA'', A' \to bA', A' \to cA', A'' \to aX, A'' \to bA'', c \to cA''\} \cup$$
$$\{X \to aX, X \to bX, X \to cX, X \to a, X \to b, X \to c\}$$
$$\sigma = S$$

Ewentualnie możemy przy pierwszej produkcji narzucić wystąpienie trzech znaków a, pozwalając też na wstawienie pomiędzy nimi czego sobie tylko nie zażyczymy:

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, A, B\}$$
$$\Sigma = \{a, b, c\}$$
$$P = \{S \to AaAaAaA, A \to aA, A \to bA, A \to cA, A \to \epsilon\}$$
$$\sigma = S$$

6.1.4 Język L nad alfabetem $\Sigma = \{a, b, c\}$, zawierający wszystkie słowa, takie które zaczynają się i kończą na ten sam symbol

Czyli startując wybieramy jedną ze wszystkich możliwych opcji pod względem pierwszego i ostatniego symbolu, a w środku będzie już „wolna amerykanka".

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, X\}$$
$$\Sigma = \{a, b, c\}$$
$$P = \{S \to aXa, S \to bXb, S \to cXc, X \to aX, A \to bX, A \to cX, X \to a, X \to b, X \to c\}$$
$$\sigma = S$$

6.1.5 Język L nad alfabetem $\Sigma = \{a, b, c\}$, zawierający wszystkie słowa, takie których długość jest parzysta

Zakładamy, że 0 też jest parzyste, więc musimy ze startowego stworzyć ϵ, jeśli nie to możemy dołożyć po jednym nieterminalnym z każdej strony i zawsze jak sobie dobijamy, to dzieje się to parzyście, a później ten nieterminalny zamieniamy na dowolny dostępny symbol.

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, X\}$$
$$\Sigma = \{a, b, c\}$$
$$P = \{S \to XSX, S \to \epsilon, X \to a, X \to b, X \to c\}$$
$$\sigma = S$$

6.1.6 Język L nad alfabetem $\Sigma = \{0, 1\}$, zawierający wszystkie słowa, takie których długość jest nieparzysta, a środkowy symbol to '1'

Znowu będziemy sobie nadbudowywać z obu stron, wszystkie dobudowane symbole będą miały dowolność, natomiast środkowy będziemy mogli zamienić tylko na 1.

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, X\}$$
$$\Sigma = \{0, 1\}$$
$$P = \{S \to XSX, S \to 1, X \to 0, X \to 1\}$$
$$\sigma = S$$

6.1.7 Język L nad alfabetem $\Sigma = \{0, 1\}$, zawierający wszystkie słowa, takie których ilość symboli '1' będzie większa niż ilość symboli '0'

Wystartujemy na jeden z trzech sposobów albo po prostu jedynką albo jedynką z lewej albo jedynką z prawej, później będziemy mogli rozszerzać zamieniając nieterminalny na dwa nieterminalne, ale tak, żeby z tych dwóch dało się albo dalej rozszerzać w ten sam sposób albo zmontować więcej jedynek lub równo z ilością zer, ale ze względu na start i tak będzie to spełniało warunki.

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, X\}$$
$$\Sigma = \{0, 1\}$$
$$P = \{S \to 1, S \to X1, S \to 1X, S \to X1X, X \to XX, X \to YX, X \to XY, X \to 1, Y \to 0, Y \to \epsilon\}$$
$$\sigma = S$$

6.1.8 Język L nad alfabetem $\Sigma = \{0, 1\}$, zawierający wszystkie słowa, takie których ilość symboli '1' będzie 2 razy większa niż ilość symboli '0'

Spróbujemy jednym terminalnym. W końcu istnieje tylko kilka możliwych kombinacji takich, żeby dwa razy więcej 1 niż 0 mamy. Będziemy mogli dokładać kolejne możliwe kombinacje wraz z możliwością generacji oraz kończącą produkcję do ϵ, no bo w końcu zero też jest dwa razy większe od zera...

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S\}$$
$$\Sigma = \{0, 1\}$$
$$P = \{S \to 1S10, S \to 1S01, S \to 0S11, S \to 11S0, S \to 10S1, S \to 01S1, S \to \epsilon\}$$
$$\sigma = S$$

6.1.9 Język $L = \left\{a^n b^m c^k : k = n + m\right\}$

Bardzo standardowy przykład i raczej szybko się z nim uporamy, generalnie naprodukujemy sobie najpierw tyle c co a z możliwością tworzenia dalej pomiędzy nimi i zrobimy tam tyle c co b, oczywiście zachowując odpowiednią kolejność. Generalnie dopuszczam też sytuację gdzie liczba n lub m są równe 0, ma to odpowiednie konsekwencje dla ilości znaków c.

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, X\}$$
$$\Sigma = \{a, b, c\}$$
$$P = \{S \to aSc, S \to X, X \to bXc, X \to \epsilon\}$$
$$\sigma = S$$

6.1.10 Język $L = \left\{a^n b^m c^k : k \neq n + m\right\}$

Skoro k musi być różne od sumy $n + m$ to możemy rozpatrzeć że albo symboli c będzie więcej albo mniej i takie dwa przypadki sobie rozpatrzymy. Produkcje M oraz X odpowiadają za generowanie przypadków, gdzie znaków c będzie na pewno mniej niż sumy a oraz b. Analogicznie D oraz Y odpowiadają za przypadek, gdzie znaków c na pewno będzie więcej.

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, M, X, D, Y\}$$
$$\Sigma = \{a, b, c\}$$
$$P = \{S \to aM, S \to bX, S \to Dc, M \to aM, M \to aMc, M \to X, X \to bX, X \to bXc, X \to \epsilon\} \cup$$
$$\{D \to aDc, D \to Y, Y \to bYc, Y \to Yc, Y \to \epsilon\}$$
$$\sigma = S$$

6.1.11 Język $L = \{a^n b^m c^k : n \neq m \vee m \neq k\}$

Znów rozpatrzymy dwa przypadki, jeden dla sytuacji gdy ilość znaków a jest inna niż znaków b, drugi gdy znaków b jest inna niż znaków c. Na początku mamy gramatycznego 'ifa', wybieramy jedną z tych opcji np. poprzez produkcję $S \to Xc$, wtedy możemy naprodukować dowolną ilość znaków c, po czym przechodzimy do produkowania znaków a oraz b, przy czym musimy skończyć w taki sposób, że ich ilość będzie różna. Analogicznie załatwiamy drugi przypadek.

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S, X, Y, Z, Z', Z'', C, C', C''\}$$
$$\Sigma = \{a, b, c\}$$
$$P = \{S \to Xc, S \to aY\} \cup$$
$$\{X \to Xc, X \to Z, Z \to aZb, Z \to Z', Z \to Z'', Z' \to aZ', Z' \to a, Z'' \to Z''b, Z'' \to b\} \cup$$
$$\{Y \to aY, Y \to C, C \to bCc, C \to C', C \to C'', C' \to bC', C' \to b, C'' \to C''c, C'' \to c\}$$
$$\sigma = S$$

6.1.12 Język pusty

Wydaje mi się, że będzie generowany np. przez gramatykę bez produkcji albo gramatykę bez alfabetu terminalnego lub najogólniej bez produkcji, które byłyby w stanie wygenerować jakiekolwiek zdanie.

$$G = (N, \Sigma, P, \sigma)$$
$$N = \{S\}$$
$$\Sigma = \{s\}$$
$$P = \emptyset$$
$$\sigma = S$$

6.2 Podaj gramatyki regularne generujące języki:

6.2.1 Język L nad alfabetem $\Sigma = \{0, 1\}$, składający się ze wszystkich słów, które zaczynają się symbolem '0' i kończą symbolem '1'

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{S \to 0A, A \to 0A, A \to 1A, A \to 1\}$$

6.2.2 Język L nad alfabetem $\Sigma = \{0, 1\}$, składający się ze wszystkich słów, które zaczynają się i kończą takim samym symbolem

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{S \to 0, S \to 1, S \to 0A, A \to 0A, A \to 1A, A \to 0, S \to 1B, B \to 0B, B \to 1B, B \to 1\}$$

6.2.3 Język L nad alfabetem $\Sigma = \{0, 1\}$, składający się ze wszystkich słów, które zawierają co najmniej trzy symbole '0'

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{S \to 0A, S \to 1S, A \to 0B, A \to 1A, B \to 0C, B \to 1B, C \to 1C, C \to 0C, C \to \epsilon\}$$

6.2.4 Język L nad alfabetem $\Sigma = \{0,1\}$, składający się ze wszystkich słów, które zawierają podciąg '101'

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{S \to 0S, S \to 1A, A \to 1A, A \to 0B, B \to 1C, B \to 0S, C \to 1C, C \to 0C, C \to \epsilon\}$$

6.2.5 Język L nad alfabetem $\Sigma = \{0,1\}$, składający się ze wszystkich słów, które nie zawierają podciągu '110'

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{S \to 1A, S \to 0S, S \to \epsilon, A \to 1B, A \to 0S, A \to \epsilon, B \to 1B, B \to \epsilon\}$$

6.2.6 Język L nad alfabetem $\Sigma = \{0,1\}$, składający się ze wszystkich słów, które nie zawierają wystąpienia trzech takich samych symboli bezpośrednio po sobie

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{S \to 0A, S \to 1B, A \to 0A', A \to 1B, A' \to 1B, A' \to \epsilon, B \to 0A, B \to 1B', B' \to 0A, B \to \epsilon\}$$

6.2.7 b i c

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{S \to 0A, A \to 0A', A \to 1A, A' \to 0A'', A' \to 1A', A'' \to 0, A'' \to 0A'', A'' \to 1A'', S \to 1B\} \cup$$

$$\{B \to 0B', B \to 1B, B' \to 0B'', B' \to 1B', B'' \to 0B''', B'' \to 1B'', B''' \to 1, B''' \to 1B''', B \to 0B'''\}$$

6.2.8 a lub e

Przy lub proponowałbym zrobić takie przejście alternatywowe $S \to S', S \to S''$, gdzie jedno odpowiadałoby za podpunkt a, a drugie za podpunkt e.

6.2.9 b i c i e

Tutaj można by spróbować w ten sposób, że łapiemy dwa przypadki, albo $S \to 1A1$ i wtedy musimy to A rozwinąć w taką gramatykę, żeby złapać te trzy zera, ale przy okazji nie złapać tego 110, czyli będziemy uważali na początku, żeby nie wystartować jako 10 z tego A, ale później mamy luzik, bo po prostu wstawimy taką gramatykę co robi c i e (taaa luzik...),

drugi przypadek $S \to 0B0$, i przy B wystarczy że złapiemy jeszcze jedno zero i przy okazji nie złapiemy ciągu '110', tylko tutaj musimy też uważać, bo nie możemy zakończyć na '10', ale za to na początku luźniej.

6.2.10 d lub e

Analogicznie do jednego z powyższych podpunktów (te z lub są duże, ale jeszcze nie takie najgorsze i.

6.2.11 a i c i f

No tutaj to już nam wyjdzie niezły automat... Trzeba by wziąć i rozpatrywać każde przejście, pamiętając tak ze trzy przejścia wstecz, no i dałoby się, ale to tyle roboty, że no cóż...

6.3 Podaj gramatyki regularne generujące języki takie same jak podane wyrażenia regularne

6.3.1 $0^*(1+0)0^*$

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{A \to 0A, A \to 0B, A \to 1B, B \to 0B, B \to \epsilon\}$$

6.3.2 $(11 + 00)^*$

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{A \to 11A, A \to 00A, A \to \epsilon\}$$

6.3.3 $1^*(0 + 00 + 10)^*$

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{A \to 1A, A \to 0B, A \to 00B, A \to 10B, B \to 0B, B \to 00B, B \to 10B, B \to \epsilon\}$$

6.3.4 $0^* + 10 + 1^*(0^*0)1$

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{A \to \epsilon, A \to 0B, B \to 0B, B \to 0, A \to 10, A \to 1C, A \to 0D, C \to 1C, C \to 1D, D \to 0D, D \to 1\}$$

6.3.5 $(1 + 00^* + 10^*1)^*$

Produkcje dla gramatyki regularnej, która to zrobi:

$$P = \{A \to 0A, A \to 1A, A \to 0C, C \to 0C, C \to 0A, A \to 1D, D \to 0F, F \to 0F, F \to F1A, A \to \epsilon\}$$

6.4 Podaj gramatyki kontekstowe o ile istnieją dla następujących języków:

6.4.1 $L = \{a^n b^n c^n : n > 0\}$

Produkcje dla kontekstowej, która to zrobi:

$$P = \{S \to aSBC, S \to aBC, aB \to ab, bB \to bb, bC \to bc, cC \to cc, CB \to CB', CB' \to C'B', C'B' \to C'C, C'C \to BC\}$$

6.4.2 $L = \left\{a^n b^{2n} c^n : n > 0\right\}$

Produkcje dla kontekstowej, która to zrobi:

$$P = \{S \to aSBC, S \to aBC, aB \to abb, bB \to bbb, bC \to bc, cC \to cc, CB \to CB', CB' \to C'B', C'B' \to C'C, C'C \to BC\}$$

6.4.3 $L = \left\{a^n b^{2n} c^{3n} : n > 0\right\}$

Produkcje dla kontekstowej, która to zrobi:

$$P = \{S \to aSBC, S \to aBC, aB \to abb, bB \to bbb, bC \to bccc, cC \to cccc, CB \to CB', CB' \to C'B'\} \cup$$
$$\{C'B' \to C'C, C'C \to BC\}$$

6.4.4 $L = \left\{ww : w \in \{0,1\}^*, |w| \geqslant 1\right\}$

https://math.stackexchange.com/questions/163830/context-sensitive-grammar-for-the-copy-language
Ale nawet chyba ta z linku nie wystarczy :/

6.4.5 $L = \{a, a2, a4, a8, \ldots\}$

Do tego da się fajną struktur frazowych, ale kontekstowej nie wymyślę chyba :/

Tamta struktur frazowych to (uwaga, to nie jest poprawne rozwiązanie tego zadania, ale występowało na egzaminach):

$$P = \{S \to a, S \to LSQ, La \to aaL, LLQQ \to LQ, aLQ \to a\}$$

I ta powyższa fajnie działa, no ale nie jest kontekstowa, co gorsza nie mam pojęcia jak moglibyśmy ją na kontekstową przerobić :/

6.4.6 Język L nad alfabetem $\Sigma = \{a, b, c\}$, składający się ze wszystkich słów, które zawierają taką samą ilość symboli a, b i c.

Działanie będzie wyglądało mniej więcej tak, najpierw produkujemy sobie jakąś ilość symboli nieterminalnych ABC w tej kolejności lub jak kazaliby to też wyprowadzać, to w jakiś sposób pomieszanych. Następnie, gdy mamy już taką ilość symboli nieterminalny, jaką chcieliśmy mieć terminalnych, możemy zająć się poprzestawianiem niektórych symboli tak, aby możliwa była dowolna kombinacja, stworzymy więc zestawy sześciu analogicznych produkcji, które będą odpowiadały za przemieszanie A z B, A z C, B z C, B z A, C z A oraz C z B (gdybyśmy tworzyli gramatykę struktur frazowych, no to wystarczyłoby sześć produkcji, a tak to musimy korzystać z drogi naokoło w celu zamienienia). Gdy mamy już wszystko poustawiane zajmujemy się zamianą odpowiednich nieterminalnych A, B, C w terminalne a, b, c.

Produkcje dla kontekstowej, która to zrobi:

$$P = \{S \to ABC, S \to ABCS\} \cup$$

$$\{AB \to AB', AB' \to A'B', A'B' \to BB', BB' \to BA\} \cup$$

$$\{AC \to AC'', AC'' \to A''C'', A''C'' \to CC'', CC'' \to CA\} \cup$$

$$\{BA \to BA''', BA''' \to B'''A''', B'''A''' \to AA''', AA''' \to AB\} \cup$$

$$\{BC \to BC'''', BC'''' \to B''''C'''', B''''C'''' \to CC'''', CC'''' \to CB\} \cup$$

$$\{CA \to CA'''', CA'''' \to C''''A'''', C''''A'''' \to AA'''', AA'''' \to AC\} \cup$$

$$\{CB \to CB''''', CB''''' \to C'''''B''''', C'''''B''''' \to BB''''', BB''''' \to BC\} \cup$$

$$\{A \to a, B \to b, C \to c\}$$

6.5 Jaki język generuje gramatyka, czy istnieje gramatyka wyższej klasy w hierarchii Chomsky'ego generująca ten sam język?

$$P = \{A \to dB, B \to DB, B \to dB, B \to d, DD \to D, dDd \to ded\}$$

Językiem będą dowolne ciągi o długości większej niż jeden, zaczynające się i kończące na znaku d, zawierające w sobie dowolną ilość znaków d oraz dowolną ilość znaków e, przy czym dwa znaki e nie mogą wystąpić obok siebie.

Produkcje dla gramatyki regularnej, która zrobi to samo:

$$P = \{A \to dB, B \to eC, B \to dB, C \to dB, B \to d\}$$

6.6 Jaki język generuje gramatyka, czy istnieje gramatyka wyższej klasy w hierarchii Chomsky'ego generująca ten sam język?

$$P = \{H \to QRX, Q \to fQ, Q \to fR, RR \to X, fX \to f\}$$

Język : $\{f^n\}, n > 0$

No i oczywiście istnieje na niego regularna.

6.7 Określ język generowany przez gramatykę:

$$P = \{S \to 0SBC, S \to 0BC, CB \to BC, 0B \to 01, 1B \to 11, 1C \to 12, 2C \to 22\}$$

Język : $\{0^n 1^n 2^n\}, n > 0$