**Εθνικό Μετσόβιο Πολυτεχνείο**



*Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών*

**Θεμελιώδη Θέματα Επιστήμης Υπολογιστών, 2019-20**

**1η Σειρά Γραπτών Ασκήσεων**

**(αυτόματα – τυπικές γλώσσες – γραμματικές - λογική – υπολογισιμότητα – πολυπλοκότητα)**

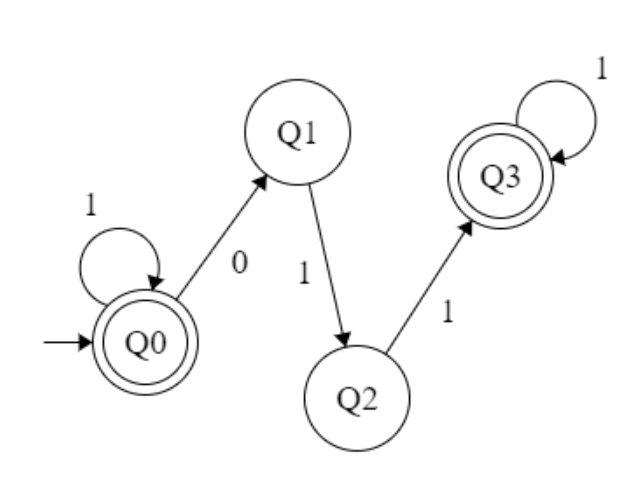
**Ονοματεπώνυμο:** *Θοδωρής Αράπης*

*(el18028, theodoraraps2000@gmail.com)*

**Άσκηση 1.**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Q** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **Q0** | **Q0** | **Q1** | **Q2** | **Q0** | **Q1** | **Q3** | **Q0** | **Q1** | **Q2** | **Q0** |
| **Q1** | **Q1** | **Q3** | **Q0** | **Q1** | **Q2** | **Q0** | **Q1** | **Q3** | **Q0** | **Q1** |
| **Q2** | **Q2** | **Q0** | **Q1** | **Q3** | **Q0** | **Q1** | **Q2** | **Q0** | **Q1** | **Q3** |
| **Q3** | **Q2** | **Q0** | **Q1** | **Q3** | **Q0** | **Q1** | **Q2** | **Q0** | **Q1** | **Q3** |

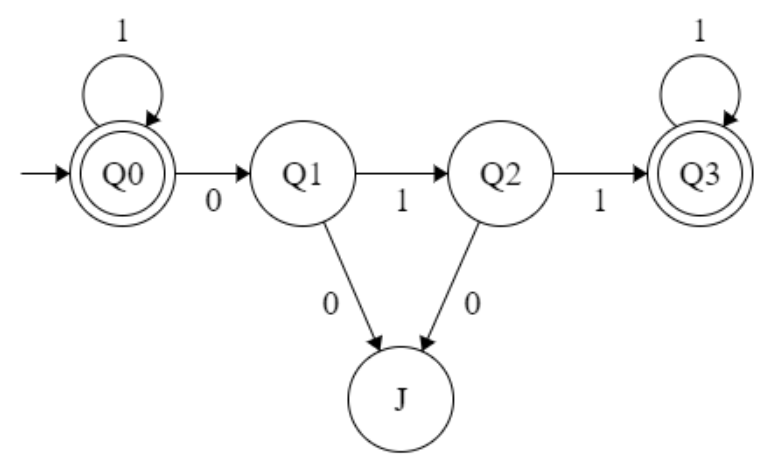
**Άσκηση 2.**

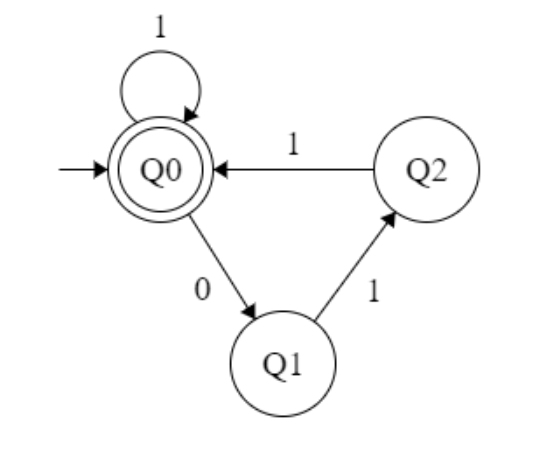
**L1:**

(i)

|  |  |  |
| --- | --- | --- |
| **Q** | **0** | **1** |
| **Q0** | **Q1** | **Q0** |
| **Q1** | **Ø** | **Q2** |
| **Q2** | **Ø** | **Q3** |
| **Q3** | **Q1** | **Q3** |

(ii) Ισοδύναμο DFA:



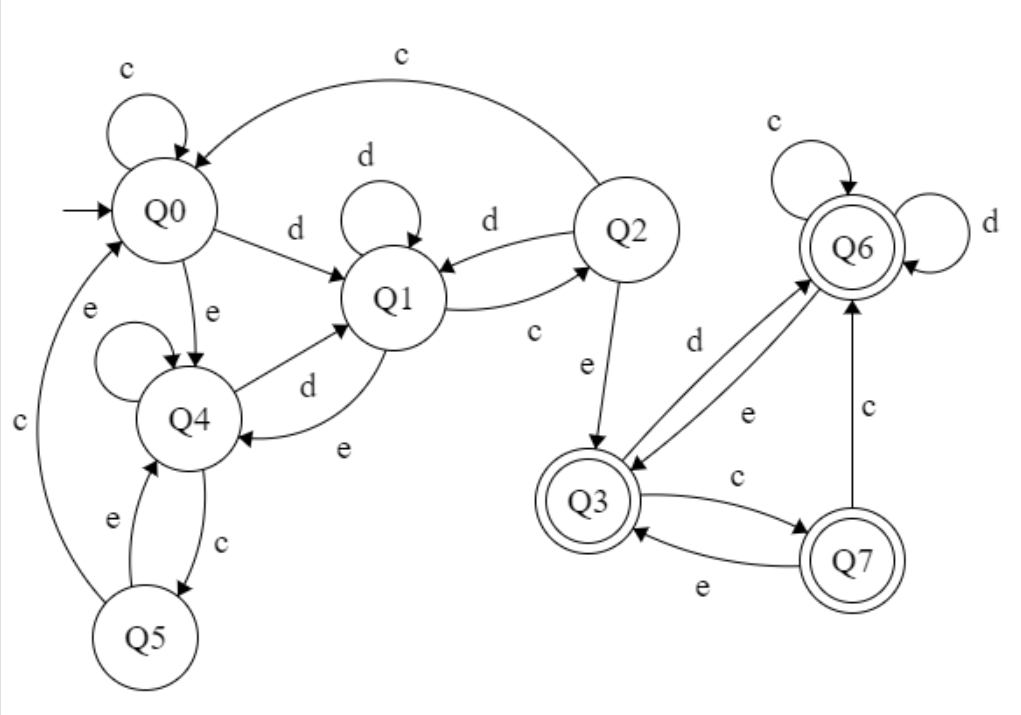
Ελαχιστοποίηση DFA:

|  |
| --- |
| **Q0** |
| **Q1** | **Χ0** |
| **Q2** | **Χ0** | **Χ1** |
| **Q3** |  | **Χ0** | **Χ0** |
|  | **Q0** | **Q1** | **Q2** | **Q3** |

Ελαχιστοποιημένο DFA

Με εφαρμογή του θεωρήματος Myhill-Nerode, παραπάνω, αποδεικνύεται ότι το DFA στο (i) δεν είναι ελάχιστο.

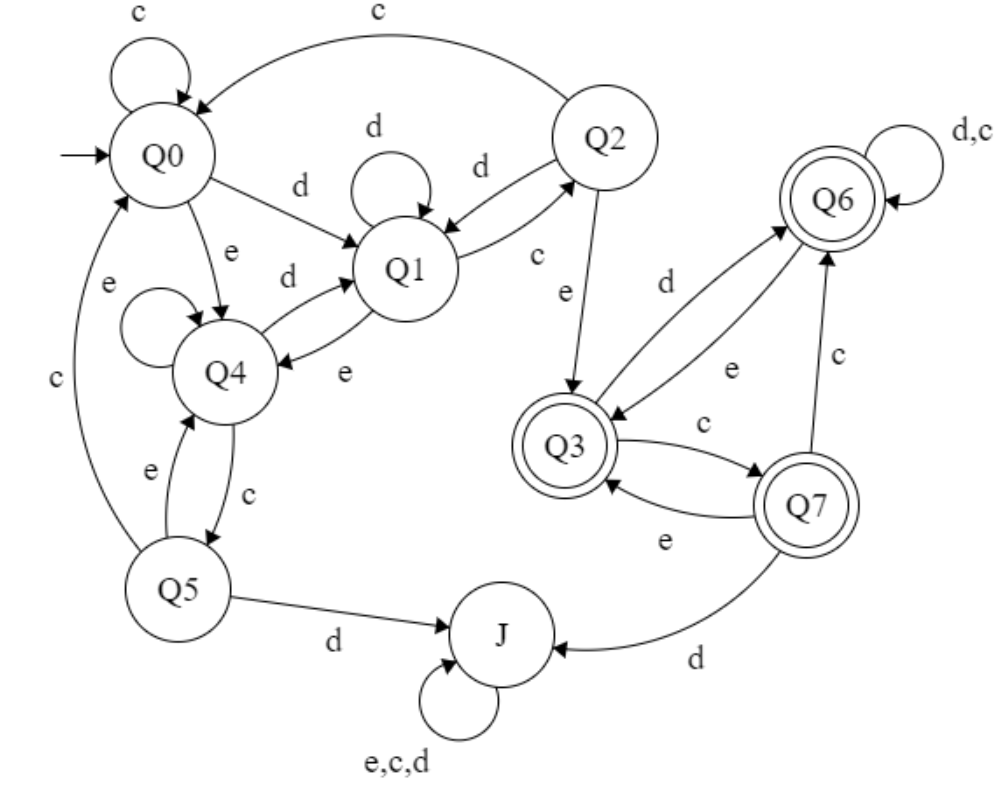
(iii) Κανονική παράσταση της γλώσσας L1: ***1\*(011+)\****

**L2:**

(i)

|  |  |  |  |
| --- | --- | --- | --- |
| **Q** | **a** | **d** | **c** |
| **Q0** | **Q0** | **Q1** | **Q4** |
| **Q1** | **Q2** | **Q1** | **Q4** |
| **Q2** | **Q0** | **Q1** | **Q3** |
| **Q3** | **Q7** | **Q6** | **Q3** |
| **Q4** | **Q5** | **Q1** | **Q4** |
| **Q5** | **Q0** | **Ø** | **Q4** |
| **Q6** | **Q6** | **Q6** | **Q3** |
| **Q7** | **Q6** | **Ø** | **Q3** |

(ii) Ισοδύναμο DFA:



Ελαχιστοποίηση DFA:

|  |
| --- |
| **Q0** |
| **Q1** | **Χ3** |
| **Q2** | **Χ2** | **Χ2** |
| **Q3** | **Χ0** | **Χ0** | **Χ0** |
| **Q4** | **Χ2** | **Χ3** | **Χ2** | **Χ0** |
| **Q5** | **Χ1** | **Χ3** | **Χ2** | **Χ0** | **Χ1** |
| **Q6** | **Χ0** | **Χ0** | **Χ0** | **Χ2** | **Χ0** | **Χ0** |
| **Q7** | **Χ0** | **Χ0** | **Χ0** | **Χ0** | **Χ0** | **Χ0** | **Χ0** |
|  | **Q0** | **Q1** | **Q2** | **Q3** | **Q4** | **Q5** | **Q6** | **Q7** |

Με εφαρμογή του θεωρήματος Myhill-Nerode, παραπάνω, αποδεικνύεται ότι το DFA στο (i) είναι ελάχιστο (αφού δεν υπάρχουν καταστάσεις που να μπορούν να συγχονευτούν).

(iii) Κανονική παράσταση της γλώσσας L2: [***c\*[(ee\*c(e+c)\*c)\*+d+ed(e+d)\*]d\*c(d+c)\*(c+[(ee\*c(e+c)\*c)\*+d+ed(e+d)\*]d\*c(d+c)\*)\*e+c(e+c)\*c+c\*[(ee\*c(e+c)\*c)\*+d+ed(e+d)\*]d\*c(d+c)\*(c+[(ee\*c(e+c)\*c)\*+d+ed(e+d)\*]d\*c(d+c)\*)\*e+d(e+d)\*]c\*d\****

Πιο απλά:

Για ***λ = c\*[(ee\*c(e+c)\*c)\*+d+ed(e+d)\*]d\*c(d+c)\****

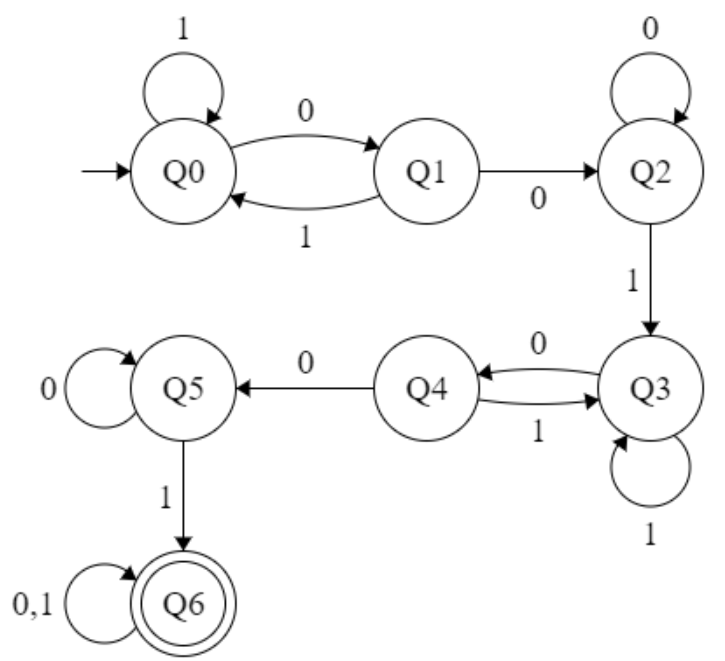
θα έχω:

***[λ(cλ)\*e+c(e+c)\*c+λ(cλ)\*e+d(e+d)\*]c\*d\****

**L3:**

(i)

|  |  |  |
| --- | --- | --- |
| **Q** | **0** | **1** |
| **Q0** | **Q1** | **Q0** |
| **Q1** | **Q2** | **Q0** |
| **Q2** | **Q2** | **Q3** |
| **Q3** | **Q4** | **Q3** |
| **Q4** | **Q5** | **Q3** |
| **Q5** | **Q5** | **Q6** |
| **Q6** | **Q6** | **Q6** |



(ii) Ισοδύναμο DFA:

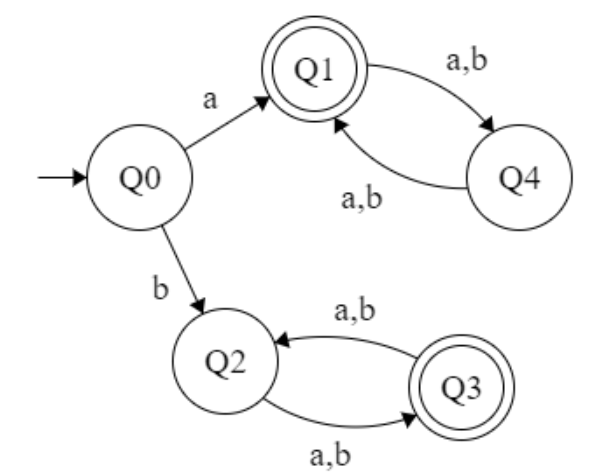
Στην πραγματικότητα το παραπάνω είναι DFA. Προκειμένου να ήταν αισθητή η διαφόρα DFA-NFA θα μπορούσαμε να προσθέσουμε στο προηγούμενο ορισμένες μεταβάσεις (π.χ. δ(Q2, 0)Q0), ώστε να έχουμε πολλαπλές μεταβάσεις από μία κατάσταση για μία δεδομένη είσοδο.

Ελαχιστοποίηση DFA:

|  |
| --- |
| **Q0** |
| **Q1** | **Χ5** |
| **Q2** | **Χ4** | **Χ4** |
| **Q3** | **Χ3** | **Χ3** | **Χ3** |
| **Q4** | **Χ2** | **Χ2** | **Χ2** | **Χ2** |
| **Q5** | **Χ1** | **Χ1** | **Χ1** | **Χ1** | **Χ1** |
| **Q6** | **Χ0** | **Χ0** | **Χ0** | **Χ0** | **Χ0** | **Χ0** |
|  | **Q0** | **Q1** | **Q2** | **Q3** | **Q4** | **Q5** | **Q6** |  |

Με εφαρμογή του θεωρήματος Myhill-Nerode, παραπάνω, αποδεικνύεται ότι το DFA στο (i) είναι ελάχιστο (αφού δεν υπάρχουν καταστάσεις που να μπορούν να συγχονευτούν).

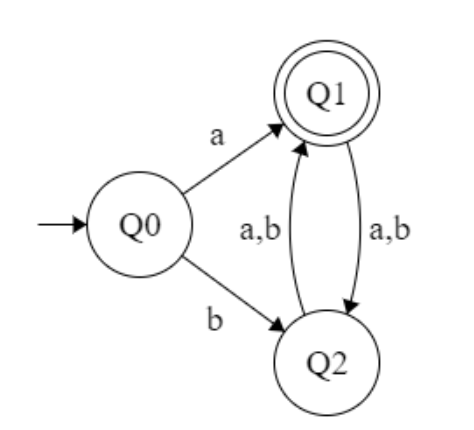
(iii) Κανονική παράσταση της γλώσσας L3: ***(1+0)\*001(1+0)\*001(1+0)\****

**L4:**

|  |  |  |
| --- | --- | --- |
| **Q** | **a** | **b** |
| **Q0** | **Q1** | **Q2** |
| **Q1** | **Q4** | **Q4** |
| **Q2** | **Q3** | **Q3** |
| **Q3** | **Q2** | **Q2** |
| **Q4** | **Q1** | **Q1** |

(ii) Ισοδύναμο DFA:

Το παραπάνω είναι και DFA.



|  |  |
| --- | --- |
| **Q0** |  |
| **Q1** | **Χ0** |
| **Q2** | **Χ1** | **Χ0** |
| **Q3** | **Χ0** |  | **Χ0** |
| **Q4** | **Χ1** | **Χ0** |  | **Χ0** |
|  | **Q0** | **Q1** | **Q2** | **Q3** | **Q4** |

Ελαχιστοποίηση DFA:

Ελαχιστοποιημένο DFA

Με εφαρμογή του θεωρήματος Myhill-Nerode, παραπάνω, αποδεικνύεται ότι το DFA στο (i) δεν είναι ελάχιστο.

(iii) Κανονική παράσταση της γλώσσας L4: ***[a+b(a+b)](aa+ab+ba+bb)\****

**Άσκηση 3.**

α) Η γλώσσα δεν είναι κανονική γιατί δεν υπάρχει αυτόματο που να μπορεί να “μετράει” το πλήθος των ‘0’ ή των ‘1’. Επομένως δεν υπάρχει κάποιος τρόπος να ελέγξει αν όντως υπάρχουν διπλάσσια ‘0’ από ‘1’.

Απόδειξη:

Θα χρησιμοποιήσουμε το Pumping Lemma.

Έστω ότι η γλώσσα είναι κανονική. Θέτω nZ το Pumping length και αποδεικνύω για κάθε n. Διαλέγω zL τέτοιο ώστε |z| = 3n ≥ n (2n το πλήθος των ‘0’ και n το πλήθος των ‘1’). Η z μπορεί να γραφεί: z = uvw με |uv| ≤ n και |v| ≥ 1. Αρκέι να μελετήσουμε τα strings για τα οποία ισχύει: z = 02n1nL. Τότε αναγκαστικά v = 0k, k ≥ 1. Διαλέγω i = 2: uvvw = 02n+k1nL. Άτοπο. Άρα δεν είναι κανονική γλώσσα.

|  |
| --- |
| \*\*\*Απόδειξη για το ότι αν μια γλώσσα δεν είναι κανονική τότε ούτε η συμπληρωματική της θα είναι κανονική:  Έστω ένα DFA που αποδέχεται μία γλώσσα L. Ισχύει ότι αν αντιστραφούν οι καταστάσεις που αποδέχεται με τις καταστάσεις που δεν αποδέχεται, τότε δημιουργείται ένα DFA που αποδέχεται την συμπληρωματική της L. Αν είχαμε NFA τότε θα έπρεπε πρώτα να μετατραπεί στο ισοδύναμό του DFA και μετά να παίρναμε το αντεστραμμένο αυτού. Συνεπώς αν μία γλωσσα είναι κανονική και επομένως υπάρχει αυτόματο που να την αποδέχεται, τότε (πάντα) και η συμπληρωματική της θα είναι κανονική (αφού θα υπάρχει αυτόματο που να την αποδέχεται, άμα κάνουμε την προαναφερθείσα αντιστροφή). Επομένως εύκολα συμπεραίνουμε πως αν μία γλώσσα δεν είναι κανονική τότε ούτε και η συμπληρωματική της θα είναι κανονική.  π.χ. |

β) Η γλώσσα δεν είναι κανονική. Αρκεί να δείξω ότι η γλώσσα που αποδέχεται παλινδρομικές συμβολοσειρές (δηλαδή η συμπληρωματική της) δεν είναι κανονική. Οπότε σύμφωνα με την απόδειξη (\*\*\*) ούτε αυτή θα είναι κανονική.

Η L = {w ∈ {0, 1}\* : η w είναι παλινδρομική} δεν είναι κανονική.

Απόδειξη:

Θα χρησιμοποιήσουμε το Pumping Lemma.

Έστω ότι η γλώσσα είναι κανονική. Θέτω nϵZ το Pumping length και αποδεικνύω για κάθε n. Διαλέγω zL τέτοιο ώστε |z| = 2n ≥ n (2n το μήκος της z). Η z μπορεί να γραφεί: z = uvw με |uv| ≤ n και |v| ≥ 1. Τότε:

z = για i = 2: uvvw = , η οποία δεν ανήκει στην L διότι δεν είναι παλινδρομική. Άρα έχουμε άτοπο. Επομένως δεν είναι κανονική γλώσσα. Άρα από (\*\*\*) ούτε η {w ∈ {0, 1}\* : η w δεν είναι παλινδρομική} είναι κανονική.

γ) Η γλώσσα είναι κανονική αφού οι συμβολοσειρές της γλώσσας έχουν πεπερασμένο μήκος. Συνεπώς, υπάρχει αυτόματο που να μπορεί να περιγράψει όλες τις συμβολοσειρές της γλώσσας και να τις αποδέχεται.

δ) Η γλώσσα δεν είναι κανονική γιατί δεν υπάρχει αυτόματο που να μπορεί να “μετράει” το πλήθος των ‘0’ ή των ‘1’. Επομένως δεν υπάρχει κάποιος τρόπος να ελέγξει αν όντως υπάρχει διαφορετικό πλήθος ‘0’ και ‘1’. Αρκεί να δείξω ότι η συμπληρωματική της δεν είναι κανονική. Οπότε σύμφωνα με την απόδειξη (\*\*\*) ούτε αυτή θα είναι κανονική.

Η L = {w ∈ {0, 1}\* : 0n1m όπου n = m} δεν είναι κανονική.

Απόδειξη:

Θα χρησιμοποιήσουμε το Pumping Lemma.

Έστω ότι η γλώσσα είναι κανονική. Θέτω nZ το Pumping length και αποδεικνύω για κάθε n. Διαλέγω zL τέτοιο ώστε |z| = 2n ≥ n (n το πλήθος των ‘0’ και n το πλήθος των ‘1’). Η z μπορεί να γραφεί: z = uvw με |uv| ≤ n και |v| ≥ 1. Τότε:

z = για i = 2: uvvw = ,

η οποία δεν ανήκει στην L διότι δεν είναι παλινδρομική. Άρα έχουμε άτοπο. Επομένως δεν είναι κανονική γλώσσα. Άρα από (\*\*\*) ούτε η {w ∈ {0, 1}\* : 0n1m όπου n ≠ m} είναι κανονική.

**Άσκηση 4.**

α) Η γλώσσα που παράγει η G είναι η: {w {a, b}\* : το πλήθος των ‘a’ είναι μεγαλύτερο ή ισο από το πλήθος των ‘b’}, όπου το b μπορεί να έχεις και μηδενικό πλήθος.

β) Η γραμματική για τη γλώσσα θα είναι γραμματική χωρίς συμφραζόμενα

και θα είναι η εξής: G: S aaaAbb, A aaaAbb|ε.

Θα χρειαστούμε ένα αυτόματο στοίβας για να αποδεχτούμε αυτήν την γλώσσα. Το αυτόματο αυτό θα λειτουργεί ως εξής: Θα μετράει το πλήθος των ‘a’ που εισέρχονται (χρησιμοποιώντας την πράξη modulo3) και κάθε φορά που διαβάζει τρία ‘a’ θα κάνει push δύο ‘b’ στην στοίβα. Έτσι, όταν στην είσοδο αρχίσει να διαβάζει ‘b’, για κάθε ένα που διαβάζει θα αφαιρεί και ένα ‘b’ από την στοίβα μέχρι αυτή να αδειάσει.

γ) Η γραμματική για τη γλώσσα θα είναι γραμματική με συμφραζόμενα και θα είναι η εξής:

G: S A111B, A111B 0A111B0|0A111B0|1A111B1|1A111B0|111

**Άσκηση 5.**

Έστω L μία γλώσσα χωρίς συμφραζόμενα και G η γραμματική της. Θα εξετάσουμε αν η LR είναι και αυτή γλώσσα χωρίς συμφραζόμενα. Έστω GR η γραμματική της LR, προκειμένου να ισχύει:

Αν G: Α α, τότε GR: Α αR

Αν G: Α w1Bw2, Β u, τότε GR: Α w1RBw2R, Β uR

Όπου στην πρώτη έχουμε w1uw2 και στην δευτερη w1RuRw2R = (w1uw2)R

Αν G: Α ε, τότε GR: Α ε

Παρατηρούμε λοιπόν οτι σε κάθε περίπτωση προκύπτει πράγματι η ανάστροφη συμβολοσειρά. Άρα, οι γλώσσες χωρίς συμφραζόμενα είναι κλειστές ως προς την πράξη αναστροφή.

**Άσκηση 6.**

Halting Problem

Έστω ότι το πρόβλημα είναι επιλύσιμο. Είναι γνωστό ότι το σύνολο όλων των προγραμμάτων είναι αριθμήσιμο. Ορίζουμε μία μηχανιστική απαρίθμηση (Π1, Π2, Π3, ... , Πn) όλων των προγραμμάτων. Δημιουργούμε ένα πρόγραμμα Π το οποίο να ελέγχει αν ένα πρόγραμμα τερματίζει ή όχι με τέτοιον τρόπο ώστε αν το Πν, νΝ, με είσοδο ν τερματίζει τότε και το Π να τερματίζει. Το Π, όμως, θα πρέπει να ανήκει στην παραπάνω αρίθμηση. Έστω k, Π = Πk, ο δείκτης του Π. Δίνοντας, λοιπόν, ως είσοδο το k, παρατηρούμε ότι το Πk(k) θα τερματίζει ανν το Πk τερματίζει, το οποίο συμβαίνει όταν το Πk(k) δεν τερματίζει ποτέ. Άτοπο. Άρα το Halting Problem είναι μη αποκρίσιμο.

Totality Problem

Αρκεί να δείξουμε ότι το Halting Problem ανάγεται στο Totality Problem.

Αφού κανένας αλγόριθμος δεν μπορεί να επιλύσει το HP, τότε κανένας αλγόριθμος δεν θα μπορεί να επιλύσει το TP. Για κάθε πρόγραμμα Πν με είσοδο ν κατασκευάζουμε ένα πρόγραμμα Πν’ , το οποίο λαμβάνει μία αφηρημένη είσοδο, την αγνοεί και τρέχει το Πν(ν). Το Πν’ θα τερματίζει για κάθε είσοδο αν και μόνο αν το Πν τερματίζει για είσοδο ν. Συνεπώς, ένας αλγόριθμος που μας πληροφορεί για το εάν το Πν’ τερματίζει για κάθε είσοδο, μας πληροφορεί και για το εάν τερματίζει για είσοδο ν. Έχουμε λοιπόν μία λύση για το HP.

Halting Problem p Totality Problem

**Άσκηση 7.**

Ο αλγόριθμος είναι ο εξής:

Αρχικά, εφόσον γνωρίζουμε το πλήθος των literals (αλλιώς διασχίζουμε μία φορά τον τύπο και το βρίσκουμε), δημιουργούμε έναν πινακα α[Ν] (integer), όπου Ν το πλήθος των literals και θέτουμε σε κάθε θέση την τιμή ‘1’. Ύστερα διατρέχουμε ξανά τον τύπο και εξετάζουμε τις φράσεις ανάμεσα στις παρενθέσεις. Τοποθετούμε σε έναν πίνακα μεταβλητού μεγέθους τις μεταβλητές που διαβάζονται κατά αύξουσα σειρά δείκτη. Ελέγχουμε αν υπάρχουν σε γειτονικές θέσεις μεταβλητές και το συμπλήρωμά τους. Αν ναι τότε προχωράμε στις επόμενες παρενθέσεις και επαναλαμβάνουμε.

* Αν σε κάποιο ζεύγος παρενθέσεων δεν βρούμε συμπληρωματικές μεταβλητές στις γειτονικές θέσεις, τότε για κάθε συμπληρωματική μεταβλητή (xi), στον μεταβλητού μεγεθους πίνακα, βάζουμε ‘0’ στην θέση α[i]. Τέλος, εκττυπώνουμε «ΝΑΙ» και για κάθε i από 0 εώς και N-1 εκτυπώνουμε την ανάθεση [π.χ. (TRUE, FALSE, FALSE, … )], όπου η πρώτη τιμή καθορίζεται από το α[0] (αν α[0] = 0 τοτε «TRUE» αλλιώς «FALSE»), η δεύτερη από το α[1] κ.ο.κ.
* Αν σε κάθε ζεύγος παρενθέσεων βρούμε συμπληρωματικές μεταβλητές, τοτε εκτυπώνουμε «ΟΧΙ».
* Αν η είσοδος είναι κενή τότε ο αλγόριθμος θα εκτυπώνει «ΟΧΙ».

**Άσκηση 8.**

α) Αρχικά παρατηρούμε ότι αν ένα υπογράφημα του G είναι κλίκα, τότε, αν αντιστρέψουμε τον G, η κλίκα αυτή θα αποτελεί ένα ανεξάρτητο σύνολο κορυφών για τον G’ και αντίστροφα. Συνεπώς, κάθε αλγόριθμος που εντοπίζει ένα ανεξάρτητο σύνολο κορυφών (μεγέθους τουλάχιστον k) σε έναν γράφο G, μπορεί να χρησιμοποιηθεί για την εύρεση κλίκας (μεγέθους τουλάχιστον k) στον ίδιο γράφο. Αυτό συμβαίνει διότι τα σύνολα των κορυφών που δεν είναι ανεξάρτητα, είναι κλίκες. Πιο αναλυτικά, έστω S ένα ανεξάρτητο σύνολο για έναν δεδομένο γράφο G = (V, E). Τότε, για κάθε ακμή e = (x, y), eE και x, yV, το πολύ μία από τις δύο κορυφές της x, y θα ανήκει στο S. Ως εκ τούτου, τουλάχιστον μία από αυτές θα ανήκει στο SR (SR = V – S). Συνεπώς, κάθε ακμή έχει τουλάχιστον μία κορυφή της στο SR. Άρα, το SR είναι κλίκα.

Είναι, λοιπόν, προφανές ότι αν ένας αποδοτικός αλγόριθμος επιλύει το πρόβλημα Clique, μπορεί να χρησιμοποιηθεί για να επιλύσει το πρόβλημα Independent Set για τουλάχιστον κάποιο μέγεθος k, αν τον βάλουμε να αναζητήσει αν ο G έχει μία κλίκα μεγέθους το πολύ n-k (όπου n το μέγεθος του γράφου). Συνεπώς, λόγω αναγωγής, αφού το Independent Set είναι NP-complete, τότε και το Clique θα είναι NP-complete.

Independent Set p Clique

β) Έχοντας αποδείξει παραπάνω ότι το Clique είναι NP-complete, μπορούμε να δείξουμε ότι αυτό ανάγεται στο Dense Subgraph. Θα ισχύει:

Έστω μία είσοδος (G, k) του προβλήματος Clique. Εφόσον ο G έχει τουλάχιστον μία κλίκα C μεγέθους k και πάνω, τότε είναι προφανές πως θα υπάρχει μία κλίκα Ck μεγέθους ακριβώς k (CkC). Είναι γνωστό ότι ένας πλήρης γράφος (και συνεπώς εδώ η Ck) έχει πλήθος ακμών: , όπου k το πλήθος των κορυφών του. Επομένως, γνωρίζοντας αυτά, μπορούμε να λύσουμε το πρόβλημα Clique για (G, k), λύνοντας το πρόβλημα Dense Subgraph για γράφο G, a = k και b = . Συνεπώς, το Clique ανάγεται στο Dense Subgraph. Άρα και το Dense Subgraph είναι NP-complete.

Clique p Dense Subgraph