ΚΕΦΑΛΑΙΟ 3 ΣΥΝΤΑΚΤΙΚΑ ΣΤΟΙΧΕΙΑ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Στόχος

Στόχος του Κεφαλαίου αυτού είναι να περιγράψει με σύντομο τρόπο κάποια βασικά στοιχεία γλωσσών προγραμματισμού τύπου Context-free (Γραμματικές χωρίς συμφραζόμενα) μια και στη συνέχεια του βιβλίου θα αναφερόμαστε μόνο σε τέτοιες γλώσσες.

Προσδοκώμενα αποτελέσματα

'Οταν θα έχετε μελετήσει το κεφάλαιο αυτό θα μπορείτε να:

- εξηγήσετε γιατί χρησιμοποιούμε Context-free γραμματικές στην σύνταξη των γλωσσών προγραμματισμού
- αιτιολογήσετε γιατί οι Context-free γραμματικές είναι ισχυρότερες από τις Κανονικές Εκφράσεις ή τις Κανονικές γραμματικές
- δείξετε ότι μια γραμματική είναι διφορούμενη
- αποδιφοροποιήσετε μια διφορούμενη γραμματική
- περιγράψετε πώς παράγονται τα δένδρα ανίχνευσης ενός Συντακτικού Αναλυτή
- δώσετε τους τυπικούς ορισμούς του Chomsky για γραμματικές τύπου 0, 1,2, και 3.

Έννοιες κλειδιά

Context-free γραμματικές, γραμματικές Chomsky, διφορούμενες γραμματικές, συντακτικές κατηγορίες.

Εισαγωγικές παρατηρήσεις

Είδαμε στο προηγούμενο κεφάλαιο ότι η λεκτική δομή των tokens μπορεί να προσδιοριστεί με Κανονικές Εκφράσεις (ΚΕ) και ότι από τις ΚΕ μπορούμε αυτόματα να κατασκευάσουμε ένα Λεκτικό Αναλυτή που αναγνωρίζει τις λεκτικές δομές (tokens) μιας γλώσσας.

Με παρόμοιο τρόπο θα χειριστούμε στα επόμενα κεφάλαια την Συντακτική Ανάλυση (ΣΑ). Για τον συντακτικό καθορισμό μιας γλώσσας προγραμματισμού θα χρησιμοποιήσουμε ένα συμβολισμό που καλείται BNF (Backus-Naur-Form), και θα ασχοληθούμε με γραμματικές τύπου Context-Free ή αλλοιώς γραμματικές Chomsky τύπου 2 (τις γραμματικές αυτές θα τις ορίσουμε στην ενότητα 3.5).

Θα περιγράψουμε πώς μια γραμματική ορίζει μια γλώσσα, και θα εξετάσουμε ποιά χαρακτηριστικά γλωσσών προγραμματισμού, μπορούν ή δεν μπορούν να οριστούν από γραμματικές τύπου Context-free. Δεν πρόκειται να ασχοληθούμε με την θεωρία τυπικών (formal) γλωσσών προγραμματισμού, μια και αυτό είναι το αντικείμενο της Θεματικής Υποενότητας 'Αυτόματα και Τυπικές Γλώσσες' αλλά θα αναφέρουμε μόνο εκείνα τα στοιχεία που είναι απολύτως απαραίτητα για να κατανοήσετε την λειτουργία της Συντακτικής Ανάλυσης.

DRAFT

ΕΝΟΤΗΤΑ 3.1 ΓΡΑΜΜΑΤΙΚΕΣ CONTEXT-FREE

Είναι σύνηθες να ορίζουμε μερικές δομές γλωσσών προγραμματισμού με ένα αναδρομικό τρόπο (recursively). Για παράδειγμα, εάν E σημαίνει **έκφραση** (expression) και S_i σημαίνει **εντολή** (statement) τότε τα παρακάτω είναι επίσης εντολές (τα E_i είναι επίσης εκφράσεις).

if E **then** S_1 **else** S_2 (1)

begin S_1 **;** S_2 **;…;** S_n **end** (2)

Μια έκφραση είναι και η

$E_1 + E_2$ (3)

Αν χρησιμοποιήσουμε τις συντακτικές κατηγορίες "statement", και "expression" στην κλάση των εντολών και των εκφράσεων αντίστοιχα, τότε η (1) μπορεί να γραφτεί σύμφωνα με τον **Κανόνα** (Production -Rewriting Rule):

statement → **if** expression **then** statement **else** statement (4)

η δε (3) μπορεί να γραφτεί σύμφωνα με τον κανόνα:

expression → expression + expression

Η (2) όμως παρουσιάζει πρόβλημα αν γραφτεί σαν:

statement → **begin** statement **;** . . . **;** statement **end**

διότι χρειαζόμαστε κάθε κανόνας να έχει ακριβή αριθμό από σύμβολα.

Για να γραφτεί η (2) σωστά χρειάζεται να εισάγουμε άλλη μια συντακτική κατηγορία την οποία θα ονομάσουμε "statement-list". Η (2) τώρα μπορεί να περιγραφεί από τους εξής δυο κανόνες:

statement → **begin** statement-list **end**

statement-list → statement (5)

| statement; statement-list

Όπου το σύμβολο "|" σημαίνει είτε. Δηλαδή ο δεύτερος κανόνας της (5) έχει δυο εναλλακτικά δεξιά μέρη.

Ένα σύνολο από κανόνες όπως η (5) είναι ένα παράδειγμα γραμματικής. Μια γραμματική περιλαμβάνει 4 οντότητες: **Τερματικά** σύμβολα, **Μη Τερματικά** σύμβολα, το **Αρχικό σύμβολο** και ένα σύνολο από **Κανόνες** (Terminals, Nonterminals, Start Symbol, Productions).

Τερματικά σύμβολα είναι τα βασικά σύμβολα που σχηματίζουν τις συμβολοσειρές της γλώσσας. Τα tokens και τα τερματικά σύμβολα είναι συνώνυμα στις γλώσσες προγραμματισμού. Π.χ. **begin**, **end**, **;**, + είναι τερματικά σύμβολα.

Μη Τερματικά σύμβολα (συνώνυμα με τις συντακτικές κατηγορίες και τις συντακτικές μεταβλητές), είναι ειδικά σύμβολα που υποδηλώνουν συμβολοσειρές οι οποίες εκφράζουν την συντακτική δομή της γλώσσας (πχ το statement στον κανόνα: statement → **begin** statement-list **end**).

Ένα από τα Μη Τερματικά σύμβολα έχει επιλεγεί σαν Αρχικό σύμβολο και υποδηλώνει την γλώσσα.

Κανόνες είναι οι συντακτικοί κανόνες που καθορίζουν τους τρόπους με τους οποίους οι συντακτικές κατηγορίες μπορούν να κατασκευαστούν η μια από την άλλη και από τα Τερματικά σύμβολα. Οι κανόνες συνίστανται από ένα Μη Τερματικό σύμβολο ακολουθούμενο από ένα → (ή

DRAFT

::=) και ακολουθούμενο από μια συμβολοσειρά αποτελούμενη από Τερματικά και Μη Τερματικά σύμβολα.

Με τον συμβολισμό BNF (Backus-Naur-Form), η (5) περιγράφεται από τους παρακάτω κανόνες

statement → **begin** statement-list **end**
statement-list → statement
statement-list → statement**;** statement-list

Παράδειγμα 1 / Κεφ. 3. Μια Context-free γραμματική η οποία μπορεί να περιγράψει απλές αριθμητικές εκφράσεις είναι η (6) η οποία έχει σαν Μη Τερματικά σύμβολα τα Ε και Α, με το Ε να είναι το Αρχικό σύμβολο. Τα δε Τερματικά σύμβολα είναι τα:

id + - * / ↑ ( )

Τέλος οι Κανόνες είναι οι: (6)

E → E A E | **(**E**)** | **-** E | **id**
A → **+** | **-** | ***** | **/** | **↑**

Οι περισσότεροι συμβολισμοί που θα συναντήσουμε στη συνέχεια του βιβλίου είναι προφανείς, εκτός των εξής συμβάσεων:

- Κεφαλαία γράμματα από το τέλος της Λατινικής αλφαβήτου (X,Y,Z κλπ) παριστάνουν είτε Μη Τερματικά είτε Τερματικά.
- Μικρά γράμματα από το τέλος της Λατινικής αλφαβήτου (u,v,...,z) παριστάνουν συμβολοσειρές από Τερματικά.
- Μικρά Ελληνικά (α,β,γ) παριστάνουν συμβολοσειρές από σύμβολα της γραμματικής. (δηλαδή συμβολοσειρές από Τερματικά ή/και Μη Τερματικά σύμβολα) Π.χ. $A\rightarrow\alpha_1$

Η σειρά από τους Κανόνες
$A\rightarrow \alpha_1$, $A\rightarrow\alpha_2$,..., $A\rightarrow \alpha_k$ (A- Κανόνες)
συντομεύεται στην
$A\rightarrow\alpha_1 |\alpha_2 |...|\alpha_k$, όπως έγινε στο (6).

3.1.1 ΔΥΝΑΤΟΤΗΤΕΣ ΤΩΝ CONTEXT-FREE ΓΡΑΜΜΑΤΙΚΩΝ (CF)

Οι Context-free γραμματικές μπορούν να περιγράψουν την περισσότερη, αλλά όχι ολόκληρη την σύνταξη των γλωσσών προγραμματισμού (A. V. Aho, R. Sethi, J. D. Ullman, Compilers: Principles techniques, and Tools, Addison-Wesley, 1986). Είδαμε ότι οι Κανονικές Εκφράσεις περιγράφουν την σύνταξη των tokens. Οποιαδήποτε συντακτική δομή περιγράφεται από Κανονικές Εκφράσεις μπορεί επίσης να περιγραφεί και από κάποια Context-free γραμματική.

Παράδειγμα 2 / Κεφ. 3. Κανονική Έκφραση:

(a | b) (a | b | 0 | 1)*

και η CF γραμματική:

S→aA | bA
A→aA | bA | 0A | 1A | ε

DRAFT

περιγράφουν την ίδια γλώσσα διότι και οι δυο δημιουργούν τις ίδιες συμβολοσειρές τερματικών συμβόλων.

Διάφοροι λόγοι μας αναγκάζουν να χρησιμοποιούμε Κανονικές Εκφράσεις αν και θα μπορούσαμε να χρησιμοποιήσουμε Context-free γραμματικές αντ' αυτών. Πρώτον οι λεκτικοί κανόνες είναι απλοί και δεν χρειάζεται συμβολισμός τύπου Context-free γραμματικών οι οποίοι είναι δυσκολότεροι να γίνουν αντιληπτοί. Δεύτερον είναι ευκολότερο να κατασκευάσουμε αναγνωριστές (recognizers) από τις Κανονικές Εκφράσεις παρά από Context-free γραμματικές.

Δεν υπάρχουν ακριβείς οδηγίες για το τι πρέπει να περιληφθεί στους λεκτικούς κανόνες και τι στους συντακτικούς. Οι Κανονικές Εκφράσεις είναι πιο κατάλληλες για να περιγράφουν τη δομή των λεκτικών κατασκευών (identifiers, constants, keywords, κλπ), ενώ οι Context-free γραμματικές είναι απαραίτητες για την περιγραφή απεριόριστα φωλιασμένων δομών (**begin - end**, **if - then - else**, κλπ).

Άσκηση Αυτοαξιολόγησης 1 / Κεφ. 3

Μπορείτε να εξηγήσετε πού και γιατί χρησιμοποιούνται οι Κανονικές Εκφράσεις και πού οι Context Free γραμματικές στις γλώσσες προγραμματισμού**;**

ΕΝΟΤΗΤΑ 3.2 ΔΗΜΙΟΥΡΓΙΑ ΤΩΝ ΔΕΝΔΡΩΝ ΑΝΙΧΝΕΥΣΗΣ

Στην γραμματική $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$ (7)

όπου Ε σημαίνει έκφραση, ο Κανόνας $E \rightarrow -E$ δηλώνει ότι μια έκφραση με μείον μπροστά της είναι επίσης μια έκφραση. Αυτό περιγράφεται από την σχέση

$E \Rightarrow -E$ που διαβάζεται: "η Ε παράγει (derives) - Ε".

Η $E \rightarrow (E)$ λέει ότι μπορούμε να αντικαταστήσουμε μια εμφάνιση της Ε σε οποιαδήποτε συμβολοσειρά από σύμβολα της γραμματικής, με (Ε).

Μια άλλη περίπτωση είναι η:

$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(\mathbf{id})$

μια τέτοια ακολουθία αντικαταστάσεων ονομάζεται ***παραγωγή*** (derivation) του **-(id)** από το Ε.

Σε πιο αφηρημένη μορφή λέμε ότι $\alpha A \beta \Rightarrow \alpha\gamma\beta$ αν $A \rightarrow \gamma$ είναι ένας Κανόνας και α και β είναι αυθαίρετες συμβολοσειρές από σύμβολα της γραμματικής.

Αν $\alpha_1 \Rightarrow \alpha_2 \Rightarrow ... \Rightarrow \alpha_n$, λέμε ότι η α_1 ***παράγει*** α_n .

Το σύμβολο $\Rightarrow$ μεταφράζεται σε "παράγει σε ένα βήμα", ενώ το $\Rightarrow^*$ μεταφράζεται σε "παράγει σε μηδέν ή περισσότερα βήματα".

Έτσι ισχύει ότι:

1. $\alpha \Rightarrow^* \alpha$ $\forall$ συμβολοσειρά α ($\forall$ = για κάθε)
2. αν $\alpha \Rightarrow^* \beta$ και $\beta \Rightarrow \gamma$, τότε $\alpha \Rightarrow^* \gamma$

Ακόμη $\alpha \Rightarrow^+ \beta$ σημαίνει "το α παράγει β σε ένα ή περισσότερα βήματα".

Δοθείσης μιας context-free γραμματικής G με Αρχικό σύμβολο S μπορούμε να χρησιμοποιήσουμε την σχέση $\Rightarrow^+$ για να ορίσουμε την **L(G) (την γλώσσα που δημιουργείται από την G).** Οι συμβολοσειρές της L(G) μπορούν να περιέχουν μόνο Τερματικά σύμβολα της G. Λέμε ότι μια συμβολοσειρά w από Τερματικά ανήκει στην L(G) εάν και μόνο εάν $S \Rightarrow^+ w$. Η

DRAFT

συμβολοσειρά w καλείται **πρόταση** (sentence**)** της G. Εάν S ⇒* α, όπου το α μπορεί να περιέχει και Μη Τερματικά, τότε λέμε ότι το α είναι μια **προτασιακή μορφή** (sentential form) της γραμματικής G.

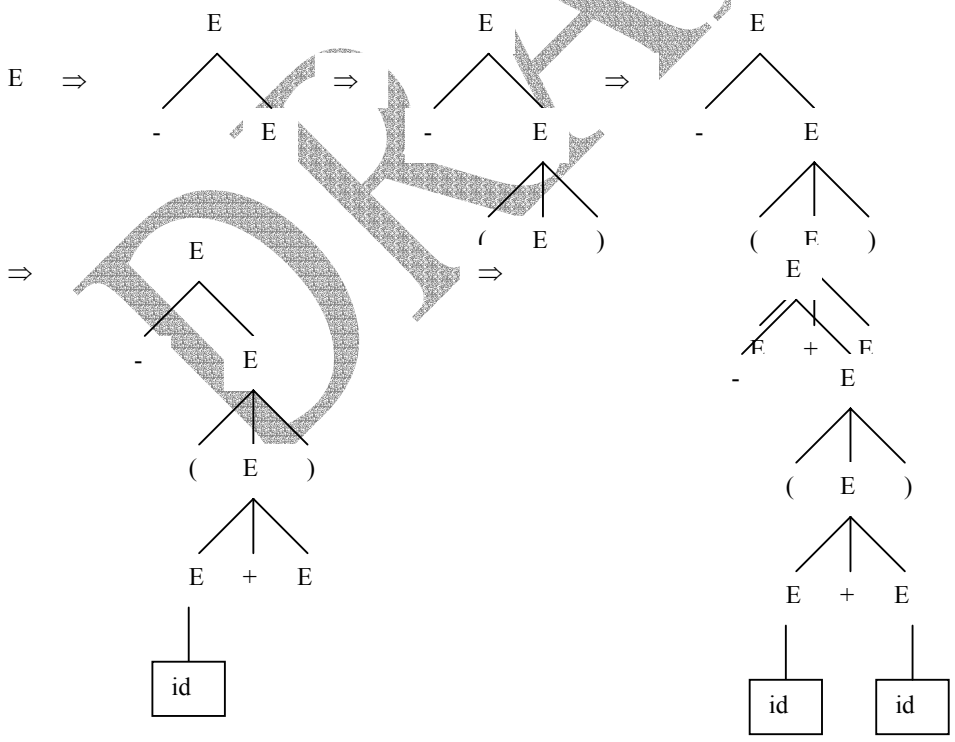
Άσκηση Αυτοαξιολόγησης 2 / Κεφ. 3

Μπορείτε να εξηγήσετε γιατί η συμβολοσειρά **-(id**+ **id**) είναι μια από τις προτάσεις της γλώσσας που προκύπτει από την γραμματική (7)**;**

Για να καταλάβουμε καλύτερα πώς λειτουργούν μερικοί Συντακτικοί Αναλυτές θα θεωρήσουμε παραγωγές (derivations) στις οποίες μόνο το αριστερότερο Μη Τερματικό σύμβολο μιας προτασιακής μορφής αντικαθίσταται σε κάθε βήμα. Τέτοιες παραγωγές καλούνται **αριστερές παραγωγές** (Leftmost Derivations). Ανάλογα ορίζονται και **οι δεξιές παραγωγές** (Rightmost Derivations).

Μια γραφική απεικόνιση των παραγωγών είναι τα Δένδρα Ανίχνευσης Σαν παράδειγμα ας δούμε την παραγωγή E ⇒$^{+}$ **-(id+ id)** (8) με βήματα. Κατ' αρχήν θεωρούμε το E σαν ρίζα του αρχικού δένδρου και εφαρμόζοντας με σειρά τους κανόνες E→**-**E, E→**(**E**)**, E→E+E, E→**id,** και E→**id** παίρνουμε την παρακάτω σειρά από παραγωγές οι οποίες αντιστοιχούν στην τμηματική κατασκευή του δένδρου ανίχνευσης όπως δείχνεται στο Σχήμα 3.1.

E ⇒ -E ⇒ **-(**E**)** ⇒ **-(**E+E**)** ⇒ **-(id**+E**)** ⇒ **-(id+id)**



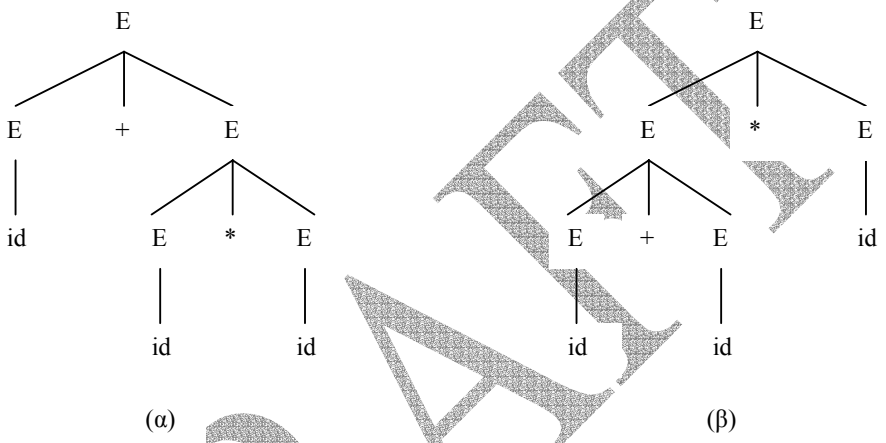
Σχήμα 3.1 Αριστερή παραγωγή του δένδρου ανίχνευσης της πρότασης –**(id+id)**

DRAFT

Δραστηριότητα 1 / Κεφ. 3 Η πρόταση **id+id*id** της γραμματικής (7), έχει δύο διαφορετικές αριστερές παραγωγές. Μπορείς να εξηγήσεις γιατί**;**

E	⇒	E + E	E	⇒	E * E
	⇒	id + E		⇒	E + E * E
	⇒	id + E * E		⇒	id + E * E
	⇒	id + id * E		⇒	id + id * E
	⇒	id + id * id		⇒	id + id * id

με δύο αντίστοιχα δένδρα ανίχνευσης :



Σχήμα 3.2 Τα δένδρα ανίχνευσης της πρότασης **id+id*id**

Πράγματι αν ξεκινήσετε με τον κανόνα E→E+E θα καταλήξετε στο δένδρο (α) που αντιστοιχεί στην άνω αριστερά παραγωγή του Σχήματος 3.2 ενώ αν ξεκινήσετε με τον κανόνα E→E*E θα καταλήξετε στο δένδρο (β) το οποίο αντιστοιχεί στην άνω δεξιά παραγωγή του ίδιου σχήματος. Ο λόγος για τον οποίο η πρόταση **id+id*id** έχει δύο διαφορετικά δένδρα ανίχνευσης είναι διότι η γραμματική (7) είναι **διφορούμενη**. Αν παρατηρήσετε λίγο την γραμματική αυτή θα δείτε ότι δεν εκφράζει τις προτεραιότητες ούτε τις προσεταιριστικότητες των τελεστών. Για παράδειγμα η γραμματική αυτή υποδηλώνει ότι οι τελεστές + και * έχουν την ίδια προτεραιότητα. Επιπλέον δεν έχουμε τρόπο να αποφασίσουμε ποιόν από τους εναλλακτικούς κανόνες πρέπει να χρησιμοποιήσουμε σε κάθε παραγωγή. Έτσι μπορούμε να φτιάξουμε είτε το ένα δένδρο είτε το άλλο.

ΕΝΟΤΗΤΑ 3.3 ΔΙΦΟΡΟΥΜΕΝΕΣ ΓΡΑΜΜΑΤΙΚΕΣ

Διφορούμενη/ασαφής λέγεται μια γραμματική η οποία παράγει περισσότερα από ένα δένδρα ανίχνευσης, για μια πρόταση. Για ορισμένους τύπους συντακτικών αναλυτών, θέλουμε η γραμματική να γίνει σαφής γιατί αλλιώς δεν μπορούμε να προσδιορίσουμε ένα μοναδικό δένδρο για μια πρόταση. Σε μερικές περιπτώσεις γίνεται χρήση Διφορούμενων γραμματικών μαζί με κανόνες αποσαφήνισης.

DRAFT

Έστω η γραμματική: E→E+E | E-E | E*E | E/E | E↑E | **(**E**)** | **-**E | **id** (9)

Η γραμματική αυτή όπως και η (7) είναι διφορούμενη. Μπορούμε όμως να απο-διφοροποιήσουμε και τις δύο, καθορίζοντας προσεταιριστικότητα και προτεραιότητα στους τελεστές. Ας υποθέσουμε ότι θέλουμε να τους δώσουμε τις γνωστές από τα Μαθηματικά προτεραιότητες:

- (unary minus) ↑ * / + -

και επίσης να ορίσουμε τον τελεστή ↑ να είναι δεξιά προσεταιριστικός

[δηλαδή, a ↑ b ↑c = a ↑(b ↑c)], και τους υπόλοιπους τελεστές να είναι αριστερά-προσεταιριστικοί.[δηλαδή, a-b-c = (a-b)-c].

Σύμφωνα με τους κανόνες αυτούς το δένδρο (β) του Σχήματος 3.2 δεν είναι σωστό για την πρόταση **id**+**id***id, μια και εδώ εμφανίζεται το + να έχει μεγαλύτερη προτεραιότητα από το *.

Μπορούμε να ξαναγράψουμε τις γραμματικές ενσωματώνοντας σε αυτές τους κανόνες προσεταιρισμού και προτεραιότητας.

Παράδειγμα 3 / Κεφ. 3

Ας πάρουμε την διφορούμενη γραμματική (9). Κατ' αρχήν εισάγουμε ένα Μη Τερματικό σύμβολο για κάθε επίπεδο προτεραιότητας.

Ονομάζουμε *Element* μια υποέκφραση η οποία είναι μη υποδιαιρέσιμη άρα είναι είτε ένας identifier ή μια έκφραση με παρενθέσεις. Έχουμε λοιπόν τον κανόνα:

element→(expression) | **id**

Κατόπιν εισάγουμε την κατηγορία των *primaries* τα οποία είναι elements με ένα ή περισσότερους από τους τελεστές με την μεγαλύτερη προτεραιότητα:

primary→- primary | element

Κατόπιν εισάγουμε τα *factors* σαν ακολουθίες από ένα ή περισσότερα primaries συνδεδεμένα με τον τελεστή ύψωσης σε δύναμη (↑)

factor→primary ↑ factor | primary

Σημειώνουμε ότι η εκλογή primary ↑ factor, αντί του factor ↑ primary κάνει τις εκφράσεις a↑b↑c να ομαδοποιούνται σαν a↑(b↑c).

Στη συνέχεια εισάγουμε το *term*, που είναι ακολουθία από ένα ή περισσότερα factors που συνδέονται με τους τελεστές πολλαπλασιασμού (*, /).

term→term * factor | term / factor | factor

Οι κανόνες αυτοί αναγκάζουν τα terms να ομαδοποιούνται από τα αριστερά [δηλαδή, a * b * c = (a * b) * c].

Τέλος τα *expressions* είναι ακολουθίες από ένα ή περισσότερα terms και συνδέονται με τους προσθετικούς τελεστές (+,-).

Expression → expression+term | expression-term | term

Και εδώ ο κανόνας επιβάλλει αριστερή προσεταιριστικότητα.

Έτσι η απο-διφοροποιημένη γραμματική είναι η:

DRAFT

expression→expression + term (10)
| expression - term
| term
term→term * factor
| term / factor
| factor
factor→Primary ↑ factor
| Primary
Primary→-Primary
| element
element→(expression)
| **id**

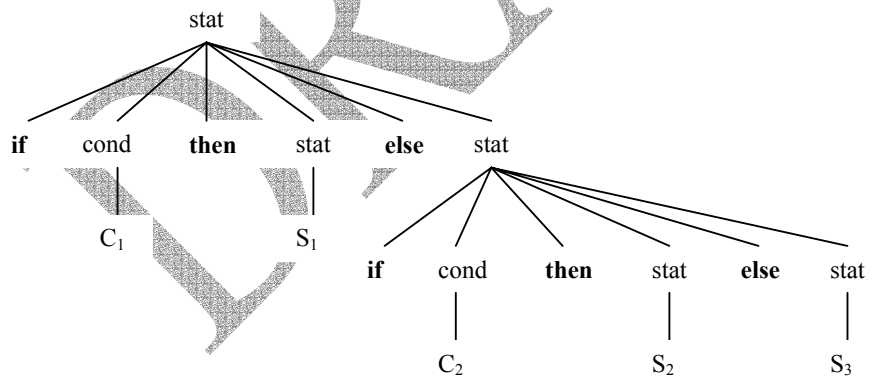
Δραστηριότητα 2 / Κεφ. 3 Το παρακάτω κομμάτι μιάς Context Free γραμματικής δημιουργεί εντολές υπό συνθήκη. Όμως η γραμματική αυτή είναι διφορούμενη. Μπορείτε να εξηγήσετε γιατί και στη συνέχεια να την αποδιφοροποιήσετε**;** Το πρώτο μέρος είναι εύκολο, το δεύτερο όμως είναι μάλλον δύσκολο γιαυτό να μην απογοητευθείτε αν δεν τα καταφέρετε.

stat→ **if** cond **then** stat
| **if** cond **then** stat **else** stat (11)
| other – stat

Για να δούμε αν η γραμματική (11) είναι διφορούμενη πρέπει να εξετάσουμε μήπως για κάποια πρότασή της προκύπτουν περισσότερα από ένα δένδρα ανίχνευσης. Έτσι βλέπουμε ότι η πρόταση

if C_1 **then** S_1 **else if** C_2 **then** S_2 **else** S_3

θα δώσει το παρακάτω δένδρο:



Σχήμα 3.3 Δένδρο ανίχνευσης της **if** C_1 **then** S_1 **else if** C_2 **then** S_2 **else** S_3

Η (11) όμως είναι τελικά διφορούμενη διότι η πρόταση

if C_1 **then if** C_2 **then** S_1 **else** S_2 (12)

έχει τα δύο δένδρα ανίχνευσης που δίνονται στο Σχήμα 3.4.

Σε όλες τις γλώσσες προγραμματισμού που περιέχουν τέτοιες εντολές προτιμάται το δένδρο (α). Ο γενικός κανόνας που χρησιμοποιείται είναι: κάθε **else** να ζευγαρώνεται με το πλησιέστερο προηγούμενο αζευγάρωτο **then**. Αυτό αποτελεί τον κανόνα αποδιφοροποίησης ο οποίος, αν θέλουμε, μπορεί να εισαχθεί στην γραμματική (11) και να πάρουμε την εξής:

DRAFT

stat → matched - stat | unmatched - stat

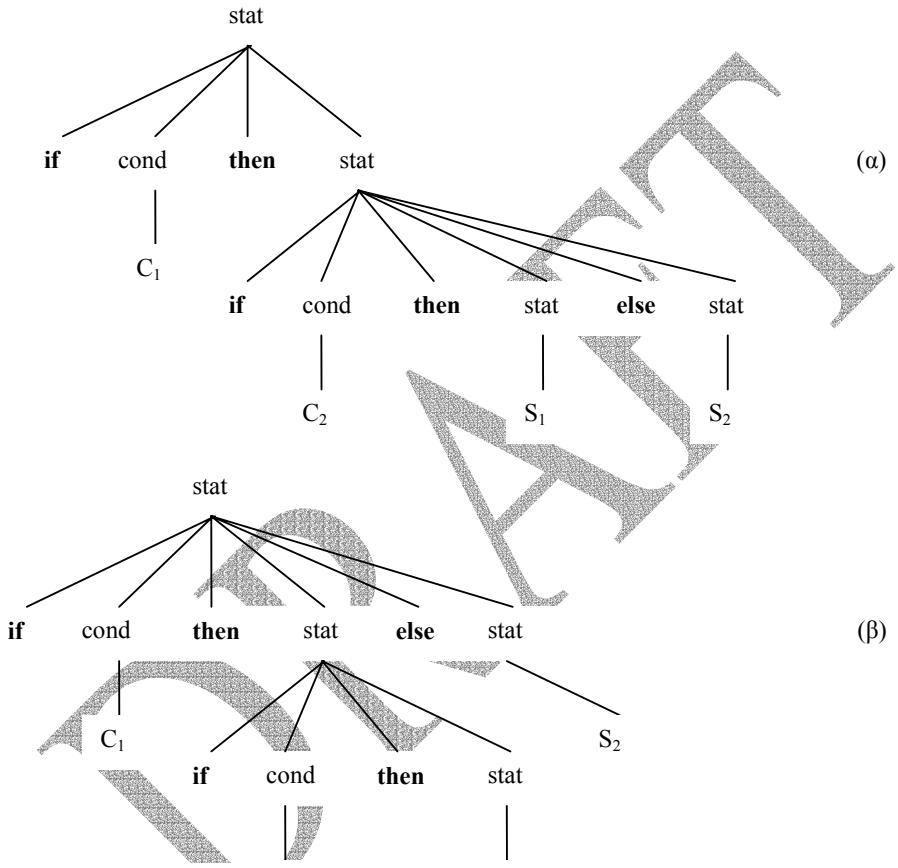
matched - stat → **if** cond **then** matched - stat **else** matched - stat

| other - stat

unmatched - stat → **if** cond **then** stat

| **if** cond **then** matched - stat **else** unmatched - stat

Η γραμματική αυτή δημιουργεί τις ίδιες συμβολοσειρές με την (11) αλλά επιτρέπει ένα μόνο δένδρο για την (12), δηλαδή αυτό που συσχετίζει κάθε **else** με το προηγούμενο αζευγάρωτο **then**. Στο Κεφάλαιο 4 θα δούμε και ένα άλλο τρόπο αποδιφοροποίησης της (11).



Σχήμα 3.4 Τα δένδρα ανίχνευσης της **if** C_1 **then if** C_2 **then** S_1 **else** S_2

Άσκηση Αυτοαξιολόγησης 3 / Κεφ. 3

Σας δίνεται η διφορούμενη γραμματική (όπου **con** υποδηλώνει constant)

E→ E+E| E*E| **(**E**)**| **id**| **con**

Προσπαθείστε να την αποδιφοροποιήσετε εφαρμόζοντας την μεθοδολογία που περιγράψαμε στην ενότητα 3.3.

ΕΝΟΤΗΤΑ 3.4 ΙΕΡΑΡΧΙΑ ΓΛΩΣΣΩΝ ΚΑΤΑ CHOMSKY

Ο Chomsky καθόρισε τέσσερις βασικές κλάσεις γλωσσών μέσω γραμματικών, οι οποίες είναι τετράδες (V,T,P,Z) όπου:

1.	V είναι η αλφάβητος (Τερματικά και Μη Τερματικά σύμβολα)

2.	T, (T⊂V) είναι το σύνολο των τερματικών συμβόλων
3.	P είναι ένα πεπερασμένο σύνολο από κανόνες (επαναγραφής) και
4.	Z είναι το αρχικό ή διακεκριμένο σύμβολο της γραμματικής.

Η γλώσσα μιας γραμματικής είναι το σύνολο των τερματικών ακολουθιών (terminal strings) τα οποία μπορούν να δημιουργηθούν από το αρχικό σύμβολο Z.

Η διαφορά μεταξύ των τεσσάρων τύπων γραμματικών ευρίσκεται στην μορφή των κανόνων επαναγραφής που επιτρέπονται στο P.

Μία γραμματική G είναι (Chomsky) **τύπου 0** ή **phrase structure** γραμματική αν οι κανόνες του P έχουν την μορφή

(0) $u \rightarrow v$, με $u \in V^+$ και $v \in V^*$

Δηλαδή, το αριστερό u μπορεί να είναι μια ακολουθία συμβόλων και το δεξιό μέρος να είναι άδειο.

Μια γραμματική G είναι **τύπου 1** ή **context sensitive** (εξαρτώμενη από το περιβάλλον κείμενο) όταν περιορίσουμε τους κανόνες επαναγραφής στην μορφή:

(1) $xUy \rightarrow xuy$, με $U \in V\text{-}T$, $x,y \in V^*$, και $u \in V^+$

Ο όρος context sensitive αναφέρεται στο γεγονός ότι επιτρέπεται να ξαναγράψουμε το U σαν u μόνο μέσα στο περιβάλλον x...y.

Μια γραμματική G είναι **τύπου 2** ή **Context-Free (**ανεξάρτητη περιβάλλοντος) αν όλοι οι κανόνες έχουν την μορφή:

(2) $U \rightarrow u$, με $U \in V\text{-}T$ και $u \in V^*$

Η κλάση αυτή καλείται ανεξάρτητη περιβάλλοντος διότι το U μπορεί να ξαναγραφεί σαν u ανεξάρτητα από το υπόλοιπο περιβάλλον κείμενο στο οποίο ευρίσκεται το U. Όπως φαίνεται από τον ορισμό σε μια CF γραμματική ένας κανόνας μπορεί να έχει την μορφή $U \rightarrow \varepsilon$ (ε=κενό). Δοθείσης μιας CF γραμματικής G, μπορεί να κατασκευασθεί μια **ε-free** γραμματική G1 (η οποία δεν περιέχει κανόνα της μορφής $U \rightarrow \varepsilon$) , τέτοια ώστε L(G1) = L(G) - {ε}. Επιπλέον αν η G είναι μη διφορούμενη τέτοια είναι και η G1.

Περιορίζοντας τους κανόνες ακόμη περισσότερο, στην μορφή :

(3) $U \rightarrow N$ ή $U \rightarrow WN$, με $N \in T$, και U και $W \in V\text{-}T$

έχουμε μια γραμματική **τύπου 3** ή Κανονική γραμματική (Regular grammar).

Παράδειγμα 4 / Κεφ. 3 Παραδείγματα κανόνων από Κανονικές Γραμματικές

Παρατηρήστε ότι τα Μη Τερματικά σύμβολα μπορούν να περιβάλλονται από "<" και ">" για λόγους καλύτερης αναγνωσιμότητας των κανόνων.

<s-delim> →+ | - | (|)

<d-delim> →<slash> / | <slash>* | <ast>* | <colon> =
<slash> → /
<ast> → *
<colon> → :

DRAFT

Άσκηση Αυτοαξιολόγησης 4 / Κεφ. 3

Οι Κανονικές Εκφράσεις για ονόματα μεταβλητών και για ακέραιες σταθερές είναι l(l |d)* και d(d)* αντίστοιχα (l υποδηλώνει letter και d υποδηλώνει digit). Προσπαθήστε να τις γράψετε με την βοήθεια των Κανονικών γραμματικών.

ΣΥΝΟΨΗ ΚΕΦΑΛΑΙΟΥ

Στο κεφάλαιο αυτό είδαμε μια άτυπη περιγραφή των γραμματικών τύπου Context Free που είναι οι γραμματικές οι οποίες κατά κύριο λόγο χρησιμοποιούνται για τον συντακτικό ορισμό των γλωσσών προγραμματισμού. Αναφέραμε βέβαια και τους τυπικούς ορισμούς του Chomsky για λόγους πληρότητας αλλά και για να αναφερθούμε και στις Κανονικές γραμματικές με τις οποίες είδαμε ότι μπορούμε να εκφράσουμε τις λεκτικές δομές των γλώσσών προγραμματισμού. Επειδή οι Κανονικές γραμματικές είναι στην ουσία υποσύνολο των Context Free γραμματικών θα μπορούσαμε να πούμε ότι με τις Context Free γραμματικές μπορούμε να εκφράσουμε ολόκληρη την συντακτική και λεκτική δομή των γλωσσών.

Δώσαμε έμφαση στις παραγωγές από το Αρχικό σύμβολο της γραμματικής και ορίσαμε την πρόταση και την προτασιακή μορφή τις οποίες θα χρησιμοποιήσουμε στο επόμενο κεφάλαιο. Έμφαση, και μάλιστα ιδιαίτερη, δώσαμε στις διφορούμενες γραμματικές και στο πώς τις αποδιφοροποιούμε.

Απάντηση άσκησης 1 /κεφ. 3

Οι Κανονικές Εκφράσεις χρησιμοποιούνται στις γλώσσες προγραμματισμού για να περιγράψουν τις λεκτικές τους δομές, δηλαδή τα tokens. Αντίθετα, για να περιγράψουμε τις συντακτικές δομές της γλώσσας (εντολές) χρησιμοποιούμε τις Context Free γραμματικές. Μπορούμε φυσικά να χρησιμοποιήσουμε Context Free γραμματικές για να περιγράψουμε και τις λεκτικές δομές της γλώσσας, όμως επειδή οι Context Free γραμματικές είναι πολύ πιο ισχυρό εργαλείο από τις Κανονικές Εκφράσεις, είναι και πολύ πιο δύσκολο να φτιάξουμε 'αναγνωριστές' για Context Free γραμματικές απότι είναι για Κανονικές Εκφράσεις.

Απάντηση άσκησης 2 /κεφ. 3

Μια συμβολοσειρά από τερματικά σύμβολα μιας γραμματικής είναι πρόταση της γλώσσας που δημιουργείται από την γραμματική αυτή εάν προκύπτει μέσα από μια σειρά παραγωγών οι οποίες ξεκινούν από το Αρχικό σύμβολο της γραμματικής. Έτσι, δεδομένου ότι το Αρχικό σύμβολο της γραμματικής (7) είναι το E, με την παρακάτω σειρά παραγωγών καταλήγουμε στην πρόταση -(**id**+ **id**).

$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(E + \mathbf{id}) \Rightarrow \mathbf{-(id+ id)}$

Η παραγωγή αυτή είναι *δεξιότερη* παραγωγή. Στην ίδια πρόταση μπορείτε να καταλήξετε και με την εξής *αριστερότερη* παραγωγή

$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(\mathbf{id}+E) \Rightarrow \mathbf{-(id+ id)}$

DRAFT

Οι συμβολοσειρές -E, -(E), -(E+E), -(E+**id**) και –(**id**+E) είναι *προτασιακές* μορφές.

Απάντηση άσκησης 3 /κεφ. 3

Ο λόγος που η γραμματική αυτή είναι διφορούμενη είναι διότι έχοντας ένα μόνο κανόνα με πέντε εναλλακτικά δεξιά μέρη ορίζει ένα μόνο επίπεδο προτεραιότητας και έτσι οι τελεστές +, * αλλά και οι μη υποδιαιρέσιμες εκφράσεις (E), **id** και **con** έχουν όλα την ίδια προτεραιότητα. Θυμηθήτε ότι για να ορίσουμε τις προτεραιότητες εισάγουμε καινούρια Μη Τερματικά σύμβολα (τα οποία τα είχαμε αναφέρει ως *συντακτικές κατηγορίες*). Επίσης η γραμματική αυτή δεν εκφράζει ούτε την προσεταιριστικότητα των τελεστών. Και πάλι θυμηθήτε ότι την προσεταιριστικότητα την εκφράζουμε με *τον τρόπο που γράφουμε τους κανόνες*. Για παράδειγμα ο κανόνας

$$X \rightarrow X@Y$$

αποδίδει αριστερή προσεταιριστικότητα για τον τελεστή @, ενώ αντίθετα ο κανόνας

$$X \rightarrow Y@X$$

αποδίδει δεξιά προσεταιριστικότητα για τον τελεστή @.

Μετά από αυτά μπορούμε να προχωρήσουμε στην αποδιφοροποίηση.

Εισάγουμε πρώτα ένα νέο Μη Τερματικό σύμβολο Elem για τις μη διαιρέσιμες εφράσεις με την υψηλότερη προτεραιότητα,

Elem → **(**E**)**| **id**| **con**

Κατόπιν εισάγουμε το σύμβολο Fact για τον τελεστή πολλαπλασιασμού ο οποίος είναι αριστερά προσεταιριστικός και έχει την αμέσως χαμηλότερη προτεραιότητα.

Fact → Fact * Elem| Elem

Τέλος, έχουμε το Αρχικό σύμβολο της γραμματικής E για τον τελεστή πρόσθεσης που έχει την χαμηλότερη προτεραιότητα και είναι επίσης αριστερά προσεταιριστικός.

E → E+ Fact| Fact

Έτσι η αρχική διφορούμενη γραμματική

E→ E+E| E*E| **(**E**)**| **id**| **con**

Μετασχηματίστηκε στην αποδιφοροποιημένη μορφή

E → E+ Fact| Fact

Fact → Fact * Elem| Elem

Elem → **(**E**)**| **id**| **con**

Απάντηση άσκησης 4 /κεφ. 3

Οι Κανονικές γραμματικές σύμφωνα με τον ορισμό τους έχουν την μορφή

$U \rightarrow N$ ή $U \rightarrow WN$, με $N \in T$ και U και $W \in V-T$

Έτσι, αν συμβολίσουμε το l με letter και το d με digit, για πιο ευανάγνωστους κανόνες, και επιπλέον χρησιμοποιήσουμε τα Μη Τερματικά σύμβολα <id> και <int> για να εκφράσουμε τα

DRAFT

ονόματα μεταβλητών και τις ακέραιες σταθερές αντίστοιχα τότε οι παρακάτω κανόνες είναι οι ζητούμενοι μια και παράγουν τις ίδιες συμβολοσειρές τερματικών συμβόλων.

<id> → letter
|<id> letter
|<id>digit
<int> → digit
|<int> digit

Παρατηρήστε ότι η Κανονική Έκφραση d(d)* θα μπορούσε να είχε δοθεί και με τον ισοδύναμο ορισμό της d^+ χωρίς φυσικά να αλλάξει τίποτα.

DRAFT