Κεφάλαιο 5

Context free γραμματικές και γλώσσες

5.1 Ένας τρόπος περιγραφής απλών αριθμητικών εκφράσεων

Ας υποθέσουμε ότι θέλουμε να ορίσουμε το τι σημαίνει σωστά δομημένη αριθμητική έκφραση. Για να κάνουμε τα πράγματα πιο απλά (χωρίς να χαθεί η ουσία) ας περιοριστούμε σε αριθμητικές εκφράσεις όπου οι μόνες πράξεις είναι η πρόσθεση (+) και ο πολλαπλασιασμός (*), και όπου ισχύουν οι συνηθισμένοι κανόνες για τις παρενθέσεις. Ας υποθέσουμε επίσης ότι οι βασικές ποσότητες που φτιάχνουν μια έκφραση συμβολίζονται όλες με το γράμμα x. Για παράδειγμα η έκφραση $x * (x + x) + x$ είναι μια σωστή έκφραση ενώ η $xx + *()$ δεν είναι.

Έχουμε δηλαδή ένα πολύ περιορισμένο αλφάβητο

$$\Sigma = \{+, *, (,), x\}$$

και προσπαθούμε να ορίσουμε τη γλώσσα των σωστών εκφράσεων, που είναι ένα κομμάτι του Σ^*.

Ένας γρήγορος και καθαρός τρόπος να τις ορίσουμε είναι να σκεφτούμε το πώς τις φτιάχνουμε: κολλάμε μαζί, με συγκεκριμένους κανόνες, μικρότερες εκφράσεις και φτιάχνουμε μεγαλύτερες. Δίνουμε έτσι τον εξής ορισμό.

Ορισμός 33. Η λέξη x είναι σωστή έκφραση. Επίσης αν οι λέξεις w και v είναι σωστές εκφράσεις, τότε σωστές εκφράσεις είναι επίσης και οι λέξεις (w), $w + v$, $w * v$. Τέλος, σωστές εκφράσεις είναι μόνο οι λέξεις που προκύπτουν από τους άνω κανόνες.

Έτσι η λέξη $x * (x + x) + x$ είναι σωστή έκφραση επειδή οι λέξεις x και $(x + x) + x$ είναι σωστές, και η δεύτερη είναι σωστή επειδή επειδή οι $(x + x)$ και x είναι σωστές και η πρώτη από αυτές είναι σωστή επειδή η $x + x$ είναι σωστή και, τέλος, αυτή είναι σωστή επειδή η x είναι σωστή.

Ορισμοί σαν τον παραπάνω (αλλά και αρκετά πιο περίπλοκοι) κωδικοποιούνται στις λεγόμενες context free γραμματικές. Πριν δώσουμε τον ακριβή ορισμό για αυτές ας πούμε ότι η context free γραμματική που αντιστοιχεί στον παραπάνω ορσιμό είναι η:

1. $S \to x$
2. $S \to (S)$
3. $S \to S + S$
4. $S \to S * S$

Με το σύμβολο S συμβολίζουμε μια 'μεταβλητή' λέξη, η οποία μπορεί να αντικατασταθεί με το δεξί μέλος μιας από τις *παραγωγές* ($S \to \ldots$) από τις οποίες απαρτίζεται η γραμματική. Η λογική είναι ότι ξεκινάμε με τη λέξη S και εφαρμόζουμε σε αυτήν συνεχώς κάποιους από τους κανόνες παραγωγής μέχρι να πάρουμε τη λέξη που θέλουμε. Αν αυτό καταστεί εφικτό τότε, και μόνο τότε, η λέξη αυτή θα είναι στην context free γλώσσα που περιγράφεται από την άνω context free γραμματική.

Με την παρακάτω ακολουθία μετασχηματισμών προκύπτει π.χ. η λέξη $x * (x + x) + x$

$$\begin{array}{lll} S & \to & S+S & \text{(κανόνας παραγωγής 3)} \\ & \to & S*S+S & \text{(κανόνας παραγωγής 4)} \\ & \to & S*(S)+S & \text{(κανόνας παραγωγής 2)} \\ & \to & S*(S+S)+S & \text{(κανόνας παραγωγής 3)} \\ & \to & x*(x+x)+x & \text{(κανόνας παραγωγής 1, τέσσερεις φορές)} \end{array}$$

Εύκολα βλέπει κανείς ότι με όποιο τρόπο και να χρησιμοποιήσουμε τους κανόνες παραγωγής δε μπορούμε να πάρουμε από την S τη λέξη $xx + *()$.

5.2 Ορισμός context free γραμματικών και των γλωσσών τους

Ορισμός 34. Μια context free γραμματική G πάνω από ένα αλφάβητο Σ (τα *τερματικά σύμβολα*) είναι μια πεπερασμένη συλλογή από

- *μη τερματικά σύμβολα*, που συνήθως τα συμβολίζουμε με κεφαλαία λατινικά γράμματα, και που περιλαμβάνουν το διακεριμένο σύμβολο S (αρχικό μη τερματικό σύμβολο)
- κανόνες παραγωγής $X \to w$, όπου X είναι ένα μη τερματικό σύμβολο και w είναι μια λέξη από γράμματα του Σ και μη τερματικά σύμβολα (η w μπορεί να είναι και η κενή λέξη).

Αν G είναι μια context free γραμματική (CFG) και w, v είναι δύο λέξεις από τερματικά ή μη τερματικά σύμβολα, τότε γράφουμε

$$w \overset{G}{\Rightarrow} v$$

αν με χρήση ενός κανόνα παραγωγής $X \to \alpha$ μπορεί να προκύψει η λέξη v από τη λέξη w. Αυτό σημαίνει ότι αντικαθιστούμε μια εμφάνιση του μη τερματικού συμβόλου X στη λέξη w με τη λέξη α (που απαρτίζεται από τερματικά και μη τερματικά σύμβολα) και με την αντικατάσταση αυτή προκύπτει η λέξη v.

Για παράδειγμα, αν G είναι η γραμματική που ορίσαμε παραπάνω τότε ισχύει

$$x + S \overset{G}{\Rightarrow} x + (S)$$

μια και από την αριστερή λέξη προκύπτει η δεξιά αν χρησιμοποιηθεί ο κανόνας 2.

Ορίζουμε επίσης

$$w \overset{G}{\underset{*}{\Rightarrow}} v$$

να σημαίνει ότι υπάρχει πεπερασμένη ακολουθία λέξεων $v_1, \ldots, v_n$ ώστε

$$w \overset{G}{\Rightarrow} v_1 \overset{G}{\Rightarrow} \cdots \overset{G}{\Rightarrow} v_n \overset{G}{\Rightarrow} v,$$

ότι δηλ. η λέξη v μπορεί να προκύψει από τη λέξη w με επανειλημμένη χρήση των κανόνων παραγωγής της G. Στην παραπάνω γραμματική για τις εκφράσεις ισχύει, για παράδειγμα,

$$S \overset{G}{\underset{*}{\Rightarrow}} (S + S)$$

αφού μπορούμε από τη λέξη S να πάμε στη λέξη $(S + S)$ εφαρμόζοντας πρώτα τον κανόνα 2 και μετά τον κανόνα 3.

Έχοντας στα χέρια μας τον συμβολισμό αυτό μπορούμε εύκολα να ορίσουμε πλέον ποια είναι η *γλώσσα* που αντιστοιχεί σε μια λέξη (από τερματικά ή μη σύμβολα).

Ορισμός 35. Αν G είναι μια CFG και w μια λέξη από τερματικά ή μη τερματικά σύμβολα της G τότε η γλώσσα της w ορίζεται ως

$$L(w) = L_G(w) = \left\{ x \in \Sigma^* : \quad w \overset{G}{\underset{*}{\Rightarrow}} x \right\}.$$

Απαρτίζεται δηλ. η $L(w)$ από εκείνες τις λέξεις *χωρίς μη τερματικά σύμβολα* που μπορούν να παραχθούν σε πεπερασμένο πλήθος βημάτων από τη λέξη w με τους κανόνες παραγωγής της G.

Τέλος ορίζουμε *την* γλώσσα της G.

Ορισμός 36. Αν G είναι μια CFG η γλώσσα $L(G)$ ορίζεται να είναι η γλώσσα $L(S)$. Μια γλώσσα L λέγεται *context free γλώσσα* (CFL) αν είναι η γλώσσα κάποιας context free γραμματικής.

Είναι δηλ. η γλώσσα της G όλες οι λέξεις του Σ^* που παράγονται από το αρχικό μη τερματικό σύμβολο S.

Θα χρησιμοποιούμε συνήθως τη συντομογραφία

$$X \to w_1|w_2|\ldots|w_n$$

για να υποδηλώσουμε μια ομάδα από κανόνες παραγωγής

$$X \to w_1, \; X \to w_2, \; \ldots, X \to w_n,$$

με το ίδιο αριστερό μέλος.

Άσκηση 57. Ποια είναι η γλώσσα της γραμματικής με κανόνες $S \to \epsilon \mid 0S0 \mid 1S1$;

Άσκηση 58. Δώστε μια CFG για τη γλώσσα $\{0^n 1^n : \quad n = 1, 2, \ldots\}$.

Άσκηση 59. Ανατρέξτε πίσω στον ορισμό του τι είναι κανονική έκφραση. Αν σταθεροποιήσουμε το αλφάβητο σε, ας πούμε, $\Sigma = \{a, b\}$, δώστε μια CFG για τη γλώσσα των κανονικών εκφράσεων πάνω από το Σ.

5.3 Ένα ενδιαφέρον παράδειγμα με πλήρη απόδειξη

Ας δούμε τώρα ένα παράδειγμα μιας CFG με παραπάνω από ένα μη τερματικό σύμβολο. Η γραμματική αυτή θα έχει ως γλώσσα της όλες τις λέξεις του $\{a, b\}^*$ που έχουν ίδιο πλήθος από a και b. Θυμίζουμε εδώ ότι αυτή η γλώσσα δεν είναι κανονική (αποδεικνύεται εύκολα αυτό με χρήση του Λήμματος Άντλησης (Θεώρημα 8).

Η γραμματική G_1 είναι η ακόλουθη:

1. $S \to aB \mid bA \mid \epsilon$
2. $A \to aS \mid bAA$
3. $B \to aBB \mid bS$

Θεώρημα 12. *Η γλώσσα $L(G_1)$ απαρτίζεται από εκείνες τις λέξεις του $\{a, b\}^*$ με ίδιο πλήθος από a και b.*

Απόδειξη: Κατ' αρχήν τα μόνα τερματικά σύμβολα που εμφανίζονται στους κανόνες της G_1 είναι τα a και b, άρα η $L(G_1)$ είναι υποσύνολο του $\{a, b\}^*$.

Ας συμβολίζουμε για μια λέξη w του $\{a, b\}^*$ με $A(w)$ και $B(w)$ αντίστοιχα το πλήθος των a και b που περιέχει.

Πρέπει να δείξουμε ότι ισχύει $A(w) = B(w)$ αν και μόνο αν $S \overset{G_1}{\underset{*}{\Rightarrow}} w$. Αλλά αν προσπαθήσουμε να δείξουμε μόνο αυτό θα συναντήσουμε δυσκολίες. Παρ' ότι δείχνει αντιφατικό μας διευκολύνει το να δείξουμε παράλληλα και άλλες δύο προτάσεις. Έτσι θα δείξουμε με επαγωγή ως προς το μήκος της λέξης w τις εξής τρείς ισοδυναμίες.

1. $w \in L(S)$ αν και μόνο αν $A(w) = B(w)$.

2. $w \in L(A)$ αν και μόνο αν $A(w) = B(w) + 1$.

3. $w \in L(B)$ αν και μόνο αν $B(w) = A(w) + 1$.

Ο ρόλος δηλ. των μη τερματικών συμβόλων A και B στην G_1 είναι να παράγουν όλες τις λέξεις με ακριβώς ένα a παραπάνω (για το A) και ακριβώς ένα b παραπάνω (για το B).

Όπως είπαμε συμβολίζουμε με n το μήκος της λέξης w και κάνουμε επαγωγή ως προς n. Αν $n = 0$, τότε $w = \epsilon$ και w παράγεται από το S με τον τελευταίο κανόνα παραγωγής του S ενώ δεν παράγεται από τα A ή B, αφού όλοι οι κανόνες παραγωγής των A και B παράγουν μη κενές λέξεις. Βλέπουμε έτσι ότι ισχύουν και οι τρεις ισοδυναμίες για τη λέξη $w = \epsilon$, τη μοναδική λέξη με μήκος $n = 0$.

Υποθέτουμε τώρα επαγωγικά ότι ισχύουν και οι τρεις άνω ισοδυναμίες για κάθε λέξη w με $|w| \le n - 1$.

Έστω τώρα w μια λέξη με μήκος n. Αποδεικνύουμε την πρώτη ισοδυναμία: $w \in L(S) \iff A(w) = B(w)$.

$\underline{w \in L(S) \Longrightarrow A(w) = B(w)}$

Αφού $w \in L(S)$ υπάρχει μια ακολουθία παραγωγών της G_1 που αρχίζει από το S και καταλήγει στη w. Αν το πρώτο γράμμα της w είναι a τότε στην πρώτη παραγωγή αναγκαστικά χρησιμοποιείται ο κανόνας $S \to aB$. Αυτό συνεπάγεται ότι $w = ax$ όπου x είναι μια λέξη για την οποία ισχύει $x \in L(B)$, και $|x| \le n - 1$. Από την επαγωγική υπόθεση έπεται ότι $B(x) = A(x) + 1$ και συνεπώς $A(w) = B(w)$. Ομοίως, αν το πρώτο γράμμα της w είναι b τότε ο πρώτος κανόνας παραγωγής από το S στο w είναι αναγκαστικά ο $S \to bA$, άρα $w = by$, με $y \in L(A)$ και $|y| = n-1$. Από την επαγωγική υπόθεση $A(y) = B(y)+1$ από το οποίο προκύπτει $A(w) = B(w)$.

$\underline{A(w) = B(w) \Longrightarrow w \in L(S)}$

Αν το πρώτο γράμμα της w είναι a τότε $w = ax$ με $A(x) = B(x) - 1$, και $|x| = n - 1$. Από την επαγωγική υπόθεση προκύπτει ότι $x \in L(B)$, άρα υπάρχει τρόπος να παράγουμε το x από το μη τερματικό σύμβολο B. Αρχίζοντας τότε από το S, εφαρμόζουμε τον κανόνα παραγωγής $S \to aB$ και ακολούθως παράγουμε από το B το x. Συνολικά έχουμε παραγάγει έτσι το w από το S δείχνοντας σε αυτή την περίπτωση $w \in L(S)$. Ομοίως, αν το πρώτο γράμμα της w είναι το b τότε γράφεται $w = by$ με $|y| = n - 1$ και $A(y) = B(y) + 1$, άρα, με την επαγωγική υπόθεση, ισχύει $y \in L(A)$. Για να παραγάγουμε λοιπόν τη w από το S ξεκινούμε εφαρμόζοντας τον κανόνα $S \to bA$, και ακολούθως παράγουμε από το A το y, παίρνοντας έτσι τελικά το w.

Εδώ έχουμε δείξει το επαγωγικό βήμα για την πρώτη ισοδυναμία.

Δείχνουμε τώρα το επαγωγγικό βήμα για την ισοδυναμία $w \in L(A) \iff A(w) = B(w) + 1$. Προσέξτε ότι εδώ υπάρχει ασυμμετρία στην απόδειξη όσον αφορά το ρόλο που παίζει το a και το b ως πρώτο γράμμα της λέξης w.

$\underline{w \in L(A) \Longrightarrow A(w) = B(w) + 1}$

Έστω ότι το πρώτο γράμμα της λέξης w είναι το a. Τότε αναγκαστικά η πρώτη παραγαγωγή από το A στο w είναι η $A \to aS$, οπότε $w = ax$ με $|x| = n - 1$ και αναγκαστικά τότε το x πρέπει να είναι παραγόμενο από το S, άρα $x \in L(S)$ και από την επαγωγική υπόθεση $A(x) = B(x)$, από το οποίο προκύπτει ότι $A(w) = B(w) + 1$.

Αν το πρώτο γράμμα της w είναι το b τότε ο πρώτος κανόνας που εφαρμόζεται στην παραγωγή της w από το A είναι αναγκαστικά ο $A \to bAA$, οπότε έχουμε $w = bx_1x_2$ όπου $x_1, x_2 \in L(A)$, και, φυσικά, $|x_1|, |x_2| \le n-1$. Από την επαγωγική υπόθεση τώρα προκύπτει ότι $A(x_1) = B(x_1) + 1$ και $A(x_2) = B(x_2) + 1$, από τα οποία προκύπτει φυσικά ότι $A(w) = B(w) + 1$.

$\underline{A(w) = B(w) + 1 \Longrightarrow w \in L(A)}$

Αν το πρώτο γράμμα της w είναι το a τότε $w = ax$ με $A(x) = B(x)$, $|x| = n - 1$, οπότε από την επαγωγική υπόθεση προκύπτει $x \in L(S)$. Παράγοντας τώρα από το A με τον κανόνα παραγωγής $A \to aS$ και συνεχίζοντας παράγοντας από το S το x, καταλήγουμε σε μια ακολουθία παραγωγής του w από το A, δηλ. $w \in L(A)$.

Αν το πρώτο γράμμα της w είναι το b τότε $w = bx$ με $|x| = n - 1$ και $A(x) = B(x) + 2$. Ισχύει όμως το παρακάτω Λήμμα.

Λήμμα 1. *Αν για μια λέξη $x \in \{a, b\}^*$ ισχύει $A(x) = B(x) + 2$ τότε το x σπάει ως $x = x_1x_2$ όπου $A(x_1) = B(x_1) + 1$ και $A(x_2) = B(x_2) + 1$.*

Άσκηση 60. Δείξτε το Λήμμα 1.

Οπότε η λέξη x σπάει σε δυο κομμάτια x_1 και x_2, $x = x_1x_2$, με μήκος το πολύ $n-2$ η καθε μία και με $A(x_i) = B(x_i)+1$, $i = 1, 2$. Από την επαγωγική υπόθεση έχουμε τώρα $x_1, x_2 \in L(A)$. Για να παραγάγουμε λοιπόν από το A τη λέξη w αρχίζουμε με τον κανόνα παραγωγής $A \to bAA$ και ακολούθως αναπτύσσουμε το πρώτο A σε x_1 και το δεύτερο σε x_2, πετυχαίνοντας έτσι συνολικά μια παραγωγή της w από το A.

Παραλείπουμε την απόδειξη του επαγωγικού βήματος για την τρίτη ισοδυναμία μια και αυτή είναι εντελώς παρόμοια με τη δεύτερη ισοδυναμία, με τους ρόλους των a, b και A, B εναλλαγμένους.
□

5.4 Οι κανονικές γλώσσες είναι και context free

Θεώρημα 13. *Κάθε κανονική γλώσσα είναι και context free.*

Το αντίστροφο φυσικά δεν ισχύει όπως δείχνουν πολλά από τα παραδείγματα που έχουμε δει ως τώρα, π.χ. η γλώσσα $\{0^n1^n : \ n = 1, 2, \ldots\}$.

Απόδειξη. Έστω L κανονική γλώσσα. Αυτό σημαίνει ότι υπάρχει μια κανονική έκφραση r για τη γλώσσα. Αρκεί να δείξουμε ότι κάθε κανονική έκφραση περιγράφει μια context free γλώσσα.

Αρχίζουμε από τις απλούστερες κανονικές εκφράσεις και το δείχνουμε σταδιακά για όλες. Κάνουμε δηλ. επαγωγή ως προς το μήκος της κανονικής έκφρασης. Αν η κανονική έκφραση r έχει μήκος 1, τότε είναι αναγκαστικά ένα γράμμα του $\Sigma = \{\alpha_1, \ldots, \alpha_k\}$, το σύμβολο ϵ ή το σύμβολο $\emptyset$ (ανατρέξτε πίσω στον Ορισμό 27). Αν είναι γράμμα του Σ δηλ. $r = \alpha_j$ για κάποιο $j \in \{1, \ldots, k\}$, τότε η γλώσσα $L(r)$ είναι το μονοσύνολο $\{\alpha_j\}$, το οποίο δίνεται από την CFG $S \to \alpha_j$. Αν $r = \epsilon$ τότε $L(r) = \{\epsilon\}$ που δίνεται από την CFG $S \to \epsilon$, και αν $r = \emptyset$ τότε $L(r) = \emptyset$ και αυτό το σύνολο δίνεται από τη CFG $S \to S$, που προφανώς δεν παράγει τίποτα.

Αν τώρα το μήκος της έκφρασης r είναι μεγαλύτερο του 1 τότε, με βάση τον ορισμό των κανονικών εκφράσεων, η r είναι της μορφής

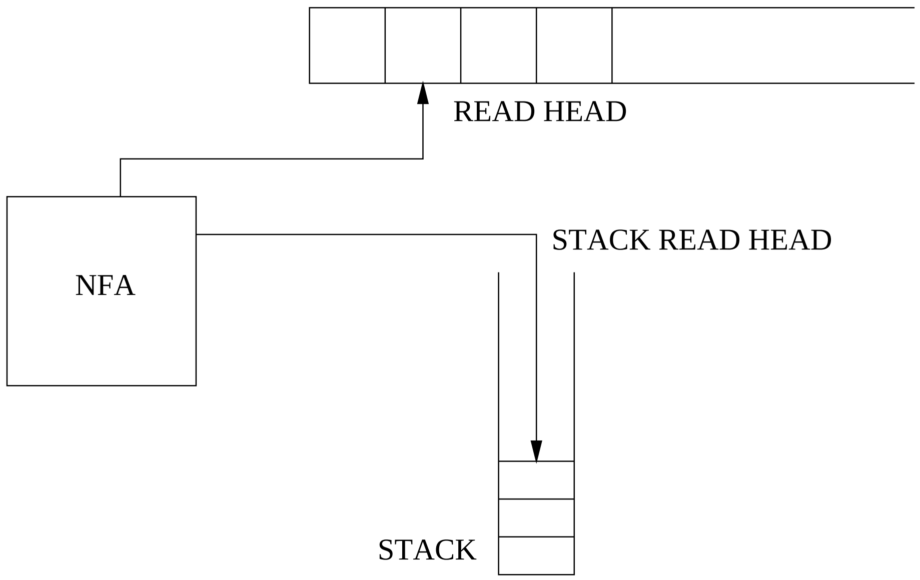
- $r = (st)$, με s, t κανονικές εκφράσεις μικρότερου μήκους. Από την επαγωγική υπόθεση υπάρχουν context free γραμματικές G_1 και G_2 τέτοιες ώστε $L(G_1) = L(s)$ και $L(G_2) = L(t)$. Για να φτιάξουμε μια CFG για τη γλώσσα $L(r) = L(s)L(t)$ μετονομάζουμε κατ΄ αρχήν όλα τα μη τερματικά σύμβολα της G_1 και της G_2 ώστε να είναι διαφορετικά μεταξύ τους και διαφορετικά από S, και ονομάζουμε τα δύο αρχικά μη τερματικά σύμβολα σε S_1 και S_2 για τις G_1 και G_2 αντίστοιχα. Η CFG για τη γλώσσα $L(r)$ έχει ως κανόνες όλους τους κανόνες της G_1 και της G_2 συν τον κανόνα $S \to S_1S_2$, που είναι και ο μόνος κανόνας για το αρχικό μη τερματικό σύμβολο S.
- $r = (s+t)$. Όπως και πριν όλα μόνο που ο επιπλέον κανόνας της γραμματικής δεν είναι τώρα ο $S \to S_1S_2$ αλλά οι δύο κανόνες $S \to S_1 \mid S_2$.
- $r = (s^*)$. Μετονομάζουμε το αρχικό μη τερματικό σύμβολο της CFG για το s σε S_1 (ή κάποιο άλλο όνομα αν αυτό υπάρχει ήδη στη γραμματική) και προσθέτουμε τον κανόνα $S \to \epsilon \mid S_1S$.

□

Άσκηση 61. Υπάρχει και άλλος τρόπος να δείξουμε ότι κάθε κανονική γλώσσα είναι κανονική. Μπορούμε να δουλέψουμε κατ΄ ευθείαν πάνω στο DFA ή NFA που αναγνωρίζει τη γλώσσα μας και να φτιάξουμε μέσω αυτού τη γραμματική. Διαλέξτε ένα από τα DFA ή NFA (η δουλειά είναι ουσιαστικά η ίδια) που εμφανίζονται σε αυτές τις σημειώσεις, π.χ. το αυτόματο που αναγνωρίζει τις λέξεις του $\{0, 1\}^*$ με περιττό πλήθος από μηδενικά και περιττό πλήθος από άσσους (Σχήμα 2). Αντιστοιχείστε ένα μη τερματικό σύμβολο σε κάθε κατάσταση του αυτομάτου και βρείτε πώς πρέπει να φτιάξετε τους κανόνες παραγωγής ώστε να προκύψει μια CFG που να περιγράφει την ίδια γλώσσα. Διατυπώστε ένα γενικό τρόπο ώστε να πηγαίνετε από DFA ή NFA σε ισοδύναμη (που να περιγράφει την ίδια γλώσσα) CFG, *χωρίς να περνάτε από την αντίστοιχη κανονική έκφραση.*

5.5 Το αυτόματο με στοίβα (Push Down Automaton)

Όπως ακριβώς οι κανονικές γλώσσες έχουν αντίστοιχες μηχανές, τα DFA, ή τα NFA ή τα ϵ-NFA, που αναγνωρίζουν ακριβώς το σύνολο των κανονικών γλωσσών, έτσι υπάρχει και μια μηχανή, το αυτόματο με στοίβα, ή Push Down Automaton (PDA) που έχει την ιδότητα ότι μια γλώσσα L είναι context free αν και μόνο αν υπάρχει PDA που την αναγνωρίζει. Δε θα δείξουμε αυτή την ισοδυναμία εδώ (όπως είχαμε κάνει για τις κανονικές γλώσσες και τα πεπερασμένα αυτόματα).



Σχήμα 19. Ένα αυτόματο με στοίβα

Ένα PDA (Σχήμα 19) αποτελείται από τρία μέρη

1. Ένα NFA που το βλέπουμε σα 'μονάδα ελέγχου' της μηχανής. Είναι σημαντικό εδώ να τονίσουμε ότι δε μπορούμε εδώ να αντικαταστήσουμε το NFA με DFA – <u>τα αυτόματα με στοίβα είναι μη ντετερμινιστικές μηχανές</u>.
2. Μια ταινία ανάγνωσης (read tape) που διαβάζεται από τον έλεγχο (NFA) με μια κεφαλή ανάγνωσης (read head) η οποία αρχίζει από τα αριστερά της ταινίας ανάγνωσης και κινείται μόνο προς τα δεξιά, και το πολύ κατά ένα σε κάθε βήμα (κύκλο) της μηχανής. Η προς αναγνώριση λέξη πρέπει να τοποθετηθεί εξ αρχής πάνω σε αυτή την ταινία από τον 'χρήστη' της μηχανής.
3. Μια στοίβα (stack). Η στοίβα είναι μια απεριόριστη ποσότητα μνήμης την οποία όμως μπορούμε να διαχειριστούμε με πολύ περιορισμένο τρόπο. Φανταζόμαστε τη στοίβα σα μια (αρχικά κενή) στήλη από θέσεις μνήμης, που αρχίζει από το επίπεδό μας και εκτείνεται απείρως προς τα πάνω. Σε κάθε θέση μνήμης μπορούμε να γράψουμε ένα οποιοδήποτε από τα γράμματα του αλφαβήτου μας ή ένα πεπερασμένο αριθμό από άλλα ειδικά σύμβολα που ενδεχομένως μας διευκολύνουν στον 'προγραμματισμό' του PDA. Στη στήλη αυτή μπορούμε να κάνουμε τις εξής τρεις πράξεις μόνο:
   - Μπορούμε να δούμε τα περιεχόμενα της πρώτης (από πάνω προς τα κάτω) θέσης μνήμης που είναι μη κενή. Αν η στοίβα είναι κενή μπορούμε να το διαπιστώσουμε αυτό.
   - Μπορούμε να προσθέσουμε κάτι στη στοίβα αλλά μόνο ακριβώς στη θέση μνήμης πάνω από την πρώτη (από πάνω προς τα κάτω) γεμάτη θέση μνήμης. Η πράξη αυτή αυξάνει λοιπόν το ύψος της στοίβας κατά 1.
   - Μπορούμε να διαγράψουμε την πιο πάνω θέση μνήμης (αν η στοίβα είναι κενή αυτή η πράξη δεν έχει κανένα αποτέλεσμα). Η πράξη αυτή ελαττώνει το ύψος της στοίβας κατά ένα αν η στοίβα δεν ήταν κενή.

Το NFA του αυτομάτου διαβάζει πάλι τα γράμματα της προς αναγνώριση λέξης ένα προς ένα, από αριστερά προς τα δεξιά, χωρίς και πάλι τη δυνατότητα να γυρίσει πίσω προς τα πίσω και να ξαναδιαβάσει την αρχή της λέξης. Και πάλι σε κάθε βήμα το NFA έχει τη δυνατότητα να εκτελέσει μια κίνηση ανάμεσα σε διάφορες δυνατές κινήσεις. Μια λέξη αναγνωρίζεται από το PDA αν κάποια επιλογή κινήσεων του NFA μπορεί να οδηγήσει σε αποδοχή της λέξης (το οποίο συμβαίνει αν στο τέλος το NFA είναι σε τελική κατάσταση).

Η σύνδεση του NFA με τη στοίβα γίνεται ως εξής: σε κάθε μετάβαση το NFA δεν επιλέγει το σύνολο των δυνατών επομένων καταστάσεων με κριτήριο μόνο το σε ποια κατάσταση βρίσκεται αυτό αλλά κοιτώντας ταυτόχρονα και ποιο είναι το περιεχόμενο της πάνω θέσης μνήμης της στοίβας (η μόνη θέση μνήμης που μπορεί άλλωστε να δει). Επίσης σε κάθε μετάβαση το NFA εκτελεί ενδεχομένως και μια από τις τρεις επιτρεπτές πράξεις στη στοίβα (διαβάζει την κορυφή της, προσθέτει κάτι στην κορυφή της, πετάει την πάνω θέση μνήμης χαμηλώνοντας την κορυφή της στοίβας κατά ένα, ή δεν κάνει τίποτα στη στοίβα).

Το μοντέλο του PDA είναι λειτουργικά ισοδύναμο με προγραμματισμό σε μια συνηθισμένη γλώσσα προγραμματισμού (π.χ. στην C) με τους εξής περιορισμούς και προσθήκες.

- Η μνήμη που χρησιμοποιεί το πρόγραμμά μας πρέπει να είναι συνολικά πεπερασμένη και γνωστή εξ' αρχής, ανεξάρτητα από το ποιο θα είναι το input (η προς αναγνώριση λέξη). Στην πράξη, αυτό το εξασφαλίζουμε δηλώνοντας μερικές μεταβλητές ακέραιου τύπου μόνο (int στη C) και λαμβάνοντας υπόψιν ότι κάθε τέτοια μεταβλητή δε μπορεί να μεγαλώσει απεριόριστα, έχει δηλ. συγκεκριμένο αριθμό από bits (δυαδικά ψηφία) που αντιστοιχούν σε αυτή.

- Η πρόσβαση στο input γίνεται μέσω της συνάρτησης `int INP();` η οποία κάθε φορά που καλείται επιστρέφει και το επόμενο γράμμα της λέξης που εξετάζουμε. Όταν έχει διαβάσει όλα τα γράμματα επιστρέφει από κει και πέρα την ειδική τιμή -1.

- Υπάρχουν ακόμη άλλες τρεις 'συναρτήσεις βιβλιοθήκης' με τις οποίες γίνεται ο χειρισμός της στοίβας, και μια συνάρτηση ακόμη με την οποία μπαίνει μέσα στη γλώσσα ο μη ντετερμινισμός που είναι απαραίτητος στο PDA. Οι συναρτήσεις για τη στοίβα είναι οι εξής:

`int READ();`	Επιστρέφει τα περιεχόμενα της πάνω θέσης μνήμης της στοίβας ή -1 αν η στοίβα είναι άδεια.
`POP();`	αδειάζει μια θέση μνήμης από τη στοίβα ή δεν κάνει τίποτα αν η στοίβα είναι ήδη άδεια
`PUSH(int x);`	Γράφει τον αριθμό `x` στην κορυφή της στοίβας, σε μια νέα θέση μνήμης

- Η συνάρτηση `NF` μέσω της οποίας κωδικοποιείται η μη ντετερμιστική συμπεριφορά περιγράφεται στην επόμενη παράγραφο.

Μη ντετερμινιστικά 'προγράμματα'

Γενικά ο μηχανισμός του μη ντετερμινισμού χρησιμοποιείται σε προβλήματα αποφάσεων, σε ερωτήματα δηλαδή που φιλοδοξούμε να λύσουμε υπολογιστικά και τα οποία επιδέχονται ΝΑΙ ή ΟΧΙ απάντηση, όπως ακριβώς είναι και το πρόβλημα που μας ενδιαφέρει εδώ, να αποφασίζουμε δηλ. αν μια λέξη που μας δίνεται ανήκει σε μια γλώσσα L ή όχι.

Η συνάρτηση μέσω της οποίας 'υλοποιείται' ο μη ντετερμινισμός είναι η `int NF(int x);` η οποία επιστρέφει *μη ντετερμινιστικά* μια από τις επιλογές $0, 1, \ldots, x-1$. Το νόημα της συνάρτησης αυτής είναι ότι εμείς δεν έχουμε κανένα έλεγχο στο ποια από τις δυνατές επιλογές αυτή επιλέγει, αλλά πρέπει να γράψουμε το πρόγραμμά μας με τέτοιο τρόπο ώστε *αν αυτή η συνάρτηση επιστρέψει κατά τις κλήσεις της τις κατάλληλες επιλογές* τότε να κάνουμε τη λέξη δεκτή, αν αυτή είναι μέσα στη γλώσσα. Αλλά, δεν πρέπει να σε καμία περίπτωση να κάνουμε δεκτή μια λέξη που δεν ανήκει στη γλώσσα όποιες τιμές κι αν επιστρέψει η `NF`.

Για να γίνει κατανοητός ο τρόπος χρήσης της `NF` δείχνουμε παρακάτω μια συνάρτηση σε C η οποία παίρνει δύο ορίσματα `number` και `vector` και επιστρέφει 1 αν ο ακέραιος `number` περιέχεται στον πίνακα `vector` (ο οποίος έχει 1000 στοιχεία, τα `vector[0], ... , vector[999]`) και 0 αν δεν περιέχεται.

```
int NumberInVector(int number, int vector[1000])
{   int location;
    location = NF(1000);
```

```
    if(vector[location]==number) return 1;
    return 0;
}
```

Πρέπει εδώ να ξεκαθαρίσουμε ότι η συνάρτηση `NumberInVector` λύνει το πρόβλημα που θέλουμε υπό την εξής έννοια:

- Αν ο αριθμός `number` είναι μέσα στον πίνακα `vector` τότε υπάρχει τρόπος να απαντήσει η `NF` ώστε ο αλγόριθμός μας να δώσει τη σωστή απάντηση, δηλ. 1.
- Αν ο αριθμός `number` δεν είναι μέσα στον πίνακα `vector` τότε, ό,τι και να απαντήσει η `NF`, ο αλγόριθμός μας δεν υπάρχει περίπτωση να κάνει λάθος (αφού ελέγχει αν όντως βρίσκεται ο `number` στη θέση `vector[location]` πριν απαντήσει) και απαντάει πάντα 0.

Αν δεν είχαμε στη διάθεσή μας τη συνάρτηση `NF` θα είμασταν αναγκασμένοι, λίγο-πολύ, να κάνουμε χίλιους ελέγχους για να διαπιστώσουμε αν ο δεδομένος αριθμός είναι μέσα στον πίνακα ή όχι. Παρατηρείστε ότι τώρα δεν υπάρχει ανακύκλωση κανενός είδους μέσα στη συνάρτηση και ότι η απάντηση βρίσκεται σε σταθερό χρόνο! Ο αλγόριθμος της συνάρτησης `NumberInVector` 'μαντεύει' τη θέση στην οποία βρίσκεται στο `vector` ο αριθμός και απλά ελέγχει μετά αν είναι όντως έτσι πριν απαντήσει 1 ή 0, ώστε να αποφύγει να κάνει λάθος στην περίπτωση που ο αριθμός δεν είναι μέσα. Ο αλγόριθμος αυτός δηλ. δεν υπάρχει περίπτωση να απαντήσει 1 ενώ η απάντηση είναι 0, αλλά μπορεί κάλλιστα να απαντήσει 0 όταν η απάντηση είναι 1. Αυτή η ασυμμετρία είναι πολύ χαρακτηριστική στους μη ντετερμινιστικούς αλγορίθμους.

Πρέπει να είναι φανερό με αυτό το παράδειγμα, ότι μη ντετερμινιστικοί αλγόριθμοι δεν μπορούν να υλοποιηθούν. Είναι απλά ένα χρήσιμο θεωρητικό κατασκεύασμα.

5.6 Παραδείγματα PDA

Η γλώσσα $L_1 = \{0^n 1^n : \quad n = 1, 2, \ldots\}$

Η γλώσσα $L_1 = \{0^n 1^n : \quad n = 1, 2, \ldots\}$ είναι context free (η γραμματική είναι: $S \to \epsilon \mid 0S1$). Θα δώσουμε εδώ ένα PDA για τη γλώσσα αυτή. Ισοδύναμα, θα περιγράψουμε το PDA αυτό σα μια συνάρτηση σε C, σύμφωνα με αυτά που είπαμε στην προηγούμενη παράγραφο. Η συνάρτηση αυτή θα μπορεί να χρησιμοποιεί σταθερή σε μέγεθος μνήμη και θα έχει πρόσβαση στη στοίβα μέσω των συναρτήσεων `READ, POP, PUSH` που ορίσαμε στην προηγούμενη παράγραφο. Τυγχάνει για τη γλώσσα αυτή να μη χρειάζεται ο μη ντετερμινισμός οπότε δε θα χρησιμοποιήσουμε τη συνάρτηση `NF`.

Το πρόγραμμα-PDA φαίνεται παρακάτω. Η συνάρτηση `f` επιστρέφει 1 αν η λέξη (που τη διαβάζουμε με διαδοχικές κλήσεις στη συνάρτηση `INP()`) ανήκει στην L_1 και 0 αλλιώς. Η στρατηγική της συνάρτησης είναι η εξής. Όσο διαβάζουμε μηδενικά τα σπρώχνουμε πάνω στη στοίβα. Για κάθε άσσο που διαβάζουμε πετάμε κάτι από την στοίβα. Επίσης προσέχουμε ποτέ να μη διαβάσουμε 0 μετά που έχουμε διαβάσει κάποιο 1. Αν στο τέλος η στοίβα καταλήξει άδεια δεχόμαστε τη λέξη.

Ότι ακολουθεί το σύμβολο // αποτελεί σχόλιο και δεν είναι τμήμα του προγράμματος.

```
int f()
{
    int i, SeenOne=0;
    a: i = INP();
    if(i==0) {  //Εδώ διαβάζουμε το επόμενο γράμμα
     if(SeenOne) return 0; //0 μετά από 1 απορρίπτεται
     PUSH(0); goto a; //Μηδενικά σπρώχνονται στη στοίβα
    }
    if(i==1) {
     SeenOne=1;
```

```
  if(-1 == READ()) return 0; //΄Αδειασε η στοίβα πριν την ώρα της
  POP(); goto a; //Για κάθε 1 πετάμε ένα 0 από τη στοίβα
 }
 if(-1 == READ()) return 1; //Αν έχει τελειώσει η λέξη και έχει αδειάσει η στοίβα, δεχόμαστε
}
```

Παρατηρείστε ότι, πέρα από τη στοίβα, το πρόγραμμα αυτό χρησιμοποιεί πεπερασμένη και προκαθορισμένη μνήμη. Για την ακρίβεια χρησιμοποιεί όλες κι όλες δύο μεταβλητές, τις `i` (που χρησιμοποιείται για να κρατάει το επόμενο γράμμα ή -1) και `SeenOne` (που φροντίζουμε να είναι 0 όσο δεν έχουμε ακόμη δει άσσο και μετά να είναι 1). Η μεταβλητή `i` παίρνει τρεις διαφορετικές τιμές, αρκούν δηλ. 2 λογικά bits για να την αποθηκεύσουμε (μια και με αυτά κωδικοποιούμε 4 διαφορετικές τιμές), ενώ για τη μεταβλητή `SeenOne` που παίρνει δύο τιμές αρκεί ένα λογικό bit.

Η γλώσσα $L_2 = \{xx^R : \ \ x \in \{0,1\}^*\}$

Η γλώσσα $L_2 = \{xx^R : \ \ x \in \{0,1\}^*\}$ (x^R είναι η λέξη x γραμμένη ανάποδα) εύκολα φαίνεται ότι είναι context free. Μια CFG γι΄ αυτήν είναι απλούστατα η

$$S \to \epsilon \mid 0S0 \mid 1S1.$$

Παρακάτω δείχνουμε μια συνάρτηση `g` η οποία υλοποιεί ένα PDA για την αναγνώριση της γλώσσας L_2. Εδώ χρησιμοποιείται η συνάρτηση `NF`, πρόκειται δηλ. για ένα μη ντετερμινιστικό πρόγραμμα. Η στρατηγική είναι η εξής. Μέχρι τη μέση της ανάγνωσης της λέξης σπρώχνουμε τα γράμματα όπως τα διαβάζουμε πάνω στη στοίβα. Από τη μέση και πέρα για κάθε γράμμα που διαβάζουμε από το input πετάμε και ένα γράμμα από τη στοίβα και το συγκρίνουμε με αυτό που διαβάσαμε από το input. (Προσέξτε ότι τα γράμματα θα πεταχτούν από τη στοίβα με αντίστροφη σειρά από αυτή με την οποία γράφτηκαν.) Αν είναι ίδια συνεχίζουμε (μέχρι να τελειώσει το input) αλλιώς απορρίπτουμε τη λέξη.

Η σημαντική παρατήρηση εδώ είναι ότι δεν μπορούμε να ξέρουμε πότε έχουμε φτάσει τη μέση της λέξης εισόδου! Αυτή τη βοήθεια αναλαμβάνει να μας δώσει η συνάρτηση `NF()`, η οποία μας λέει ακριβώς αυτό. Στο παρακάτω πρόγραμμα κάθε κλήση στην `NF()` κατά τη διάρκεια της εκτέλεσης του προγράμματος ισοδυναμεί με μια ερώτηση (στο Θεό, αν προτιμάτε) για το αν φτάσαμε τη μέση της λέξης ή όχι. Αν ο Θεός μας βοηθήσει με τη σωστή απάντηση θα δεχτούμε τη λέξη, μόνο όμως αν πρέπει να τη δεχτούμε. Δε μπορεί δηλ. να μας κοροϊδέψει με τρόπο ώστε να δεχτούμε μια λέξη ενώ δε θα έπρεπε (επαναλαμβάνουμε ότι είναι πολύ σημαντική αυτή η ασυμμετρία στα μη ντετερμινιστικά προγράμματα).

```
int g()
{
 int i, r, mid=0;
 a:  i = INP(); //Εδώ διαβάζουμε το επόμενο γράμμα
 if(mid == 0) mid = NF(2); //΄Εχουμε περάσει τη μέση της λέξης;
 if(mid == 0) { //΄Οχι, δε την περάσαμε ακόμη
  PUSH(i); goto a;
 }
 else { //΄Εχουμε περάσει τη μέση
  r = READ(); //Διάβασε την κορυφή της στοίβας
  if(r == -1) return 0; //΄Αδεια στοίβα, απόρριψη
  POP();
  if(r != i) return 0; //Δεν ταιριάζουν, απόρριψη
  goto a; //Ταιριάζουν, συνέχισε
 }
 if(-1 == READ()) return 1; //Κανένα πρόβλημα και στοίβα άδεια, άρα δεχόμαστε
}
```

Η γραμμή του παραπάνω προγράμματος όπου χρησιμοποιείται ο μη ντετερμινισμός είναι η `if(mid == 0) mid = NF(2);`, όπου (αν η μεταβλητή `mid` δεν έχει ήδη γίνει μια φορά 1, οπότε και μένει για πάντα από κει και πέρα) θέτουμε τη μεταβλητή `mid` σε 0 ή 1 με μια κλήση στην `NF(2)`. Όταν η `NF` επιστρέψει 1 θεωρούμε από κει και πέρα ότι έχουμε περάσει τη μέση, και προσπαθούμε με το πρόγραμμα να δούμε αν, με αυτή την υπόθεση μπορούμε να δεχτούμε τη λέξη.

Το πρόγραμμα αυτό χρησιμοποιεί τρεις μεταβλητές κάθε μια από τις οποίες παίρνει τιμές μέσα στο σύνολο $\{0, 1, -1\}$, οπότε δύο λογικά bits αρκούν για την αποθήκευση κάθε μιας από αυτές.

5.7 Το Λήμμα Άντλησης για context free γλώσσες, και η εφαρμογή του

Όπως και στην περίπτωση των κανονικών γλωσσών υπάρχει και για τις context free γλώσσες, ένα 'Λήμμα Άντλησης' που είναι πολύ χρήσιμο στο να αποδεικνύουμε ότι ορισμένες γλώσσες δεν είναι context free. Η μορφή του είναι πολύ παρόμοια με το Λήμμα Άντλησης για κανονικές γλώσσες κα ο τρόπος χρήσης του επίσης.

Θεώρημα 14. (Λήμμα Άντλησης για context free γλώσσες)
Έστω ότι η γλώσσα L είναι context free. Τότε υπάρχει ένας φυσικός αριθμός n τέτοιος ώστε για κάθε λέξη $z \in L$ με $|z| \ge n$ να μπορούμε να γράψουμε

$$z = uvwxy,$$

όπου

(α) $|vx| \ge 1$,

(β) $|vwx| \le n$, *και*

(γ) για κάθε $i \ge 0$ *ισχύει* $uv^iwx^iy \in L$.

Δε θα δώσουμε απόδειξη του θεωρήματος αυτού, αλλά θα δούμε πώς χρησιμοποιείται για να δείξουμε ότι μια γλώσσα δεν είναι context free.

Η γλώσσα $L_1 = \{a^nb^nc^n : \quad n \ge 0\}$ δεν είναι context free

Ας υποθέσουμε ότι η L_1 είναι context free. Έστω τότε n ο αριθμός που αναφέρεται στο Θεώρημα 14 και $z = a^{2n}b^{2n}c^{2n} \in L_1$. Από το Θεώρημα 14 η λέξη z γράφεται ως $z = uvwxy$ όπου ισχύουν τα συμπεράσματα του Θεωρήματος. Παρατηρούμε ότι η λέξη vwx, που σύμφωνα με το Θεώρημα έχει μήκος το πολύ n, δε μπορεί να περιέχει και a και c, ακριβώς επειδή το μήκος της δε φτάνει να 'γεφυρώσει' τα b της λέξης z. Χωρίς βλάβη της γενικότητας υποθέτουμε ότι από τη λέξη αυτή λείπουν τα a (και το επιχείρημα είναι εντελώς παρόμοιο αν λείπουν τα c). Από το Θεώρημα 14 έπεται ότι η λέξη $uwy \in L_1$ (εφαρμόσαμε το Θεώρημα με $i = 0$). Αλλά για να πάμε από τη λέξη z στην uwy σβήσαμε το u και το x, τα οποία δεν έχουν a μέσα, άρα το πλήθος των a στη λέξη uwy έχει παραμείνει $2n$. Το μήκος της uwy όμως είναι αυστηρά μικρότερο του $6n$ (εδώ χρησιμοποιούμε το (α) από τα συμπεράσματα του Θεωρήματος), άρα δε μπορεί η λέξη uwy να ανήκει στην L_1, όως είχαμε υποθέσει. Άρα η L_1 δεν είναι context free.

Άσκηση 62. Δείξτε ότι η γλώσσα $\{xx : \quad x \in \{0, 1\}^*\}$ δεν είναι context free.

Άσκηση 63. Δείξτε ότι η γλώσσα $\{(ab)^k(cd)^k : \quad k \in \mathbf{N}\}$ δεν είναι κανονική.