Εθνικό Μετσόβιο Πολυτεχνείο
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Διπλωματική εργασία :

# Συστήματα μεταγραφής και ο συνδυαστής $S$

Παναγιώτης Χείλαρης

Τριμελής επιτροπή Καθηγητών :

Επιβλέπων : Στάθης Ζάχος

Μέλη : Τίμος Σελλής
Γιώργος Κολέτσος

Αθήνα, 2001

# Συστήματα μεταγραφής και ο συνδυαστής $S$

# Συστήματα μεταγραφής και ο συνδυαστής $S$

Παναγιώτης Χείλαρης

έκδοση 0.929 (κωδικοποίηση utf-8)

*στην οικογένειά μου*

**Περίληψη.** Τα συστήματα μεταγραφής όρων είναι άλλος ένας φορμαλισμός, ο οποίος μπορεί να χρησιμοποιηθεί για την θεμελίωση της θεωρίας υπολογισμού. Τα δύο βασικά θέματα στα συστήματα μεταγραφής είναι η συμβολή (αν αποκλίνουσες ακολουθίες υπολογισμού συμβάλλουν αργότερα σε κάποιο κοινό αποτέλεσμα) και ο τερματισμός (αν δηλαδή τελειώνει ένας υπολογισμός καταλήγοντας σε κάποιο τελικό αποτέλεσμα, το οποίο ονομάζουμε κανονική μορφή), ιδιότητες που θεωρούνται εν γένει επιθυμητές (αν υπάρχουν αμφότερες, το σύστημα ονομάζεται πλήρες). Η μέθοδος της συμπλήρωσης Knuth-Bendix βοηθά να μετατρέψουμε ένα σύστημα που ανήκει σε μία ειδική κλάση σε πλήρες. Μία σημαντική κατηγορία συστημάτων είναι τα ορθογώνια συστήματα, τα οποία έχουν την ιδιότητα της συμβολής.

Οι όροι μπορούν να ιδωθούν ως δένδρα επί ενός αλφαβήτου συναρτησιακών συμβόλων. Έχει νόημα να μελετήσουμε σύνολα, ή αλλιώς γλώσσες, δένδρων, καθώς και αυτόματα που αναγνωρίζουν τέτοιες γλώσσες και γραμματικές δένδρων. Τα αυτόματα και οι γλώσσες δένδρων έχουν άμεσες εφαρμογές στα συστήματα μεταγραφής όρων.

Ένα κλασικό σύστημα μεταγραφής είναι η συνδυαστική λογική. Με τους συνδυαστές $S$ και $K$, η συνδυαστική λογική μπορεί να θεωρηθεί ως ένα Turing-πλήρες υπολογιστικό σύστημα. Ένα απλούστερο σύστημα είναι αυτό στο οποίο θεωρούμε μόνον τον συνδυαστή $S$, ο λεγόμενος $S$-λογισμός. Για όρους που αποτελούνται μόνον από $S$, δίνουμε μία διάταξη και έναν απλό αλγόριθμο που δίνει διαδοχικά όλους τους όρους στην διάταξη. Στον $S$-λογισμό απεδείχθη, πρόσφατα, ότι είναι αποκρίσιμο αν ένας όρος έχει κανονική μορφή ή όχι.

Δίνουμε πολλά προγράμματα για την επεξεργασία όρων στα πλαίσια του $S$-λογισμού, τα οποία μας βοήθησαν να συλλέξουμε αρκετά στατιστικά δεδομένα. Επίσης δίνουμε έναν κατάλογο των μη κανονικοποιήσιμων τέτοιων όρων για μήκος όρου μέχρι και 10.

**Abstract.** Term rewriting systems are another formalism, that can be used in the foundation of the theory of computation. The two basic issues in term rewriting systems are confluence (whether two diverging computation sequences converge later to some common result) and termination (whether a computation terminates, reaching some final result, which is called normal form), properties that are considered desirable in general (if both properties exist, the system is called complete). The Knuth-Bendix completion method helps in converting some system in a special class to a complete one. An importatnt category of systems are orthogonal systems, which have the confluence property.

Terms can be seen as trees over an alphabet of functional symbols. It is sensible to study sets, or else languages, of trees, and also automata recognizing such languages and tree grammars. Tree automata and tree languages have immediate applications in tree rewriting systems.

A classic rewriting system is combinatory logic. With combinators $S$ and $K$, combinatory logic can be considered as a Turing-complete computational system. A simpler system, called the $S$-calculus, is the one where only the combinator $S$ is considered. For terms consisting only of $S$'s, we give an ordering and a simple algorithm that gives consecutively all terms in the ordering. It was recently proved that in $S$-calculus it is decidable whether a term has a normal form or not.

We give many programs for manipulating terms in the context of $S$-calculus ; these programs helped us collect statistical data. We also give a list of all non normalizing such terms for length up to 10.

# Περιεχόμενα

# Κατάλογος σχημάτων

# Κατάλογος πινάκων

# Εισαγωγή

Το παρόν σύγγραμμα εκπονήθηκε στα πλαίσια της διπλωματικής εργασίας μου στο Εθνικό Μετσόβιο Πολυτεχνείο.

Τα *συστήματα μεταγραφής* (rewrite systems) αποτελούν έναν ακόμη φορμαλισμό που μπορεί να περιγράψει τις υπολογιστικές διαδικασίες. Είναι ένα πλήρες μοντέλο υπολογισμού με την έννοια ότι αποδεικνύεται ισοδύναμο με τις μηχανές Turing. Σε ένα *αφηρημένο σύστημα μεταγραφής*, υπάρχει ένα σύνολο (που μπορούμε να θεωρήσουμε ότι περιέχει τα διάφορα αντικείμενα υπολογισμού : εισόδους, ενδιάμεσα και τελικά αποτελέσματα) και μία *σχέση μεταγραφής* που συνδέει (σε ένα βήμα υπολογισμού) τα αντικείμενα του παραπάνω συνόλου. Το ενδιαφέρον είναι ότι είναι δυνατόν με τα συστήματα μεταγραφής να περιγραφούν εν γένει *μη αιτιοκρατικοί* υπολογισμοί, αφού ενδέχεται από ένα αντικείμενο να είναι δυνατόν να προκύψουν διάφορα άλλα σε ένα βήμα μεταγραφής.

Τα *συστήματα μεταγραφής όρων* είναι ίσως η σημαντικότερη κατηγορία συστημάτων μεταγραφής. Πολλές άλλες κατηγορίες συστημάτων μεταγραφής, όπως τα *συστήματα μεταγραφής συμβολοσειρών*, μπορούν να θεωρηθούν ως ειδικές περιπτώσεις των συστημάτων μεταγραφής όρων. Τα συστήματα μεταγραφής όρων έχουν επιπλέον δομή, σε σχέση με τα αφηρημένα συστήματα μεταγραφής, αφ' ενός επειδή ορίζονται επί ενός συνόλου που έχει επιπλέον δομή (το σύνολο των όρων επί ενός αλφαβήτου συναρτησιακών συμβόλων) και αφ' ετέρου επειδή απαιτείται επιπλέον δομή στην σχέση που ορίζει την μεταγραφή.

Μία σημαντική εφαρμογή των συστημάτων μεταγραφής είναι στα συστήματα εξισώσεων, όπου είναι δυνατόν σε ορισμένες περιπτώσεις να προσανατολιστούν με τέτοιον τρόπο οι εξισώσεις (και να γίνουν επομένως κανόνες μεταγραφής) και ενδεχομένως να προστεθούν νέοι κανόνες, ούτως ώστε να έχουμε αλγόριθμο που αποκρίνεται για την αντίστοιχη αρχική θεωρία.

Μερικά θέματα όπως οι γράφοι, τα δένδρα παρουσιάζονται περιληπτικά μέσα στο κείμενο όποτε αυτό είναι απαραίτητο. Εκτενώς παρουσιάζονται αυτόματα που ενεργούν επί δένδρων (ή αλλιώς όρων) και οι συγγενής έννοια των γραμματικών δένδρων καθώς έχουν πολλές εφαρμογές στα συστήματα

μεταγραφής όρων.

Αν δούμε τα συστήματα μεταγραφής ως συστήματα υπολογισμού, έχουν ενδιαφέρον τα παρακάτω δύο ερωτήματα :

- Οι υπολογισμοί που αρχίζουν από ένα αντικείμενο καταλήγουν πάντα σε κάποιο τελικό αποτέλεσμα ;
- Αποκλίνοντες κλάδοι (εξαιτίας της μη αιτιοκρατίας) από την ίδιο αρχικό αντικείμενο υπολογισμού μπορούν να συγκλίνουν πάλι σε ένα κοινό αντικείμενο ;

Το πρώτο είναι το πρόβλημα του *τερματισμού*. Το δεύτερο είναι το πρόβλημα της *συμβολής*. Τα παραπάνω προβλήματα μπορούν να συνδεθούν με αντίστοιχα προβλήματα στα πλαίσια των μηχανών Turing οπότε είναι εν γένει μη αποκρίσιμα.

Η παρουσίαση των συστημάτων μεταγραφής έχει σκοπό την εισαγωγή στο σύστημα του *S-λογισμού*. Το σύστημα του $S$-λογισμού είναι ένα υποσύνολο της συνδυαστικής λογικής. Η συνδυαστική λογική είναι ένα σχετικά απλό σύστημα μεταγραφής, το οποίο όμως αποδεικνύεται πλήρες κατά Turing. Αυτό έχει ως συνέπεια πολλά προβλήματα, όπως αυτό του τερματισμού, να είναι μη αποκρίσιμα. Όμως, το υποσύστημα του $S$-λογισμού, όπως αποδείχθηκε πρόσφατα [104, 105, 110], έχει πολλά από αυτά τα προβλήματα αποκρίσιμα.

Το πρώτο μέρος του παρόντος συγγράμματος αφιερώνεται στα *συστήματα μεταγραφής*. Στο πρώτο κεφάλαιο, δίνονται κάποιες βασικές έννοιες για τις *σχέσεις*. Στο δεύτερο κεφάλαιο, παρουσιάζονται τα *αφηρημένα συστήματα μεταγραφής*. Στο τρίτο κεφάλαιο, μελετώνται οι έννοιες της *συμβολής* και του *τερματισμού* στα πλαίσια των αφηρημένων συστημάτων μεταγραφής. Στο τέταρτο κεφάλαιο, εισάγεται η έννοια του *όρου*. Στο πέμπτο κεφάλαιο, ορίζονται τα *συστήματα μεταγραφής όρων*, δίνονται διάφορα παραδείγματα τέτοιων συστημάτων και διατυπώνονται διάφορες ιδιότητες, ενώ αναφέρεται επίσης η αποκρισιμότητα διαφόρων σχετικών προβλημάτων και διάφορα κριτήρια τερματισμού για τέτοια συστήματα. Στο έκτο κεφάλαιο, ορίζεται η έννοια του *κρισίμου ζεύγους*, ως εμποδίου για την συμβολή, και η μέθοδος της *συμπλήρωσης* Knuth-Bendix. Στο έβδομο κεφάλαιο, παρουσιάζεται η σημαντική υποκατηγορία των *ορθογωνίων* συστημάτων μεταγραφής όρων και μία απόδειξη για την συμβολή σε αυτού του είδους τα συστήματα.

Το δεύτερο μέρος αναφέρεται σε *γλώσσες και αυτόματα δένδρων*. Στο όγδοο κεφάλαιο, μελετώνται τα *αυτόματα δένδρων* ως αναγνωριστές για γλώσσες δένδρων. Στο ένατο κεφάλαιο, δίνουμε έναν εναλλακτικό τρόπο περιγραφής γλωσσών δένδρων : *τις γραμματικές δένδρων*. Στο δέκατο κεφάλαιο, αναφερόμαστε σε *επεκτάσεις* των αυτομάτων δένδρων με περισσότερες δυνατότητες και σε *εφαρμογές* των αυτομάτων δένδρων, κυρίως στα συστήματα μεταγραφής όρων.

Στο τρίτο μέρος παρουσιάζεται το σύστημα του $S$-*λογισμού*. Στο ενδέκατο κεφάλαιο, παρουσιάζονται τα αντικείμενα μεταγραφής του $S$-λογισμού, που είναι οι $S$-*όροι*, ορίζεται μία διάταξη στους $S$-όρους και δίνεται ένας *αλγόριθμος εύρεσης του επομένου* $S$-*όρου*. Στο δωδέκατο κεφάλαιο, γίνεται λόγος για τον $S$-λογισμό, ως υποσύστημα της *συνδυαστικής λογικής*, και τις *ιδιότητές* του. Στο δέκατο τρίτο κεφάλαιο, δίνεται το περίγραμμα της απόδειξης της *αποκρισιμότητας* του ερωτήματος αν κάποιος $S$-όρος είναι κανονικοποιήσιμος. Στο δέκατο τέταρτο κεφάλαιο, δίνονται περιγραφές *απείρων αλυσίδων* αναγωγών στον $S$-λογισμό για μερικές δύσκολες περιπτώσεις, σύμφωνα με την μέθοδο Ζάχου.

Μετά, ακολουθούν τα παραρτήματα. Στο πρώτο παράρτημα, δίνονται τα *προγράμματα* που έχουν γραφτεί σε γλώσσα C και αφορούν $S$-όρους. Στο δεύτερο παράρτημα, έχουμε προγράμματα που αφορούν κυρίως *τεχνολόγηση και μετατροπή* $S$-όρων, γραμμένα στα πλαίσια του μετα-μεταγλωττιστή eli. Στο τρίτο παράρτημα, από μία κανονική γραμματική δένδρων του Waldmann [104, 105] για τους κανονικοποιήσιμους $S$-όρους, κατασκευάζεται μία γραμματική χωρίς συμφραζόμενα και ο αντίστοιχος τεχνολογητής με την βοήθεια του εργαλείου btyacc. Στο τέταρτο παράρτημα, δίνεται κατάλογος των μη κανονικοποιήσιμων όρων μήκους μέχρι και 10.

Ακολουθούν βιβλιογραφικές αναφορές. Μετά δίνεται ένας ταξινομημένος κατάλογος των ελληνικών όρων με την μετάφρασή τους στα αγγλικά. Τέλος, υπάρχει εκτεταμένο ευρετήριο όρων και συγγραφέων. Οι όροι και συγγραφείς με ελληνικούς χαρακτήρες προτάσσονται αυτών με λατινικούς χαρακτήρες. Συνήθως, όποτε ένας όρος με λατινικούς χαρακτήρες εμφανίζεται στο δεύτερο μέρος του ευρετηρίου, δίνεται απλώς αναφορά στον ελληνικό όρο στο πρώτο τμήμα του ευρετηρίου, οπότε το ευρετήριο λειτουργεί εν μέρει και ως μετάφραση στα ελληνικά, των ξένων όρων.

**Ευχαριστίες.** Σε αυτό το σημείο, αισθάνομαι την ανάγκη να ευχαριστήσω όλους όσους συνετέλεσαν στην ασχολία μου με το θέμα της διπλωματικής εργασίας και βοήθησαν στην ολοκλήρωση αυτής με επιτυχία. Πρώτον από όλους, ευχαριστώ τον επιβλέποντα καθηγητή Στάθη Ζάχο, ο οποίος με έφερε σε επαφή με το σύστημα του $S$-λογισμού και ιδιαιτέρως με το πρόβλημα της αποκρισιμότητας της κανονικοποίησης. Η μελέτη της διδακτορικής διατριβής του [108] απετέλεσε το έναυσμα για την συγγραφή των προγραμμάτων του παραρτήματος Α. Επίσης, το δεύτερο μέρος του παρόντος συγγράμματος, που αφορά τις γλώσσες και τα αυτόματα δένδρων, δεν θα είχε συμπεριληφθεί χωρίς την δική του παρότρυνση να μελετήσω αυτό το πεδίο. Θα μου μείνει αξέχαστη η συνεργασία μας, κάποια ζεστά ανοιξιάτικα απογεύματα, για την εύρεση περιγραφής απείρων αλυσίδων αναγωγών στον $S$-λογισμό με την μέ-

θοδό του : Αποτελέσματα αυτής της προσπάθειας υπάρχουν στο κεφάλαιο 14. Εξάλλου, η συμμετοχή στην ομάδα της οποίας ηγείται στο Εργαστήριο Λογικής και Υπολογισμών (Co.Re.Lab) υπήρξε σημαντική επιστημονική εμπειρία για μένα και για αυτό ευχαριστώ όλα τα μέλη της ομάδας κατά την περίοδο εκπόνησης της διπλωματικής μου εργασίας για την θερμή υποδοχή και για την συνεργασία. Ιδιαιτέρως πρέπει να ευχαριστήσω τον Σταύρο Ρουτζούνη με τον οποίο συνεργαστήκαμε στην συγγραφή του [111] και είναι κύριος υπεύθυνος για τις λεπτομέρειες της γραμματικής της ενότητας 13.6, στην σελίδα 147. Επίσης, αν και δεν τον γνώρισα προσωπικά, ευχαριστώ τον John Ramirez, σε κώδικα TEX του οποίου βασίστηκα κατά την συγγραφή του [111]. Κώδικας TEX του John Ramirez έχει παρεισφρήσει αυτούσιος και στο παρόν σύγγραμμα : Πρόκειται για το διάγραμμα 13.1 στην σελίδα 145. Κώδικα TEX, ο οποίος χρησιμοποιήθηκε στο [111], συνεισέφερε και ο Αντώνης Καβαρνός, τον οποίον επίσης ευχαριστώ. Τέλος, θέλω να ευχαριστήσω την οικογένεια μου, για την συμπαράσταση και κατανόηση που συνεχώς επιδεικνύουν στο πρόσωπό μου. Στα μέλη της οικογένειας μου αφιερώνεται και το παρόν σύγγραμμα.

Παναγιώτης Χείλαρης
Αθήνα
Ιούλιος 2001

# Μέρος πρώτο

# Συστήματα μεταγραφής

# Κεφάλαιο 1

# Σχέσεις και ιδιότητές τους

Σύμφωνα με την διαδικασία της μεταγραφής, κάποιο αντικείμενο μεταγράφεται σε κάποιο άλλο. Έχουμε δηλαδή δύο αντικείμενα (ή στοιχεία) και μία γενικά μη συμμετρική *συσχέτιση* του πρώτου με το δεύτερο.

Αυτή η συσχέτιση μπορεί να γίνει κατά τυπικό τρόπο με την βοήθεια της έννοιας της *σχέσης*. Σε αυτό το κεφάλαιο λοιπόν εισάγουμε την έννοια της σχέσης και δίνουμε κάποιες *ιδιότητες* αυτής. Μάλιστα, η έννοια της σχέσης αποδεικνύεται πολύτιμη, αφού πολλές άλλες έννοιες, τις οποίες θα μελετήσουμε σε αυτό το κεφάλαιο, όπως οι *συναρτήσεις*, οι *σχέσεις ισοδυναμίας* και οι *διατάξεις* αποτελούν ουσιαστικά ειδικές περιπτώσεις σχέσεων.

Ο αναγνώστης που έχει το απαιτούμενο υπόβαθρο μπορεί να παραλείψει αυτό το κεφάλαιο.

## 1.1 Σχέσεις και αναγωγές

Προκειμένου να μιλήσουμε για σχέσεις, θα βασιστούμε στην έννοια του *συνόλου*. Για μία αξιωματική θεμελίωση της συνολοθεωρίας, βλέπε για παράδειγμα το [71]. Τα σύνολα θα τα συμβολίζουμε με κεφαλαία γράμματα, για παράδειγμα $A$, $B$, ενώ τα στοιχεία τους με πεζά, για παράδειγμα $a$, $b$, $x$, $y$. Επίσης, θα θεωρήσουμε τις γνωστές πράξεις στα σύνολα : ένωση ($\cup$), τομή ($\cap$), διαφορά ($-$), δυναμοσύνολο ($\mathcal{P}$), καθώς και τις έννοιες του «ανήκειν» ($\in$) και του υποσυνόλου ($\subseteq$).

Στον ορισμό της σχέσης σημαντικό ρόλο παίζει και η έννοια του *διατεταγμένου ζεύγους* το οποίο συμβολίζεται με $(x, y)$, για δύο οποιαδήποτε στοιχεία $x$, $y$. Αν τα $x$, $y$ είναι διαφορετικά, το ζεύγος $(x, y)$ είναι διαφορετικό από το $(y, x)$, δηλαδή παίζει ρόλο η *διάταξή* τους στο ζεύγος, εξ ου και ο χαρακτηρισμός «διατεταγμένο».

Για περισσότερες λεπτομέρειες, παραπέμπουμε τον ενδιαφερόμενο αναγνώ-

στη στο [71].

**1.1.1 Ορισμός.** Το *καρτεσιανό γινόμενο* ενός συνόλου $A$ επί ένα σύνολο $B$ είναι το σύνολο $A \times B \overset{\text{ορ.}}{=} \{(x, y) \mid x \in A \wedge y \in B\}$.

**1.1.2 Ορισμός.** Μία (διμελής) *σχέση* $R$ στο σύνολο $A$ είναι ένα υποσύνολο του συνόλου $A \times A$.

Εναλλακτικά, αν $(x, y) \in R$, γράφουμε $x\ R\ y$ ή και $x \to_R y$ ή απλούστερα $x \to y$ αν υπονοείται η σχέση $R$. Αφού οι σχέσεις είναι στην ουσία σύνολα, επιτρέπονται σε αυτές οι πράξεις των συνόλων (ένωση, τομή κ.τ.λ.).

**1.1.3 Ορισμός.** Στην περίπτωση των συστημάτων *αναγωγής*, για δύο στοιχεία του συνόλου $A$, έστω τα $x$, $y$ για τα οποία ισχύει $x \to y$, λέμε ότι το $x$ *ανάγεται σε ένα βήμα* (ή *μεταγράφεται*) στο $y$.

Πρέπει πάντως να παρατηρήσουμε ότι είναι δυνατόν να γενικευτεί η έννοια της διμελούς σχέσης, οπότε αυτή να είναι ένα υποσύνολο του καρτεσιανού γινομένου $A \times B$ δύο οποιωνδήποτε συνόλων $A$ και $B$, δηλαδή να μην ισχύει απαραιτήτως $A = B$, όπως στον ορισμό 1.1.2. Στην περίπτωση, όμως, των συστημάτων μεταγραφής χρησιμοποιούμε πάντοτε σχέσεις σε ένα μόνον σύνολο $A$, αφού μας ενδιαφέρουν αναγωγές που γίνονται στα πλαίσια του συνόλου $A$. Η έννοια της $R \subseteq A \times B$ θα μας φανεί όμως χρήσιμη στον ορισμό της συνάρτησης.

**1.1.4 Ορισμός.** Μία *συνάρτηση* $f$ από το σύνολο $A$ στο σύνολο $B$ (συμβολίζεται $f\colon A \to B$) είναι μία σχέση $R \subseteq A \times B$, τέτοια ώστε για κάθε $x \in A$ να υπάρχει ακριβώς ένα $y \in B$ τέτοιο ώστε $x\ R\ y$. Το μοναδικό αυτό $y$ για κάθε $x$ συμβολίζουμε συνήθως ως $f(x)$ ή $fx$ ή $f_x$.

**1.1.5 Ορισμός** (Συμπλήρωμα)**.** Έστω σχέση $\to_R$. Το *συμπλήρωμα* της σχέσης ορίζεται ως εξής : $\not\to_R \overset{\text{ορ.}}{=} \{(x, y) \mid (x, y) \notin R\}$. Εναλλακτικά, γράφουμε $\overline{R}$ ή $R^{\mathrm{C}}$.

**1.1.6 Ορισμός** (Σύνθεση)**.** Έστω δύο σχέσεις $R_1$ και $R_2$ στο σύνολο $A$, η *σύνθεσή* τους ορίζεται ως $R_1 \circ R_2 \overset{\text{ορ.}}{=} \{(x, z) \mid \exists y\colon (x, y) \in R_1 \wedge (y, z) \in R_2\}$. Εναλλακτικά, η σύνθεση γράφεται $\to_{R_1} \circ \to_{R_2}$.

Η σύνθεση μπορεί να θεωρηθεί ως μία πράξη στο σύνολο των σχέσεων. Πιο συγκεκριμένα, αν οι σχέσεις είναι επί του συνόλου $A$, το σύνολο αυτό είναι το $\mathcal{P}(A \times A)$ (κατά τα συνήθη συμβολίζουμε το *δυναμοσύνολο* ενός συνόλου $X$ ως $\mathcal{P}(X) \overset{\text{ορ.}}{=} \{S \mid S \subseteq X\}$) και η σύνθεση είναι μία συνάρτηση

$$\text{`}\circ\text{'}\colon \quad \mathcal{P}(A \times A) \times \mathcal{P}(A \times A) \quad \to \quad \mathcal{P}(A \times A)$$

Στην παρακάτω πρόταση δίνονται κάποιες χρήσιμες ιδιότητες της σύνθεσης :

**1.1.7 Πρόταση.** *Έστω σχέσεις $R_1$, $R_2$, $R_3$, $R_4$.*

(α) *Η πράξη της σύνθεσης είναι* προσεταιριστική, *δηλαδή*

$$(R_1 \circ R_2) \circ R_3 = R_1 \circ (R_2 \circ R_3)$$

(β) *Η πράξη της σύνθεσης* δεν είναι αντιμεταθετική, *δηλαδή γενικά ισχύει*

$$R_1 \circ R_2 \neq R_2 \circ R_1$$

(γ) *Η πράξη της σύνθεσης είναι* επιμεριστική *ως προς την ένωση :*

$$(R_1 \cup R_2) \circ R_3 = (R_1 \circ R_3) \cup (R_2 \circ R_3)$$
$$R_1 \circ (R_2 \cup R_3) = (R_1 \circ R_2) \cup (R_1 \circ R_3)$$

(δ) *Αν $R_1 \subseteq R_3$ και $R_2 \subseteq R_4$, τότε $R_1 \circ R_2 \subseteq R_3 \circ R_4$.*

Έχει ενδιαφέρον να αναζητήσουμε το ουδέτερο στοιχείο της σύνθεσης.

**1.1.8 Ορισμός.** $\overset{0}{\to} \overset{\text{ορ.}}{=} 1_A \overset{\text{ορ.}}{=} \equiv \overset{\text{ορ.}}{=} \{(x,x) \mid x \in A\}$

**1.1.9 Πρόταση.** *Η σχέση $\overset{0}{\to}$ είναι το ουδέτερο στοιχείο της σύνθεσης.*

Η σχέση $\overset{0}{\to}$ ονομάζεται και *σχέση ισότητας* στο σύνολο $A$.

**1.1.10 Ορισμός.** $\overset{=}{\to} \overset{\text{ορ.}}{=} \to \cup \overset{0}{\to}$

Μπορούμε, φυσικά, να συνθέσουμε μία σχέση με τον εαυτό της και μάλιστα περισσότερες από μία φορά.

**1.1.11 Ορισμός.** Για $n \geq 0$ : $\overset{n+1}{\to} \overset{\text{ορ.}}{=} \to \circ \overset{n}{\to}$.

*1.1.12 Παρατήρηση.* Δεδομένης της προσεταιριστικότητας της σύνθεσης, ο παραπάνω ορισμός είναι ισοδύναμος με τον $\overset{n+1}{\to} \overset{\text{ορ.}}{=} \overset{n}{\to} \circ \to$ και γενικά μπορούμε να χρησιμοποιούμε οποιονδήποτε από τους δύο, ανάλογα με την περίσταση.

**1.1.13 Ορισμός.** $\overset{+}{\to} \overset{\text{ορ.}}{=} \bigcup_{k>0} \overset{k}{\to}$

**1.1.14 Ορισμός.** $\overset{*}{\to} \overset{\text{ορ.}}{=} \bigcup_{k\geq 0} \overset{k}{\to}$. Αν $x \overset{*}{\to} y$, λέμε ότι το $x$ *παράγει* το $y$ ή ότι το $x$ *ανάγεται* στο $y$.

Αντί για το $\overset{*}{\to}$, χρησιμοποιείται επίσης ο συμβολισμός $\twoheadrightarrow$. Αυτός ο συμβολισμός είναι ιδιαίτερα χρήσιμος κατά την δημιουργία διαγραμμάτων, στα οποία τα βέλη δεν είναι σταθερά προσανατολισμένα, οπότε δεν είναι πάντοτε εύκολη η τοποθέτηση του αστερίσκου (βλέπε [59]).

*1.1.15 Παρατήρηση.* Πρακτικά, αν ισχύει $a \xrightarrow{*} b$, τότε ισοδύναμα υπάρχει μία πιθανώς κενή, πεπερασμένη ακολουθία βημάτων αναγωγής $a \equiv x_0 \to x_1 \to \cdots \to x_n \equiv b$.

Κάποιες φορές είναι χρήσιμο να θεωρήσουμε όλους τους τρόπους αναγωγής, των οποίων το πλήθος των βημάτων είναι φραγμένο από κάποιον αριθμό, έστω $n$ :

**1.1.16 Ορισμός.** $\xrightarrow{\leq n} \stackrel{\text{ορ.}}{=} \bigcup_{0 \leq k \leq n} \xrightarrow{k}$

*1.1.17 Παρατήρηση.* Προφανώς ισχύει $\xrightarrow{\leq 1} = \xrightarrow{=}$.

**1.1.18 Ορισμός** (Αντιστροφή)**.** $R^{-1} \stackrel{\text{ορ.}}{=} \{(y, x) \mid (x, y) \in R\}$. Συνηθίζουμε να γράφουμε $\leftarrow$ αντί για $\to^{-1}$. Η $\leftarrow = \to^{-1}$ ονομάζεται *αντίστροφη* της $\to$. Γενικά, για να υποδηλώσουμε την αντίστροφη μιας σχέσης χρησιμοποιούμε το κατοπτρικό σύμβολο της εν λόγω σχέσης.

Η παρακάτω πρόταση συνδέει την αντιστροφή με την σύνθεση :

**1.1.19 Πρόταση.** $(R_1 \circ R_2)^{-1} = R_2^{-1} \circ R_1^{-1}$

**1.1.20 Ορισμός.** $\leftrightarrow \stackrel{\text{ορ.}}{=} \to \cup \leftarrow$

**1.1.21 Ορισμός.** $\overset{*}{\leftrightarrow} \stackrel{\text{ορ.}}{=} (\leftrightarrow)^*$. Η σχέση $\overset{*}{\leftrightarrow}$ ονομάζεται και σχέση *μετατρεψιμότητας*. Αν $x \overset{*}{\leftrightarrow} y$, λέμε ότι τα $x$, $y$ είναι *μετατρέψιμα*.

## 1.2 Ιδιότητες σχέσεων

**1.2.1 Ορισμός** (Ομοδυναμία ή αυτοδυναμία)**.** Η $\to$ είναι *ομοδύναμη* ή *αυτοδύναμη* (στα αγγλικά : idempotent) αν και μόνον αν $\to \circ \to = \to$.

**1.2.2 Παράδειγμα.** Για κάθε σχέση $\to$, η σχέση $\xrightarrow{*}$ (βλέπε ορισμό 1.1.14) είναι ομοδύναμη.

**1.2.3 Ορισμός** (Μεταβατικότητα)**.** Η $\to$ είναι *μεταβατική* αν και μόνον αν $\to \circ \to \subseteq \to$, δηλαδή αν και μόνον αν για κάθε $x$, $y$, $z$, για τα οποία ισχύει $x \to y$ και $y \to z$, ισχύει επίσης $x \to z$.

*1.2.4 Παρατήρηση.* Η ιδιότητα της ομοδυναμίας συνεπάγεται αυτήν της μεταβατικότητας, αλλά όχι το αντίστροφο.

**1.2.5 Ορισμός** (Συμμετρικότητα)**.** Η $\to$ είναι *συμμετρική* αν και μόνον αν $\to = \leftarrow$, δηλαδή αν και μόνον αν για κάθε $x$, $y$ για τα οποία ισχύει $x \to y$, ισχύει επίσης $y \to x$.

**1.2.6 Ορισμός** (Αντισυμμετρικότητα)**.** Η $\to$ είναι *αντισυμμετρική* αν και μόνον αν $\to \cap \leftarrow \subseteq \overset{0}{\to}$, δηλαδή αν και μόνον αν για κάθε $x$, $y$ για τα οποία ισχύει $x \to y$ και $y \to x$, τα $x$ και $y$ είναι ίσα.

**1.2.7 Ορισμός** (Ανακλαστικότητα)**.** Η $\to$ είναι *ανακλαστική* αν και μόνον αν $\overset{0}{\to} \subseteq \to$, δηλαδή αν και μόνον αν για κάθε $x \in A$, ισχύει $x \to x$.

**1.2.8 Ορισμός.** Το *κλείσιμο* μίας σχέσης $R$ ως προς μία *ιδιότητα* $P$ είναι η μικρότερη[1] σχέση που περιέχει την $R$ και έχει την ιδιότητα $P$.

**1.2.9 Παράδειγμα.** Έστω η μεταβατική ιδιότητα και μία σχέση $R$. Το μεταβατικό κλείσιμο της $R$ είναι η μικρότερη μεταβατική σχέση που περιέχει την $R$.

*1.2.10 Παρατήρηση.* Για να δείξουμε ότι μία σχέση $R_{\text{κλ.}}$ είναι το κλείσιμο μίας σχέσης $R$ ως προς μία ιδιότητα $P$, αρκεί να δείξουμε ότι κάθε σχέση που περιέχει την $R$ και έχει την ιδιότητα $P$, αναγκαστικά περιέχει και την $R_{\text{κλ.}}$.

**1.2.11 Πρόταση.** *Η* $\overset{+}{\to}$ *είναι το* μεταβατικό κλείσιμο *της* $\to$*.*

*Απόδειξη.* Έστω σχέση $\to'$ που περιέχει την $\to$ και έχει την ιδιότητα της μεταβατικότητας. Για να δείξουμε ότι η $\overset{+}{\to}$ περιέχεται στην $\to'$ αρκεί, σύμφωνα με τον ορισμό 1.1.13, να δείξουμε ότι για κάθε $n \geq 1$, $x \overset{n}{\to} y$ συνεπάγεται $x \to' y$. Αυτό μπορεί να δειχθεί με επαγωγή στο $n$ της $\overset{n}{\to}$. Για $n = 1$, η $\overset{1}{\to}$ είναι η $\to$, άρα περιέχεται στην $\to'$. Έστω ότι για $n = k$ η $\overset{k}{\to}$ περιέχεται στην $\to'$. Έτσι, ισχύει $\to \subseteq \to'$ και $\overset{k}{\to} \subseteq \to'$, οπότε χρησιμοποιώντας την ιδιότητα της σύνθεσης από την πρόταση 1.1.7δ : $\to \circ \overset{k}{\to} \subseteq \to' \circ \to'$. Όμως, αφού η $\to'$ είναι μεταβατική, ισχύει (βλέπε ορισμό 1.2.3) $\to' \circ \to' \subseteq \to'$ και τελικά, λόγω του ορισμού 1.1.11, $\overset{k+1}{\to} \subseteq \to'$. □

**1.2.12 Πρόταση.** *Η* $\leftrightarrow$ *είναι το* συμμετρικό κλείσιμο *της* $\to$*.*

*Απόδειξη.* Έστω σχέση $\to'$ που περιέχει την $\to$ και έχει την ιδιότητα της συμμετρικότητας. Αν $x \leftrightarrow y$ τότε θα πρέπει (βλέπε ορισμό 1.1.20), είτε α) $x \to y$, οπότε $x \to' y$, είτε β) $y \to x$ και όχι $x \to y$, οπότε $y \to' x$ και αφού η $\to'$ είναι συμμετρική, θα έχουμε $x \to' y$. Σε κάθε περίπτωση, για κάθε $x, y \in A$, αν $x \leftrightarrow y$, τότε $x \to' y$, δηλαδή $\leftrightarrow \subseteq \to'$. □

1. Η έννοια της μικρότερης σχέσης είναι η εξής : οποιαδήποτε άλλη σχέση που έχει τις ζητούμενες ιδιότητες (δηλαδή περιέχει την $R$ και έχει την ιδιότητα $P$) περιέχει την μικρότερη σχέση. Ο ορισμός προϋποθέτει σιωπηλά την μοναδικότητα της μικρότερης αυτής σχέσης, κάτι που ενδέχεται να μην ισχύει πάντοτε. Για αυτό λοιπόν, όποτε χρησιμοποιούμε την έννοια του κλεισίματος, θα θεωρούμε τέτοιες ιδιότητες ώστε να εξασφαλίζεται η μοναδικότητα του κλεισίματος.

**1.2.13 Πρόταση.** *Η* $\overset{=}{\to}$ *είναι το* ανακλαστικό κλείσιμο *της* $\to$.

*Απόδειξη.* Έστω σχέση $\to'$ που περιέχει την $\to$ και έχει την ιδιότητα της ανακλαστικότητας. Αν $x \overset{=}{\to} y$ τότε θα πρέπει (βλέπε ορισμό 1.1.10), είτε α) $x \to y$, οπότε $x \to' y$, είτε β) τα $x$ και $y$ είναι το ίδιο στοιχείο και δεν ισχύει $x \to y$, οπότε $x \overset{0}{\to} y$ και αφού η $\to'$ είναι ανακλαστική, θα έχουμε $x \to' y$. Σε κάθε περίπτωση, για κάθε $x, y \in A$, αν $x \overset{=}{\to} y$, τότε $x \to' y$, δηλαδή $\overset{=}{\to} \subseteq \to'$. □

**1.2.14 Πρόταση.** *Η* $\overset{*}{\to}$ *είναι το* μεταβατικό ανακλαστικό κλείσιμο[2] *της* $\to$.

*Απόδειξη.* Η απόδειξη είναι παρόμοια με έναν συνδυασμό των αποδείξεων των προτάσεων 1.2.11 και 1.2.13. □

# 1.3 Ισοδυναμίες

**1.3.1 Ορισμός** (Σχέση ισοδυναμίας)**.** Μία σχέση λέγεται *σχέση ισοδυναμίας* αν και μόνον αν έχει τις ιδιότητες της μεταβατικότητας, της συμμετρικότητας και της ανακλαστικότητας. Μία σχέση ισοδυναμίας συχνά συμβολίζεται με $\sim$.

**1.3.2 Ορισμός.** Έστω $I \subseteq \mathcal{P}(A)$, το $I$ ονομάζεται *επικάλυψη* του $A$, αν $A = \bigcup I$. Μία επικάλυψη $I$ του $A$ ονομάζεται *διαμέριση* αν $\emptyset \notin I$ και ανά δύο τα στοιχεία του $I$ είναι ξένα μεταξύ τους.

**1.3.3 Ορισμός.** Έστω σχέση ισοδυναμίας $R$ στο $A$. Για κάθε $x \in A$ ορίζεται το *σύνολο ισοδυναμίας*[3] του $x$ ως εξής : $x/R \overset{\text{ορ.}}{=} \{y \mid (x, y) \in R\}$. Το σύνολο όλων των συνόλων ισοδυναμίας (για κάθε $x \in A$) συμβολίζεται με $A/R$.

**1.3.4 Πρόταση.** *Τα σύνολα ισοδυναμίας που ορίζει μία σχέση ισοδυναμίας επί του συνόλου* $A$ *αποτελούν μία διαμέριση του* $A$*. Αντίστροφα, κάθε διαμέριση του* $A$ *ορίζει μία σχέση ισοδυναμίας επί του* $A$*, της οποίας τα σύνολα ισοδυναμίας είναι ακριβώς αυτή η διαμέριση.*

Παρακάτω θα δείξουμε ότι η σχέση $\overset{*}{\leftrightarrow}$ είναι σχέση ισοδυναμίας. Για αυτό θα μας βοηθήσει το παρακάτω λήμμα :

**1.3.5 Λήμμα.** *Αν* $\eta \to$ *είναι συμμετρική τότε είναι και* $\eta \overset{*}{\to}$.

2. Το κλείσιμο μίας σχέσης ως προς δύο ή περισσότερες ιδιότητες σημαίνει ότι η προκύπτουσα σχέση έχει ταυτόχρονα και τις δύο ή περισσότερες ιδιότητες.

3. Χρησιμοποιείται επίσης συχνά ο όρος «κλάση ισοδυναμίας». Αποφύγαμε την χρήση όμως αυτού του όρου, αν και φαίνεται ότι επικρατεί, επειδή στην συνολοθεωρία ο όρος κλάση έχει ευρύτερη έννοια από αυτήν του συνόλου.

*Απόδειξη.* Έστω ότι ισχύει $x \overset{*}{\to} y$. Τότε, από την παρατήρηση 1.1.15, υπάρχει πιθανώς κενή ακολουθία πεπερασμένου μήκους : $x \equiv s_0 \to \cdots \to s_n \equiv y$. Επειδή όμως η $\to$ είναι συμμετρική ισχύει και $y \equiv s_n \to \cdots \to s_0 \equiv x$ ή αλλιώς $y \overset{*}{\to} x$, δηλαδή και η $\overset{*}{\to}$ είναι συμμετρική. □

**1.3.6 Πρόταση.** *Η σχέση $\overset{*}{\leftrightarrow}$ είναι σχέση ισοδυναμίας.*

*Απόδειξη.* Από το λήμμα 1.3.5, αφού η $\leftrightarrow$ είναι *συμμετρική*, είναι και η $\overset{*}{\leftrightarrow}$. Όμως, η $\overset{*}{\leftrightarrow}$ είναι το μεταβατικό ανακλαστικό κλείσιμο της $\leftrightarrow$, άρα είναι επιπλέον και *μεταβατική* και *ανακλαστική*. □

Μάλιστα, ισχύει κάτι ακόμα πιο ισχυρό για την σχέση $\overset{*}{\leftrightarrow}$ :

**1.3.7 Πρόταση.** *Η $\overset{*}{\leftrightarrow}$ είναι το* συμμετρικό μεταβατικό ανακλαστικό κλείσιμο *της $\to$.*

*Απόδειξη.* Έστω σχέση $\to'$ που περιέχει την $\to$ και έχει τις ιδιότητες της συμμετρικότητας, της μεταβατικότητας και της ανακλαστικότητας. Η $\overset{*}{\leftrightarrow}$ (βλέπε ορισμό 1.1.21) γράφεται : $\overset{0}{\to} \cup \leftrightarrow \cup \overset{2}{\leftrightarrow} \cup \cdots \cup \overset{n}{\leftrightarrow} \cup \cdots$

Αν $x \overset{0}{\to} y$, αφού η $\overset{*}{\leftrightarrow}$ είναι ανακλαστική έχουμε και $x \to' y$.

Αν $x \leftrightarrow y$, ισχύει ένα από τα παρακάτω : α) $x \to y$, οπότε $x \to' y$, αφού η $\to'$ περιέχει την $\to$, β) $y \to x$, οπότε επειδή η $\to'$ είναι συμμετρική $x \to' y$.

Τώρα, επαγωγικά δείχνουμε ότι, για κάθε $n$, η $x \overset{n}{\leftrightarrow} y$ συνεπάγεται $x \to' y$. Την βάση, δηλαδή για $n = 1$, την δείξαμε παραπάνω. Υποθέτουμε ότι η $\overset{k}{\leftrightarrow}$ περιέχεται στην $\to'$. Έχουμε :

$$\begin{aligned} \overset{k+1}{\leftrightarrow} &= \leftrightarrow \circ \overset{k}{\leftrightarrow} && \text{από τον ορισμό 1.1.11} \\ &\subseteq \to' \circ \to' && \text{αφού } \leftrightarrow \subseteq \to', \overset{k}{\leftrightarrow} \subseteq \to' \text{ (από υπόθεση)} \\ &\subseteq \to' && \text{λόγω μεταβατικής ιδιότητας της } \to' \end{aligned}$$ □

# 1.4 Διατάξεις

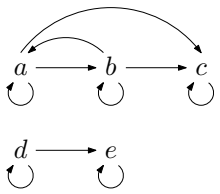
Μία σχέση σε ένα σύνολο επιβάλλει κατά κάποιον τρόπο μία «διάταξη» στο σύνολο, αφού για κάθε ζεύγος στοιχείων (του συνόλου) που ανήκει στην σχέση, το πρώτο στοιχείο του ζεύγους ανάγεται στο δεύτερο. Ανάλογα με κάποιες επιπλέον ιδιότητες που έχει μία σχέση, ορίζουμε διαφόρων ειδών διατάξεις.

Η ενότητα αυτή βασίζεται στην αντίστοιχη ενότητα στο [104].

**1.4.1 Ορισμός** (Σχεδόν διάταξη)**.** Μία σχέση σε ένα σύνολο $A$ είναι *σχεδόν διάταξη* (quasi-order) όταν είναι ανακλαστική και μεταβατική.

Μία σχεδόν διάταξη συμβολίζεται συνήθως με $\gtrsim$ (ή και με $\geq$). Επίσης, γράφουμε $\lesssim$ αντί για $\gtrsim^{-1}$.

**1.4.2 Παράδειγμα.** Έστω $A = \{a, b, c, d, e\}$ και σχέση $\gtrsim$ για την οποία ισχύουν ακριβώς τα εξής : $a \gtrsim a$, $a \gtrsim b$, $a \gtrsim c$, $b \gtrsim a$, $b \gtrsim b$, $b \gtrsim c$, $c \gtrsim c$, $d \gtrsim d$, $d \gtrsim e$, $e \gtrsim e$. Η παραπάνω σχέση παριστάνεται σε μορφή κατευθυνόμενου γράφου στο σχήμα 1.1, όπου οι κατευθυνόμενες ακμές συνδέουν στοιχεία που σχετίζονται μέσω της $\gtrsim$ (προφανώς παίζει ρόλο και η κατεύθυνση της κάθε ακμής). Μπορεί εύκολα να αποδειχθεί ότι η σχέση $\gtrsim$ είναι μία σχεδόν διάταξη.




Σχήμα 1.1 – Μία σχεδόν διάταξη

*1.4.3 Παρατήρηση.* Από τον ορισμό 1.3.1, προκύπτει ότι μία σχέση είναι ισοδυναμία όταν είναι σχεδόν διάταξη και συμμετρική.

Δίνουμε χωρίς απόδειξη την παρακάτω πρόταση :

**1.4.4 Πρόταση.** *Για κάθε σχεδόν διάταξη $\gtrsim$, η σχέση $\sim \overset{\text{ορ.}}{=} \gtrsim \cap \lesssim$ είναι σχέση ισοδυναμίας.*

Πρέπει πάντως να σημειώσουμε ότι η παραπάνω σχέση δεν είναι το συμμετρικό μεταβατικό ανακλαστικό κλείσιμο της $\gtrsim$, αφού στην γενική περίπτωση η $\sim$ δεν περιέχει την $\gtrsim$. Η χρησιμότητα της $\sim$ θα φανεί παρακάτω όταν αναφερθούμε σε μερικές διατάξεις.

Για την σχεδόν διάταξη του παραδείγματος 1.4.2 (βλέπε και σχήμα 1.1), η σχέση ισοδυναμίας $\sim$ δίνει σύνολα ισοδυναμίας ως εξής :

$$A/\sim = \{\{a, b\}, \{c\}, \{d\}, \{e\}\}$$

Αν, αντί για την ιδιότητα της συμμετρικότητας, απαιτήσουμε για μία σχεδόν διάταξη την ιδιότητα της αντισυμμετρικότητας, προκύπτει μία άλλου είδους διάταξη :

**1.4.5 Ορισμός** (Μερική διάταξη)**.** Μία σχέση είναι *μερική διάταξη* (partial order) όταν είναι σχεδόν διάταξη και αντισυμμετρική.

Τις μερικές διατάξεις συμβολίζουμε συνήθως με $\geq$, οπότε κατά τα συνήθη γράφουμε και $\leq$ αντί για $\geq^{-1}$.

Υπενθυμίζουμε ότι η ιδιότητα της αντισυμμετρικότητας για μία σχέση $R$ επιβάλλει για δύο διαφορετικά στοιχεία $x$ και $y$ να ισχύει το *πολύ* ένα από τα $x\ R\ y$ και $y\ R\ x$. Η σχέση του παραδείγματος 1.4.2 δεν είναι μερική διάταξη επειδή ισχύει ταυτόχρονα $a \gtrsim b$ και $b \gtrsim a$, οπότε η $\gtrsim$ δεν είναι αντισυμμετρική. Αν δεν ίσχυε ακριβώς ένα από τα $a \gtrsim b$, $b \gtrsim a$, δηλαδή αν καταργούσαμε μόνον μία από τις δύο ακμές μεταξύ $a$ και $b$ στο σχήμα 1.1, τότε η $\gtrsim$ θα ήταν μερική διάταξη.

Θα μπορούσαμε να πούμε ότι στοιχεία όπως τα $a$ και $b$ παραπάνω είναι πειστήρια για την απόρριψη μίας σχεδόν διάταξης ως υποψήφιας μερικής διάταξης. Όμως, η ισοδυναμία $\sim \overset{\text{op.}}{=} \gtrsim \cap \lesssim$ (από την πρόταση 1.4.4) «συμπτύσσει» τέτοια στοιχεία σε ίδια σύνολα ισοδυναμίας. Αν επεκτείνουμε λοιπόν την σχέση $\gtrsim$ από το σύνολο $A$ στο σύνολο $A/\!\sim$, έτσι ώστε δύο σύνολα ισοδυναμίας να σχετίζονται αν και μόνον αν κάθε τους στοιχείο σχετίζεται (μπορεί να αποδειχθεί εύκολα ότι αυτό γίνεται κατά μοναδικό τρόπο χωρίς αντιφάσεις), τότε προκύπτει αναγκαστικά μία μερική διάταξη των συνόλων ισοδυναμίας, επειδή αποκλείεται για δύο στοιχεία $x, y \in A$ σε διαφορετικά σύνολα ισοδυναμίας να ισχύει ταυτόχρονα $x \gtrsim y$ και $y \gtrsim x$. Έτσι, στο παράδειγμα 1.4.2, πέραν των ανακλαστικών εκφάνσεων της $\gtrsim$ (εννοείται της επέκτασης στο σύνολο $A/\!\sim$), ισχύει $\{a, b\} \gtrsim \{c\}$ και $\{d\} \gtrsim \{e\}$.

Μπορεί λοιπόν να αποδειχθεί και τυπικά η παρακάτω πρόταση :

**1.4.6 Πρόταση.** *Η σχεδόν διάταξη $\gtrsim$ επιβάλλει μία μερική διάταξη στα σύνολα ισοδυναμίας της $\sim\ = \gtrsim \cap \lesssim$.*

**1.4.7 Ορισμός.** Μία σχεδόν διάταξη (γενικότερα μία σχέση) είναι *ολική ή γραμμική* στο σύνολο $A$ όταν $\gtrsim \cup \lesssim = A \times A$.

Προφανώς, αν μία σχεδόν διάταξη $\gtrsim$ (ή και γενικότερα μία σχέση) είναι ολική, τότε για κάθε δύο στοιχεία $x, y \in A$ θα ισχύει τουλάχιστον ένα εκ των $x \gtrsim y$ και $x \lesssim y$ (ενδέχεται πάντως να ισχύουν και τα δύο). Αυτό σημαίνει πρακτικά ότι μπορούμε να διατάξουμε οποιαδήποτε δύο στοιχεία μέσω της $\gtrsim$. Αντιθέτως, η σχεδόν διάταξη του παραδείγματος 1.4.2 δεν είναι ολική αφού για παράδειγμα δεν μπορούμε να διατάξουμε τα στοιχεία $c$ και $d$. Αυτό μπορεί να φανεί και από τον γράφο του σχήματος 1.1, ο οποίος έχει δύο συνεκτικές συνιστώσες (τα $a$, $b$, $c$ στην μία και τα $d$, $e$ στην άλλη). Σε μία ολική σχέση, ο αντίστοιχος γράφος έχει το πολύ μία συνεκτική συνιστώσα (αυτό δεν σημαίνει ότι κάθε σχέση που αντιστοιχεί σε γράφο με το πολύ μία συνεκτική συνιστώσα είναι ολική).

Τελικά, δίνουμε την έννοια της αποκαλούμενης (απλώς) διάταξης :

**1.4.8 Ορισμός** (Διάταξη)**.** Μία σχέση λέγεται *διάταξη* όταν είναι μερική διάταξη και ολική.

**1.4.9 Ορισμός.** Το *γνήσιο μέρος* $>$ μίας σχεδόν διάταξης $\gtrsim$ ορίζεται ως εξής : $> \overset{\text{ορ.}}{=} \gtrsim - \lesssim$. Ο ορισμός είναι επίσης άμεσος για τις μερικές διατάξεις και τις διατάξεις, αφού πρόκειται απλώς για περιπτώσεις σχεδόν διατάξεων. Επίσης, κατά τα γνωστά, γράφουμε $<$ αντί για $>^{-1}$.

*1.4.10 Παρατήρηση.* Αν έχουμε μία διάταξη σε ένα σύνολο $A$, τότε για οποιαδήποτε δύο στοιχεία $x$, $y$ του συνόλου ισχύει ακριβώς ένα από τα παρακάτω : είτε $x < y$, είτε $x = y$, είτε $x > y$.

# Κεφάλαιο 2

# Αφηρημένα συστήματα μεταγραφής

Ένα αφηρημένο σύστημα μεταγραφής είναι ένα σύνολο εφοδιασμένο με μία σχέση. Ένα αφηρημένο σύστημα μεταγραφής είναι στην ουσία ένα σύστημα μεταγραφής, στο οποίο όμως θεωρείται τελείως αδιάφορη η εσωτερική δομή του. Οι κανόνες μεταγραφής ορίζονται από την σχέση, η οποία επιβάλλει μία εξωτερική δομή στο σύνολο.

Βέβαια, η εσωτερική δομή ενός συστήματος μεταγραφής ενδέχεται να παίζει σημαντικό ρόλο στην συμπεριφορά αυτού. Πολλές χρήσιμες ιδιότητες, όμως, είναι ανεξάρτητες της εσωτερικής δομής και έτσι επιβάλλεται η μελέτη των αφηρημένων συστημάτων μεταγραφής.

Η παρουσίαση των αφηρημένων συστημάτων μεταγραφής βασίζεται κυρίως στην αντίστοιχη ενότητα του [59]. Άλλες αναφορές είναι τα [1, 52, 77, 89].

## 2.1 Συστήματα μεταγραφής

**2.1.1 Ορισμός.** Ένα (*αφηρημένο*) *σύστημα μεταγραφής* είναι μία δομή $\mathcal{A} = \langle A, (\to_a)_{a\in I}\rangle$ που αποτελείται από ένα σύνολο $A$ και μία ακολουθία δυαδικών σχέσεων $\to_a$, με $a \in I$.

Ο παραπάνω ορισμός δίνεται στο [59]. Πάντως, μπορούμε να θεωρήσουμε την σχέση $\to \stackrel{\text{ορ.}}{=} \bigcup_{a\in I} \to_a$, οπότε σε κάθε περίπτωση έχουμε ένα σύνολο και μία σχέση μεταγραφής. Με αυτήν την θεώρηση, οι ιδιότητες της παραπάνω σχέσης ταυτίζονται με τις ιδιότητες του συστήματος μεταγραφής. Παρακάτω θα δίνουμε τους ορισμούς των ιδιοτήτων και τις προτάσεις για ένα από τα δύο, είτε την σχέση $\to$, είτε το σύστημα μεταγραφής.

Πολλές φορές, για να περιγράψουμε ένα σύστημα μεταγραφής θα δίνουμε ακριβώς τα ζεύγη των στοιχείων από το $A$ για τα οποία ισχύει η σχέση $\to$.

Για παράδειγμα, με :

$$a \to b, \quad a \to c, \quad b \to d, \quad c \to d$$

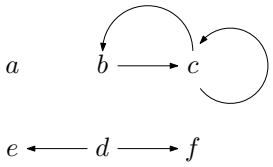
εννοείται το σύστημα :

$$\mathcal{A} = \langle \{a, b, c, d\}, \{(a, b), (a, c), (b, d), (c, d)\} \rangle$$

Επίσης, ένα σύστημα μεταγραφής μπορεί να παρασταθεί με την μορφή *γράφου*[1] : Τα στοιχεία του $A$ είναι οι κορυφές του γράφου και οι (κατευθυνόμενες) ακμές τα ζεύγη της σχέσης $R$. Για παράδειγμα, το σύστημα :

$$\mathcal{A} = \langle \{a, b, c, d, e, f\}, \{(b, c), (c, b), (c, c), (d, e), (e, f)\} \rangle$$

παριστάνεται στο σχήμα 2.1.




Σχήμα 2.1 – Γράφος συστήματος μεταγραφής

*2.1.2 Παρατήρηση.* Ο γράφος που αντιστοιχεί σε ένα σύστημα μεταγραφής προφανώς ενδέχεται να έχει άπειρες κορυφές ή/και ακμές, όπως αυτός του σχήματος 2.2 για το σύστημα $\mathcal{A} = \langle \mathbb{N}, R \rangle$, με $R \stackrel{\text{ορ.}}{=} \{(x, y) \mid y = x + 1\}$.

$$0 \to 1 \to 2 \to \cdots \to \cdots$$

Σχήμα 2.2 – Άπειρος γράφος συστήματος μεταγραφής

## 2.2 Πρόγονοι, απόγονοι, σύνολο ισοδυναμίας

**2.2.1 Ορισμός.** Το σύνολο των *απογόνων* ενός στοιχείου $x \in A$ είναι :

$$\Delta^*_{\mathcal{A}}(x) \stackrel{\text{ορ.}}{=} \{y \in A \mid x \xrightarrow{*} y\}$$

**2.2.2 Ορισμός.** Το σύνολο των *προγόνων* ενός στοιχείου $x \in A$ είναι :

$$\nabla^*_{\mathcal{A}}(x) \stackrel{\text{ορ.}}{=} \{y \in A \mid y \xrightarrow{*} x\}$$

1. Για μία σύντομη και γενική αναφορά στην έννοια του γράφου, βλέπε την τελευταία ενότητα αυτού του κεφαλαίου.

Από τον ορισμό είναι προφανές ότι στο σύνολο των απογόνων ή και προγόνων ενός στοιχείου $x$ ανήκει και το ίδιο το στοιχείο $x$, επειδή $x \xrightarrow{0} x$. Έτσι, είναι χρήσιμο κάποιες φορές να θεωρήσουμε τα σύνολα *γνησίων απογόνων* και *γνησίων προγόνων* που ορίζονται ως εξής :

**2.2.3 Ορισμός.**

$$\Delta^+_{\mathcal{A}}(x) \overset{\text{ορ.}}{=} \{y \in A \mid x \xrightarrow{+} y\} \quad \text{και} \quad \nabla^+_{\mathcal{A}}(x) \overset{\text{ορ.}}{=} \{y \in A \mid y \xrightarrow{+} x\}$$

Εναλλακτικά, οι γνήσιοι απόγονοι ονομάζονται *διάδοχοι* (από τον αγγλικό όρο successors) και οι γνήσιοι πρόγονοι *προκάτοχοι* (από τον αγγλικό όρο predecessors). Για αυτό, κάποιες φορές συμβολίζονται και ως εξής : $\mathrm{pred}(x)$, $\mathrm{succ}(x)$ αντιστοίχως.

Σε άλλες περιπτώσεις, είναι χρήσιμο να θεωρήσουμε απογόνους (ή διαδόχους) και προγόνους (ή προκατόχους) σε ένα βήμα από ένα στοιχείο $x \in A$ :

**2.2.4 Ορισμός.** Ένα στοιχείο $y \in A$ είναι *άμεσος απόγονος* (*άμεσος διάδοχος*) του $x \in A$ αν το $x$ ανάγεται στο $y$, δηλαδή $x \to y$. Ένα στοιχείο $y \in A$ είναι *άμεσος πρόγονος* (*άμεσος προκάτοχος*) του $x \in A$ αν το $y$ ανάγεται στο $x$, δηλαδή $y \to x$.

Γενικά, πάντως πρέπει να παρατηρήσουμε ότι δεν υπάρχει πρότυπη ορολογία, ούτε πρότυπος συμβολισμός, όσον αφορά τους απογόνους, διαδόχους κ.τ.λ.

Αν τώρα θεωρήσουμε την σχέση μετατρεψιμότητας μπορούμε να κατασκευάσουμε παρομοίως όπως στην περίπτωση των απογόνων και προγόνων ένα σύνολο, για κάθε στοιχείο του $A$, ως εξής :

**2.2.5 Ορισμός.** Το *σύνολο ισοδυναμίας*[2] ενός στοιχείου $x \in A$ είναι :

$$[x]^*_{\mathcal{A}} \overset{\text{ορ.}}{=} \{y \in A \mid x \overset{*}{\leftrightarrow} y\}$$

Πολλές φορές παραλείπουμε τον δείκτη $\mathcal{A}$ όταν είναι σαφές ή δεν έχει σημασία σε ποιο αφηρημένο σύστημα μεταγραφής αναφερόμαστε. Επίσης, για σύνολο $B \subseteq A$, ορίζονται ανάλογα τα $\Delta^*_{\mathcal{A}}(B)$, $\nabla^*_{\mathcal{A}}(B)$, $[B]^*_{\mathcal{A}}$ κ.τ.λ.

**2.2.6 Ορισμός** (Ζευξιμότητα)**.** Για κάθε σχέση $\to$ η σχέση *ζευξιμότητας* ορίζεται ως εξής : $\downarrow \overset{\text{ορ.}}{=} \xrightarrow{*} \circ \xleftarrow{*}$.

Αν $x \downarrow y$ λέμε ότι τα $x$ και $y$ είναι *ζεύξιμα*. Προφανώς, $x \downarrow y$ αν και μόνον αν $\Delta^*(x) \cap \Delta^*(y) \neq \emptyset$, δηλαδή τα δύο στοιχεία έχουν κάποιον κοινό απόγονο.

2. Ο ορισμός είναι ταυτόσημος με τον 1.3.3, απλά χρησιμοποιείται διαφορετικός συμβολισμός.

**2.2.7 Ορισμός** (Συναντησιμότητα)**.** Για κάθε σχέση $\rightarrow$ η σχέση *συναντησιμότητας* ορίζεται ως εξής : $\uparrow \stackrel{ορ.}{=} \stackrel{*}{\leftarrow} \circ \stackrel{*}{\rightarrow}$.

Αν $x \uparrow y$ λέμε ότι τα $x$ και $y$ είναι *συναντήσιμα*. Προφανώς, $x \uparrow y$ αν και μόνον αν $\nabla^*(x) \cap \nabla^*(y) \neq \emptyset$, δηλαδή τα δύο στοιχεία έχουν κάποιον κοινό πρόγονο.

## 2.3 Η (μη) αιτιοκρατία

Μπορούμε να αντιστοιχίσουμε ένα σύστημα υπολογισμού σε ένα σύστημα μεταγραφής $\mathcal{A} = \langle A, \rightarrow \rangle$ ως εξής : Για κάθε δυνατό στοιχείο εισόδου και κάθε δυνατό ενδιάμεσο ή τελικό αποτέλεσμα του υπολογιστικού συστήματος θεωρούμε και ένα διαφορετικό στοιχείο στο σύνολο $A$. Όλα τα παραπάνω στοιχεία ονομάζονται *αντικείμενα υπολογισμού*. Αν ένα αντικείμενο υπολογισμού μετατρέπεται σε ένα άλλο, σε *ένα βήμα υπολογισμού*, σύμφωνα με την υπολογιστική διαδικασία του υπολογιστικού συστήματος, τότε θεωρούμε στην σχέση $R$ το διατεταγμένο ζεύγος του πρώτου με το δεύτερο αντικείμενο και αυτό το κάνουμε για όλα τα δυνατά υπολογιστικά βήματα και για όλα τα δυνατά αντικείμενα.

Είναι επίσης γνωστό ότι αν για κάθε αντικείμενο υπολογισμού που δεν είναι τελικό αποτέλεσμα, υπάρχει ακριβώς ένα επόμενο (σε ένα βήμα υπολογισμού) αντικείμενο στην διαδικασία υπολογισμού, δηλαδή αν από οποιοδήποτε αντικείμενο υπολογισμού δεν υπάρχουν *αποκλίνουσες διαδικασίες υπολογισμού*, το υπολογιστικό σύστημα ονομάζεται *αιτιοκρατικό* (ή ντετερμινιστικό). Σύμφωνα με την πιο πάνω αντιστοιχία των υπολογιστικών συστημάτων με συστήματα μεταγραφής, δίνουμε παρακάτω τον αντίστοιχο ορισμό στα πλαίσια των συστημάτων μεταγραφής.

**2.3.1 Ορισμός.** Ένα σύστημα μεταγραφής είναι *αιτιοκρατικό* (ντετερμινιστικό) όταν το σύνολο των αμέσων απογόνων οποιουδήποτε στοιχείου είναι είτε κενό, είτε μονοσύνολο.

Αν δεν υπάρχει περιορισμός στο μέγεθος των συνόλων αμέσων απογόνων, τότε το σύστημα ενδέχεται να μην είναι αιτιοκρατικό, οπότε σε αυτήν την περίπτωση ονομάζεται *μη αιτιοκρατικό*.

Πρακτικά, σε ένα αιτιοκρατικό σύστημα, δεδομένου ενός οποιουδήποτε στοιχείου, είτε δεν μπορούμε να εφαρμόσουμε κανένα βήμα μεταγραφής, είτε αυτό μεταγράφεται σε ένα μόνον άλλο στοιχείο.

Για παράδειγμα, το σύστημα του σχήματος 2.1 είναι μη αιτιοκρατικό επειδή το $d$ μπορεί να μεταγραφεί είτε στο $e$ είτε στο $f$. Αντίθετα, το σύστημα του σχήματος 2.2 είναι αιτιοκρατικό.

## 2.4 Υποσυστήματα, γράφοι αναγωγής

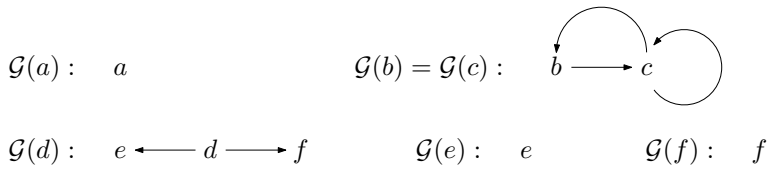
**2.4.1 Ορισμός** (Υποσύστημα)**.** Έστω $\mathcal{A} = \langle A, \rightarrow_\alpha \rangle$ και $\mathcal{B} = \langle B, \rightarrow_\beta \rangle$ δύο αφηρημένα συστήματα μεταγραφής. Το $\mathcal{A}$ είναι *υποσύστημα* του $\mathcal{B}$ (συμβολίζεται ως $\mathcal{A} \subseteq \mathcal{B}$) αν :

- $A \subseteq B$
- η $\alpha$ είναι ο περιορισμός της $\beta$ στο $A$, δηλαδή για κάθε $a, b \in A$ ισχύει $a \rightarrow_\beta b \iff a \rightarrow_\alpha b$
- το $A$ είναι κλειστό ως προς την $\beta$, δηλαδή για κάθε $a \in A$ ισχύει ότι : $a \rightarrow_\beta b \implies b \in A$

Επίσης, λέμε ότι το $\mathcal{B}$ είναι *επέκταση* του $\mathcal{A}$.

**2.4.2 Ορισμός** (Γράφος αναγωγών)**.** Έστω $\mathcal{A} = \langle A, \rightarrow \rangle$ και $a \in A$. Ο *γράφος αναγωγών* του $a$ συμβολίζεται ως $\mathcal{G}(a)$ και είναι το μικρότερο υποσύστημα του $A$ που περιέχει το $a$. Το $\mathcal{G}(a)$ έχει στοιχεία τους απογόνους του $a$ (στους οποίους συμπεριλαμβάνεται και το $a$) και την σχέση $\rightarrow$ περιορισμένη στους απογόνους του $a$, δηλαδή $\mathcal{G}(a) = \langle \Delta^*(a), \rightarrow \cap (\Delta^*(a) \times \Delta^*(a)) \rangle$.

**2.4.3 Παράδειγμα.** Για το σύστημα του σχήματος 2.1, οι γράφοι αναγωγών για κάθε ένα από τα $a$, $b$, $c$, $d$, $e$, $f$ φαίνονται στο σχήμα 2.3.




Σχήμα 2.3 – Γράφοι αναγωγών

## 2.5 Γενικά περί γράφων

Σε αυτήν την ενότητα δίνουμε μία γενική ιδέα για τους γράφους καθώς και κάποιες χρήσιμες ιδιότητές τους, ιδιαίτερα στα πλαίσια των συστημάτων μεταγραφής. Για περισσότερες λεπτομέρειες σχετικά με την Γραφοθεωρία παραπέμπουμε τον ενδιαφερόμενο αναγνώστη σε εισαγωγικό επίπεδο στο [15] και σε πιο υψηλό επίπεδο στα [32, 8].

**2.5.1 Ορισμός.** Ένας *γράφος* αποτελείται από ένα σύνολο $V$ *κορυφών* (ή κόμβων) και από ένα σύνολο $E$ *ακμών*, δηλαδή συνδέσεων μεταξύ των παραπάνω κορυφών.

Ο παραπάνω ορισμός είναι εσκεμμένα ασαφής, αφού η έννοια της ακμής, δηλαδή της σύνδεσης μεταξύ κορυφών δεν είναι σαφώς καθορισμένη, αλλά εξαρτάται από την εφαρμογή στην οποία χρησιμοποιείται ο γράφος. Για παράδειγμα, αν ένας γράφος αντιστοιχεί σε ένα αφηρημένο σύστημα μεταγραφής, έχει έννοια να ορίσουμε τις ακμές ως διατεταγμένα ζεύγη των κορυφών και επομένως το σύνολο $E$ των ακμών ως ένα υποσύνολο του $V \times V$, δηλαδή ως μία σχέση επί του $V$. Τότε, αφού έχει σημασία η κατεύθυνση μίας σύνδεσης, ο γράφος ονομάζεται *κατευθυνόμενος*. Σε άλλες εφαρμογές δεν έχει σημασία η κατεύθυνση της σύνδεσης, οπότε η κάθε ακμή είναι απλώς ένα μη διατεταγμένο ζεύγος κορυφών, δηλαδή ένα σύνολο δύο κορυφών, και ο γράφος ονομάζεται *μη κατευθυνόμενος*. Σε άλλες πάλι περιπτώσεις ενδέχεται να έχουμε περισσότερες από μία ακμές μεταξύ δύο κορυφών (τότε το $E$ δεν είναι σύνολο με την αυστηρή έννοια) οπότε έχουμε έναν *πολυγράφο*· ή, ακόμα, κάθε ακμή να είναι επιγεγραμμένη (labelled) με ένα χαρακτηριστικό, έστω για παράδειγμα ένα βάρος οπότε έχουμε έναν *γράφο με βάρη*.

**2.5.2 Ορισμός.** Ένας γράφος ονομάζεται *άπειρος* αν το σύνολο $V$ είναι άπειρο. Αλλιώς ονομάζεται *πεπερασμένος*.

**2.5.3 Ορισμός** (Μονοπάτια και κύκλοι)**.**

- Ένα *μονοπάτι* σε έναν γράφο είναι μία πεπερασμένη, διατεταγμένη ακολουθία κορυφών τέτοια ώστε για δύο διαδοχικές στην ακολουθία κορυφές υπάρχει σύνδεση από την πρώτη στην δεύτερη κορυφή (ενδεχομένως χρειάζεται να λάβουμε υπ' όψιν την κατεύθυνση των ακμών). Λέμε ότι το μονοπάτι συνδέει την πρώτη με την τελευταία κορυφή της ακολουθίας.
- Ένα μονοπάτι ονομάζεται *απλό* αν όλες οι κορυφές της ακολουθίας, με πιθανή εξαίρεση την πρώτη και την τελευταία, είναι διακεκριμένες.
- Ένα απλό μονοπάτι ονομάζεται *κύκλος* αν η πρώτη και η τελευταία κορυφή ταυτίζονται.

**2.5.4 Ορισμός** (Συνεκτικότητα)**.** Ένας γράφος ονομάζεται *συνεκτικός* αν υπάρχει μονοπάτι από οποιαδήποτε κορυφή του γράφου προς οποιαδήποτε άλλη κορυφή του γράφου.

Για παράδειγμα μπορεί εύκολα να αποδειχθεί ότι για οποιοδήποτε σύστημα μεταγραφής και για οποιοδήποτε στοιχείο του $a$, ο γράφος αναγωγών $\mathcal{G}(a)$ είναι συνεκτικός γράφος, αν δεν λάβουμε υπ' όψιν την κατεύθυνση των ακμών.

**2.5.5 Ορισμός** (Ακυκλικότητα)**.** Ένας γράφος ονομάζεται *ακυκλικός* αν δεν περιέχει κύκλους.

Για παράδειγμα, ο γράφος του σχήματος 2.1 δεν είναι ακυκλικός, ενώ αυτός του σχήματος 2.2 είναι.

# Κεφάλαιο 3

# Συμβολή και τερματισμός

Η συμβολή και ο τερματισμός είναι δύο εξαιρετικά σημαντικές ιδιότητες των συστημάτων μεταγραφής, ιδιαίτερα αν αυτά ιδωθούν ως υπολογιστικά συστήματα.

Όπως είδαμε στο προηγούμενο κεφάλαιο, ένα σύστημα μεταγραφής ενδέχεται να μην είναι αιτιοκρατικό. Η έλλειψη αιτιοκρατίας οδηγεί στην δυνατότητα από ένα στοιχείο να προκύπτουν διάφορα αποτελέσματα, ανάλογα με τις επιλογές που γίνονται σε κάθε βήμα υπολογισμού. Η συμβολή εξασφαλίζει ότι αν δύο διαδικασίες μεταγραφής (οι οποίες βέβαια ξεκινούν από το ίδιο στοιχείο) «αποκλίνουν» και οδηγούνται σε διαφορετικά ενδιάμεσα αποτελέσματα, τότε είναι πάντοτε δυνατόν να κάνουμε αυτά τα ενδιάμεσα αποτελέσματα, εφαρμόζοντας τα κατάλληλα βήματα μεταγραφής σε κάθε ένα από αυτά, να «συγκλίνουν» σε έναν κοινό απόγονο.

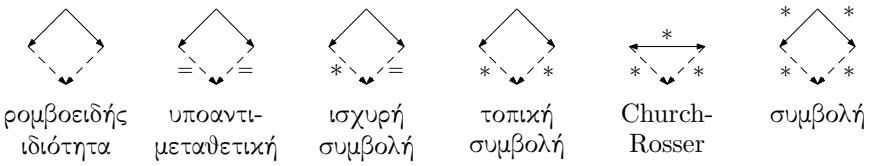
Σε ένα υπολογιστικό σύστημα, έχει εξαιρετικό ενδιαφέρον να γνωρίζουμε αν και υπό ποιες συνθήκες μία διαδικασία υπολογισμού σταματά (δίνοντας κάποιο αποτέλεσμα) ή συνεχίζει επ' άπειρον. Στα συστήματα μεταγραφής η έννοια του τερματισμού, μας επιτρέπει να θέσουμε και να απαντήσουμε αντίστοιχα ερωτήματα.

## 3.1 Συμβολή

Παρακάτω δίνουμε μερικές ιδιότητες που σχετίζονται με την συμβολή. Θεωρούμε πάντοτε αφηρημένο σύστημα αναγωγής $\mathcal{A} = \langle A, \to \rangle$. Η παρουσίαση βασίζεται κυρίως στην αντίστοιχη ενότητα του [48].

**3.1.1 Ορισμός.** Μία σχέση $\to$ είναι *τοπικώς συμβάλλουσα* (locally confluent) αν ισχύει $\leftarrow \circ \to \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$. Με άλλα λόγια, για κάθε $a$, $b$, $c$ με $a \to b$ και $a \to c$, υπάρχει $d$, τέτοιο ώστε $b \overset{*}{\to} d$ και $c \overset{*}{\to} d$ (βλέπε και σχήμα 3.1).

Η ιδιότητα της τοπικής συμβολής αναφέρεται επίσης ως *ασθενής συμβολή* ή και *ασθενής Church-Rosser ιδιότητα* (σύντομα WCR από τον αγγλικό όρο "weakly Church-Rosser").




Σχήμα 3.1 – Ιδιότητες σχετικές με την συμβολή

**3.1.2 Ορισμός.** Μία σχέση $\to$ έχει την *ρομβοειδή ιδιότητα* (συμβολισμός : $\diamondsuit$) εφόσον ισχύει $\leftarrow \circ \to \subseteq \to \circ \leftarrow$. Με άλλα λόγια, για κάθε $a$, $b$, $c$ με $a \to b$ και $a \to c$, υπάρχει $d$, τέτοιο ώστε $b \to d$ και $c \to d$ (βλέπε και σχήμα 3.1).

Η επιλογή του όρου «ρομβοειδής ιδιότητα» προέρχεται από το αγγλικό "diamond property" και από το αντίστοιχο διάγραμμα στο σχήμα 3.1.

**3.1.3 Ορισμός.** Μία σχέση $\to$ είναι *ομοιόμορφα συμβάλλουσα* (uniformly confluent), ή συμβολικά $\mathrm{WCR}^1$, εφόσον ισχύει $\leftarrow \circ \to \subseteq 1_A \cup \to \circ \leftarrow$. Με άλλα λόγια, για κάθε $a$, $b$, $c$ με $a \to b$ και $a \to c$, είναι $b \equiv c$ ή υπάρχει $d$, τέτοιο ώστε $b \to d$ και $c \to d$.

**3.1.4 Ορισμός.** Μία σχέση $\to$ είναι *υποαντιμεταθετική* (subcommutative), ή συμβολικά $\mathrm{WCR}^{\leq 1}$, εφόσον ισχύει $\leftarrow \circ \to \subseteq \overset{=}{\to} \circ \overset{=}{\leftarrow}$. Με άλλα λόγια, για κάθε $a$, $b$, $c$ με $a \to b$ και $a \to c$, υπάρχει $d$, τέτοιο ώστε $b \overset{=}{\to} d$ και $c \overset{=}{\to} d$ (βλέπε και σχήμα 3.1).

*3.1.5 Παρατήρηση.* Αν μία σχέση είναι ανακλαστική, η ρομβοειδής ιδιότητα, η υποαντιμεταθετικότητα και η ομοιόμορφη συμβολή είναι ισοδύναμες.

**3.1.6 Ορισμός.** Μία σχέση $\to$ είναι *ισχυρά συμβάλλουσα* (strongly confluent), ή συμβολικά SCR, εφόσον ισχύει $\leftarrow \circ \to \subseteq \overset{*}{\to} \circ \overset{=}{\leftarrow}$. Με άλλα λόγια, για κάθε $a$, $b$, $c$ με $a \to b$ και $a \to c$, υπάρχει $d$, τέτοιο ώστε $b \overset{*}{\to} d$ και $c \overset{=}{\to} d$ (βλέπε και σχήμα 3.1).

**3.1.7 Ορισμός.** Μία σχέση $\to$ έχει την *ιδιότητα Church-Rosser* (CR) αν $\overset{*}{\leftrightarrow} \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$. Με άλλα λόγια, για κάθε $b$, $c$ με $b \overset{*}{\leftrightarrow} c$, υπάρχει $d$, τέτοιο ώστε $b \overset{*}{\to} d$ και $c \overset{*}{\to} d$ (βλέπε και σχήμα 3.1).

*3.1.8 Παρατήρηση.* Με την ορολογία και τον συμβολισμό του ορισμού 2.2.6, μία σχέση έχει την ιδιότητα Church-Rosser αν για οποιαδήποτε δύο (όχι απαραίτητα διαφορετικά μεταξύ τους) στοιχεία που είναι μετατρέψιμα συνεπάγεται ότι τα δύο αυτά στοιχεία είναι και ζεύξιμα, δηλαδή $\overset{*}{\leftrightarrow} \subseteq \downarrow$.

Αν θεωρήσουμε ότι για μία σχέση, το μεταβατικό ανακλαστικό κλείσιμο αυτής έχει την ρομβοειδή ιδιότητα, φαίνεται να προκύπτει μία ασθενέστερη της Church-Rosser ιδιότητα :

**3.1.9 Ορισμός.** Μία σχέση $\rightarrow$ είναι *συμβάλλουσα* αν $\overset{*}{\leftarrow} \circ \overset{*}{\rightarrow} \subseteq \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow}$. Με άλλα λόγια, για κάθε $a$, $b$, $c$ με $a \overset{*}{\rightarrow} b$ και $a \overset{*}{\rightarrow} c$, υπάρχει $d$, τέτοιο ώστε $b \overset{*}{\rightarrow} d$ και $c \overset{*}{\rightarrow} d$ (βλέπε και σχήμα 3.1).

*3.1.10 Παρατήρηση.* Χρησιμοποιώντας επιπλέον την ορολογία και τον συμβολισμό του ορισμού 2.2.7, μία σχέση είναι συμβάλλουσα αν για οποιαδήποτε δύο στοιχεία (όχι απαραίτητα διαφορετικά μεταξύ τους) που είναι συναντήσιμα συνεπάγεται ότι τα δύο αυτά στοιχεία είναι και ζεύξιμα, δηλαδή $\uparrow \subseteq \downarrow$. Χρησιμοποιώντας την ορολογία των προγόνων και απογόνων, μία σχέση είναι συμβάλλουσα αν δύο στοιχεία (όχι απαραίτητα διαφορετικά μεταξύ τους) τα οποία έχουν κοινό πρόγονο έχουν αναγκαστικά και κοινό απόγονο.

Αν και η ιδιότητα της *συμβολής* φαίνεται ασθενέστερη της Church-Rosser, ισχύει το παρακάτω θεώρημα :

**3.1.11 Θεώρημα.** *Μία σχέση είναι Church-Rosser αν και μόνον αν είναι συμβάλλουσα.*

*Απόδειξη.*

Για κάθε σχέση προφανώς ισχύει $\overset{*}{\rightarrow} \circ \overset{*}{\leftarrow} \subseteq \overset{*}{\leftrightarrow}$, επομένως αν μία σχέση είναι Church-Rosser, τότε είναι και συμβάλλουσα.

Αν μία σχέση είναι συμβάλλουσα, θα δείξουμε ότι είναι Church-Rosser με επαγωγή στο πλήθος των αλλαγών κατεύθυνσης της $\overset{*}{\leftrightarrow}$. Για την απόδειξη, προκύπτουν βολικοί οι συμβολισμοί των σχέσεων ζευξιμότητας ($\downarrow$) και συναντησιμότητας ($\uparrow$) από τους ορισμούς 2.2.6 και 2.2.7, αντιστοίχως.

Προφανώς ισχύει $\uparrow^* = \overset{*}{\leftrightarrow}$ (υπενθυμίζουμε τον γνωστό συμβολισμό από τον ορισμό 1.1.14 : η $\uparrow^*$ είναι το μεταβατικό ανακλαστικό κλείσιμο της $\uparrow$). Αρκεί λοιπόν να δείξουμε ότι για κάθε $n \geq 0$ ισχύει $\uparrow^n \subseteq \downarrow$. Ποιοτικά, το $n$ συμβολίζει το πλήθος των αλλαγών κατεύθυνσης από τα «αριστερά» προς τα «δεξιά» της σχέσης $\overset{*}{\rightarrow}$ κατά την «μετατροπή» ενός στοιχείου σε κάποιο άλλο.

Για $n = 0$, προφανώς ισχύει $\uparrow^0 \subseteq \downarrow$ (όπου $\uparrow^0$ η σχέση ισότητας).

Υποθέτουμε ότι για $n = k$ ισχύει $\uparrow^k \subseteq \downarrow$. Τότε έχουμε :

$$\begin{aligned} \uparrow^{k+1} &= \uparrow \circ \uparrow^k && \text{από τον ορισμό 1.1.11} \\ &\subseteq \downarrow \circ \downarrow && \text{αφού } \uparrow \subseteq \downarrow \text{ (συμβολή), } \uparrow^k \subseteq \downarrow \text{ (από υπόθεση)} \\ &\subseteq \downarrow && \text{βλέπε παρακάτω, λήμμα 3.1.12} \end{aligned}$$

□


Σχήμα 3.2 – Ισοδυναμία ιδιοτήτων Church-Rosser και συμβολής

**3.1.12 Λήμμα.** *Αν $\eta \to$ είναι συμβάλλουσα, τότε $\downarrow \circ \downarrow \subseteq \downarrow$.*

*Απόδειξη.* Έστω $x$, $y$ για τα οποία ισχύει $x \downarrow \circ \downarrow y$. Από τον ορισμό της σύνθεσης, υπάρχει $w$, τέτοιο ώστε $x \downarrow w$ και $w \downarrow y$. Αυτό σημαίνει ότι τα $x$, $w$ έχουν κοινό απόγονο, έστω το $u$, καθώς και τα $w$, $y$ έχουν κοινό απόγονο, έστω το $v$. Τότε όμως, το $w$ είναι κοινός πρόγονος των $u$, $v$, δηλαδή $u \uparrow v$. Τότε όμως, επειδή $\eta \to$ είναι συμβάλλουσα, σύμφωνα και με την παρατήρηση 3.1.10, $u \downarrow v$ και τα $u$, $v$ έχουν κοινό απόγονο, έστω το $z$. Λόγω όμως και της (προφανούς) μεταβατικότητας της $\xrightarrow{*}$, το $z$ είναι και κοινός απόγονος των $x$, $y$, δηλαδή $x \downarrow y$. Όλα αυτά, μαζί με ένα μέρος του θεωρήματος 3.1.11, φαίνονται γραφικά στο σχήμα 3.2 (για να μην επιβαρύνουμε πολύ το σχήμα χρησιμοποιούμε τον συμβολισμό $\twoheadrightarrow$ αντί του $\xrightarrow{*}$). □

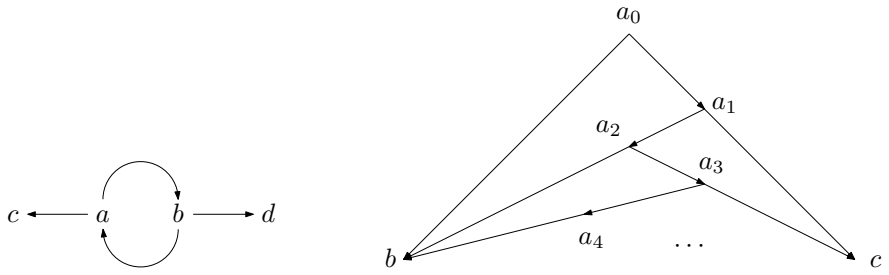
*3.1.13 Παρατήρηση.* Στην πραγματικότητα, αν $\eta \to$ είναι συμβάλλουσα, ισχύει $\downarrow \circ \downarrow = \downarrow$, επειδή $\downarrow \supseteq \xrightarrow{0}$ κι επομένως $\downarrow \circ \downarrow \supseteq \downarrow \circ \xrightarrow{0} = \downarrow$.

Αφού λοιπόν, όπως είδαμε, η συμβολή είναι ισοδύναμη με την ιδιότητα Church-Rosser, οι δύο όροι χρησιμοποιούνται εναλλάξ για να δηλώσουν την στην ουσία κοινή έννοια, ενώ χρησιμοποιείται και στις δύο περιπτώσεις ο συμβολισμός CR. Έτσι, δικαιολογείται και το "CR" στην ασθενή συμβολή (WCR) καθώς και στις $\mathrm{WCR}^1$ και $\mathrm{WCR}^{\leq 1}$.

Ας σημειωθεί ότι η συμβολή συνεπάγεται την τοπική συμβολή, αλλά όχι το αντίστροφο, όπως θα φανεί στα δύο ακόλουθα παραδείγματα.

**3.1.14 Παράδειγμα** (Newman [77])**.** Έστω $A = \{a, b, c, d\}$ και η σχέση $\to$ για την οποία ισχύουν ακριβώς τα εξής : $a \to b$, $a \to c$, $b \to a$, $b \to d$ (βλέπε σχήμα 3.3α). Το σύστημα $\mathcal{A} = \langle A, \to \rangle$ είναι τοπικώς συμβάλλον, αλλά όχι συμβάλλον. Ας σημειωθεί ότι σε αυτό το σύστημα υπάρχει ακολουθία αναγωγών όπου παρουσιάζεται *κύκλος*, δηλαδή κάποιο στοιχείο επαναλαμβάνεται, για παράδειγμα $a \to b \to a$.

**3.1.15 Παράδειγμα** (Hindley [50])**.** Έστω $A = \{b, c, a_0, a_1, \ldots, a_i, \ldots\}$ και η σχέση $\to$ για την οποία ισχύουν ακριβώς τα εξής (βλέπε και σχήμα

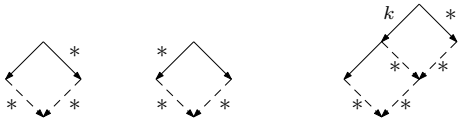



(α) (β)

Σχήμα 3.3 – Τοπικώς συμβάλλοντα, αλλά όχι συμβάλλοντα συστήματα

3.3β) : για κάθε $n \geq 0$, $a_n \to a_{n+1}$, $a_{2n} \to b$, $a_{2n+1} \to c$. Το σύστημα $\mathcal{A} = \langle A, \to \rangle$ είναι τοπικώς συμβάλλον, αλλά όχι συμβάλλον και επιπλέον, σε αντίθεση με το προηγούμενο σύστημα, δεν υπάρχει ακολουθία αναγωγών στην οποία να εμφανίζεται κύκλος.

Δίνουμε μία ακόμη ιδιότητα, την $\leftarrow \circ \overset{*}{\to} \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$, για την οποία αποδεικνύουμε ότι είναι ισοδύναμη με αυτές της συμβολής και της Church-Rosser, αν και φαίνεται ασθενέστερη της συμβολής. Η ιδιότητα φαίνεται στο σχήμα 3.4α, από όπου παρατηρούμε ότι είναι ισοδύναμη με την ιδιότητα που έχει συμμετρικό αριστερό μέλος (βλέπε σχήμα 3.4β).




(á) (β) (γ)

Σχήμα 3.4 – Η ιδιότητα $\leftarrow \circ \overset{*}{\to} \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$

**3.1.16 Θεώρημα.** *Η* $\to$ *είναι* CR *αν και μόνον αν* $\leftarrow \circ \overset{*}{\to} \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$.

*Απόδειξη.*

Προφανώς, αν η $\to$ είναι συμβάλλουσα, δηλαδή ισχύει $\overset{*}{\leftarrow} \circ \overset{*}{\to} \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$, τότε έχει και την άλλη ιδιότητα, επειδή $\leftarrow \circ \overset{*}{\to} \subseteq \overset{*}{\leftarrow} \circ \overset{*}{\to}$.

Έστω $\leftarrow \circ \overset{*}{\to} \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$. Για να δείξουμε ότι η $\to$ είναι συμβάλλουσα αρκεί να δείξουμε επαγωγικά ότι για κάθε $n \geq 0$ ισχύει $\overset{n}{\leftarrow} \circ \overset{*}{\to} \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$. Τότε η συμβολή προκύπτει άμεσα από την επιμεριστικότητα της σύνθεσης ως προς την ένωση (πρόταση 1.1.7).

Για $n = 0$ έχουμε $\overset{*}{\to} = \overset{*}{\to} \circ \overset{0}{\to} \subseteq \overset{*}{\to} \circ \overset{*}{\leftarrow}$, επειδή $\overset{0}{\to} \subseteq \overset{*}{\leftarrow}$.

Υποθέτουμε ότι για $n = k$ ισχύει $\overset{k}{\leftarrow} \circ \overset{*}{\rightarrow} \subseteq \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow}$. Τότε, εκμεταλλευόμενοι και την προσεταιριστικότητα της σύνθεσης (πρόταση 1.1.7), έχουμε (βλέπε και σχήμα 3.4γ) :

$$\begin{aligned}
\overset{k+1}{\leftarrow} \circ \overset{*}{\rightarrow} &= \leftarrow \circ \overset{k}{\leftarrow} \circ \overset{*}{\rightarrow} && \text{από τον ορισμό 1.1.11} \\
&\subseteq \leftarrow \circ \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow} && \text{από υπόθεση} \\
&\subseteq \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow} \circ \overset{*}{\leftarrow} && \text{λόγω ιδιότητας της } \rightarrow \\
&= \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow} && \text{λόγω ομοδυναμίας της } \overset{*}{\rightarrow} \qquad \square
\end{aligned}$$

Ας δούμε τώρα πώς συνδέονται μεταξύ τους οι σχετικές με την συμβολή ιδιότητες που δόθηκαν παραπάνω.

Αρχικά, μελετούμε ιδιότητες ισοδύναμες με την συμβολή. Πέραν από την ισοδυναμία των τριών ιδιοτήτων της συμβολής, της Church-Rosser και της ιδιότητας $\leftarrow \circ \overset{*}{\rightarrow} \subseteq \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow}$, από τον ορισμό της ιδιότητας της συμβολής έχουμε ότι $\mathrm{CR}(\rightarrow) \iff \Diamond(\overset{*}{\rightarrow})$ και επειδή η $\overset{*}{\rightarrow}$ είναι ανακλαστική, προκύπτει (από την παρατήρηση 3.1.5) ότι η ιδιότητα της συμβολής, $\mathrm{CR}(\rightarrow)$, είναι επίσης ισοδύναμη με τις $\mathrm{WCR}^1(\overset{*}{\rightarrow})$ και $\mathrm{WCR}^{\leq 1}(\overset{*}{\rightarrow})$. Τέλος, λόγω της ομοδυναμίας της $\overset{*}{\rightarrow}$, ισχύει $\mathrm{CR}(\rightarrow) \iff \mathrm{WCR}(\overset{*}{\rightarrow})$.

Όπως είδαμε η τοπική (ή ασθενής) συμβολή δεν συνεπάγεται απαραίτητα την συμβολή. Όμως, οι ιδιότητες $\Diamond$, $\mathrm{WCR}^1$, $\mathrm{WCR}^{\leq 1}$ και SCR μπορούν να θεωρηθούν «ενισχυμένες» εκδόσεις της τοπικής συμβολής, οι οποίες συνεπάγονται την ιδιότητα CR σύμφωνα με την παρακάτω πρόταση :

**3.1.17 Πρόταση.** *Έστω σύστημα αναγωγής $\mathcal{A} = \langle A, \rightarrow \rangle$. Τότε ισχύει :*

$$\Diamond \implies \mathrm{WCR}^1 \implies \mathrm{WCR}^{\leq 1} \implies \mathrm{SCR} \implies \mathrm{CR} \implies \mathrm{WCR}$$

*Απόδειξη.* Όλες οι συνεπαγωγές είναι απλές πλην, ίσως, της $\mathrm{SCR} \implies \mathrm{CR}$, για την οποία θα δώσουμε μία απόδειξη.

Αρχικά, πρόκειται να δείξουμε (επαγωγικά) ότι, εφόσον η $\rightarrow$ είναι SCR, για κάθε $n \geq 0$ ισχύει η ιδιότητα $\leftarrow \circ \overset{n}{\rightarrow} \subseteq \overset{*}{\rightarrow} \circ \overset{=}{\leftarrow}$.

Για $n = 0$ : $\leftarrow \circ \overset{0}{\rightarrow} = \leftarrow \subseteq \overset{*}{\rightarrow} \circ \overset{=}{\leftarrow}$.

Υποθέτουμε ότι για $n = k$ η ιδιότητα ισχύει. Τότε έχουμε (δεν χρησιμοποιούμε παρενθέσεις μεταξύ διαδοχικών συνθέσεων, επειδή ισχύει η ιδιότητα της προσεταιριστικότητας, πρόταση 1.1.7) :

$$\begin{aligned}
\leftarrow \circ \overset{k+1}{\rightarrow} &= \leftarrow \circ \overset{k}{\rightarrow} \circ \rightarrow && \text{από τον ορισμό 1.1.11} \\
&\subseteq \overset{*}{\rightarrow} \circ \overset{=}{\leftarrow} \circ \rightarrow && \text{από επαγωγική υπόθεση} \\
&= \overset{*}{\rightarrow} \circ (\leftarrow \cup \overset{0}{\rightarrow}) \circ \rightarrow && \text{επειδή } \overset{=}{\leftarrow} = \leftarrow \cup \overset{0}{\rightarrow}
\end{aligned}$$

$$
\begin{aligned}
&= \xrightarrow{*} \circ \big((\leftarrow \circ \rightarrow) \cup \rightarrow\big) && \text{επιμεριστική ιδιότητα}\\
&= (\xrightarrow{*} \circ \leftarrow \circ \rightarrow) \cup (\xrightarrow{*} \circ \rightarrow) && \text{επιμεριστική ιδιότητα}\\
&= (\xrightarrow{*} \circ \leftarrow \circ \rightarrow) \cup \xrightarrow{+} && \text{επειδή } \xrightarrow{+} = \xrightarrow{*} \circ \rightarrow\\
&\subseteq (\xrightarrow{*} \circ \xrightarrow{*} \circ \xleftarrow{=}) \cup \xrightarrow{+} && \text{λόγω ιδιότητας της } \rightarrow\\
&= (\xrightarrow{*} \circ \xleftarrow{=}) \cup \xrightarrow{+} && \text{λόγω ομοδυναμίας της } \xrightarrow{*}\\
&= \xrightarrow{*} \circ \xleftarrow{=} && \text{επειδή } \xrightarrow{+} \subseteq \xrightarrow{*} \circ \xleftarrow{=}
\end{aligned}
$$

Τότε θα είναι $\leftarrow \circ \xrightarrow{*} \subseteq \xrightarrow{*} \circ \xleftarrow{=} \subseteq \xrightarrow{*} \circ \xleftarrow{*}$, οπότε από το θεώρημα 3.1.16 η $\rightarrow$ είναι CR. □

## 3.2 Κανονικές μορφές και τερματισμός

Στους παρακάτω ορισμούς θεωρούμε σύστημα αναγωγής $\mathcal{A} = \langle A, \rightarrow \rangle$.

**3.2.1 Ορισμός** (Αλυσίδα)**.** Μία ακολουθία, πεπερασμένη ή άπειρη, που αποτελείται από στοιχεία του $A$, τέτοια ώστε για κάθε δύο διαδοχικά στοιχεία $a_i$, $a_{i+1}$ της ακολουθίας να ισχύει $a_i \rightarrow a_{i+1}$, ονομάζεται *αλυσίδα* (πεπερασμένη ή άπειρη αντιστοίχως).

**3.2.2 Ορισμός** (Κανονική μορφή)**.** Έστω $b \in A$. Αν δεν υπάρχει $x \in A$ τέτοιο ώστε $b \rightarrow x$, δηλαδή το $b$ δεν έχει (άμεσο) διάδοχο, το $b$ ονομάζεται *κανονική μορφή*· εναλλακτικά, λέμε ότι το $b$ είναι *ανάγωγο* (irreducible). Αν $a \in A$ και $a \xrightarrow{*} b$, για κάποια κανονική μορφή $b$, λέμε ότι το $a$ *έχει κανονική μορφή*.

**3.2.3 Παράδειγμα.** Έστω το σύστημα :

$$a \rightarrow b, \qquad a \rightarrow c, \qquad c \rightarrow c$$

Η $c \rightarrow c \rightarrow \cdots \rightarrow c \rightarrow \cdots$ είναι μία άπειρη αλυσίδα, ενώ η $a \rightarrow b$ και η $a \rightarrow c \rightarrow c$ είναι πεπερασμένες. Το $b$ είναι κανονική μορφή, ενώ κανένα από τα $a$, $c$ δεν είναι. Το $a$ έχει κανονική μορφή (προφανώς και το $b$, ως ήδη ον σε κανονική μορφή), αλλά το $c$ δεν έχει.

**3.2.4 Ορισμός.** Ορίζουμε μία νέα σχέση, την $\xrightarrow{!}$, ως εξής : $a \xrightarrow{!} b$ αν και μόνον αν $a \xrightarrow{*} b$ και το $b$ είναι κανονική μορφή.

Αν δούμε ένα σύστημα αναγωγής ως υπολογιστικό σύστημα, οι κανονικές μορφές αντιστοιχούν στα *τελικά αποτελέσματα*. Σε ένα υπολογιστικό σύστημα, δίνουμε κάποια είσοδο $a$ και αναμένουμε μία έξοδο $b$, ως αποτέλεσμα, τέτοια ώστε $a \xrightarrow{!} b$.

**3.2.5 Ορισμός** (Ασθενής κανονικοποίηση)**.** Αν κάθε $a \in A$ έχει κανονική μορφή, η σχέση $\rightarrow$ ονομάζεται *ασθενώς κανονικοποιούσα* (ή σύντομα WN από τον αγγλικό όρο "weakly normalizing").

*3.2.6 Παρατήρηση.* Η ιδιότητα της ασθενούς κανονικοποίησης δεν απαγορεύει την ύπαρξη απείρων αλυσίδων, αφού ενδέχεται ένα στοιχείο να έχει άπειρη αλυσίδα αναγωγών, αλλά ταυτόχρονα και μία άλλη αλυσίδα (πεπερασμένη) που να καταλήγει σε κανονική μορφή.

**3.2.7 Ορισμός** (Ισχυρή κανονικοποίηση ή τερματισμός)**.** Αν δεν υπάρχουν άπειρες αλυσίδες στο $\mathcal{A} = \langle A, \rightarrow \rangle$ τότε η $\rightarrow$ ονομάζεται *ισχυρώς κανονικοποιούσα* (ή σύντομα SN από τον αγγλικό όρο "strongly normalizing").

Εναλλακτικά, χρησιμοποιούνται οι εξής όροι : το σύστημα $\mathcal{A}$ (ή η σχέση $\rightarrow$) τερματίζει ή είναι Noetherian [1].

*3.2.8 Παρατήρηση.* Είναι προφανές ότι SN $\Rightarrow$ WN, αλλά όχι το αντίστροφο (βλέπε παρατήρηση 3.2.6).

Ένα εύκολο αλλά χρήσιμο αποτέλεσμα είναι το εξής :

**3.2.9 Πρόταση.** *Η σχέση $\rightarrow$ τερματίζει αν και μόνον αν το μεταβατικό κλείσιμο αυτής, δηλαδή η σχέση $\overset{+}{\rightarrow}$, τερματίζει.*

*3.2.10 Παρατήρηση.* Προφανώς, για οποιαδήποτε σχέση $\rightarrow$, ακόμα κι αν η $\rightarrow$ τερματίζει, το μεταβατικό ανακλαστικό κλείσιμο αυτής, δηλαδή η σχέση $\overset{*}{\rightarrow}$, δεν τερματίζει ποτέ.

## 3.3 Καλώς εδραιωμένες σχέσεις

Σε αυτήν την ενότητα θα αναφερθούμε στην έννοια της καλώς εδραιωμένης (well founded) σχέσης και πώς ισοδυναμεί αυτή με την έννοια του τερματισμού. Όμως, πριν από αυτό θα πρέπει να δώσουμε τον ορισμό του ελαχιστικού σημείου.

**3.3.1 Ορισμός.** Έστω σχέση $\rightarrow$ στο σύνολο $A$. Ένα στοιχείο $m$ είναι *ελαχιστικό* σε ένα $S \subseteq A$ ως προς την σχέση $\rightarrow$, αν $m \in S$ και δεν υπάρχει $x \in S$ τέτοιο ώστε $m \rightarrow x$. Πολλές φορές παραλείπουμε το σύνολο στο οποίο και την την σχέση ως προς την οποία το στοιχείο είναι ελαχιστικό, αν αυτά εννοούνται από τα συμφραζόμενα.

*3.3.2 Παρατήρηση.* Προφανώς, οι κανονικές μορφές στο σύστημα $\mathcal{A} = \langle A, \rightarrow \rangle$ είναι ελαχιστικά στοιχεία του συνόλου $A$.

1. Προς τιμήν του μαθηματικού Alvin Noether.

**3.3.3 Ορισμός** (Καλώς εδραιωμένη σχέση)**.** Μία σχέση είναι *καλώς εδραιωμένη* αν κάθε μη κενό υποσύνολο $S$ του $A$ έχει ελαχιστικό σημείο (στο $S$).

**3.3.4 Πρόταση.** *Μία σχέση τερματίζει αν και μόνον αν είναι καλώς εδραιωμένη.*

*Απόδειξη.*

Έστω ότι μία σχέση δεν τερματίζει. Αυτό σημαίνει ότι υπάρχει άπειρη αλυσίδα. Αν συγκεντρώσουμε όλα τα (άπειρα το πλήθος) στοιχεία της αλυσίδας σε ένα σύνολο $S$, τότε αυτό το σύνολο προφανώς είναι μη κενό και δεν έχει ελαχιστικό στοιχείο, αφού για κάθε στοιχείο της αλυσίδας $a \in S$ υπάρχει $b \in S$ τέτοιο ώστε $a \to b$. Επομένως, η σχέση δεν είναι καλώς εδραιωμένη.

Έστω, τώρα, ότι μία σχέση δεν είναι καλώς εδραιωμένη. Αυτό σημαίνει ότι υπάρχει μη κενό υποσύνολο $S$ του $A$, το οποίο δεν έχει ελαχιστικό στοιχείο. Διαλέγουμε ένα οποιοδήποτε στοιχείο του $S$, έστω το $x$. Αφού το $S$ δεν έχει ελαχιστικό στοιχείο υπάρχει $y$ (όχι απαραιτήτως διαφορετικό από το $x$), τέτοιο ώστε $x \to y$. Ομοίως, υπάρχει $z$, τέτοιο ώστε $y \to z$. Αυτή η κατασκευή μπορεί να συνεχιστεί επ' άπειρον, οπότε κατασκευάζεται μία άπειρη αλυσίδα. Επομένως, η σχέση δεν τερματίζει. □

Η παραπάνω πρόταση είναι ισοδύναμη με το *αξίωμα των εξαρτημένων επιλογών* από την αξιωματική συνολοθεωρία :

**Αξίωμα των εξαρτημένων επιλογών.** *Για κάθε σύνολο $A$ και για κάθε σχέση $R$ στο $A$, αν $a \in A$ και για κάθε $x \in A$ υπάρχει $y \in A$, τέτοιο ώστε $x\ R\ y$, τότε υπάρχει συνάρτηση $f\colon \mathbb{N} \to A$, τέτοια ώστε $f(0) = a$ και για κάθε $n \in \mathbb{N}$, να ισχύει $f(n)\ R\ f(n+1)$.*

Για μία απόδειξη αυτής της ισοδυναμίας, βλέπε [71, πρόταση 8.17]. Μάλιστα, κατά την απόδειξη της πρότασης 3.3.4, παρεισέφρησε η χρήση του παραπάνω αξιώματος, όταν κατασκευάσαμε την άπειρη αλυσίδα (βλέπε για παράδειγμα [57]). Το αξίωμα των εξαρτημένων επιλογών είναι μία ειδική περίπτωση του *αξιώματος της επιλογής* (βλέπε [71, κεφάλαιο 8]) :

**Αξίωμα της επιλογής.** *Για οποιαδήποτε δύο σύνολα $A$, $B$ και για κάθε διμελή σχέση $R \subseteq A \times B$, αν για κάθε $x \in A$ υπάρχει $y \in B$ τέτοιο ώστε $x\ R\ y$, τότε υπάρχει συνάρτηση $f\colon A \to B$, τέτοια ώστε για κάθε $x \in A$ να ισχύει $x\ R\ f(x)$.*

Για να αποδείξουμε ιδιότητες μίας καλώς εδραιωμένης σχέσης μπορούμε να χρησιμοποιήσουμε την *καλώς εδραιωμένη επαγωγή* :

**3.3.5 Πρόταση** (Καλώς εδραιωμένη επαγωγή)**.** *Έστω καλώς εδραιωμένη σχέση $\to$ στο σύνολο $A$ και κατηγόρημα $P(x)$, όπου $x \in A$.*

$$\begin{aligned} &\textit{Αν} \quad \forall x \in A\colon \Big(\big(\forall y \in A\colon x \to y \implies P(y)\big) \implies P(x)\Big), && \textit{(υπόθεση)} \\ &\textit{τότε} \quad \forall x \in A\colon P(x). && \textit{(συμπέρασμα)} \end{aligned}$$

*Απόδειξη.* Έστω ότι υπάρχει $x_0 \in A$ για το οποίο δεν ισχύει $P(x_0)$. Τότε υπάρχει αναγκαστικά $x_1 \in A$ τέτοιο ώστε $x_0 \to x_1$ και να μην ισχύει $P(x_1)$, γιατί αν δεν υπήρχε θα έπρεπε σύμφωνα με την υπόθεση να ισχύει $P(x_0)$. Ομοίως, υπάρχει $x_2 \in A$ με $x_1 \to x_2$ και δεν ισχύει $P(x_2)$ και αυτή η κατασκευή μπορεί να συνεχιστεί επ' άπειρον δίνοντας μία άπειρη αλυσίδα $x_0 \to x_1 \to \ldots$· άτοπο, αφού η σχέση $\to$ είναι καλώς εδραιωμένη, δηλαδή τερματίζει. □

Παρατηρήστε ότι αν η υπόθεση της πρότασης 3.3.5 είναι αληθής, προκύπτει ότι για κάθε ελαχιστικό σημείο $x_{\min}$ στο $A$ ως προς την σχέση $\to$, το κατηγόρημα $P$ είναι αληθές : Πράγματι, αφού δεν υπάρχει $y$ τέτοιο ώστε $x_{\min} \to y$, η $\forall y \in A\colon x_{\min} \to y \implies P(y)$ είναι αληθής κι επομένως ισχύει $P(x_{\min})$.

Πρακτικά, η καλώς εδραιωμένη επαγωγή, χρησιμοποιείται ως εξής : Για να αποδείξουμε ότι μία ιδιότητα $P$ ισχύει για όλα τα στοιχεία ενός συνόλου $A$, βρίσκουμε μία καλώς εδραιωμένη σχέση στο $A$ και κατόπιν αποδεικνύουμε πρώτα ότι η ιδιότητα $P$ ισχύει για τις κανονικές μορφές (βάση) και μετά ότι ισχύει για κάθε άλλο τυχόν στοιχείο του $A$, υποθέτοντας ότι ισχύει για όλα τα στοιχεία που είναι άμεσοι απόγονοι του τυχόντος στοιχείου (επαγωγικό βήμα).

Η *επαγωγή στους φυσικούς αριθμούς* είναι μία ειδική περίπτωση της καλώς εδραιωμένης επαγωγής : Αρκεί να θεωρήσουμε $A = \mathbb{N}$ και $x \to y$ αν και μόνον αν $x = y + 1$.

## 3.4 Μοναδικότητα κανονικών μορφών

Η μοναδικότητα των κανονικών μορφών επιβάλλει κάθε στοιχείο ενός συνόλου ενός συστήματος αναγωγής να μην έχει παραπάνω από μία κανονική μορφή.

Παρακάτω δίνουμε για ένα σύστημα $\mathcal{A} = \langle A, \to \rangle$, τρεις ιδιότητες που αφορούν την μοναδικότητα των κανονικών μορφών.

**3.4.1 Ορισμός.** Ένα σύστημα έχει την *ιδιότητα της κανονικής μορφής* (normal form property, συμβολικά NF) αν κάθε στοιχείο του $A$ που μετατρέπεται σε μία κανονική μορφή, ανάγεται σε αυτήν την κανονική μορφή,

δηλαδή αν για κάθε $a \in A$ και για κάθε κανονική μορφή $b$, τέτοια ώστε $a \overset{*}{\leftrightarrow} b$, έχουμε επίσης $a \xrightarrow{*} b$.

**3.4.2 Ορισμός.** Ένα σύστημα έχει την *ιδιότητα της μοναδικής κανονικής μορφής* (unique normal form property, συμβολικά UN) αν κανονικές μορφές διαφορετικές μεταξύ τους δεν είναι μετατρέψιμες, δηλαδή αν για τις κανονικές μορφές $b$, $c$ ισχύει $b \overset{*}{\leftrightarrow} c$, τότε αναγκαστικά $b \equiv c$.

**3.4.3 Ορισμός.** Ένα σύστημα έχει την *ιδιότητα της μοναδικής κανονικής μορφής ως προς την αναγωγή* (unique normal form property with respect to reduction, συμβολικά $\text{UN}^{\rightarrow}$) αν κανένα στοιχείο του $A$ δεν ανάγεται σε κανονικές μορφές διαφορετικές μεταξύ τους, δηλαδή αν για κάθε $a \in A$ για το οποίο υπάρχουν κανονικές μορφές $b$, $c$, τέτοιες ώστε $a \xrightarrow{*} b$ και $a \xrightarrow{*} c$, τότε αναγκαστικά $b \equiv c$.

Παρακάτω δίνουμε μία πρόταση που συνδέει τις παραπάνω ιδιότητες, αλλά κυρίως δίνει και την σχέση τους με την ιδιότητα της συμβολής.

**3.4.4 Πρόταση.** $\text{CR} \implies \text{NF} \implies \text{UN} \implies \text{UN}^{\rightarrow}$

Η απόδειξη είναι πολύ απλή και για αυτό παραλείπεται. Η παραπάνω πρόταση δικαιολογεί εν μέρει την εισαγωγή της έννοιας της συμβολής, αφού αυτή συνεπάγεται μοναδικότητα των κανονικών μορφών υπό οποιαδήποτε από τις τρεις ιδιότητες NF, UN και $\text{UN}^{\rightarrow}$.

Ας σημειωθεί πως δεν μπορούμε να αντιστρέψουμε καμμία από τις συνεπαγωγές της πρότασης 3.4.4. Παρακάτω, δίνουμε τρία αντιπαραδείγματα που αποδεικνύουν τον παραπάνω ισχυρισμό.

**3.4.5 Παράδειγμα.** Έχουμε :

$\text{CR} \not\Leftarrow \text{NF}$. Αρκεί να βρούμε ένα μη συμβάλλον σύστημα που να μην έχει κανονικές μορφές, για παράδειγμα το : $\circlearrowleft a \leftarrow b \rightarrow c \circlearrowright$.

$\text{NF} \not\Leftarrow \text{UN}$. Το σύστημα $\circlearrowleft a \leftarrow b \rightarrow c$ έχει μία μόνον κανονική μορφή, την $c$, επομένως είναι UN, αλλά ενώ το $a$ μετατρέπεται στο $c$, δεν ανάγεται σε αυτό, άρα δεν είναι NF.

$\text{UN} \not\Leftarrow \text{UN}^{\rightarrow}$. Το σύστημα $a \leftarrow b \rightarrow \overset{\curvearrowleft}{c} \leftarrow d \rightarrow e$ εύκολα δεικνύεται ότι είναι $\text{UN}^{\rightarrow}$, αλλά οι διαφορετικές μεταξύ τους κανονικές μορφές $a$, $e$ είναι μετατρέψιμες, άρα δεν είναι UN.

Η μοναδικότητα των κανονικών μορφών, ακόμη και υπό την ασθενέστερη έκφανσή της, την $\text{UN}^{\rightarrow}$, επιβάλλει η σχέση $\xrightarrow{!}$ να είναι μία *μερική συνάρτηση*. Για τον τυπικό ορισμό της μερικής συνάρτησης, χρειάζεται η ήδη ορισθείσα έννοια της συνάρτησης (βλέπε ορισμό 1.1.4). Η συνάρτηση του ορισμού 1.1.4 ονομάζεται για αντιπαράθεση και ως *ολική συνάρτηση*.

**3.4.6 Ορισμός.** Μία *μερική συνάρτηση* $f$ από το σύνολο $A$ στο σύνολο $B$ (συμβολίζεται $f\colon A \rightharpoonup B$) είναι μία σχέση $R \subseteq A \times B$, τέτοια ώστε υπάρχει υποσύνολο $A' \subseteq A$ και (ολική) συνάρτηση $f\colon A' \to B$. Το υποσύνολο $A'$ στο οποίο είναι ορισμένη η συνάρτηση το συμβολίζουμε με $\mathrm{dom}(f)$.

Έχουμε λοιπόν, στην περίπτωση της μοναδικότητας των κανονικών μορφών, ότι $\xrightarrow{!}\colon A \rightharpoonup A$, δηλαδή κάθε στοιχείο του $A$ είτε δεν έχει κανονική μορφή, είτε έχει ακριβώς μία κανονική μορφή.

## 3.5 Συγκλίνοντα συστήματα

Όπως είδαμε, οι ιδιότητες της συμβολής και του τερματισμού είναι ιδιαίτερα χρήσιμες σε ένα σύστημα μεταγραφής. Αν μάλιστα συνδυάζονται και οι δύο σε ένα σύστημα, τότε πολλά δύσκολα έως μη επιλύσιμα προβλήματα στην γενική περίπτωση των συστημάτων μεταγραφής, λύνονται σχετικά απλά, όπως, για παράδειγμα, το πρόβλημα της μετατρεψιμότητας δύο στοιχείων, το οποίο ανάγεται στον υπολογισμό των κανονικών μορφών των δύο στοιχείων. Πράγματι, μπορούμε για κάθε στοιχείο να υπολογίσουμε μία κανονική μορφή του, ανεξάρτητα από την επιλογή της μεταγραφής σε κάθε βήμα, επειδή το σύστημα τερματίζει (δεν υπάρχει άπειρη αλυσίδα) και η συμβολή συνεπάγεται την μοναδικότητα της κανονικής μορφής.

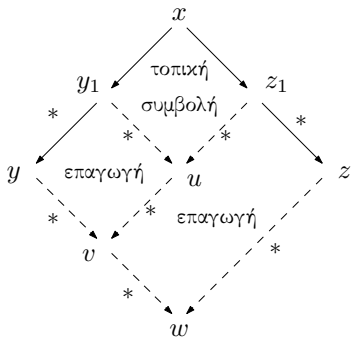
**3.5.1 Ορισμός.** Ένα σύστημα αναγωγής ονομάζεται *συγκλίνον* όταν είναι τερματίζον (SN) και συμβάλλον (CR).

Ένα συγκλίνον σύστημα ονομάζεται εναλλακτικά *πλήρες* ή και *κανονικό*.

Στον παραπάνω ορισμό μπορούμε να χαλαρώσουμε την απαίτηση για συμβολή με την ασθενέστερη της τοπικής συμβολής, όπως επιτρέπει το παρακάτω σημαντικό αποτέλεσμα :

**3.5.2 Θεώρημα.** *Σε ένα σύστημα που τερματίζει, η τοπική συμβολή είναι ισοδύναμη με την συμβολή, δηλαδή* $\mathrm{SN} \implies (\mathrm{WCR} \iff \mathrm{CR})$ *ή αλλιώς* $(\mathrm{SN} \wedge \mathrm{WCR}) \implies \mathrm{CR}$.

Το παραπάνω θεώρημα απεδείχθη για πρώτη φορά από τον Newman στο [77] και για αυτό αναφέρεται και ως λήμμα του Newman. Η απόδειξη όμως στο [77] είναι ιδιαίτερα περίπλοκη. Παρακάτω δίνουμε μία κομψή απόδειξη από τον Huet [52], βασισμένη στην έννοια της καλώς εδραιωμένης επαγωγής (βλέπε πρόταση 3.3.5). Η ιδέα της απόδειξης φαίνεται στο σχήμα 3.5 και έτσι το θεώρημα 3.5.2 αναφέρεται επίσης και ως το ρομβοειδές λήμμα (diamond lemma).




Σχήμα 3.5 – Περί ρομβοειδούς λήμματος

*Απόδειξη του ρομβοειδούς λήμματος (θεώρημα 3.5.2).* Θεωρούμε το κατηγόρημα $P(x)$ που ορίζεται ως εξής : $\forall y, z\colon y \leftarrow x \rightarrow z \implies y \downarrow z$, δηλαδή αν θεωρήσουμε δύο οποιουσδήποτε απογόνους του $x$ αυτοί είναι ζεύξιμοι. Η ιδιότητα της συμβολής είναι ισοδύναμη με το $\forall x\colon P(x)$.

Έστω $y \overset{m}{\leftarrow} x \overset{n}{\rightarrow} z$. Αν ένα τουλάχιστον από τα $m, n$ είναι μηδέν, προκύπτει αμέσως ότι $y \downarrow z$ (εδώ περιέχεται και η περίπτωση των κανονικών μορφών). Αλλιώς ισχύει $y \overset{*}{\leftarrow} y_1 \leftarrow x \rightarrow z_1 \overset{*}{\rightarrow} z$ για κάποια $y_1$, $z_1$. Για να δείξουμε ότι ισχύει $P(x)$ για το τυχόν $x$, θα χρησιμοποιήσουμε την καλώς εδραιωμένη επαγωγή (πρόταση 3.3.5) : Θα δείξουμε ότι αν για κάθε $x'$ με $x \rightarrow x'$ υποθέσουμε ότι ισχύει $P(x')$, τότε αναγκαστικά ισχύει $P(x)$.

Από την ιδιότητα της τοπικής συμβολής, αφού $y_1 \leftarrow x \rightarrow z_1$, υπάρχει κοινός απόγονος των $y_1$ και $z_1$, έστω $u$.

Αν υποθέσουμε ότι ισχύει $P(y_1)$ (επαγωγική υπόθεση) οι δύο απόγονοι $y$, $u$ του $y_1$ είναι ζεύξιμοι, έστω στο $v$, δηλαδή $y \overset{*}{\rightarrow} v \overset{*}{\leftarrow} u$.

Τώρα, αν υποθέσουμε ότι ισχύει $P(z_1)$ (επαγωγική υπόθεση) οι δύο απόγονοι $v$, $z$ του $z_1$ είναι ζεύξιμοι, έστω στο $w$, δηλαδή $v \overset{*}{\rightarrow} w \overset{*}{\leftarrow} z$. Όμως, αφού το $v$ είναι απόγονος του $y$, τελικά $y \downarrow z$. □

# Κεφάλαιο 4

# Όροι

Έως τώρα αναφερθήκαμε, σε αφηρημένα συστήματα μεταγραφής, στα οποία η εσωτερική δομή του συνόλου επί του οποίου λαμβάνουν χώρα οι μεταγραφές δεν λαμβάνεται υπ' όψιν. Σε ορισμένα σύνολα όμως, αν λάβουμε υπ' όψιν την εσωτερική δομή είναι τότε δυνατόν να μελετήσουμε με ευκολότερο τρόπο (και πολλές φορές υπό την έννοια της αποκρισιμότητας) ιδιότητες του αντίστοιχου συστήματος μεταγραφής. Μία τέτοια οικογένεια συνόλων, είναι τα σύνολα *όρων* και τα αντίστοιχα συστήματα μεταγραφής όρων.

Προτού όμως μελετήσουμε τις ιδιότητες των συστημάτων μεταγραφής όρων, πρέπει να ορίσουμε τυπικά μία άλγεβρα επί των όρων. Το κεφάλαιο αυτό βασίζεται κυρίως στα αντίστοιχα κεφάλαια των [48, 104] καθώς και στα [59, 30].

## 4.1 Αλφάβητο και όροι

**4.1.1 Ορισμός** (Αλφάβητο)**.** Ένα *αλφάβητο* (ranked alphabet, signature, vocabulary) είναι ένα αριθμήσιμο σύνολο $\mathcal{F}$ συναρτησιακών συμβόλων, τέτοιο ώστε σε κάθε σύμβολο $f \in \mathcal{F}$ αντιστοιχίζεται ένας φυσικός αριθμός που υποδεικνύει το *πλήθος των ορισμάτων* του συναρτησιακού συμβόλου $f$. Ισχύει $\mathcal{F} = \bigcup_{n\geq 0} \mathcal{F}_n$, όπου $\mathcal{F}_n$ : το σύνολο συναρτησιακών συμβόλων με πλήθος ορισμάτων ίσο με $n$. Τα συναρτησιακά σύμβολα με πλήθος ορισμάτων ίσο με 0 έχουν νόημα και ονομάζονται *σταθερές*.

Τυπικά, η παραπάνω αντιστοίχιση γίνεται μέσω μίας συνάρτησης $a\colon \mathcal{F} \to \mathbb{N}$, δηλαδή το πλήθος των ορισμάτων του συμβόλου $f$ είναι ίσο με $a(f)$ (από το αγγλικό arity). Βέβαια, ο παραπάνω ορισμός δεν προβλέπει την ύπαρξη συναρτησιακών συμβόλων με μεταβλητό πλήθος ορισμάτων, αλλά η επέκταση της θεωρίας σε μία τέτοια περίπτωση δεν είναι ιδιαίτερα δύσκολη.

*4.1.2 Παρατήρηση.* Πολλές φορές, για λόγους συντομίας, χρησιμοποιούμε παρενθέσεις και κόμματα όταν δίνουμε ένα συναρτησιακό σύμβολο, για να δηλώσουμε το πλήθος ορισμάτων αυτού. Έτσι, για παράδειγμα, το αλφάβητο $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2$ με $\mathcal{F}_0 = \{a\}$, $\mathcal{F}_1 = \{g\}$, $\mathcal{F}_2 = \{f\}$, το δηλώνουμε ως εξής : $\mathcal{F} = \{f(,), g(), a\}$.

**4.1.3 Ορισμός** (Όροι)**.** Έστω ότι δίνεται ένα αλφάβητο $\mathcal{F}$ και ένα αριθμήσιμο άπειρο σύνολο $\mathcal{V}$ μεταβλητών με $\mathcal{F} \cap \mathcal{V} = \emptyset$. Το *σύνολο των όρων* επί των $\mathcal{F}$ και $\mathcal{V}$ συμβολίζεται με $\mathcal{T}(\mathcal{F}, \mathcal{V})$ και είναι το μικρότερο σύνολο για το οποίο ισχύουν :

- $\mathcal{T}(\mathcal{F}, \mathcal{V}) \supseteq \mathcal{V}$, δηλαδή περιέχει όλες τις μεταβλητές·
- $\mathcal{T}(\mathcal{F}, \mathcal{V}) \supseteq \mathcal{F}_0$, δηλαδή περιέχει όλες τις σταθερές·
- για κάθε σύμβολο $f \in \mathcal{F}_n$, με $n \geq 1$ και για κάθε έναν από τους όρους $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, ισχύει και $f(t_1, \ldots, t_i) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$· οι όροι $t_1, \ldots, t_n$ είναι τα ορίσματα της συνάρτησης $f$.

Συνήθως οι μεταβλητές συμβολίζονται με τα γράμματα $x$, $y$ και οι όροι με τα γράμματα $t$, $s$.

**4.1.4 Ορισμός** (Θεμελιώδεις όροι)**.** Όροι στους οποίους δεν εμφανίζονται μεταβλητές ονομάζονται *θεμελιώδεις όροι* (ground terms). Το σύνολο των θεμελιωδών όρων συμβολίζεται με $\mathcal{T}(\mathcal{F})$.

Συνήθως, για να αποφύγουμε την μελέτη ιδιαζουσών περιπτώσεων, υποθέτουμε ότι το σύνολο $\mathcal{F}$ περιέχει τουλάχιστον μία σταθερά. Αυτό έχει ως συνέπεια, το σύνολο $\mathcal{T}(\mathcal{F})$ να μην είναι κενό.

**4.1.5 Ορισμός** (Γραμμικοί όροι)**.** Όροι στους οποίους η κάθε μεταβλητή δεν εμφανίζεται πάνω από μία φορά ονομάζονται *γραμμικοί όροι* (linear terms).

**4.1.6 Ορισμός** (Ριζικό σύμβολο)**.** Το *ριζικό σύμβολο* ενός όρου $t$ συμβολίζεται με $\mathrm{root}(t)$ και ορίζεται από $\mathrm{root}(t) = t$ αν $t \in V$ και $\mathrm{root}(t) = f$ αν $t = f(t_1, \ldots, t_n)$.

**4.1.7 Παράδειγμα.** Έστω αλφάβητο $\mathcal{F} = \{f(,,), g(,), h(), a, b, c\}$ και σύνολο μεταβλητών $\mathcal{V} = \{x, y, z, u, x_1, x_2, \ldots\}$.

Ένας θεμελιώδης όρος είναι ο $f(g(a, b), h(a), c)$. Οποιοσδήποτε θεμελιώδης όρος είναι γραμμικός (τετριμμένα) όπως και ο $f(x, h(y), a)$ είναι γραμμικός, ενώ ο $g(x, h(x))$ δεν είναι.

Το ριζικό σύμβολο του $f(g(a, b), h(a), c)$ είναι το $f$.

## 4.2 Οι όροι ως δένδρα· θέσεις και υποόροι

Η έννοια του δένδρου είναι χρήσιμη αφού μας επιτρέπει να παραστήσουμε με γραφικό τρόπο οποιονδήποτε όρο. Από την Θεωρία Γράφων έχουμε τον ακόλουθο ορισμό :

**4.2.1 Ορισμός.** *Δένδρο* ονομάζεται οποιοσδήποτε (μη κατευθυνόμενος) γράφος που είναι συνεκτικός και ακυκλικός[1].

Πέραν των άλλων, ένα δένδρο έχει την ιδιότητα μεταξύ δύο διαφορετικών κόμβων του να υπάρχει ακριβώς ένα απλό μονοπάτι (βλέπε ορισμό 2.5.3).

Δεδομένης της «συμμετρικότητας» του παραπάνω ορισμού ως προς οποιοδήποτε κόμβο του δένδρου, είναι προφανές ότι όλοι οι κόμβοι ενός δένδρου είναι ισοδύναμοι μεταξύ τους. Σκοπεύουμε να τοποθετήσουμε κάθε εμφάνιση συμβόλου σε έναν κόμβο ενός κατάλληλου πεπερασμένου δένδρου, να «επιγράψουμε», όπως λέμε, τους κόμβους του δένδρου.

Όμως, κάθε εμφάνιση συμβόλου του $\mathcal{F} \cup \mathcal{V}$ σε έναν όρο συνδέεται αφ' ενός με την συνάρτηση της οποίας είναι άμεσο όρισμα, εκτός φυσικά αν η εμφάνιση αντιστοιχεί σε ριζικό σύμβολο, και αφ' ετέρου με τα πιθανά ορίσματα που αντιστοιχούν σε αυτήν την εμφάνιση, εκτός φυσικά αν πρόκειται για μεταβλητή ή σταθερά. Έτσι, λόγου χάρη, η μοναδική εμφάνιση του $g$ στον όρο $f(g(a,b),h(a),c)$ του παραδείγματος 4.1.7 αντιστοιχίζεται αφ' ενός στην εμφάνιση του συμβόλου $f$ και αφ' ετέρου στις εμφανίσεις των συμβόλων $a, b$ (όχι του ορίσματος $a$ της $h$). Αυτό σημαίνει ότι οι γειτονικοί κόμβοι ενός επιγεγραμμένου κόμβου δεν είναι ισοδύναμοι με την έννοια ότι κάποιος αντιστοιχεί στην συνάρτηση της οποίας είναι άμεσο όρισμα η επιγραφή και αφ' ετέρου κάποιοι αντιστοιχούν σε ενδεχόμενα ορίσματα της επιγραφής.

Για αυτόν τον λόγο, πρέπει να δώσουμε μία κατεύθυνση στο δένδρο τέτοια ώστε κάθε κόμβος να έχει το πολύ έναν κόμβο (τον ονομάζουμε κόμβο *γονέα*) που αντιστοιχεί στην συνάρτηση της οποίας είναι άμεσο όρισμα και τόσους κόμβους (τους ονομάζουμε κόμβους *παιδιά*) όσα τα ορίσματα του.

Αυτό μπορεί να αποδειχθεί ότι γίνεται αν διαλέξουμε έναν κόμβο, τον οποίο θα ονομάσουμε *ρίζα* και θεωρήσουμε ότι όλοι οι κόμβοι με τους οποίους συνδέεται είναι παιδιά του.[2] Η ρίζα είναι ο μόνος κόμβος του δένδρου που δεν έχει γονέα.

Τελικά, το δένδρο που αντιστοιχεί σε έναν όρο $t$ κατασκευάζεται ως εξής : Έστω $t = f(t_1, \ldots, t_n)$. Αρχικά, τοποθετούμε το ριζικό σύμβολο $f$ στην

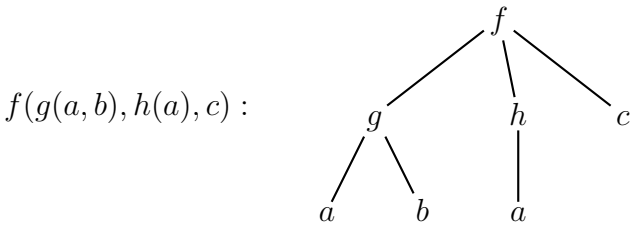
1. Για τους ορισμούς της συνεκτικότητας και της ακυκλικότητας, βλέπε ενότητα 2.5, σελίδα 17 κ.ε.

2. Μάλιστα, το δένδρο του ορισμού 4.2.1 ονομάζεται *δένδρο χωρίς ρίζα* (unrooted).

ρίζα του δένδρου και θεωρούμε τόσους κόμβους παιδιά της ρίζας όσο είναι το πλήθος των ορισμάτων του ριζικού συμβόλου και επαναλαμβάνουμε την διαδικασία για κάθε κόμβο, θεωρώντας αυτόν ως ρίζα και το αντίστοιχο ριζικό σύμβολο του εκάστοτε όρου $t_i$, για $1 \leq i \leq n$. Στο τέλος, οι μεταβλητές και οι σταθερές τοποθετούνται σε *φύλλα* του δένδρου, δηλαδή κόμβους οι οποίοι δεν έχουν παιδιά. Πρέπει να σημειώσουμε ότι για κάθε συνάρτηση που εμφανίζεται σε έναν όρο έχει σημασία η διάταξη των ορισμάτων αυτής. Έτσι, κατά σύμβαση, προσανατολίζουμε την δενδρική αναπαράσταση του όρου έτσι ώστε η ρίζα του να βρίσκεται ψηλότερα από οποιονδήποτε άλλο κόμβο και για κάθε συνάρτηση τα αριστερότερα ορίσματα να αντιστοιχούν σε αριστερότερα υποδένδρα.

Δεδομένης της παραπάνω ανάλυσης, χρησιμοποιούμε για την αναπαράσταση του κάθε όρου, αυτό που τυπικά ονομάζεται *πεπερασμένο διατεταγμένο δένδρο*.

**4.2.2 Παράδειγμα.** Ο όρος $f(g(a,b),h(a),c)$ (παράδειγμα 4.1.7) δίνεται στο σχήμα 4.1 μαζί με την δενδρική του αναπαράσταση.




Σχήμα 4.1 – Ένας όρος ως δένδρο

Κατά μία άλλη έννοια, ίσως πιο χρήσιμη στο πλαίσιο των όρων, ένα δένδρο με ρίζα μπορεί να οριστεί ως ένα *κλειστό ως προς το πρόθεμα σύνολο* (prefix closed set). Πριν όμως ορίσουμε την παραπάνω έννοια, είναι χρήσιμη η έννοια της συμβολοσειράς.

**4.2.3 Ορισμός** (Συμβολοσειρές και πρόθεμα)**.** Έστω ένα σύνολο $\Sigma$. Το σύνολο των πεπερασμένων ακολουθιών επί του $\Sigma$ συμβολίζεται με $\Sigma^*$. Κάθε $w \in \Sigma^*$ ονομάζεται *συμβολοσειρά* (string) επί του $\Sigma$ και συνήθως σημειώνεται ως εξής : $w = a_1 a_2 \ldots a_n$, δηλαδή απλώς παραθέτοντας τα στοιχεία $a_i \in \Sigma$, για $1 \leq i \leq n$. Ο αριθμός $n$ είναι το λεγόμενο *μήκος* της συμβολοσειράς, το οποίο, για οποιαδήποτε συμβολοσειρά $w$, συμβολίζεται με $\mathrm{strlen}(w)$ (από το string length).

Στο $\Sigma^*$ ορίζουμε την πράξη της *συνένωσης* (συμβολίζεται με . ή με απλή παράθεση των προς συνένωση λέξεων) δύο συμβολοσειρών $u = u_1 \ldots u_n$ και

$v = v_1 \ldots v_m$ ως εξής :

$$u.v \overset{\text{ορ.}}{=} uv \overset{\text{ορ.}}{=} u_1 \ldots u_n v_1 \ldots v_m$$

Προφανώς, η πράξη της συνένωσης είναι προσεταιριστική.

Στο σύνολο των συμβολοσειρών ανήκει και η *κενή συμβολοσειρά* (συμβολίζεται με $\varepsilon$ ή κάποιες φορές και με $\lambda$), η οποία είναι η μοναδική συμβολοσειρά μήκους 0 και αποτελεί το ουδέτερο στοιχείο για την πράξη της συνένωσης.

Έστω $w \in \Sigma^*$. Αν υπάρχουν $u, v \in \Sigma^*$ τέτοια ώστε $w = uv$, η συμβολοσειρά $u$ ονομάζεται *πρόθεμα* (prefix) της $w$.

Έστω ότι δίνεται μία συμβολοσειρά $w \in \Sigma^*$, μήκους $n$. Τότε μπορούμε να υπολογίσουμε όλα τα προθέματα αυτής αφαιρώντας διαδοχικά ένα ένα τα συστατικά σύμβολα της $w$, αρχίζοντας από το τέλος της. Έτσι, προκύπτουν $n + 1$ το πλήθος προθέματα, ενώ μεταξύ αυτών συμπεριλαμβάνεται η ίδια η συμβολοσειρά $w$ καθώς και η κενή συμβολοσειρά.

Σύμφωνα με τα παραπάνω, έχει έννοια ο παρακάτω ορισμός :

**4.2.4 Ορισμός** (Κλειστό ως προς το πρόθεμα σύνολο)**.** Έστω ένα σύνολο $X$ το οποίο είναι υποσύνολο κάποιου συνόλου $\Sigma^*$. Τότε αυτό το $X$ είναι *κλειστό ως προς το πρόθεμα*, αν για κάθε $w \in X$, κάθε πρόθεμα του $w$ ανήκει επίσης στο $X$.

*4.2.5 Παρατήρηση.* Ένα υποσύνολο του $\Sigma^*$ ονομάζεται *γλώσσα*, άρα το $X$ στον παραπάνω ορισμό είναι γλώσσα.

Η έννοια του κλειστού ως προς το πρόθεμα συνόλου εφαρμόζεται στην περίπτωση των δένδρων αν θεωρήσουμε ως συμβολοσειρές τις μοναδικές για κάθε κόμβο διαδρομές από την ρίζα στον εκάστοτε κόμβο. Είναι προφανές ότι για κάθε διαδρομή από την ρίζα σε έναν κόμβο, οποιοδήποτε πρόθεμα αυτής αντιστοιχεί επίσης σε μία διαδρομή προς έναν κόμβο «εσωτερικότερο» του πρώτου.

Τότε ένας όρος είναι απλώς μία συνάρτηση από αυτό το κλειστό ως προς το πρόθεμα σύνολο στο σύνολο $\mathcal{F} \cup \mathcal{V}$, με την προϋπόθεση, βέβαια, το πλήθος των παιδιών του κάθε κόμβου να είναι ίσο με το πλήθος των ορισμάτων της συνάρτησης που επιγράφεται επί αυτού.

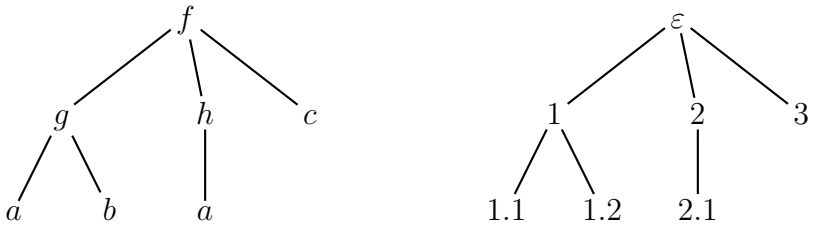
Προφανώς, τα δυνατά κλειστά ως προς το πρόθεμα σύνολα που περιγράφουν ένα δένδρο είναι άπειρα, ανάλογα με το πώς θα επιλέξουμε το σύνολο για την κατασκευή των συμβολοσειρών, αλλά και τις ίδιες τις συμβολοσειρές. Για αυτόν τον λόγο, για κάθε όρο ορίζουμε ένα συγκεκριμένο μοναδικό τέτοιο σύνολο ως εξής : Επιλέγουμε τις συμβολοσειρές των θετικών ακεραίων, έτσι ώστε να λαμβάνουμε όσο το δυνατόν μικρότερους αριθμούς. Ακολουθεί ο τυπικός ορισμός :

**4.2.6 Ορισμός.** Το *σύνολο θέσεων* $\mathrm{Pos}(t) \subseteq \mathbb{N}^*$ ενός όρου $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ ορίζεται ως εξής :

$$\mathrm{Pos}(t) = \begin{cases} \{\varepsilon\}, \text{ αν } \mathrm{root}(t) \in \mathcal{V} \cup \mathcal{F}_0 \\ \{\varepsilon\} \cup \{i.p \mid p \in \mathrm{Pos}(t_i), i = 1, \ldots, n\}, \\ \qquad \text{αν } t = f(t_1, \ldots, t_n) \text{ και } f \notin \mathcal{V} \cup \mathcal{F}_0 \end{cases}$$

Κάθε στοιχείο του συνόλου $\mathrm{Pos}(t)$ ονομάζεται *θέση* του όρου $t$.

**4.2.7 Παράδειγμα.** Οι θέσεις του όρου $f(g(a, b), h(a), c)$ (του παραδείγματος 4.1.7) φαίνονται στο σχήμα 4.2 δίπλα στην δενδρική του αναπαράσταση.




Σχήμα 4.2 – Δένδρο και θέσεις του

Προφανώς, για τα στοιχεία ενός συνόλου θέσεων ορίζεται η μερική διάταξη $\leq$ ως εξής : $p \leq q$ αν η $p$ είναι πρόθεμα της $q$. Αν $p \leq q$ λέμε ότι η $p$ είναι *πάνω* από την $q$ ή ότι η $q$ είναι *κάτω* από την $p$. Το αυστηρό μέρος της διάταξης ορίζεται, κατά τα συνήθη, ως εξής : $p < q$ αν και μόνον αν $p \leq q$ και $p \neq q$.

Αν δεν ισχύει κανένα από τα $p \geq q$ ή $q \geq p$ λέμε ότι οι θέσεις $p$, $q$ είναι *ξένες μεταξύ τους* ή *ανεξάρτητες* ή *παράλληλες* (συμβολισμός : $p \parallel q$).

**4.2.8 Ορισμός.** Το σύνολο των θέσεων ενός όρου $t$ στις οποίες εμφανίζεται μεταβλητή ορίζεται ως εξής :

$$\mathcal{V}\mathrm{Pos}(t) \stackrel{\text{ορ.}}{=} \{p \in \mathrm{Pos}(t) \mid t|_p \in \mathcal{V}\},$$

ενώ το σύνολο των θέσεων στις οποίες δεν βρίσκονται μεταβλητές :

$$\mathcal{F}\mathrm{Pos}(t) \stackrel{\text{ορ.}}{=} \{p \in \mathrm{Pos}(t) \mid \mathrm{root}(t|_p) \in \mathcal{F}\}.$$

Τα δύο σύνολα $\mathcal{V}\mathrm{Pos}(t)$, $\mathcal{F}\mathrm{Pos}(t)$ αποτελούν διαμέριση (βλέπε ορισμό 1.3.2 στην σελίδα 8) του $\mathrm{Pos}(t)$.

**4.2.9 Ορισμός.** Το σύνολο των μεταβλητών που εμφανίζονται στον όρο $t$ ορίζεται ως εξής :

$$\mathrm{Var}(t) \stackrel{\text{ορ.}}{=} \{t|_p \mid p \in \mathcal{V}\mathrm{Pos}(t)\}$$

**4.2.10 Ορισμός.** Για κάθε $p \in \mathrm{Pos}(t)$ ορίζεται ο *υποόρος* του $t$ στην θέση $p$ ως εξής :

$$t|_p = \begin{cases} t, \text{ αν } p = \varepsilon \\ t_i|_q, \text{ αν } p = i.q \text{ με } 1 \leq i \leq n, \text{ όπου } t = f(t_1, \ldots, t_n) \end{cases}$$

Έστω όροι $s$, $t$. Αν για κάποια θέση $p$ του $t$ ισχύει $s = t|_p$, τότε ο $s$ είναι υποόρος του $t$, συμβολικά $s \trianglelefteq t$.

Αν $t = f(t_1, \ldots, t_n)$, οι όροι $t_i = t|_i$, ονομάζονται *άμεσοι υποόροι* του $t$.

Ο υποόρος $t|_p$ ονομάζεται *γνήσιος* αν $p \neq \varepsilon$ ή ισοδύναμα $t|_p \neq t$. Αν ο $s$ είναι γνήσιος υποόρος του $t$, αυτό συμβολίζεται με $s \triangleleft t$.

Σε πολλές περιπτώσεις δεν αρκεί να γνωρίζουμε απλώς ότι κάποιος όρος $s$ είναι υποόρος κάποιου άλλου όρου $t$, αλλά θέλουμε να ξέρουμε και πόσες φορές καθώς και σε ποιες θέσεις εμφανίζεται. Έτσι, ορίζουμε τις *εμφανίσεις* του όρου $s$ στον $t$ ως εξής :

**4.2.11 Ορισμός.** Οι *εμφανίσεις* (occurences) ενός όρου $s$ στον όρο $t$ ορίζονται ως το σύνολο :

$$\mathrm{Occ}(t, s) \stackrel{\text{ορ.}}{=} \{p \in \mathrm{Pos}(t) \mid t|_p = s\}$$

**4.2.12 Παράδειγμα.** Για τον όρο $t = f(g(a, b), h(a), c)$ (σχήμα 4.2), έχουμε $\mathrm{Occ}(t, a) = \{1.1, 2.1\}$, $\mathrm{Occ}(t, h(a)) = \{2\}$, $\mathrm{Occ}(t, h(c)) = \{\}$ (επειδή ο $h(c)$ δεν είναι καν υποόρος του $t$).

Επίσης, είναι δυνατόν να τοποθετήσουμε στην θέση του υποόρου $t|_p$ έναν άλλο όρο, έστω τον $s$. Ο νέος όρος που προκύπτει συμβολίζεται με $t[s]_p$ και τυπικά ορίζεται ως εξής :

**4.2.13 Ορισμός** (Αντικατάσταση υποόρου)**.**

$$t[s]_p = \begin{cases} s, \text{ αν } p = \varepsilon \\ f(\ldots, t_i[s]_q, \ldots), \text{ αν } p = i.q \text{ με } 1 \leq i \leq n, \text{ όπου } t = f(t_1, \ldots, t_n) \end{cases}$$

## 4.3 Συναρτήσεις επί των όρων

**4.3.1 Ορισμός** (Ύψος)**.** Το *ύψος* ενός όρου είναι το μήκος του μεγαλύτερου (μη περιέχοντος κύκλο) μονοπατιού από την ρίζα του όρου σε οποιοδήποτε σύμβολο, αυξημένο κατά ένα. Ο τυπικός ορισμός είναι ως εξής :

$$\mathrm{Height}(t) = \begin{cases} 1, \text{ αν } \mathrm{root}(t) \in \mathcal{V} \cup \mathcal{F}_0 \\ 1 + \max\limits_{i=1,\ldots,n} \{\mathrm{Height}(t_i)\}, \text{ αν } t = f(t_1, \ldots, t_n) \text{ και } f \notin \mathcal{V} \cup \mathcal{F}_0 \end{cases}$$

**4.3.2 Ορισμός** (Μέγεθος)**.** Το *μέγεθος* (size) ενός όρου $t$ συμβολίζεται με $|t|$ και είναι το πλήθος των συμβόλων (συναρτησιακών είτε μεταβλητών) που περιλαμβάνει. Τυπικά :

$$|t| = \begin{cases} 1, \text{ αν } \mathrm{root}(t) \in \mathcal{V} \cup \mathcal{F}_0 \\ 1 + \sum_{i=1}^{n} |t_i|, \text{ αν } t = f(t_1, \ldots, t_n) \text{ και } f \notin \mathcal{V} \cup \mathcal{F}_0 \end{cases}$$

**4.3.3 Παράδειγμα.** Ο όρος $f(g(a,b), h(a), c)$ (παράδειγμα 4.1.7) έχει ύψος ίσο με 3 και μέγεθος ίσο με 7 (βλέπε και σχήμα 4.1).

## 4.4 Περιβάλλοντα και αντικαταστάσεις

**4.4.1 Ορισμός.** Άτυπα, ένα *περιβάλλον* ή συμφραζόμενο (context) είναι είναι ένας όρος με μία «οπή» σε κάποια θέση $p$. Η οπή αυτή συνήθως συμβολίζεται με $\Box$ και μπορούμε εναλλακτικά να την θεωρήσουμε ως μία επιπλέον μεταβλητή ή μία επιπλέον σταθερά, που χρησιμοποιείται αποκλειστικά για αυτόν τον σκοπό. [3]

Η έννοια που έχει ένα περιβάλλον είναι ότι μπορούμε να τοποθετήσουμε στην οπή του έναν οποιονδήποτε όρο $s$, δηλαδή να αντικαταστήσουμε το $\Box$ με το $s$. Έτσι, όπως δηλώνει και το όνομά του, το περιβάλλον παρέχει ένα πλαίσιο στο οποίο ο τοποθετούμενος όρος «περιβάλλεται». Χρησιμοποιώντας τον ορισμό 4.2.13 και θεωρώντας ότι το περιβάλλον είναι ο όρος $C$ με την οπή στην θέση $p$, το αποτέλεσμα της παραπάνω αντικατάστασης είναι $C[s]_p$. Έτσι, ένα περιβάλλον συμβολίζεται συχνά ως εξής : $C[\,]_p$ ή απλώς $C[\,]$. Παρομοίως, από έναν όρο $t$ μπορεί να προκύψει ένα περιβάλλον με οπή στην θέση $p$ του όρου ως εξής : $t[\Box]_p$.

**4.4.2 Παράδειγμα.** Συνεχίζοντας στα πλαίσια του παραδείγματος 4.1.7, στην σελίδα 34, ένα παράδειγμα περιβάλλοντος είναι το $f(a, \Box, b)$. Αν το παραπάνω περιβάλλον συμβολιστεί με $C[\,]_p$, όπου $p = 2$, τότε $C[g(a,b)] = f(a, g(a,b), b)$.

Η έννοια του περιβάλλοντος μπορεί να επεκταθεί ώστε να συμπεριλάβει όρους με περισσότερες από μία οπές, δηλαδή παραπάνω από μία δεσμευμένες μεταβλητές. Αυτή η επέκταση, καθώς και η διάκριση της ύπαρξης οπής ή τετριμμένου περιβάλλοντος φαίνεται απαραίτητη για την συστηματικότερη μελέτη των όρων και την απλοποίηση των αποδείξεων (βλέπε και [69]). Πάντως, στην παρούσα εργασία, μας αρκεί ο ορισμός του περιβάλλοντος που δώσαμε παραπάνω.

3. Τυπικά, ένα περιβάλλον μπορεί να θεωρηθεί ως ένας όρος με μία δεσμευμένη μεταβλητή (βλέπε [30]).

**4.4.3 Ορισμός.** Μία *αντικατάσταση* $\sigma$ είναι μία απεικόνιση από το σύνολο των μεταβλητών $\mathcal{V}$ στο σύνολο των όρων $\mathcal{T}(\mathcal{F}, \mathcal{V})$, τέτοια ώστε μόνο ένα πεπερασμένο [4] πλήθος μεταβλητών να μην απεικονίζεται στον εαυτό του, δηλαδή τέτοια ώστε το σύνολο $\{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ να είναι πεπερασμένο.

Έστω σύνολο μεταβλητών $\{x_1, x_2, \dots\}$. Μία αντικατάσταση συχνά συμβολίζεται ως εξής :

$$\sigma = \{x_{i_1} := t_1, \dots, x_{i_n} := t_n\}$$

**4.4.4 Ορισμός.** Μία αντικατάσταση $\sigma$ από το $\mathcal{V}$ στο $\mathcal{T}(\mathcal{F}, \mathcal{V})$ μπορεί να επεκταθεί έτσι ώστε να απεικονίζει όρους σε όρους. Την επέκταση αυτή συμβολίζουμε πάλι με $\sigma$, αλλά συνήθως την γράφουμε ως επιθεματικό (postfix) τελεστή. Η επέκταση αυτή ορίζεται, κατά μοναδικό τρόπο, ώστε να ισχύει :

$$f(t_1, \dots, t_n)^\sigma = f({t_1}^\sigma, \dots, {t_n}^\sigma), \text{ για κάθε } f \in \mathcal{F} \text{ και όρους } t_1, \dots, t_n$$

Μπορούμε να συνθέσουμε δύο αντικαταστάσεις, έστω $\sigma$ και $\tau$, λαμβάνοντας μία καινούρια αντικατάσταση την οποία συμβολίζουμε $\tau \circ \sigma$ ή σε επιθεματικό συμβολισμό $\sigma\tau$. Δηλαδή για έναν όρο $t$ έχουμε $(\tau \circ \sigma)(t) = \tau(\sigma(t))$ ή απλώς $t^{\sigma\tau}$. [5] Το ουδέτερο στοιχείο της πράξης της *σύνθεσης* αντικαταστάσεων είναι η λεγόμενη *κενή* αντικατάσταση, δηλαδή αυτή που απλώς απεικονίζει κάθε μεταβλητή στον εαυτό της και η οποία συμβολίζεται με $\varepsilon$.

## 4.5 Συμπερίληψη και ενοποίηση

**4.5.1 Ορισμός.** Λέμε ότι ο όρος $s$ *συμπεριλαμβάνει* (subsumes) τον όρο $t$, αν υπάρχει αντικατάσταση $\sigma$ τέτοια ώστε $t = s^\sigma$. Εναλλακτικά, λέμε ότι ο $t$ αποτελεί *στιγμιότυπο* (instance) του $s$ και συμβολίζουμε $s \leq t$. Η σχέση $\leq$ είναι μία σχεδόν διάταξη (βλέπε ορισμό 1.4.1 στην σελίδα 9).

Αν $s \leq t$, ο όρος $s$ είναι «γενικότερος» από τον $t$. Από την πρόταση 1.4.4 (σελίδα 10) ορίζεται και η σχέση ισοδυναμίας $\doteq \stackrel{\text{ορ.}}{=} \leq \cap \geq$ που ονομάζεται *ομοιότητα* (literal similarity) [6]. Αν $s \doteq t$, οι δύο όροι είναι εξίσου γενικοί και ο ένας μπορεί να προκύψει από τον άλλον με πιθανή απλή μετονομασία των μεταβλητών που εμφανίζονται σε αυτούς.

4. Η απαίτηση για πεπερασμένο πλήθος μεταβλητών που δεν απεικονίζονται στον εαυτό τους δεν είναι σημαντική, αλλά συνηθίζεται στην βιβλιογραφία, δεδομένου ότι είναι αρκετή για την ανάπτυξη της θεωρίας των συστημάτων μεταγραφής.

5. Εδώ φαίνεται πόσο βολικός είναι ο επιθεματικός συμβολισμός, δεδομένου ότι ισχύει $(t^\sigma)^\tau = t^{\sigma\tau}$ και δεν χρειαζόμαστε καθόλου παρενθέσεις. (Βέβαια η σειρά με την οποία γίνονται οι αντικαταστάσεις είναι σημαντική, δηλαδή η πράξη της σύνθεσης δεν είναι αντιμεταθετική, αφού εν γένει $t^{\sigma\tau} \neq t^{\tau\sigma}$.)

6. Επίσης αναφέρεται ως μετονομασία (renaming) ή $\alpha$-μετατροπή ($\alpha$-conversion) στα πλαίσια του $\lambda$-λογισμού.

**4.5.2 Παράδειγμα.** Για όρους όπως στο παράδειγμα 4.1.7 έχουμε $g(x,a) \leq g(h(b),a)$ ή για την ακρίβεια $g(x,a) \lessdot g(h(b),a)$. Επίσης, $g(x,y) \doteq g(y,z)$, αλλά $g(x,x) \not\doteq g(x,y)$.

**4.5.3 Ορισμός.** Δύο όροι $s,t$ ονομάζονται *ενοποιήσιμοι* (unifiable) αν υπάρχει αντικατάσταση $\sigma$ τέτοια ώστε $s^\sigma = t^\sigma$. Σε αυτήν την περίπτωση, η αντικατάσταση $\sigma$ ονομάζεται *ενοποιητής* ή ενοποιούσα αντικατάσταση. Επίσης, λέμε ότι η «εξίσωση» $s = t$ έχει λύση την $\sigma$.

Μπορούμε να επεκτείνουμε την έννοια της ενοποίησης ακόμα και αν έχουμε περισσότερους από δύο όρους, απλώς πρέπει να υπάρχει αντικατάσταση που ενοποιεί ταυτόχρονα όλους τους όρους.

**4.5.4 Παράδειγμα.** Οι όροι $g(z,y)$ και $g(a,h(x))$ ενοποιούνται από την αντικατάσταση $\{y := h(a), z := a\}$. Αντιθέτως, οι όροι $g(x,a)$ και $g(b,x)$ δεν ενοποιούνται αν $a \neq b$.

Η συμπερίληψη και η σχέση υποόρου είναι ειδικές περιπτώσεις μίας άλλης σχεδόν διάταξης :

**4.5.5 Ορισμός.** Λέμε ότι ο όρος $s$ *περικλείει* τον $t$ αν υπάρχει υποόρος του $t$ που είναι στιγμιότυπο του $s$, συμβολικά $s \trianglelefteq t$.

Η σχέση του «περικλείειν» αναφέρεται στα αγγλικά ως encompassment ή και containment στο [53].

**4.5.6 Παράδειγμα.** Ισχύει $h(x) \lhd g(h(a),b)$, αφού ο υποόρος $h(a)$ του $g(h(a),b)$ είναι στιγμιότυπο του $h(x)$.

## 4.6 Αλγόριθμος εύρεσης γενικότερου ενοποιητή

Ας θεωρήσουμε σύνολο όρων $\mathcal{T} = \mathcal{T}(\mathcal{F},\mathcal{V})$ και το σύνολο $\mathcal{T}/\doteq$ των συνόλων ισοδυναμίας που επιβάλλει η σχέση ισοδυναμίας $\doteq$. Από εδώ και στο εξής θα αναφερόμαστε σε στοιχεία του συνόλου $\mathcal{T}/\doteq$, δηλαδή δεν θα διακρίνουμε όρους οι οποίοι προκύπτουν ο ένας από τον άλλον με μετονομασία. Το $\mathcal{T}/\doteq$ διατάσσεται από την σχέση συμπερίληψης. Για κάθε ζεύγος όρων $s$, $t$ υπάρχει πάντοτε όρος ο οποίος είναι ταυτόχρονα γενίκευση και του $s$ και του $t$ (προφανώς ο όρος μεταβλητή γενικεύει οποιονδήποτε όρο). Όμως, το σύνολο όλων των γενικεύσεων των $s$, $t$ είναι και κάτω φραγμένο· μάλιστα περιέχει όρο-γενίκευση ο οποίος αποτελεί στιγμιότυπο οποιασδήποτε άλλης γενίκευσης. Αυτός ο (μοναδικός μέχρι μετονομασίας) όρος ονομάζεται *ελάχιστη γενίκευση* και συμβολίζεται $\mathrm{glb}(s,t)$ από το greatest lower bound (βλέπε [85]).

**4.6.1 Παράδειγμα.** Οι όροι $f(a, h(b), h(a))$ και $f(c, h(a), h(c))$ έχουν ελάχιστη γενίκευση $f(x, h(y), h(x))$.

Παρομοίως, για κάθε ζεύγος όρων που είναι φραγμένοι από πάνω (δηλαδή υπάρχει όρος που είναι στιγμιότυπο και των δύο ή ισοδύναμα οι δύο όροι είναι ενοποιήσιμοι) υπάρχει ένα ελάχιστο (ή αλλιώς γενικότερο) άνω φράγμα των κοινών στιγμιοτύπων που συμβολίζεται $\mathrm{lub}(s, t)$ από το least upper bound (βλέπε [6]). Το $\mathrm{lub}(s, t)$ είναι επίσης μοναδικό μέχρι μετονομασίας.

Η έννοια της γενίκευσης (ή συμπερίληψης) μεταφέρεται με φυσικό τρόπο από τους όρους στις αντικαταστάσεις :

**4.6.2 Ορισμός.** Μία αντικατάσταση $\sigma$ ονομάζεται *γενικότερη* από την $\sigma'$ (συμβολισμός : $\sigma \leq \sigma'$) αν υπάρχει αντικατάσταση $\tau$ τέτοια ώστε $\sigma\tau = \sigma'$ (υπενθυμίζουμε την σύνθεση αντικαταστάσεων $\sigma\tau = \tau \circ \sigma$).

**4.6.3 Ορισμός.** Ο *γενικότερος ενοποιητής* (most general unifier) δύο ενοποιήσιμων όρων $s$, $t$ συμβολίζεται με $\mathrm{mgu}(s, t)$ και είναι η γενικότερη αντικατάσταση μεταξύ των ενοποιουσών αντικαταστάσεων των $s$, $t$.

**4.6.4 Παράδειγμα.** Όπως είδαμε στο παράδειγμα 4.5.4, οι όροι $g(z, y)$ και $g(a, h(x))$ ενοποιούνται από την αντικατάσταση $\{y := h(a), z := a\}$. Αυτή όμως η αντικατάσταση δεν είναι ο γενικότερος ενοποιητής, αφού ισχύει : $\mathrm{mgu}(g(z, y), g(a, h(x))) = \{y := h(x), z := a\} \lessdot \{y := h(a), z := a\}$.

Εύκολα, μπορεί να διαπιστώσει κανείς ότι η έννοια του γενικότερου ενοποιητή (mgu) συνδέεται με την έννοια του ελάχιστου άνω φράγματος (lub) ως εξής : Αν $\sigma = \mathrm{mgu}(s, t)$, τότε, για τυχαίο δυαδικό συναρτησιακό σύμβολο eq και τυχαία μεταβλητή $x$, οι όροι $s^\sigma$, $t^\sigma$ και $\mathrm{lub}(\mathrm{eq}(x, x), \mathrm{eq}(s, t))$ είναι ίσοι (πιθανόν μέχρι μετονομασίας). Από το παραπάνω προκύπτει η μοναδικότητα του γενικότερου ενοποιητή μέχρι μετονομασίας. Για περισσότερες λεπτομέρειες, βλέπε [30].

Ο πρώτος αλγόριθμος εύρεσης του γενικότερου ενοποιητή αποδίδεται στον Robinson (βλέπε [88]). Προκειμένου να περιγραφεί ένας αλγόριθμος για την ενοποίηση πρέπει να ορίσουμε τα αντικείμενα επί των οποίων εφαρμόζεται. Έτσι, θα θεωρήσουμε ένα σύνολο «εξισώσεων» προς επίλυση, το οποίο συμβολίζουμε με $E$ και μία αντικατάσταση $\sigma$. Ο αλγόριθμος ενεργεί επί του ζεύγους $(E, \sigma)$ : Σε κάθε βήμα, το $E$ αντιπροσωπεύει τις εξισώσεις που απομένει να επιλυθούν και το $\sigma$ την μερική λύση. Ένας κομψός αλγόριθμος για ενοποίηση που βασίζεται στην παραπάνω παράσταση δίνεται από τους Martelli και Montanari στο [67]. Η περιγραφή του αλγορίθμου βασίζεται στα [59, 30]. Αν θέλουμε να βρούμε τον $\mathrm{mgu}(s, t)$ δίνουμε ως αρχική είσοδο στον αλγόριθμο το ζεύγος $(\{s = t\}, \varepsilon)$. Μετά, εφαρμόζει κανείς (πιθανόν μη αιτιοκρατικά) τους λεγόμενους κανόνες MM :

1. **Αποσύνθεση**

$$(\{f(s_1,\ldots,s_n) = f(t_1,\ldots,t_n)\} \cup E, \sigma) \Rightarrow_{\text{MM}} (\{s_1 = t_1,\ldots,s_n = t_n\} \cup E, \sigma).$$

2. **Διαγραφή** τετριμμένης εξίσωσης

$$(\{x = x\} \cup E, \sigma) \Rightarrow_{\text{MM}} (E, \sigma).$$

3. **Εναλλαγή**

$$(\{t = x\} \cup E, \sigma) \Rightarrow_{\text{MM}} (\{x = t\} \cup E, \sigma)$$

αν $t \notin \mathcal{V}$ και $x \in \mathcal{V}$.

4. **Απαλοιφή** μεταβλητής

$$(\{x = t\} \cup E, \sigma) \Rightarrow_{\text{MM}} (E^{\{x:=t\}}, \sigma\{x := t\})$$

αν το $x$ δεν είναι υποόρος του $t$ (δηλαδή $x \not\trianglelefteq t$). Επεξηγούμε ότι αν είναι $E = \{s_1 = t_1,\ldots,s_n = t_n\}$, ο συμβολισμός $E^\sigma$ που χρησιμοποιήθηκε παραπάνω ορίζεται ως $\{s_1^\sigma = t_1^\sigma,\ldots,s_n^\sigma = t_n^\sigma\}$.

Οι παραπάνω κανόνες αν εφαρμοστούν μη αιτιοκρατικά δίνουν ένα δένδρο υπολογισμού. Ένα υπολογιστικό μονοπάτι του δένδρου υπολογισμού είναι *επιτυχές* αν το ζεύγος $(E, \sigma)$ φτάσει στην μορφή $(\emptyset, \sigma)$ (δηλαδή δεν έχουμε άλλες εξισώσεις προς επίλυση), οπότε το $\sigma$ αντιπροσωπεύει την τελική λύση (τον γενικότερο ενοποιητή). Αυτό ισχύει αφ' ενός επειδή οι κανόνες *δεν μεταβάλλουν* το πλήθος των λύσεων και αφ' ετέρου επειδή *διατηρούν* όλες τις λύσεις. Αντιθέτως, ένα μονοπάτι είναι ανεπιτυχές αν φτάσουμε σε ζεύγος της μορφής $(E, \sigma)$ με $E \neq \emptyset$ και δεν είναι δυνατόν να εφαρμοστεί κανένας από τους παραπάνω κανόνες. Αν οι όροι είναι ενοποιήσιμοι τότε *όλα* τα υπολογιστικά μονοπάτια είναι επιτυχή. Αν οι όροι όμως δεν είναι ενοποιήσιμοι, τότε *κάθε* υπολογιστικό μονοπάτι τερματίζει ανεπιτυχώς.

Πρέπει να παρατηρήσουμε ότι σε κάθε περίπτωση (είτε υπάρχει ενοποιητής είτε όχι) το δένδρο υπολογισμού είναι πεπερασμένο. Αυτό συμβαίνει επειδή σε κάθε βήμα (δηλαδή σε κάθε εφαρμογή κανόνα) το σύνολο $E$ των προς επίλυση εξισώσεων συνεχώς απλοποιείται· μπορεί βέβαια με τον κανόνα αποσύνθεσης να αυξάνεται το πλήθος των εξισώσεων, αλλά ταυτόχρονα μειώνεται το ύψος των όρων που συμμετέχουν σε αυτές και με αυτήν την έννοια κάθε μία από τις εξισώσεις που προκύπτει είναι απλούστερη από την αρχική. Βέβαια, από την άλλη η αντικατάσταση $\sigma$ με εφαρμογή του κανόνα απαλοιφής μεταβλητής γίνεται μεγαλύτερη. Και πάλι όμως, αυτό το γεγονός δεν μπορεί να δώσει ένα άπειρο υπολογιστικό μονοπάτι, δεδομένου ότι το μέγεθος της αντικατάστασης δεν μπορεί να μεγαλώσει περισσότερο από το (πεπερασμένο) πλήθος των μεταβλητών που εμφανίζονται στους αρχικούς προς ενοποίηση όρους.

Μπορούμε όμως να δώσουμε επιπλέον δύο κανόνες για τον ταχύτερο εντοπισμό της *αποτυχίας* ενοποίησης δύο όρων :

5. **Αδύνατη αποσύνθεση**

$$(\{f(s_1,\ldots,s_n) = g(t_1,\ldots,t_n)\} \cup E, \sigma) \Rightarrow_{\text{MM}} \textbf{Αποτυχία!}$$

6. **Έλεγχος παρουσίας**

$$(\{x = t\} \cup E, \sigma) \Rightarrow_{\text{MM}} \textbf{Αποτυχία!}$$

αν το $x$ είναι γνήσιος υποόρος του $t$ (δηλαδή $x \lhd t$).

Δεδομένης της παραπάνω ανάλυσης, μπορούμε πολύ εύκολα να φτιάξουμε έναν αιτιοκρατικό αλγόριθμο. Για την ορθότητα του αλγορίθμου, δεν έχει σημασία η επιλογή του κανόνα που εφαρμόζεται σε κάθε βήμα. Για την πολυπλοκότητα όμως του αλγορίθμου, είναι σημαντική η επιλογή του εκάστοτε κανόνα. Για παράδειγμα, ανεξέλεγκτη χρήση του κανόνα της απαλοιφής μπορεί να οδηγήσει σε εκθετική πολυπλοκότητα χρόνου. Με κατάλληλη επιλογή δομών δεδομένων και επιλογή κανόνων μπορεί να κατασκευαστεί *σχεδόν γραμμικός* αλγόριθμος (βλέπε [4]). Στην βιβλιογραφία υπάρχει και *γραμμικός* αλγόριθμος, ο οποίος όμως δεν είναι τόσο πρακτικός (βλέπε [83]).

**4.6.5 Παράδειγμα.** Έστω οι όροι $f(x, h(x), g(x,u))$, $f(x,z,g(g(y,y),z))$. Έχουμε :

$$\begin{aligned}
&(\{f(x,h(x),g(x,u)) = f(x,z,g(g(y,y),z))\},\varepsilon) && \Rightarrow^{\textbf{Αποσύνθεση}}_{\text{MM}}\\
&(\{x = x, h(x) = z, g(x,u) = g(g(y,y),z)\},\varepsilon) && \Rightarrow^{\textbf{Διαγραφή}}_{\text{MM}}\\
&(\{h(x) = z, g(x,u) = g(g(y,y),z)\},\varepsilon) && \Rightarrow^{\textbf{Εναλλαγή}}_{\text{MM}}\\
&(\{z = h(x), g(x,u) = g(g(y,y),z)\},\varepsilon) && \Rightarrow^{\textbf{Απαλοιφή}}_{\text{MM}}\\
&(\{g(x,u) = g(g(y,y),h(x))\},\{z := h(x)\}) && \Rightarrow^{\textbf{Αποσύνθεση}}_{\text{MM}}\\
&(\{x = g(y,y), u = h(x)\},\{z := h(x)\}) && \Rightarrow^{\textbf{Απαλοιφή}}_{\text{MM}}\\
&(\{u = h(g(y,y))\},\{x := g(y,y), z := h(g(y,y))\}) && \Rightarrow^{\textbf{Απαλοιφή}}_{\text{MM}}\\
&(\emptyset,\{x := g(y,y), z := h(g(y,y)), u := h(g(y,y))\})
\end{aligned}$$

Δηλαδή οι αρχικοί όροι είναι ενοποιήσιμοι με γενικότερο ενοποιητή την αντικατάσταση : $\{x := g(y,y), z := h(g(y,y)), u := h(g(y,y))\}$.

Ας δούμε και μία περίπτωση όπου οι όροι δεν είναι ενοποιήσιμοι (εδώ θα χρησιμοποιήσουμε και τους κανόνες αποτυχίας) : Έστω οι $g(x,y)$ και $g(y,h(x))$. Έχουμε :

$$\begin{aligned}
&(\{g(x,y) = g(y,h(x))\},\varepsilon) && \Rightarrow^{\textbf{Αποσύνθεση}}_{\text{MM}}\\
&(\{x = y, y = h(x)\},\varepsilon) && \Rightarrow^{\textbf{Απαλοιφή}}_{\text{MM}}
\end{aligned}$$

$(\{y = h(y)\}, \{x := y\})$ $\Rightarrow_{\text{MM}}^{\textbf{Έλεγχος παρουσίας}}$

**Αποτυχία!**

# Κεφάλαιο 5

# Συστήματα μεταγραφής όρων

Στο προηγούμενο κεφάλαιο δώσαμε τον ορισμό των όρων, των αντικειμένων δηλαδή που μεταγράφονται στα πλαίσια ενός συστήματος μεταγραφής όρων. Το μόνο που μένει είναι να περιγράψουμε την διαδικασία μεταγραφής. Όμως, δεν συνηθίζεται αυτή η σχέση, όπως στην περίπτωση των αφηρημένων συστημάτων μεταγραφής, να είναι μία οποιαδήποτε διμελής σχέση επί του συνόλου των όρων. Αντιθέτως, η σχέση ενός συστήματος μεταγραφής όρων έχει κάποιους περιορισμούς στην δομή της, ούτως ώστε να εκμεταλλευόμαστε ιδιότητες που σχετίζονται με γενικότερες σχέσεις που υπάρχουν στους όρους, όπως αυτήν της συμπερίληψης, ή διαδικασίες όπως αυτήν της αντικατάστασης.

Αυτό το κεφάλαιο βασίζεται κυρίως στα [59, 30, 31]. Άλλες αναφορές είναι τα [1, 84].

## 5.1 Σχέση και κανόνες μεταγραφής

Δίνουμε τον ορισμό του συστήματος μεταγραφής όρων :

**5.1.1 Ορισμός.** Ένα *σύστημα μεταγραφής όρων* (term rewriting system ή trs για συντομία) είναι η πλειάδα $(\mathcal{F}, \mathcal{R})$, όπου $\mathcal{F}$ : αλφάβητο συναρτησιακών συμβόλων και $\mathcal{R} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathcal{T}(\mathcal{F}, \mathcal{V})$ : το σύνολο κανόνων, ενώ ο κάθε κανόνας $(l, r)$ συμβολίζεται με $l \to r$. Πολλές φορές, το σύστημα μεταγραφής το συμβολίζουμε και ως $\mathcal{R}^{\mathcal{F}}$ ή απλώς $\mathcal{R}$.

Συνήθως, επιβάλλουμε τους εξής περιορισμούς στην μορφή κάθε κανόνα $l \to r$ : $l \notin \mathcal{V}$ και $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$. Οι περιορισμοί αυτοί αποκλείουν κάποιες παθολογικές καταστάσεις : ο πρώτος έχει την έννοια ότι δεν είναι δυνατόν να εφαρμόσουμε έναν κανόνα σε οποιονδήποτε όρο/υποόρο, ενώ ο δεύτερος ότι

δεν είναι δυνατόν κατά την μεταγραφή να εισάγουμε άσχετους καινούριους υποόρους. Παρ' όλα αυτά, σε ορισμένες περιπτώσεις, ειδικά αν οι κανόνες μεταφράζονται σε εξισώσεις ($l = r$), έχει νόημα να καταργήσουμε τους παραπάνω περιορισμούς, προκειμένου να διατηρήσουμε κάποιες επιθυμητές ιδιότητες συμμετρίας. Για περισσότερες λεπτομέρειες, πάνω σε αυτό το θέμα, βλέπε [45].

Για τις σχέσεις επί των όρων που θα χρησιμοποιήσουμε στα συστήματα μεταγραφής όρων απαιτούμε επιπλέον δομή :

**5.1.2 Ορισμός.** Μία διμελής σχέση $\rightarrow$ ορισμένη επί συνόλου όρων είναι *σχέση μεταγραφής όρων* όταν είναι *κλειστή ως προς τα περιβάλλοντα και τις αντικαταστάσεις*, δηλαδή όταν για τους όρους $s$, $t$ ισχύει $s \rightarrow t$, τότε $C[s] \rightarrow C[t]$ και $s^\sigma \rightarrow t^\sigma$, για κάθε περιβάλλον $C$ και κάθε αντικατάσταση $\sigma$.

*5.1.3 Παρατήρηση.* Η ιδιότητα της κλειστότητας ως προς τα περιβάλλοντα αναφέρεται και ως *μονοτονικότητα* (monotonicity) ή ιδιότητα *υποκατάστασης* (replacement), ενώ της κλειστότητας ως προς τις αντικαταστάσεις και ως *σταθερότητα* (stability) ή *συμβατότητα* (compatibility) ή ιδιότητα *πλήρους αμεταβλητότητας* (full invariance).

Εύκολα μπορεί να δειχθεί ότι η αντίστροφη, καθώς και το μεταβατικό, το συμμετρικό, το ανακλαστικό κλείσιμο μίας σχέσης μεταγραφής είναι όλα σχέσεις μεταγραφής. Επίσης, η ένωση και η τομή δύο σχέσεων μεταγραφής είναι σχέση μεταγραφής.

**5.1.4 Ορισμός.** Η *σχέση μεταγραφής που προκύπτει από τους κανόνες ενός συστήματος μεταγραφής όρων* $(\mathcal{F}, \mathcal{R})$ είναι η μικρότερη σχέση μεταγραφής όρων (όπως ορίστηκε παραπάνω) που περιέχει την $\mathcal{R}$ και συμβολίζεται με $\rightarrow_{\mathcal{R}}$. Ισοδύναμα, $s \rightarrow_{\mathcal{R}} t$ αν και μόνον αν υπάρχει περιβάλλον $C$ και αντικατάσταση $\sigma$, τέτοια ώστε για κάποιον κανόνα $l \rightarrow r \in \mathcal{R}$, να ισχύει : $s = C[l^\sigma]$ και $t = C[r^\sigma]$. Σε αυτήν την περίπτωση, λέμε ότι το $s$ *μεταγράφεται* ή *ανάγεται* στο $t$ με *συστολή της ανάξιμης έκφρασης* (contraction of reducible expression ή απλώς redex) $l^\sigma$. Ως ανάξιμη έκφραση (ή *ανάξιμο*) μπορεί να χρησιμοποιηθεί οποιοδήποτε στιγμιότυπο αριστερού μέλους ενός κανόνα. Το αποτέλεσμα της μεταγραφής στο ανάξιμο $l^\sigma$, δηλαδή το $r^\sigma$, ονομάζεται *συνεσταλμένο* (contractum). Προκειμένου να δείξουμε την θέση $p$ του αναξίμου που συστέλλεται, τον κανόνα $l \rightarrow r$ και την αντικατάσταση $\sigma$ που εφαρμόζονται, χρησιμοποιούμε επίσης συμβολισμούς της μορφής : $s \rightarrow_p t$, $s \rightarrow_{p,l\rightarrow r} t$, $s \rightarrow_{p,\sigma,l\rightarrow r} t$.

Οι όροι «συστολή», «αναγωγή», «ανάξιμο», κ.τ.λ., προέρχονται από την κλασική εφαρμογή των συστημάτων μεταγραφής στην απλοποίηση εκφράσεων, όπου μία έκφραση (το ανάξιμο ή redex) σε έναν όρο συστέλλεται,

δηλαδή απλοποιείται ή σμικρύνεται, δίνοντας μία ισοδύναμή της (το συνεσταλμένο ή contractum). Στην πραγματικότητα, όπως είδαμε στο κεφάλαιο περί αφηρημένων συστημάτων μεταγραφής, υπάρχουν συστήματα τα οποία ενδέχεται να μην τερματίζουν και να μην «συστέλλουν» τα αντικείμενα μεταγραφής τους.

*5.1.5 Παρατήρηση.* Μπορούμε να θεωρήσουμε ότι οι κανόνες είναι *καθολικά ποσοτικοποιημένοι* (universally quantified) ως προς τις μεταβλητές, δεδομένου ότι, κατά την διαδικασία μεταγραφής, στην θέση των μεταβλητών αντικαθίσταται οποιοσδήποτε όρος. Έτσι, δεν έχει σημασία ποιες ακριβώς μεταβλητές χρησιμοποιούμε σε κάθε κανόνα, αλλά η αντιστοιχία που υπάρχει μεταξύ αριστερού και δεξιού μέρους. Αυτό σημαίνει ότι ένα σύστημα μεταγραφής όρων διατηρεί την σχέση μεταγραφής του, αν απλώς *μετονομάσουμε* τις μεταβλητές. Αυτή η ιδιότητα θα μας φανεί χρήσιμη παρακάτω.

## 5.2 Παραδείγματα και ιδιότητες

Σε αυτήν την ενότητα θα δώσουμε κάποια παραδείγματα συστημάτων μεταγραφής όρων. Θα δούμε επίσης ιδιότητες που σχετίζονται με την συμβολή και τον τερματισμό, όπως αυτά ορίστηκαν στο κεφάλαιο 3, αλλά στα πλαίσια των συστημάτων μεταγραφής όρων.

**5.2.1 Παράδειγμα** (Συνδυαστική λογική)**.** Έστω σύστημα με σταθερές $S$, $K$, $I$ και την συνάρτηση εφαρμογής (application) Ap με δύο ορίσματα. Η *συνδυαστική λογική* είναι το σύστημα μεταγραφής όρων με κανόνες :

$$\begin{aligned} \mathrm{Ap}(\mathrm{Ap}(\mathrm{Ap}(S,x),y),z)) &\to \mathrm{Ap}(\mathrm{Ap}(x,z),\mathrm{Ap}(y,z)) \\ \mathrm{Ap}(\mathrm{Ap}(K,x),y) &\to x \\ \mathrm{Ap}(I,x) &\to x \end{aligned}$$

Συνήθως, η συνάρτηση Ap συμβολίζεται ως ο ενθεματικός τελεστής «$\circ$», οπότε οι παραπάνω κανόνες γίνονται :

$$\begin{aligned} (((S \circ x) \circ y) \circ z) &\to ((x \circ z) \circ (y \circ z))) \\ ((K \circ x) \circ y) &\to x \\ (I \circ x) &\to x \end{aligned}$$

Επιπλέον απλοποίηση προκύπτει αν καταργήσουμε τον τελεστή «$\circ$» συμβολίζοντας την εφαρμογή $(x \circ y)$ με την απλή παράθεση $(xy)$ και επιπλέον αν θεωρήσουμε ότι η εφαρμογή προσεταιρίζει από τα αριστερά, δηλαδή $xyz = ((xy)z)$, οπότε μειώνεται σημαντικά το πλήθος των απαραιτήτων παρενθέσεων. Τότε

οι κανόνες γίνονται :

$$\begin{aligned} Sxyz &\to xz(yz) \\ Kxy &\to x \\ Ix &\to x \end{aligned}$$

Το σύστημα της συνδυαστικής λογικής δεν είναι τερματίζον, αφού, για παράδειγμα, για τον ακόλουθο όρο $\Omega$ έχουμε :

$$\Omega \stackrel{\text{op.}}{=} SII(SII) \to_S I(SII)(I(SII)) \stackrel{2}{\to}_I SII(SII)$$

Παρ' όλα αυτά, το σύστημα της συνδυαστικής λογικής είναι συμβάλλον, επειδή ανήκει στην κατηγορία των ορθογωνίων συστημάτων (βλέπε κεφάλαιο 7), τα οποία χαρακτηρίζονται από την ιδιότητα της συμβολής.

**5.2.2 Παράδειγμα** (Εύρεση ελαχίστου, μεγίστου φυσικών αριθμών)**.** Παριστάνουμε τους φυσικούς αριθμούς με την λεγόμενη εναδική αναπαράσταση ως όρους χρησιμοποιώντας ως σταθερά το 0 που παριστάνει τον αριθμό 0 και για τον αριθμό $n$ εφαρμόζουμε $n$ φορές στην σταθερά 0 το συναρτησιακό σύμβολο (ενός ορίσματος) $s$. Για παράδειγμα, ο αριθμός 3 παριστάνεται ως $s(s(s(0)))$.

Για το ελάχιστο δύο φυσικών αριθμών, θεωρούμε το συναρτησιακό σύμβολο δύο ορισμάτων min, και έχουμε τους εξής κανόνες :

$$\min(0,x) \to 0, \quad \min(x,0) \to 0, \quad \min(s(x),s(y)) \to s(\min(x,y))$$

Ομοίως, για το μέγιστο δύο φυσικών αριθμών, έχουμε για το max :

$$\max(0,x) \to x, \quad \max(x,0) \to x, \quad \max(s(x),s(y)) \to s(\max(x,y))$$

Το παραπάνω σύστημα είναι μη αιτιοκρατικό. Για παράδειγμα, για τον όρο $\max(0,0)$ είναι δυνατόν να εφαρμοστούν δύο κανόνες (βέβαια έχουν το ίδιο δεξιό μέλος : 0). Εύκολα φαίνεται ότι το σύστημα είναι συμβάλλον, αφού τα δύο ζεύγη κανόνων

$$\min(0,x) \to 0, \quad \min(x,0) \to 0 \quad \text{και } \max(0,x) \to x, \quad \max(x,0) \to x$$

έχουν το ίδιο δεξιό μέλος. Επιπλέον, το σύστημα είναι τερματίζον. Οι κανονικές μορφές είναι οι φυσικοί αριθμοί : 0, $s(0)$, $s(s(0))$ κ.τ.λ., ενώ κάθε παράσταση της μορφής $\min(s^m(0),s^n(0))$ ή $\max(s^m(0),s^n(0))$ καταλήγει στο ελάχιστο ή μέγιστο, αντιστοίχως, των ορισμάτων.

Για παράδειγμα :

$$\max(s(0),\min(s(s(0)),s(s(s(0))))) \stackrel{*}{\to} s(s(0))$$

**5.2.3 Παράδειγμα** (Ταξινόμηση εισαγωγής)**.** Αυτό το παράδειγμα προέρχεται από τα [30] και [31]. Θεωρούμε λίστες φυσικών αριθμών τις οποίες παριστάνουμε ως εμφωλευμένα ζεύγη κατά την παράδοση των γλωσσών συναρτησιακού προγραμματισμού. Για τον επαγωγικό ορισμό των λιστών, αρχίζουμε με την κενή λίστα $\varepsilon$ και από την λίστα $y$ και τον φυσικό αριθμό $x$ κατασκευάζουμε την λίστα $x : y$, με πρώτο στοιχείο το $x$. Για παράδειγμα η λίστα :

$$[2, 1, 4, 1] \text{ παριστάνεται ως } 2 : 1 : 4 : 1 : \varepsilon,$$

όπου θεωρούμε ότι ο ενθεματικός τελεστής «:» προσεταιρίζει από τα δεξιά για να αποφύγουμε τις παρενθέσεις. [1]

Θα θεωρήσουμε ότι στο σύστημά μας συμπεριλαμβάνονται οι κανόνες για τα min, max από το παράδειγμα 5.2.2.

Η ταξινόμηση εισαγωγής λειτουργεί ως εξής : Για να ταξινομήσει μία λίστα εισάγει το πρώτο στοιχείο της λίστας στην σωστή θέση του υπολοίπου της λίστας, αφού όμως πρώτα ταξινομήσει (αναδρομικά) το υπόλοιπο της λίστας. Έχουμε λοιπόν τους κανόνες (ο τελεστής «:» αντιστοιχεί σε συναρτησιακό σύμβολο δύο ορισμάτων) :

$$\text{sort}(\varepsilon) \to \varepsilon, \quad \text{sort}(x : y) \to \text{insert}(x, \text{sort}(y))$$
$$\text{insert}(x, \varepsilon) \to x : \varepsilon, \quad \text{insert}(x, y : z) \to \min(x, y) : \text{insert}(\max(x, y), z)$$

Μπορεί να δειχθεί ότι το παραπάνω σύστημα λειτουργεί όπως θα περίμενε κανείς, δηλαδή τερματίζει πάντοτε με αποτέλεσμα μία ταξινομημένη λίστα, για οποιαδήποτε «καλώς» κατασκευασμένη είσοδο· για παράδειγμα :

$$\text{sort}(2 : 4 : 1 : 2 : \varepsilon) \xrightarrow{*} 1 : 2 : 2 : 4 : \varepsilon$$

Φυσικά, δεν έχουν νόημα «κακώς» κατασκευασμένοι όροι όπως ο $\text{sort}(s(0))$, όπου το όρισμα της sort δεν είναι λίστα. Επιπλέον το σύστημα, αν και μη αιτιοκρατικό, είναι συμβάλλον (πάλι επειδή είναι ορθογώνιο· βλέπε κεφάλαιο 7).

Για ακόμη περισσότερα ενδιαφέροντα συστήματα μεταγραφής όρων, παραπέμπουμε τον ενδιαφερόμενο αναγνώστη στο [31].

## 5.3 Αποκρίσιμες και μη ιδιότητες

Έστω ότι δίνεται ένα σύστημα μεταγραφής όρων. Μπορούμε να απαντήσουμε στο ερώτημα αν αυτό είναι τερματίζον; Λόγου χάρη, στο παράδειγμα 5.2.1 βρήκαμε έναν όρο, τον $\Omega$, που έχει μία άπειρη αλυσίδα αναγωγών. Παρ' όλα αυτά, δεν υπάρχει γενική αλγοριθμική διαδικασία που μπορεί

1. Αυτή είναι ακριβώς η αναπαράσταση των λιστών στην γλώσσα προγραμματισμού Haskell.

να αποκρίνεται για την ιδιότητα *τερματισμού* για οποιοδήποτε σύστημα μεταγραφής.

Γενικά, η μέθοδος που χρησιμοποιείται για την απόδειξη μη αποκρισιμότητας στηρίζεται στην αναγωγή των προβλημάτων που αφορούν τα συστήματα μεταγραφής σε μη αποκρίσιμα προβλήματα που αφορούν μηχανές Turing, όπως το πρόβλημα τερματισμού (halting problem). Στο [59] μπορεί κανείς να βρει μία σχετικά απλή τέτοια απόδειξη, η οποία βασίζεται στο [54]. Ακόμη και σε συστήματα με περιορισμό στο πλήθος των κανόνων, το πρόβλημα είναι μη αποκρίσιμο : Στο [28] αποδεικνύεται η μη αποκρισιμότητα για συστήματα με δύο κανόνες. Μάλιστα, για μία απόδειξη (περισσότερο περίπλοκη) ότι η ιδιότητα δεν είναι αποκρίσιμη ούτε για συστήματα με έναν μόνον κανόνα, βλέπε [23].

Φυσικά, για συγκεκριμένες υποκατηγορίες συστημάτων, η ιδιότητα του τερματισμού είναι αποκρίσιμη. Μάλιστα, στην ενότητα 5.4 θα δώσουμε μία κατηγορία συστημάτων μεταγραφής όρων, τα λεγόμενα *θεμελιώδη*, όπου η ιδιότητα του τερματισμού, καθώς και άλλες ιδιότητες είναι αποκρίσιμες.

Παρόμοια αποτελέσματα μη αποκρισιμότητας έχουμε και για την ιδιότητα της *συμβολής*.

Επίσης, εν γένει δεν είναι αποκρίσιμα τα παρακάτω προβλήματα :
- Δίνονται δύο όροι $s, t$. Είναι μετατρέψιμοι ; Δηλαδή, ισχύει $s \overset{*}{\leftrightarrow} t$ ; Είναι γνωστό και ως *λεκτικό πρόβλημα* (word problem).
- Δίνεται όρος $t$. Έχει ο όρος κανονική μορφή ; Είναι γνωστό και ως πρόβλημα απόφασης για την κανονικοποίηση.
- Δίνεται όρος $t$. Έχει ο όρος άπειρη αλυσίδα αναγωγής ;

Μάλιστα, στο σύστημα της συνδυαστικής λογικής (παράδειγμα 5.2.1), κανένα από τα παραπάνω τρία προβλήματα δεν είναι αποκρίσιμο.

## 5.4 Ιδιότητες σχετικές με κανόνες μεταγραφής

Παρακάτω δίνουμε ορισμένες ιδιότητες, συντακτικής υφής, που αφορούν τους κανόνες ενός συστήματος μεταγραφής :

**5.4.1 Ορισμός.** Ένας κανόνας $l \to r$ ονομάζεται :
- *αριστερογραμμικός* (left-linear) : όταν ο $l$ είναι γραμμικός όρος (δηλαδή κάθε μεταβλητή εμφανίζεται το πολύ μία φορά στον $l$)
- *δεξιογραμμικός* (right-linear) : όταν ο $r$ είναι γραμμικός
- *γραμμικός* (linear) : όταν οι $l$, $r$ είναι γραμμικοί
- *μη διαγράφων* (non-erasing) : όταν $\mathrm{Var}(l) = \mathrm{Var}(r)$
- *καταρρέων* (collapsing) : όταν $r \in \mathcal{V}$

– *διπλασιάζων* (duplicating) : όταν μία μεταβλητή εμφανίζεται περισσότερες φορές στον $r$ από ότι στον $l$

Αυτές οι ιδιότητες μεταφέρονται αναλόγως και στα συστήματα μεταγραφής. Για παράδειγμα, ένα σύστημα στο οποίο όλοι οι κανόνες είναι αριστερογραμμικοί, ονομάζεται επίσης αριστερογραμμικό.

Μία άλλη συντακτική ιδιότητα των κανόνων που χαρακτηρίζει ένα σύστημα μεταγραφής δίνεται παρακάτω :

**5.4.2 Ορισμός.** Ένα σύστημα μεταγραφής όρων ονομάζεται *θεμελιώδες* όταν για κάθε κανόνα, το αριστερό και το δεξιό μέλος είναι θεμελιώδεις όροι.

Τα θεμελιώδη συστήματα μεταγραφής είναι αρκετά πιο αδύναμα ως προς την υπολογιστική τους ισχύ από τα συστήματα στα οποία εμφανίζονται μεταβλητές στους κανόνες. Αυτό, όμως, έχει ως αποτέλεσμα ορισμένες ιδιότητες όπως αυτές του τερματισμού και της συμβολής να είναι αποκρίσιμες, σε αντίθεση με την γενική περίπτωση. Πιο συγκεκριμένα :

**5.4.3 Πρόταση.** *Σε ένα θεμελιώδες σύστημα μεταγραφής όρων, η ιδιότητα της συμβολής είναι αποκρίσιμη.*

*Απόδειξη.* Βλέπε [25, 24, 81]. □

**5.4.4 Πρόταση.** *Σε ένα θεμελιώδες σύστημα μεταγραφής όρων, η ιδιότητα του τερματισμού είναι αποκρίσιμη.*

*Απόδειξη.* Βλέπε [54]. □

## 5.5 Στρατηγικές μεταγραφής

Η έλλειψη αιτιοκρατίας στα συστήματα μεταγραφής όρων μπορούμε να θεωρήσουμε ότι προκύπτει αφ' ενός από την δυνατότητα να επιλέξουμε διαφορετικούς κανόνες μεταγραφής και αφ' ετέρου από την πιθανή παρουσία πολλών αναξίμων σε διαφορετικές θέσεις. Με τις στρατηγικές μεταγραφής επιδιώκεται να εξαλειφθεί η έλλειψη αιτιοκρατίας του δεύτερου είδους, δηλαδή με αυτές αντιστοιχίζεται σε κάθε όρο μία συγκεκριμένη ακολουθία αναγωγών, σε συγκεκριμένες θέσεις, σύμφωνη με την σχέση μεταγραφής. Οι δύο κυριότερες στρατηγικές προκύπτουν ανάλογα με το αν επιλέγουμε *ελαχιστικό* ή *μεγιστικό* ανάξιμο :

**5.5.1 Ορισμός** (innermost)**.** Μία στρατηγική λέγεται *εσωτερικότερη* αν επιλέγει πάντοτε ανάξιμο που δεν περιέχει εντός του άλλο ανάξιμο.

**5.5.2 Ορισμός** (outermost)**.** Μία στρατηγική λέγεται *εξωτερικότερη* αν επιλέγει πάντοτε ανάξιμο που δεν περιέχεται σε άλλο ανάξιμο.

Οι παραπάνω στρατηγικές ορίζουν διμελείς σχέσεις, οι οποίες δεν είναι σχέσεις μεταγραφής όρων σύμφωνα με τον ορισμό 5.1.2, δεδομένου ότι δεν είναι κλειστές ως προς αντικαταστάσεις η εσωτερικότερη (innermost) και περιβάλλοντα και αντικαταστάσεις η εξωτερικότερη (outermost).

Μάλιστα, οι σχέσεις μεταγραφής που ορίζονται από τις δύο στρατηγικές, εσωτερικότερη και εξωτερικότερη, συμβολίζονται ως $\xrightarrow[\mathrm{i}]{}$ και $\xrightarrow[\mathrm{o}]{}$, αντιστοίχως. Επιπλέον έχει νόημα να αναζητήσουμε ιδιότητες (ως προς τον τερματισμό, τις κανονικές μορφές, την συμβολή κ.τ.λ.) για αυτές τις σχέσεις (βλέπε [48]). Αρκετά χρήσιμες είναι, δεδομένου συστήματος $\mathcal{R}$ με σχέση μεταγραφής όρων $\rightarrow$, οι εξής ιδιότητες, : Αν η σχέση $\xrightarrow[\mathrm{i}]{}$ είναι ασθενώς κανονικοποιούσα (WN) λέμε ότι το σύστημα μεταγραφής $\mathcal{R}$ είναι WIN (weakly innermost normalizing), ενώ αν η $\xrightarrow[\mathrm{i}]{}$ είναι ισχυρώς κανονικοποιούσα (SN) λέμε ότι το σύστημα είναι SIN (strongly innermost normalizing)· αντίστοιχα από την σχέση $\xrightarrow[\mathrm{o}]{}$ έχουμε τις ιδιότητες WON και SON για το σύστημα $\mathcal{R}$. Κατά τα γνωστά, αντίστοιχες ιδιότητες ορίζονται για μεταγραφές που ξεκινούν από κάποιον όρο $t$ και συμβολίζονται WIN($t$), SIN($t$) κ.τ.λ.

Ας σημειωθεί ότι οι παραπάνω στρατηγικές δεν καταργούν εντελώς την έλλειψη αιτιοκρατίας του δεύτερου είδους, δεδομένου ότι τα ελαχιστικά ή τα μεγιστικά ανάξιμα ενδέχεται να είναι περισσότερα από ένα. Για αυτόν τον λόγο, συνήθως επιβάλλουμε να γίνεται μεταγραφή του αναξίμου (ελαχιστικού ή μεγιστικού αντιστοίχως) στην αριστερότερη θέση (δηλαδή αυτού που έχει την πρώτη θέση στην λεξικογραφική διάταξη των θέσεων· βλέπε 4.2.6, σελίδα 38). Τότε λέμε ότι έχουμε *αριστερότερη* (leftmost) στρατηγική και οι παραπάνω στρατηγικές αναφέρονται ως *αριστερότερη εσωτερικότερη* (leftmost innermost) και *αριστερότερη εξωτερικότερη* (leftmost outermost).

Η εσωτερικότερη στρατηγική έχει το πλεονέκτημα ότι εν γένει χρειάζεται λιγότερα βήματα για να φτάσει στο ίδιο αποτέλεσμα με την εξωτερικότερη, αν δεν υπάρχουν διαγράφοντες κανόνες : Διαισθητικά, έστω ότι από έναν όρο υπάρχει διπλασιασμός χρησιμοποιώντας την εξωτερικότερη στρατηγική και διπλασιάζεται ένα ανάξιμο. Αυτό σημαίνει ότι οι ίδιοι κανόνες θα εφαρμοστούν εις διπλούν στα δύο νέα αντίγραφα του ανάξιμου. Αν αντίθετα εφαρμοστεί εσωτερικότερη στρατηγική, το ανάξιμο που πρόκειται να διπλασιαστεί καταλήγει πρώτα σε κανονική μορφή, εκτελώντας μόνον μία φορά τα αντίστοιχα βήματα μεταγραφής. Αν όμως το σύστημα είναι διαγράφον (erasing), τότε ενδέχεται με την εσωτερικότερη αναγωγή να κάνουμε περισσότερα βήματα, γιατί μπορεί να εφαρμοστεί μεταγραφή σε ένα ανάξιμο που κατόπιν διαγράφεται από κανόνα που εφαρμόζεται σε εξωτερικότερη θέση. Αντίθετα, με την εξωτερικότερη μεταγραφή εφαρμόζεται πρώτα ο κανόνας που διαγράφει το εσωτερι-

κότερο ανάξιμο. Μάλιστα, η ακραία περίπτωση είναι το εσωτερικό ανάξιμο να μην έχει καν κανονική μορφή, αλλά να αρχίζει από αυτό μία άπειρη αλυσίδα αναγωγών, οπότε η εσωτερικότερη στρατηγική δεν τερματίζει ποτέ. Αυτό το φαινόμενο περιγράφεται με την έννοια της κανονικοποιούσας στρατηγικής :

**5.5.3 Ορισμός.** Μία στρατηγική ονομάζεται *κανονικοποιούσα* (normalizing) αν από έναν όρο, οδηγεί σε κανονική μορφή, αν φυσικά ο όρος έχει κανονική μορφή.

*5.5.4 Παρατήρηση.* Μία κανονικοποιούσα στρατηγική είναι χρήσιμη αν ένας όρος είναι WN (weakly normalizing), δηλαδή έχει κανονική μορφή, αλλά όχι SN (strongly normalizing), δηλαδή υπάρχει από αυτόν και μία άπειρη αλυσίδα αναγωγών. Τότε, η κανονικοποιούσα στρατηγική μας εξασφαλίζει ότι θα βρούμε την κανονική μορφή.

Εύκολα προκύπτει ότι η εξωτερικότερη στρατηγική είναι κανονικοποιούσα ενώ η εσωτερικότερη δεν είναι.

**5.5.5 Παράδειγμα.** Αυτό το παράδειγμα είναι προσαρμοσμένο από ένα αντίστοιχο παράδειγμα από το [109]. Έστω το σύστημα με τους δύο κανόνες (χρησιμοποιούμε την εναδική παράσταση αριθμών) :

$$f(0,y) \to 0 \qquad f(s(x),y) \to f(x,f(s(x),y)).$$

Για τον όρο $f(s(0),y)$, η εσωτερικότερη στρατηγική δίνει :

$$f(s(0),y) \underset{i}{\to} f(0,f(s(0),y)) \underset{i}{\to} f(0,f(0,f(s(0),y))) \underset{i}{\to} \ldots,$$

δηλαδή έχουμε μία άπειρη αλυσίδα αναγωγών, ενώ η εξωτερικότερη στρατηγική καταλήγει σε κανονική μορφή ως εξής :

$$f(s(0),y) \underset{o}{\to} f(0,f(s(0),y)) \underset{o}{\to} 0.$$

Η εσωτερικότερη στρατηγική χρησιμοποιείται συνήθως στις προστακτικές γλώσσες προγραμματισμού, όπου πριν κληθεί μία συνάρτηση υπολογίζονται πρώτα οι τιμές όλων των ορίσματων της (call by value). Σε αυτήν την περίπτωση έχουμε την λεγόμενη *πρόθυμη αποτίμηση* (eager evaluation). Αντίθετα, η εξωτερικότερη στρατηγική χρησιμοποιείται συχνά στις συναρτησιακές γλώσσες προγραμματισμού, όπου έχουμε την λεγόμενη *οκνηρή αποτίμηση* (lazy evaluation).

Είναι επίσης δυνατόν σε μία στρατηγική να γίνονται περισσότερες από μία αναγωγές ταυτόχρονα.

**5.5.6 Ορισμός.** *Παράλληλες* στρατηγικές είναι αυτές στις οποίες ανάγονται ταυτοχρόνως ανάξιμα σε περισσότερες από μία, παράλληλες μεταξύ τους, θέσεις (παράλληλη εσωτερικότερη και παράλληλη εξωτερικότερη στρατηγική).

**5.5.7 Ορισμός.** Η στρατηγική *πλήρους αντικατάστασης* είναι μία στρατηγική πολλών βημάτων στην οποία όλα τα ανάξιμα μεταγράφονται ταυτοχρόνως.

Για περισσότερες λεπτομέρειες και παραδείγματα βλέπε [59, 109].

## 5.6 Τμηματικές ιδιότητες

Μεγάλο ενδιαφέρον παρουσιάζει η τμηματική μελέτη ενός συστήματος μεταγραφής. Μπορούμε να μελετήσουμε ξεχωριστά υποσύνολα των κανόνων του συστήματος (τα λεγόμενα τμήματα ή modules) και κατόπιν να ερευνήσουμε αν οι ιδιότητες των τμημάτων συνεχίζουν να ισχύουν στο όλο σύστημα.

Αν θεωρήσουμε λοιπόν δύο συστήματα μεταγραφής, ο απλούστερος τρόπος «ένωσης» τους προκύπτει αν θεωρήσουμε μαζί τους κανόνες αυτών :

**5.6.1 Ορισμός.** Η *ένωση* δύο συστημάτων μεταγραφής $\mathcal{R}_1^{\mathcal{F}_1}$, $\mathcal{R}_2^{\mathcal{F}_2}$ ορίζεται ως εξής : $\mathcal{R}_1^{\mathcal{F}_1} \cup \mathcal{R}_2^{\mathcal{F}_2} \stackrel{\text{ορ.}}{=} (\mathcal{R}_1 \cup \mathcal{R}_2)^{\mathcal{F}_1 \cup \mathcal{F}_2}$.

Δυστυχώς, ακόμα και πολύ απλά «τμήματα» $\mathcal{R}_1$, $\mathcal{R}_2$ δεν διατηρούν, γενικά, τις ιδιότητες μετά από ένωση :

**5.6.2 Παράδειγμα.** Έστω : $\mathcal{R}_1 \colon \{a \to b\}$, $\mathcal{R}_2 \colon \{a \to c\}$. Έχουμε $\mathrm{CR}(\mathcal{R}_1)$ και $\mathrm{CR}(\mathcal{R}_2)$, αλλά προφανώς η ένωση $\mathcal{R}_1 \cup \mathcal{R}_2$ δεν είναι Church-Rosser.

Έστω $\mathcal{R}_1 \colon \{a \to b\}$, $\mathcal{R}_2 \colon \{b \to a\}$. Έχουμε $\mathrm{SN}(\mathcal{R}_1)$ και $\mathrm{SN}(\mathcal{R}_2)$, αλλά προφανώς η ένωση $\mathcal{R}_1 \cup \mathcal{R}_2$ δεν τερματίζει.

Μία ιδανική περίπτωση είναι τα δύο συστήματα να μην έχουν κοινά (συναρτησιακά) σύμβολα :

**5.6.3 Ορισμός.** Δύο συστήματα $\mathcal{R}_1^{\mathcal{F}_1}$, $\mathcal{R}_2^{\mathcal{F}_2}$ ονομάζονται *ξένα μεταξύ τους* όταν $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$.

Θα πρέπει μόνον να δώσουμε προσοχή στην περίπτωση που τα δύο συστήματα έχουν κοινά συναρτησιακά σύμβολα. Για αυτές τις περιπτώσεις έχουμε :

**5.6.4 Ορισμός.** Το *ξένο άθροισμα* (disjoint sum) δύο συστημάτων μεταγραφής $\mathcal{R}_1^{\mathcal{F}_1}$, $\mathcal{R}_2^{\mathcal{F}_2}$ συμβολίζεται με $\mathcal{R}_1 \oplus \mathcal{R}_2$ και είναι η ένωση των $\mathcal{R}_1$, $\mathcal{R}_2$ όταν $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$, δηλαδή τα συστήματα είναι ξένα μεταξύ τους· αν όμως τα συστήματα έχουν κοινά σύμβολα θεωρούμε τα συστήματα $\mathcal{R}_1'$, $\mathcal{R}_2'$ με μετονομασμένα αλφάβητα συμβόλων $\mathcal{F}_1'$, $\mathcal{F}_2'$ αντιστοίχως, τέτοια ώστε $\mathcal{F}_1' \cap \mathcal{F}_2' = \emptyset$ και θέτουμε $\mathcal{R}_1 \oplus \mathcal{R}_2 \stackrel{\text{ορ.}}{=} \mathcal{R}_1' \cup \mathcal{R}_2'$

Φυσικά, η πιθανή μετονομασία των συναρτησιακών συμβόλων δεν επηρεάζει τις ιδιότητες του προκύπτοντος συστήματος.

**5.6.5 Παράδειγμα.** Θεωρώντας τα συστήματα του παραδείγματος 5.6.2, ενώ με την ένωση δεν διατηρείται καμμία ιδιότητα, με το ξένο άθροισμα διατηρείται και η συμβολή και ο τερματισμός.

Μία διαφορετική προσέγγιση στην μελέτη των τμηματικών ιδιοτήτων ακολουθείται από τους Kurihara και Kaji [65]. Αυτοί θεωρούν ότι, δεδομένων δύο συστημάτων μεταγραφής, εκτελούμε πρώτα όλες τις μεταγραφές στα πλαίσια του ενός συστήματος, μέχρι να φτάσουμε σε κανονική μορφή για το πρώτο σύστημα, και κατόπιν μεταγραφές στο δεύτερο σύστημα, πάλι μέχρι να φτάσουμε σε κανονική μορφή. Μπορούμε να εναλλάξουμε όσες φορές θέλουμε τα δύο συστήματα μεταγραφής φτάνει να εξαντλούμε για το κάθε ένα τις πιθανές αναγωγές μέχρι κανονικής μορφής πριν να πάμε στο άλλο. Η σχέση που προκύπτει από την παραπάνω διαδικασία συμβολίζεται με $\blacktriangleright$. Μάλιστα, στο [65] αποδεικνύονται τα εξής :

**5.6.6 Θεώρημα.** *Αν τα $\mathcal{R}_1$, $\mathcal{R}_2$ είναι ξένα μεταξύ τους, τότε η σχέση $\blacktriangleright$ τερματίζει* (SN)*.*

**5.6.7 Θεώρημα.** *Αν τα $\mathcal{R}_1$, $\mathcal{R}_2$ είναι ξένα μεταξύ τους και πλήρη (δηλαδή τερματίζοντα και συμβάλλοντα), τότε η σχέση $\blacktriangleright$ είναι πλήρης* (SN)*.*

## 5.7 Κριτήρια τερματισμού

Όπως είδαμε στην ενότητα 5.3 το πρόβλημα του τερματισμού για ένα σύστημα μεταγραφής είναι εν γένει *μη* αποκρίσιμο. Παρ' όλα αυτά, δεδομένης της σημασίας του τερματισμού για την υπολογιστική έκφανση των συστημάτων μεταγραφής, είναι χρήσιμο να έχουμε αποδείξεις τερματισμού για κάποιες κατηγορίες συστημάτων. Σε αυτήν την ενότητα, θα αναφέρουμε ορισμένα κριτήρια, τεχνικές και μεθόδους για τέτοιες αποδείξεις. Η παρουσίαση ακολουθεί κυρίως το [48], αλλά και τα [59, 31].

Αρχικά, παρατηρούμε ότι ο τερματισμός ενός συστήματος μεταγραφής $\mathcal{R}^{\mathcal{F}}$ επί του $\mathcal{T}(\mathcal{F},\mathcal{V})$ είναι ισοδύναμος με τον τερματισμό του $\mathcal{R}^{\mathcal{F}}$ επί του συνόλου όλων των *θεμελιωδών* όρων, με την προϋπόθεση, φυσικά, ότι το $\mathcal{T}(\mathcal{F})$ είναι μη κενό. Επομένως, το σύστημα τερματίζει αν και μόνον αν το μεταβατικό κλείσιμο της σχέσης μεταγραφής ($\xrightarrow{+}$) είναι καλώς εδραιωμένη μερική διάταξη (βλέπε ορισμούς 3.3.3 και 1.4.5). Έτσι, μπορούμε να περιοριστούμε στην μελέτη μόνον του συνόλου των θεμελιωδών όρων ως προς τον τερματισμό.

Μία συνήθης μέθοδος για την απόδειξη του τερματισμού ενός συστήματος μεταγραφής όρων $\mathcal{R}^{\mathcal{F}}$ είναι ο ορισμός μίας μερικής διάταξης $>$ επί του $\mathcal{T}(\mathcal{F})$ με τις εξής ιδιότητες : η $>$ είναι καλώς εδραιωμένη και για κάθε ζεύγος

θεμελιωδών όρων $s$, $t$ ισχύει αν $s \to t$, τότε $s > t$. Τότε είναι προφανές ότι το εν λόγω σύστημα τερματίζει. Και αντίστροφα όμως, αν θεωρήσουμε ένα τερματίζον σύστημα, τότε το μεταβατικό κλείσιμο της σχέσης μεταγραφής ($\stackrel{+}{\to}$) έχει τις παραπάνω ιδιότητες.

Η προηγούμενη μέθοδος μπορεί να επεκταθεί στην *μέθοδο της καλώς εδραιωμένης απεικόνισης* (well-founded mapping method) [55] όπου ορίζεται μία καλώς εδραιωμένη μερική διάταξη $>_D$ επί συνόλου $D$, καθώς και μία απεικόνιση $\tau\colon \mathcal{T}(\mathcal{F}) \to D$, ούτως ώστε να έχουμε : $s > t$ αν και μόνον αν $\tau(s) >_D \tau(t)$. Στην ειδική περίπτωση που το σύνολο $D$ είναι μία $\mathcal{F}$-άλγεβρα και η απεικόνιση $\tau$ ο μοναδικός $\mathcal{F}$-ομομορφισμός από το $\mathcal{T}(\mathcal{F})$ στο $D$, έχουμε τη λεγόμενη *αύξουσα ερμηνεία* (increasing interpretation) [55, 66] ή αλλιώς *μέθοδο της μονότονης άλγεβρας* (monotone algebra method) [112]. Σε αυτήν την περίπτωση, το περιέχεσθαι της σχέσης $\to$ στην $>$ προκύπτει από την κλειστότητα της $>$ ως προς τα περιβάλλοντα και την κλειστότητα της σχέσης του περιέχεσθαι για τους κανόνες ως προς τις θεμελιώδεις αντικαταστάσεις :

$$\forall\, l \to r \in \mathcal{R}^{\mathcal{F}}\colon \forall \sigma\colon l^{\sigma} > r^{\sigma},$$

όπου $\sigma$ : $\mathcal{T}(\mathcal{F})$-θεμελιώδης αντικατάσταση.

Μία ακόμη πιο ειδική περίπτωση είναι να θεωρήσουμε ότι αυτή η $\mathcal{F}$-άλγεβρα είναι ακριβώς η άλγεβρα όρων $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Τότε έχουμε τους παρακάτω ορισμούς :

**5.7.1 Ορισμός.** Μία *διάταξη μεταγραφής* (rewrite ordering) είναι μία μερική διάταξη επί του $\mathcal{T}(\mathcal{F}, \mathcal{V})$ που είναι σχέση μεταγραφής (βλέπε ορισμό 5.1.2).

**5.7.2 Ορισμός.** Μία *διάταξη αναγωγής* (reduction ordering) είναι μία καλώς εδραιωμένη διάταξη μεταγραφής.

Η διάταξη αναγωγής είναι χρήσιμη επειδή ένα σύστημα μεταγραφής είναι τερματίζον αν και μόνον αν το μεταβατικό κλείσιμο της σχέσης μεταγραφής του περιέχεται σε κάποια διάταξη αναγωγής. Συνήθως, σε μία απόδειξη τερματισμού με τον παραπάνω τρόπο, για μία υποψήφια διάταξη αναγωγής είναι δύσκολο να δείξουμε το καλώς εδραιωμένο αυτής. Ο Dershowitz στο [27] εισήγαγε την πολλή χρήσιμη έννοια των *διατάξεων απλοποίησης* (simplification orderings), οι οποίες έχουν εκ κατασκευής το καλώς εδραιωμένο. Το αποτέλεσμα αυτό βασίζεται στο *θεώρημα δένδρων του Kruskal* (Kruskal tree theorem) [62], που αποτελεί γενίκευση του λήμματος του Higman [49], (για μια απλούστερη μη κατασκευαστική απόδειξη του θεωρήματος βλέπε [75], η οποία αναπαράγεται στα [30, 80]).

Αναφέρουμε εδώ επιγραμματικά μερικές τεχνικές για την απόδειξη τερματισμού : bag ordering, lexicographic path ordering, multiset path ordering, general path ordering, καθώς και σχετική βιβλιογραφία : [30, 31, 97, 29, 46].

## 5.8 Άλλα συστήματα μεταγραφής

Μία σημαντική υποκατηγορία των συστημάτων μεταγραφής όρων που έχει μελετηθεί από μόνη της πολύ νωρίτερα είναι τα συστήματα μεταγραφής συμβολοσειρών (semi-Thue systems) :[2]

**5.8.1 Ορισμός.** Ένα *σύστημα μεταγραφής συμβολοσειρών* (string rewriting system ή srs για συντομία) είναι η πλειάδα $(\Sigma, R)$, όπου $\Sigma$ : αλφάβητο απλών συμβόλων και $R \subseteq \Sigma^* \times \Sigma^*$ : το σύνολο κανόνων, ενώ ο κάθε κανόνας $(l, r)$ συμβολίζεται με $l \to r$.

Η σχέση μεταγραφής ορίζεται ως εξής στα συστήματα semi-Thue : Μία συμβολοσειρά μεταγράφεται σε μία άλλη, αν η αρχική συμβολοσειρά περιέχει ως υποσυμβολοσειρά κάτι που εμφανίζεται σε αριστερό μέλος κανόνα του συστήματος, έστω $l \to r$, και αντικαταστήσουμε στην τελική συμβολοσειρά το $l$ με το $r$.

Η διαφορά των συστημάτων Thue από τα συστήματα semi-Thue είναι απλώς η έλλειψη προσανατολισμού των κανόνων : Ένας κανόνας μπορεί να εφαρμοστεί και προς τις δύο κατευθύνσεις, δηλαδή οποιοδήποτε μέλος μπορεί να αντικαταστήσει το άλλο σε μία συμβολοσειρά. Τα συστήματα Thue, που ονομάστηκαν έτσι προς τιμήν του Νορβηγού μαθηματικού Axel Thue (1863–1922), αποτελούν την πρώτη γνωστή έννοια συστήματος μεταγραφής [101].

Για περισσότερα είδη συστημάτων μεταγραφής (για παράδειγμα, συστήματα μεταγραφής υψηλότερης τάξης), βλέπε [59, 30, 31], καθώς και για τις σχέσεις μεταξύ τους, βλέπε το σχήμα στο [103, σελίδα vii].

2. Για μία άμεση μετατροπή των συμβολοσειρών σε όρους, βλέπε παρατήρηση 8.1.3 στην σελίδα 80.

# Κεφάλαιο 6

# Κρίσιμα ζεύγη και συμπλήρωση

Σε αυτό το κεφάλαιο θα δούμε πώς μπορούμε από ένα σύνολο εξισώσεων, σε ορισμένες περιπτώσεις, να κατασκευάσουμε ένα σύστημα μεταγραφής όρων που θα είναι συμβάλλον και τερματίζον (δηλαδή κανονικό) και θα είναι *ισοδύναμο* με το σύστημα εξισώσεων. Ατύπως, μπορούμε να δούμε ένα σύστημα εξισώσεων απλώς ως ένα σύστημα μεταγραφής όρων χωρίς προσανατολισμό στους κανόνες. Ως ισοδυναμία ορίζουμε από τους κανόνες των δύο συστημάτων να προκύπτουν τα ίδια λογικά επακόλουθα. Η σημασία της εύρεσης ενός ισοδύναμου κανονικού συστήματος έγκειται, ως γνωστόν, στην δυνατότητα να αποφανθούμε για την θεωρία που προκύπτει από τις αντίστοιχες εξισώσεις (equational theory) του συστήματος εξισώσεων, υπολογίζοντας απλώς κανονικές μορφές στο κανονικό σύστημα μεταγραφής όρων. Η μέθοδος που χρησιμοποιείται για την κατασκευή ενός κανονικού συστήματος ονομάζεται *συμπλήρωση* (completion) [60].

Πρώτα όμως, θα πρέπει να ορίσουμε την δύστροπη μεν, σημαντική δε έννοια του κρισίμου ζεύγους και να δούμε πώς αυτή σχετίζεται με την έννοια της τοπικής συμβολής (και κατ' επέκτασιν, υπό ορισμένες προϋποθέσεις, γενικά με την έννοια της συμβολής).

## 6.1 Κρίσιμα ζεύγη (critical pairs)

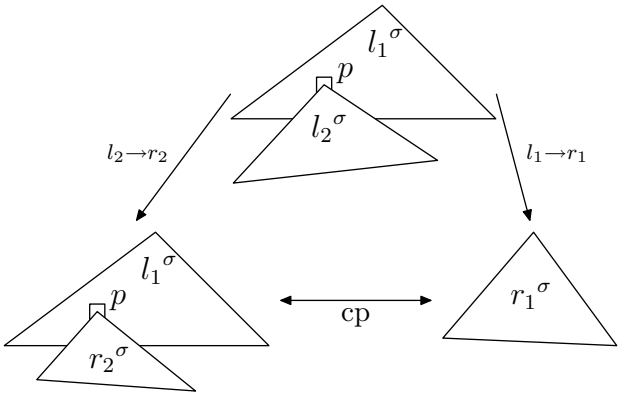
Μία εξαιρετικά σημαντική έννοια για την μελέτη ιδιοτήτων συμβολής σε συστήματα μεταγραφής όρων είναι αυτή του κρισίμου ζεύγους. Ο ορισμός είναι αρκετά περίπλοκος, αλλά αξίζει τον κόπο αφού η έννοια έχει σημαντικές εφαρμογές.

**6.1.1 Ορισμός.** Έστω ότι δίνεται σύστημα μεταγραφής όρων και ισχύουν τα παρακάτω :

- Έστω κανόνες : $l_1 \to r_1$, $l_2 \to r_2$ του συστήματος. Υποθέτουμε ότι δεν έχουν κοινές μεταβλητές (αυτό μπορεί να γίνει πάντοτε, δεδομένης της δυνατότητας μετονομασίας, όπως προκύπτει από την παρατήρηση 5.1.5 στην σελίδα 49). Επίσης, ενδέχεται να πρόκειται για τον ίδιο κανόνα (οπότε εφαρμόζουμε οπωσδήποτε την μετονομασία).
- Υπάρχει θέση $p \in \mathrm{Pos}(l_1)$, για την οποία $l_1|_p \notin \mathcal{V}$, ή ισοδύναμα υπάρχει θέση $p \in \mathcal{F}\mathrm{Pos}(l_1)$. Επιπλέον αν οι $l_1 \to r_1$, $l_2 \to r_2$ είναι ο ίδιος κανόνας, θα πρέπει $p \neq \varepsilon$.
- Οι $l_1|_p$, $l_2$ έχουν γενικότερο ενοποιητή (mgu) $\sigma$.

Τότε το $(l_1[r_2]_p{}^\sigma, r_1{}^\sigma)$ ονομάζεται *κρίσιμο ζεύγος* (critical pair).

Το παραπάνω σημαίνει ότι από τον όρο $l_1{}^\sigma$, επειδή αυτός ισούται με τον $l_1[l_2]_p{}^\sigma$ μπορεί να προκύψει ένας εκ των $l_1[r_2]_p{}^\sigma$, $r_1{}^\sigma$, ανάλογα με το αν εφαρμοστεί ο κανόνας $l_2 \to r_2$ ή ο $l_1 \to r_1$ αντιστοίχως. Λέμε ότι το κρίσιμο ζεύγος προκύπτει από *υπέρθεση* του $l_2 \to r_2$ στον $l_1 \to r_1$ (βλέπε και σχήμα 6.1) ή ότι τα ανάξιμα $l_1{}^\sigma$, $l_2{}^\sigma$ *επικαλύπτονται κρισίμως* (critically overlap).




Σχήμα 6.1 – Κρίσιμο ζεύγος

*6.1.2 Παρατήρηση.* Ο παραπάνω ορισμός δεν είναι συμμετρικός, δηλαδή αν το $(s, t)$ είναι κρίσιμο ζεύγος δεν είναι απαραίτητα και το $(t, s)$.

**6.1.3 Ορισμός.** Αν για τους όρους $s$, $u$, $t$ ισχύει $s \leftarrow u \to t$, η $s \leftarrow u \to t$ ονομάζεται *κορυφή*. Αν επιπλέον τα ανάξιμα των μεταγραφών $u \to s$, $u \to t$ αποτελούν κρίσιμο ζεύγος, τότε ονομάζεται *κρίσιμη κορυφή*. Μία κορυφή λέγεται *ζεύξιμη* αν $s \downarrow t$.

**6.1.4 Ορισμός.** Το σύνολο των κρισίμων ζευγών ενός συστήματος μεταγραφής $\mathcal{R}$ συμβολίζεται με $\mathrm{cp}(\mathcal{R})$ και προφανώς συνιστά διμελή σχέση. Το συμμετρικό κλείσιμο αυτής της σχέσης συμβολίζουμε με $\underset{\mathrm{cp}(\mathcal{R})}{\leftrightarrow}$ ή απλώς $\underset{\mathrm{cp}}{\leftrightarrow}$.

Πολλές φορές θεωρούμε το κλείσιμο της $\underset{\text{cp}}{\leftrightarrow}$ ως προς τα περιβάλλοντα και τις αντικαταστάσεις (για το οποίο χρησιμοποιούμε τον ίδιο συμβολισμό : $\underset{\text{cp}}{\leftrightarrow}$). Με την νέα αυτήν σχέση συνδέονται όλοι οι όροι που μπορεί να προκύψουν από μία κρίσιμη κορυφή.

## 6.2 Κρίσιμα ζεύγη και τοπική συμβολή

Σε αυτήν την ενότητα θα διερευνήσουμε την σχέση της τοπικής συμβολής με την έννοια των κρισίμων ζευγών όπως ορίστηκε παραπάνω. Μία χρήσιμη ιδιότητα είναι η εξής :

**6.2.1 Ορισμός.** Ένα σύστημα μεταγραφής έχει την ιδιότητα *ζευξιμότητας των κρισίμων ζευγών* (joinability of critical pairs ή σύντομα JCP), αν για κάθε κρίσιμο ζεύγος $(s, t)$, ισχύει $s \downarrow t$ (ή με σχέσεις $\underset{\text{cp}}{\leftrightarrow} \subseteq \downarrow$).

Η παραπάνω ιδιότητα, που σχετίζεται με τα κρίσιμα ζεύγη, δικαιολογεί την εισαγωγή της έννοιας των κρισίμων ζευγών, αφού συνεπάγεται αυτήν της τοπικής συμβολής, όπως αποδεικνύεται στο παρακάτω σημαντικό θεώρημα [60] :

**6.2.2 Θεώρημα** (Λήμμα κρισίμων ζευγών, critical pair lemma)**.** *Σε ένα σύστημα μεταγραφής όρων οι ιδιότητες της τοπικής συμβολής και της ζευξιμότητας κρισίμων ζευγών είναι ισοδύναμες* (WCR $\iff$ JCP).

*Απόδειξη.* (Βλέπε και [16].)

Η συνεπαγωγή WCR $\implies$ JCP είναι προφανής, αφού $\underset{\text{cp}}{\leftrightarrow} \subseteq \leftarrow \circ \rightarrow$.

Για την κατεύθυνση JCP $\implies$ WCR θα πρέπει να ερευνήσουμε όλων των ειδών τις κορυφές που μπορεί να παρουσιαστούν και αν αυτές είναι ζεύξιμες.

Ας υποθέσουμε λοιπόν ότι έχουμε την κορυφή $s \leftarrow u \rightarrow t$ και ότι οι $t$, $s$ προκύπτουν από τον $u$ με συστολή αναξίμων που βρίσκονται στις θέσεις $p_1$, $p_2$ του όρου $u$ και μέσω των κανόνων $l_1 \rightarrow r_1$, $l_2 \rightarrow r_2$ αντιστοίχως. Συμβολικά :

$$u \rightarrow_{p_1, l_1 \rightarrow r_1} t, \quad u \rightarrow_{p_2, l_2 \rightarrow r_2} s.$$

Αν $p_1 \parallel p_2$, δηλαδή τα ανάξιμα βρίσκονται σε *παράλληλες* μεταξύ τους θέσεις, έχουμε αμέσως :

$$t \rightarrow_{p_2, l_2 \rightarrow r_2} u', \quad s \rightarrow_{p_1, l_1 \rightarrow r_1} u',$$

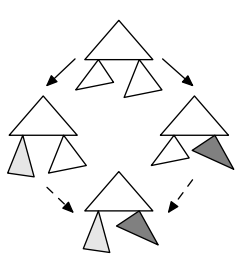
όπου ο $u'$ συγκεκριμένος όρος (βλέπε σχήμα 6.2α).

Στις υπόλοιπες περιπτώσεις, οι κανόνες επικαλύπτονται.

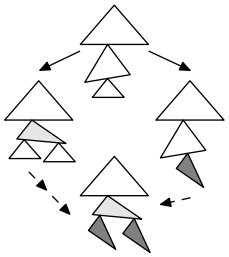
Αν δεν επικαλύπτονται κρισίμως, τότε έχουμε επικάλυψη σε θέση μεταβλητής, δηλαδή αν η $p_2$ δεν είναι πάνω από την $p_1$ θα ισχύει $p_2 \in \mathcal{V}\text{Pos}(u)$. Τότε έχουμε τα λεγόμενα *εμφωλευμένα ανάξιμα* (nested redexes). Η εφαρμογή

οποιουδήποτε από τους δύο κανόνες δεν αποκλείει πιθανώς αργότερα την εφαρμογή του άλλου, οπότε πάλι καταλήγουμε στον ίδιο όρο. Απλά επειδή ενδέχεται ο κανόνας να πολλαπλασιάζει τις μεταβλητές, ίσως χρειαστεί να εφαρμόσουμε παραπάνω από μία φορές τον άλλο κανόνα (βλέπε για παράδειγμα σχήμα 6.2β). Πρέπει μόνον να δώσουμε προσοχή στην περίπτωση που ο αριστερός όρος ενός κανόνα δεν είναι γραμμικός ως προς την μεταβλητή που επικαλύπτεται στον άλλον κανόνα. Πάλι, όμως δεν χάνεται η δυνατότητα να εφαρμόσουμε αργότερα τον άλλο κανόνα (βλέπε για παράδειγμα σχήμα 6.2γ) και να καταλήξουμε τελικά στον ίδιο όρο.

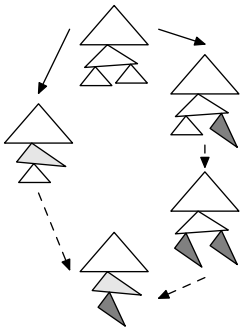
Η άλλη περίπτωση είναι να έχουμε κρίσιμη επικάλυψη κανόνων, οπότε έχουμε και *κρίσιμη κορυφή*. Αν όμως έχουμε κρίσιμη κορυφή, αυτή έχει προκύψει από κρίσιμο ζεύγος, οπότε λόγω της ιδιότητας JCP, $s \downarrow t$ (βλέπε σχήμα 6.2δ). □



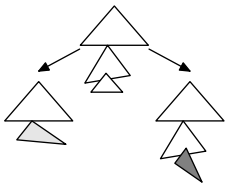
(α) παράλληλα ανάξιμα



(β) επικάλυψη μεταβλητής



(γ) επανάληψη στο αριστερό μέρος



(δ) κρίσιμη κορυφή

Σχήμα 6.2 – Λήμμα κρισίμων ζευγών

Από την παραπάνω απόδειξη είναι εμφανές ότι το λήμμα κρισίμων ζευγών μπορεί να διατυπωθεί και ως εξής :

**6.2.3 Λήμμα.** *Για κάθε κορυφή* $s \leftarrow u \rightarrow t$ *ισχύει* $s \downarrow t$ *ή* $s \underset{\text{cp}}{\leftrightarrow} t$.

Από το λήμμα κρισίμων ζευγών προκύπτει άμεσα και το εξής : Δεδομένου ότι τα πεπερασμένα συστήματα μεταγραφής όρων έχουν πεπερασμένο πλήθος κρισίμων ζευγών, η ιδιότητα της συμβολής είναι αποκρίσιμη για δεδομένο τερματίζον πεπερασμένο σύστημα. Αυτό το κριτήριο συμβολής ονομάζεται *έλεγχος υπέρθεσης* (superposition test) [60].

## 6.3 Συστήματα εξισώσεων

Η παρουσίαση σε αυτήν την ενότητα ακολουθεί αρκετά το [59] και λιγότερο τα [30, 31].

Ατύπως, ένα *σύστημα εξισώσεων* είναι ένα σύστημα μεταγραφής όρων χωρίς προσανατολισμό στους κανόνες. Περισσότερο τυπικά, ένα σύστημα εξισώσεων $E$ αποτελείται από ένα αλφάβητο συναρτησιακών συμβόλων $\mathcal{F}$ και από ένα σύνολο εξισώσεων της μορφής $l = r$ όπου $l$, $r$ : όροι επί του $\mathcal{F}$, όχι απαραίτητα θεμελιώδεις.

**6.3.1 Παράδειγμα.** Δίνουμε ένα σύστημα εξισώσεων που περιγράφει ένα μέρος της θεωρίας ομάδων :

$$x \cdot e = x, \quad e \cdot x = x, \quad i(x) \cdot (x \cdot y) = y,$$

όπου $e$ : σταθερά (ουδέτερο στοιχείο), $i()$ : συνάρτηση (αντιστροφής), «$\cdot$» : ενθεματικός τελεστής, δηλαδή συνάρτηση δύο ορισμάτων (η πράξη της ομάδας).

Οι αρχικές εξισώσεις του συστήματος ονομάζονται και αξιώματα. Αυτό που μας ενδιαφέρει είναι να αποδεικνύουμε και άλλες εξισώσεις που συνεπάγονται λογικά από τα αξιώματα. Τυπικά, χρησιμοποιούμε τους παρακάτω κανόνες για να αποδεικνύουμε νέες εξισώσεις :

- $s =_E t$, αν $s = t \in E$
- $t =_E t$, για κάθε όρο $t$ (ανακλαστικότητα)
- $\dfrac{s =_E t}{t =_E s}$ (συμμετρικότητα)
- $\dfrac{t_1 =_E t_2, \quad t_2 =_E t_3}{t_1 =_E t_3}$ (μεταβατικότητα)
- $\dfrac{s =_E t}{s^\sigma =_E t^\sigma}$, για κάθε αντικατάσταση $\sigma$
- $\dfrac{s_1 =_E t_1, \quad \ldots, s_n =_E t_n}{f(s_1, \ldots, s_n) =_E f(t_1, \ldots, t_n)}$, για κάθε $n \in \mathbb{N}$, $f \in \mathcal{F}_n$

Αν η εξίσωση $s = t$ αποδεικνύεται σύμφωνα με τους παραπάνω κανόνες από τα αξιώματα της $E$, τότε γράφουμε $E \vdash s = t$ ή $s =_E t$.

Μία εναλλακτική (αλλά τελικά ισοδύναμη) περιγραφή των παραπάνω κανόνων μπορεί να γίνει ως εξής. Θεωρούμε ως αξιώματα τις εξισώσεις του συστήματος $E$ και επιπλέον για κάθε όρο $t$ το :

– $t = t$ (ανακλαστικότητα)

και τους κανόνες :

– $\dfrac{t_1 = t_2}{t_2 = t_1}$ (συμμετρία)

– $\dfrac{t_1 = t_2, \quad t_2 = t_3}{t_1 = t_3}$ (μεταβατικότητα)

– $\dfrac{t_1 = t_2}{t_1^{\{x:=t\}} = t_2^{\{x:=t\}}}$ (αντικατάσταση 1)

– $\dfrac{t_1 = t_2}{t^{\{x:=t_1\}} = t^{\{x:=t_2\}}}$ (αντικατάσταση 2)

Οι δύο τελευταίοι κανόνες για την αντικατάσταση μπορούν να συμπτυχθούν σε έναν γενικότερο :

– $\dfrac{t_1 = t_2, \quad t = t'}{t_1^{\{x:=t\}} = t_2^{\{x:=t'\}}}$ (αντικατάσταση)

**6.3.2 Παράδειγμα.** Με την βοήθεια των παραπάνω κανόνων μπορούμε να αποδείξουμε με βάση το σύστημα εξισώσεων του παραδείγματος 6.3.1 την εξίσωση $i(i(x)) = x$.

Μία $\mathcal{F}$-*άλγεβρα* $\mathcal{A}$ είναι ένα σύνολο $A$ με συναρτήσεις $f^{\mathcal{A}} \colon A^n \to A$ για κάθε $f \in \mathcal{F}_n$ και για κάθε $n \in \mathbb{N}$ (η κάθε σταθερά $c$ αντιστοιχίζεται σε $c^{\mathcal{A}} \in A$). Έστω τώρα ότι δίνεται μία εξίσωση $s = t$ όρων επί του $\mathcal{F}$. Αν η καθολικά ποσοτικοποιημένη πρόταση που αντιστοιχεί στην $s = t$ είναι αληθής στην τυχαία $\mathcal{F}$-άλγεβρα $\mathcal{A}$ λέμε ότι η $s = t$ είναι *έγκυρη* στην $\mathcal{A}$. Αν κάθε εξίσωση ενός συστήματος εξισώσεων $E$ είναι έγκυρη σε μία $\mathcal{F}$-άλγεβρα $\mathcal{A}$, λέμε ότι η $\mathcal{A}$ είναι *μοντέλο* του $E$. Γράφουμε $E \vdash s = t$ αν σε κάθε μοντέλο του $E$ η εξίσωση $s = t$ είναι έγκυρη.

Ένα σημαντικό αποτέλεσμα πληρότητας (που συνδέει την εγκυρότητα με την αποδειξιμότητα βάσει των κανόνων) είναι το εξής :

**6.3.3 Θεώρημα** (Birkhoff [5])**.** *Έστω σύστημα εξισώσεων $E$ επί του αλφαβήτου $\mathcal{F}$ και όροι $s$, $t$ επί του $\mathcal{F}$. Τότε :*

$$E \models s = t \iff E \vdash s = t.$$

Ένα πολύ σημαντικό πρόβλημα που τίθεται σε αυτά τα συστήματα είναι το *πρόβλημα εγκυρότητας* (validity problem ή uniform word problem) :

> Δίνεται εξίσωση $s = t$ και σύστημα εξισώσεων $E$ επί του αλφαβήτου $\mathcal{F}$. Ισχύει $E \models s = t$ ή όχι ;

Από το θεώρημα του Birkhoff το πρόβλημα ανάγεται στον έλεγχο αν $E \vdash s = t$ ή όχι. Εδώ φαίνεται η χρησιμότητα των κανονικών συστημάτων μεταγραφής όρων. Αν μπορούμε να βρούμε ένα κανονικό σύστημα $\mathcal{R}$, τέτοιο ώστε :

$$s \overset{*}{\leftrightarrow}_{\mathcal{R}} t \iff E \vdash s = t,$$

τότε, αν και το σύστημα μεταγραφής έχει πεπερασμένο πλήθος κανόνων, μπορούμε να κατασκευάσουμε και αλγόριθμο απόκρισης για το πρόβλημα εγκυρότητας ως εξής : Αν δίνεται η εξίσωση $s = t$, τότε υπολογίζουμε την κανονική μορφή για τα $s$, $t$, έστω $s'$, $t'$ αντιστοίχως και αν $s' \equiv t'$, απαντάμε θετικά, αλλιώς αρνητικά.

Στην επόμενη ενότητα, θα αναφερθούμε σε μία μέθοδο για την κατασκευή ενός ισοδύναμου κανονικού συστήματος μεταγραφής όρων από ένα σύνολο εξισώσεων.

## 6.4 Συμπλήρωση (Completion)

Μία ειδική περίπτωση του λήμματος κρισίμων ζευγών (θεώρημα 6.2.2), η οποία ισχύει λόγω του λήμματος του Newman (θεώρημα 3.5.2 στην σελίδα 30) και αφορά τερματίζοντα συστήματα, είναι η εξής :

**6.4.1 Λήμμα.** *Σε ένα τερματίζον σύστημα μεταγραφής όρων, οι ιδιότητες της συμβολής και της ζευξιμότητας κρισίμων ζευγών είναι ισοδύναμες, δηλαδή :* $\mathrm{SN} \implies (\mathrm{CR} \iff \mathrm{JCP})$.

Βασιζόμενοι σε ιδέες του Evans [37], οι Knuth και Bendix [60] απέδειξαν το λήμμα κρισίμων ζευγών και κατόπιν εισήγαγαν την αλγοριθμική μέθοδο της *συμπλήρωσης* (completion) για την εύρεση ενός ισοδύναμου κανονικού συστήματος (μάλιστα έδωσαν και πρόγραμμα Fortran). Η βασική ιδέα της συμπλήρωσης Knuth-Bendix είναι η εξής : Για κάθε κρίσιμο ζεύγος $(s, t)$ που δεν είναι ζεύξιμο, προσθέτουμε έναν από τους κανόνες $s \to t$, $t \to s$ στο σύστημα. Βέβαια, η προσθήκη νέων κανόνων στο σύστημα μπορεί να δημιουργήσει νέα κρίσιμα ζεύγη, τα οποία πρέπει επίσης να αντιμετωπιστούν. Πάντως, σε ορισμένες περιπτώσεις αυτή η διαδικασία καταλήγει κάποια στιγμή σε ένα συμβάλλον σύστημα.

Τα τελευταία χρόνια, η συμπλήρωση τοποθετήθηκε σε ένα πιο αφηρημένο πλαίσιο (βλέπε [2] και παρουσίαση στα [30, 59]). Παρακάτω περιγράφουμε περιληπτικά αυτήν την προσέγγιση και παραπέμπουμε για περισσότερες λεπτομέρειες τον ενδιαφερόμενο αναγνώστη στα [30, 31], αλλά και στο [59].

Ο αλγόριθμος συμπλήρωσης ενεργεί επί ενός ζεύγους $(E, R)$. Το $E$ είναι ένα σύνολο εξισώσεων ενός συστήματος εξισώσεων και το $R$ είναι ένα

σύνολο κανόνων ενός συστήματος μεταγραφής. Και τα δύο συστήματα, φυσικά, ορίζονται επί του ιδίου αλφαβήτου $\mathcal{F}$. Στόχος μας είναι ο αλγόριθμος να δέχεται ως είσοδο το ζεύγος $E, \emptyset$ και να δίνει ως έξοδο $\emptyset, R$ και οι κανόνες $R$ περιγράφουν ένα κανονικό σύστημα μεταγραφής όρων. Προκειμένου να επιλεγεί ο προσανατολισμός ενός κρισίμου ζεύγους απαιτείται επίσης μία διάταξη αναγωγής (βλέπε ορισμό 5.7.2 στην σελίδα 58). Για περισσότερες λεπτομέρειες σχετικά με τα βήματα : deduce, orient, delete, simplify, collapse του αλγορίθμου, βλέπε [31]. Φυσικά, ο αλγόριθμος δεν είναι απαραίτητο να τερματίζει πάντοτε : Ενδέχεται να δημιουργούνται συνεχώς νέα κρίσιμα ζεύγη και να μην καταλήγουμε ποτέ σε κάποιο κανονικό σύστημα μεταγραφής. Κατά καιρούς έχουν προταθεί διάφορες επεκτάσεις της συμπλήρωσης που λειτουργούν επιτυχώς σε περισσότερες κλάσεις συστημάτων εξισώσεων (βλέπε, για παράδειγμα, [84]).

**6.4.2 Παράδειγμα.** Η μέθοδος της συμπλήρωσης εφαρμοζόμενη στο παράδειγμα 6.3.1 δίνει το ακόλουθο κανονικό σύστημα μεταγραφής :

$$e \cdot x \to x, \quad x \cdot e \to x, \quad i(x) \cdot x \to e, \quad x \cdot i(x) \to e,$$
$$i(e) \to e, \quad i(i(x)) \to x, \quad i(x) \cdot (x \cdot y) \to y, \quad x \cdot (i(x) \cdot y) \to y.$$

Για λεπτομέρειες σχετικά με το πώς προκύπτει το σύστημα από τον αλγόριθμο συμπλήρωσης, βλέπε [30]. Για ένα άλλο παράδειγμα, βλέπε [59].

# Κεφάλαιο 7

# Ορθογώνια συστήματα μεταγραφής όρων

Ένα σημαντικό υποσύνολο των συστημάτων μεταγραφής όρων είναι τα *ορθογώνια συστήματα μεταγραφής όρων*. Σε αυτό το είδος των συστημάτων απαιτούμε επιπλέον δομή : θέλουμε οι κανόνες να είναι αριστερογραμμικοί και να μην παρουσιάζονται κρίσιμα ζεύγη.

Τα ορθογώνια συστήματα αποτελούν μία υποκατηγορία συστημάτων μεταγραφής με χρήσιμες ιδιότητες, με κυριότερη μεταξύ αυτών αυτήν της *συμβολής*. Για αυτόν τον λόγο έχουν τύχει αρκετής μελέτης και έχουν προκύψει σημαντικά αποτελέσματα. Εξάλλου, πολλά χρήσιμα συστήματα μεταγραφής όρων, όπως η συνδυαστική λογική, είναι ορθογώνια.

Η παρουσίαση αυτού του κεφαλαίου βασίζεται κυρίως στο [59].

## 7.1 Ορθογώνια συστήματα

Προτού ορίσουμε τα ορθογώνια συστήματα, θα δώσουμε έναν ορισμό για την απουσία κρισίμων ζευγών σε ένα σύστημα μεταγραφής όρων :

**7.1.1 Ορισμός.** Ένα σύστημα μεταγραφής όρων ονομάζεται *μη επικαλύπτον* (non overlapping) αν δεν έχει κρίσιμα ζεύγη (βλέπε ορισμό 6.1.1, σελίδα 61).

Τελικά, έχουμε :

**7.1.2 Ορισμός.** Ένα σύστημα μεταγραφής όρων ονομάζεται *ορθογώνιο* (orthogonal) αν είναι αριστερογραμμικό (βλέπε ορισμό 5.4.1, σελίδα 52) και μη επικαλύπτον.

Μπορεί επίσης να δοθεί και μία ασθενέστερη εκδοχή της ορθογωνιότητας :

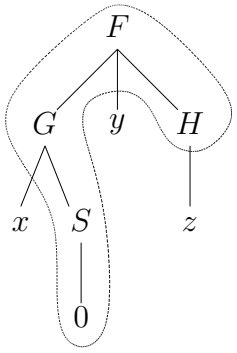
**7.1.3 Ορισμός.** Ένα σύστημα μεταγραφής όρων ονομάζεται *ασθενώς ορθογώνιο* (weakly orthogonal) αν είναι αριστερογραμμικό και τα κρίσιμα ζεύγη που περιέχει είναι *τετριμμένα*, δηλαδή αν $(s, t)$ κρίσιμο ζεύγος, τότε $s \equiv t$.

Ο ορισμός των ορθογωνίων συστημάτων βασίζεται στην έννοια του κρισίμου ζεύγους, η οποία είναι κάπως περίπλοκη. Όμως, όπως αναφέρεται στο [59], η ιδιότητα της απουσίας κρισίμων ζευγών είναι πιο εύκολο να περιγραφεί. Εδώ αναπαράγουμε το παράδειγμα από το [59].

**7.1.4 Παράδειγμα** ([59])**.** Έστω το σύστημα $\mathcal{R}$ με κανόνες $r_1$, $r_2$, $r_3$ :

$$\begin{aligned} r_1\colon &\quad F(G(x, S(0)), y, H(z)) &&\to x \\ r_2\colon &\quad G(x, S(S(0))) &&\to 0 \\ r_3\colon &\quad P(G(x, S(0))) &&\to S(0) \end{aligned}$$

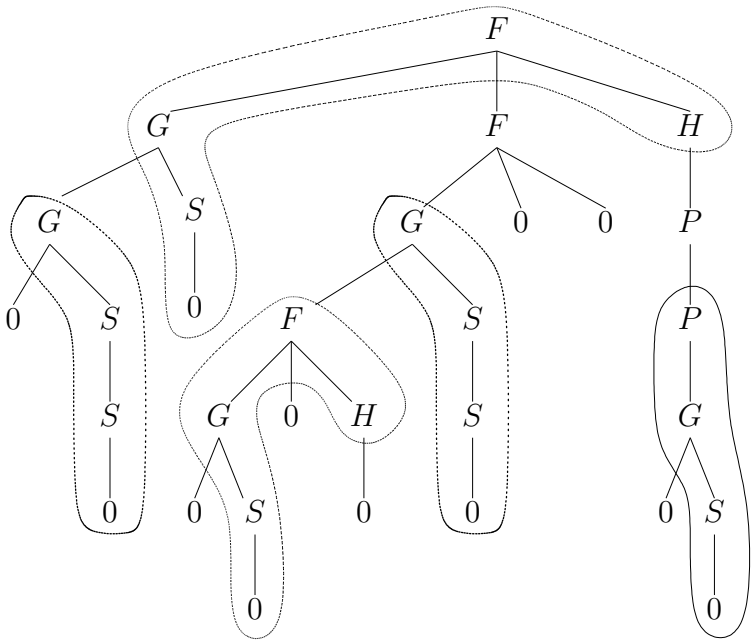
το οποίο είναι ορθογώνιο. Σε ένα ορθογώνιο σύστημα κάθε κανόνας μεταγραφής είναι αριστερογραμμικός, οπότε δεν υπάρχει συσχέτιση μεταξύ των θέσεων του όρου στο αριστερό μέλος του κανόνα οι οποίες αντιστοιχούν σε μεταβλητές. Έτσι, το αριστερό μέλος του κανόνα μπορεί να περιγραφεί από το λεγόμενο *πρότυπο* (pattern) του κανόνα. Στο παράδειγμά μας, το πρότυπο του κανόνα $r_1$ φαίνεται στο σχήμα 7.1 να περικλείεται στην εστιγμένη καμπύλη.




Σχήμα 7.1 – Πρότυπο κανόνα $r_1$

Επιπλέον, αφού το σύστημα είναι ορθογώνιο, δεν έχουμε επικαλύψεις και επομένως σε κάθε όρο τα πρότυπα των κανόνων που μπορούν να εφαρμοστούν δεν επικαλύπτονται. Αυτό φαίνεται για τον όρο :

$$\begin{aligned} F(&G(G(0, S(S(0))), S(0)), \\ &F(G(F(G(0, S(0)), 0, H(0)), S(S(0))), 0, 0), \\ &H(P(P(G(0, S(0)))))) \end{aligned}$$



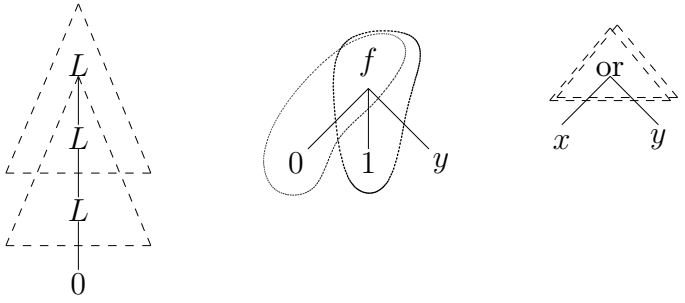

Σχήμα 7.2 – Μη επικαλυπτόμενα πρότυπα πάνω σε όρο

για το σύστημα του παραδείγματός μας στο σχήμα 7.2.

Αντιθέτως, σε ένα μη ορθογώνιο σύστημα μπορεί να έχουμε επικαλύψεις όπως φαίνεται, για παράδειγμα, για τις τρεις περιπτώσεις κανόνων :

1. $L(L(x)) \to 0$
2. $f(0, x, y) \to 0$ και $f(x, 1, y) \to 1$
3. $\mathrm{or}(x, y) \to x$ και $\mathrm{or}(x, y) \to y$

και για όρους όπως στο σχήμα 7.3.




Σχήμα 7.3 – Επικαλυπτόμενα πρότυπα πάνω σε όρους

Πρέπει να σημειώσουμε ότι αν δεν υπάρχει μία από τις δύο ιδιότητες (αριστερογραμμικότητα και απουσία κρισίμων ζευγών) τότε πιθανόν να μην έχουμε συμβολή, όπως φαίνεται από τα ακόλουθα παραδείγματα.

**7.1.5 Παράδειγμα** ([52]). Το σύστημα :

$$\mathcal{R}: \quad f(x,x) \to a, \quad f(x,g(x)) \to b, \quad c \to g(c)$$

δεν έχει κρίσιμα ζεύγη, αλλά είναι μη αριστερογραμμικό. Επιπλέον έχουμε :

$$a \leftarrow f(c,c) \to f(c,g(c)) \to b,$$

δηλαδή δεν έχουμε μοναδικότητα κανονικών μορφών και επομένως ούτε συμβολή.

**7.1.6 Παράδειγμα.** Το σύστημα επί του αλφαβήτου $\{L^1, 0\}$ με μοναδικό κανόνα μεταγραφής τον $L(L(x)) \to 0$ είναι αριστερογραμμικό, αλλά έχει την κρίσιμη κορυφή :

$$L(0) \leftarrow L(L(L(0))) \to 0,$$

δηλαδή δεν έχουμε μοναδικότητα κανονικών μορφών και επομένως ούτε συμβολή.

Από υπολογιστική άποψη, τα μη αριστερογραμμικά συστήματα, απαιτούν, προκειμένου να εφαρμοστεί μη αριστερογραμμικός κανόνας, να γίνει έλεγχος συντακτικής ισότητας υποόρων σε συγκεκριμένες θέσεις του υπό μεταγραφή όρου. Αυτό όμως ενδέχεται να είναι υπολογιστικά απαιτητικό, οπότε είναι προτιμότερο για συστήματα μεταγραφής που χρησιμοποιούνται ως υπολογιστικά συστήματα να είναι αριστερογραμμικά. Επιπλέον θέλουμε σε ένα σύστημα μεταγραφής που χρησιμοποιείται για υπολογισμούς να έχουμε και την ιδιότητα της συμβολής, ούτως ώστε αποκλίνοντες υπολογισμοί να φτάνουν στο ίδιο τελικό αποτέλεσμα. Όπως θα δούμε στην επόμενη ενότητα, τα ορθογώνια συστήματα έχουν και την ιδιότητα της συμβολής.

## 7.2 Συμβολή στα ορθογώνια συστήματα

Σε αυτήν την ενότητα προτιθέμεθα να αποδείξουμε το θεώρημα :

**7.2.1 Πρόταση** (Rosen [89]). *Κάθε ορθογώνιο σύστημα μεταγραφής είναι συμβάλλον.*

Για να δείξουμε το θεώρημα, χρειαζόμαστε την έννοια της *παράλληλης αναγωγής* και μία ιδιότητα αυτής που χαρακτηρίζεται ως *λήμμα παραλλήλων κινήσεων* (parallel moves lemma), από το [22].

**7.2.2 Ορισμός** (παράλληλη αναγωγή)**.** Έστω σχέση μεταγραφής $\to$. Ορίζουμε την σχέση $\overset{\parallel}{\to}$ ως εξής : $s \overset{\parallel}{\to} t$, αν υπάρχουν $u_0$, $u_1$, ..., $u_n$ τέτοια ώστε

$$s \equiv u_0 \to_{p_0} u_1 \to_{p_1} \ldots \to_{p_{n-1}} u_n \equiv t$$

και οι θέσεις $p_0$, $p_1$, ..., $p_{n-1}$ είναι ανά δύο παράλληλες.

*7.2.3 Παρατήρηση.* Παρατηρούμε ότι η σχέση $\overset{\parallel}{\to}$ έχει την εξής ιδιότητα :

$$\overset{*}{\to} \subseteq (\overset{\parallel}{\to})^* \subseteq (\overset{*}{\to})^* = \overset{*}{\to},$$

οπότε έχουμε άμεσα ότι : $(\overset{\parallel}{\to})^* = \overset{*}{\to}$. Επιπλέον, η σχέση της παράλληλης αναγωγής μπορεί να θεωρηθεί ανακλαστική (δεν γίνεται αναγωγή σε καμμία θέση). Αυτό σημαίνει ότι η ρομβοειδής ιδιότητα ($\diamondsuit$) και η υποαντιμεταθετικότητα (WCR[1]) ταυτίζονται (βλέπε παρατήρηση 3.1.5 στην σελίδα 20). Τότε από την πρόταση 3.1.17 στην σελίδα 24, έχουμε ότι αν η παράλληλη αναγωγή είναι WCR[1] ή $\diamondsuit$, τότε είναι και CR, οπότε είναι CR και η $\to$. Επομένως, προκειμένου να δείξουμε το θεώρημα 7.2.1, είναι αρκετό να δείξουμε την παρακάτω πρόταση :

**7.2.4 Πρόταση.** *Σε ένα ορθογώνιο σύστημα, η σχέση $\overset{\parallel}{\to}$ έχει την ρομβοειδή ιδιότητα, δηλαδή $\overset{\parallel}{\leftarrow} \circ \overset{\parallel}{\to} \subseteq \overset{\parallel}{\to} \circ \overset{\parallel}{\leftarrow}$ (ορισμός 3.1.2, σελίδα 20).*

*Απόδειξη.* Θεωρούμε την κορυφή $s \overset{\parallel}{\leftarrow} u \overset{\parallel}{\to} t$. Έστω ότι ο όρος $s$ έχει προκύψει με αναγωγές στις (παράλληλες) θέσεις $p_1$, ..., $p_m$ και ο $t$ με αναγωγές στις $q_1$, ..., $q_m$. Πρέπει να δείξουμε ότι, όταν το σύστημα είναι ορθογώνιο, σε κάθε περίπτωση, μπορούμε ξεκινώντας από κάθε ένα από τους όρους $s$ και $t$ και εφαρμόζοντας ξεχωριστά στον καθέναν από αυτούς αναγωγές *μόνον* σε παράλληλες μεταξύ τους θέσεις να φτάσουμε σε έναν κοινό όρο $v$. Αυτό θα το κάνουμε ανάλογα με το πώς σχετίζεται η κάθε θέση σε κάθε ένα από τα δύο σύνολα :

$$\{p_1, \ldots, p_m\} \quad \{q_1, \ldots, q_m\}$$

με όλες τις θέσεις του άλλου συνόλου.

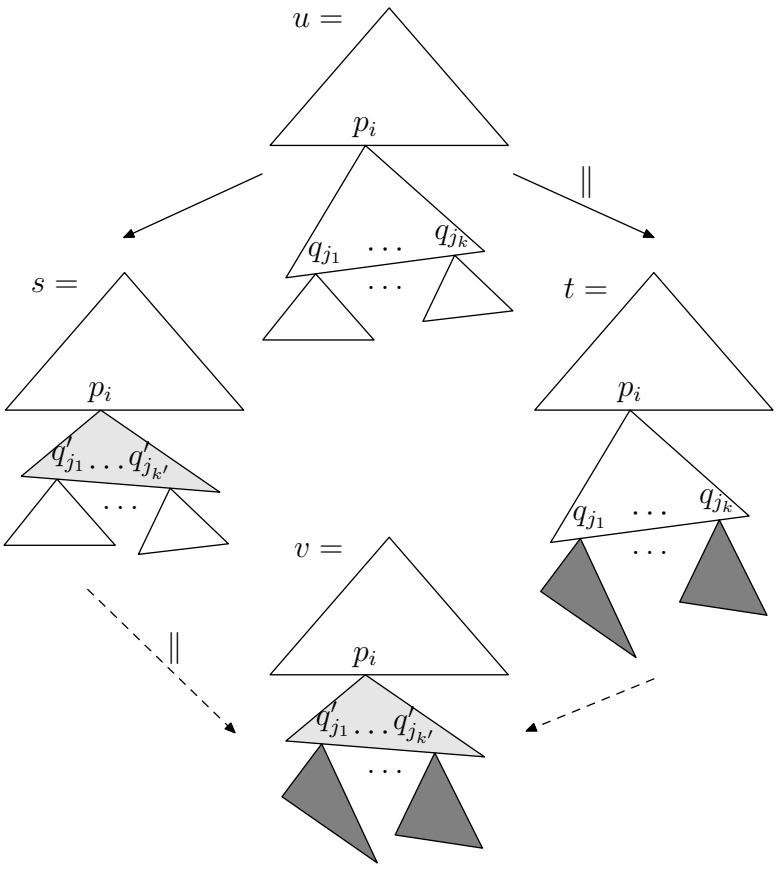
Αρχικά, έχουμε την περίπτωση : $p_i = q_j$, για κάποια $i$, $j$. Αφού το σύστημα είναι ορθογώνιο δεν έχουμε κρίσιμα ζεύγη. Αυτό σημαίνει ότι έχουμε εφαρμογή του ίδιου κανόνα, στην ίδια θέση, οπότε προκύπτει ο ίδιος υποόρος (αποκλείεται επικάλυψη μεταβλητής γιατί βρισκόμαστε στην ίδια θέση). Έτσι, οι δύο όροι $s$, $t$ δεν διαφέρουν σε αυτό το σημείο.

Μετά έχουμε την περίπτωση στην οποία μία θέση στο ένα σύνολο είναι παράλληλη με όλες τις θέσεις του άλλου συνόλου. Χωρίς βλάβη της γενικότητας, υποθέτουμε ότι αυτή η θέση είναι η $p_i$ στον $s$. Τότε, για να προσεγγίσουμε τον όρο $u$ από τον $t$, εφαρμόζουμε την αντίστοιχη αναγωγή στην θέση $p_i$.

Τέλος, θεωρούμε την περίπτωση στην οποία μία θέση δεν είναι ξένη με κάποιες θέσεις του άλλου συνόλου. Χωρίς βλάβη της γενικότητας υποθέτουμε ότι έχουμε $p_i$ και $\{q_j \mid j \in J\}$, για κάποιο σύνολο δεικτών $J$ από 1 έως $m$ και ισχύουν τα εξής :

- Ακριβώς για τα $j \in J$ έχουμε $p_i < q_j$, δηλαδή η θέση $p_i$ είναι πιο κοντά στην ρίζα από οποιαδήποτε άλλη θέση του συνόλου $\{q_j \mid j \in J\}$.
- Προφανώς, οι θέσεις του συνόλου $\{q_j \mid j \in J\}$ είναι παράλληλες μεταξύ τους.

Για κάθε ζεύγος $(p_i, q_j)$, όπου $j \in J$, οι αντίστοιχοι δύο κανόνες μεταγραφής που εφαρμόζονται επικαλύπτονται υποχρεωτικά σε θέση μεταβλητής (αφού δεν έχουμε κρίσιμα ζεύγη), δηλαδή έχουμε μία περίπτωση εμφωλευμένων αναξίμων (η κατάσταση είναι παρόμοια με αυτήν του σχήματος 6.2β, σελίδα 64). Στον όρο $s$ έχει γίνει η αναγωγή στην θέση $p_i$ και έστω στον όρο $t$ στις θέσεις $q_{j_1}$, ..., $q_{j_k}$ (θεωρούμε απλώς μία αρίθμηση των στοιχείων του συνόλου $\{q_j \mid j \in J\}$, για να δείξουμε τις θέσεις στο σχήμα 7.4).




Σχήμα 7.4 – Κατασκευή κοινού όρου με παράλληλη αναγωγή

Είναι σαφές πλέον τι πρέπει να κάνουμε για να καταλήξουμε από τους $s$ και $t$ σε έναν κοινό απόγονο $u$, όσον αφορά τις θέσεις $p_i$ και $q_{j_1}$, ..., $q_{j_k}$ : Από τον $s$ κάνουμε τις αναγωγές των εμφωλευμένων αναξίμων σε κάθε μία από τις καινούριες θέσεις $q'_{j_1}$, ..., $q'_{j_{k'}}$ που έχουν προκύψει.[1] Αφού οι θέσεις $q'_{j_1}$, ..., $q'_{j_{k'}}$ είναι παράλληλες, όλες η παραπάνω αναγωγές μπορούν να γίνουν σε ένα βήμα παράλληλης αναγωγής. Επίσης, από τον $t$ μπορούμε, επειδή το σύστημα είναι αριστερογραμμικό να πάμε μόλις σε ένα βήμα εφαρμόζοντας τον κανόνα για την θέση $p_i$ στον κοινό απόγονο $u$ (δεν υπάρχει περίπτωση να απαιτείται από τον κανόνα της θέσης $p_i$ ισότητα μεταξύ υποόρου σε κάποια θέση $q_j$ με κάποιον άλλο υποόρο που δεν βρίσκεται μεταξύ των θέσεων $\{q_j \mid j \in J\}$). Ας σημειωθεί πως αν το σύστημα δεν είναι αριστερογραμμικό, μπορεί να έχουμε μία περίπτωση όπως στο σχήμα 6.2γ στην σελίδα 64 και δεν μπορούμε να φτάσουμε σε κοινό απόγονο με μία παράλληλη αναγωγή.

Πιο περίπλοκες περιπτώσεις δεν υπάρχουν να ερευνήσουμε, λόγω της μεταβατικότητας της παραλληλίας των θέσεων· για παράδειγμα αποκλείεται να συναντήσουμε μία περίπτωση $p_i < q_j < p_l$. □

## 7.3 Κανονικές μορφές και τερματισμός

Όπως είδαμε, στα ορθογώνια συστήματα έχουμε την ιδιότητα της συμβολής. Τι γίνεται όμως με ιδιότητες που αφορούν τον τερματισμό και τις κανονικές μορφές; Αυτή η ενότητα βασίζεται στο [48, ενότητα 3.1].

Όλες οι παρακάτω ιδιότητες ισχύουν σε ένα ορθογώνιο σύστημα $\mathcal{R}$ (βλέπε [18, 89, 79, 59]) :

**7.3.1 Πρόταση.** $\mathrm{WIN}(\mathcal{R}) \iff \mathrm{SIN}(\mathcal{R}) \iff \mathrm{SN}(\mathcal{R})$.
*Παρομοίως :* $\forall t\colon \mathrm{WIN}(t) \iff \mathrm{SIN}(t) \iff \mathrm{SN}(t)$.

Ας δούμε τώρα τι συμβαίνει αν έχουμε διαγραφές :

**7.3.2 Πρόταση.** *Αν* $s \to_{p,\sigma,l\to r} t$ *(βλέπε ορισμό 5.1.4 στην σελίδα 48) και ισχύει* $\mathrm{SN}(t)$ *και* $\neg\mathrm{SN}(s)$*, τότε ο όρος* $s|_p$ *περιέχει έναν γνήσιο υποόρο* $s' = x^\sigma$*, για κάποιο* $x \in \mathrm{Var}(l)$*, με* $\neg\mathrm{SN}(s')$.

Αυτό σημαίνει ότι ο μη τερματίζων όρος $s' = x^\sigma$ *διαγράφεται* από το βήμα μεταγραφής $s \to_{p,\sigma,l\to r} t$ και ότι ο κανόνας $l \to r$ είναι διαγράφων (βλέπε και ορισμό 5.4.1 στην σελίδα 52), αφού $x \in \mathrm{Var}(l) - \mathrm{Var}(r)$ επειδή $\mathrm{SN}(t)$. Φυσικά,

1. Αν ο κανόνας που εφαρμόζεται στην θέση $p_i$ είναι διαγράφων ενδέχεται κάποιες θέσεις αναγωγής να χάνονται, δηλαδή να μην υπάρχει για κάθε $q_j$ η αντίστοιχη $q'_j$, αλλά αυτό δεν είναι πρόβλημα. Επιπλέον επιτρέπονται διπλασιάζοντες κανόνες, οπότε ενδέχεται από μία αρχική θέση $q_j$ να προκύπτουν περισσότερες από μία θέσεις, αλλά ούτε αυτό είναι πρόβλημα.

το βήμα μεταγραφής δεν είναι εσωτερικότερο, αφού το $s'$ είναι ανάξιμο (ως $\neg\mathrm{SN}(s')$). Έτσι, μπορεί να προκύψει εύκολα ότι αν $s \xrightarrow[\mathrm{i}]{} t$ και $\mathrm{SN}(t)$, τότε υποχρεωτικά $\mathrm{SN}(s)$.

Εύκολα επίσης προκύπτουν τα εξής αν ένα ορθογώνιο σύστημα $\mathcal{R}$ δεν είναι διαγράφον (non-erasing· συμβολισμός $\mathrm{NE}(\mathcal{R})$) :

**7.3.3 Πρόταση.** *Έστω ορθογώνιο σύστημα $\mathcal{R}$ με $\mathrm{NE}(\mathcal{R})$, τότε :*

$$\mathrm{WN}(\mathcal{R}) \iff \mathrm{WIN}(\mathcal{R}) \iff \mathrm{SIN}(\mathcal{R}) \iff \mathrm{SN}(\mathcal{R})$$

*και παρομοίως :*

$$\forall t\colon \mathrm{WN}(t) \iff \mathrm{WIN}(t) \iff \mathrm{SIN}(t) \iff \mathrm{SN}(t).$$

Το τελευταίο θα μας φανεί χρήσιμο στο κεφάλαιο 12, για το σύστημα του $S$-λογισμού, και σημαίνει ότι αν ένας όρος έχει κανονική μορφή τότε τερματίζει πάντοτε και αντιθέτως αν από αυτόν αρχίζει μία άπειρη αλυσίδα αναγωγών τότε αποκλείεται να έχει κανονική μορφή.

# Μέρος δεύτερο

# Γλώσσες δένδρων

# Κεφάλαιο 8

# Γλώσσες και πεπερασμένα αυτόματα δένδρων

Όπως είδαμε, οι όροι είναι στενά συνδεδεμένοι με τα δένδρα, δεδομένου ότι οποιοσδήποτε όρος μπορεί να αναπαρασταθεί σε δενδρική μορφή. Επίσης, είναι γνωστό ότι για κάποια σύνολα συμβολοσειρών (ή αλλιώς γλώσσες) υπάρχουν αυτόματα, τα οποία αποκρίνονται αν μία συμβολοσειρά εισόδου ανήκει σε ένα συγκεκριμένο σύνολο (βλέπε, για παράδειγμα, [109, 51]). Στην περίπτωση των συμβολοσειρών τα απλούστερα τέτοια αυτόματα (και τα λιγότερο ισχυρά από άποψη πλήθους γλωσσών που αναγνωρίζουν) είναι τα λεγόμενα αυτόματα πεπερασμένων καταστάσεων. Σε αυτό το κεφάλαιο θα ορίσουμε παρόμοια αυτόματα τα οποία λειτουργούν επί δένδρων και θα μελετήσουμε τις ιδιότητές τους. Θα δούμε ότι πολλά αποτελέσματα μεταφέρονται σχεδόν αυτούσια από τις γλώσσες συμβολοσειρών στις γλώσσες δένδρων, αλλά υπάρχουν και κάποιες διαφορές.

Τα αυτόματα δένδρων χρησιμοποιήθηκαν πρώτα ως εργαλεία στο πλαίσιο της επαλήθευσης κυκλωμάτων (circuit verification) στα τέλη της δεκαετίας του 1950 από τον Church [17] και κατόπιν από τους Trakhtenbrot [102], Büchi [11], Rabin [87], Doner [34], Thatcher [99, 100] και άλλους. Στην δεκαετία του 1970 μελετήθηκαν περισσότερο ως αυτόνομα αντικείμενα και στην επόμενη δεκαετία βρήκαν εφαρμογές στην επαλήθευση προγραμμάτων.

Το κεφάλαιο αυτό βασίζεται κυρίως στο [20]. Άλλες αναφορές είναι τα [43, 44].

## 8.1 Γλώσσες δένδρων

Ως γνωστόν, δεδομένου ενός αλφαβήτου (μη συναρτησιακών) συμβόλων $\Sigma$, οποιοδήποτε υποσύνολο του $\Sigma^*$ ονομάζεται γλώσσα συμβολοσειρών (βλέ-

πε παρατήρηση 4.2.5 στην σελίδα 37).

Οι γλώσσες δένδρων ορίζονται σε πλήρη αντιστοιχία με τις γλώσσες συμβολοσειρών :

**8.1.1 Ορισμός.** Έστω ένα αλφάβητο $\mathcal{F}$ συναρτησιακών συμβόλων. Οποιοδήποτε υποσύνολο του $\mathcal{T}(\mathcal{F})$ ονομάζεται *γλώσσα δένδρων*.

**8.1.2 Παράδειγμα.** Έστω ότι δίδεται το αλφάβητο συναρτησιακών συμβόλων $\mathcal{F} = \{f(,), g(), a\}$. Το σύνολο των όρων από το παραπάνω αλφάβητο για τους οποίους το πλήθος των εμφανίσεων του $a$ στον εκάστοτε όρο είναι πρώτος αριθμός αποτελεί μία γλώσσα δένδρων, την οποία έστω ότι ονομάζουμε $L$, τέτοια ώστε $L = \{t \in \mathcal{T}(\mathcal{F}) \mid |\mathrm{Occ}(t, a)| \text{ είναι πρώτος}\}$.

*8.1.3 Παρατήρηση.* Οι γλώσσες δένδρων αποτελούν γενίκευση των γλωσσών συμβολοσειρών με την έννοια ότι μπορούμε να παραστήσουμε κάθε συμβολοσειρά σε δενδρική μορφή ως εξής : Αν δίδεται ένα αλφάβητο $\Sigma$, τότε κατασκευάζουμε ένα αλφάβητο συναρτησιακών συμβόλων ως εξής : σε κάθε $x \in \Sigma$ αποδίδουμε πλήθος ορισμάτων ίσο με 1 και τέλος προσθέτουμε ένα νέο σύμβολο σταθερά, διαφορετικό των συμβόλων του $\Sigma$, έστω το $\#$. Έτσι, για παράδειγμα, η συμβολοσειρά $abbab$ γίνεται το δένδρο $a(b(b(a(b(\#)))))$. Βλέπε και ορισμό 5.8.1 στην σελίδα 59.

## 8.2 Πεπερασμένα αυτόματα bottom-up

**8.2.1 Ορισμός.** Ένα *πεπερασμένο μη αιτιοκρατικό αυτόματο που λειτουργεί από τα κάτω προς τα πάνω* (bottom-up non-deterministic finite tree automaton ή για συντομία BUNFTA) είναι μία πλειάδα $\mathcal{A} = (\mathcal{F}, Q, Q_\mathrm{f}, \Delta)$, όπου $\mathcal{F}$ : αλφάβητο συναρτησιακών συμβόλων, $Q$ : πεπερασμένο σύνολο (των λεγόμενων καταστάσεων), $Q_\mathrm{f} \subseteq Q$ : το σύνολο των τελικών καταστάσεων, και $\Delta$ : ένα σύνολο κανόνων μετάβασης της μορφής $f(q_1, \ldots, q_n) \to q$, με $n \geq 0$, $f \in \mathcal{F}_n$, $q, q_1, \ldots, q_n \in Q$.

Σε αντίθεση με τα πεπερασμένα αυτόματα συμβολοσειρών, τα πεπερασμένα αυτόματα δένδρων που λειτουργούν από τα κάτω προς τα πάνω δεν έχουν αρχικές καταστάσεις. Όμως, για τα σύμβολα-σταθερές, οι κανόνες μετάβασης έχουν την μορφή $a \to q$, οπότε αυτοί οι κανόνες σηματοδοτούν την αρχή λειτουργίας του αυτομάτου, αφού αυτό αρχίζει να λειτουργεί από τα φύλλα του δένδρου.

Η έννοια της λειτουργίας από τα κάτω προς τα πάνω, σημαίνει ότι αφού το αυτόματο «αποδώσει» καταστάσεις στα φύλλα του δένδρου, συνεχίζει, για όσο αυτό είναι δυνατόν, βάσει των κανόνων που περιέχονται στο σύνολο

$\Delta$, να «αποδίδει» καταστάσεις επαγωγικά σε όλο και υψηλότερες θέσεις του δένδρου, μέχρι να φτάσει, αν πάλι αυτό είναι δυνατόν, στην ρίζα.

Αν θεωρήσουμε το αλφάβητο συναρτησιακών συμβόλων $\mathcal{F}$ επαυξημένο με το σύνολο $Q$, των καταστάσεων, αφού πρώτα αποδώσουμε στις καταστάσεις πλήθος ορισμάτων ίσο με 0 (δηλαδή τις θεωρούμε ως σταθερές), τότε το σύνολο των κανόνων μετάβασης ορίζει ένα θεμελιώδες σύστημα μεταγραφής όρων (βλέπε ορισμό 5.4.2 στην σελίδα 53). Την σχέση μεταγραφής που επιβάλλεται από το παραπάνω σύστημα (βλέπε ορισμό 5.1.4 στην σελίδα 48) την συμβολίζουμε με $\rightarrow_{\mathcal{A}}$. Κατά τα συνήθη, το ανακλαστικό μεταβατικό κλείσιμο αυτής συμβολίζεται με $\xrightarrow{*}_{\mathcal{A}}$.

**8.2.2 Ορισμός.** Ένας (θεμελιώδης) όρος $t \in \mathcal{T}(\mathcal{F})$ γίνεται αποδεκτός από το αυτόματο $\mathcal{A} = (\mathcal{F}, Q, Q_{\mathrm{f}}, \Delta)$ αν και μόνον αν $t \xrightarrow{*}_{\mathcal{A}} q$, για κάποια τελική κατάσταση $q$.

**8.2.3 Ορισμός.** Έστω ένα αυτόματο (BUNFTA) $\mathcal{A}$. Λέμε ότι η γλώσσα που *αναγνωρίζει* το $\mathcal{A}$ είναι το σύνολο των δένδρων που γίνονται αποδεκτά από το $\mathcal{A}$ και το οποίο συμβολίζουμε με $L(\mathcal{A})$.

Μία γλώσσα $L$ ονομάζεται *αναγνωρίσιμη* όταν υπάρχει αυτόματο $\mathcal{A}$, για το οποίο $L = L(\mathcal{A})$.

Δύο αυτόματα $\mathcal{A}_1$, $\mathcal{A}_2$ ονομάζονται ισοδύναμα όταν αναγνωρίζουν την ίδια γλώσσα, δηλαδή όταν $L(\mathcal{A}_1) = L(\mathcal{A}_2)$.

**8.2.4 Παράδειγμα.** Ας θεωρήσουμε το αλφάβητο $\mathcal{F} = \{f(,), g(), a\}$ και την γλώσσα $L$ των όρων της μορφής $g(g(t))$, με $t \in \mathcal{T}(\mathcal{F})$. Το αυτόματο $\mathcal{A} = (\mathcal{F}, \{q, q_g, q_{gg}\}, \{q_{gg}\}, \Delta)$, όπου

$$\begin{aligned}\Delta = \{ \quad & a \to q, \quad g(q) \to q, \quad f(q,q) \to q, \\ & g(q) \to q_g, \quad g(q_q) \to q_{gg} \quad \},\end{aligned}$$

αναγνωρίζει την γλώσσα $L$.

Η συνθήκη $t \xrightarrow{*}_{\mathcal{A}} q$, με $q \in Q_{\mathrm{f}}$ μας λέει απλώς ότι το αυτόματο $\mathcal{A}$ αποδέχεται τον όρο $t$, χωρίς να μας λέει πώς έγινε αυτό. Για αυτόν τον λόγο, ορίζουμε την περιεκτικότερη σε πληροφορία σχετικά με την διαδικασία απόδοσης καταστάσεων κατά την λειτουργία του αυτομάτου έννοια της *εκτέλεσης* (run) :

**8.2.5 Ορισμός.** *Εκτέλεση* ονομάζουμε μία συμβατή με το σύνολο κανόνων μετάβασης $\Delta$ μερική[1] συνάρτηση $r\colon \mathrm{Pos}(t) \rightharpoonup Q$, δηλαδή αν $p \in \mathrm{Pos}(t)$ και

1. Βλέπε ορισμό 3.4.6 στην σελίδα 30.

$\mathrm{root}(t|_p) = f \in \mathcal{F}_n$, ενώ για την $r$ ισχύουν : $r(p) = q$, $r(p.i) = q_i$, για κάθε $i$ με $1 \leq i \leq n$, τότε $f(q_1, \ldots, q_n) \to q \in \Delta$.

Μία εκτέλεση $r$ είναι *ολοκληρωμένη* αν $\varepsilon \in \mathrm{dom}(r)$, δηλαδή η $r$ είναι ορισμένη στην κορυφαία θέση του όρου. Αν επιπλέον $r(\varepsilon) \in Q_\mathrm{f}$, τότε η εκτέλεση ονομάζεται *επιτυχής*.

Με την βοήθεια της έννοιας της εκτέλεσης μπορούμε να διερευνήσουμε πότε ένα αυτόματο αποδέχεται έναν όρο. Πρώτα από όλα, ας σημειώσουμε ότι επειδή το αυτόματο είναι μη αιτιοκρατικό, ίσως υπάρχουν κανόνες μετάβασης με το ίδιο αριστερό μέλος, οπότε είναι δυνατόν για τον ίδιο όρο να προκύψουν πολλές διαφορετικές εκτελέσεις στο ίδιο αυτόματο, ανάλογα με την δυνατότητα επιλογής διαφορετικών τέτοιων κανόνων στην ίδια θέση. Αν ένας όρος γίνεται αποδεκτός από ένα αυτόματο, τότε υπάρχει τουλάχιστον μία επιτυχής εκτέλεση. Αν όμως δεν υπάρχει επιτυχής εκτέλεση για έναν όρο, τότε αυτός δεν γίνεται αποδεκτός και όλες οι εκτελέσεις ανήκουν σε δύο κατηγορίες : είτε είναι ολοκληρωμένες και μη επιτυχείς ($r(\varepsilon) \in Q - Q_\mathrm{f}$), είτε δεν είναι καν ολοκληρωμένες ($\varepsilon \notin \mathrm{dom}(r)$).

## 8.3 Αυτόματα με $\varepsilon$-κινήσεις.

Στην περίπτωση των αυτομάτων συμβολοσειρών έχει νόημα να ορίσουμε αυτόματα στα οποία έχουμε μετάβαση από μία κατάσταση σε μία άλλη χωρίς την επεξεργασία οποιουδήποτε συμβόλου. Ορίζουμε κάτι εντελώς ανάλογο στην περίπτωση των αυτομάτων δένδρων :

**8.3.1 Ορισμός.** Ένα BUNFTA *με $\varepsilon$-κινήσεις* είναι ακριβώς όπως ένα BUNFTA (βλέπε ορισμό 8.2.1) με την επιπλέον ιδιότητα να περιέχει κανόνες μετάβασης της μορφής $q \to q'$, όπου $q$, $q'$ : καταστάσεις του αυτομάτου. (Αυτού του είδους οι κανόνες ονομάζονται $\varepsilon$-κανόνες.)

Το σύνολο των κανόνων μετάβασης ορίζει ένα θεμελιώδες σύστημα μεταγραφής, ακριβώς όπως στα απλά αυτόματα (χωρίς $\varepsilon$-κινήσεις), ενώ εντελώς ανάλογα προκύπτει και η σχέση μεταγραφής.

**8.3.2 Παράδειγμα.** Έστω το αλφάβητο $\mathcal{F} = \{g(), h(), a\}$ και η γλώσσα $L = \{g^m(h^n(a)) \mid m, n \in \mathbb{N}\}$. Το αυτόματο $\mathcal{A} = (\mathcal{F}, \{q, q_g\}, \{q_g\}, \Delta)$, όπου

$$\Delta = \{ \quad a \to q, \quad h(q) \to q, \quad q \to q_g, \quad g(q_g) \to q_g \quad \},$$

αναγνωρίζει την γλώσσα $L$.

Για την παραπάνω γλώσσα, μπορεί κάποιος εύκολα να κατασκευάσει ένα ισοδύναμο αυτόματο χωρίς $\varepsilon$-κινήσεις που να την αναγνωρίζει. Αυτό είναι μία γενικότερη ιδιότητα, όπως προκύπτει από την παρακάτω πρόταση :

**8.3.3 Πρόταση.** *Αν $\mathcal{A}$ είναι αυτόματο με ε-κανόνες, τότε υπάρχει αυτόματο $\mathcal{A}'$ χωρίς ε-κανόνες, ισοδύναμο με το $\mathcal{A}$ (δηλαδή $L(\mathcal{A}) = L(\mathcal{A}')$).*

*Απόδειξη.* Έστω αυτόματο $\mathcal{A} = (\mathcal{F}, Q, Q_{\rm f}, \Delta)$ με ε-κανόνες. Διαμερίζουμε το σύνολο $\Delta$, στο σύνολο $\Delta_\varepsilon$ που περιέχει μόνον τους ε-κανόνες και στο $\Delta_\mathcal{F}$ που περιέχει τους υπόλοιπους κανόνες.

Ορίζουμε το ε-κλείσιμο$(q)$ ως το σύνολο των καταστάσεων $q'$ για τις οποίες $q \xrightarrow{*}_\mathcal{A} q'$. Αν θεωρήσουμε ότι το σύνολο $\Delta_\varepsilon$ μπορεί να παραστήσει τις ακμές ενός πεπερασμένου κατευθυνόμενου γράφου, με κορυφές τις καταστάσεις του αυτομάτου, τότε η εύρεση του ε-κλείσιμο$(q)$ για κάθε κατάσταση $q \in Q$ είναι εφικτή αλγοριθμικά σε πολυωνυμικό χρόνο, αφού σχετίζεται με το πρόβλημα της ύπαρξης ή όχι μονοπατιού από την $q$ στις υπόλοιπες καταστάσεις (γνωστό ως πρόβλημα REACHABILITY· βλέπε [82]).

Κατασκευάζουμε το σύνολο κανόνων $\Delta'_\mathcal{F}$ ως εξής :

$$f(q_1, \ldots, q_n) \to q' \in \Delta'_\mathcal{F} \iff f(q_1, \ldots, q_n) \to q \in \Delta_\mathcal{F} \quad \text{και} \quad q' \in \varepsilon\text{-κλείσιμο}(q)$$

με τους οποίους σκοπεύουμε να αντικαταστήσουμε τους κανόνες του $\Delta_\varepsilon$. Τότε προκύπτει ισοδύναμο του $\mathcal{A}$ αυτόματο, έστω το $\mathcal{A}$, χωρίς ε-κανόνες : $\mathcal{A}' = (\mathcal{F}, Q, Q_{\rm f}, \Delta_\mathcal{F} \cup \Delta'_\mathcal{F})$. Η ισοδυναμία των αυτομάτων αποδεικνύεται εύκολα με επαγωγή στο μήκος των αλυσίδων αναγωγής των θεμελιωδών συστημάτων μεταγραφής που ορίζουν τα $\mathcal{A}$ και $\mathcal{A}'$. □

Τότε, ποιο είναι το νόημα της εισαγωγής των ε-κινήσεων, αφού δεν προσφέρουν επιπλέον δυνατότητες αναγνώρισης στα απλά αυτόματα ; Η απάντηση είναι ότι με χρήση των ε-κανόνων αφ' ενός μεν περιγράφονται με φυσικότερο τρόπο ορισμένες γλώσσες, αφ' ετέρου δε γίνονται ευκολότερα κάποιες αποδείξεις.

## 8.4 Πλήρη και μειωμένα αυτόματα

Πολλές φορές, μία εκτέλεση σταματά πριν φτάσει στην ρίζα του δένδρου (φυσικά τότε δεν είναι επιτυχής). Μάλιστα, ενδέχεται για κάποιον όρο και ένα αυτόματο να μην υπάρχει ολοκληρωμένη εκτέλεση. Προκειμένου να φροντίσουμε να υπάρχει τουλάχιστον μία ολοκληρωμένη εκτέλεση για κάθε θεμελιώδη όρο επιβάλλουμε την ιδιότητα της πληρότητας :

**8.4.1 Ορισμός.** Ένα αυτόματο ονομάζεται *πλήρες* (complete) αν υπάρχει τουλάχιστον ένας κανόνας $f(q_1, \ldots, q_n) \to q \in \Delta$, για κάθε $f \in \mathcal{F}$ και $q_1, \ldots, q_n \in Q$.

**8.4.2 Παράδειγμα.** Το αυτόματο του παραδείγματος 8.2.4 δεν είναι πλήρες.

Μπορούμε εύκολα να μετατρέψουμε οποιοδήποτε αυτόματο (BUNFTA) σε πλήρες : Αρκεί να θεωρήσουμε μία επιπλέον μη τελική κατάσταση $q_\pi$ και να προσθέσουμε στο σύνολο των κανόνων όσους κανόνες υπολείπονται προκειμένου το αυτόματο να γίνει πλήρες, πάντοτε θέτοντας ως δεξιό μέλος την $q_\pi$. Έτσι, αποδείξαμε το παρακάτω θεώρημα :

**8.4.3 Θεώρημα.** *Για κάθε αναγνωρίσιμη γλώσσα υπάρχει πλήρες BUNFTA που την αναγνωρίζει.*

*8.4.4 Παρατήρηση.* Η μετατροπή ενός αυτομάτου σε πλήρες ενδέχεται να προκαλέσει σημαντική αύξηση του πλήθους των κανόνων αφού για κάθε σύμβολο $f \in \mathcal{F}_n$ υπάρχουν $n^{|Q|}$ κανόνες όπου το ριζικό σύμβολο του αριστερού μέλους είναι το $f$.

Τώρα, θα αναφερθούμε στα αυτόματα που χαρακτηρίζονται *μειωμένα* και πώς μετατρέπουμε τυχόν αυτόματο σε μειωμένο.

Ενδέχεται να υπάρχουν σε ένα αυτόματο καταστάσεις οι οποίες για οποιονδήποτε όρο και σε οποιαδήποτε εκτέλεση δεν εμφανίζονται ποτέ. Προφανώς, αυτές οι καταστάσεις είναι «άχρηστες», με την έννοια ότι ακόμη και αν καταργηθούν δεν μεταβάλλεται η γλώσσα που αναγνωρίζεται από το αυτόματο. Οι υπόλοιπες καταστάσεις, όμως, θεωρούνται «χρήσιμες» και για αυτές δίνουμε τον παρακάτω τυπικό ορισμό :

**8.4.5 Ορισμός.** Μία κατάσταση $q$ ενός αυτομάτου $\mathcal{A}$ ονομάζεται *προσβάσιμη* (accesible) αν υπάρχει θεμελιώδης όρος $t$ τέτοιος ώστε $t \xrightarrow{*}_{\mathcal{A}} q$.

**8.4.6 Ορισμός.** Ένα αυτόματο ονομάζεται *μειωμένο* (reduced) αν όλες του οι καταστάσεις είναι προσβάσιμες.

**8.4.7 Παράδειγμα.** Το αυτόματο του παραδείγματος 8.2.4 είναι μειωμένο. Αν όμως σε αυτό προσθέσουμε, για παράδειγμα, μία επιπλέον κατάσταση, έστω $q_x$, χωρίς την προσθήκη επιπλέον κανόνων, τότε παύει να είναι μειωμένο.

Πώς όμως μετατρέπουμε ένα αυτόματο σε μειωμένο το οποίο αναγνωρίζει την ίδια γλώσσα; Αρκεί να βρούμε το σύνολο $Q_{\rm a}$ των προσβασίμων καταστάσεων. Ο αλγόριθμος για το παραπάνω είναι πολύ απλός : Αρχικά, τοποθετούμε στο σύνολο $Q_{\rm a}$ όλες τις καταστάσεις οι οποίες είναι δεξιά μέλη κανόνων της μορφής $a \to q$, για κάθε σταθερά $a$. Κατόπιν επαναλαμβάνουμε την εξής διαδικασία : Για κάθε κανόνα $f(q_1, \ldots, q_n) \to q$, αν $q_1, \ldots, q_n \in Q_{\rm a}$, τότε προσθέτουμε την $q$ στο $Q_{\rm a}$. Η παραπάνω διαδικασία επαναλαμβάνεται μέχρι να μην είναι δυνατό να προστεθεί κατάσταση στο σύνολο $Q_{\rm a}$. Προφανώς, όλες οι καταστάσεις του $Q_{\rm a}$ είναι προσβάσιμες και με απλή επαγωγή μπορούμε να δείξουμε ότι είναι όλες οι προσβάσιμες καταστάσεις. Επομένως, αποδείξαμε το παρακάτω θεώρημα :

**8.4.8 Θεώρημα.** *Για κάθε αναγνωρίσιμη γλώσσα υπάρχει μειωμένο BUNFTA που την αναγνωρίζει.*

## 8.5 Αιτιοκρατικά αυτόματα

Στα μη αιτιοκρατικά αυτόματα, που είδαμε ως τώρα, δεν υπάρχει περιορισμός στην μορφή των κανόνων μετάβασης του συνόλου $\Delta$. Η αιτιοκρατία θα προκύψει αν επιβάλλουμε έναν περιορισμό στην μορφή των κανόνων :

**8.5.1 Ορισμός.** Ένα αυτόματο ονομάζεται *αιτιοκρατικό* ή (πλήρως) *αιτιοκρατικό πεπερασμένο αυτόματο δένδρων που λειτουργεί από τα κάτω προς τα πάνω* (bottom-up deterministic finite tree automaton ή για συντομία BUDFTA), όταν δεν υπάρχουν δύο κανόνες στο σύνολο μετάβασης με το ίδιο αριστερό μέρος, ούτε υπάρχουν $\varepsilon$-κανόνες.

*8.5.2 Παρατήρηση.* Από τον παραπάνω ορισμό συμπεραίνουμε ότι μπορούμε στην περίπτωση των αιτιοκρατικών αυτομάτων να μιλάμε για συνάρτηση μετάβασης $\delta$, αντί για μία γενική σχέση $\Delta$.

Αν δούμε την έννοια της πληρότητας (ορισμός 8.4.1) σε σχέση με τα αιτιοκρατικά αυτόματα, προκύπτει άμεσα το παρακάτω :

**8.5.3 Πόρισμα.** *Σε ένα πλήρες αιτιοκρατικό αυτόματο, υπάρχει ακριβώς μία ολοκληρωμένη εκτέλεση για κάθε θεμελιώδη όρο.*

Στην περίπτωση των γλωσσών συμβολοσειρών είναι γνωστό ότι τα αιτιοκρατικά και τα μη αιτιοκρατικά αυτόματα έχουν την ίδια υπολογιστική δύναμη. Το ίδιο ισχύει (για τα αυτόματα που λειτουργούν από τα κάτω προς τα πάνω) και στην περίπτωση των δένδρων :

**8.5.4 Θεώρημα** (Ισοδυναμία BUNFTA και BUDFTA)**.** *Για κάθε αναγνωρίσιμη (από BUNFTA) γλώσσα υπάρχει αιτιοκρατικό αυτόματο (BUDFTA) που την αναγνωρίζει.*

*Απόδειξη.* Έστω μη αιτιοκρατικό αυτόματο $\mathcal{A} = (\mathcal{F}, Q, Q_{\mathrm{f}}, \Delta)$. Κατασκευάζουμε αιτιοκρατικό αυτόματο $\mathcal{A}_{\mathrm{D}} = (\mathcal{F}, Q_{\mathrm{D}}, Q_{f\mathrm{D}}, \Delta_{\mathrm{D}})$ που αναγνωρίζει την ίδια ακριβώς γλώσσα με το $\mathcal{A}$, βασιζόμενοι στην λεγόμενη «κατασκευή υποσυνόλου» (subset construction) : Θέτουμε $Q_{\mathrm{D}} = \mathcal{P}(Q)$ (δηλαδή οι καταστάσεις του $\mathcal{A}_{\mathrm{D}}$ είναι τα υποσύνολα από καταστάσεις του $\mathcal{A}$). Τις καταστάσεις του αυτομάτου $\mathcal{A}_{\mathrm{D}}$ συμβολίζουμε με $s$, $s_1$ κ.τ.λ. Το σύνολο $\Delta_{\mathrm{D}}$ αποτελείται από τους κανόνες $f(s_1, \ldots, s_n) \to s$ (και μόνον) για τους οποίους ισχύει

$$s = \{q \in Q \mid \exists q_1 \in s_1, \ldots, \exists q_n \in s_n \colon f(q_1, \ldots, q_n) \to q \in \Delta\}.$$

Τέλος, το σύνολο $Q_{f\mathrm{D}}$ των τελικών καταστάσεων του $\mathcal{A}_{\mathrm{D}}$ περιέχει όλα τα υποσύνολα καταστάσεων που περιέχουν τουλάχιστον μία τελική κατάσταση του αυτομάτου $\mathcal{A}$.

Εύκολα μπορούμε να δείξουμε πλέον ότι τα δύο αυτόματα αναγνωρίζουν την ίδια γλώσσα. □

Με την παραπάνω κατασκευή το πλήθος των καταστάσεων του αιτιοκρατικού αυτομάτου αυξάνεται εκθετικά σε σχέση με το πλήθος των καταστάσεων του μη αιτιοκρατικού αυτομάτου. Ενδέχεται όμως πολλές από τις καταστάσεις-σύνολα να μην είναι προσβάσιμες, οπότε να μπορούμε να τις καταργήσουμε. Δυστυχώς, αυτό δεν συμβαίνει σε όλες τις περιπτώσεις μη αιτιοκρατικών αυτομάτων, αφού προκύπτει εκθετικό πλήθος προσβασίμων καταστάσεων για το ισοδύναμο αιτιοκρατικό αυτόματο, όπως φαίνεται από το παρακάτω παράδειγμα (όμοιο παράδειγμα χρησιμοποιείται και στις γλώσσες συμβολοσειρών) :

**8.5.5 Παράδειγμα.** Έστω $\mathcal{F} = \{a(), b(), \#\}$ και

$$L = \{t \in \mathcal{T}(\mathcal{F}) \mid \mathrm{root}(t|_{1^n}) = a\}.$$

Προφανώς οι όροι επί του $\mathcal{F}$ είναι μονοδιάστατοι και $\mathrm{root}(t|_{1^n})$ είναι το σύμβολο που απέχει $n$ θέσεις από την ρίζα του εκάστοτε όρου $t$.

Το αυτόματο $\mathcal{A} = (\mathcal{F}, Q, Q_{\mathrm{f}}, \Delta)$ με σύνολο $Q = \{q, q_1, \ldots, q_{n+1}\}$, μια τελική κατάσταση : $Q_{\mathrm{f}} = \{q_{n+1}\}$ και σχέση μετάβασης :

$$\begin{aligned}
\Delta = \{ \quad \# &\to q, & a(q) &\to q,\\
b(q) &\to q, & a(q) &\to q_1,\\
b(q_1) &\to q_2, & a(q_1) &\to q_2,\\
&\ldots\\
b(q_n) &\to q_{n+1}, & a(q_n) &\to q_{n+1} \quad \}
\end{aligned}$$

δεν είναι αιτιοκρατικό, αναγνωρίζει την $L$ και έχει $n+2$ καταστάσεις. Διαισθητικά, ένα αιτιοκρατικό αυτόματο που αναγνωρίζει την $L$ θα πρέπει οπωσδήποτε κατά την λειτουργία του να αποθηκεύει τα $n+1$ τελευταία σύμβολα, οπότε θα έχει τουλάχιστον $2^{n+1}$ καταστάσεις. Για μία τυπική απόδειξη ότι δεν μπορεί να κατασκευαστεί μικρότερο BUDFTA, μπορεί, για παράδειγμα, να χρησιμοποιηθεί το θεώρημα 8.8.4, σελίδα 93.

Αντί βέβαια να θεωρήσουμε όλα τα υποσύνολα του $\mathcal{P}(Q)$ εξ αρχής στο σύνολο καταστάσεων του αιτιοκρατικού αυτομάτου, είναι δυνατόν να φτιάξουμε το αιτιοκρατικό αυτόματο προσθέτοντας σταδιακά τις προσβάσιμες μόνον καταστάσεις-σύνολα. Αυτό μπορεί να γίνει προσθέτοντας μόνον τις καταστάσεις-σύνολα $s$ της απόδειξης του θεωρήματος 8.5.4, μέχρι να μην είναι δυνατόν να προστεθεί άλλη κατάσταση-σύνολο.

# 8.6 Ιδιότητες κλεισίματος

## Πράξεις μεταξύ συνόλων

Αν δούμε τις αναγνωρίσιμες γλώσσες ως σύνολα και θεωρήσουμε τις πράξεις μεταξύ συνόλων, μεγάλο ενδιαφέρον παρουσιάζει αν αυτές το σύνολο των αναγνωρίσιμων γλωσσών είναι κλειστό ως προς αυτές τις πράξεις.

Μιλούμε για γλώσσες ορισμένες σε ένα κοινό αλφάβητο συναρτησιακών συμβόλων, έστω $\mathcal{F}$. Το *συμπλήρωμα* μίας γλώσσας $L \subseteq \mathcal{T}(\mathcal{F})$ είναι η γλώσσα $\overline{L} = \mathcal{T}(\mathcal{F}) - L$.

Έχουμε το παρακάτω βασικό θεώρημα :

**8.6.1 Θεώρημα.** *Το σύνολο των αναγνωρίσιμων γλωσσών δένδρων είναι κλειστό ως προς την ένωση, το συμπλήρωμα και την τομή.*

*Απόδειξη.*

Έστω οι αναγνωρίσιμες γλώσσες $L_1$, $L_2$ που αναγνωρίζονται από τα αυτόματα $\mathcal{A}_1 = (Q_1, \mathcal{F}, Q_{f1}, \Delta_1)$, $\mathcal{A}_2 = (Q_2, \mathcal{F}, Q_{f2}, \Delta_2)$ αντιστοίχως. Μπορούμε να υποθέσουμε ότι $Q_1 \cap Q_2 = \emptyset$ (αν δεν συμβαίνει αυτό, μπορούμε πάντοτε απλώς να μετονομάσουμε καταλλήλως τις καταστάσεις του ενός αυτομάτου, χωρίς να αλλοιώσουμε την γλώσσα που αυτό αναγνωρίζει). Η ένωση $L_1 \cup L_2$ αναγνωρίζεται από το αυτόματο

$$\mathcal{A} = (Q_1 \cup Q_2, \mathcal{F}, Q_{f1} \cup Q_{f2}, \Delta_1 \cup \Delta_2).$$

Για το συμπλήρωμα $\overline{L}$, μετατρέπουμε πρώτα το αυτόματο που αναγνωρίζει την $L$ σε πλήρες και αιτιοκρατικό και μετά αρκεί να εναλλάξουμε τις τελικές με τις μη τελικές καταστάσεις.

Για την τομή ισχύει $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ οπότε η απόδειξη ανάγεται στα προηγούμενα. □

## Ομομορφισμοί δένδρων

Ως γνωστόν, οι κανονικές γλώσσες συμβολοσειρών είναι κλειστές ως προς τους ομομορφισμούς. Σε αυτήν την ενότητα θα ορίσουμε την έννοια του ομομορφισμού στα δένδρα και θα δούμε αντίστοιχες ιδιότητες κλεισίματος.

Στους ομομορφισμούς συμβολοσειρών, κάθε σύμβολο από το αρχικό αλφάβητο $\Sigma$ αντιστοιχίζεται σε μία συμβολοσειρά ενός άλλου αλφαβήτου $\Sigma'$.

Αντιστοίχως, στους ομομορφισμούς δένδρων θα θεωρούμε ένα αρχικό συναρτησιακό αλφάβητο $\mathcal{F}$ και θα αντιστοιχίζουμε κάθε συναρτησιακό του σύμβολο σε έναν όρο επί ενός άλλου αλφαβήτου, έστω $\mathcal{F}'$.[2] Όμως, η κατάσταση

2. Τα δύο αλφάβητα $\mathcal{F}$ και $\mathcal{F}'$ δεν απαγορεύεται να έχουν και κάποια κοινά συναρτησιακά σύμβολα.

δεν είναι τόσο απλή όσο στις συμβολοσειρές, επειδή το αρχικό σύμβολο, πιθανώς έχει κάποια ορίσματα, τα οποία θα πρέπει, επίσης να τοποθετηθούν με κάποιον τρόπο στον νέο όρο. Τυπικά, έχουμε :

**8.6.2 Ορισμός** (Ομομορφισμός)**.** Ένας *ομομορφισμός δένδρων* είναι μία απεικόνιση $h_{\mathcal{F}}$ που αντιστοιχίζει σε κάθε συναρτησιακό σύμβολο $f \in \mathcal{F}_n$ (δηλαδή με $n$ ορίσματα) έναν όρο του $\mathcal{T}(\mathcal{F}', \mathcal{X}_n)$, όπου $X_n = \{x_1, \ldots, x_n\}$,[3] δηλαδή $n$ μεταβλητές, όσες και τα ορίσματα του συναρτησιακού συμβόλου $f$.

Αυτήν την απεικόνιση την επεκτείνουμε σε μία απεικόνιση $h\colon \mathcal{T}(\mathcal{F}) \to \mathcal{T}(\mathcal{F}')$ (δηλαδή από όρους επί του $\mathcal{F}$ σε όρους επί του $\mathcal{F}'$) ως εξής :

- $h(c) = h_{\mathcal{F}}(c)$, αν $c$ : σταθερά,[4]
- $h(f(t_1, \ldots, t_n)) = h_{\mathcal{F}}(f)^{\{x_1 := h(t_1), \ldots, x_n := h(t_n)\}}$, δηλαδή εφαρμόζουμε την αντικατάσταση (βλέπε ορισμό 4.4.3 στην σελίδα 41) $\{x_1 := h(t_1), \ldots, x_n := h(t_n)\}$ στον όρο $h_{\mathcal{F}}(f) \in \mathcal{T}(\mathcal{F}', \mathcal{X}_n)$.

Κάνουμε άλλη μία επέκταση :

**8.6.3 Ορισμός** (Ομομορφισμός για γλώσσα δένδρων)**.** Ένας ομομορφισμός $h$ επί όρων/δένδρων επεκτείνεται σε γλώσσες δένδρων ως εξής :

$$h(L) = \bigcup_{t \in L} h(t)$$

**8.6.4 Παράδειγμα.** Αν δούμε λογικές παραστάσεις (boolean expressions) ως όρους επί του αλφαβήτου $\mathcal{F} = \{and(,), or(,), not(), true, false\}$ μπορούμε με τον ομομορφισμό που απεικονίζει το $and$ ως εξής :

$$h_{\mathcal{F}}(and) = not(or(not(x_1), not(x_2)))$$

και όλα τα υπόλοιπα σύμβολα στην θέση τους, να μετατρέψουμε κάθε λογική παράσταση σε μία ισοδύναμή της χωρίς το $and$, δηλαδή επί του αλφαβήτου $\mathcal{F}' = \{or(,), not(), true, false\}$.

Επίσης, έχει νόημα να θεωρήσουμε την αντίστροφη διαδικασία του ομομορφισμού :

**8.6.5 Ορισμός** (Αντίστροφος ομομορφισμός)**.** Έστω ομομορφισμός $h$ επί γλωσσών. Ορίζουμε και συμβολίζουμε τον *αντίστροφό* του ως εξής :

$$h^{-1}(L) = \{t \mid h(t) \in L\}$$

Οι γλώσσες συμβολοσειρών είναι κλειστές ως προς ομομορφισμούς (συμβολοσειρών). Δεν ισχύει, όμως, το ίδιο για τις γλώσσες δένδρων, όπως φαίνεται από το παρακάτω αντιπαράδειγμα :

3. Για $n = 0$, προφανώς ορίζουμε $\mathcal{X}_0 = \emptyset$.
4. Προφανώς, ο όρος $h_{\mathcal{F}}(c)$ είναι θεμελιώδης όρος.

**8.6.6 Παράδειγμα.** Έστω $\mathcal{F} = \{d(), g(), a\}$, $\mathcal{F}' = \{f(,), g(), a\}$. Θεωρούμε τον ομομορφισμό για τον οποίο $h_{\mathcal{F}}(d) = f(x_1, x_1)$, ενώ δεν μεταβάλλει τα υπόλοιπα σύμβολα ($g$ και $a$). Ουσιαστικά ο ομομορφισμός στο δένδρο αντικαθιστά όπου $d$ το $f$ και διπλασιάζει το όρισμα του $d$. Αν θεωρήσουμε την γλώσσα $L = \{d(g^i(a)) \mid i \geq 0\}$, η οποία εύκολα αποδεικνύεται ότι είναι αναγνωρίσιμη, τότε $h(L) = \{f(g^i(a), g^i(a)) \mid i \geq 0\}$. Όμως, η $h(L)$ δεν είναι αναγνωρίσιμη (βλέπε ενότητα για το λήμμα μετάντλησης· μάλιστα, η $h(L)$ είναι η γλώσσα του παραδείγματος 8.7.1).

Παρ' όλα αυτά, οι αναγνωρίσιμες γλώσσες δένδρων είναι κλειστές ως προς μία ειδική περίπτωση ομομορφισμών, τους λεγόμενους γραμμικούς :

**8.6.7 Ορισμός.** Ένας ομομορφισμός $h_{\mathcal{F}}$ λέγεται *γραμμικός* αν για κάθε συναρτησιακό σύμβολο $f \in \mathcal{F}$ ο όρος $h_{\mathcal{F}}(f)$ είναι γραμμικός (βλέπε ορισμό 4.1.5, σελίδα 34).

**8.6.8 Θεώρημα.** *Έστω $h$ γραμμικός ομομορφισμός δένδρων και $L$ αναγνωρίσιμη γλώσσα δένδρων, τότε η $h(L)$ είναι αναγνωρίσιμη.*

*Απόδειξη.* Βλέπε [20]. □

Παρατηρούμε εδώ λοιπόν μία διαφορά σε σχέση με την αναγνωρισιμότητα στα δένδρα και την κανονικότητα στις συμβολοσειρές. Αυτή η διαφορά οφείλεται στον ορισμό των ομομορφισμών δένδρων, η οποία επιτρέπει την εμφάνιση μη γραμμικών όρων, οι οποίοι «καταστρέφουν» την αναγνωρισιμότητα.

Επιπλέον οι αναγνωρίσιμες γλώσσες είναι κλειστές ως προς οποιονδήποτε αντίστροφο ομομορφισμό :

**8.6.9 Θεώρημα.** *Έστω $h$ ομομορφισμός δένδρων και $L$ αναγνωρίσιμη γλώσσα δένδρων, τότε η $h^{-1}(L)$ είναι αναγνωρίσιμη.*

*Απόδειξη.* Βλέπε [20]. □

Μάλιστα, ισχύει το παρακάτω :

**8.6.10 Πρόταση.** *Η κλάση των αναγνωρίσιμων γλωσσών είναι η μικρότερη κλάση γλωσσών δένδρων που είναι κλειστή ως προς γραμμικούς ομομορφισμούς και αντίστροφους ομομορφισμούς.*

## 8.7 Λήμμα μετάντλησης (pumping)

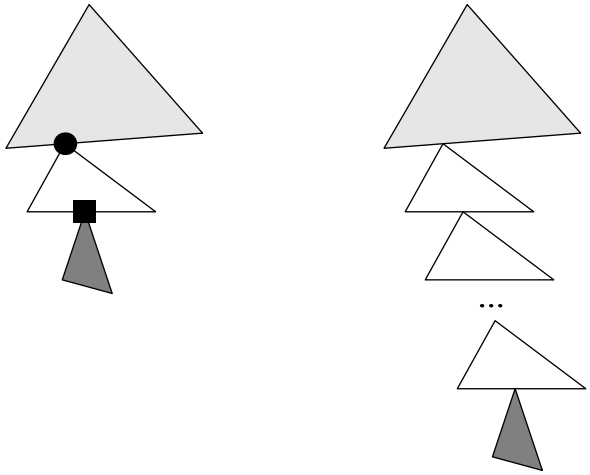
Υπάρχουν γλώσσες δένδρων, οι οποίες δεν είναι δυνατόν να αναγνωριστούν από κάποιο πεπερασμένο αυτόματο. Ένα χαρακτηριστικό παράδειγμα είναι οι γλώσσες στις οποίες απαιτούμε τα παιδιά ενός κόμβου να είναι ίσα (να πρόκειται δηλαδή για τον ίδιο όρο), όπου το μέγεθος των παιδιών δεν είναι φραγμένο. Διαισθητικά, μία τέτοια γλώσσα δεν είναι δυνατόν να αναγνωριστεί δεδομένου ότι ένα πεπερασμένο αυτόματο, λόγω του αναγκαστικά πεπερασμένου πλήθους των καταστάσεών του, μπορεί να «αποθηκεύσει» μόνον φραγμένου μήκους πληροφορία. Για να γίνει όμως έλεγχος της ισότητας δύο μη φραγμένων όρων, απαιτείται μη φραγμένος χώρος. Ας δούμε ένα συγκεκριμένο παράδειγμα αυτής της περίπτωσης, μαζί με μία τυπική απόδειξη :

**8.7.1 Παράδειγμα.** Έστω η γλώσσα $L = \{f(g^i(a), g^i(a)) \mid i \in \mathbb{N}\}$. Ας υποθέσουμε ότι υπάρχει αυτόματο που αναγνωρίζει την γλώσσα και έστω ότι διαθέτει $k$ καταστάσεις. Θεωρούμε τον όρο $t = f(g^k(a), g^k(a))$ και μία επιτυχή εκτέλεση του αυτομάτου επί αυτού του όρου. Σε αυτήν την εκτέλεση του αυτομάτου, υπάρχουν $k+1$ θέσεις στον αριστερό υποόρο $g^k(a)$, επομένως, κατά την επιτυχή εκτέλεση, δύο διαφορετικές θέσεις του $g^k(a)$, έστω $p_1$, $p_2$ με $p_1 < p_2$, επιγράφονται με την ίδια κατάσταση. Αν τώρα, τοποθετήσουμε στην θέση $p_1$ των υποόρο που βρίσκεται στην θέση $p_2$, προκύπτει ένας όρος μικρότερος του $g^k(a)$, δηλαδή $g^m(a) = g^k(a)[g^k(a)|_{p_2}]_{p_1}$, με $m < k$, τέτοιος ώστε υπάρχει επιτυχής εκτέλεση για τον $t' = f(g^m(a), g^k(a))$. Αυτό σημαίνει ότι και ο όρος $t'$ αναγνωρίζεται από το αυτόματο, κάτι που είναι άτοπο, αφού $m < k$ και το αυτόματο υποτίθεται ότι αναγνωρίζει την γλώσσα $L$.

Μπορούμε, γενικεύοντας την παραπάνω απόδειξη, να δώσουμε ένα χαρακτηριστικό των αναγνωρίσιμων γλωσσών. Το αποτέλεσμα είναι το λεγόμενο λήμμα μετάντλησης, το οποίο είναι αρκετά χρήσιμο για να αποδεικνύουμε ότι μία γλώσσα δένδρων δεν είναι αναγνωρίσιμη.

**8.7.2 Λήμμα** (Λήμμα μετάντλησης (pumping lemma))**.** *Έστω αναγνωρίσιμη γλώσσα δένδρων $L$. Τότε υπάρχει σταθερά $k$, τέτοια ώστε για κάθε $t \in L$ με* $\mathrm{Height}(t) > k$*, να υπάρχουν περιβάλλοντα $C$, $C'$, εκ των οποίων το $C'$ μη τετριμμένο, και όρος $u$, για τα οποία $t = C[C'[u]]$ και για κάθε $n \geq 0$ ισχύει $C[C'^n[u]] \in L$.*

*Απόδειξη.* Η κανονική γλώσσα $L$ αναγνωρίζεται από πεπερασμένο αυτόματο, το οποίο υποθέτουμε ότι έχει $k$ καταστάσεις. Για οποιονδήποτε όρο $t \in L$ ο οποίος έχει ύψος μεγαλύτερο από $k$ υπάρχει μία τουλάχιστον διαδρομή από την ρίζα σε κάποιο φύλλο, η οποία έχει μήκος μεγαλύτερο από $k$. Αν θεωρήσουμε τους κόμβους αυτής της διαδρομής, δεδομένου ότι σε κάθε επιτυχή εκτέλεση




Σχήμα 8.1 – Λήμμα μετάντλησης (pumping lemma)

το πεπερασμένο αυτόματο αποδίδει σε αυτούς κάποια κατάσταση, οπωσδήποτε μία κατάσταση επαναλαμβάνεται, την οποία έστω ότι ονομάζουμε $q$.

Ας υποθέσουμε ότι έχουμε την κατάσταση $q$ στις θέσεις $p_1, p_2$ με $p_1 < p_2$, οπότε $p_1.\Delta p_{12} = p_2$, για την θέση $\Delta p_{12} \neq \varepsilon$. Τότε θέτοντας

$$C = t[\Box]_{p_1}, \qquad C' = t|_{p_1}[\Box]_{\Delta p_{12}}, \qquad u = t|_{p_2}$$

έχουμε $t = C[C'[u]]$. Τώρα για κάθε κάθε $n \geq 0$, ο όρος $C[C'^n[u]] \in L$, επειδή υπάρχει επιτυχής εκτέλεση, στην οποία η κατάσταση $q$ αποδίδεται σε κάθε μία από τις θέσεις $p_1.(\Delta p_{12})^i$, για κάθε $i$ με $0 \leq i \leq n$. Βλέπε και σχήμα 8.1, όπου ο όρος $t = C[C'[u]]$ είναι αριστερά και ο όρος $C[C'^n[u]]$ δεξιά, ενώ οι θέσεις $p_1$ και $p_2$ του $t$ σημειώνονται με κουκκίδα και τετράγωνο αντιστοίχως. □

Μία κλασική εφαρμογή στα αυτόματα συμβολοσειρών είναι ότι γλώσσες στις οποίες απαιτούμε κάποιο πλήθος συμβόλων να είναι πρώτος αριθμός δεν είναι κανονικές. Θα δώσουμε ένα παρόμοιο παράδειγμα στην περίπτωση των δένδρων :

**8.7.3 Παράδειγμα.** Έστω το αλφάβητο $\mathcal{F} = \{f(,), a\}$ και η γλώσσα δένδρων $L = \{t \in \mathcal{T}(\mathcal{F}) \mid |\mathrm{Pos}(t)| \text{ είναι πρώτος}\}$. Ας υποθέσουμε ότι η $L$ είναι αναγνωρίσιμη. Τότε, από το λήμμα μετάντλησης, για κάποιο $k$, οποιοσδήποτε όρος $t \in L$ με ύψος μεγαλύτερο από $k$ γράφεται $t = C[C'[u]]$, ενώ αν το πλήθος των θέσεων που έχει καθένας από τους $t$, $C$, $C'$, $u$ (για τα περιβάλλοντα δεν υπολογίζουμε την θέση της οπής) είναι $n_t$, $n_C$, $n_{C'}$, $n_u$, αντιστοίχως, τότε το $n_t = n_C + n_{C'} + n_u$ είναι πρώτος αριθμός και επιπλέον $n_{C'} > 0$ (αφού το περιβάλλον $C'$ δεν είναι τετριμμένο). Όμως, από το λήμμα μετάντλησης, για κάθε $n \in \mathbb{N} : C[C'^n[u]] \in L$ και αν διαλέξουμε $n = n_C + n_u$ προκύπτει όρος με

πλήθος θέσεων $n_C + (n_C + n_u)n_{C'} + n_u = (n_C + n_u)(n_{C'} + 1)$, που προφανώς δεν είναι πρώτος αριθμός αριθμός· άτοπο.

*8.7.4 Παρατήρηση.* Παρατηρούμε ότι η σταθερά $k$ του λήμματος μετάντλησης σχετίζεται με το πλήθος $|Q|$ των καταστάσεων του αυτομάτου. Τέλος, εύκολα μπορεί να δει κανείς ότι για αυτόματο $\mathcal{A} = (\mathcal{F}, Q, Q_{\mathrm{f}}, \Delta)$, η γλώσσα $L(\mathcal{A})$ είναι μη κενή αν και μόνον αν υπάρχει όρος $t \in L(\mathcal{A})$ με $\mathrm{Height}(t) \leq |Q|$. Παρομοίως, η $L(\mathcal{A})$ είναι άπειρη αν και μόνον αν υπάρχει όρος $t \in L(\mathcal{A})$ τέτοιος ώστε $|Q| < \mathrm{Height}(t) \leq 2|Q|$.

## 8.8 Θεώρημα Myhill-Nerode

Το θεώρημα Myhill-Nerode (βλέπε [74, 76]) είναι ένα σημαντικό εργαλείο για την απόδειξη ύπαρξης ελαχίστων αιτιοκρατικών αυτομάτων στην περίπτωση των κανονικών γλωσσών συμβολοσειρών. Βασίζεται στην κλειστότητα ως προς την συνένωση από τα δεξιά σχέσεων ισοδυναμίας που αφορούν συμβολοσειρές και οι οποίες σχέσεις για αυτό χαρακτηρίζονται ως *δεξιά αναλλοίωτες* (right invariant· βλέπε και [51]).

Σε αυτήν την ενότητα θα διατυπώσουμε ένα αντίστοιχο θεώρημα για τις γλώσσες δένδρων. Πρώτα όμως θα πρέπει να ορίσουμε μία αντίστοιχη έννοια ισοδυναμίας με μία προσαρμοσμένη ιδιότητα. Τέτοιες ισοδυναμίες τις ονομάζουμε *ισοτιμίες* στα πλαίσια των γλωσσών δένδρων.

**8.8.1 Ορισμός.** Έστω το σύνολο όρων $\mathcal{T}(\mathcal{F})$ και μία σχέση ισοδυναμίας $\equiv$ επί αυτού. Η $\equiv$ ονομάζεται *ισοτιμία* (congruence) αν είναι κλειστή ως προς τα περιβάλλοντα, δηλαδή αν για κάθε περιβάλλον $C$ ισχύει : Αν για τους όρους $s$, $t$ έχουμε $s \equiv t$, τότε $C[s] \equiv C[t]$.

Μπορεί εύκολα να αποδειχθεί ότι ο παραπάνω ορισμός είναι ισοδύναμος με τον εξής : Αν για κάθε $n$ και για κάθε $f \in \mathcal{F}_n$

$$s_1 \equiv t_1, \ldots, s_n \equiv t_n \quad \text{συνεπάγεται} \quad f(s_1, \ldots, s_n) \equiv f(t_1, \ldots, t_n)$$

τότε αν η $\equiv$ είναι σχέση ισοδυναμίας ονομάζεται ισοτιμία.

**8.8.2 Ορισμός.** Λέμε ότι μία ισοτιμία είναι *πεπερασμένου δείκτη* (of finite index) αν τα σύνολα ισοδυναμίας της είναι πεπερασμένα το πλήθος.

Για κάθε γλώσσα δένδρων $L$ μπορούμε να ορίσουμε μία ισοτιμία ως εξής :

**8.8.3 Ορισμός.** Η *ισοτιμία που αντιστοιχεί στην γλώσσα* $L$ συμβολίζεται με $\equiv_L$ και ορίζεται ως εξής $s \equiv_L t$ αν και μόνον αν για κάθε περιβάλλον $C$ : $C[s] \in L$ αν και μόνον αν $C[t] \in L$.

Μένει να δώσουμε το θεώρημα :

**8.8.4 Θεώρημα** (Myhill-Nerode)**.** *Τα παρακάτω είναι ισοδύναμα για μία γλώσσα $L$ :*

(i) *η $L$ είναι αναγνωρίσιμη·*

(ii) *η $L$ είναι ένωση κάποιων συνόλων ισοδυναμίας μίας ισοτιμίας πεπερασμένου δείκτη·*

(iii) *η σχέση $\equiv_L$ είναι ισοτιμία πεπερασμένου δείκτη.*

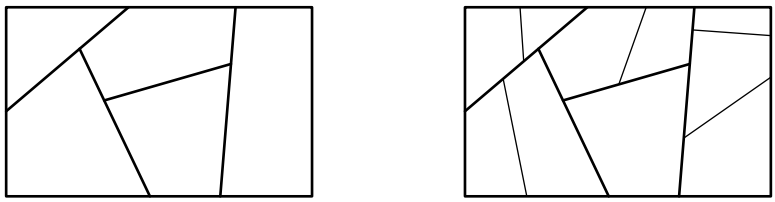
*Απόδειξη.*
(i) $\Longrightarrow$ (ii). Έστω πλήρες BUDFTA $\mathcal{A} = (\mathcal{F}, Q, Q_{\rm f}, \delta)$. Για κάθε όρο $t$ υπάρχει ακριβώς μία κατάσταση $q$ τέτοια ώστε $t \xrightarrow{*}_{\mathcal{A}} q$· με άλλα λόγια η $q$ είναι η κατάσταση που αποδίδεται από το αυτόματο $\mathcal{A}$ κατά την (μοναδική και ολοκληρωμένη) εκτέλεση στην ρίζα του όρου $t$. Ορίζουμε την $\equiv_{\mathcal{A}}$ επί του $s \equiv_{\mathcal{A}} t$ αν και μόνον αν για τους όρους $s$ και $t$ το αυτόματο αποδίδει την ίδια κατάσταση στην ρίζα τους. Η $\equiv_{\mathcal{A}}$ είναι προφανώς σχέση ισοδυναμίας και πολύ εύκολα προκύπτει ότι είναι και ισοτιμία. Επιπλέον είναι ισοτιμία πεπερασμένου δείκτη, δεδομένου ότι τα σύνολα ισοδυναμίας της είναι ακριβώς όσες και οι προσβάσιμες καταστάσεις του αυτομάτου (πεπερασμένες το πλήθος). Τελικά, η γλώσσα $L(\mathcal{A})$ είναι ένωση των συνόλων ισοδυναμίας της $\equiv_{\mathcal{A}}$ που αντιστοιχούν σε τελικές (και προσβάσιμες) καταστάσεις του αυτομάτου.

(ii) $\Longrightarrow$ (iii). Υπενθυμίζουμε τον συμβολισμό $t/\equiv$ για το σύνολο ισοδυναμίας του $t$ στην σχέση $\equiv$. Αρκεί να δείξουμε ότι για κάθε $t \in \mathcal{T}(\mathcal{F}) : t/\equiv_{\mathcal{A}} \subseteq t/\equiv_L$, οπότε $|\mathcal{T}(\mathcal{F})/\equiv_L| \leq |\mathcal{T}(\mathcal{F})/\equiv_{\mathcal{A}}|$. Πράγματι : $x \in t/\equiv_{\mathcal{A}}$, ή $x \equiv_{\mathcal{A}} t$ κι αφού η $\equiv_{\mathcal{A}}$ είναι σχέση ισοτιμίας για κάθε περιβάλλον $C : C[x] \equiv_{\mathcal{A}} C[t]$, δηλαδή οι $C[x]$, $C[t]$ αντιστοιχούν στην ίδια κατάσταση και έτσι συνεπάγεται ότι $C[x] \in L$ αν και μόνον αν $C[t] \in L$, δηλαδή $C[x] \equiv_L C[t]$ ή $x \in t/\equiv_L$.

(iii) $\Longrightarrow$ (i). Θα συμβολίζουμε, για συντομία το $\cdot/\equiv_L$ με $[\cdot]$. Κατασκευάζουμε πλήρες BUDFTA $\mathcal{A}_{\rm min} = (\mathcal{F}, Q, Q_{\rm f}, \delta)$ με $Q = [\mathcal{T}(\mathcal{F})]$, $Q_{\rm f} = \{[t] \mid t \in L\}$. Η συνάρτηση μετάβασης είναι $\delta(f([q_1], \ldots, [q_n])) = [q]$, αν $f(t_1, \ldots, t_n) \in q$, για κάποια $t_1 \in q_1$, ..., $t_n \in q_n$. Εύκολα πλέον μπορεί να δειχθεί ότι το $\mathcal{A}_{\rm min}$ αναγνωρίζει την $L$. □

*8.8.5 Παρατήρηση.* Εύκολα μπορεί να δει κανείς ότι το $\mathcal{A}_{\rm min}$ είναι και μειωμένο δεδομένου ότι αφού τα σύνολα ισοδυναμίας δεν είναι κενά υπάρχει πάντοτε για κάθε $q \in Q$ κάποιο $t$ με $t \xrightarrow{*}_{\mathcal{A}_{\rm min}} q$.

*8.8.6 Παρατήρηση.* Σε κάθε περίπτωση, η σχέση $\equiv_{\mathcal{A}}$ αποτελεί *εκλέπτυνση* (refinement) της $\equiv_{L(\mathcal{A})}$, δηλαδή και οι δύο σχέσεις διαμερίζουν το $\mathcal{T}(\mathcal{F})$ και επιπλέον ένα σύνολο ισοδυναμίας της $\equiv_{\mathcal{A}}$ δεν μπορεί να περιέχει κοινά στοιχεία με δύο ή παραπάνω διαφορετικά σύνολα ισοδυναμίας της $\equiv_{L(\mathcal{A})}$ (βλέπε και σχήμα 8.2).




Σχήμα 8.2 – Εκλέπτυνση σχέσης ισοδυναμίας

Για περισσότερες πληροφορίες και ιστορικά στοιχεία σχετικά με το θεώρημα Myhill-Nerode για δένδρα, βλέπε [61].

## 8.9 Ελάχιστο πλήρες αιτιοκρατικό αυτόματο

Στην προηγούμενη ενότητα ονομάσαμε $\mathcal{A}_{\min}$ το αυτόματο που προκύπτει από την ισοτιμία $\equiv_L$. Το αυτόματο αυτό πράγματι είναι το ελάχιστο (minimum) ως προς το πλήθος των καταστάσεων που μπορεί να αναγνωρίσει την γλώσσα :

**8.9.1 Πρόταση.** *Δεν υπάρχει πλήρες αιτιοκρατικό αυτόματο $\mathcal{A}$ που αναγνωρίζει την $L$ και έχει λιγότερες καταστάσεις από το $\mathcal{A}_{min}$.*

*Απόδειξη.* Από την απόδειξη του θεωρήματος Myhill-Nerode έχουμε ότι για κάθε αυτόματο $\mathcal{A}$ με $L = L(\mathcal{A})$ ισχύει $|\mathcal{T}(\mathcal{F})/\equiv_L| \leq |\mathcal{T}(\mathcal{F})/\equiv_{\mathcal{A}}|$, δηλαδή το πλήθος των προσβασίμων καταστάσεων του $\mathcal{A}$ δεν μπορεί να είναι μικρότερο από το πλήθος των καταστάσεων του $\mathcal{A}_{\min}$. Επιπλέον, από την παρατήρηση 8.8.5, το $\mathcal{A}_{\min}$ είναι μειωμένο, οπότε δεν επιδέχεται περαιτέρω μείωση καταστάσεων. □

Επίσης, το $\mathcal{A}_{\min}$ είναι στην ουσία και μοναδικό :

**8.9.2 Πρόταση.** *Το αυτόματο $\mathcal{A}_{min}$ είναι μοναδικό μέχρι μετονομασίας των καταστάσεων.*

*Απόδειξη.* Έστω αιτιοκρατικό αυτόματο $\mathcal{A}$ ισοδύναμο με το $\mathcal{A}_{\min}$ που έχει ίσο πλήθος καταστάσεων με το $\mathcal{A}_{\min}$. Οι καταστάσεις των δύο αυτομάτων αντιστοιχούν στα σύνολα ισοδυναμίας των $\equiv_{\mathcal{A}}$ και $\equiv_{L(\mathcal{A})}$. Αφού όμως η $\equiv_{\mathcal{A}}$ αποτελεί εκλέπτυνση της $\equiv_{L(\mathcal{A})}$ και έχουν τον ίδιο (πεπερασμένο) δείκτη, διαμερίζουν το $\mathcal{T}(\mathcal{F})$ σε ακριβώς τα ίδια σύνολα ισοδυναμίας. □

Μένει να περιγράψουμε πλέον έναν αλγόριθμο για την μετατροπή ενός (πλήρους) BUDFTA στο ελάχιστο πλήρες BUDFTA. Ο αλγόριθμος είναι παρόμοιος με την περίπτωση των αυτομάτων συμβολοσειρών, αφού ουσιαστικά έγκειται στην αναζήτηση καταστάσεων του αυτομάτου που μπορούν να ενοποιηθούν σε μία κατάσταση του ελαχίστου BUDFTA.

Αν δύο καταστάσεις δεν μπορούν να ενοποιηθούν, τότε τις ονομάζουμε διακρίσιμες (distinguishable). Δύο καταστάσεις $q_1$, $q_2$ είναι *διακρίσιμες* (αντίστοιχα λέμε ότι το μη διατεταγμένο ζεύγος καταστάσεων $\{q_1, q_2\}$ είναι διακρίσιμο) αν υπάρχουν δύο κανόνες στο $\Delta$ τέτοιοι ώστε

$$f(\ldots, q_1, \ldots) \to q_x, \quad f(\ldots, q_2, \ldots) \to q_y,$$

για κάποιο $f \in \mathcal{F}$ και τα αριστερά μέλη των δύο κανόνων διαφέρουν μόνον ως προς τα $q_1$, $q_2$, ενώ οι καταστάσεις $q_1$, $q_2$ είναι διακρίσιμες. Προφανώς, ο παραπάνω ορισμός είναι αναδρομικός, οπότε χρειαζόμαστε και μία βάση για την αναδρομή μας : Αρχικά, χωρίζουμε τις καταστάσεις του αρχικού BUDFTA σε αρχικές και τελικές (μία αρχική και μία τελική κατάσταση δεν είναι δυνατόν να ενοποιηθούν, άρα είναι εκ των προτέρων γνωστό ότι είναι διακρίσιμες).

Ο αλγόριθμος λοιπόν λειτουργεί ως εξής : Θεωρούμε όλα τα δυνατά ζεύγη καταστάσεων. Σε πρώτη φάση χαρακτηρίζουμε ως διακρίσιμα τα ζεύγη στα οποία η μία κατάσταση είναι αρχική και η άλλη τελική. Κατόπιν επαναλαμβάνουμε την εξής φάση : Για όλα τα ζεύγη που δεν έχουν ακόμα χαρακτηριστεί ως διακρίσιμα ελέγχουμε αν υπάρχουν κάποιοι κανόνες στο $\Delta$ που να τα κάνουν διακρίσιμα (βάσει των ήδη χαρακτηρισμένων ως διακρισίμων ζευγών). Σταματάμε, αν σε κάποια επανάληψη της παραπάνω φάσης δεν χαρακτηριστεί κάποιο καινούριο ζεύγος ως διακρίσιμο (γιατί;). Τέλος, ενοποιούμε καταλλήλως τις καταστάσεις που δεν έχουν χαρακτηριστεί ως διακρίσιμες, τροποποιώντας ανάλογα και το $\Delta$, και έχουμε το ελάχιστο αυτόματο.

**8.9.3 Παράδειγμα.** Έστω $\mathcal{F} = \{f(,), g(), a\}$. Ένα αυτόματο που αναγνωρίζει την γλώσσα δένδρων $L = \{g^n(a) \mid n > 0\} \subset \mathcal{T}(\mathcal{F})$ και έχει τέσσερις καταστάσεις είναι το $\mathcal{A} = (\mathcal{F}, \{q_a, q_1, q_2, q_d\}, \{q_1, q_2\}, \Delta)$, όπου

$$\begin{aligned} \Delta = \{ \quad & a \to q_a, \quad g(q_a) \to q_1, \\ & g(q_1) \to q_2, \quad g(q_2) \to q_1, \quad g(q_d) \to q_d, \\ & f(q_a, q_a) \to q_d, \quad \ldots, \quad f(q_d, q_d) \to q_d \qquad \} \end{aligned}$$

(για οποιεσδήποτε καταστάσεις $x$, $y$ έχουμε $f(x, y) \to q_d$).

Εφαρμόζοντας τον αλγόριθμο ελαχιστοποίησης αρχικά διακρίνουμε τις καταστάσεις σε αρχικές και τελικές :

| | | | |
|---|---|---|---|
| $q_1$ | $\times$ | | |
| $q_2$ | $\times$ | | |
| $q_d$ | | $\times$ | $\times$ |
| | $q_a$ | $q_1$ | $q_2$ |

Στην επόμενη φάση, για το ζεύγος $\{q_a, q_d\}$, εξαιτίας των κανόνων :

$$g(q_a) \to q_1, \quad g(q_d) \to q_d$$

προκύπτει ότι είναι διακρίσιμο αφού οι $q_1$, $q_d$ έχουν ήδη χαρακτηριστεί ως διακρίσιμες, οπότε :

| | | | |
|---|---|---|---|
| $q_1$ | $\times$ | | |
| $q_2$ | $\times$ | | |
| $q_d$ | $\times$ | $\times$ | $\times$ |
| | $q_a$ | $q_1$ | $q_2$ |

Στην επανάληψη της φάσης, το μοναδικό εναπομείναν ζεύγος ($\{q_1, q_2\}$) δεν μπορεί να χαρακτηριστεί ως διακρίσιμο, οπότε ο αλγόριθμος τερματίζει δίνοντας το ελάχιστο αυτόματο $\mathcal{A}_{\min} = (\mathcal{F}, \{q_a, q_{12}, q_d\}, \{q_{12}\}, \Delta_{\min})$, όπου

$$\begin{aligned}\Delta_{\min} = \{ \quad & a \to q_a, \quad g(q_a) \to q_{12}, \quad g(q_{12}) \to q_{12}, \quad g(q_d) \to q_d, \\ & f(q_a, q_a) \to q_d, \quad \ldots, \quad f(q_d, q_d) \to q_d \qquad\qquad \}\end{aligned}$$

## 8.10 Πεπερασμένα αυτόματα top-down

Τα αυτόματα που μελετήσαμε ως τώρα, αρχίζουν την λειτουργία τους από τα φύλλα ενός δένδρου και αποδίδουν καταστάσεις στους κόμβους μέχρι να φτάσουν στην ρίζα, λειτουργούν, δηλαδή, από τα κάτω προς τα πάνω (bottom-up). Σε αυτήν την ενότητα θα ασχοληθούμε με αυτόματα που λειτουργούν *από τα πάνω προς τα κάτω* (top-down), δηλαδή αποδίδουν κατάσταση πρώτα στην ρίζα και προχωρούν προς τα κάτω μέχρι τα φύλλα.

**8.10.1 Ορισμός.** Ένα *πεπερασμένο μη αιτιοκρατικό αυτόματο που λειτουργεί από τα πάνω προς τα κάτω* (top-down non-deterministic finite tree automaton ή για συντομία TDNFTA) είναι μία πλειάδα $\mathcal{A} = (\mathcal{F}, Q, Q_{\rm i}, \Delta)$, όπου $\mathcal{F}$ : αλφάβητο συναρτησιακών συμβόλων, $Q$ : πεπερασμένο σύνολο καταστάσεων, $Q_{\rm i} \subseteq Q$ : το σύνολο των *αρχικών* καταστάσεων, και $\Delta$ : ένα σύνολο κανόνων μετάβασης της μορφής $q \to f(q_1, \ldots, q_n)$, με $n \geq 0$, $f \in \mathcal{F}_n$, $q, q_1, \ldots, q_n \in Q$.

Όπως και στην περίπτωση των αυτομάτων που λειτουργούν από τα κάτω προς τα πάνω, οι κανόνες μεταγραφής ορίζουν ένα θεμελιώδες σύστημα μεταγραφής όρων από το σύνολο $\mathcal{T}(\mathcal{F} \cup Q)$ και συνεκδοχικά μία σχέση μεταγραφής $\to_{\mathcal{A}}$ με ανακλαστικό μεταβατικό κλείσιμο $\overset{*}{\to}_{\mathcal{A}}$.

Λέμε ότι ο όρος $t$ γίνεται *αποδεκτός* από το αυτόματο $\mathcal{A}$ όταν $q \xrightarrow{*}_{\mathcal{A}} t$, για κάποια αρχική κατάσταση $q$ ($q \in Q_{\rm i}$). Παρομοίως, ορίζεται η γλώσσα που αναγνωρίζει ένα top-down αυτόματο, καθώς και η έννοια της εκτέλεσης σε έναν όρο/δένδρο (επιγραφή των κόμβων του δένδρου με καταστάσεις, συμβατή με κανόνες μετάβασης) και ειδικότερα της επιτυχούς εκτέλεσης (όταν η ρίζα του δένδρου επιγράφεται με αρχική κατάσταση).

Η υπολογιστική ισχύς των top-down αυτομάτων είναι ίδια με αυτήν των bottom-up, όπως επιβεβαιώνεται από την παρακάτω πρόταση :

**8.10.2 Πρόταση.** *Για κάθε αναγνωρίσιμη (από BUNFTA) γλώσσα, υπάρχει επίσης TDNFTA που την αναγνωρίζει.*

*Απόδειξη.* Αν δίδεται BUNFTA $\mathcal{A}_{\rm BU} = (\mathcal{F}, Q, Q_{\rm f}, \Delta)$, τότε κατασκευάζουμε TDNFTA $\mathcal{A}_{\rm TD} = (\mathcal{F}, Q, Q_{\rm i}, \Delta')$, με $Q_{\rm i} = Q_{\rm f}$ και το σύνολο κανόνων μετάβασης $\Delta'$ έχει τους κανόνες του $\Delta$ με ανεστραμμένα, όμως, βέλη (ή αλλιώς : το αριστερό μέλος του κάθε κανόνα του $\Delta$ γίνεται δεξιό στο $\Delta'$ και το αντίστροφο). Εύκολα πλέον μπορεί να αποδείξει κανείς ότι ένας όρος γίνεται αποδεκτός από το $\mathcal{A}_{\rm TD}$ αν και μόνον αν γίνεται αποδεκτός από το $\mathcal{A}_{\rm BU}$, άρα τα $\mathcal{A}_{\rm TD}$ και $\mathcal{A}_{\rm BU}$ είναι ισοδύναμα. □

Όπως και στα bottom-up αυτόματα, θα ορίσουμε τα αυστηρώς αιτιοκρατικά αυτόματα :

**8.10.3 Ορισμός.** Ένα top-down αυτόματο ονομάζεται *αιτιοκρατικό* (σύντομα TDDFTA), όταν έχει μόνον μία αρχική κατάσταση και δεν υπάρχουν δύο κανόνες με το ίδιο αριστερό μέρος.

Το πρόβλημα είναι ότι τα αιτιοκρατικά top-down αυτόματα είναι γνησίως ασθενέστερα ως προς τις δυνατότητες αναγνώρισης σε σχέση με τα bottom-up αυτόματα. Διαισθητικά, οι ιδιότητες ενός δένδρου που γίνεται αποδεκτό από ένα αιτιοκρατικό top-down αυτόματο μπορούν να εξαρτώνται μόνον από την μορφή των διαδρομών/θέσεων και όχι από το περιεχόμενο των υποόρων. Πράγματι, υπάρχει αναγνωρίσιμη γλώσσα που δεν γίνεται αποδεκτή από top-down αυτόματο : Έστω $\mathcal{F} = \{f(,), a\}$ και η γλώσσα $L = \{f(a,b), f(a,b)\}$ που είναι πεπερασμένη, άρα και αναγνωρίσιμη. Εύκολα μπορεί να δει κανείς ότι αν υπάρχει TDDFTA που αναγνωρίζει την $L$, τότε θα πρέπει να αποδέχεται και τον όρο $f(a,a)\cdot$ άτοπο.

## 8.11 Αποκρισιμότητα προβλημάτων

Λόγω των σχετικά περιορισμένων υπολογιστικών δυνατοτήτων των πεπερασμένων αυτομάτων δένδρων, τα σημαντικότερα προβλήματα που τα αφορούν είναι αποκρίσιμα.

Προφανώς, το πρόβλημα αν κάποιος όρος είναι *αποδεκτός* από ένα αυτόματο είναι γραμμικό στην περίπτωση ενός DFTA, ενώ πολυωνυμικό στην περίπτωση ενός NFTA.

Με την βοήθεια του λήμματος μετάντλησης μπορούμε να αποδείξουμε ότι είναι αποκρίσιμα τα προβλήματα που αφορούν αν μία αναγνωρίσιμη γλώσσα είναι *κενή*, *πεπερασμένη*, έχει *κενό συμπλήρωμα*, αλλά υπάρχουν και ταχύτεροι αλγόριθμοι, για κάθε περίπτωση.

Το πρόβλημα αν δύο πεπερασμένα αυτόματα δένδρων $\mathcal{A}_1$ και $\mathcal{A}_2$ είναι *ισοδύναμα* (δηλαδή αναγνωρίζουν την ίδια γλώσσα) είναι επίσης αποκρίσιμο, αν μετατρέψουμε τα αυτόματα σε αιτιοκρατικά και κατόπιν τα ελαχιστοποιήσουμε και τα συγκρίνουμε (βλέπε και πρόταση 8.9.2). Εναλλακτικά μπορούμε να κατασκευάσουμε το αυτόματο που αναγνωρίζει την συμμετρική διαφορά των δύο γλωσσών $L(\mathcal{A}_1)$, $L(\mathcal{A}_2)$ :

$$\left(L(\mathcal{A}_1) \cap \overline{L(\mathcal{A}_2)}\right) \cup \left(L(\mathcal{A}_2) \cap \overline{L(\mathcal{A}_1)}\right)$$

και να ελέγξουμε αν η γλώσσα αυτή είναι κενή.

# Κεφάλαιο 9

# Κανονικές γραμματικές δένδρων

Στο προηγούμενο κεφάλαιο, αφού είδαμε γενικά τις γλώσσες δένδρων, μελετήσαμε ιδιαιτέρως αυτές που αναγνωρίζονται από τα πεπερασμένα αυτόματα δένδρων. Στην περίπτωση των γλωσσών συμβολοσειρών, τα πεπερασμένα αυτόματα συνδέονται με τις κανονικές γραμματικές (είτε αριστερογραμμικές, είτε δεξιογραμμικές). Παρομοίως, στην περίπτωση των δένδρων, τα πεπερασμένα αυτόματα δένδρων, σχετίζονται με τις λεγόμενες κανονικές γραμματικές δένδρων.

Επίσης, σε αυτό το κεφάλαιο θα δείξουμε την σχέση των κανονικών γραμματικών δένδρων με τις γραμματικές χωρίς συμφραζόμενα συμβολοσειρών.

Το παρόν κεφάλαιο βασίζεται στο δεύτερο κεφάλαιο του [20].

## 9.1 Γραμματικές δένδρων

Αρχικά, θα δώσουμε τον γενικό ορισμό της γραμματικής δένδρων (tree grammar) :

**9.1.1 Ορισμός.** Μία *γραμματική δένδρων* είναι μία πλειάδα $(S, N, \mathcal{F}, P)$, όπου $S$ : το λεγόμενο *αξίωμα*, $N$ : το σύνολο των *μη τερματικών* συμβόλων, στο οποίο περιέχεται και το $S$, $\mathcal{F}$ : το σύνολο των *τερματικών* συμβόλων, $P$ : το σύνολο των *κανόνων παραγωγής*.

Οι *κανόνες παραγωγής* είναι της μορφής $\alpha \to \beta$, όπου $\alpha$, $\beta$ : δένδρα από το $\mathcal{T}(\mathcal{F} \cup N, \mathcal{V})$, όπου $\mathcal{V}$ ένα σύνολο μεταβλητών και το δένδρο $\alpha$ περιέχει τουλάχιστον ένα μη τερματικό σύμβολο. Επιπλέον, απαιτούμε να ισχύουν τα εξής : α) $\mathcal{F} \cap N = \emptyset$, β) κάθε στοιχείο του $\mathcal{F} \cup N$ να έχει σταθερό πλήθος ορισμάτων σε όλη την γραμματική, γ) το πλήθος ορισμάτων του αξιώματος $S$ είναι 0.

Οι γραμματικές του παραπάνω τύπου ονομάζονται και *γενικές* αφού δεν υπάρχει κανένας περιορισμός στην μορφή των κανόνων. Αυτές οι γραμματικές έχουν παρόμοιες δυνατότητες με τις γενικές γραμματικές συμβολοσειρών και για αυτό πολλά προβλήματα που τις αφορούν είναι μη αποκρίσιμα. Πιο πολύ ενδιαφέρον παρουσιάζουν γραμματικές στις οποίες επιβάλλουμε κάποιους περιορισμούς στην μορφή των κανόνων. Στην περίπτωση των δένδρων ορίζουμε συνήθως δύο κλάσεις γραμματικών : τις *κανονικές γραμματικές* και τις *γραμματικές χωρίς συμφραζόμενα*. Μάλιστα, υπάρχει αντίστοιχο θεώρημα ιεράρχησης με αυτό των γραμματικών συμβολοσειρών για τις τρεις κλάσεις γραμματικών δένδρων :

$$\text{κανονικές} \subset \text{χωρίς συμφραζόμενα} \subset \text{γενικές}$$

Με τις κανονικές γραμματικές δένδρων θα ασχοληθούμε στην επόμενη ενότητα, ενώ με τις γραμματικές δένδρων χωρίς συμφραζόμενα στην ενότητα 9.7, σελίδα 112.

## 9.2 Κανονικές γραμματικές δένδρων

Για να προκύψει η κλάση των *κανονικών γραμματικών δένδρων*, περιορίζουμε την μορφή των κανόνων ως εξής :

**9.2.1 Ορισμός.** Μία γραμματική δένδρων ονομάζεται *κανονική* όταν ισχύουν οι εξής περιορισμοί : α) όλα τα μη τερματικά σύμβολα έχουν πλήθος ορισμάτων 0 (είναι σταθερές) και β) όλοι οι κανόνες παραγωγής είναι της μορφής $A \to \beta$, όπου $\beta \in \mathcal{T}(\mathcal{F} \cup N)$ και το $A$ είναι μη τερματικό σύμβολο.

**9.2.2 Παράδειγμα.** Η γραμματική

$$G = (\text{List}, \{\text{List}, \text{Nat}\}, \{0, \text{nil}, s(), \text{cons}(,)\}, P)$$

όπου $P$ αποτελείται από τους :

$$\begin{aligned} &\text{List} \to \text{nil} \\ &\text{List} \to \text{cons}(\text{Nat}, \text{List}) \\ &\text{Nat} \to 0 \\ &\text{Nat} \to s(\text{Nat}) \end{aligned}$$

και περιγράφει λίστες φυσικών αριθμών ($s$ : συνάρτηση επομένου φυσικού αριθμού) με συμβολισμό τύπου LISP (cons).

Η έννοια της γραμματικής είναι χρήσιμη, γιατί περιγράφει μία παραγωγική διαδικασία όρων από το αξίωμα. Αυτή η παραγωγική διαδικασία περιγράφεται από την σχέση παραγωγής :

**9.2.3 Ορισμός.** Η *σχέση παραγωγής* (derivation relation) της κανονικής γραμματικής $G$ ορίζεται επί του $\mathcal{T}(\mathcal{F} \cup N)$ ως εξής : $\gamma \Rightarrow_G \delta$ αν και μόνον αν υπάρχει κανόνας παραγωγής στην $G : X \to \beta$ και περιβάλλον $C$, τέτοια ώστε $\gamma = C[X]$ και $\delta = C[\beta]$.

Για την γλώσσα που παράγεται από μία κανονική γραμματική αρκεί να περιοριστούμε σε όρους που περιέχουν μόνον τερματικά σύμβολα :

**9.2.4 Ορισμός.** Η γλώσσα δένδρων που *παράγεται* από την κανονική γραμματική $G$ συμβολίζεται με $L(G)$ και είναι το σύνολο :

$$L(G) \stackrel{\text{ορ.}}{=} \{t \in \mathcal{T}(\mathcal{F}) \mid S \stackrel{+}{\Rightarrow}_G t\}$$

όπου $S$ το αξίωμα της γραμματικής $G$.

**9.2.5 Παράδειγμα.** Έστω η γραμματική του παραδείγματος 9.2.2. Μία ακολουθία παραγωγών είναι η εξής :

$$\begin{aligned} \text{List} &\Rightarrow_G \text{cons}(\text{Nat}, \text{List}) \Rightarrow_G \text{cons}(s(\text{Nat}), \text{List}) \\ &\Rightarrow_G \text{cons}(s(0), \text{List}) \Rightarrow_G \text{cons}(s(0), \text{nil}) \end{aligned}$$

Γενικά, από την γραμματική είναι δυνατόν να παραχθούν όλες οι λίστες φυσικών αριθμών.

**9.2.6 Ορισμός.** *Κανονική γλώσσα δένδρων* (regular tree language) ονομάζεται μία γλώσσα αν είναι δυνατόν να παραχθεί από μία κανονική γραμματική δένδρων.

## 9.3 Απλοποίηση και κανονικοποίηση κανονικών γραμματικών

Είναι προφανές ότι η ίδια κανονική γλώσσα μπορεί να παράγεται από περισσότερες από δύο γραμματικές, διαφορετικές μεταξύ τους.

**9.3.1 Ορισμός** (Ισοδυναμία γραμματικών)**.** Δύο γραμματικές $G_1$, $G_2$ ονομάζονται *ισοδύναμες* όταν $L(G_1) = L(G_2)$.

Μία γραμματική $G$ ενδέχεται να έχει τερματικά σύμβολα τα οποία δεν εμφανίζονται ποτέ σε κανένα από τα δένδρα που παράγει η $G$. Επίσης, ενδέχεται κάποιο μη τερματικό σύμβολο να μην εμφανίζεται σε καμμία από όλες τις δυνατές ακολουθίες παραγωγών που μπορεί να προκύψουν από την γραμματική $G$. Είναι προφανές ότι σύμβολα με τις παραπάνω ιδιότητες είναι ουσιαστικά *άχρηστα* και μπορούν (μαζί με τους κανόνες παραγωγής στους οποίους εμπλέκονται) να παραλειφθούν, προκειμένου να *απλοποιηθεί* η γραμματική.

Στους επόμενους ορισμούς θα προσπαθήσουμε να χαρακτηρίσουμε τυπικά τα χρήσιμα σύμβολα μίας γραμματικής.

**9.3.2 Ορισμός.** Ένα μη τερματικό σύμβολο $A$ μίας γραμματικής δένδρων $G$ ονομάζεται *παραγωγικό* (productive) αν υπάρχει ακολουθία παραγωγών : $A \stackrel{+}{\Rightarrow}_G t$, με $t \in \mathcal{T}(\mathcal{F})$.

**9.3.3 Ορισμός.** Ένα μη τερματικό σύμβολο $A$ μίας γραμματικής δένδρων $G$ ονομάζεται *προσβάσιμο* (reachable) από το αξίωμα αν υπάρχει ακολουθία παραγωγών : $S \stackrel{+}{\Rightarrow}_G C[A]$, όπου $S$ : το αξίωμα και $C$ ένα περιβάλλον.

Στον παρακάτω ορισμό χαρακτηρίζουμε τις απλοποιημένες γραμματικές :

**9.3.4 Ορισμός.** Μία γραμματική δένδρων ονομάζεται *μειωμένη* (reduced) αν όλα τα μη τερματικά της σύμβολα είναι παραγωγικά και προσβάσιμα.

Είναι δυνατόν να υπολογίσουμε κάθε ένα από τα σύνολα των παραγωγικών και των προσβάσιμων μη τερματικών συμβόλων μίας γραμματικής δένδρων. Και τα δύο σύνολα κατασκευάζονται επαναληπτικά. Στο σύνολο των παραγωγικών συμβόλων, το οποίο αρχικά είναι κενό, προσθέτουμε ένα μη τερματικό $A$ αν υπάρχει κανόνας $A \to \alpha$ όπου όλα τα μη τερματικά που εμφανίζονται στο δεξιό μέλος (δηλαδή στο $\alpha$) είναι ήδη στο σύνολο των παραγωγικών συμβόλων.[1] Στο σύνολο των προσβάσιμων μη τερματικών συμβόλων, αρχικά τοποθετούμε πρώτα το αξίωμα $S$ και μετά σε κάθε επανάληψη κάθε μη τερματικό που βρίσκεται στον δεξιό μέλος κανόνα του οποίου το αριστερό μέλος (δηλαδή το τερματικό σύμβολο) έχει ήδη χαρακτηριστεί προσβάσιμο. Και στις δύο περιπτώσεις, η διαδικασία τερματίζεται αν σε έναν γύρο σάρωσης όλων των κανόνων δεν προστεθεί νέο μη τερματικό στο εκάστοτε σύνολο.

Ο αλγόριθμος λοιπόν που μετατρέπει μία οποιαδήποτε γραμματική σε μειωμένη αρκεί πρώτα να υπολογίσει το σύνολο των παραγωγικών συμβόλων και να αφαιρέσει από την γραμματική τα υπόλοιπα σύμβολα καθώς και τους κανόνες στους οποίους τα τελευταία εμπλέκονται και κατόπιν, στην γραμματική που προκύπτει, να υπολογίσει τα προσβάσιμα σύνολα και μετά να αφαιρέσει τα υπόλοιπα (μαζί με τους αντίστοιχους κανόνες) από την γραμματική. Η γραμματική που προκύπτει είναι ισοδύναμη με την αρχική και μειωμένη. Ας σημειωθεί ότι έχει σημασία η σειρά αφαίρεσης πρώτα των μη παραγωγικών και μετά των μη προσβάσιμων συμβόλων.

Αποδεικνύεται λοιπόν η παρακάτω πρόταση :

**9.3.5 Πρόταση.** *Μία κανονική γραμματική δένδρων είναι ισοδύναμη με μία μειωμένη γραμματική δένδρων.*

Παρ' όλα αυτά, μπορούμε να περιορίσουμε ακόμα περισσότερο την μορφή μίας γραμματικής και να διατηρήσουμε όλες τις δυνατότητες παραγωγής των κανονικών γραμματικών.

1. Προφανώς, αν το $\alpha$ δεν περιέχει κανένα μη τερματικό σύμβολο, δηλαδή είναι όλα τερματικά, το $A$ τοποθετείται στο σύνολο των παραγωγικών μη τερματικών συμβόλων.

**9.3.6 Ορισμός.** Μία γραμματική δένδρων $G$ ονομάζεται *κανονικοποιημένη* (normalized) αν όλοι οι κανόνες της γραμματικής είναι της μορφής $A \to a$ ή $A \to f(A_1, \ldots, A_n)$, όπου $a$ : σταθερά, $f \in \mathcal{F}_n$ και τα $A_1$, ..., $A_n$ : μη τερματικά.

Ισχύει λοιπόν η παρακάτω πρόταση :

**9.3.7 Πρόταση.** *Μία κανονική γραμματική δένδρων είναι ισοδύναμη με μία κανονικοποιημένη γραμματική δένδρων.*

*Ιδέα της απόδειξης.* Για κάθε κανόνα της κανονικής γραμματικής της μορφής $A \to f(t_1, \ldots, t_n)$ που δεν είναι στην απαιτούμενη μορφή, εισάγουμε τόσα νέα μη τερματικά σύμβολα όσοι οι όροι από τους $t_1$, ..., $t_n$ που δεν είναι μη τερματικά. Κατόπιν αντικαθιστούμε στον αρχικό κανόνα τα νέα μη τερματικά και προσθέτουμε όλους τους κανόνες με αριστερό μέλος τα νέα τερματικά σύμβολα και δεξιό μέλος τους αντίστοιχούς τους όρους. Η διαδικασία επαναλαμβάνεται μέχρι όλοι οι κανόνες να φτάσουν στην επιθυμητή μορφή (αν και πιθανόν προστίθενται νέοι κανόνες, η διαδικασία τερματίζει επειδή το ύψος των όρων σε προβληματικούς κανόνες συνεχώς μειώνεται).

Ενδέχεται να υπάρχουν στην γραμματική και κανόνες της μορφής $A \to B$, όπου $A$, $B$ μη τερματικά σύμβολα. Αυτό σημαίνει ότι τα μη τερματικά $A$, $B$ είναι «ισοδύναμα» και μπορεί να μείνει μόνον το ένα από τα δύο στην γραμματική και ταυτόχρονα να καταργηθεί ο κανόνας $A \to B$, αφού πρώτα στους υπόλοιπους κανόνες αντικατασταθεί το μη τερματικό που καταργείται από αυτό που μένει. Σε κάθε κλάση όμως ισοδυναμίας ενδέχεται να περιέχονται περισσότερα του ενός μη τερματικά (για παράδειγμα από τους κανόνες $A \to B$ και $B \to C$ και τα τρία εμπλεκόμενα μη τερματικά είναι «ισοδύναμα») και ο εντοπισμός τους πρέπει να γίνει με κάποιον αλγόριθμο μεταβατικού κλεισίματος (υπάρχει σχέση με το πρόβλημα REACHABILITY· βλέπε [82]). □

## 9.4 Κανονικές γραμματικές και πεπερασμένα αυτόματα

Μπορεί κανείς να δει ότι οι κανονικές γραμματικές δένδρων συνδέονται άμεσα με τα αυτόματα δένδρων που λειτουργούν από τα πάνω προς τα κάτω (top down). Μάλιστα, οι κανόνες μίας κανονικοποιημένης γραμματικής δένδρων είναι σχεδόν παρόμοιοι με τους κανόνες μετάβασης ενός TDNFTA. Οι λεπτομέρειες της απόδειξης ότι τα πεπερασμένα αυτόματα δένδρων και οι κανονικές γραμματικές δένδρων αποτελούν μία εναλλακτική περιγραφή των αναγνωρίσιμων (ή κανονικών) γλωσσών αφήνονται στον αναγνώστη, οπότε προκύπτει άμεσα το παρακάτω βασικό θεώρημα :

**9.4.1 Θεώρημα.** *Μία γλώσσα είναι αναγνωρίσιμη (recognizable), δηλαδή αναγνωρίζεται από πεπερασμένο αυτόματο δένδρων, αν και μόνον αν παράγεται από κανονική γραμματική δένδρων.*

*Ιδέα της απόδειξης.* Επαγωγή στο μήκος των ακολουθιών παραγωγών. □

## 9.5 Κανονικές παραστάσεις και εξισώσεις

Σε αυτήν την ενότητα θα δώσουμε δύο ακόμα εναλλακτικούς τρόπους περιγραφής της κλάσης των κανονικών γλωσσών, τις κανονικές παραστάσεις και τις κανονικές εξισώσεις.

### Κανονικές παραστάσεις

Μία βασική πράξη στα πλαίσια των συμβολοσειρών, που είναι απαραίτητη στον ορισμό των κανονικών παραστάσεων, είναι αυτή της παράθεσης (concatenation). Για να ορίσουμε κανονικές παραστάσεις στα πλαίσια των δένδρων, χρειαζόμαστε μία αντίστοιχη πράξη για τα δένδρα. Η παράθεση συμβολοσειρών είναι απλή : τοποθετούμε την δεύτερη συμβολοσειρά μετά το πέρας της πρώτης. Η αντίστοιχη πράξη στα δένδρα, όμως, δεν μπορεί να είναι τόσο απλή : δεν είναι δυνατόν να πάρουμε δύο δένδρα και να τα ενώσουμε με απλό τρόπο. Θα πρέπει με κάποιον τρόπο να αφήσουμε μία οπή στο πρώτο δένδρο και εκεί να τοποθετήσουμε το δεύτερο δένδρο.

Για να επιτύχουμε το παραπάνω, προσθέτουμε στο εκάστοτε αλφάβητο συναρτησιακών συμβόλων με το οποίο κατασκευάζονται οι όροι ένα νέο σύνολο σταθερών $\mathcal{K}$ οι οποίες θα παίζουν το ρόλο των οπών και συνήθως συμβολίζονται $\Box_1$, $\Box_2$, ... . Με την βοήθεια αυτών ορίζουμε την έννοια της *δενδρικής αντικατάστασης* :

**9.5.1 Ορισμός** (Δενδρική αντικατάσταση)**.** Η *δενδρική αντικατάσταση* των $\Box_1$, ..., $\Box_n$ από τις γλώσσες $L_1$, ..., $L_n$ στον όρο $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{K})$ συμβολίζεται :

$$t\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}$$

και ορίζεται επαγωγικά ως εξής :

- $\Box_i\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\} = L_i$ για $i = 1, \ldots, n$
- $a\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\} = \{a\}$, για $a \in \mathcal{F}_0$
- $f(s_1, \ldots, s_n)\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\} =$
  $\{f(t_1, \ldots, t_n) \mid t_i \in s_i\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}\}$

Επίσης, θεωρούμε την επέκταση της δενδρικής αντικατάστασης από όρους σε γλώσσες :

$$L\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\} \stackrel{\text{ορ.}}{=} \bigcup_{t \in L} \{t\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}\}$$

Πρακτικά, προκύπτουν νέοι όροι ως εξής : αντικαθιστούμε σε κάθε οπή από το σύνολο $\mathcal{K}$, έναν όρο από την αντίστοιχη στην οπή γλώσσα. Μία σημαντική διαφορά σε σχέση με την αντικατάσταση όρων (βλέπε ορισμό 4.4.3, σελίδα 41) είναι ότι στην δενδρική αντικατάσταση αν μία οπή (έστω η $\Box_1$) εμφανίζεται παραπάνω από μία φορές στον όρο τότε μπορούμε να τοποθετήσουμε διαφορετικό όρο από την ίδια γλώσσα (έστω την $L_1$) στις διαφορετικές εμφανίσεις της οπής (ενώ αντίθετα στην αντικατάσταση όρων τοποθετείται συγκεκριμένος όρος σε περισσότερες από μία εμφανίσεις της ίδιας μεταβλητής).

Τώρα πλέον με την βοήθεια της έννοιας της δενδρικής αντικατάστασης μπορούμε να ορίσουμε την *παράθεση* για γλώσσες δένδρων :

**9.5.2 Ορισμός.** Έστω γλώσσες $L$, $M$ του $\mathcal{T}(\mathcal{F}, \mathcal{K})$ και $\Box \in \mathcal{K}$. Τότε η *παράθεση* της $M$ στην $L$ μέσω της οπής $\Box$ ορίζεται ως εξής :

$$L \,._{\Box}\, M \stackrel{\text{ορ.}}{=} L\{\Box \leftarrow M\} = \bigcup_{t \in L} \{t\{\Box \leftarrow M\}\}$$

Τέλος, ορίζουμε μία έννοια κλεισίματος, αντίστοιχη του κλεισίματος κατά Kleene (Kleene closure) στις συμβολοσειρές, αφού πρώτα ορίσουμε έναν τρόπο να επαναλαμβάνουμε την παράθεση :

**9.5.3 Ορισμός.** Έστω γλώσσα $L$ του $\mathcal{T}(\mathcal{F}, \mathcal{K})$ και $\Box \in \mathcal{K}$. Τότε η *επαναληπτική παράθεση* της $L$ στον εαυτό της μέσω της οπής $\Box$ ορίζεται αναδρομικά, για κάθε $n \in \mathbb{N}$, ως εξής :

$$L^{n,\Box} \quad \stackrel{\text{ορ.}}{=} \quad \begin{cases} \{\Box\}, & n = 0 \\ L^{n-1,\Box} \quad \cup \quad L^{n-1,\Box} \,._{\Box}\, L, & n > 0 \end{cases}$$

ενώ το *κλείσιμο* της γλώσσας $L$ μέσω της οπής $\Box$ ορίζεται ως εξής :

$$L^{*,\Box} = \bigcup_{n \in \mathbb{N}} L^{n,\Box}$$

Ας σημειωθεί ότι $\Box \in L^{*,\Box}$ για κάθε γλώσσα $L$.

Μπορεί πλέον κανείς να αποδείξει, με την βοήθεια αυτομάτων δένδρων ή κανονικών γραμματικών, ιδιότητες *κλειστότητας* σε ό,τι αφορά τις κανονικές γλώσσες δένδρων για τις πράξεις της παράθεσης και του κλεισίματος. Παραθέτουμε τις σχετικές προτάσεις χωρίς απόδειξη, ενώ παραπέμπουμε τον ενδιαφερόμενο αναγνώστη στο [20] για τις αποδείξεις :

**9.5.4 Πρόταση.** *Έστω $L$, $L_1$, ..., $L_n$ κανονικές γλώσσες του $\mathcal{T}(\mathcal{F}, \mathcal{K})$, τότε η $L\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}$ είναι κανονική γλώσσα.*

**9.5.5 Πρόταση.** *Έστω $L$ κανονική γλώσσα του $\mathcal{T}(\mathcal{F}, \mathcal{K})$, τότε η $L^{*,\Box}$ είναι κανονική γλώσσα.*

Πλέον μπορούμε να ορίσουμε επαγωγικά τις κανονικές παραστάσεις και τις γλώσσες που παριστάνουν :

**9.5.6 Ορισμός.** Το σύνολο των *κανονικών παραστάσεων* επί του αλφαβήτου $\mathcal{F}$ και του συνόλου $\mathcal{K}$ συμβολίζεται με $\mathrm{RegExp}(\mathcal{F}, \mathcal{K})$ και είναι το μικρότερο σύνολο που περιέχει :

- το κενό σύνολο $\emptyset$ που παριστάνει την κενή γλώσσα,
- για κάθε $a \in \mathcal{F}_0 \cup K$, την $a$ που παριστάνει την γλώσσα $\{a\}$,
- για κάθε $f \in \mathcal{F}_n$ και για κανονικές παραστάσεις $r_1$, ..., $r_n$ που παριστάνουν τις γλώσσες $L_1$, ..., $L_n$ αντιστοίχως, την $f(r_1, \ldots, r_n)$ που παριστάνει την γλώσσα $\{f(s_1, \ldots, s_n) \mid s_1 \in L_1, \ldots, s_n \in L_n\}$,
- για κανονικές παραστάσεις $r_1$, $r_2$ που παριστάνουν τις γλώσσες $L_1$, $L_2$ αντιστοίχως, την $r_1 + r_2$ που παριστάνει την γλώσσα $L_1 \cup L_2$,
- για κανονικές παραστάσεις $r_1$, $r_2$ που παριστάνουν τις γλώσσες $L_1$, $L_2$ αντιστοίχως και για $\Box \in \mathcal{K}$, την $r_1 \cdot_\Box r_2$ που παριστάνει την γλώσσα $L_1 \cdot_\Box L_2$,
- για κανονική παράσταση $r$ που παριστάνει την γλώσσα $L$ και για $\Box \in \mathcal{K}$, την $r^{*,\Box}$ που παριστάνει την γλώσσα $L^{*,\Box}$.

**9.5.7 Παράδειγμα.** Η γλώσσα του παραδείγματος 9.2.2 μπορεί να παρασταθεί με την βοήθεια της παρακάτω κανονικής παράστασης :

$$\mathrm{List} = \mathrm{cons}(\ s(\Box_1)^{*,\Box_1} \cdot_{\Box_1} 0\ , \Box_2)^{*,\Box_2} \cdot_{\Box_2} \mathrm{nil}$$

Οι φυσικοί αριθμοί παριστάνονται ως $\mathrm{Nat} = s(\Box_1)^{*,\Box_1} \cdot_{\Box_1} 0$.

Όπως αναφέραμε νωρίτερα, οι κανονικές παραστάσεις αποτελούν άλλη μία εναλλακτική περιγραφή των κανονικών γλωσσών δένδρων :

**9.5.8 Θεώρημα.** *Μία γλώσσα είναι δυνατόν να παρασταθεί από κανονική παράσταση αν και μόνον είναι αναγνωρίσιμη.*

*Ιδέα της απόδειξης.* Η κατεύθυνση «$\Rightarrow$» είναι εύκολο να αποδειχθεί με επαγωγή στην δομή της κανονικής παράστασης (ορισμός 9.5.6), με την βοήθεια και των προτάσεων 9.5.4 και 9.5.5.

Για την κατεύθυνση «$\Leftarrow$» θεωρούμε αυτόματο που αναγνωρίζει την γλώσσα, έστω με πλήθος καταστάσεων $n = |Q|$, και υποθέτουμε ότι οι καταστάσεις

είναι $q_1$, ..., $q_n$. Για $1 \leq i \leq n$, $0 \leq j \leq n$ και $K \subset Q$, ορίζουμε την γλώσσα δένδρων $L_i^{j,K}$ ως εξής : Η γλώσσα $L_i^{j,K}$ περιέχει ακριβώς τους όρους για τους οποίους υπάρχει εκτέλεση του αυτομάτου που αποδίδει την κατάσταση $q_i$ στην ρίζα του εκάστοτε όρου και στους υπόλοιπους κόμβους καταστάσεις με αρίθμηση μικρότερη ή ίση του $j$, ενώ τα φύλλα έχουν κατά σύμβαση εξ αρχής επιγραφεί με κάποια κατάσταση του συνόλου $K$. Μπορούμε να δείξουμε, με επαγωγή στο $j$, ότι η $L_i^{j,K}$ έχει κανονική παράσταση. Για $j = 0$, η $L_i^{0,K}$ είναι πεπερασμένη, ενώ για $j > 0$ χρησιμοποιώντας μία ιδέα των Floyd [38] και Warshall [107] (ειδική περίπτωση της μεθόδου δυναμικού προγραμματισμού στα [58, 68]) :

$$L_i^{j,K} = L_i^{j-1,K} \cup L_i^{j-1,K\cup\{q_j\}} \cdot_{q_j} (L_j^{j-1,K\cup\{q_j\}})^{*,q_j} \cdot_{q_j} L_j^{j-1,K}$$

οπότε και αυτή έχει κανονική παράσταση. Προφανώς, η γλώσσα που αποδέχεται το αυτόματο προκύπτει από ένωση των παραπάνω γλωσσών που αντιστοιχούν σε τελικές καταστάσεις και είναι η

$$L = \bigcup_{\{i|q_i\in Q_f\}} L_i^{n,\emptyset}$$

Για τις λεπτομέρειες της απόδειξης, βλέπε [20]. □

## Κανονικές εξισώσεις

Ένας άλλος τρόπος να περιγράψουμε την γραμματική του παραδείγματος 9.2.2 είναι με την βοήθεια των «εξισώσεων» :

$$\begin{aligned} \text{List} &= \text{nil} + \text{cons}(\text{Nat}, \text{List}) \\ \text{Nat} &= 0 + s(\text{Nat}) \end{aligned}$$

με μεταβλητές List και Nat.

Τυπικά :

**9.5.9 Ορισμός.** Έστω μεταβλητές $X_1$, $X_2$, ..., $X_n$ που αντιστοιχούν σε γλώσσες δένδρων και για $1 \leq i \leq n$, $1 \leq j \leq m_i$, οι $s_{ij}$ είναι όροι του $\mathcal{T}(\mathcal{F}, \{X_1, \ldots, X_n\})$, τότε ένα *σύστημα κανονικών εξισώσεων* $S$ είναι ένα σύστημα εξισώσεων της μορφής :

$$\begin{aligned} X_1 &= s_{11} + \cdots + s_{1m_1} \\ &\cdots \\ X_n &= s_{n1} + \cdots + s_{nm_n} \end{aligned}$$

Μία *λύση* του $S$ είναι μία $n$-άδα γλωσσών δένδρων $L_1, \ldots, L_n$ για την οποία ισχύει για κάθε $i$ με $1 \leq i \leq n$ :

$$L_i = \bigcup_{1 \leq j \leq m_i} (s_{ij}\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\})$$

Μας ενδιαφέρει η εύρεση της μικρότερης δυνατής λύσης και για αυτό ορίζουμε την μερική διάταξη :

$(L_1, \ldots, L_n) \leq (L'_1, \ldots, L'_n)$ αν και μόνον αν για κάθε $i = 1, \ldots, n : L_i \subseteq L'_i$

Προφανώς, το μικρότερο δυνατό στοιχείο ως προς αυτήν την μερική διάταξη είναι το $(\emptyset, \ldots, \emptyset)$.

Κατόπιν, για κάθε σύστημα με $n$ εξισώσεις, ορίζουμε τον τελεστή σταθερού σημείου $\tau\colon \mathcal{T}(\mathcal{F})^n \to \mathcal{T}(\mathcal{F})^n$ ως εξής :

$$\tau(L_1, \ldots, L_n) := (L'_1, \ldots, L'_n), \quad \text{όπου}$$
$$L'_1 = L_1 \cup \bigcup_{1 \leq j \leq m_1} s_{1j}\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\}$$
$$\ldots$$
$$L'_n = L_n \cup \bigcup_{1 \leq j \leq m_n} s_{nj}\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\}$$

Η μικρότερη δυνατή λύση προκύπτει ως το *ελάχιστο σταθερό σημείο* του τελεστή $\tau$, δηλαδή ως η μικρότερη δυνατή $n$-άδα γλωσσών $S$ σύμφωνα με την παραπάνω μερική διάταξη, για την οποία ισχύει $\tau(S) = S$. Από το θεώρημα Knaster-Tarski (βλέπε [98, 109]), επειδή ο τελεστής $\tau$ είναι *συνεχής*[2], η μικρότερη αυτή λύση υπάρχει πάντα και είναι ίση με

$$S = \bigcup \{\tau^i(\emptyset, \ldots, \emptyset) \mid i \in \mathbb{N}\}$$

Μπορούμε πλέον να αποδείξουμε το παρακάτω :

**9.5.10 Θεώρημα.** *Το ελάχιστο σταθερό σημείο ενός συστήματος κανονικών εξισώσεων είναι μία πλειάδα κανονικών γλωσσών δένδρων. Αντιστοίχως, για κάθε κανονική γλώσσα δένδρων υπάρχει σύστημα κανονικών εξισώσεων, τέτοιο ώστε η γλώσσα ισούται με κάποιο στοιχείο της πλειάδας που προκύπτει ως το ελάχιστο σταθερό σημείο του συστήματος.*

2. Ένας τελεστής $\tau$ είναι *συνεχής* αν είναι μονότονος, δηλαδή αν για δύο $n$-άδες γλωσσών ισχύει αν $S_1 \leq S_2$ τότε $\tau(S_1) \leq \tau(S_2)$, και αν για κάθε αύξουσα ακολουθία $n$-άδων γλωσσών $S_0 \leq S_1 \leq \ldots$ (ονομάζεται αλυσίδα) ισχύει : $\tau(\bigcup_{i \in \mathbb{N}} S_i) = \bigcup_{i \in \mathbb{N}} \tau(S_i)$.

*Ιδέα της απόδειξης.* Για κάθε μεταβλητή $X_i$ ενός συστήματος κανονικών εξισώσεων ορίζουμε μία κανονική γραμματική $G_i$ με αξίωμα αυτήν την μεταβλητή και κανόνες που προκύπτουν άμεσα από τις εξισώσεις του συστήματος. Κατόπιν δείχνουμε ότι ισχύει

$$\tau^i(\emptyset, \ldots, \emptyset) \subseteq (L(G_1), \ldots, L(G_n)), \text{ για κάθε } i \in \mathbb{N}$$

και ότι

$$(L(G_1), \ldots, L(G_n)) \subseteq \bigcup\{\tau^i(\emptyset, \ldots, \emptyset) \mid i \in \mathbb{N}\}$$

Αντιστρόφως, για κάθε κανονική γραμματική ορίζουμε ένα σύστημα κανονικών εξισώσεων με μεταβλητές τα μη τερματικά της γραμματικής και εξισώσεις που προκύπτουν άμεσα από τους κανόνες της γραμματικής. Στην ελάχιστη λύση του συστήματος και συγκεκριμένα στο στοιχείο που αντιστοιχεί στο αξίωμα της γραμματικής έχουμε την γλώσσα που παράγει η γραμματική. □

## 9.6 Σχέση γραμματικών λέξεων χωρίς συμφραζόμενα και κανονικών γραμματικών δένδρων

Ας θυμηθούμε τι είναι μία γραμματική λέξεων (ή συμβολοσειρών) χωρίς συμφραζόμενα :

**9.6.1 Ορισμός.** Μία *γραμματική λέξεων χωρίς συμφραζόμενα* είναι μία πλειάδα $G = (V, \Sigma, P, S)$, όπου

- $V$ : σύνολο μη τερματικών συμβόλων,
- $\Sigma$ : σύνολο τερματικών συμβόλων (απλό αλφάβητο), που δεν έχει κοινά στοιχεία με το σύνολο $V$,
- $P$ : σύνολο κανόνων παραγωγής, της μορφής $A \to \beta$, με $A \in V$ και $\beta \in (V \cup T)^*$,
- $S$ : το αξίωμα της γραμματικής, με $S \in V$.

Τώρα, μένει να ορίσουμε μία σχέση παραγωγής σύμφωνα με την οποία, αρχίζοντας από το αξίωμα της γραμματικής, θα μπορούμε με διαδοχικές αντικαταστάσεις να καταλήγουμε σε συμβολοσειρές επί του αλφαβήτου (μη τερματικών συμβόλων).

**9.6.2 Ορισμός.** Για την γραμματική (συμβολοσειρών) $G = (V, \Sigma, P, S)$ ορίζουμε την *σχέση παραγωγής* επί του $(V \cup \Sigma)^*$ : $\gamma_1 A \gamma_2 \Rightarrow_G \gamma_1 \beta \gamma_2$ αν και μόνον αν $\gamma_1$, $\gamma_2 \in (V \cup \Sigma)^*$ και υπάρχει ο κανόνας $A \to \beta$ στο $P$.

Η *γλώσσα* (συμβολοσειρών) που παράγει η γραμματική (συμβολοσειρών) $G$ είναι

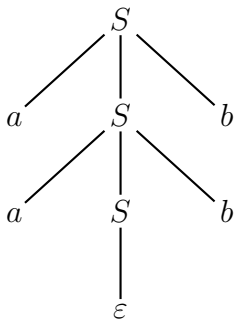
$$L(G) \stackrel{\text{ορ.}}{=} \{w \in \Sigma^* \mid S \stackrel{+}{\Rightarrow}_G w\}$$

**9.6.3 Παράδειγμα.** Για την γραμματική $G$ επί του αλφαβήτου $\Sigma = \{a, b\}$ και με κανόνες παραγωγής : $S \to aSb$ και $S \to \varepsilon$, μία πιθανή ακολουθία παραγωγών είναι η :

$$S \Rightarrow_G aSb \Rightarrow_G aaSbb \Rightarrow_G aabb$$

Η γλώσσα που παράγει η $G$ είναι : $L(G) = \{a^n b^n \mid n \in \mathbb{N}\}$.

*9.6.4 Παρατήρηση.* Κάθε ακολουθία παραγωγών μπορεί να περιγραφεί από ένα *συντακτικό δένδρο*. Στο σχήμα 9.1 φαίνεται το συντακτικό δένδρο που αντιστοιχεί στην ακολουθία παραγωγών του παραδείγματος 9.6.3. Για τον τυπικό ορισμό της έννοιας του συντακτικού δένδρου, παραπέμπουμε τον ενδιαφερόμενο αναγνώστη στο [51].




Σχήμα 9.1 – Συντακτικό δένδρο

Η παραπάνω παρατήρηση μας επιτρέπει να συνδέσουμε τις γραμματικές λέξεων χωρίς συμφραζόμενα με τις κανονικές γραμματικές δένδρων : Φαίνεται να υπάρχει ομοιότητα μεταξύ των συντακτικών δένδρων μίας γραμματικής λέξεων χωρίς συμφραζόμενα και των παραγωγών μίας κανονικής γραμματικής δένδρων. Για να προκύψει από το συντακτικό δένδρο η αντίστοιχη παραγόμενη συμβολοσειρά, αρκεί να θεωρήσουμε από τα αριστερά στα δεξιά το *φύλλωμα* του δένδρου. Για τον λόγο αυτό εισάγουμε τον παρακάτω τελεστή :

**9.6.5 Ορισμός.** Ο τελεστής Yield εφαρμόζεται σε όρους και υπολογίζεται ως εξής :

$$\text{Yield}(t) \stackrel{\text{ορ.}}{=} \begin{cases} t, \text{ αν } \text{root}(t) \in \mathcal{F}_0 \\ \text{Yield}(t_1) \ldots \text{Yield}(t_n), \text{ αν } t = f(t_1, \ldots, t_n) \text{ και } f \notin \mathcal{F}_0 \end{cases}$$

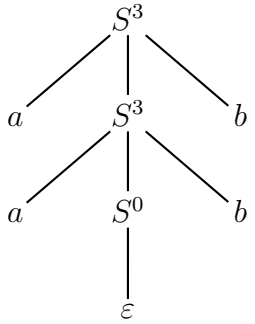
Ο τελεστής επεκτείνεται σε γλώσσες δένδρων ως εξής :

$$\text{Yield}(L) \stackrel{\text{ορ.}}{=} \{\text{Yield}(t) \mid t \in L\}$$

Μία ακόμα λεπτομέρεια που μένει να διευθετήσουμε είναι η εξής : Ένα μη τερματικό σύμβολο σε μία γραμματική λέξεων χωρίς συμφραζόμενα ενδέχεται να εμφανίζεται στο αριστερό μέρος κανόνων όπου το δεξιό μέλος έχει διαφορετικά μήκη. Για παράδειγμα, στην γραμματική του παραδείγματος 9.6.3, το δεξιό μέλος του κανόνα $S \to aSb$ έχει μήκος 3, ενώ αυτό του $S \to \varepsilon$ μηδενικό. Αυτό συνεπάγεται ότι πιθανόν στα διάφορα συντακτικά δένδρα, κόμβοι επιγεγραμμένοι με το εν λόγω τερματικό σύμβολο έχουν μεταβλητό πλήθος ορισμάτων. Κάτι τέτοιο είναι εμφανές στο σχήμα 9.1. Αντίθετα, οι γλώσσες δένδρων ορίζονται επί αλφαβήτων συναρτησιακών συμβόλων με σταθερό πλήθος ορισμάτων (βλέπε ορισμό 4.1.1 στην σελίδα 33).

Για τον λόγο αυτό, σε κάθε μη τερματικό σύμβολο της γραμματικής, σημειώνουμε (ως εκθέτη στο σύμβολο) και το πλήθος των παιδιών στο αντίστοιχο συντακτικό δένδρο. Προφανώς, για κανόνες της μορφής $A \to \varepsilon$, σημειώνουμε $A^0$.

Έτσι, με αυτήν την σύμβαση το δένδρο του σχήματος 9.1 παριστάνεται όπως φαίνεται στο σχήμα 9.2.




Σχήμα 9.2 – Συντακτικό δένδρο με πλήθος παιδιών

Το *σύνολο των συντακτικών δένδρων* μίας γραμματικής λέξεων χωρίς συμφραζόμενα είναι όλα τα δυνατά συντακτικά δένδρα στην παραπάνω μορφή (δηλαδή με σημειωμένο το πλήθος παιδιών) που μπορούν να προκύψουν από την γραμματική.

**9.6.6 Πρόταση.** *Έστω $G$ γραμματική λέξεων χωρίς συμφραζόμενα. Τότε το σύνολο των συντακτικών δένδρων της $G$ είναι κανονική γλώσσα δένδρων.*

*Απόδειξη.* Έστω $G = (V, \Sigma, P, S)$. Κατασκευάζουμε την κανονική γραμματική δένδρων $G = (S, V, \mathcal{F}, P')$ με

$$\mathcal{F} = \Sigma \cup \{\varepsilon\} \cup \{A^n \mid A \in N \text{ και } A \to \beta \in P, \text{ με } \mathrm{strlen}(\beta) = n\}$$

και για τους κανόνες παραγωγής :

$$\text{αν } A \to \varepsilon \in P, \text{ τότε } A \to A^0(\varepsilon) \in P'\cdot$$

$$\text{αν } A \to a_1 \dots a_p \in P, \text{ τότε } A \to A^p(a_1, \dots, a_p) \in P'.$$

Πλέον αρκεί να δείξουμε ότι $L(G) = \text{Yield}(L(G'))$ (με επαγωγή στο μήκος των ακολουθιών παραγωγής). □

**9.6.7 Πρόταση.** *Έστω $L$ κανονική γλώσσα δένδρων. Τότε η* $\text{Yield}(L)$ *είναι γραμματική λέξεων χωρίς συμφραζόμενα.*

*Απόδειξη.* Θεωρούμε ότι η $L$ παράγεται από κανονικοποιημένη γραμματική δένδρων $G = (S, N, \mathcal{F}, P)$. Κατασκευάζουμε γραμματική λέξεων χωρίς συμφραζόμενα $G' = (N, \mathcal{F}_0, P', S)$ έτσι ώστε ο κανόνας $X \to X_1 \dots X_n$ (αντίστοιχα ο $X \to a$) να είναι στο $P'$ αν και μόνον αν ο $X \to f(X_1, \dots, X_n)$ (αντίστοιχα ο $X \to a$) είναι στο $P$, για κάποιο $f \in \mathcal{F}_n$. Πλέον αρκεί να δείξουμε ότι $L(G') = \text{Yield}(L(G))$ (με επαγωγή στο μήκος των ακολουθιών παραγωγής). □

*9.6.8 Παρατήρηση.* Παρ' όλα αυτά υπάρχει κανονική γλώσσα δένδρων η οποία δεν είναι κάποιο σύνολο συντακτικών δένδρων μίας γραμματικής λέξεων χωρίς συμφραζόμενα.

Πράγματι αν θεωρήσουμε την κανονική γλώσσα δένδρων που αποτελείται μόνον από τον όρο $f(g(a), g(b))$ και το αντίστοιχο συντακτικό δένδρο με πλήθος παιδιών $f^2(g^1(a), g^1(b))$, τότε μία οποιαδήποτε γραμματική δένδρων θα έπρεπε να περιέχει κανόνες της μορφής :

$$F \to GG \qquad G \to a \qquad G \to b$$

με αξίωμα το $F$. Τότε όμως, θα ήταν δυνατό και το συντακτικό δένδρο $f^2(g^1(a), g^1(a))$· άτοπο.

## 9.7 Γραμματικές δένδρων χωρίς συμφραζόμενα

Όπως είπαμε και στην αρχή του κεφαλαίου υπάρχει μία ιεραρχία από κλάσεις γραμματικών δένδρων, ανάλογα με τους περιορισμούς που θέτουμε στην μορφή των κανόνων. Είδαμε ήδη στα προηγούμενα τις κανονικές γραμματικές δένδρων, που έχουν σημαντικούς περιορισμούς στην μορφή των κανόνων, αλλά τα περισσότερα προβλήματα που συνδέονται με αυτές είναι τουλάχιστον αποκρίσιμα. Βέβαια, δεδομένου ότι, όπως φάνηκε σε αυτό το κεφάλαιο, συνδέονται με τα πεπερασμένα αυτόματα δένδρων, οι κανονικές γραμματικές δένδρων χαρακτηρίζονται από περιορισμένες εκφραστικές δυνατότητες (δεν μπορούν για παράδειγμα να εκφράσουν μη γραμμικότητες). Από την άλλη, στην κορυφή της ιεραρχίας, χωρίς κανέναν περιορισμό στην μορφή των κανόνων έχουμε τις *γενικές* γραμματικές δένδρων (βλέπε ορισμό 9.1.1 στην σελίδα 99).

Για αυτές, αποδεικνύεται ότι έχουν ίδιες υπολογιστικές δυνατότητες με τις μηχανές Turing (μπορεί να τις δει κανείς ως παραλλαγή των συστημάτων μεταγραφής), με συνέπεια τα σημαντικότερα προβλήματα που τις αφορούν να είναι μη αποκρίσιμα.

Στο ενδιάμεσο, και ως προς τις εκφραστικές δυνατότητες, αλλά και ως προς το πλήθος των αποκρίσιμων προβλημάτων, βρίσκονται οι *γραμματικές δένδρων χωρίς συμφραζόμενα*.

**9.7.1 Ορισμός.** Μία *γραμματική δένδρων χωρίς συμφραζόμενα* είναι μία γραμματική δένδρων $G = (S, N, \mathcal{F}, P)$ (βλέπε ορισμό 9.1.1 στην σελίδα 99) στην οποία οι κανόνες είναι της μορφής $X(x_1, \ldots, x_n) \to \beta$, όπου $X$ : μη τερματικό σύμβολο με $n$ ορίσματα, $x_1, \ldots, x_n$ : μεταβλητές από ένα σύνολο $\mathcal{V}$ και $\beta$ : όρος του $\mathcal{T}(\mathcal{F} \cup N, \{x_1, \ldots, x_n\})$.

Η σχέση παραγωγής στην περίπτωση των γραμματικών δένδρων χωρίς συμφραζόμενα είναι κάπως πιο περίπλοκη, λόγω της παρουσίας μη τερματικών συμβόλων με μη μηδενικό πλήθος ορισμάτων, σε σύγκριση με την αντίστοιχη σχέση για κανονικές γραμματικές.

**9.7.2 Ορισμός.** Η *σχέση παραγωγής* (derivation relation) της γραμματικής χωρίς συμφραζόμενα $G$ ορίζεται επί του $\mathcal{T}(\mathcal{F} \cup N)$ ως εξής : $\gamma \Rightarrow_G \delta$ αν και μόνον αν υπάρχει κανόνας παραγωγής στην $G$ : $X(x_1, \ldots, x_n) \to \beta$, περιβάλλον $C$ και αντικατάσταση $\sigma = \{x_1 := t_1, \ldots, x_n := t_n\}$, τέτοια ώστε $\gamma = C[X(t_1, \ldots, t_n)]$ και $\delta = C[\beta^\sigma]$.

Αμέσως, έχουμε και τον ορισμό για την γλώσσα που παράγεται από μία γραμματική χωρίς συμφραζόμενα :

**9.7.3 Ορισμός.** Η γλώσσα δένδρων που *παράγεται* από την γραμματική χωρίς συμφραζόμενα $G$ συμβολίζεται με $L(G)$ και είναι το σύνολο :

$$L(G) \stackrel{\text{ορ.}}{=} \{t \in \mathcal{T}(\mathcal{F}) \mid S \stackrel{+}{\Rightarrow}_G t\}$$

όπου $S$ το αξίωμα της γραμματικής $G$.

Μία τέτοια γλώσσα δένδρων ονομάζεται, φυσικά, *γλώσσα δένδρων χωρίς συμφραζόμενα* (context free tree language).

Οι γραμματικές χωρίς συμφραζόμενα έχουν μελετηθεί στα πλαίσια των σχημάτων αναδρομής για την προδιαγραφή προγραμμάτων. Ένα μη τερματικό F με μη μηδενικό πλήθος ορισμάτων μπορεί να ιδωθεί ως το όνομα μίας συνάρτησης και ο κανόνας $\mathrm{F}(x_1, \ldots, x_n) \to t$ ως ο ορισμός της.

**9.7.4 Παράδειγμα.** Έστω η γραμματική χωρίς συμφραζόμενα με αξίωμα Prog, σύνολο μη τερματικών $\{\mathrm{Prog}, \mathrm{Nat}, \mathrm{Fact}()\}$, σύνολο τερματικών $\{0, s(), p(), \mathit{not}(), \mathit{eq}(,), \mathit{mult}(,), \mathit{if}(,,)\}$ και κανόνες

$$\begin{aligned} \mathrm{Prog} &\to \mathrm{Fact}(\mathrm{Nat}) \\ \mathrm{Nat} &\to 0 \mid s(\mathrm{Nat}) \\ \mathrm{Fact}(x) &\to \mathit{if}(\mathit{eq}(x, 0), s(0), \mathit{mult}(x, \mathrm{Fact}(p(x)))) \end{aligned}$$

Όπως μπορεί να διαπιστώσει εύκολα κανείς, το μη τερματικό Fact αντιστοιχεί στην συνάρτηση «παραγοντικό». (Το $p$ αντιστοιχεί στην συνάρτηση μείωσης κατά 1.)

Μία πιθανή ακολουθία παραγωγών είναι η εξής :

$$\begin{aligned} &\mathrm{Prog} \Rightarrow_G \mathrm{Fact}(\mathrm{Nat}) \Rightarrow_G \mathrm{Fact}(s(\mathrm{Nat})) \Rightarrow_G \mathrm{Fact}(s(s(\mathrm{Nat}))) \Rightarrow_G \\ &\quad \mathrm{Fact}(s(s(0))) \Rightarrow_G \mathit{if}(\mathit{eq}(s(s(0)), 0), s(0), \mathit{mult}(s(s(0)), \mathrm{Fact}(p(s(s(0)))))) \end{aligned}$$

# Κεφάλαιο 10

# Εφαρμογές και επεκτάσεις αυτομάτων δένδρων

Σε αυτό το σύντομο κεφάλαιο θα αναφερθούμε σε κάποιες εφαρμογές των αυτομάτων δένδρων. Μας ενδιαφέρουν ιδιαιτέρως εφαρμογές που έχουν σχέση με τα συστήματα μεταγραφής. Η βασική αναφορά είναι το [20] καθώς και το [47].

Επιπλέον, θα αναφερθούμε επιγραμματικά σε κάποιες επεκτάσεις των περασμένων αυτομάτων δένδρων, οι οποίες τους προσθέτουν υπολογιστικές δυνατότητες, ώστε να αναγνωρίζουν ευρύτερες κλάσεις γλωσσών δένδρων. (Στα πλαίσια των γραμματικών δένδρων είδαμε ήδη μία επέκταση σε σχέση με τις κανονικές γραμματικές δένδρων : τις γραμματικές δένδρων χωρίς συμφραζόμενα, στην ενότητα 9.7, σελίδα 112.)

Το παρόν κεφάλαιο είναι επίσης βασισμένο στο [20].

## 10.1 Εφαρμογές αυτομάτων δένδρων

Οι πρώτες εφαρμογές των αυτομάτων δένδρων συνδέονται με τα *κυκλώματα* (circuits) ως υπολογιστές αναδρομικών συναρτήσεων και την ασθενή μοναδική λογική δεύτερης τάξης με $k$ ακολούθους (weak second order monadic logic with $k$ successors, WSkS) [17, 102, 11, 87, 33, 34, 99, 100]. Για περισσότερες λεπτομέρειες, βλέπε [20].

Στα πλαίσια των συστημάτων μεταγραφής όρων, τα αυτόματα δένδρων έχουν πολλές εφαρμογές.

Η έννοια του «περικλείειν» (βλέπε ορισμό 4.5.5, σελίδα 42) είναι χρήσιμη, επειδή σε ένα σύστημα μεταγραφής όρων, αν ένας όρος $s$ μεταγράφεται, τότε αυτό σημαίνει ότι υπάρχει ένα αριστερό μέλος κανόνα, έστω $l$, τέτοιο ώστε ο όρος $s$ περικλείει τον $l$ (δηλαδή $s \trianglelefteq l$). Ένα βασικό αποτέλεσμα είναι το εξής :

**10.1.1 Πρόταση.** *Έστω $l$ γραμμικός όρος. Το σύνολο των όρων που περικλείουν τον $l$ είναι αναγνωρίσιμο από μη αιτιοκρατικό αυτόματο δένδρων μεγέθους $O(|l|)$.*

*Απόδειξη.* Βλέπε [20] για την κατασκευή. □

Δεδομένων των ιδιοτήτων κλεισίματος των αναγνωρίσιμων γλωσσών (βλέπε θεώρημα 8.6.1, σελίδα 87) και της προηγούμενης πρότασης, προκύπτει άμεσα το παρακάτω πόρισμα :

**10.1.2 Πόρισμα.** *Έστω αριστερογραμμικό σύστημα μεταγραφής όρων $\mathcal{R}$. Το σύνολο των όρων που ανάγονται είναι αναγνωρίσιμο. Το σύνολο των όρων που δεν ανάγονται (κανονικές μορφές) είναι αναγνωρίσιμο.*

Το τελευταίο αποτέλεσμα υπάρχει στο [42]. Φυσικά, υπάρχουν και συστήματα μεταγραφής όπου το σύνολο των όρων που ανάγονται δεν είναι αποκρίσιμο. Παρ' όλα αυτά, στο [63] υπάρχει απόδειξη ότι το πρόβλημα «δεδομένου συστήματος μεταγραφής όρων $\mathcal{R}$, είναι το σύνολο των κανονικών μορφών αναγνωρίσιμο ;» είναι αποκρίσιμο.

Έστω το παρακάτω πρόβλημα : «Δίνονται δύο όροι $s$, $t$ σε ένα σύστημα μεταγραφής. Ισχύει $s \xrightarrow{*} t$ ;». Είναι γνωστό ότι το προηγούμενο πρόβλημα είναι εν γένει μη αποκρίσιμο (για παράδειγμα στην συνδυαστική λογική· βλέπε παράδειγμα 5.2.1, σελίδα 49). Παρ' όλα αυτά, αν η σχέση μεταγραφής πολλών βημάτων $\xrightarrow{*}$ διατηρεί την αναγνωρισιμότητα (δηλαδή αν για κάθε αναγνωρίσιμη γλώσσα $L$, το σύνολο $S^*(L) \stackrel{\text{ορ.}}{=} \{t | s \xrightarrow{*} t, s \in L\}$ είναι επίσης αναγνωρίσιμο), τότε το παραπάνω πρόβλημα είναι αποκρίσιμο. Στα [9, 90, 21, 26, 19, 56] δίνονται κάποιες κατηγορίες συστημάτων για τις οποίες αποδεικνύεται ότι έχουν την παραπάνω ιδιότητα της διατήρησης της αναγνωρισιμότητας.

Εφαρμογές των αυτομάτων δένδρων συναντούμε επίσης στον λογικό προγραμματισμό [40].

## 10.2 Επεκτάσεις αυτομάτων δένδρων

### Αυτόματα εναλλαγής (alternation)

Υπάρχει μία έλλειψη συμμετρίας στα μη αιτιοκρατικά αυτόματα δένδρων : Αν δούμε τους κανόνες μετάβασης ενός, έστω bottom-up, αυτομάτου, μπορεί να υπάρχουν πολλοί κανόνες με το ίδιο αριστερό μέλος, έστω $l$, και με διάφορες καταστάσεις στο δεξιό μέλος : $q_1$, ..., $q_n$. Οι καταστάσεις συνδέονται, τρόπον τινά, με λογική διάζευξη και αυτό το συμβολίζουμε ως εξής :

$$l \to q_1 \vee \cdots \vee q_n,$$

δηλαδή το δεξιό μέλος αποτελεί πεπερασμένη διάζευξη καταστάσεων. Αυτή η ασυμμετρία είναι εμφανής αν θελήσουμε να υπολογίσουμε το συμπλήρωμα της γλώσσας που αποδέχεται ένα μη αιτιοκρατικό αυτόματο : Η αιτιοκρατικοποίησή του είναι αναπόφευκτη. Αντίθετα, στα *αυτόματα εναλλαγής δένδρων* (alternating tree automata· βλέπε [93, 73]) επιτρέπουμε το δεξιό μέλος να είναι συνδυασμός καταστάσεων σε DNF (disjunctive normal form : διαζευκτική κανονική μορφή) ή CNF (conjuctive normal form : συζευκτική κανονική μορφή), δηλαδή επιτρέπουμε επιπλέον την λογική σύζευξη μεταξύ καταστάσεων, για παράδειγμα :

$$l \to (q_1 \wedge q_2) \vee (q_3 \wedge q_4)$$

Έτσι, η μετατροπή ενός αυτομάτου εναλλαγής σε αυτό που αναγνωρίζει το συμπλήρωμα γίνεται απλώς με αλλαγή των «$\vee$» σε «$\wedge$» και αντιστρόφως και εναλλαγή τελικών με αρχικές καταστάσεις. Τα αυτόματα εναλλαγής δένδρων έχουν εφαρμογές στην χρονική λογική (βλέπε, για παράδειγμα, [72, 64]). Για την έννοια της *εναλλαγής* (alternation) στους υπολογισμούς, παραπέμπουμε τον ενδιαφερόμενο αναγνώστη στο [14].

## Αυτόματα με περιορισμούς

Τα πεπερασμένα αυτόματα όπως είδαμε (με την βοήθεια του λήμματος μετάντλησης ή pumping lemma) δεν μπορούν να αναγνωρίσουν κάποιες γλώσσες. Μία περίπτωση είναι γλώσσες της μορφής

$$\{f(t,t) \mid t \in \mathcal{T}(\mathcal{F})\}.$$

Για να αναγνωρίζονται τέτοιες γλώσσες, μία λύση είναι να προσθέσουμε στο αυτόματο την δυνατότητα να ελέγχει τους υποόρους σε διαφορετικές θέσεις, να αποφαίνεται αν είναι ίσοι, και ανάλογα με το αποτέλεσμα της σύγκρισης να μεταβαίνει σε κάποια κατάσταση.

Τυπικά, σε ένα *αυτόματο με περιορισμούς* κάθε κανόνας είναι προσαυξημένος με κάποιον περιορισμό που αφορά ισότητα όρων σε συγκεκριμένες θέσεις του αριστερού μέλους του κανόνα.

**10.2.1 Παράδειγμα.** Ένα αυτόματο με περιορισμούς που αναγνωρίζει την γλώσσα

$$L = \{f(t,t) \mid t \in \mathcal{T}(\mathcal{F})\}, \text{ με } \mathcal{F} = \{f(,), a\}$$

έχει κανόνες :

$$a \to q, \quad f(q,q) \to q, \quad f(q,q) \stackrel{1=2}{\to} q_{\mathrm{f}},$$

όπου $q_{\mathrm{f}}$ η μόνη τελική κατάσταση.

Αν θεωρήσουμε μόνον ισότητες, η κλάση των αυτομάτων με περιορισμούς δεν είναι κλειστή ως προς συμπλήρωμα. Για αυτό, συνήθως θεωρούμε ότι μπορούμε στους περιορισμούς να έχουμε και *ανισότητες*, για παράδειγμα : $1 \neq 2$. Επίσης, μπορούμε να θεωρήσουμε σε έναν περιορισμό οποιονδήποτε λογικό συνδυασμό πεπερασμένου πλήθους ισοτήτων και ανισοτήτων. Με αυτούς τους ορισμούς, η κλάση είναι κλειστή ως προς τις λογικές πράξεις. Για περισσότερες λεπτομέρειες, βλέπε [70].

Η παραπάνω κλάση αυτομάτων με περιορισμούς είναι αρκετά γενική και δεν είναι αποκρίσιμα όλα τα προβλήματα που την αφορούν. Για παράδειγμα, το πρόβλημα αν η γλώσσα που αποδέχεται ένα αυτόματο με περιορισμούς είναι κενή είναι μη αποκρίσιμο. Η απόδειξη γίνεται με αναγωγή του προβλήματος αντιστοίχισης του Post (Post Correspondence Problem ή σύντομα PCP, βλέπε [109])· βλέπε [20].

Για τον λόγο αυτό θεωρούμε μία πιο περιορισμένη κλάση αυτομάτων με περιορισμούς, τα *αυτόματα με περιορισμούς μεταξύ αδελφών*. Όπως δηλώνει και το όνομά τους, σε αυτά επιτρέπονται μόνον περιορισμοί της μορφής $i = j$ ή $i \neq j$, όπου $i$, $j$ θετικοί ακέραιοι, δηλαδή ο έλεγχος για (αν)ισότητα γίνεται μόνον σε άμεσους υποόρους (αδελφούς). Η νέα κλάση είναι αρκετά ισχυρή (το αυτόματο του παραδείγματος 10.2.1 είναι αυτής της μορφής), έχει τις ιδιότητες κλειστότητας και επιπλέον το πρόβλημα της κενής γλώσσας είναι αποκρίσιμο (βλέπε [7]).

## Μετατροπείς (transducers)

Μία επίσης σημαντική επέκταση των αυτομάτων δένδρων προκύπτει αν σε αυτά προσθέσουμε την δυνατότητα να παρέχουν έξοδο κάποιο άλλο δένδρο. Σε αυτήν την περίπτωση, έχουμε τους λεγόμενους *μετατροπείς δένδρων* (tree transducers). Σε σχέση με τους αντίστοιχους μετατροπείς στα πλαίσια των αυτομάτων συμβολοσειρών (πεπερασμένα αυτόματα με έξοδο, για παράδειγμα μηχανές Moore ή Mealy, μετατροπείς συμβολοσειρών· βλέπε [51]), οι μετατροπείς δένδρων έχουν περισσότερες δυνατότητες και μάλιστα μπορούν να μοντελοποιήσουν μετατροπές που εξαρτώνται από την σύνταξη (syntax directed transformations), όπως αυτές που συναντώνται σε έναν μεταγλωττιστή. Για περισσότερες λεπτομέρειες, σχετικά με τους μετατροπείς δένδρων, παραπέμπουμε τον ενδιαφερόμενο αναγνώστη στο [41].

# Μέρος τρίτο

# Ο συνδυαστής $S$

# Κεφάλαιο 11

# $S$-όροι

Προτού μελετήσουμε τον $S$-λογισμό, είναι σκόπιμο να αναφερθούμε στα αντικείμενα μεταγραφής αυτού, που είναι οι $S$-όροι.

Ο $S$-λογισμός είναι ένα υποσύστημα της συνδυαστικής λογικής (βλέπε παράδειγμα 5.2.1, σελίδα 49), το οποίο περιέχει μόνον τον *συνδυαστή* $S$.

Ο Smullyan στο [94] δίνει στους διαφόρους συνδυαστές της συνδυαστικής λογικής ονόματα πτηνών. Αποκαλεί τον συνδυαστή $S$ *starling* (ψαρ, ψαρόνι).

## 11.1 Τυπικός ορισμός $S$-όρων

Μπορούμε να ορίσουμε τους $S$-όρους με την βοήθεια ενός αλφαβήτου $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_2$, με $\mathcal{F}_0 = \{S\}$ και $\mathcal{F}_2 = \{\circ\}$, όπου ο «$\circ$» νοείται συνήθως ως ενθεματικός τελεστής και αναφέρεται ως τελεστής εφαρμογής (βλέπε ενότητα 4.1, σελίδα 33, για τους ορισμούς του αλφαβήτου και των όρων). Υπενθυμίζουμε ότι θεωρούμε πάντοτε και ένα αριθμήσιμο σύνολο μεταβλητών. Αναλυτικότερα, έχουμε :

**11.1.1 Ορισμός.** Ένας $S$-όρος ορίζεται επαγωγικά ως εξής :
- Το $S$ και οι μεταβλητές είναι $S$-όροι.
- Αν τα $t_1$, $t_2$ είναι $S$-όροι, το $(t_1 \circ t_2)$ είναι επίσης $S$-όρος.

Τίποτε άλλο δεν είναι $S$-όρος.

Κατά τα γνωστά, ένας $S$-όρος χωρίς μεταβλητές ονομάζεται θεμελιώδης.

## 11.2 Αναπαράσταση $S$-όρων

Έστω ότι δίνεται ο $S$-όρος :

$$(((S \circ (S \circ S)) \circ S) \circ (S \circ ((S \circ S) \circ S)))$$

Δεδομένου ότι έχουμε μόνο έναν τελεστή, τον «∘», μπορούμε να τον παραλείψουμε εντελώς, οπότε ο παραπάνω $S$-όρος γράφεται :

$$(((S(SS))S)(S((SS)S)))$$

Επιπλέον, προκειμένου να μειώσουμε το πλήθος των παρενθέσεων, θεωρούμε ότι ο τελεστής «∘» προσεταιρίζει από τα αριστερά. Με αυτήν την παραδοχή, ο πιο πάνω όρος γράφεται :

$$S(SS)S(S(SSS))$$

Η παραπάνω παράσταση (κατάργηση τελεστή και προσεταιρισμός από τα αριστερά) είναι σαφώς επηρεασμένη από τις αντίστοιχες συμβάσεις στην συνδυαστική λογική (βλέπε παράδειγμα 5.2.1 στην σελίδα 49).

Επίσης, συνήθως χρησιμοποιούμε τις εξής δύο συντομογραφίες :

$$A \stackrel{\text{ορ.}}{=} SSS \quad \text{και} \quad B \stackrel{\text{ορ.}}{=} S(SS),$$

οπότε ο αρχικός $S$-όρος τελικά γράφεται :

$$BS(SA)$$

Επίσης, μπορούμε να καταργήσουμε εντελώς την χρήση παρενθέσεων αν χρησιμοποιήσουμε την λεγόμενη προθεματική (ή πολωνική) γραφή[1]. Τότε ο αρχικός όρος γράφεται :

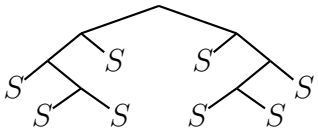
$$oooSoSSSoSooSSS,$$

όπου το «$o$» είναι στην ουσία ο τελεστής εφαρμογής, ο οποίος όμως τώρα χρησιμοποιείται ως *προθεματικός* δυαδικός τελεστής. Ας σημειωθεί πως αν καταργήσουμε εντελώς τις παρενθέσεις, θα πρέπει να διατηρήσουμε τον τελεστή της εφαρμογής στην αναπαράσταση. Αυτή η γραφή δεν είναι ίσως τόσο ευανάγνωστη από τον άνθρωπο, αλλά επιτρέπει ευκολότερη τεχνολόγηση (parsing) από τον υπολογιστή.

Για κάθε όρο $t$, με $\text{prefix}(t)$ συμβολίζουμε την συμβολοσειρά της αναπαράστασής του σε προθεματική γραφή, η οποία έχει ένα λιγότερο $o$ από το πλήθος των $S$, επομένως γενικά ισχύει : $\text{strlen}(\text{prefix}(t)) = 2\,\text{len}(t) - 1$ (για το len, βλέπε παρακάτω, ορισμό 11.2.1). Η $\text{prefix}(t)$ ονομάζεται και *προθεματική αναπαράσταση* (prefix notation) του $t$, εξ ου και ο συμβολισμός.

Τέλος, στην ενότητα 4.2, σελίδα 35, είδαμε ότι κάθε όρος αναπαρίσταται και ως δένδρο. Το ίδιο κάνουμε και με τους $S$-όρους. Για παράδειγμα, η δενδρική αναπαράσταση του αρχικού όρου φαίνεται στο σχήμα 11.1.

1. Αυτός ο τρόπος γραφής οφείλεται στον Jan Łukasiewicz (1878-1956), ο οποίος τον εισήγαγε προκειμένου να γράφει προτάσεις της μαθηματικής λογικής χωρίς παρενθέσεις.




Σχήμα 11.1 – Ο όρος $BS(SA)$ ως δένδρο

Παρακάτω, σε αυτό κεφάλαιο, θα ασχοληθούμε κυρίως με θεμελιώδεις $S$-όρους. Για διευκόλυνση, αλλά και για ιστορικούς λόγους, δίνουμε τον παρακάτω ορισμό :

**11.2.1 Ορισμός.** Το *μήκος* (length) ενός $S$-όρου $t$ συμβολίζεται με $\mathrm{len}(t)$ και ορίζεται ως το πλήθος των $S$ που ο όρος περιέχει.

*11.2.2 Παρατήρηση.* Το *μήκος* $\mathrm{len}(t)$ ενός $S$-όρου συνδέεται άμεσα με το μέγεθος $|t|$ αυτού, όπως αυτό δίνεται από τον ορισμό 4.3.2, στην σελίδα 40, ως εξής : $|t| = 2\,\mathrm{len}(t) - 1$, δεδομένου ότι υπάρχουν ακριβώς $\mathrm{len}(t) - 1$ πράξεις εφαρμογής στον εν λόγω όρο. Προφανώς, δύο όροι έχουν το ίδιο μήκος αν και μόνον αν έχουν το ίδιο μέγεθος.

## 11.3 Απαρίθμηση $S$-όρων μήκους $n$

Έστω $C_n$ το πλήθος των $S$-όρων μήκους $n$. Προφανώς, υπάρχει ακριβώς ένας όρος μήκους ίσου με 1, ο $S$. Αν $n > 1$, τότε για κάθε $i$ τέτοιο ώστε $1 \leq i < n$ συνδυάζουμε όλους του όρους με μήκος $i$ με όλους τους όρους με μήκος $n-i$. Επομένως, έχουμε τον ακόλουθο αναδρομικό τύπο για το πλήθος των $S$-όρων μήκους $n$ :

$$C_1 = 1, \quad C_n = \sum_{i=1}^{n-1} C_i\, C_{n-i}, \ \text{για } n > 1 \tag{11.1}$$

Θεωρούμε την ακολουθία $\{C_n\}_{n=1}^{\infty}$. Οι 25 πρώτοι όροι της παραπάνω ακολουθίας είναι : 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324.

## 11.4 Αριθμοί Catalan· κλειστός τύπος

Συχνά κάνουμε μία αλλαγή στην μεταβλητή δείκτη, έτσι ώστε

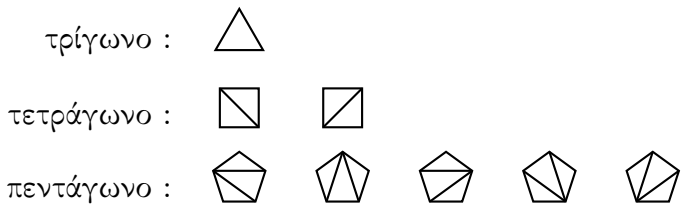
$$a_n = C_{n+1}, \quad \text{για } n \geq 0 \tag{11.2}$$

Οι ακολουθίες $\{a_n\}_{n=0}^{\infty}$ και $\{C_n\}_{n=1}^{\infty}$, αν εξαιρέσουμε την μετατόπιση, είναι ακριβώς ίδιες :

| | | | | | | |
|---|---|---|---|---|---|---|
| $n$: | 0 | 1 | 2 | 3 | 5 | ... |
| $C_n$: | | 1 | 1 | 2 | 5 | ... |
| $a_n$: | 1 | 1 | 2 | 5 | 14 | ... |

Οι αριθμοί $a_n$ (καμμία φορά και οι $C_n$) ονομάζονται *αριθμοί Catalan* προς τιμήν του Eugène Charles Catalan (1814–1894). Ο Catalan ανακάλυψε τους αριθμούς αυτούς κατά την προσπάθεια του να λύσει το πρόβλημα της διαγωνιοποίησης[2] κυρτού πολυγώνου. Το πρόβλημα είχε λυθεί ήδη από τον 18ο αιώνα από τον Segner, αλλά η λύση που έδωσε ο Catalan είναι σαφώς κομψότερη [13]. Ο Euler είχε επίσης ασχοληθεί επιτυχώς με την απλοποίηση της λύσης του προβλήματος, καθώς και ο Binet, περίπου την ίδια εποχή με τον Catalan, γύρω στο 1838.

Πράγματι, αν δίνεται ένα κυρτό πολύγωνο με $n+2$ πλευρές (προφανώς πρέπει $n \geq 1$), τότε μπορεί να αποδειχθεί ότι υπάρχουν $a_n$ τρόποι να το διαγωνιοποιήσουμε. Για τα τρία πρώτα κυρτά πολύγωνα, οι δυνατές διαγωνιοποιήσεις φαίνονται στο σχήμα 11.2.




Σχήμα 11.2 – Διαγωνιοποίηση

Οι αριθμοί Catalan εμφανίζονται σε πάρα πολλές περιπτώσεις απαρίθμησης συνδυαστικών αντικειμένων (βλέπε για παράδειγμα το [96]).

Αναζητούμε έναν κλειστό (μη αναδρομικό) τύπο για το $C_n$ ή το $a_n$.

Από τις (11.1) και (11.2), προκύπτει εύκολα ο παρακάτω αναδρομικός τύπος για την ακολουθία $a_n$ :

$$a_0 = 1, \quad a_n = \sum_{i=0}^{n-1} a_i \, a_{n-1-i}, \ \text{για } n > 0 \tag{11.3}$$

Τώρα, θεωρούμε την δυναμοσειρά με συντελεστές τα στοιχεία της ακο-

2. Η διαγωνιοποίηση ενός κυρτού πολυγώνου είναι ο χωρισμός του σε τρίγωνα, μέσω μη τεμνομένων διαγωνίων του. Για να διαγωνιοποιηθεί ένα πολύγωνο με $s$ πλευρές απαιτούνται ακριβώς $s-3$ διαγώνιοι, οπότε προκύπτουν $s-2$ τρίγωνα.

λουθίας $a_n$ :

$$g(x) = \sum_{i=0}^{\infty} a_i x^i \tag{11.4}$$

Τετραγωνίζοντας την (11.4), λαμβάνουμε :

$$\begin{aligned}
g^2(x) &= \Big(\sum_{i=0}^{\infty} a_i x^i\Big)\Big(\sum_{i=0}^{\infty} a_i x^i\Big) = \sum_{n=0}^{\infty}\Big(\sum_{i=0}^{n} a_i a_{n-i}\Big)x^n \\
&= \frac{1}{x}\sum_{n=0}^{\infty}\Big(\sum_{i=0}^{n} a_i a_{n-i}\Big)x^{n+1} \\
&= \frac{1}{x}\sum_{m=1}^{\infty}\Big(\sum_{i=0}^{m-1} a_i a_{m-1-i}\Big)x^{m} && m = n+1 \\
&= \frac{1}{x}\sum_{m=1}^{\infty} a_m x^m && \text{λόγω της (11.3)} \\
&= \frac{1}{x}(g(x) - a_0 x^0) = \frac{1}{x}(g(x) - 1)
\end{aligned}$$

και τελικά $x\,g^2(x) - g(x) + 1 = 0$, η οποία έχει λύση, ως προς $g(x)$ :

$$g(x) = \frac{1 \pm \sqrt{1-4x}}{2x},$$

επειδή όμως πρέπει $g(0) = a_0 = C_1 = 1$, μόνη δεκτή λύση είναι η

$$g(x) = \frac{1 - \sqrt{1-4x}}{2x} \tag{11.5}$$

Το ανάπτυγμα της $\sqrt{1-4x}$ δίνεται από την διωνυμική σειρά (βλέπε, για παράδειγμα, [95]) και είναι :

$$\sqrt{1-4x} = (1 + (-4x))^{1/2} = \sum_{m=0}^{\infty} \binom{1/2}{m} (-4x)^m \tag{11.6}$$

Έτσι, τελικά η (11.5) γράφεται (όπου $m = n+1$) :

$$g(x) = \frac{1}{2x}\Big(1 - \sum_{m=0}^{\infty} \binom{1/2}{m} (-4x)^m\Big) = \sum_{n=0}^{\infty} \binom{1/2}{n+1} (-1)^n\, 2^{2n+1}\, x^n$$

δηλαδή για τους συντελεστές ισχύει

$$a_n = \binom{1/2}{n+1} (-1)^n\, 2^{2n+1}$$

Με επαγωγή στο $\mathbb{N}$ μπορούμε εύκολα να δείξουμε ότι

$$a_n = \frac{1}{n+1}\binom{2n}{n}$$

οπότε από την (11.2), μετά από πράξεις, προκύπτει

$$C_n = \frac{1}{2n-1}\binom{2n-1}{n} \tag{11.7}$$

για $n \geq 1$ (βλέπε και [92, 108]).

## 11.5 Παραγωγή $S$-όρων μήκους $n$

Η παραγωγή όλων των ($C_n$ το πλήθος) $S$-όρων μήκους $n$ είναι ένα ενδιαφέρον πρόβλημα, αφού προκειμένου να μελετήσουμε τους $S$-όρους χρειαζόμαστε έναν συστηματικό τρόπο που να μας τους εμφανίζει. Η μέθοδος που χρησιμοποιείται έχει πολλές ομοιότητες με αυτήν της απαρίθμησης.

Έστω, $A_n$ το σύνολο των $S$-όρων μήκους $n$. Τότε έχουμε τον αναδρομικό τύπο :

$$A_1 = \{S\}, \quad A_n = \bigcup_{i=1}^{n-1} \{(x \circ y) \mid x \in A_i,\, y \in A_{n-i}\},\ \text{για } n > 1$$

Το πρόβλημα με τον παραπάνω αλγόριθμο είναι η υψηλή υπολογιστική πολυπλοκότητά του : Αν ζητηθεί να παραχθούν όλοι οι $S$-όροι μήκους $n$, τότε ο αλγόριθμος αναγκαστικά υπολογίζει και όλους τους όρους μήκους μικροτέρου του $n$. Επιπλέον, κάθε ένας από τους μικρότερους όρους ενδέχεται να χρησιμοποιείται περισσότερες από μία φορές από τον αλγόριθμο, ως συστατικό στοιχείο διαφορετικών όρων μήκους $n$. Επομένως, δεν είναι τόσο απλό να οργανωθεί ο αλγόριθμος κατά τέτοιον τρόπο ώστε να έχουμε οικονομία χώρου και αποφυγή πολλαπλής παραγωγής επαναλαμβανόμενων όρων μικρότερου μήκους.

## 11.6 Διάταξη $S$-όρων

Σκοπεύουμε να ορίσουμε μία διάταξη (βλέπε ορισμό 1.4.8) μεταξύ των θεμελιωδών $S$-όρων, την οποία συμβολίζουμε με «$<$». Πρώτα από όλα, αν για δύο όρους ισχύει $\text{len}(t_1) < \text{len}(t_2)$, τότε $t_1 < t_2$. Τώρα, για την διάταξη των $S$-όρων του ιδίου μήκους, θα μας φανεί χρήσιμη η προθεματική γραφή των $S$-όρων : Σε αυτήν την περίπτωση οι $S$-όροι ουσιαστικά αναπαρίστανται

ως συμβολοσειρές αποτελούμενες από «$o$» και «$S$», οπότε έχει νόημα να τις διατάξουμε *λεξικογραφικά* (κατά σύμβαση θεωρούμε ότι το «$o$», δηλαδή το σύμβολο της εφαρμογής, προηγείται του «$S$»). Διατυπώνουμε συγκεντρωμένα και τυπικά τον ορισμό :

**11.6.1 Ορισμός** (Διάταξη $S$-όρων)**.** Έστω $t_1$, $t_2$ $S$-όροι, τότε $t_1 < t_2$ αν και μόνον αν $\text{len}(t_1) < \text{len}(t_2)$ ή όταν $\text{len}(t_1) = \text{len}(t_2)$, τότε η προθεματική γραφή του $t_1$ προηγείται αυτής του $t_2$ στην λεξικογραφική διάταξη.

Αν τώρα ορίσουμε το σύνολο

$$\{(a, b) \mid a, b \in \mathbb{N} \text{ με } a \geq 1,\ 1 \leq b \leq C_a\}$$

και αντιστοιχίσουμε στο στοιχείο $(a, b)$ του συνόλου τον $S$-όρο μήκους $a$ που εμφανίζεται $b$-οστός στην λεξικογραφική διάταξη μεταξύ των $S$-όρων μήκους $a$, έχουμε ορίσει μία αμφιμονοσήμαντη συνάρτηση από το παραπάνω σύνολο στο σύνολο των θεμελιωδών $S$-όρων.

**11.6.2 Παράδειγμα.** Δίνουμε την διάταξη των $S$-όρων μήκους μέχρι και 4 : $S < SS < A < B < AS < BS < SS(SS) < SA < SB$.

*11.6.3 Παρατήρηση.* Στην ουσία ορίσαμε μία λεξικογραφική διάταξη στην οποία έχει όμως προτεραιότητα το μήκος των $S$-όρων.

Μεταξύ των όρων ιδίου μήκους, έστω $n > 1$, ο πρώτος όρος στην λεξικογραφική διάταξη είναι ο

$$t_n^{\text{first}} = \underbrace{o \ldots o}_{n-1\ o}\underbrace{S \ldots SS}_{n\ S}$$

και τελευταίος ο

$$t_n^{\text{last}} = \underbrace{oS \ldots oS}_{n-1\ oS} S$$

Θα μπορούσαμε να παραγάγουμε όλους τους όρους μήκους $n$ αν αρχίζαμε με τον πρώτο όρο στην διάταξη και κάθε φορά βρίσκαμε τον αμέσως επόμενο στην διάταξη, μέχρι να φτάσουμε στον τελευταίο όρο.

## 11.7 Εύρεση επομένου $S$-όρου

Στην προηγούμενη ενότητα δώσαμε μία διάταξη των (θεμελιωδών) $S$-όρων ή ισοδύναμα των πλήρων δυαδικών δένδρων. Σε αυτή την ενότητα θα δώσουμε έναν αλγόριθμο που δεδομένου ενός όρου δίνει τον επόμενο στην παραπάνω ορισθείσα διάταξη.

*Αλγόριθμος για την εύρεση του επομένου $S$-όρου.*

**Είσοδος :** ένας $S$-όρος σε προθεματική αναπαράσταση

**Έξοδος :** ο επόμενος του

- Αν ο όρος δεν τελειώνει σε $oSS$ [1η περίπτωση], τότε :
  - Βρες τον δεξιότερο υποόρο $oSS$ που έχει δεξιότερά του άλλον υποόρο $S$.
  - Ενάλλαξε την θέση αυτού με τον αμέσως δεξιότερο $S$.
- Αλλιώς (ο όρος τελειώνει σε $oSS$) [2η περίπτωση] :
  - Βρες τον δεξιότερο εσωτερικό υποόρο $oSS$ που έχει δεξιότερά του άλλον υποόρο $S$ (προφανώς αυτός δεν είναι ο υποόρος $oSS$ στο τέλος της προθεματικής αναπαράστασης).
  - Μέτρησε το πλήθος των «$o$» (έστω $n_o$) και το πλήθος των «$S$» (έστω $n_S$) που ακολουθούν (από τα δεξιά) τον ευρεθέντα στο παραπάνω βήμα $oSS$.
  - Ο επόμενος όρος έχει την ίδια προθεματική αναπαράσταση (αρχίζοντας από τα αριστερά) με τον αρχικό μέχρι τον παραπάνω ευρεθέντα υποόρο $oSS$. Κατόπιν, αντικατάστησε αυτόν τον $oSS$ με τον $S$ και μετά γράψε διαδοχικά $n_o + 1$ «$o$» και $n_S + 1$ «$S$».

Παρατηρήσατε ότι στον $t_n^{\text{last}}$, ο μόνος υποόρος $oSS$ είναι στο τέλος της προθεματικής αναπαράστασης και άρα δεν υπάρχει υποόρος $oSS$ που να έχει δεξιότερα $S$ και άρα ο αλγόριθμος δεν λειτουργεί με είσοδο τον παραπάνω όρο. Μπορούμε όμως εύκολα να τον τροποποιήσουμε ούτως ώστε να δίνει τον επόμενο όρο, ο οποίος είναι φυσικά ο $t_{n+1}^{\text{first}}$.

Σκοπεύουμε να αποδείξουμε την ορθότητα του παραπάνω αλγορίθμου.

Ας σημειωθεί, πως το πλήθος των συμβολοσειρών μήκους $2n-1$ που αποτελούνται από $n$ «$S$» και $n-1$ εφαρμογές είναι όσο και το πλήθος των συνδυασμών των $2n-1$ ανά $n$, δηλαδή μεγαλύτερο από το $C_n$. Αυτό προκύπτει από τον περιορισμό που επιβάλλει η δημιουργία όρων στην μορφή της συμβολοσειράς (εφαρμογή ακολουθούμενη από δύο όρους) και σημαίνει ότι κάποιες από τις συμβολοσειρές δεν αντιστοιχούν σε $S$-όρους. Μπορούμε να περιορίσουμε περισσότερο το πλήθος των θεωρουμένων συμβολοσειρών, αν παρατηρήσουμε ότι κάθε όρος σε προθεματική μορφή αρχίζει με μία εφαρμογή και τελειώνει σε δύο «$S$». Ακόμη και τότε όμως το πλήθος είναι μεγαλύτερο από το πλήθος των $S$-όρων.

Επειδή ο αλγόριθμός μας στην ουσία επεξεργάζεται συμβολοσειρές, θα πρέπει πριν από όλα να εξασφαλίσουμε ότι δεν υπάρχει περίπτωση να δώσει ως έξοδο συμβολοσειρά που δεν αντιστοιχεί σε $S$-όρο.

Για να δείξουμε την ορθότητα θα μελετήσουμε ξεχωριστά τις δύο περιπτώσεις.

**Πρώτη περίπτωση.** Είναι βέβαιο ότι η συμβολοσειρά από «$o$» και «$S$» που δίνει ως έξοδο ο αλγόριθμος αντιστοιχεί όντως σε $S$-όρο λόγω της εναλλαγής υποόρων εντός της συμβολοσειράς εισόδου (που θεωρείται δεδομένο ότι είναι $S$-όρος σε προθεματική αναπαράσταση).

Επίσης, εξασφαλίζεται ότι ο αλγόριθμος δεν δίνει ως έξοδο όρο που να μην είναι μετά στην διάταξη. Πράγματι, αφού εναλλάσσεται ένας όρος $oSS$ με έναν δεξιότερα ευρισκόμενό του $S$, το αριστερότερο σύμβολο που αλλάζει στην συμβολοσειρά είναι ένα «$o$» στην θέση του οποίου τοποθετείται ένα «$S$», άρα οι δύο συμβολοσειρές (άρα και οι όροι) έχουν την δέουσα λεξικογραφική διάταξη.

Το μόνο πρόβλημα που θα μπορούσε να έχει ο αλγόριθμος, λοιπόν, είναι το εξής : να μην κατασκευάζει πάντοτε τον ακριβώς επόμενο όρο, δηλαδή σε κάποιες περιπτώσεις να τον υπερπηδά και να δίνει ως έξοδο κάποιον άλλο όρο, που βρίσκεται βεβαίως πιο μετά στην διάταξη.

Υπενθυμίζουμε ότι στην πρώτη περίπτωση, που μελετάμε, ο δεξιότερος δίφυλλος υποόρος του όρου εισόδου δεν είναι ο $oSS$, ή ισοδύναμα η προθεματική αναπαράσταση δεν τελειώνει σε $oSS$. Αυτό συνεπάγεται ότι η αντίστοιχη συμβολοσειρά τελειώνει σε $SSS$. Επομένως, αν σαρώσουμε την συμβολοσειρά από τα δεξιά προς τα αριστερά, η πρώτη εφαρμογή που θα συναντήσουμε θα ακολουθείται από ένα πλήθος «$S$» μεγαλύτερο ή ίσο του 3. Παρατηρήσατε ότι σε αυτήν την περίπτωση, η εναλλαγή του δεξιότερου υποόρου $oSS$ με τον αμέσως δεξιότερο του $S$ είναι ισοδύναμη με την εναλλαγή του δεξιότερου «$o$» με το «$S$» ακριβώς δεξιά του, κάτι που απλοποιεί σημαντικά την επεξεργασία που γίνεται στην συμβολοσειρά (βλέπε σχήμα 11.3). Η αλλαγή του δεξιότερου «$o$» εξασφαλίζει και ότι δεν υπάρχει ενδιάμεσος όρος που πιθανόν υπερπηδάται.


Σχήμα 11.3 – Εύρεση επομένου $S$-όρου : πρώτη περίπτωση

**Δεύτερη περίπτωση.** Μένει να αναλύσουμε την κατάσταση όταν η προθεματική αναπαράσταση του όρου εισόδου τελειώνει σε $oSS$. Η απόδειξη της ορθότητας του αλγορίθμου στην δεύτερη περίπτωση όρου εισόδου είναι λίγο πιο δύσκολη από ότι στην πρώτη περίπτωση. Στο σχήμα 11.4 δίνουμε την εφαρμογή του αλγορίθμου στην δεύτερη περίπτωση : το πρώτο σύμβολο «$o$» που σημειώνεται είναι και το αριστερότερο που αλλάσσεται στην προθεματική

αναπαράσταση, δηλαδή αριστερά αυτού η προθεματική αναπαράσταση παραμένει ίδια.

$$\ldots oSS\underbrace{S\ldots S}_{k\ S}\underbrace{oS\ldots oS}_{l\ oS}oSS \overset{\text{next}}{\to} \ldots S\underbrace{ooo\ldots o}_{l+2\ o}\underbrace{SS\ldots SSS}_{k+l+3\ S}$$

Σχήμα 11.4 – Εύρεση επομένου $S$-όρου : δεύτερη περίπτωση

Ας δείξουμε, πρώτα, ότι ο αλγόριθμος στην δεύτερη περίπτωση δίνει συμβολοσειρά που αντιστοιχεί σε προθεματική αναπαράσταση $S$-όρου. Παρομοίως, όπως και στην πρώτη περίπτωση, θα δείξουμε ότι στην ουσία έχουμε αντικατάσταση υποόρων εντός του όρου εισόδου. Για να δούμε καλύτερα τους εμπλεκόμενους υποόρους γράφουμε τους όρους του σχήματος 11.4 ως εξής :

$$\ldots oSS\underbrace{S\ldots S}_{k\ S}\, t_{l+2}^{\text{last}} \overset{\text{next}}{\to} \ldots S\, t_{l+3}^{\text{first}}\underbrace{S\ldots S}_{k\ S}$$

Δηλαδή, θεωρούμε τους εξής $k+2$ τελευταίους υποόρους του όρου εισόδου από τα αριστερά προς τα δεξιά : τον $oSS$, τους $k$ το πλήθος $S$ και τον $t_{l+2}^{\text{last}}$. Αντί για αυτούς τοποθετούμε στις αντίστοιχες θέσεις του όρο εξόδου τους (επίσης $k+2$ το πλήθος και από τα αριστερά προς τα δεξιά) : τον $S$, τον $t_{l+3}^{\text{first}}$ και $k$ το πλήθος $S$. Ουσιαστικά, απλώς, ο $oSS$ αντικαταστάθηκε από $S$, ο πρώτος $S$ από $t_{l+3}^{\text{first}}$ και ο $t_{l+2}^{\text{last}}$ από $S$.

Επίσης, εύκολα φαίνεται ότι δεν μπορεί παρά ο όρος εξόδου να είναι μετά στην (λεξικογραφική) διάταξη από τον όρο εισόδου, δεδομένου ότι το αριστερότερο στοιχείο της συμβολοσειράς που αλλάσσεται είναι ένα «$o$» που γίνεται «$S$».

Μένει να δείξουμε ότι δεν υπάρχει κάποιος ενδιάμεσος στην διάταξη μεταξύ του όρου εισόδου και του όρου εξόδου που να υπερπηδάται από τον αλγόριθμο. Παρατηρώντας την συμβολοσειρά εξόδου, δεδομένου ότι η συμβολοσειρά $o\ldots oS\ldots S$ στο τέλος είναι η λεξικογραφικά μικρότερη στο μήκος $k+2l+5$, έχουμε να πούμε ότι, όντως, αν το αριστερότερο στοιχείο που αλλάζει είναι το συγκεκριμένο «$o$» (που γίνεται $S$) ακριβώς πριν από την συμβολοσειρά $o\ldots oS\ldots S$, δεν υπάρχει ενδιάμεσος όρος. Μήπως όμως η υπόθεση για το αριστερότερο στοιχείο που αλλάζει δεν είναι σωστή; Μήπως το αριστερότερο στοιχείο που αλλάζει βρίσκεται δεξιότερα; Θα δείξουμε ότι αυτό είναι άτοπο. Πράγματι, αν υποθέσουμε ότι το στοιχείο που αλλάζει βρίσκεται δεξιότερα δεν μπορεί να είναι παρά κάποιο από τα «$o$» του υποόρου $t_{l+2}^{\text{last}}$ (επειδή θέλουμε επόμενο όρο στην λεξικογραφική διάταξη). Επειδή οι αλλαγές περιορίζονται στα δεξιά αυτού του «$o$» του υποόρου $t_{l+2}^{\text{last}}$ και ο υποόρος $t_{l+2}^{\text{last}}$ πηγαίνει μέχρι το τέλος της προθεματικής αναπαράστασης του

όρου εισόδου, έχουμε άτοπο, αφού, διαισθητικά, είναι σαν να προσπαθούμε να βρούμε επόμενο όρο του $t_{l+2}^{\text{last}}$ με το ίδιο μήκος, πράγμα προφανώς αδύνατο όπως έχει οριστεί η λεξικογραφική διάταξη.

Τυπικά, αν θεωρήσουμε τον όρο $t_{l+2}^{\text{last}}$ και τον σαρώσουμε από τα δεξιά στα αριστερά κάθε φορά που συναντούμε ένα «$o$» έχουμε ήδη σαρώσει ακριβώς ένα παραπάνω «$S$» από ότι «$o$» (βλέπε για παράδειγμα στον πίνακα 11.1 την σάρωση για τον όρο $t_7^{\text{last}}$). Αυτό, όπως θα δείξουμε παρακάτω, είναι μία οριακή κατάσταση.

Πίνακας 11.1 – Σάρωση $oSoSoSoSoSS$ από τα δεξιά

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| όρος : | $o$ | $S$ | $o$ | $S$ | $o$ | $S$ | $o$ | $S$ | $o$ | $S$ | $o$ | $S$ | $S$ |
| σαρωμένα $o$ : | 6 | 5 | 5 | 4 | 4 | 3 | 3 | 2 | 2 | 1 | 1 | 0 | 0 |
| σαρωμένα $S$ : | 7 | 7 | 6 | 6 | 5 | 5 | 4 | 4 | 3 | 3 | 2 | 2 | 1 |

Αν προσπαθήσουμε να βρούμε επόμενο για κάποιον όρο της μορφής $t_{l+2}^{\text{last}}$ θα πρέπει να «προωθήσουμε» προς τα δεξιά ένα «$o$» και στην θέση του να τοποθετήσουμε ένα από τα δεξιότερα ευρισκόμενα «$S$». Αυτή όμως η εσωτερική αλλαγή θα σημαίνει ότι, κατά την σάρωση (όπως περιγράφηκε παραπάνω) του νέου όρου, σε κάποιο σημείο θα έχουμε ίσο πλήθος σαρωμένων «$o$» και «$S$» και αυτό είναι άτοπο, όπως μας πληροφορεί το παρακάτω λήμμα :

**11.7.1 Λήμμα.** *Σαρώνοντας την προθεματική αναπαράσταση ενός S-όρου από τα δεξιά στα αριστερά, σε κάθε στιγμή, το συνολικό πλήθος των «S» που συναντούμε είναι πάντοτε μεγαλύτερο από το συνολικό πλήθος των «o».*

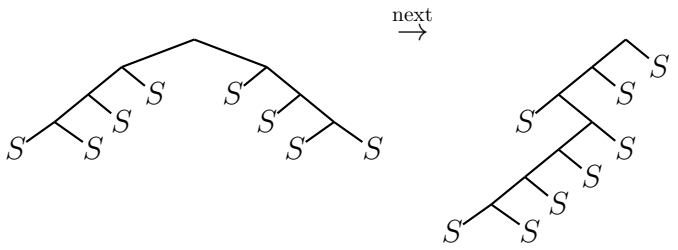
*Απόδειξη.* Με επαγωγή στην δομή των $S$-όρων. Για τον $S$-όρο μήκους 1 σε προθεματική αναπαράσταση, η ιδιότητα προφανώς ισχύει. Αν ισχύει για τις προθεματικές αναπαραστάσεις $x$ και $y$, θα δείξουμε ότι ισχύει και για την $oxy$ : Σαρώνοντας (από δεξιά) την $oxy$ αρχικά συναντούμε την $y$, οπότε η ιδιότητα ισχύει. Μόλις σαρώσουμε όλην την $y$ έχουμε σαρώσει ακριβώς ένα $S$ παραπάνω από όσα $o$, οπότε όσο σαρώνουμε την $x$ στην $oxy$ η ιδιότητα επίσης ισχύει. Τέλος, κατά την σάρωση του αριστερότερου $o$ της $oxy$ έχουμε σαρώσει πάλι ακριβώς ένα $S$ παραπάνω από όσα $o$, αφού το $oxy$ είναι αναπαράσταση $S$-όρου. □

**11.7.2 Παράδειγμα.** Στον πίνακα 11.2 φαίνεται η ισχύς του λήμματος για τον όρο $ooSSoSooSSoSS$.

*11.7.3 Παρατήρηση.* Το λήμμα 11.7.1 δεν ισχύει μόνον για όρους εισόδου της δεύτερης περίπτωσης, αλλά για κάθε $S$-όρο.

Πίνακας 11.2 – Σάρωση $ooSSoSooSSoSS$ από τα δεξιά

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| όρος : | $o$ | $o$ | $S$ | $S$ | $o$ | $S$ | $o$ | $o$ | $S$ | $S$ | $o$ | $S$ | $S$ |
| σαρωμένα $o$ : | 6 | 5 | 4 | 4 | 4 | 3 | 3 | 2 | 1 | 1 | 1 | 0 | 0 |
| σαρωμένα $S$ : | 7 | 7 | 7 | 6 | 5 | 5 | 4 | 4 | 4 | 3 | 2 | 2 | 1 |




Σχήμα 11.5 – Εύρεση επομένου του $AS(SB)$

**11.7.4 Παράδειγμα.** Στο σχήμα 11.5 φαίνεται το αποτέλεσμα της εφαρμογής του αλγορίθμου στον όρο $AS(SB)$ (υπάγεται στην δεύτερη περίπτωση).

Μία υλοποίηση του αλγορίθμου εύρεσης επομένου όρου βρίσκεται στην συνάρτηση nextSterm στο αρχείο sterm.h (βλέπε σελίδα 161 κ.ε.), ενώ ένα πρόγραμμα που χρησιμοποιεί αυτήν την υλοποίηση είναι το nextsterm, στην ενότητα A.7, σελίδα 224.

# Κεφάλαιο 12

# Ο $S$-λογισμός

Ο $S$-λογισμός είναι ένα υποσύστημα της συνδυαστικής λογικής (βλέπε παράδειγμα 5.2.1, σελίδα 49), το οποίο περιέχει μόνον τον *συνδυαστή* $S$ με κανόνα μεταγραφής :

$$Sxyz \to xz(yz).$$

Η συνδυαστική λογική, όπως θα δείξουμε παρακάτω, χαρακτηρίζεται από *συνδυαστική πληρότητα*, ακόμα και αν περιέχει έναν μόλις επιπλέον του $S$ συνδυαστή, τον $K$, με κανόνα μεταγραφής :

$$Kxy \to x.$$

Αυτό έχει ως συνέπεια ότι η συνδυαστική λογική είναι πλήρης και κατά Turing.

Όμως, το ερώτημα είναι αν μία απλούστερη θεωρία όπως ο $S$-λογισμός χαρακτηρίζεται από κάποιες αποκρίσιμες ιδιότητες παρουσιάζει αρκετό ενδιαφέρον. Όπως θα δούμε παρακάτω, για αρκετά μη αποκρίσιμα προβλήματα στην συνδυαστική λογική, τα αντίστοιχα προβλήματα του $S$-λογισμού έχουν αλγόριθμο επίλυσης. Το σημαντικότερο από αυτά ίσως είναι το αποκαλούμενο *πρόβλημα τερματισμού* το οποίο είναι σαφώς μη επιλύσιμο στην συνδυαστική λογική, αλλά επιλύσιμο στον $S$-λογισμό, όπως απέδειξε πρόσφατα ο Waldmann [104, 105]. Ο Ζάχος [110] έδωσε μία απλούστερη απόδειξη, το περίγραμμα της οποίας παρουσιάζουμε στο κεφάλαιο 13. Παρ' όλα αυτά, υπάρχουν ακόμα αρκετά ανοικτά (σχετικά με το αν είναι επιλύσιμα ή όχι) προβλήματα στα πλαίσια του $S$-λογισμού, με χαρακτηριστικότερο παράδειγμα το *λεκτικό πρόβλημα* (word problem).

## 12.1 Συνδυαστική λογική

Η συνδυαστική λογική (combinatory logic : **CL**) είναι ένα σύστημα μεταγραφής όρων συγκεκριμένης μορφής.

Στο παράδειγμα 5.2.1, σελίδα 49, δώσαμε αυτό που συνήθως ονομάζεται συνδυαστική λογική, βασισμένη στους συνδυαστές $S$, $K$, $I$. Τώρα, θα δώσουμε έναν γενικότερο ορισμό :

**12.1.1 Ορισμός.** Ένας όρος στην συνδυαστική λογική ονομάζεται *γνήσιος* ή *αγνός* (pure) όταν αποτελείται μόνον από μεταβλητές και παρουσίες της εφαρμογής.

**12.1.2 Ορισμός.** Η *συνδυαστική λογική* με *συνδυαστές βάσης* $C_1$, ..., $C_n$ συμβολίζεται με $\mathbf{CL}(C_1, \ldots, C_n)$ και είναι το σύστημα μεταγραφής όρων το οποίο έχει

- *αλφάβητο συμβόλων* αποτελούμενο ακριβώς από τις $n$ σταθερές $C_1$, ..., $C_n$ και τον δυαδικό τελεστή *εφαρμογής* (ο οποίος παριστάνεται με «∘» ως ενθεματικός τελεστής) [1],
- *σύνολο κανόνων* αποτελούμενο ακριβώς από $n$ κανόνες, έναν για κάθε *συνδυαστή βάσης* της μορφής $C_i x_1 x_2 \cdots x_{k_i} \to t_i$, όπου $t_i$ γνήσιος όρος, που δεν περιέχει μεταβλητές εκτός των $x_1$, $x_2$, ..., $x_{k_i}$.

**12.1.3 Παράδειγμα.** Η συνδυαστική λογική όπως ορίστηκε στο παράδειγμα 5.2.1 είναι το σύστημα $\mathbf{CL}(S, K, I)$.

Υπενθυμίζουμε τους κανόνες για τους συνδυαστές $S$, $K$, $I$ :

$$\begin{aligned} Sxyz &\to xz(yz) \\ Kxy &\to x \\ Ix &\to x \end{aligned}$$

Τυπικά, οι συνδυαστές μπορούν να περιγραφούν ως συναρτήσεις με την βοήθεια του λ-λογισμού :

$$S = \lambda xyz.xz(yz) \tag{12.1}$$

$$K = \lambda xy.x \tag{12.2}$$

$$I = \lambda x.x \tag{12.3}$$

Ο συνδυαστής $S$ δρα επί τριών ορισμάτων, ο $K$ επί δύο και ο $I$ επί ενός.

Παρατηρούμε ότι ένα σύστημα συνδυαστικής λογικής εξαρτάται άμεσα από τους συνδυαστές βάσης που περιέχει. Μπορεί να δει κανείς, για παράδειγμα, εύκολα, ότι το $\mathbf{CL}(I)$ συμπεριφέρεται πολύ διαφορετικά από το $\mathbf{CL}(K)$. Ενδέχεται όμως σε κάποιο σύστημα, με κάποιον όρο που αποτελείται από συνδυαστές βάσης να είναι δυνατόν να εξομοιωθεί η συμπεριφορά κάποιου άλλου

1. Συνήθως, όπως αναφέρεται και στο παράδειγμα 5.2.1, η εφαρμογή $(x \circ y)$ συμβολίζεται απλώς με παράθεση $(xy)$ ενώ, για να μειώσουμε το πλήθος των παρενθέσεων, θεωρούμε ότι η εφαρμογή προσεταιρίζει από τα αριστερά, δηλαδή $xyz = ((xy)z)$.

συνδυαστή, που δεν ανήκει στους συνδυαστές βάσης (πιθανόν, όμως, με περισσότερα του ενός βήματα μεταγραφής). Χαρακτηριστικό παράδειγμα είναι ο συνδυαστής ταυτότητας $I$ που μπορεί να εξομοιωθεί, για παράδειγμα, από τον όρο $SKK$, όπως θα δείξουμε παρακάτω.

Ένα σύστημα του οποίου οι συνδυαστές βάσης έχουν την ιδιότητα να εξομοιώσουν οποιονδήποτε συνδυαστή ονομάζεται *συνδυαστικά πλήρες* :

**12.1.4 Ορισμός.** Ένα σύστημα συνδυαστικής λογικής λέγεται *συνδυαστικά πλήρες* (combinatorially complete) αν για κάθε $n > 0$ και για κάθε γνήσιο όρο $t$ με $\mathrm{Var}(t) \subseteq \{x_1, \ldots, x_n\}$ υπάρχει θεμελιώδης όρος $s$ τέτοιος ώστε $sx_1 \ldots x_n \xrightarrow{*} t$.

**12.1.5 Πρόταση.** *Το* $\mathbf{CL}(S, K, I)$ *είναι συνδυαστικά πλήρες.*

*Απόδειξη.* Για κάθε όρο $t$ θα κατασκευάσουμε κάποιον συγκεκριμένο όρο $\lfloor t \rfloor_x$ τέτοιον ώστε $\lfloor t \rfloor_x x \xrightarrow{*} x$ και στον $\lfloor t \rfloor_x$ δεν εμφανίζεται η μεταβλητή $x$, ενώ $\mathrm{Var}(\lfloor t \rfloor_x) \subseteq \mathrm{Var}(t)$.

Η κατασκευή γίνεται ως εξής :

– Αν $t = x \in \mathcal{V}$, τότε $\lfloor x \rfloor_x = I$.

– Αν $x \notin t$, τότε $\lfloor t \rfloor_x = Kt$.

– Αν $t = uv$, τότε $\lfloor uv \rfloor_x = S \lfloor u \rfloor_x \lfloor v \rfloor_x$.

Τότε ο θεμελιώδης όρος $s$ για τον οποίο $sx_1 \ldots x_n \xrightarrow{*} t$ είναι απλώς ο $s = \lfloor \ldots \lfloor \lfloor t \rfloor_{x_1} \rfloor_{x_2} \ldots \rfloor_{x_n}$. □

Δεδομένου ότι οι συνδυαστές $S$, $K$ μπορούν να εξομοιώσουν τον συνδυαστή ταυτότητας $I$ ως εξής :

$$SKtx \to Kx(tx) \to x$$

όπου $t$ οποιοσδήποτε όρος (π.χ. $t = K$), άμεσο είναι το παρακάτω :

**12.1.6 Πόρισμα.** *Το* $\mathbf{CL}(S, K)$ *είναι συνδυαστικά πλήρες.*

Υπάρχουν επίσης συνδυαστές που μπορούν από μόνοι τους να αποτελέσουν βάση για ένα συνδυαστικά πλήρες σύστημα. Μάλιστα, ο Schönfinkel [91] δίνει μία μέθοδο για την κατασκευή ενός τέτοιου συνδυαστή. Δυστυχώς, ένας τέτοιος συνδυαστής δεν έχει την τόσο απλή μορφή που έχουν οι $S$, $K$ ως συναρτήσεις του λ-λογισμού και δεν μπορεί να περιγραφεί με την εφαρμογή του συνδυαστή σε ένα σταθερό πλήθος μεταβλητών που δίνει έναν γνήσιο όρο, όπως στον ορισμό 12.1.2, σελίδα 134.

Ο Fokker στο [39] παραθέτει από την βιβλιογραφία διαφόρους συνδυαστές (των Meredith, Böhm, Barendregt, Rosser) που αποτελούν από μόνοι τους

βάση για συνδυαστικά πλήρες σύστημα και δίνει έναν απλό συνδυαστή με αυτήν την ιδιότητα, τον :

$$X = \lambda f.fS(\lambda xyz.x)$$

όπου το $S$ από την (12.1). Οι $K$, $S$ εξομοιώνονται ως εξής :

$$K\colon XX, \qquad S\colon X(XX).$$

Στην [3] ο Barker δίνει την απλή στον ορισμό της και πλήρη κατά Turing γλώσσα προγραμματισμού Iota, η οποία βασίζεται σε έναν τέτοιο συνδυαστή, τον $\iota$ :

$$\iota = \lambda f.fSK$$

όπου τα $S$, $K$ δίνονται από τις (12.1), (12.2). Οι γνωστοί μας συνδυαστές εξομοιώνονται ως εξής :

$$I\colon \iota\iota, \qquad K\colon \iota(\iota(\iota\iota)), \qquad S\colon \iota(\iota(\iota(\iota\iota))).$$

## 12.2 Ιδιότητες $S$-λογισμού

Ο $S$-λογισμός είναι απλώς το σύστημα $\mathbf{CL}(S)$. Τα αντικείμενα μεταγραφής του $\mathbf{CL}(S)$ είναι $S$-όροι. Προφανώς, το σύστημα $\mathbf{CL}(S)$ δεν είναι συνδυαστικά πλήρες : Για παράδειγμα, δεν είναι δυνατόν να «σβήσει» κανείς κάτι από έναν $S$-όρο, αφού ο μόνος κανόνας που υπάρχει στο $\mathbf{CL}(S)$ είναι *μη διαγράφων* (βλέπε ορισμό 5.4.1).

Το σύστημα του $S$-λογισμού είναι *ορθογώνιο* (βλέπε ορισμό 7.1.2 στην σελίδα 69). Αυτό σημαίνει ότι επιπλέον (από το θεώρημα 7.2.1 στην σελίδα 72) είναι και *συμβάλλον*. Λόγω της συμβολής, από την πρόταση 3.4.4 στην σελίδα 29, κάθε όρος έχει το πολύ μία κανονική μορφή (*μοναδικότητα κανονικών μορφών*). Επίσης, από την πρόταση 7.3.3 στην σελίδα 76, επειδή το σύστημα του $S$-λογισμού είναι μη διαγράφον, για κάθε $S$-όρο ισχύει το εξής : *είτε έχει κανονική μορφή, είτε έχει άπειρη αλυσίδα αναγωγών.*

Όλους τους $S$-όρους μπορούμε να τους περιγράψουμε με την εξίσωση σταθερού σημείου : $\mathcal{M} = S + \mathcal{M}\mathcal{M}$. Οι όροι που είναι κανονικές μορφές περιγράφονται παρομοίως από την εξίσωση σταθερού σημείου : $\mathcal{N} = S + S\mathcal{N} + S\mathcal{N}\mathcal{N}$. Υπάρχουν δηλαδή όροι που έχουν κανονική μορφή. Το ερώτημα είναι αν υπάρχουν και όροι που έχουν άπειρη αλυσίδα αναγωγών. Ένας τέτοιος όρος είναι ο $SASA$ (βλέπε παράδειγμα 12.4.1 για μία απόδειξη με την μέθοδο Ζάχου [108]).

Πολλές φορές μας ενδιαφέρει ένας υποόρος του όρου που προκύπτει μετά από μία αναγωγή :

**12.2.1 Ορισμός.** Για $S$-όρους $s$, $t$, ορίζουμε την σχέση $s \xrightarrow{\bigcirc} t$, αν υπάρχει περιβάλλον $C$ τέτοιο ώστε $s \to C[t]$.

Ορίζουμε επίσης το μεταβατικό και το μεταβατικό ανακλαστικό κλείσιμο της $\xrightarrow{\bigcirc}$, τα οποία συμβολίζουμε $\xrightarrow{\oplus}$ και $\xrightarrow{\circledast}$ αντιστοίχως.

## 12.3 Βρόχοι και κύκλοι

Όπως, είπαμε υπάρχουν στον $S$-λογισμό όροι που δεν κανονικοποιούνται. Όμως, οι άπειρες αλυσίδες αναγωγής, που φαίνεται να μπορούν να περιγραφούν με την μέθοδο Ζάχου, δεν φαίνεται να είναι δυνατόν να περιγραφούν με άλλον απλούστερο τρόπο. Πιο συγκεκριμένα, εικάζεται [104] ότι στον $S$-λογισμό δεν παρατηρούνται βρόχοι :

**12.3.1 Ορισμός** (Βρόχος)**.** *Βρόχος* είναι μία αναγωγή : $X \xrightarrow{+} C[X\sigma]$, όπου $X$ : όρος που πιθανόν περιέχει μεταβλητές, $C$ : περιβάλλον, $\sigma$ : αντικατάσταση.

Αντί για αυτό, στο [104], ο Waldmann απέδειξε το εξής ασθενέστερο αποτέλεσμα :

**12.3.2 Πρόταση.** *Στο σύστημα* $\mathbf{CL}(S)$ *δεν παρατηρούνται θεμελιώδεις βρόχοι.*

**12.3.3 Ορισμός** (Θεμελιώδης βρόχος)**.** *Θεμελιώδης βρόχος* είναι μία αναγωγή $t \xrightarrow{+} C[t]$, όπου $t$ : θεμελιώδης όρος, $C$ : περιβάλλον.

Από το 1978, ο Ζάχος [110] είχε αποδείξει το ακόμα ασθενέστερο αποτέλεσμα :

**12.3.4 Πρόταση.** *Στο σύστημα* $\mathbf{CL}(S)$ *δεν παρατηρούνται κύκλοι.*

**12.3.5 Ορισμός** (Κύκλος)**.** *Κύκλος* είναι μία αναγωγή $t \xrightarrow{+} t$, όπου $t$ : όρος.

## 12.4 Η μέθοδος Ζάχου

Το 1978, στην διδακτορική του διατριβή [108], ο Ζάχος διατύπωσε μία μέθοδο για να αποδεικνύει κανείς ότι ένας $S$-όρος δεν έχει κανονική μορφή, περιγράφοντας μία ακολουθία υποόρων που εμφανίζονται στην άπειρη αλυσίδα αναγωγών.

Έστω ότι θέλουμε να αποδείξουμε ότι ο $S$-όρος $t$ δεν έχει κανονική μορφή. Ορίζουμε έναν όρο $D$ και μία ακολουθία όρων $D_{\{i\}}$, οριζόμενη με πρωταρχική αναδρομή, δηλαδή δίνουμε τον $D_0$ και ορίζουμε τον $D_{n+1}$ με την βοήθεια του όρου $D_n$. Τότε αρκεί να δείξουμε τα παρακάτω λήμματα, για να έχει ο όρος $t$ άπειρη αλυσίδα αναγωγών (ισοδύναμα να μην έχει κανονική μορφή) :

**Λήμμα 1.** $t \xrightarrow{\circledast} DD_0$.

**Λήμμα 2.** $DD_n \xrightarrow{\circledast} DD_{n+1}$, *για κάθε* $n \in \mathbb{N}$.

Συνήθως, για την απόδειξη του λήμματος 2, χρήσιμα είναι τα παρακάτω βοηθητικά λήμματα :

**Λήμμα Α.** $D_nM \xrightarrow{\circledast} D_0M$, *για κάθε* $n \in \mathbb{N}$.

**Λήμμα Β.** $D_0M \xrightarrow{\circledast} DM$.

Για να γίνουν τα παραπάνω περισσότερο κατανοητά, δίνουμε ένα απλό παράδειγμα εφαρμογής της μεθόδου :

**12.4.1 Παράδειγμα.** Έστω ο όρος $SASA$. Ορίζουμε :

$$D = SA(SA), \quad D_0 = S(SA)D, \quad D_{n+1} = SAD_n.$$

Αρχικά αποδεικνύουμε τα λήμματα Α και Β :

**Λήμμα Α.** $D_nM \xrightarrow{\circledast} D_0M$, *για κάθε* $n \in \mathbb{N}$.

*Απόδειξη.* $D_nM \equiv SAD_{n-1}M \xrightarrow{\bigcirc} D_{n-1}M \xrightarrow{\circledast} D_0M$ □

**Λήμμα Β.** $D_0M \xrightarrow{\circledast} DM$.

*Απόδειξη.* $D_0M \equiv S(SA)DM \xrightarrow{\bigcirc} DM$ □

Κατόπιν, αποδεικνύουμε το λήμμα 1 και μετά το λήμμα 2 (το τελευταίο με την βοήθεια των λημμάτων Α και Β) :[2]

**Λήμμα 1.** $SASA \xrightarrow{\circledast} DD_0$.

*Απόδειξη.* $SASA \longrightarrow AA(SA) \xrightarrow{2} A(SA)D \xrightarrow{2} SADD_0 \xrightarrow{\bigcirc} DD_0$ □

**Λήμμα 2.** $DD_n \xrightarrow{\circledast} DD_{n+1}$, *για κάθε* $n \in \mathbb{N}$.

*Απόδειξη.* $DD_n \equiv SA(SA)D_n \longrightarrow AD_nD_{n+1} \xrightarrow{②} D_nD_{n+1} \xrightarrow[\text{A}]{\circledast} D_0D_{n+1} \xrightarrow[\text{B}]{\circledast} DD_{n+1}$ □

Από τα λήμματα 1 και 2, προκύπτει άμεσα ότι $SASA\!\uparrow$.

2. Στις αποδείξεις χρησιμοποιούμε συχνά τον κανόνα δύο βημάτων (αναγωγών) για το $A$ : $Axy \xrightarrow{2} (xy)(Sxy)$, ενώ κάποιες φορές διατηρούμε τον υποόρο $xy$ του παραπάνω, συμβολικά : $Axy \xrightarrow{②} xy$. Παρομοίως, για τον όρο $B$.

Στο [108] περιέχονται αποδείξεις μη κανονικοποίησης για όλους τους μη κανονικοποιήσιμους όρους μεγέθους μέχρι και 9, με την παραπάνω μέθοδο.

Πάντως, πρέπει να παρατηρήσουμε ότι δεν είναι όλες οι αποδείξεις μη κανονικοποίησης με την μέθοδο Ζάχου απλές και σύντομες όπως στο παράδειγμα 12.4.1. Όσο μεγαλώνει το μήκος των όρων, εμφανίζονται όροι, για τους οποίους η απόδειξη είναι μακροσκελής, αλλά επιπλέον, πριν από αυτό, είναι δύσκολος ο εντοπισμός των κατάλληλων $D$, $D_0$, $D_{n+1}$, για να λειτουργήσει η απόδειξη. Σε αυτήν την περίπτωση, χρήσιμα είναι προγράμματα που βοηθούν στον εντοπισμό των $D$, $D_0$, $D_{n+1}$ (βλέπε ενότητα A.4, στην σελίδα 203). Μερικές δύσκολες περιπτώσεις αποδείξεων δίνονται στο κεφάλαιο 14.

Τέλος, η μέθοδος Ζάχου (όπως παρατηρείται στο [111]) μπορεί να χρησιμοποιηθεί και για την απόδειξη μη κανονικοποιησιμότητας συνόλων $S$-όρων. Για μία τέτοια απόδειξη, βλέπε ενότητα 14.3, στην σελίδα 155.

## 12.5 Αποκρισιμότητα προβλημάτων

Η αποκρισιμότητα της κανονικοποίησης των $S$-όρων παρέμενε ανοικτό πρόβλημα για περισσότερο από 20 χρόνια. Τελικά, το έλυσε ο Waldman [104, 105]. Στο κεφάλαιο 13 παρουσιάζουμε το περίγραμμα της απλούστερης απόδειξης του Ζάχου, που υπάρχει στο [110].

Το βασικότερο ανοικτό πρόβλημα στο σύστημα $\mathbf{CL}(S)$ είναι το *λεκτικό πρόβλημα* :

**12.5.1 Ορισμός** (Λεκτικό πρόβλημα)**.** Δίνονται δύο $S$-όροι $x$, $y$ (έστω και θεμελιώδεις). Ισχύει $x \overset{*}{\leftrightarrow} y$ ;

*12.5.2 Παρατήρηση.* Λόγω της αποκρισιμότητας της κανονικοποίησης και της δυνατότητας υπολογισμού της κανονικής μορφής σε περίπτωση κανονικοποίησης, το πρόβλημα είναι επιλύσιμο αν ένας τουλάχιστον εκ των $x$, $y$ κανονικοποιείται. Η δυσκολία είναι στην περίπτωση που αμφότεροι οι $x$, $y$ έχουν άπειρη αλυσίδα αναγωγής. Εικάζεται [104, 110] ότι και σε αυτήν την περίπτωση το πρόβλημα είναι αποκρίσιμο, πιθανώς περιγράφοντας με κάποιον πρότυπο τρόπο κάθε μία άπειρη αλυσίδα αναγωγών. Μία απόδειξη όμως φαίνεται ακόμα εκτός των δυνατοτήτων μας.

Ένα άλλο ανοικτό πρόβλημα είναι το εξής : Είναι πάντοτε δυνατόν να περιγραφεί μερικώς μία άπειρη αλυσίδα αναγωγών με την μέθοδο Ζάχου ; Στο κεφάλαιο 14 δίνουμε μερικές αποδείξεις μη κανονικοποίησης με την μέθοδο Ζάχου. Ακόμη και για την περίπτωση $B(SS)(B(BS))$, που φάνηκε αρχικά

δύσκολη, βρέθηκε περιγραφή με την μέθοδο Ζάχου (βλέπε ενότητα 14.2, σελίδα 153). Προς την κατεύθυνση εύρεσης της περιγραφής, μπορούν να βοηθήσουν προγράμματα βασισμένα στον κώδικα της ενότητας Α.4, στην σελίδα 203. Παρ' όλα αυτά, δεν έχει βρεθεί μία μηχανιστική διαδικασία που να δίνει για κάθε μη κανονικοποιήσιμο όρο την περιγραφή με την μέθοδο Ζάχου. Εικάζεται ότι και αυτό το πρόβλημα έχει θετική απάντηση.

# Κεφάλαιο 13

# Κανονικοποίηση

Σε αυτό το κεφάλαιο παρουσιάζουμε μία μέθοδο για την αποκρισιμότητα της κανονικοποίησης ενός $S$-όρου. Το πρόβλημα αν είναι αποκρίσιμο αν ένας $S$-όρος κανονικοποιείται ή όχι παρέμενε άλυτο για περισσότερο από 20 χρόνια. Στην διδακτορική του διατριβή [108], το 1978, ο Ζάχος είκασε την αποκρισιμότητα του προβλήματος :

> Nachdem aber alle versuchten Beweisansätze für die Nichtentscheidbarkeit gescheitert sind, wage ich die folgende *Vermutung: Es is entscheidbar ob ein S-Term eine Normalform hat.*

Από την τελευταία φράση, μεταφράζουμε στα ελληνικά :

**Εικασία.** *Είναι αποκρίσιμο αν ένας $S$-όρος έχει κανονική μορφή.*

Τελικά, ο Waldmann [104, 105] έδειξε ότι το πρόβλημα είναι αποκρίσιμο. Ο Ζάχος έδωσε μία απλούστερη απόδειξη στο [110], το περίγραμμα της οποίας θα παρουσιάσουμε ακολούθως. Για τις λεπτομέρειες της απόδειξης, παραπέμπουμε στο [110].

## 13.1 Αναγωγές σε σύνολα $S$-όρων

Τα σύνολα $S$-όρων τα συμβολίζουμε με καλλιγραφικά κεφαλαία λατινικά γράμματα, για παράδειγμα $\mathcal{X}$, $\mathcal{Y}$.

Ένα σημαντικό βήμα για την επίλυση του προβλήματος αποκρισιμότητας απετέλεσε η παρατήρηση ότι μπορούμε να επεκτείνουμε την έννοια της αναγωγής από $S$-όρους σε σύνολα $S$-όρων [104, 105, 110]. Πιο συγκεκριμένα :

**13.1.1 Ορισμός.** Για σύνολα $S$-όρων $\mathcal{X}$, $\mathcal{Y}$, ορίζουμε την σχέση $\mathcal{X} \xrightarrow{\oplus} \mathcal{Y}$, αν για κάθε $x \in \mathcal{X}$, υπάρχει $y \in \mathcal{Y}$, τέτοιο ώστε $x \xrightarrow{\oplus} y$ (βλέπε ορισμό 12.2.1).

Λέμε ότι το σύνολο $\mathcal{X}$ είναι *κλειστό ως προς* $\xrightarrow{\oplus}$ αν $\mathcal{X} \xrightarrow{\oplus} \mathcal{X}$, το οποίο και συμβολίζουμε : $\mathcal{X}\uparrow_{\oplus}$.

*13.1.2 Παρατήρηση.* Αν $\mathcal{X}\uparrow_{\oplus}$, τότε κανένας όρος του $\mathcal{X}$ δεν έχει κανονική μορφή.

## 13.2 Σύνολα μη κανονικοποιήσιμων $S$-όρων

Πριν παραθέσουμε τα σύνολα μη κανονικοποιήσιμων όρων που χρησιμοποιούνται στην απόδειξη της αποκρισιμότητας, θα δώσουμε κάποιους απαραίτητους συμβολισμούς και ορισμούς, καθώς και κάποιες σχετικές τεχνικές προτάσεις.

Για τα σύνολα $S$-όρων, χρησιμοποιούμε συμβολισμούς παρόμοιους με τις κανονικές παραστάσεις (regular expressions), για παράδειγμα, γράφουμε $S$ αντί για $\{S\}$, γράφουμε $\mathcal{X} + \mathcal{Y}$ αντί για $\mathcal{X} \cup \mathcal{Y}$ κ.τ.λ. Το σύνολο *όλων* των $S$-όρων παριστάνεται με $\mathcal{M}$ και προκύπτει από το ελάχιστο σταθερό σημείο της εξίσωσης :

$$\mathcal{M} = S + \mathcal{M}\mathcal{M}$$

Το σύνολο των $S$-όρων σε κανονική μορφή παριστάνεται με $\mathcal{N}$ και προκύπτει από το ελάχιστο σταθερό σημείο της εξίσωσης :

$$\mathcal{N} = S + S\mathcal{N} + S\mathcal{N}\mathcal{N}$$

Για κάθε σύνολο $S$-όρων $\mathcal{X}$ ορίζουμε το συμπλήρωμά του :

$$\overline{\mathcal{X}} \overset{\text{ορ.}}{=} \mathcal{M} - \mathcal{X}$$

Με τους παραπάνω συμβολισμούς, ορίζουμε :

$$\mathcal{Q}_1 \overset{\text{ορ.}}{=} \overline{S}, \quad \mathcal{Q}_2 \overset{\text{ορ.}}{=} \overline{S + SS}, \quad \mathcal{Q}_3 \overset{\text{ορ.}}{=} \overline{S + SS + B}$$

Άμεσα, προκύπτουν τα εξής αποτελέσματα :

$$\mathcal{Q}_1 = SS + \mathcal{Q}_2, \qquad \mathcal{M}\mathcal{Q}_i \subseteq \mathcal{Q}_{i+1} \subseteq \mathcal{Q}_i \quad \text{για } i \in \{1, 2\}, \qquad \mathcal{M}\mathcal{Q}_3 \subseteq \mathcal{Q}_3$$

**13.2.1 Ορισμός.** Για σύνολα $S$-όρων $\mathcal{X}$ (σύνολο προθέματος) και $\mathcal{Y}$ (σύνολο βάσης), ορίζουμε αναδρομικά το σύνολο $\mathcal{X}^n[\mathcal{Y}]$ ως εξής :

$$\mathcal{X}^n[\mathcal{Y}] = \begin{cases} \mathcal{Y}, & n = 0 \\ \mathcal{X}(\mathcal{X}^{n-1}[\mathcal{Y}]), & n > 0 \end{cases}$$

Το σύνολο όλων των παραπάνω όρων συμβολίζεται :

$$\mathcal{X}^*[\mathcal{Y}] = \bigcup_{n\geq 0} \mathcal{X}^n[\mathcal{Y}] = \mathcal{X}^0[\mathcal{Y}] + \mathcal{X}^1[\mathcal{Y}] + \mathcal{X}^2[\mathcal{Y}] + \dots$$

το οποίο είναι και το ελάχιστο σταθερό σημείο της εξίσωσης :

$$\mathcal{X}^*[\mathcal{Y}] = \mathcal{Y} + \mathcal{X}(\mathcal{X}^*[\mathcal{Y}])$$

Μία πρόταση που βοηθά να αποδείξουμε ότι κάποια σύνολα περιέχουν μη κανονικοποιήσιμους όρους είναι η εξής :

**13.2.2 Πρόταση.** *Για τυχόντα σύνολα S-όρων $\mathcal{X}$, $\mathcal{Y}$ ισχύει :*

$$(S\mathcal{M})^*[\mathcal{X}]\mathcal{Y} \xrightarrow{\circledast} \mathcal{X}\mathcal{Y},$$

*οπότε και :*

$$\text{αν } \mathcal{X}\mathcal{Y}\uparrow, \text{ τότε } (S\mathcal{M})^*[\mathcal{X}]\mathcal{Y}\uparrow.$$

*Απόδειξη (με επαγωγή).* Η βάση της επαγωγής είναι $(S\mathcal{M})^0[\mathcal{X}]\mathcal{Y} = \mathcal{X}\mathcal{Y}$. Έστω ότι ισχύει η επαγωγική υπόθεση : $(S\mathcal{M})^k[\mathcal{X}]\mathcal{Y} \xrightarrow{\circledast} \mathcal{X}\mathcal{Y}$, τότε :

$$(S\mathcal{M})^{k+1}[\mathcal{X}]\mathcal{Y} = S\mathcal{M}((S\mathcal{M})^k[\mathcal{X}])\mathcal{Y} \xrightarrow{\circ} (S\mathcal{M})^k[\mathcal{X}]\mathcal{Y} \xrightarrow{\circledast} \mathcal{X}\mathcal{Y}$$

(το τελευταίο από την επαγωγική υπόθεση). □

Παρακάτω δίνουμε ένα σύνολο με την βοήθεια του οποίου θα κατασκευάσουμε σύνολα μη κανονικοποιήσιμων όρων.

**13.2.3 Ορισμός.** $\mathcal{E} \overset{\text{ορ.}}{=} (SS)^*[\mathcal{Q}_2\mathcal{Q}_1]$.

*13.2.4 Παρατήρηση.* [110] $\mathcal{Q}_1\mathcal{E} \subseteq \mathcal{E}$.

Διατυπώνουμε δύο προτάσεις, χωρίς απόδειξη, που αφορούν σύνολα μη κανονικοποιήσιμων όρων. Για τις αποδείξεις, βλέπε [110].

**13.2.5 Πρόταση.** $\mathcal{E}\mathcal{E}\uparrow_{\oplus}$.

**13.2.6 Πρόταση.** $\mathcal{Q}_3\mathcal{Q}_2\mathcal{Q}_1\uparrow$.

Τελικά, άμεσα, από τα παραπάνω, προκύπτει :

**13.2.7 Πόρισμα.** $(\mathcal{Q}_3\mathcal{Q}_2\mathcal{Q}_1 + \mathcal{E}\mathcal{E})\uparrow_{\oplus}$.

Το παραπάνω αποτέλεσμα είναι το βασικό εργαλείο για να αποδεικνύουμε την μη κανονικοποίηση $S$-όρων [110].

## 13.3 Κατηγοριοποίηση

Για να δείξουμε το θεώρημα για την αποκρισιμότητα της κανονικοποίησης στην επόμενη ενότητα, αρκεί να περιοριστούμε σε όρους της μορφής $\mathcal{NN}$, επειδή για κάθε $S$-όρο $xy$, μπορούμε να εφαρμόσουμε αναδρομικά τον αλγόριθμο σε κάθε έναν από τους ριζικούς υποόρους $x$, $y$ και αν τουλάχιστον ένας από τους $x$, $y$ δεν κανονικοποιείται, να αποκριθούμε ότι ο $xy$ δεν κανονικοποιείται, αλλιώς να θεωρήσουμε τις κανονικές μορφές των $x$, $y$.

Θα κατηγοριοποιήσουμε όλους τους κανονικοποιήσιμους $S$-όρους σε διάφορα κατάλληλα σύνολα : $\mathcal{H}_0$, $\mathcal{H}_1$, $\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$ και στην επόμενη ενότητα θα δούμε πώς συμπεριφέρεται ως προς την κανονικοποίηση κάθε συνδυασμός ριζικών υποόρων από τα παραπάνω σύνολα.

Αρχικά χωρίζουμε το $\mathcal{N}$ με την βοήθεια των ακολούθων συνόλων :

$$\mathcal{H}_0 \stackrel{\text{ορ.}}{=} (SS+B)^*[S+S\mathcal{N}+SBS+SB(SS)]$$
$$\mathcal{H}_1 \stackrel{\text{ορ.}}{=} (SS+B)^*[\mathcal{Q}_3\mathcal{Q}_2+S\mathcal{Q}_3\mathcal{M}]$$

Εύκολα προκύπτουν τα εξής : $\mathcal{H}_0 \subseteq \mathcal{N}$, ενώ το $\mathcal{H}_1$ περιέχει και μη κανονικούς όρους (αυτό δεν είναι πρόβλημα, αφού θεωρούμε το σύνολο $\mathcal{H}_1 \cap \mathcal{N}$ παρακάτω), και τα $\mathcal{H}_0$, $\mathcal{H}_1$ είναι ξένα μεταξύ τους.

**13.3.1 Πρόταση.** *Τα $\mathcal{H}_0$ και $\mathcal{H}_1$ καλύπτουν το $\mathcal{N}$.*

*Απόδειξη.* Βλέπε [110]. □

Κατόπιν, χωρίζουμε το σύνολο $\mathcal{H}_0$ με την βοήθεια των :

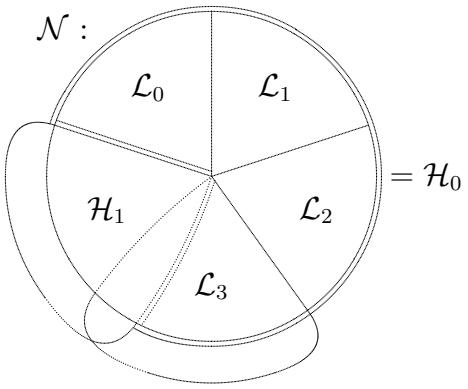
$$\mathcal{L}_0 \stackrel{\text{ορ.}}{=} (SS)^*[S+S\mathcal{N}]$$
$$\mathcal{L}_1 \stackrel{\text{ορ.}}{=} (SS)^*[BS+SBS]$$
$$\mathcal{L}_2 \stackrel{\text{ορ.}}{=} (SS)^*[B(SS)+BB]$$
$$\mathcal{L}_3 \stackrel{\text{ορ.}}{=} (SS)^*[SB(SS)+B\mathcal{Q}_3]$$

Εύκολα προκύπτουν τα εξής : $\mathcal{L}_{012} \stackrel{\text{ορ.}}{=} \mathcal{L}_0+\mathcal{L}_1+\mathcal{L}_2 \subseteq \mathcal{H}_0$, ενώ το $\mathcal{L}_3$ περιέχει και μη κανονικούς όρους και όρους εκτός του $\mathcal{H}_0$ (αλλά αυτό δεν αποτελεί πρόβλημα), και τα $\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$ είναι ξένα μεταξύ τους.

**13.3.2 Πρόταση.** *Τα $\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$ καλύπτουν το $\mathcal{H}_0$.*

*Απόδειξη.* Βλέπε [110]. □

Από τις προτάσεις 13.3.1 και 13.3.2, διαμερίζουμε το $\mathcal{N}$ όπως φαίνεται στο σχήμα 13.1 (ο κύκλος παριστάνει το $\mathcal{N}$, η διπλή γραμμή περικλείει το




Σχήμα 13.1 – Κατηγοριοποίηση κανονικοποιήσιμων $S$-όρων

$\mathcal{H}_0$· παρατηρήσατε ότι τα $\mathcal{H}_1$ και $\mathcal{L}_3$ έχουν κοινά στοιχεία τόσο εντός όσο και εκτός του $\mathcal{N}$, αλλά αυτό δεν είναι πρόβλημα).

Έχουμε λοιπόν $\mathcal{H}_0 \subseteq \mathcal{N} \subseteq \mathcal{H}_{01}$ και $\mathcal{L}_{012} \subseteq \mathcal{H}_0 \subseteq \mathcal{L}_{0123}$.

Πρέπει να σημειώσουμε ότι αυτή η κατηγοριοποίηση έχει ομοιότητες με την αντίστοιχη του Waldmann [104, 105].

## 13.4 Αποκρισιμότητα κανονικοποίησης

**13.4.1 Θεώρημα.** *Υπάρχει αλγόριθμος που αποκρίνεται αν ένας δεδομένος $S$-όρος έχει κανονική μορφή.*

Η απόδειξη του παραπάνω θεωρήματος βασίζεται στα παρακάτω εννέα μέρη (έχουμε τους επιπλέον συμβολισμούς : $\mathcal{L}_{23} \overset{\text{ορ.}}{=} \mathcal{L}_2 + \mathcal{L}_3$ και $\mathcal{L}_{123} \overset{\text{ορ.}}{=} \mathcal{L}_1 + \mathcal{L}_2 + \mathcal{L}_3$). Για τις λεπτομερείς αποδείξεις τους (οι οποίες, όπως είπαμε, βασίζονται σημαντικά στο πόρισμα 13.2.7) παραπέμπουμε στο [110].

**Μέρος 1.** $\mathcal{L}_0\mathcal{N}\downarrow$

**Μέρος 2.** $\mathcal{H}_0\mathcal{L}_0\downarrow$

**Μέρος 3.** $\mathcal{L}_1\mathcal{H}_0\downarrow$

**Μέρος 4.** $\mathcal{L}_2\mathcal{L}_1\downarrow$

**Μέρος 5.** $\mathcal{L}_{23}\mathcal{L}_{23}\uparrow$

**Μέρος 6.** $\mathcal{L}_3\mathcal{L}_1\uparrow$

**Μέρος 7.** $\mathcal{H}_1\mathcal{Q}_2\uparrow$

**Μέρος 8.** $\mathcal{L}_{123}\mathcal{H}_1\uparrow$

Τα παραπάνω φαίνονται στον ακόλουθο πίνακα :

| | $\mathcal{L}_0$ | $\mathcal{L}_1$ | $\mathcal{L}_2$ | $\mathcal{L}_3$ | $\mathcal{H}_1$ |
|---|---|---|---|---|---|
| $\mathcal{L}_0$ | $\downarrow_1$ | $\downarrow_1$ | $\downarrow_1$ | $\downarrow_1$ | $\downarrow_1$ |
| $\mathcal{L}_1$ | $\downarrow_2$ | $\downarrow_3$ | $\downarrow_3$ | $\downarrow_3$ | $\uparrow_8$ |
| $\mathcal{L}_2$ | $\downarrow_2$ | $\downarrow_4$ | $\uparrow_5$ | $\uparrow_5$ | $\uparrow_8$ |
| $\mathcal{L}_3$ | $\downarrow_2$ | $\uparrow_6$ | $\uparrow_5$ | $\uparrow_5$ | $\uparrow_8$ |

και $\quad \mathcal{H}_1\mathcal{Q}_2\uparrow_7$

όπου στην πρώτη στήλη είναι οι αριστεροί ριζικοί υποόροι και στην πρώτη γραμμή οι δεξιοί ριζικοί υποόροι. Οι δείκτες στα βέλη δεικνύουν το αντίστοιχο μέρος.

Εύκολα μπορεί να δει κανείς ότι τα παραπάνω οκτώ μέρη καλύπτουν όλες τις περιπτώσεις όρων στο $\mathcal{NN}$, εκτός από το ακόλουθο σύνολο όρων :

$$(\mathcal{H}_1 \cap \mathcal{N})(S + SS).$$

Για αυτό χρειαζόμαστε το ένατο (και τελευταίο) μέρος :

**Μέρος 9.** *Αν* $(\mathcal{H}_1 \cap \mathcal{N})(S + SS)\uparrow$ *ή όχι, ανάγεται στα μέρη 1 έως 8.*

*Ιδέα της απόδειξης.* Ανάγουμε την κανονικοποίηση ή όχι όρου στο $\mathcal{N}(SS)$ στα μέρη 1 έως 8. Κατόπιν ανάγουμε την κανονικοποίηση όρου στο $\mathcal{N}S$ στην κανονικοποίηση ή όχι του $\mathcal{N}(SS)$ και τα μέρη 1 έως 8. □

## 13.5 Αλγόριθμος κανονικοποίησης

Από την προηγούμενη ενότητα, προκύπτει ο ακόλουθος αλγόριθμος (διατυπωμένος σε ψευδογλώσσα) που αποκρίνεται περί κανονικοποίησης και σε θετική περίπτωση δίνει την κανονική μορφή του όρου εισόδου :

**function** normal($X$ : $S$-term) **returning** "no" or $S - term$ in NF ;
**begin**
  **if** $X = S$ **then return** $X$ **else**
  **begin** find_root_subterms($X$) **returning** $Y, Z$ ;
    $Y_1 := normal(Y)$; $Z_1 := normal(Z)$ ;
    **if** ($Y_1$="no") **or** ($Z_1$="no") **then return** "no" **else**
    **begin** $X := (Y_1 Z_1)$ ;
      **if** ($Y_1 \in \mathcal{L}_0$) **or** ($Y_1 \in \mathcal{H}_0$ **and** $Z_1 \in \mathcal{L}_0$) **or**
        ($Y_1 \in \mathcal{L}_1$ **and** $Z_1 \in \mathcal{H}_0$) **or** ($Y_1 \in \mathcal{L}_2$ **and** $Z_1 \in \mathcal{L}_1$) **then**
      **begin** $W$ :=reduce_repeatedly($X$) ; **return** $W$ **end**
      **else if** ($Z_1 \neq S$) **and** ($Z_1 \neq SS$) **then return** "no"
        **else** (* Comment : $Z_1 = S$ or $SS$ **and** $Y_1 = SN_1N_2$ *)
        **begin** $Y_2 := normal(N_1Z_1)$; $Z_2 := normal(N_2Z_1)$ ;
          **if** ($Y_2$="no") **or** ($Z_2$="no") **then return** "no"
          **else return** $normal(Y_2Z_2)$
        **end end end end**

## 13.6 Περιγραφή με γραμματική· αλγόριθμος

Με βάση την ενότητα 2.2, σελίδα 14, και ιδιαίτερα τον ορισμό 2.2.2, ορίζουμε την έννοια των προκατόχων ενός συνόλου $S$-όρων :

**13.6.1 Ορισμός.** $\mathrm{pred}(\mathcal{X}) \stackrel{\text{ορ.}}{=} \{y \mid y \xrightarrow{*} x \wedge x \in \mathcal{X}\}$.

Επίσης, ορίζουμε την σχετική έννοια των αριστερών ριζικών υποόρων των προκατόχων ενός συνόλου $S$-όρων, με σταθερό δεξιό ριζικό υποόρο $z$ :

**13.6.2 Ορισμός.** $\mathrm{pred}_z(\mathcal{X}) \stackrel{\text{ορ.}}{=} \{y \mid yz \xrightarrow{*} x \wedge x \in \mathcal{X}\}$.

*13.6.3 Παρατήρηση.* Όπως μπορεί κανείς να προβλέψει και από το μέρος 9 του θεωρήματος 13.4.1, εισάγουμε την έννοια $\mathrm{pred}_z$, για να θεωρήσουμε $z = S, SS$.

Προφανώς, $\mathrm{pred}(\mathcal{N})$ είναι το σύνολο των κανονικοποιήσιμων όρων. Παρακάτω δίνουμε, χωρίς απόδειξη, όπως και στο [111], μία αρχική απόπειρα για μία γραμματική χωρίς συμφραζόμενα[1] που παραγάγει το σύνολο $\mathrm{pred}(\mathcal{N})$.

$$
\begin{aligned}
\mathcal{N}_\infty &\to S + SS\mathcal{N}_\infty + \mathcal{N}_\infty S \\
\mathrm{pred}(\mathcal{L}_0) &\to \mathcal{L}_0^a + \mathcal{N}_\infty, \qquad \mathcal{L}_0^a \to SS\mathcal{L}_0^a + S\,\mathrm{pred}(\mathcal{N}) \\
\mathrm{pred}(\mathcal{L}_1) &\equiv \mathcal{L}_1^{\mathrm{p}}, \qquad \mathcal{L}_1^{\mathrm{p}} \to SS\mathcal{L}_1^{\mathrm{p}} + \mathcal{L}_1^a S, \qquad \mathcal{L}_1^a \to B + SB + SS\mathcal{L}_1^a \\
\mathrm{pred}(\mathcal{L}_2) &\equiv \mathcal{L}_2^{\mathrm{p}}, \qquad \mathcal{L}_2^{\mathrm{p}} \to BB + A(SS) + B(SS) + SS\mathcal{L}_2^{\mathrm{p}} + \mathcal{L}_2^a S \\
&\qquad \mathcal{L}_2^a \to SS\mathcal{L}_2^a + \mathcal{L}_2^b S, \qquad \mathcal{L}_2^b \to B + SS\mathcal{L}_2^b \\
\mathrm{pred}(\mathcal{H}_0) &\equiv \mathcal{H}_0^{\mathrm{p}}, \\
&\qquad \mathcal{H}_0^{\mathrm{p}} \to S + S\,\mathrm{pred}(\mathcal{N}) + SBS + SB(SS) + (SS + B)\mathcal{H}_0^{\mathrm{p}} \\
&\qquad\qquad + \mathcal{H}_0^a(SS) + \mathcal{H}_0^b S \\
&\qquad \mathcal{H}_0^a \to \mathcal{N}_\infty + \mathcal{L}_1^b, \qquad \mathcal{H}_0^b \to \mathcal{H}_0^a + \mathcal{H}_0^c, \\
&\qquad \mathcal{H}_0^c \to SS\mathcal{H}_0^c + \mathcal{H}_0^d S, \quad \mathcal{H}_0^d \to SS\mathcal{H}_0^d + S\mathcal{N}_\infty
\end{aligned}
$$

και τελικά :

$$
\begin{aligned}
\mathrm{pred}(\mathcal{N}) \to\ & \mathrm{pred}(\mathcal{L}_0)\,\mathrm{pred}(\mathcal{N}) + \mathrm{pred}(\mathcal{L}_1)\,\mathrm{pred}(\mathcal{H}_0) + \mathrm{pred}(\mathcal{L}_2)\,\mathrm{pred}(\mathcal{L}_1) \\
& + \mathrm{pred}(\mathcal{H}_0)\,\mathrm{pred}(\mathcal{L}_0) + \mathrm{pred}_S(\mathcal{N})\,S + \mathrm{pred}_{SS}(\mathcal{N})\,(SS)
\end{aligned}
$$

1. Η γραμματική δίνεται με τους όρους σε ενθεματική αναπαράσταση, για διευκόλυνση στην ανάγνωση, αλλά για να είμαστε απόλυτα τυπικοί και να αποφύγουμε τις παρενθέσεις, θεωρούμε ότι οι όροι θα πρέπει να είναι σε προθεματική αναπαράσταση.

Δεν παρέχουμε κανόνες για τις περιπτώσεις $\mathrm{pred}_S(\mathcal{N})$ και $\mathrm{pred}_{SS}(\mathcal{N})$ που αντιστοιχούν στο μέρος 9 του θεωρήματος 13.4.1. Αυτοί οι κανόνες είναι περίπλοκοι και δεν μπορούν να επαληθευθούν ευκόλως. Μία άλλη γραμματική δένδρων που περιέχει και κανόνες για τα $\mathrm{pred}_S(\mathcal{N})$ και $\mathrm{pred}_{SS}(\mathcal{N})$ υπάρχει στο [105] και αποτελεί την βάση για τον τεχνολογητή που δίνουμε στο παράρτημα Γ.

Ακόμη όμως και με την βοήθεια των παραπάνω κανόνων, μπορούμε να βελτιώσουμε τον αλγόριθμο της ενότητας 13.5, ο οποίος απαιτεί πάντοτε κανονικοποίηση των ριζικών υποόρων, ως εξής : Χρησιμοποιούμε την παραπάνω γραμματική για να τεχνολογήσουμε τους ριζικούς υποόρους και να αποκριθούμε περί της κανονικοποίησης χωρίς αναγωγές. Μόνο στις περιπτώσεις που το μέρος 9 του θεωρήματος 13.4.1 πρέπει να εφαρμοστεί, χρησιμοποιούνται αναγωγές, όπως στον αρχικό αλγόριθμο. Ο νέος αλγόριθμος που προκύπτει είναι ένας ενδιάμεσος αλγόριθμος ανάμεσα στον αρχικό αλγόριθμο και έναν πλήρη τεχνολογητή, που συνδυάζει αμφότερες τις τεχνικές τους :

**function** normal($X$ : $S$-term) **returning** "no" or $S-term$ in NF ;
**begin**
  **if** $X = S$ **then return** $X$ **else**
  **begin** find_root_subterms($X$) **returning** $Y, Z$ ;
    **if** ($Y \in \mathrm{pred}(\mathcal{L}_0)$) **or** ($Y \in \mathrm{pred}(\mathcal{H}_0)$ **and** $Z \in \mathrm{pred}(\mathcal{L}_0)$) **or**
     ($Y \in \mathrm{pred}(\mathcal{L}_1)$ **and** $Z \in \mathrm{pred}(\mathcal{H}_0)$) **or**
     ($Y \in \mathrm{pred}(\mathcal{L}_2)$ **and** $Z \in \mathrm{pred}(\mathcal{L}_1)$) **then**
    **begin** $W$ :=reduce_repeatedly($X$) ; **return** $W$ **end**
    **else**
      **if** ($Z \neq S$) **and** ($Z \neq SS$) **then return** "no"
      **else**
      **begin** $Y_1 := normal(Y)$; (* $Y_1 = SN_1N_2$ *)
        $Y_2 := normal(N_1Z_1)$; $Z_2 := normal(N_2Z_1)$ ;
        **if** ($Y_2$="no") **or** ($Z_2$="no") **then return** "no"
        **else return** $normal(Y_2Z_2)$
      **end end end**

## 13.7 Στατιστικά κανονικοποιήσιμων όρων

Στον πίνακα 13.1 (από το [111]) φαίνεται το πλήθος των μη κανονικοποιήσιμων $S$-όρων μήκους μέχρι 18, ανά αρχικό μήκος. Δίνεται επίσης, για συγκεκριμένο αρχικό μήκος, προσεγγιστικά, ο λόγος των μη κανονικοποιήσιμων όρων προς το συνολικό πλήθος όρων τέτοιου αρχικού μήκους.

Πίνακας 13.1 – Πλήθος μη κανονικοποιήσιμων $S$-όρων

| μήκος | συνολικά | χωρίς NF | λόγος |
|---|---|---|---|
| 6 | 42 | 0 | 0 |
| 7 | 132 | 2 | 0.015 |
| 8 | 429 | 41 | 0.09 |
| 9 | 1 430 | 276 | 0.19 |
| 10 | 4 862 | 1 481 | 0.30 |
| 11 | 16 796 | 6 829 | 0.40 |
| 12 | 58 786 | 29 288 | 0.49 |
| 13 | 208 012 | 119 946 | 0.57 |
| 14 | 742 900 | 477 885 | 0.64 |
| 15 | 2 674 440 | 1 870 502 | 0.69 |
| 16 | 9 694 845 | 7 238 607 | 0.74 |
| 17 | 35 357 670 | 27 805 037 | 0.78 |
| 18 | 129 644 790 | 106 291 141 | 0.81 |

## 13.8 Μεγάλες κανονικές μορφές

Στον πίνακα 13.2 δίνουμε τον μέγιστο αριθμό βημάτων αναγωγής, χρησιμοποιώντας την στρατηγική του τρίτου ορίσματος, που απαιτούνται για να φτάσουμε σε κανονική μορφή, αρχίζοντας από κανονικοποιήσιμους όρους του ιδίου μήκους, για μήκη από 7 έως και 15. Είναι ενδιαφέρον ότι, για κάθε μήκος, μόνον ένας όρος αντιστοιχεί στον μέγιστο αριθμό βημάτων. Οι εν λόγω αρχικοί όροι επίσης παρουσιάζονται στον πίνακα 13.2.

Πίνακας 13.2 – Μέγιστο πλήθος αναγωγών

| αρχ. μήκος | αναγωγές | αρχικός όρος | μήκος NF |
|---|---|---|---|
| 7 | 10 | $SBSSS$ | 40 |
| 8 | 11 | $SASSSS$ | 25 |
| 9 | 22 | $B(SS)(SS)SS$ | 169 |
| 10 | 44 | $SA(SS)SSSS$ | 1561 |
| 11 | 44 | $S(SA(SS)SSSS)$ | 1562 |
| 12 | 48 | $SS(SA(SS))SSSS$ | 1563 |
| 13 | 55 | $SA(ASSS)SSS$ | 3066 |
| 14 | 68 | $SA(ASSSS)SSS$ | 18661 |
| 15 | 83 | $SA(ASSSSS)SSS$ | 38815 |

Στον πίνακα 13.3 δίνουμε το μέγιστο μήκος των κανονικών μορφών, για κάθε μήκος αρχικών όρων από 7 έως και 15. Σημειώσατε ότι, αντίθετα με τον μέγιστο αριθμό βημάτων αναγωγής, εν γένει υπάρχουν πολλοί αρχικοί όροι ιδίου μήκους που καταλήγουν σε κανονικές μορφές του μεγίστου μήκους (για παράδειγμα, για μήκος αρχικού όρου ίσο με 10 έχουμε τρεις τέτοιους όρους). Επίσης, για το ίδιο μήκος, ο αρχικός όρος με μέγιστο αριθμό βημάτων αναγωγής μέχρι κανονικής μορφής δεν βρίσκεται εν γένει μεταξύ των αρχικών όρων με τις μεγαλύτερες κανονικές μορφές (για παράδειγμα ο όρος $SA(ASSS)SSS$ για μήκος 13).

Πίνακας 13.3 – Μέγιστο μήκος κανονικών μορφών

| αρχ. μήκος | μεγ. μήκος NF |
|---:|---:|
| 7 | 41 |
| 8 | 42 |
| 9 | 169 |
| 10 | 1561 |
| 11 | 1562 |
| 12 | 1563 |
| 13 | 7811 |
| 14 | 18661 |
| 15 | 111973 |

Η μελέτη των μεγάλων κανονικών μορφών που προκύπτουν από το σύνολο των αρχικών όρων ιδίου μήκους, μεγαλύτερου του 15, ούτως ώστε να συμπληρωθούν περαιτέρω οι πίνακες 13.2 και 13.3, απαιτεί μεγαλύτερη υπολογιστική δυνατότητα από αυτήν που διαθέτουμε.

Ο Waldmann [104] αναφέρεται στους αρχικούς όρους με μεγάλες κανονικές μορφές, σε αρκετά βήματα αναγωγής, ως *τέρατα* (monsters). Αποδεικνύει ότι ο όρος $SA(SS)^n[SS]SSS$ έχει κανονική μορφή μεγέθους $n^{\Theta(n)}$, στην οποία καταλήγουμε μετά από $\Theta(n^2)$ βήματα αναγωγής.

# Κεφάλαιο 14

# Άπειρες αλυσίδες αναγωγών

Σε αυτό το κεφάλαιο δίνουμε μερικές αποδείξεις μη κανονικοποιησιμότητας με την μέθοδο Ζάχου (βλέπε ενότητα 12.4, στην σελίδα 137, καθώς και [108]). Στην ενότητα 14.1 εφαρμόζουμε την μέθοδο για τον όρο $A(SA)S$. Αυτή η περίπτωση δεν μας δυσκόλεψε ιδιαίτερα αν εξαιρέσει κανείς το επιπλέον βοηθητικό λήμμα Ε. Μία περίπτωση που μας δυσκόλεψε ήταν αυτή που περιγράφεται στην ενότητα 14.2, για τον όρο $B(SS)(B(BS))$. Μάλιστα, δίνουμε δύο αποδείξεις : πρώτα αυτήν που βρήκαμε αρχικά και δεν είναι απολύτως σύμφωνη με την μορφή των λημμάτων Α και Β της μεθόδου Ζάχου και μετά μία μετατροπή της προηγούμενης που είναι σύμφωνη. Τέλος, δίνεται και μία απόδειξη μη κανονικοποιησιμότητας για την ίδια μέθοδο, όχι για κάποιον όρο, αλλά για οποιονδήποτε όρο από ένα σύνολο όρων, το $\mathcal{L}_2\mathcal{L}_2$, στην ενότητα 14.3.

## 14.1 Η περίπτωση $A(SA)S$

Ορίζουμε :

$$D = S(SA)S \qquad D' = SS(BD)$$
$$D_0 = SD \qquad D'_n = S(SD')(SD_n)$$
$$D_{n+1} = SD_n(SD'_n)$$

**Λήμμα 1.** $B(SS)(B(BS)) \xrightarrow{\circledast} DD_0$

*Απόδειξη.*

$B(SS)(B(BS)) \xrightarrow{②} B(BS)(SS(B(BS))) \xrightarrow{②}$
$SS(B(BS))(BS(SS(B(BS)))) \equiv SS(B(BS))(BSD') \xrightarrow{○}$
$B(BS)(BSD') \xrightarrow{○} BS(BSD') \equiv DD_0$ □

**Λήμμα 2.** $DD_n \xrightarrow{\circledast} DD_{n+1}$

Θα δείξουμε το παραπάνω (λήμμα 2) με την βοήθεια των παρακάτω επιπλέον λημμάτων A, B, Δ και E :

**Λήμμα A.** $D_nM \xrightarrow{\circledast} D_0M$

*Απόδειξη.* $D_nM \equiv SD_{n-1}(SD'_{n-1})M \xrightarrow{\bigcirc} D_{n-1}M \xrightarrow{\circledast} D_0M$ □

**Λήμμα B.** $D_0M \xrightarrow{\circledast} DM$

Στην απόδειξη του παραπάνω θα μας βοηθήσουν τα παρακάτω λήμματα.

**Λήμμα Γ.** $D_0M \xrightarrow{\circledast} D'M(S(SD')M)$

*Απόδειξη.*

$D_0M \equiv BSD'M \xrightarrow{2} S(SD')(D'(SD'))M \xrightarrow{\bigcirc} D'(SD')M \equiv$
$SS(B(BS))(SD')M \longrightarrow S(SD')(B(BS)(SD'))M \xrightarrow{\bigcirc} B(BS)(SD')M \xrightarrow{2}$
$S(BS(SD'))(SD'(BS(SD')))M \xrightarrow{\bigcirc} BS(SD')M \xrightarrow{2}$
$S(S(SD'))(SD'(S(SD')))M \xrightarrow{\bigcirc} SD'(S(SD'))M \longrightarrow D'M(S(SD')M)$ □

**Λήμμα Δ.** $D'M \xrightarrow{\circledast} DM$

*Απόδειξη.* $D'M \equiv SS(BD)M \xrightarrow{\bigcirc} BDM \xrightarrow{\bigcirc} DM$ □

*Απόδειξη του λήμματος B.*

$D_0M \xrightarrow[\Gamma]{\circledast} D'M \xrightarrow[\Delta]{\circledast} DM$ □

Αποδεικνύουμε ακόμη ένα βοηθητικό λήμμα :

**Λήμμα E.** $D_0(SM) \xrightarrow{\circledast} MM'$, *όπου* $M' = S(SD')(SM)$

*Απόδειξη.*

$D_0(SM) \xrightarrow[\Gamma]{\circledast} D'(SM)(S(SD')(SM)) \equiv D'(SM)M' \equiv BS(SM)M' \xrightarrow{2}$
$S(S(SM))(SM(S(SM)))M' \xrightarrow{\bigcirc} SM(S(SM))M' \xrightarrow{\bigcirc} MM'$ □

*Απόδειξη του λήμματος 2.*

$DD_n \equiv BSD_n \xrightarrow{②} D_n(SD_n) \xrightarrow[\mathrm{A}]{\circledast} D_0(SD_n) \xrightarrow[\mathrm{E}]{\circledast} D_nD'_n \xrightarrow[\mathrm{A}]{\circledast} D_0D'_n \xrightarrow[\mathrm{B}]{\circledast}$
$DD'_n \equiv BSD'_n \xrightarrow{②} D'_n(SD'_n) \equiv S(SD')(SD_n)(SD'_n) \longrightarrow$
$SD'(SD'_n)(SD_n(SD'_n)) \equiv SD'(SD'_n)D_{n+1} \xrightarrow{\bigcirc} D'D_{n+1} \xrightarrow[\Delta]{\circledast} DD_{n+1}$ □

## 14.2 Η περίπτωση $B(SS)(B(BS))$

Ορίζουμε :

$$D' = SS(B(BS)) \qquad D = BSD'$$
$$D_0 = SD \qquad D'_n = S(SD')D_n$$
$$D_{n+1} = S(D_n(SD'_n))$$

Όπως συνήθως, αρκεί να δείξουμε τα παρακάτω δύο λήμματα :[1]

**Λήμμα 1.** $B(SS)(B(BS)) \xrightarrow{\circledast} DD_0$

*Απόδειξη.*

$B(SS)(B(BS)) \xrightarrow{②} B(BS)(SS(B(BS))) \xrightarrow{②}$
$SS(B(BS))(BS(SS(B(BS)))) \equiv SS(B(BS))(BSD') \xrightarrow{○}$
$B(BS)(BSD') \xrightarrow{○} BS(BSD') \xrightarrow{②} BSD'(S(BSD')) \equiv D(SD) \equiv DD_0$ □

**Λήμμα 2.** $DD_n \xrightarrow{\circledast} DD_{n+1}$

Θα δείξουμε το παραπάνω με την βοήθεια των παρακάτω επιπλέον λημμάτων A′ και B′ :

**Λήμμα A′.** $D_nXM \xrightarrow{\circledast} DM$

*Απόδειξη.* Θα το δείξουμε επαγωγικά.
Για $n = 0$ έχουμε $D_0XM \equiv SDXM \xrightarrow{○} DM$.
Για $n \geq 0$ έχουμε $D_{n+1}XM \equiv S(D_n(SD'_n))XM \xrightarrow{○} D_n(SD'_n)M$, δηλαδή της μορφής $D_nXM$. □

**Λήμμα B′.** $DM \xrightarrow{\circledast} M(SM)M'$, *όπου* $M' = S(SD')M$.[2]

*Απόδειξη.*

$DM \equiv BSD'M \xrightarrow{2} S(SD')(D'(SD'))M \xrightarrow{○} D'(SD')M \equiv$
$SS(B(BS))(SD')M \longrightarrow S(SD')(B(BS)(SD'))M \xrightarrow{○} B(BS)(SD')M \xrightarrow{2}$
$S(BS(SD'))(SD'(BS(SD')))M \xrightarrow{○} BS(SD')M \xrightarrow{2}$
$S(S(SD'))(SD'(S(SD')))M \xrightarrow{○} SD'(S(SD'))M \longrightarrow D'M(S(SD')M) \equiv$
$D'MM' \equiv SS(B(BS))MM' \longrightarrow SM(B(BS)M)M' \xrightarrow{○} B(BS)MM' \xrightarrow{2}$
$S(BSM)(M(BSM))M' \xrightarrow{○} BSMM' \xrightarrow{2} S(SM)(M(SM))M' \xrightarrow{○}$
$M(SM)M'$ □

1. Στις αποδείξεις χρησιμοποιούμε συχνά τον κανόνα δύο βημάτων (αναγωγών) για το $B$ : $Bxy \xrightarrow{2} S(xy)(y(xy))$, ενώ συνήθως διατηρούμε τον υποόρο $y(xy)$ του παραπάνω, συμβολικά : $Bxy \xrightarrow{②} y(xy)$.

2. Παρατηρήσατε ότι ο ορισμός του $M'$ συναρτήσει του $M$ ταυτίζεται με τον ορισμό του $D'_n$ συναρτήσει του $D_n$. Αυτό θα φανεί χρήσιμο παρακάτω.

*Απόδειξη του λήμματος 2.*

$DD_n \xrightarrow[\text{B}]{\circledast} D_n(SD_n)D'_n \xrightarrow[\text{A}]{\circledast} DD'_n \xrightarrow[\text{B}]{\circledast} D'_n(SD'_n) \equiv S(SD')D_n(SD'_n) \longrightarrow$
$SD'(SD'_n)(D_n(SD'_n)) \xrightarrow{\bigcirc} D'(D_n(SD'_n)) \equiv SS(B(BS))(D_n(SD'_n)) \xrightarrow{\bigcirc}$
$B(BS)(D_n(SD'_n)) \xrightarrow{\bigcirc} BS(D_n(SD'_n)) \xrightarrow{②} D_n(SD'_n)(S(D_n(SD'_n))) \equiv$
$D_n(SD'_n)D_{n+1} \xrightarrow[\text{A}]{\circledast} DD_{n+1}$ □

Αυτή είναι η πρώτη απόδειξη μη κανονικοποίησης που βρήκαμε για τον όρο $B(SS)(B(BS))$. Το πρόβλημα είναι ότι τα λήμματα A′ και B′ δεν είναι στην μορφή που απαιτείται από την μέθοδο Ζάχου όπως διατυπώνεται στην ενότητα 12.4, σελίδα 137. Μία άλλη απόδειξη ακολουθεί, η οποία δεν έχει αυτό το πρόβλημα. Με βάση τους παραπάνω ορισμούς, ορίζουμε επιπλέον :

$$\text{νέο}\_D = BS \quad \text{και} \quad \text{νέο}\_D_n = D_n(SD'_n)$$

Τελικά, η απόδειξη απλοποιείται στην :[3]

$$\begin{aligned} D &= BS & D' &= SS(BD) \\ D_0 &= DD' & D'_n &= S(SD')(SD_n) \\ D_{n+1} &= SD_n(SD'_n) \end{aligned}$$

(Παρατήρηση : Ισχύει $\text{νέο}\_D_n = \text{παλαιό}\_D_{n+1}$)

**Λήμμα 1.** $B(SS)(B(BS)) \xrightarrow{\circledast} DD_0$

*Απόδειξη.*

$B(SS)(B(BS)) \xrightarrow{②} B(BS)(SS(B(BS))) \xrightarrow{②}$
$SS(B(BS))(BS(SS(B(BS)))) \equiv SS(B(BS))(BSD') \xrightarrow{\bigcirc}$
$B(BS)(BSD') \xrightarrow{\bigcirc} BS(BSD') \equiv DD_0$ □

**Λήμμα 2.** $DD_n \xrightarrow{\circledast} DD_{n+1}$

Θα δείξουμε το παραπάνω με την βοήθεια των παρακάτω επιπλέον λημμάτων A, B, Δ και E :

**Λήμμα A.** $D_nM \xrightarrow{\circledast} D_0M$

*Απόδειξη.* $D_nM \equiv SD_{n-1}(SD'_{n-1})M \xrightarrow{\bigcirc} D_{n-1}M \xrightarrow{\circledast} D_0M$ □

**Λήμμα B.** $D_0M \xrightarrow{\circledast} DM$

Στην απόδειξη του παραπάνω θα μας βοηθήσουν τα παρακάτω λήμματα.

3. Προφανώς τώρα ορίζουμε διαφορετικά τα $D$, $D_n$ κ.τ.λ.

**Λήμμα Γ.** $D_0M \xrightarrow{\circledast} D'M(S(SD')M)$

*Απόδειξη.*

$D_0M \equiv BSD'M \xrightarrow{^2} S(SD')(D'(SD'))M \xrightarrow{\bigcirc} D'(SD')M \equiv$
$SS(B(BS))(SD')M \longrightarrow S(SD')(B(BS)(SD'))M \xrightarrow{\bigcirc} B(BS)(SD')M \xrightarrow{^2}$
$S(BS(SD'))(SD'(BS(SD')))M \xrightarrow{\bigcirc} BS(SD')M \xrightarrow{^2}$
$S(S(SD'))(SD'(S(SD')))M \xrightarrow{\bigcirc} SD'(S(SD'))M \longrightarrow D'M(S(SD')M)$ □

**Λήμμα Δ.** $D'M \xrightarrow{\circledast} DM$

*Απόδειξη.* $D'M \equiv SS(BD)M \xrightarrow{\bigcirc} BDM \xrightarrow{\bigcirc} DM$ □

*Απόδειξη του λήμματος Β.*

$D_0M \xrightarrow[\Gamma]{\circledast} D'M \xrightarrow[\Delta]{\circledast} DM$ □

Αποδεικνύουμε ακόμη ένα βοηθητικό λήμμα :

**Λήμμα Ε.** $D_0(SM) \xrightarrow{\circledast} MM'$, *όπου* $M' = S(SD')(SM)$

*Απόδειξη.*

$D_0(SM) \xrightarrow[\Gamma]{\circledast} D'(SM)(S(SD')(SM)) \equiv D'(SM)M' \equiv BS(SM)M' \xrightarrow{^2}$
$S(S(SM))(SM(S(SM)))M' \xrightarrow{\bigcirc} SM(S(SM))M' \xrightarrow{\bigcirc} MM'$ □

*Απόδειξη του λήμματος 2.*

$DD_n \equiv BSD_n \xrightarrow{②} D_n(SD_n) \xrightarrow[\text{A}]{\circledast} D_0(SD_n) \xrightarrow[\text{E}]{\circledast} D_nD'_n \xrightarrow[\text{A}]{\circledast} D_0D'_n \xrightarrow[\text{B}]{\circledast}$
$DD'_n \equiv BSD'_n \xrightarrow{②} D'_n(SD'_n) \equiv S(SD')(SD_n)(SD'_n) \longrightarrow$
$SD'(SD'_n)(SD_n(SD'_n)) \equiv SD'(SD'_n)D_{n+1} \xrightarrow{\bigcirc} D'D_{n+1} \xrightarrow[\Delta]{\circledast} DD_{n+1}$ □

## 14.3 Σύνολο $S$-όρων : Η περίπτωση $\mathcal{L}_2\mathcal{L}_2$

Η μέθοδος Ζάχου μπορεί να χρησιμοποιηθεί και για την απόδειξη μη κανονικοποιησιμότητας συνόλων $S$-όρων. Παρακάτω, δίνουμε μία τέτοια απόδειξη από το [111] για το σύνολο $\mathcal{L}_2\mathcal{L}_2$ (βλέπε και μέρος 5, του θεωρήματος 13.4.1, σελίδα 145). [4]

Πρέπει πάντως να σημειώσουμε ότι πλέον έχουμε αναγωγές από σύνολα όρων σε σύνολα όρων, στο πνεύμα του ορισμού 13.1.1, σελίδα 141.

4. Υπενθυμίζουμε ότι $\mathcal{L}_2 = (SS)^*[B(SS) + BB]$.

Ορίζουμε :

$$D = B(SS) + BB, \quad D_0 = (SS)^*[D], \quad D_{n+1} = (SS + B)D_n$$

(προφανώς, ισχύει $\mathcal{L}_2 = D_0$), οπότε :

**Λήμμα Α.** $D_nM \xrightarrow{\circledast} D_0M$.

*Απόδειξη.* $D_nM \equiv (SS + B)D_{n-1}M \xrightarrow{\bigcirc} D_{n-1}M \xrightarrow{\circledast} D_0M$ □

**Λήμμα Β.** $D_0M \xrightarrow{\circledast} DM$.

*Απόδειξη.* $D_0M \equiv (SS)^*[D]M \xrightarrow{\circledast} DM$, από την πρόταση 13.2.2, σελίδα 143. □

Κατόπιν :

**Λήμμα 1.** $\mathcal{L}_2\mathcal{L}_2 \xrightarrow{\circledast} DD_0$.

*Απόδειξη.* $\mathcal{L}_2\mathcal{L}_2 \equiv (SS)^*[D](SS)^*[D] \xrightarrow{\circledast} D(SS)^*[D] \equiv DD_0$, πάλι από την πρόταση 13.2.2. □

**Λήμμα 2.** $DD_n \xrightarrow{\circledast} DD_{n+1}$.

*Απόδειξη.* $DD_n \equiv (B(SS) + BB)D_n \xrightarrow{②} D_n((B + SS)D_n) \equiv D_nD_{n+1} \xrightarrow[\mathrm{A}]{\circledast} D_0D_{n+1} \xrightarrow[\mathrm{B}]{\circledast} DD_{n+1}$ □

Από τα λήμματα 1 και 2, προκύπτει άμεσα ότι $\mathcal{L}_2\mathcal{L}_2\uparrow$.

# Μέρος τέταρτο

# Παραρτήματα

# Παράρτημα Α

# Προγράμματα C για $S$-όρους

Σε αυτό το παράρτημα δίνουμε μερικά προγράμματα που επεξεργάζονται ποικιλοτρόπως $S$-όρους.

Τα προγράμματα είναι γραμμένα σε γλώσσα C (για το πρότυπο βλέπε [12]). Η επιλογή αυτή έγινε κυρίως λόγω της ευρείας διάδοσης της γλώσσας. Επιπλέον, οι υπολογισμοί που απαιτούνται για την μελέτη των $S$-όρων είναι ιδιαίτερα απαιτητικοί και για αυτό χρειάζεται μία γλώσσα που δίνει γρήγορο κώδικα.

Αρχικά δίνουμε μερικές βιβλιοθήκες συναρτήσεων στα αρχεία sterm.h, sminnonf.h, schain.h. Αυτές αποτελούν την βάση για όλα τα προγράμματα που ακολουθούν.

## Α.1 Δομές δεδομένων (bintree.h)

Σε αυτήν την βιβλιοθήκη συναρτήσεων ορίζουμε τις δομές δεδομένων που χρησιμοποιούνται από τα προγράμματα. Οι $S$-όροι αναπαρίστανται ως δυαδικά δένδρα :

```
#ifndef BINTREE_H
#define BINTREE_H 1

#include <assert.h>

typedef enum  {
   SCOMB       = 'S',
   APPLY       = 'o',
   NONORMAL    = 'N'
} LabelType;
```

```
typedef struct StructBTreeNode * BTreePtr;

typedef struct StructBTreeNode  {
   LabelType  label;
   BTreePtr   left;
   BTreePtr   right;
} BTreeNode;


BTreePtr   BTreeNodeNew(void);
void       BTreeFree(BTreePtr p);
BTreePtr   BTreeCopy(BTreePtr p);
void       BTreeNodeFree(BTreePtr p);


BTreePtr   BTreeNodeNew(void)
{
   return (BTreePtr) malloc(sizeof(BTreeNode));
}

void       BTreeNodeFree(BTreePtr p)
{
   free(p);
}

void       BTreeFree(BTreePtr p)
{
   if (p == NULL) return;
   if (p->label == APPLY)
   {
      BTreeFree(p->left);
      BTreeFree(p->right);
   }
   BTreeNodeFree(p);
}

BTreePtr   BTreeCopy(BTreePtr p)
{
   BTreePtr pNew;
   if (p == NULL) return NULL;
```

```
/*   if ((pNew = BTreeNodeNew()) == NULL) return NULL; */
   pNew = BTreeNodeNew();
   assert(pNew != NULL);
   pNew->label = p->label;
   if (p->label == APPLY)
   {
      pNew->left  = BTreeCopy(p->left);
      pNew->right = BTreeCopy(p->right);
      if ((pNew->left == NULL) || (pNew->right == NULL))
      {
         BTreeFree(pNew);
         return NULL;
      }
   }
   return pNew;
}

#endif  /* bintree.h */
```

## A.2 Αρχείο sterm.h

Αυτή είναι η βασική βιβλιοθήκη συναρτήσεων που περιέχει συναρτήσεις μετατροπής όρων από την εσωτερική δενδρική παράσταση σε προθεματική, ενθεματική και παρόμοιες μορφές για εκτύπωση στην έξοδο καθώς και συναρτήσεις τεχνολόγησης (parsing) όρων. Επίσης, υπάρχει ο αλγόριθμος που δίνει τον επόμενο όρο στην διάταξη που ορίζεται στην ενότητα 11.6. Στα πλαίσια του $S$-λογισμού υπάρχουν συναρτήσεις που εφαρμόζουν τον κανόνα $Sxyz \rightarrow (xz)yz$, χρησιμοποιώντας διάφορες στρατηγικές. Τέλος, η κυριότερη συνάρτηση είναι αυτή που αφορά τον αλγόριθμο κανονικοποίησης (βλέπε και ενότητα 13.5).

```
#ifndef STERM_H

#define STERM_H 1


#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "bintree.h"
```

```
/*
#define DEBUG
#undef DEBUG
*/

void SprintPrefix(BTreePtr p);
void SprintInfix(BTreePtr p);
void SprintInfixSyn(BTreePtr p);
void SprintInfixSynDn(BTreePtr p, BTreePtr q);

void Snormalize(BTreePtr p);
int  SreduceAll(BTreePtr p);

void Sreduce1(BTreePtr p);
int  ScanReduce(BTreePtr p);
BTreePtr SreadPrefix(void);
BTreePtr SreadPrefRec(void);


int SinSMM(BTreePtr p);

int SisNormal(BTreePtr p);
int SinL0(BTreePtr p);
int SinL1(BTreePtr p);
int SinL2(BTreePtr p);
int SinL3(BTreePtr p);

int SinH0(BTreePtr p);
int SinH1(BTreePtr p);

int SinQ2(BTreePtr p);
int SinQ3(BTreePtr p);

int SinQ3Q2(BTreePtr p);
int SinBQ3(BTreePtr p);
int SinSQ3M(BTreePtr p);


int SinpredL0(BTreePtr p);
int SinpredL1(BTreePtr p);
```

```
int SinpredL2(BTreePtr p);
int SinpredH0(BTreePtr p);

int SinL0a(BTreePtr p);
int SinN00(BTreePtr p);

int SinL1b(BTreePtr p);

int SinL2b(BTreePtr p);
int SinL2c(BTreePtr p);

int SinH0b(BTreePtr p);
int SinH0c(BTreePtr p);
int SinH0d(BTreePtr p);
int SinH0e(BTreePtr p);

/* B is S(SS) */
int SisSS(BTreePtr p);
int SisB(BTreePtr p);
int SisBS(BTreePtr p);
int SisSBS(BTreePtr p);
int SisSBpSSp(BTreePtr p);
int SisBB(BTreePtr p);
int SisBpSSp(BTreePtr p);
int SisSB(BTreePtr p);
int SisApSSp(BTreePtr p);
int SisA(BTreePtr p);

/* 1 if two terms are equal, else 0 */
int SisEqual(BTreePtr p, BTreePtr q);

/* subterms first strategy */
void  SreduceSubs(BTreePtr p);

/* 3rd argument first strategy */
void  Sreduce3rdarg(BTreePtr p);

/* define reduce strategy! */
#define REDUCESTRATEGY Sreduce3rdarg
```

```
/* only one reduction strategies */
void  SreduceLeftmost1(BTreePtr p);



int Slength(BTreePtr p);
int SlengthLeftRootSub(BTreePtr p);


char * firstSterm(int length);
int    isLastSterm(const char *s, int length);
void   nextSterm(char *s, int length);
void   generateSterms(int length);

int    checkSterms(int length);


#ifdef GLOBREDCOUNT
/* reductions counter global variable */
int Sredcount;
#endif /* GLOBREDCOUNT */

/* number of S's in S-Term */
int Slength(BTreePtr p)
{
   if ((p == NULL))
   {
      return 0;
   }
   else
   {
      if (p->label == NONORMAL)
      {
         return 0;
      }
      else
      {
         if (p->label == APPLY)
         {
            return Slength(p->left) + Slength(p->right);
```

```
         }
         else /* (p->label == SCOMB) */
         {
            return 1;
         }
      }
   }
}



/* length (number of S's in S-Term)
   of left subterm of term p  */
int SlengthLeftRootSub(BTreePtr p)
{
   if ((p == NULL))
   {
      return 0;
   }
   else
   {
      if (p->label == NONORMAL)
      {
         return 0;
      }
      else
      {
         if (p->label == APPLY)
         {
            return Slength(p->left);
         }
         else /* (p->label == SCOMB) */
         {
            return 0;
         }
      }
   }
}
```

```
void SprintPrefix(BTreePtr p)
{
   if (p->label == SCOMB)
      printf(" S");
   else if (p->label == APPLY)
   {
      printf("o");
      SprintPrefix(p->left);
      SprintPrefix(p->right);
   }
}


void SprintInfix(BTreePtr p)
{
   if (p->label == SCOMB)
      printf("S");
   else if (p->label == APPLY)
   {
      SprintInfix(p->left);
      if (p->right->label == SCOMB)
      {
         printf("S");
      }
      else
      {
         printf("(");
         SprintInfix(p->right);
         printf(")");
      }
   }
}

/* print with syntomeyseis:  A = SSS,  B = S(SS) */
void SprintInfixSyn(BTreePtr p)
{
   if (p->label == SCOMB)
      printf("S");
   else if (SisA(p))
      printf("A");
```

```
   else if (SisB(p))
      printf("B");
   else if (p->label == APPLY)
   {
      SprintInfixSyn(p->left);
      if ((p->right->label == SCOMB) ||
          (SisA(p->right)) || (SisB(p->right)))
      {
         SprintInfixSyn(p->right);
      }
      else
      {
         printf("(");
         SprintInfixSyn(p->right);
         printf(")");
      }
   }
}

/* print with syntomeyseis:  A = SSS,  B = S(SS) */
/* substitute in printout term Dn with "Dn" */
void SprintInfixSynDn(BTreePtr p, BTreePtr Dn)
{
   if (SisEqual(p, Dn))
      printf("D_n");
   else if (p->label == SCOMB)
      printf("S");
   else if (SisA(p))
      printf("A");
   else if (SisB(p))
      printf("B");
   else if (p->label == APPLY)
   {
      SprintInfixSynDn(p->left, Dn);
      if ((SisEqual(p->right, Dn))          ||
          (p->right->label == SCOMB) ||
          (SisA(p->right)) || (SisB(p->right)))
      {
         SprintInfixSynDn(p->right, Dn);
      }
      else
```

```
      {
         printf("(");
         SprintInfixSynDn(p->right, Dn);
         printf(")");
      }
   }

}



/* quite bad complexity */
int SreduceAll(BTreePtr p)
{
   int reducing = 1;
   int somereduce = 0;
#ifdef DEBUGX
   printf("in SreduceAll\n");
#endif
   if (p->label == SCOMB)
      return somereduce;
   do
   {
      if (ScanReduce(p))
      {
         Sreduce1(p);
         somereduce = 1;
      }
      else
      {
         if ((reducing = (SreduceAll(p->left) ||
             SreduceAll(p->right))))
            somereduce = 1;
      }
   } while (reducing);
   return somereduce;
}


int ScanReduce(BTreePtr p)
{
```

```
   if (p->label == APPLY)
      if (p->left->label == APPLY)
         if (p->left->left->label == APPLY)
            if (p->left->left->left->label == SCOMB)
               return 1;
   /* in all other cases */
   return 0;
}


void Sreduce1(BTreePtr p)
{
   BTreePtr pNew;

   /* here, p->right is Z and
      p->left is S X Y */
   /*
              p --> o
                   / \
                  o  [Z]
                 / \
                o  [Y]
               / \
              S  [X]

   */

   /* assert(ScanReduce(p)); */

#ifdef GLOBREDCOUNT
   Sredcount++;
#ifdef DEBUGREDCOUNT
   printf("Sreduce1: reduction number: %10d\n", Sredcount);
#endif /* DEBUGREDCOUNT */
#endif /* GLOBREDCOUNT */

   pNew = BTreeCopy(p->right);
   assert(pNew != NULL);

   p->left->left->left->left  = p->left->right;
   p->left->left->left->right = p->right;
```

```
   p->right = p->left->left->left ;
   p->right->label = APPLY;
   p->left->right = pNew;


   /*               p -->
                         o
                        ----- -----
                       /           \
                      o             o
                     / \           / \
                    o  [Zcopy]   [Y] [Z]
                   / \
                  ..   [X]
   */

   /* use pNew to delete node over [X] above */
   pNew = p->left->left;
   p->left->left = pNew->right;
   BTreeNodeFree(pNew);


   /*
                    p-->o
                      /   \
                     o       o
                    / \     / \
                  [X] [Z] [Y] [Z]

   */


}


int SinSMM(BTreePtr p)
{
   if (p->label == APPLY)
      if (p->left->label == APPLY)
         if (p->left->left->label == SCOMB)
            return 1;
```

```
   /* otherwise ... */
   return 0;
}


int SisNormal(BTreePtr p)
{
   if (p->label == SCOMB) { return 1; }
   else /* if (p->label == APPLY) */
   {
      if ((p->left->label == SCOMB))
      {
         if (SisNormal(p->right)) { return 1; }
         else { return 0; }
      }
      else /* if ((p->left->label == APPLY)) */
      {
         if (p->left->left->label == SCOMB)
         {
            if (SisNormal(p->right) &&
                SisNormal(p->left->right))
            {   return 1;  }
            else { return 0;}
         }
         else
            return 0;
      }
   }
}


int SinL0(BTreePtr p)
{
   while ((p->label == APPLY) && SisSS(p->left))
   {
      p = p->right;
   }

   if (p->label == SCOMB)
   {
```

```
#ifdef DEBUGX
      printf("Found S-constant\n");
#endif
      return 1;
   }
   else if (p->left->label == SCOMB)
   {
      if (SisNormal(p->right))
         return 1;
      else
         return 0;
   }
   else
   {
      return 0;
   }
}


int SinL1(BTreePtr p)
{
   while ((p->label == APPLY) && SisSS(p->left))
   {
      p = p->right;
   }

   return SisBS(p) || SisSBS(p);
}


int SinL2(BTreePtr p)
{
   while ((p->label == APPLY) && SisSS(p->left))
   {
      p = p->right;
   }

   return SisBpSSp(p) || SisBB(p);
}
```

```
int SinL3(BTreePtr p)
{
   while ((p->label == APPLY) && SisSS(p->left))
   {
      p = p->right;
   }

   return SisSBpSSp(p) || SinBQ3(p);
}



int SinH0(BTreePtr p)
{
   while ((p->label == APPLY) &&
          (SisSS(p->left) || SisB(p->left)))
   {
      p = p->right;
   }

   if (p->label == SCOMB) return 1;

   assert(p->label == APPLY);

   if (p->left->label == SCOMB)
      return SisNormal(p->right);

   return SisSBS(p) || SisSBpSSp(p);
}


int SinH1(BTreePtr p)
{
   while ((p->label == APPLY) &&
          (SisSS(p->left) || SisB(p->left)))
   {
      p = p->right;
   }
```

```
   return SinQ3Q2(p) || SinSQ3M(p);
}


int SinQ2(BTreePtr p)
{
   return !((p->label == SCOMB) || SisSS(p));
}

int SinQ3(BTreePtr p)
{
   return !((p->label == SCOMB) || SisSS(p) || SisB(p));
}


int SinQ3Q2(BTreePtr p)
{
   if (p->label == APPLY)
      return (SinQ3(p->left) && SinQ2(p->right));
   else
      return 0;
}

int SinBQ3(BTreePtr p)
{
   if (p->label == APPLY)
      return (SisB(p->left) && SinQ3(p->right));
   else
      return 0;
}

int SinSQ3M(BTreePtr p)
{
   if (p->label == APPLY)
      if (p->left->label == APPLY)
         return ((p->left->left->label == SCOMB) &&
                 SinQ3(p->left->right));

   /* in all other cases */
   return 0;
}
```

```
int SisSS(BTreePtr p)
{
   if (p->label == APPLY)
      if ((p->left->label  == SCOMB) &&
          (p->right->label == SCOMB))
         return 1;

   return 0;  /* false otherwise */
}

/* B is S(SS) */
int SisB(BTreePtr p)
{
   if (p->label == APPLY)
      if ((p->left->label == SCOMB) &&
          SisSS(p->right))
         return 1;

   return 0;  /* false otherwise */
}


/* A is SSS */
int SisA(BTreePtr p)
{
   if (p->label == APPLY)
      if ((p->right->label == SCOMB) &&
          SisSS(p->left))
         return 1;

   return 0;  /* false otherwise */
}



int SisBS(BTreePtr p)
{
   if (p->label == APPLY)
      if (SisB(p->left) &&
```

```
          (p->right->label == SCOMB))
         return 1;

   return 0;  /* false otherwise */
}


int SisSB(BTreePtr p)
{
   if (p->label == APPLY)
      if (SisB(p->right) &&
          (p->left->label == SCOMB))
         return 1;

   return 0;  /* false otherwise */
}



int SisSBS(BTreePtr p)
{
   if (p->label == APPLY)
      if (p->right->label == SCOMB)
         if (p->left->label == APPLY)
            if ((p->left->left->label == SCOMB) &&
                SisB(p->left->right))
               return 1;

   return 0;  /* false otherwise */
}

/* recognizes SB(SS) */
int SisSBpSSp(BTreePtr p)
{
   if (p->label == APPLY)
      if (SisSS(p->right))
         if (p->left->label == APPLY)
            if ((p->left->left->label == SCOMB) &&
                SisB(p->left->right))
               return 1;
```

```
   return 0;  /* false otherwise */
}


int SisBpSSp(BTreePtr p)
{
   if (p->label == APPLY)
      if (SisB(p->left))
         if (SisSS(p->right))
            return 1;

   return 0;
}

int SisApSSp(BTreePtr p)
{
   if (p->label == APPLY)
      if (SisA(p->left))
         if (SisSS(p->right))
            return 1;

   return 0;
}




int SisBB(BTreePtr p)
{
   if (p->label == APPLY)
      if (SisB(p->left) && SisB(p->right))
         return 1;

   return 0;  /* false otherwise */
}



/* the below sometimes use normalization,
   change in algorithm imminent */
```

```
int SinpredL0(BTreePtr p)
{
   if (SinL0a(p))
      return 1;

   if (p->label == NONORMAL)
      return 0;

   return SinN00(p);
}


int SinN00(BTreePtr p)
{
#ifdef DEBUG
   printf("********** SinN00: checking ");
   SprintPrefix(p);
   printf("\n");
#endif
   if (p->label == SCOMB)
   {
      return 1;
   }
   else
   {
      if (p->label == APPLY);
        return (SinN00(p->left) && p->right->label == SCOMB)
            || (SisSS(p->left) && SinN00(p->right)) ;
   }
#ifdef DEBUGN00
   printf("********** SinN00: NONORMAL HEAD!!!\n");
#endif
   return 0;
}

int SinL0a(BTreePtr p)
{
   if (p->label == APPLY)
   {
      if (SisSS(p->left) && SinL0a(p->right))
         return 1;
```

```
      if (p->left->label == SCOMB)
      {
         assert(p->right->label != NONORMAL);
#ifdef DEBUG
         printf("!!! SinL0a calling Snormalize: ");
         SprintPrefix(p->right);
         printf("\n");
#endif
         Snormalize(p->right);
#ifdef DEBUG
         printf("!!! SinL0a successful Snormalize: ");
         SprintPrefix(p->right);
         printf("\n");
#endif
         if (p->right->label != NONORMAL)
         {
            return 1;
         }
         else /* non-normalizing!!! */
         {
            BTreeFree(p->left);
            BTreeFree(p->right);
            p->label = NONORMAL;
            return 0;
         }
      }
   }

   /* in all other cases ... */
   /* p->label == NONORMAL included here!!! */
   return 0;
}


int SinpredL1(BTreePtr p)
{
#ifdef DEBUG
   printf("SinpredL1: checking ");
   SprintPrefix(p);
   printf("\n");
```

```
#endif
   if (p->label == APPLY)
   {
      return (SisSS(p->left) && SinpredL1(p->right))
          || (SinL1b(p->left) && p->right->label == SCOMB) ;
   }
   return 0;
}


int SinL1b(BTreePtr p)
{
#ifdef DEBUG
   printf("SinL1b: checking ");
   SprintPrefix(p);
   printf("\n");
#endif
   if (p->label == APPLY)
   {
      return SisB(p) || SisSB(p) ||
             (SisSS(p->left) && SinL1b(p->right)) ;
   }

   return 0;
}


int SinpredL2(BTreePtr p)
{
#ifdef DEBUG
   printf("SinpredL2: checking ");
   SprintPrefix(p);
   printf("\n");
#endif
   if (p->label == APPLY)
   {
      return SisBB(p) || SisApSSp(p) || SisBpSSp(p)
          || (SisSS(p->left)  && SinpredL2(p->right))
          || (SinL2b(p->left) && p->right->label == SCOMB) ;
   }
   return 0;
```

```
}


int SinL2b(BTreePtr p)
{
#ifdef DEBUG
   printf("SinL2b: checking ");
   SprintPrefix(p);
   printf("\n");
#endif

   if (p->label == APPLY)
   {
      return (SisSS(p->left) && SinL2b(p->right))
          || (SinL2c(p->left) && p->right->label == SCOMB) ;
   }

   return 0;

}


int SinL2c(BTreePtr p)
{
   if (p->label == APPLY)
   {
      return SisB(p)  /* || SisSB(p) */
          || (SisSS(p->left) && SinL2c(p->right)) ;
   }

   return 0;
}


int SinpredH0(BTreePtr p)
{
#ifdef DEBUG
   printf("SinpredH0: checking ");
   SprintPrefix(p);
   printf("\n");
#endif
```

```
   if (p->label == SCOMB) return 1;

   if (SisBB(p) || SisApSSp(p) || SisBpSSp(p))
      return 1;

   if (p->label == APPLY)
   {
      if (   ( (SisSS(p->left) || SisB(p->left))
             && SinpredH0(p->right))
          || (SinH0b(p->left) && SisSS(p->right))
          || (SinH0c(p->left) && (p->right->label == SCOMB))
         )
         return 1;

      /* pred(N) case */
      if (p->left->label == SCOMB)
      {
         assert(p->right->label != NONORMAL);
#ifdef DEBUG
         printf("!!! SinpredH0 calling Snormalize: ");
         SprintPrefix(p->right);
         printf("\n");
#endif
         Snormalize(p->right);
#ifdef DEBUG
         printf("!!! SinpredH0 successful Snormalize: ");
         SprintPrefix(p->right);
         printf("\n");
#endif

         if (p->right->label != NONORMAL)
         {
            return 1;
         }
         else /* non-normalizing!!! */
         {
            BTreeFree(p->left);
            BTreeFree(p->right);
            p->label = NONORMAL;
            return 0;
```

```
         }
      }
   }
   /* all other cases ... */
   return 0;
}


int SinH0b(BTreePtr p)
{
   return SinN00(p) || SinL1b(p);
}


int SinH0c(BTreePtr p)
{
   return SinH0b(p) || SinH0d(p);
}

int SinH0d(BTreePtr p)
{
   if (p->label == APPLY)
   {
      return (SisSS(p->left)  && SinH0d(p->right))
          || (SinH0e(p->left) && (p->right->label == SCOMB));
   }
   return 0;
}

int SinH0e(BTreePtr p)
{
   if (p->label == APPLY)
   {
      return (SisSS(p->left)  && SinH0e(p->right))
          || ((p->left->label == SCOMB) && SinN00(p->right));
   }
   return 0;
}


/* CAUTION: assumes a legal infix representation
```

```
            of an S-Term  */
BTreePtr SreadPrefix(void)
{
   int c;
   int firstnode = 1;
   BTreePtr pNew, pReturn;

   pReturn = NULL;

   while ((c = getchar()) != EOF)
   {
      if (c == 'o')
      {
         pNew = BTreeNodeNew();
         pNew->label = APPLY;
         if (firstnode)
         {
            pReturn = pNew;
            firstnode = 0;
         }
         pNew->left  = SreadPrefRec();
         pNew->right = SreadPrefRec();
         break;
      }

      if (c == 'S')
      {
         pNew = BTreeNodeNew();
         pNew->label = SCOMB;
         if (firstnode)
         {
            pReturn = pNew;
            firstnode = 0;
         }
         break;
      }
   }

   return pReturn;
}
```

```
BTreePtr SreadPrefRec(void)
{
   int c;
   int firstnode = 1;
   BTreePtr pNew, pReturn;

   pReturn = NULL;

   while ((c = getchar()) != EOF)
   {
      if (c == 'o')
      {
         pNew = BTreeNodeNew();
         pNew->label = APPLY;
         if (firstnode)
         {
            pReturn = pNew;
            firstnode = 0;
         }
         pNew->left  = SreadPrefRec();
         pNew->right = SreadPrefRec();
         break;
      }

      if (c == 'S')
      {
         pNew = BTreeNodeNew();
         pNew->label = SCOMB;
         if (firstnode)
         {
            pReturn = pNew;
            firstnode = 0;
         }
         break;
      }
   }

   return pReturn;
}
```

```
/*  S-Terms are considered in prefix notation:
    e.g.: ooSoSoSSS

    for all terms with length n, i.e. with n S's
    (the number of such terms is a_n, nth Catalan Number)

    we consider the lexicographic order:

    oooo...SSSSS
    ......
    oSoSoS...oSS

    first term is:
      oooo...SSSSS with n-1 o's followed by n S's

    last term is:
      oSoSoS...oSS with n-1 oS's followed by an S

*/


char * firstSterm(int length)
{
   char *s;
   int i,j;

   s = (char *) calloc(sizeof(char), 2*length + 1);

   if (s == NULL)
      exit(2);

   for (i = 1, j = 0; i <= length - 1; i++, j++)
   {
      s[j] = 'o';
```

```
   }

   for (i = 1; i <= length; i++, j++)
   {
      s[j] = 'S';
   }

   s[j] = '\0'; /* null terminate */

   return s;

}


int isLastSterm(const char *s, int length)
{
   int i,j;
   assert(length > 0);
   if (strlen(s) != 2*length - 1)
   {
      return 0;
   }
   else
   {
      for (i = 1, j = 0; i <= length - 1; i++, j = j+2)
      {
         if ((s[j] != 'o') || (s[j+1] != 'S'))
            return 0;
      }
      if (s[j] == 'S')
      {
         return 1;
      }
      else
      {
         return 0;
      }
   }
}
```

```
void nextSterm(char *s, int length)
{
   int i, start;
   char temp;

   assert(strlen(s) == 2* length - 1);
   assert(!isLastSterm(s,length));

   start = 2*length - 4;

   i = start;

   /* */
   while (s[i] == 'S')
   {
      i--;
   }

   if (i != start)
   {
      /* exchange first S in S-cluster with current 'o' */
      s[i] = 'S';
      s[i+1] = 'o';
   }
   else
   /* at position 2n-4 we have an 'o'
      this requires special action!!!
   */
   {
      int countS;
      int counter;
      int foundcluster; /* boolean */
      int howmanyos;
      /* in next position there must be an S */
      assert(s[i-1] == 'S');

      countS = 0;
      foundcluster = 0;

      while(1) /* loop with internal break(s) */
      {
```

```
         i--;
         if (s[i] == 'S')
         {
            countS++;
         }

         if (foundcluster)
         {
            if (s[i] == 'o')
               break;
         }
         else
         {
            if ((s[i] == 'S') && (s[i+1] == 'S'))
              foundcluster = 1;
         }

      }

      s[i] = 'S';
      howmanyos = start - i - (countS - 1);

      i++;
      for (counter = 1; counter <= howmanyos; counter++, i++)
      {
         s[i] = 'o';
      }

      for (; i <= start; i++)
      {
         s[i] = 'S';
      }

   }
}


void generateSterms(int length)
{
   char *s;
```

```
   assert(length > 0);

   s = firstSterm(length);

   printf("%s ;\n", s);

   while (!(isLastSterm(s, length)))
   {
      nextSterm(s, length);
      printf("%s ;\n", s);
   }

}


int checkSterms(int length)
{
   int countNoNF = 0;
   char *s;
   char *command;

   assert(length > 0);

   command = calloc(length + 100, sizeof(char));
   assert(command != NULL);

   s = firstSterm(length);

   sprintf(command, "echo \"%s\" | runme", s);
   if (system(command) == 1)
   {
      countNoNF++;
   }

   while (!(isLastSterm(s, length)))
   {
      nextSterm(s, length);

      sprintf(command, "echo \"%s\" | runme", s);
      if (system(command) == 1)
      {
```

```
            countNoNF++;
         }
      }

      free(s);
      free(command);

      return countNoNF;

}


void  SreduceSubs(BTreePtr p)
{

      if (p->label == APPLY)
      {
         SreduceSubs(p->left);
         SreduceSubs(p->right);
      }

      if (ScanReduce(p))
      {
         Sreduce1(p);
      }
}


/* quite unoptimized */
void Sreduce3rdarg(BTreePtr p)
{
      BTreePtr piter, p3rd, pReduce;

      if (p->label == SCOMB)
         return;

      /* loop with internal break */
      do  {
         p3rd = p;
         piter = NULL;
         if (p->label == APPLY)
```

```
         if (p->left->label == APPLY)
            if (p->left->left->label == APPLY)
               piter = p->left->left->left;

      if (piter == NULL)
      {
         Sreduce3rdarg(p->right);
         if (p->left->label == APPLY)
            Sreduce3rdarg(p->left->right);
         break;
      }
      else
      {
         while (piter->label != SCOMB)
         {
            piter = piter->left;
            p3rd  = p3rd->left;
         }

         Sreduce3rdarg(p3rd->right);

         assert(ScanReduce(p3rd));

         Sreduce1(p3rd);

      }
   }
   while (1);
}


void Snormalize(BTreePtr p)
{
   /* BTreePtr pNew; */

   assert(p->label != NONORMAL);

   if ((p->label == SCOMB) || (p->label == NONORMAL))
   {
      return; /* do nothing */
   }
```

```
   else
   {
/* preliminary checks to avoid reductions in case of no NF */
#ifdef PRELIMCHECKS
      if (SinH1(p->left))
         if (SinQ2(p->right))
         {
#ifdef DEBUGPREL
            printf("PREL: no nf\n");
#endif
            BTreeFree(p->left);
            BTreeFree(p->right);
            p->label = NONORMAL;
            return;
         }

      if (SinH1(p->right))
         if (SinL3(p->left) || SinL2(p->left) ||
             SinL1(p->left))
         {
#ifdef DEBUGPREL
            printf("PREL: no nf\n");
#endif
            BTreeFree(p->left);
            BTreeFree(p->right);
            p->label = NONORMAL;
            return;
         }

      if (SinL3(p->left))
         if (SinL3(p->right) ||
             SinL2(p->right) ||
             SinL1(p->right))
         {
#ifdef DEBUGPREL
            printf("PREL: no nf\n");
#endif
            BTreeFree(p->left);
            BTreeFree(p->right);
            p->label = NONORMAL;
            return;
```

```
         }


      if (SinL3(p->right))
         if (SinL2(p->left))
         {
#ifdef DEBUGPREL
            printf("PREL: no nf\n");
#endif
            BTreeFree(p->left);
            BTreeFree(p->right);
            p->label = NONORMAL;
            return;
         }

#endif /* PRELIMCHECKS */

      Snormalize(p->left);
      if (p->left->label == NONORMAL)
      {
#ifdef DEBUGX
         printf("left no normal at %c\n", (char) p->label);
#endif
         BTreeFree(p->left);
         BTreeFree(p->right);
         p->label = NONORMAL;
         return;
      }
      Snormalize(p->right);
      if (p->right->label == NONORMAL)
      {
#ifdef DEBUGX
         printf("right no normal at %c\n", (char) p->label);
#endif
         BTreeFree(p->left);
         BTreeFree(p->right);
         p->label = NONORMAL;
         return;
      }

      /* here, subterms are in normal form */
```

```
#ifdef DEBUGX
      printf("check subterms at %c\n", (char) p->label);
#endif


      if ((SinL0(p->left)) ||
          (SinH0(p->left) && SinL0(p->right)) ||
          (SinL1(p->left) && SinH0(p->right)) ||
          (SinL2(p->left) && SinL1(p->right))
         )
      {
         /* p has a normal form
            reduce repeatedly to find it */
#ifdef DEBUG
         SprintPrefix(p);
         printf("  of converging form\n");
         printf("%d %d %d %d %d %d %d \n",
                SinL0(p->left),
                SinH0(p->left),
                SinL1(p->left),
                SinL2(p->left),
                SinL0(p->right),
                SinH0(p->right),
                SinL1(p->right));
#endif

#ifdef DEBUG
         printf("trying reducing ");
         SprintPrefix(p);
         printf("\n");
#endif
         (void) REDUCESTRATEGY(p);
#ifdef DEBUG
         printf("reduced form: ");
         SprintPrefix(p);
         printf("\n");
#endif
         return;
      }
      else
```

```
      {
         /* check if p->right != S and != SS */
         assert(p->label == APPLY);
         if ((p->right->label != SCOMB) && !(SisSS(p->right)))
         {
            BTreeFree(p->left);
            BTreeFree(p->right);
            p->label = NONORMAL;
            return;
         }
         else
         {
            /* here, p->right is S or SS and
               p->left is S N1 N2 */
            /*
                        p --> o
                             / \
                            o  [rt]
                           / \
                          o  [N2]
                         / \
                        S  [N1]

            */

            assert(ScanReduce(p));
            Sreduce1(p);

/* the code below seems not needed! */
/*
            Snormalize(p->left);
            if (p->left->label == NONORMAL)
            {
               BTreeFree(p->left);
               BTreeFree(p->right);
               p->label = NONORMAL;
               return;
            }

            Snormalize(p->right);
            if (p->right->label == NONORMAL)
```

```
            {
               BTreeFree(p->left);
               BTreeFree(p->right);
               p->label = NONORMAL;
               return;
            }
*/

            Snormalize(p);

         }
      }

   }
}


void  SreduceLeftmost1(BTreePtr p)
{
   (void) SreduceLeftmost1Rec(p);
}

/* returns 1 if reduction is made else 0 */
int   SreduceLeftmost1Rec(BTreePtr p)
{

   if (p->label == SCOMB)
      return 0;

   if (p->label == APPLY)
      if (p->left->label == APPLY)
         if (p->left->left->label == APPLY)
            if (p->left->left->left->label == SCOMB)
            {
               Sreduce1(p);
               return 1;
            }

   if (SreduceLeftmost1Rec(p->left))
      return 1;
   else
```

```
      return SreduceLeftmost1Rec(p->right);
}


int SisEqual(BTreePtr p, BTreePtr q)
{
   if (p == q) return 1;

   if ((p->label == q->label))
   {
      if (p->label == SCOMB)
         return 1;
      else
         return SisEqual(p->left,  q->left)  &&
                SisEqual(p->right, q->right)   ;
   }
   else
      return 0;
}


int SisSameFunction(
  BTreePtr x, BTreePtr xprev, BTreePtr y, BTreePtr yprev)
{
   if (x->label != y->label)
   {
      return 0;
   }

   if (x->label == SCOMB)
   {
      return 1;
   }

   if (SisEqual(x, xprev) && SisEqual(y, yprev))
   {
      return 1;
   }

   if (x->label == APPLY)
   {
```

```
      return SisSameFunction(x->left, xprev, y->left, yprev)
         && SisSameFunction(x->right, xprev, y->right, yprev);
   }

   exit(1);
}


#include "sminnonf.h"

#include "schain.h"


#endif  /* sterm.h */
```

## A.3 Αρχείο sminnonf.h

```
int SisNormalizing(BTreePtr p);
BTreePtr SminNoNF(BTreePtr p);
BTreePtr SminNoNFsame(BTreePtr p);

int SisNormalizing(BTreePtr p)
{
   int retval = 1;
   BTreePtr copy = BTreeCopy(p);
   Snormalize(copy);
   if (copy->label == NONORMAL) retval = 0;
   BTreeFree(copy);
   return retval;
}



/* INPUT: a non-normalizing term
   OUTPUT:
   returns a pointer to a newly allocated term,
   that is (the new term) the minimum (with respect to length)
   subterm of the input term that is not normalizing
 */
BTreePtr SminNoNF(BTreePtr p)
```

```
{
   int existsminleft, existsminright;
   int lenleft, lenright;
   BTreePtr minleft, minright, retp;

   existsminleft = 0;  existsminright = 0;

   assert(!SisNormalizing(p));

   if (!SisNormalizing(p->left))
   {
      existsminleft = 1;
      minleft = SminNoNF(p->left);
   }

   if (!SisNormalizing(p->right))
   {
      existsminright = 1;
      minright = SminNoNF(p->right);
   }

   if ((!existsminleft) && (!existsminright))
   {
      retp = BTreeCopy(p);
   }

   if ((existsminleft) && (!existsminright))
   {
      retp = minleft;
   }

   if ((!existsminleft) && (existsminright))
   {
      retp = minright;
   }

   if ((existsminleft) && (existsminright))
   {
      lenleft   = Slength(minleft);
      lenright  = Slength(minright);
```

```
      assert(lenleft  > 0);
      assert(lenright > 0);
      assert(lenleft  < Slength(p));
      assert(lenright < Slength(p));

      if (lenleft < lenright)
      {
         retp = minleft;
         BTreeFree(minright);
      }
      else
      {
         retp = minright;
         BTreeFree(minleft);
      }
   }

   return retp;
}


/* same as SminNoNF, but in case:
   the minimum non-normalizing subterm equals the input term,
   it DOES NOT copy the input term,
   but just returns a pointer to the input term
   (easily to be checked with p_output == p_input)
*/
BTreePtr SminNoNFsame(BTreePtr p)
{
   int existsminleft, existsminright;
   int lenleft, lenright;
   BTreePtr minleft, minright, retp;

   existsminleft = 0;  existsminright = 0;

   assert(!SisNormalizing(p));

   if (!SisNormalizing(p->left))
   {
      existsminleft = 1;
      minleft = SminNoNF(p->left);
```

```
}

if (!SisNormalizing(p->right))
{
   existsminright = 1;
   minright = SminNoNF(p->right);
}

if ((!existsminleft) && (!existsminright))
{
   retp = p;
}

if ((existsminleft) && (!existsminright))
{
   retp = minleft;
   BTreeFree(minright);
}

if ((!existsminleft) && (existsminright))
{
   retp = minright;
   BTreeFree(minleft);
}

if ((existsminleft) && (existsminright))
{
   lenleft   = Slength(minleft);
   lenright  = Slength(minright);

   assert(lenleft  > 0);
   assert(lenright > 0);
   assert(lenleft  < Slength(p));
   assert(lenright < Slength(p));

   if (lenleft < lenright)
   {
      retp = minleft;
      BTreeFree(minright);
   }
   else
```

```
      {
         retp = minright;
         BTreeFree(minleft);
      }
   }

   return retp;
}
```

## A.4 Αρχείο schain.h

Σε αυτό το αρχείο βρίσκεται πειραματικός κώδικας για την προσπάθεια περιγραφής αλυσίδας στην περίπτωση απείρων αναγωγών, σύμφωνα με την μέθοδο Ζάχου. Αυτός ο κώδικας μπορεί να φανεί χρήσιμος για την εύρεση περιγραφών απείρων αλυσίδων αναγωγών, όπως στο κεφάλαιο 14, και αποτελεί την πρώτη προσπάθεια για μία γενική διαδικασία που θα δίνει περιγραφή αλυσίδας αναγωγής σύμφωνα με την μέθοδο Ζάχου για κάθε μη κανονικοποιήσιμο όρο.

```
#ifndef SCHAIN_H

#define SCHAIN_H 1

#define DEBUG
#undef DEBUG

#ifdef DEBUG
#define listfree(list) \
   { \
      for (k = 0; k <= i; k++) \
      { \
         printf("listfree: Deleting list[%d]...\n", k); \
         if (list[k] != NULL) BTreeFree(list[k]); \
         list[k] = NULL; \
      } \
   }
#endif

#ifndef DEBUG
```

```
#define listfree(list) \
   { \
      for (k = 0; k <= i; k++) \
      { \
         if (list[k] != NULL) BTreeFree(list[k]); \
         list[k] = NULL; \
      } \
   }
#endif


#define Sresetlist(p) \
   {                     \
      listfree(list);     \
      i = 0;                \
      list[i] = BTreeCopy(p); \
      lenPossibleD = Slength(p->left); \
   }

#define SsetMinD(p) \
   {                     \
      BTreeFree(pMinD);     \
      pMinD = BTreeCopy(p); \
      lenMinD  = Slength(p->left); \
      lenPrevD = lenMinD; \
   }


#define SlistFirst(p, counterval) \
   {                     \
      i = 0; \
      list[i].term    = BTreeCopy(p); \
      list[i].lenLeft = Slength(p->left); \
      list[i].counter = counterval; \
   }

#define SlistNext(p, counterval) \
   {                     \
      list[i].term    = BTreeCopy(p); \
      list[i].lenLeft = Slength(p->left); \
      list[i].counter = counterval; \
```

```
   }

#define SisListEmpty(list) (list[0].term == NULL)


#define HUGEINT 100000000


#ifdef DEBUG
#define freeAll() \
{ \
   printf("freeall: Started cleaning: "); \
   printf("pMinD "); \
   BTreeFree(pMinD); \
   printf(" [OK], "); \
   printf("p "); \
   BTreeFree(p); \
   printf(" [OK], "); \
   printf("list "); \
   listfree(list); \
   printf(" [OK]\n"); \
}
#endif


#ifndef DEBUG
#define freeAll() \
{ \
   BTreeFree(pMinD); \
   BTreeFree(p); \
   listfree(list); \
}
#endif


#define SfreeChain() \
{ \
   BTreeFree(p); \
   for (k = 0; k <= i; k++) \
   { \
      BTreeFree(list[k].term); \
```

```
   } \
}


#define LENLIST 100
#define SchainFind SchainFindFirstSmallestRecurring

/* Input:
   a term t (non-normalizing) else function returns 2
        (term t is left untouched),
   maxcount:
      max number of reductions to make in search of NF

   Output:
   possibly D, D_0 and D_1 of chain. (D_np1 is D_1)
   we suppose D_{n+1} = f(D_n),
   where f a function such that D_1 = f(D_0)

   print D_np1 using SprintInfixSynDn(D_np1, D_0)
*/


int SchainFindFirstSmallestRecurring(
  BTreePtr t, int maxcount,
  BTreePtr *D, BTreePtr *D_0, BTreePtr *D_1)
{
   BTreePtr p, pnext;
   int counter = 0;
   int lenPossibleD;
   int lenLeft;
   int lenMinD;
   int lenPrevD;
   int i, k;
   int notfound, searchDPhase;

   struct {
     BTreePtr term;
     int      counter;
     int      lenLeft;
   } list[LENLIST];
```

```
#ifdef DEBUG
   printf("\nUsing algorithm: "
          "first smallest recurring.\n");
#endif

   *D = *D_0 = *D_1 = NULL;

   if (SisNormalizing(t))
   {
      return 2;
   }

   assert(maxcount > 0);
   assert(maxcount < 10000);

   for (k = 0; k < LENLIST; k++)
      list[k].term = NULL;

   i = 0;
   p = BTreeCopy(t);
   pnext = SminNoNF(p);
   BTreeFree(p);
   p = pnext;

   searchDPhase = 1;
   if (SinSMM(p->left))
   {
#ifdef DEBUG
      printf("Setting first element from start: D = ");
      SprintInfixSyn(p->left);
      printf(", Dstart = ");
      SprintInfixSyn(p->right);
      printf("\n");
#endif
      SlistFirst(p, counter);
   }

   while (counter < maxcount)
   {
      counter++;
      SreduceLeftmost1(p);
```

```
      pnext = SminNoNF(p);
      BTreeFree(p);
      p = pnext;
      lenLeft = Slength(p->left);

      if (SinSMM(p->left))
      {

         if (searchDPhase)
         {
            if (SisListEmpty(list))
            {
#ifdef DEBUG
               printf("Setting first: D = ");
               SprintInfixSyn(p->left);
               printf(", Dstart = ");
               SprintInfixSyn(p->right);
               printf("\n");
#endif
               SlistFirst(p, counter);
            }
            else
            {
               notfound = 1;
               for (k = i; k >= 0; k--)
               {
                  if (SisEqual(p->left,
                              list[k].term->left))
                  {
                     notfound = 0;
                     break;
                  }
               }

               if (notfound)
               {
                  i = i + 1;
                  SlistNext(p, counter);
               }
               else
               {
```

```
                    /* found possibly recurring D at pos k */
#ifdef DEBUG
                    printf("found "
                           "possibly recurring D at pos ");
                    printf("list[%d]\n", k);
#endif
                    if (k != 0)
                    {
                       BTreeFree(list[0].term);
                       list[0].term    = list[k].term;
                       list[0].counter = list[k].counter;
                       list[0].lenLeft = list[k].lenLeft;
                       list[k].term    = NULL;
                    }
                    for (k = 1; k <= i; k++)
                    {
                       BTreeFree(list[k].term);
                       list[k].term = NULL;
                    }

                    list[1].term = BTreeCopy(p);
                    list[1].counter = counter;
                    list[1].lenLeft = lenLeft;

                    i = 1;

                    searchDPhase = 0;

                    /* p must point to possible DD_0*/
                 /*   BTreeFree(p);
                    p = BTreeCopy(list[0].term); */

#ifdef DEBUG
                    printf("searchDPhase Finished: ");
                    printf(" possible DD_0 = ");
                    SprintInfixSyn(list[0].term);
                    printf(" , possible DD_1 = ");
                    SprintInfixSyn(list[1].term);
                    printf(" , at reduction %d\n", counter);
#endif
#ifdef DEBUG
```

```
                printf("Next possible D_n at list[%d] = ",
                       i);
                SprintInfixSyn(p->right);
                printf(", reduction is %d\n", counter);
#endif

             }
          }
       }
       else /* search D_{n+1} = f(D_n) phase */
       {
          if (lenLeft < list[0].lenLeft)
          {
             int backcounter, backlenLeft;
             BTreePtr backp, backpnext;
#ifdef DEBUG
             printf("Setting first: D = ");
             SprintInfixSyn(p->left);
             printf(", Dstart = ");
             SprintInfixSyn(p->right);
             printf("\n");
#endif
             backcounter = list[0].counter;

             while (backcounter < list[1].counter)
             {
                backcounter++;
                SreduceLeftmost1(backp);
                backpnext = SminNoNF(backp);
                BTreeFree(backp);
                backp = backpnext;
                backlenLeft = Slength(backp->left);
                if ((lenLeft == backlenLeft) &&
                    (SisEqual(backp->left, p->left)))
                   /* short circuit evaluation */
                {
                   BTreeFree(list[0].term);
                   list[0].term    = BTreeCopy(backp);
                   list[0].counter = backcounter;
                   list[0].lenLeft = backlenLeft;
                   BTreeFree(list[1].term);
```

```
                    list[1].term    = BTreeCopy(p);
                    list[1].counter = counter;
                    list[1].lenLeft = lenLeft;
#ifdef DEBUG
                    printf("NEW!: ");
                    printf(" possible DD_0 = ");
                    SprintInfixSyn(list[0].term);
                    printf(" , possible DD_1 = ");
                    SprintInfixSyn(list[1].term);
                    printf(" , at reduction %d\n",
                           counter);
#endif
#ifdef DEBUG
                    printf("Next possible D_n "
                           "at list[%d] = ",
                           i);
                    SprintInfixSyn(p->right);
                    printf(", reduction is %d\n",
                           counter);
#endif
                 }
              }
           }
           else if (SisEqual(p->left, list[0].term->left))
           {
              i = i + 1;
#ifdef DEBUG
              printf("Next possible D_n at list[%d] = ",
                     i);
              SprintInfixSyn(p->right);
              printf(", reduction is %d\n", counter);
#endif

              SlistNext(p, counter);
              if (i >= 2)
              {
                 for (k = 1; k <= i / 2; k++)
                 {
                    assert(i-2*k >= 0);
                    if (SisSameFunction(
                         list[i].term->right,
```

```
                        list[i-k].term->right,
                        list[i-k].term->right,
                        list[i-2*k].term->right))
                    {
                        /* setting D, D_0, D_1! */
#ifdef DEBUG
                        printf("\n***********\n");
                        printf("D = ");
                        SprintInfixSyn(list[0].term->left);
                        printf(" , counter = %d\n",
                               list[0].counter);
                        printf("D_0 = ");
                        SprintInfixSyn(
                          list[i-2*k].term->right);
                        printf(" , counter = %d\n",
                               list[i-2*k].counter);
                        printf("D_1 = ");
                        SprintInfixSyn(
                          list[i-k].term->right);
                        printf(" , counter = %d\n",
                               list[i-k].counter);
                        printf("D_2 = ");
                        SprintInfixSyn(list[i].term->right);
                        printf(" , counter = %d\n",
                               list[i].counter);
                        printf("***********\n");
#endif
                        *D   = BTreeCopy(
                                 list[0].term->left);
                        *D_0 = BTreeCopy(
                                 list[i-2*k].term->right);
                        *D_1 = BTreeCopy(
                                 list[i-k].term->right);
                        SfreeChain();
                        return 0;
                    }
                }
            }
        }
    }
```

```
   }

   SfreeChain();
   return 1;
}


/* the algorithm below does not work in most cases */
int SchainFindFirstRecurring(
  BTreePtr t, int maxcount,
  BTreePtr *D, BTreePtr *D_0, BTreePtr *D_1)
{
   BTreePtr p, pnext;
   int counter = 0;
   int lenPossibleD;
   int lenLeft;
   int lenMinD;
   int lenPrevD;
   int i, k;
   int notfound, searchDPhase;

   struct {
     BTreePtr term;
     int      counter;
     int      lenLeft;
   } list[LENLIST];


   *D = *D_0 = *D_1 = NULL;

   if (SisNormalizing(t))
   {
      return 2;
   }

   assert(maxcount > 0);
   assert(maxcount < 10000);

   for (k = 0; k < LENLIST; k++)
      list[k].term = NULL;
```

```
   i = 0;
   p = BTreeCopy(t);
   pnext = SminNoNF(p);
   BTreeFree(p);
   p = pnext;

   searchDPhase = 1;
   if (SinSMM(p->left))
   {
#ifdef DEBUG
      printf("Setting first element from start: D = ");
      SprintInfixSyn(p->left);
      printf(", Dstart = ");
      SprintInfixSyn(p->right);
      printf("\n");
#endif
      SlistFirst(p, counter);
   }

   while (counter < maxcount)
   {
      counter++;
      SreduceLeftmost1(p);
      pnext = SminNoNF(p);
      BTreeFree(p);
      p = pnext;
      lenLeft = Slength(p->left);

      if (SinSMM(p->left))
      {

         if (searchDPhase)
         {
            if (SisListEmpty(list))
            {
#ifdef DEBUG
               printf("Setting first: D = ");
               SprintInfixSyn(p->left);
               printf(", Dstart = ");
               SprintInfixSyn(p->right);
               printf("\n");
```

```
#endif
               SlistFirst(p, counter);
            }
            else
            {
               notfound = 1;
               for (k = i; k >= 0; k--)
               {
                  if (SisEqual(p->left, list[k].term->left))
                  {
                     notfound = 0;
                     break;
                  }
               }

               if (notfound)
               {
                  i = i + 1;
                  SlistNext(p, counter);
               }
               else
               {
                  /* found recurring possible D at pos k */
#ifdef DEBUG
                 printf("found recurring possible D at pos ");
                  printf("list[%d]\n", k);
#endif
                  if (k != 0)
                  {
                     BTreeFree(list[0].term);
                     list[0].term    = list[k].term;
                     list[0].counter = list[k].counter;
                     list[0].lenLeft = list[k].lenLeft;
                     list[k].term    = NULL;
                  }
                  for (k = 1; k <= i; k++)
                  {
                     BTreeFree(list[k].term);
                     list[k].term = NULL;
                  }
                  list[1].term    = BTreeCopy(p);
```

```
                list[1].counter = counter;
                list[1].lenLeft = lenLeft;

#ifdef DEBUG
                printf("searchDPhase Finished:");
                printf(" possible D = ");
                SprintInfixSyn(list[0].term);
                printf("\n");
#endif
                i = 1;
                searchDPhase = 0;
             }
          }
       }
       else /* search D_{n+1} = f(D_n) phase */
       {
          if (SisEqual(p->left, list[0].term->left))
          {
#ifdef DEBUG
             printf("Next possible D_n at list[%d] = ", i);
             SprintInfixSyn(p->right);
             printf("\n");
#endif
             i = i + 1;
             SlistNext(p, counter);
             if (i >= 2)
             {
                for (k = 1; k <= i / 2; k++)
                {
                   assert(i-2*k >= 0);
                   if (SisSameFunction(
                        list[i].term->right,
                        list[i-k].term->right,
                        list[i-k].term->right,
                        list[i-2*k].term->right))
                   {
                      /* setting D, D_0, D_1! */
#ifdef DEBUG
                      printf("\n***********\n");
                      printf("D = ");
                      SprintInfixSyn(list[0].term->left);
```

```
                        printf(" , counter = %d\n",
                               list[0].counter);
                        printf("D_0 = ");
                        SprintInfixSyn(
                          list[i-2*k].term->right);
                        printf(" , counter = %d\n",
                               list[i-2*k].counter);
                        printf("D_1 = ");
                        SprintInfixSyn(
                          list[i-k].term->right);
                        printf(" , counter = %d\n",
                               list[i-k].counter);
                        printf("D_2 = ");
                        SprintInfixSyn(
                          list[i].term->right);
                        printf(" , counter = %d\n",
                               list[i].counter);
                        printf("***********\n");
#endif
                        *D   = BTreeCopy(
                                 list[0].term->left);
                        *D_0 = BTreeCopy(
                                 list[i-2*k].term->right);
                        *D_1 = BTreeCopy(
                                 list[i-k].term->right);
                        SfreeChain();
                        return 0;
                    }
                }
            }
        }
    }
}

   SfreeChain();
   return 1;
}


int SchainFindBAD(
```

```
  BTreePtr t, int maxcount,
  BTreePtr *D, BTreePtr *D_0, BTreePtr *D_1)
{
   BTreePtr p, pnext, pMinD;
   int counter = 0;
   int lenPossibleD;
   int lenLeft;
   int lenMinD;
   int lenPrevD;
   int i, k;
   int goneUp;


   BTreePtr list[LENLIST];


   *D = *D_0 = *D_1 = NULL;

   if (SisNormalizing(t))
   {
      return 2;
   }

   assert(maxcount > 0);
   assert(maxcount < 10000);

   for (k = 0; k < LENLIST; k++)
      list[k] = NULL;

   i = 0;
   p = BTreeCopy(t);
   pnext = SminNoNF(p);
   BTreeFree(p);
   p = pnext;

   lenPossibleD = HUGEINT;
   lenMinD      = HUGEINT;
   lenPrevD     = HUGEINT;
   pMinD        = NULL;
   if (SinSMM(p->left))
   {
```

```
#ifdef DEBUG
      printf("Setting minD from start: D = ");
      SprintInfixSyn(p->left);
      printf(", Dstart = ");
      SprintInfixSyn(p->right);
      printf("\n");
#endif
      SsetMinD(p);
   }

   goneUp = 0;

   while (counter < maxcount)
   {
      counter++;
      SreduceLeftmost1(p);
      pnext = SminNoNF(p);
      BTreeFree(p);
      p = pnext;
      lenLeft = Slength(p->left);

      if (SinSMM(p->left) && (!goneUp))
      {
         if ((lenLeft > lenPrevD))
         {
            goneUp = 1;
#ifdef DEBUG
            printf("goneUp realized "
                   "(counter = %d)\n", counter);
#endif
         }

         lenPrevD = lenLeft;

         if (!goneUp)
         {
            if (lenLeft < lenMinD)
            {
#ifdef DEBUG
               printf("Setting minD: D = ");
               SprintInfixSyn(p->left);
```

```
               printf(", Dstart = ");
               SprintInfixSyn(p->right);
               printf("\n");
#endif
               SsetMinD(p);
            }

            continue;
         }
      }

      /* this follows if goneUp is true */

      if ((lenLeft < lenPossibleD) && (SinSMM(p->left)))
      {
         /* new D, start from scratch doing D0, D1 etc. */
         if (SisEqual(p->left, pMinD))
         {
#ifdef DEBUG
            printf("Setting: D = ");
            SprintInfixSyn(pMinD->left);
            printf(", Dstart = ");
            SprintInfixSyn(pMinD->right);
            printf("\n");
#endif
            Sresetlist(pMinD);

            i = 1;
#ifdef DEBUG
            printf("Next possible D_n at list[%d] = ", i);
            SprintInfixSyn(p->right);
            printf("\n");
#endif
            list[i] = BTreeCopy(p);
         }
         else
         {
#ifdef DEBUG
            printf("Setting: D = ");
            SprintInfixSyn(p->left);
            printf(", Dstart = ");
```

```
            SprintInfixSyn(p->right);
            printf("\n");
#endif
            Sresetlist(p);
         }
      }
      else if ((lenLeft == lenPossibleD) && (SinSMM(p->left)))
      {
         /* the following assertion is bad ... */
         /*assert(SisEqual(p->left, possibleD));*/
         if (SisEqual(p->left, list[0]->left))
         {
            i = i + 1;
#ifdef DEBUG
            printf("Next possible D_n at list[%d] = ", i);
            SprintInfixSyn(p->right);
            printf("\n");
#endif
            list[i] = BTreeCopy(p);
            if (i >= 2)
            {
               for (k = 1; k <= i / 2; k++)
               {
                  assert(i-2*k >= 0);
#ifdef DEBUG
          /*      printf("checkiN' ", i);
                  SprintInfixSyn(p->right);
                  printf("\n"); */
#endif
                  if (SisSameFunction(
                        list[i]->right, list[i-k]->right,
                        list[i-k]->right, list[i-2*k]->right))
                  {
                     /* setting D, D_0, D_1! */
                     *D   = BTreeCopy(list[0]->left);
                     *D_0 = BTreeCopy(list[i-2*k]->right);
                     *D_1 = BTreeCopy(list[i-k]->right);
                     freeAll();
                     return 0;
                  }
               }
```

```
            }
         }
      }
   }

   freeAll();
   return 1;
}

#endif  /* schain.h */
```

## Α.5 Πρόγραμμα fineprod

Ο πηγαίος κώδικας του προγράμματος (fineprod.c) είναι :

```
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include "sterm.h"

int main(int argc, char *argv[])
{
   int length;

   if (argc != 2)  {
      printf("usage: %s length\n", argv[0]);
      return 1;
   }
   length = atoi(argv[1]);
   generateSterms(length);
   return 0;
}
```

Το πρόγραμμα εμφανίζει όλους τους $S$-όρους μήκους όσο το πρώτο όρισμα στην γραμμή εντολών, σε λεξικογραφική διάταξη και με ένα semicolon στο τέλος του καθενός. Το εκτελέσιμο αρχείο αναφέρεται και ως fine. Για παράδειγμα :

```
> ./fine 4
oooSSSS ;
```

```
ooSoSSS ;
ooSSoSS ;
oSooSSS ;
oSoSoSS ;
```

## A.6 Πρόγραμμα printh

Το πρόγραμμα αυτό δέχεται ως ορίσματα το μήκος και τον αύξοντα αριθμό ενός $S$-όρου, βάσει της κωδικοποίησης που δίνεται στην ενότητα 11.6, και δίνει τον όρο σε προθεματική αναπαράσταση.

Ακολουθεί ο πηγαίος κώδικας (printh.c) :

```
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include "sterm.h"

int main(int argc, char *argv[])
{
   int length, termNo, i;
   char *s;

   if (argc != 3)  {
      printf("usage: %s length term_no\n", argv[0]);
      return 1;
   }

   length = atoi(argv[1]);
   assert(length > 0);
   assert(length < 100);

   termNo = atoi(argv[2]);
   assert(termNo >= 1);

   s = firstSterm(length);
   /* now s is first S-Term of length length */

   for (i = 1; i < termNo; i++)
   {
      nextSterm(s, length);
      if (isLastSterm(s, length)) break;
```

```
   }

   assert(s != NULL);
   printf("%s", s);
   return 0;
}
```

Ένα παράδειγμα κλήσης είναι :

```
> ./printh 5 6
ooSoSoSSS
```

## Α.7 Πρόγραμμα nextsterm

Το πρόγραμμα αυτό δέχεται ως όρισμα έναν όρο σε προθεματική αναπαράσταση και δίνει στην έξοδο τον επόμενο όρο στην διάταξη που ορίζεται στην ενότητα 11.6.

Ο πηγαίος κώδικας βρίσκεται στο αρχείο nextsterm.c :

```
#include <stdio.h>
#include <assert.h>
#include <string.h>
#include "sterm.h"

int main(int argc, char *argv[])
{
   int length;
   char s[1000];

   if (argc != 2)  {
      printf("usage: %s term\n", argv[0]);
      return 1;
   }

   assert((strlen(argv[1]) % 2) == 1);
   length = ( strlen(argv[1]) + 1 ) / 2 ;

   /* isStermPrefix(argv[1]); */

   strcpy(s, argv[1]);
   nextSterm(s, length);
```

```
   printf("%s\n", s);

   return 0;
}
```

Για να βρούμε τον επόμενο όρο του $SS(SS)$ δίνουμε :

```
> ./nextsterm ooSSoSS
oSooSSS
```

## A.8 Πρόγραμμα topred

Το πρόγραμμα αυτό εκτελεί μία αναγωγή στην κεφαλή του $S$-όρου που δίνεται στην πρότυπη είσοδο και δίνει στην έξοδο το αποτέλεσμα της αναγωγής (οι όροι σε προθεματική αναπαράσταση). Ο πηγαίος κώδικας (topred.c) είναι :

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "bintree.h"
#include "sterm.h"

int main(void)
{
   BTreePtr p;

   p = SreadInfix();
   if (ScanReduce(p))
      Sreduce1(p);
   if (p->label == NONORMAL)
      printf(" ");
   else
      SprintPrefix(p);

   return 0;
}
```

Ας κάνουμε μία αναγωγή του όρου $SS(SS)$ :

```
> echo "oooSSSS" | ./topred
oo S So S S
```

## Α.9 Πρόγραμμα fullred

Αυτό το πρόγραμμα εκτελεί συνεχώς αναγωγές στον όρο που δίνεται στην είσοδο, μέχρι να φτάσει (αν ποτέ συμβεί αυτό) σε κανονική μορφή (ο όρος πρέπει να δίνεται σε προθεματική αναπαράσταση). Ο πηγαίος κώδικας (fullred.c) είναι :

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "bintree.h"
#define GLOBREDCOUNT
#define DEBUGREDCOUNT
#include "sterm.h"

int main(void)
{
   BTreePtr p;

   p = SreadInfix();
   printf("input: "); SprintPrefix(p); printf("\n");

   Sredcount = 0;
   (void) REDUCESTRATEGY(p);
   if (p->label == NONORMAL)
   {
      printf("non-normalizing!\n");
      printf("reductions_made: %d\n", Sredcount);
      return -1;
   }
   else
   {
      printf("NF: "); SprintPrefix(p); printf("\n");
      printf("length: %d    reductions: %d\n",
             Slength(p), Sredcount);
   }

   return 0;
}
```

Αν δώσουμε ως είσοδο τον όρο $BSSS$ ($ooooSoSSSSS$ σε προθεματική αναπαράσταση), λαμβάνουμε :

```
> echo ooooSoSSSSS  | fullred
input: oooo So S S S S S
Sreduce1: reduction number:          1
Sreduce1: reduction number:          2
Sreduce1: reduction number:          3
Sreduce1: reduction number:          4
NF: oo Soo So S S So Soo So S S S
length: 10    reductions: 4
```

Αν όμως δώσουμε τον όρο $BSSSS$ ($ooooSoSSSSSS$ σε προθεματική αναπαράσταση, που ως γνωστόν δεν έχει κανονική μορφή, το πρόγραμμα δεν ολοκληρώνεται (ή μάλλον σταματάει όταν ο υπό επεξεργασία όρος γίνει τόσο μεγάλος που να μην χωράει στην μνήμη) :

```
> echo oooooSoSSSSSS | fullred
input: ooooo So S S S S S S
Sreduce1: reduction number:          1
Sreduce1: reduction number:          2
Sreduce1: reduction number:          3
Sreduce1: reduction number:          4
Sreduce1: reduction number:          5
Sreduce1: reduction number:          6
Sreduce1: reduction number:          7
Sreduce1: reduction number:          8
Sreduce1: reduction number:          9
...
```

## A.10 Πρόγραμμα lize (κανονικοποίηση)

Αυτό το πρόγραμμα αποφαίνεται αν ο όρος που δίνεται στην πρότυπη είσοδο (σε προθεματική αναπαράσταση) είναι κανονικοποιήσιμος ή όχι και αν είναι δίνει και την κανονική του μορφή. Ο πηγαίος κώδικας (lize.c) είναι :

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "bintree.h"
#define GLOBREDCOUNT
#define DEBUGREDCOUNT
#define DEBUG
#define DEBUGX
```

```
#include "sterm.h"

int main(void)
{
   BTreePtr p;

   p = SreadInfix();
   #ifdef GLOBREDCOUNT
   Sredcount = 0;
   #endif
   Snormalize(p);
   if (p->label == NONORMAL)
   {
      printf("no NF!");
      return -1;
   }
   else
   {
      SprintPrefix(p);
   }

   return 0;
}
```

Δίνοντας όπως παραπάνω είσοδο $BSSS$ (σε προθεματική αναπαράσταση), λαμβάνουμε τελικά την κανονική μορφή :

```
> echo ooooSoSSSSS | lize
Sreduce1: reduction number:          1
Sreduce1: reduction number:          2
Sreduce1: reduction number:          3
Sreduce1: reduction number:          4
oo Soo So S S So Soo So S S S
```

Αν όμως δώσουμε $BSSSS$ που δεν έχει κανονική μορφή λαμβάνουμε (μετά από μερικές αναγωγές) :

```
> echo oooooSoSSSSSS | lize
Sreduce1: reduction number:          1
Sreduce1: reduction number:          2
Sreduce1: reduction number:          3
Sreduce1: reduction number:          4
```

```
Sreduce1: reduction number:          5
Sreduce1: reduction number:          6
Sreduce1: reduction number:          7
no NF!
```

## A.11 Προγράμματα εύρεσης μεγίστων

Τα δύο προγράμματα που ακολουθούν βρίσκουν το μέγιστο μήκος κανονικών μορφών και το μέγιστο πλήθος βημάτων αναγωγής μεταξύ των κανονικοποιήσιμων όρων του ιδίου μήκους (βλέπε ενότητα 13.8, στην σελίδα 149, για περισσότερες λεπτομέρειες).

### Πρόγραμμα maxred

Το πρόγραμμα χρησιμοποιείται για την συμπλήρωση του πίνακα 13.2 στην σελίδα 149. Ο πηγαίος κώδικας βρίσκεται στο αρχείο maxred.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "bintree.h"

#define GLOBREDCOUNT

#include "sterm.h"
#include "string.h"

void readsemi(void);

int main(void)
{
   BTreePtr p;
   int total, countnoNF, maxred;

   #define STRINGLEN 10050
   #define CHECKLEN  10000

   char maxterms[STRINGLEN];

   total = 0;
   countnoNF = 0;
```

```
   maxred = 0;

   p = SreadInfix();
   readsemi();
   while (p != NULL)
   {
      total++;
      Sredcount = 0;
      Snormalize(p);
      if (p->label == NONORMAL)
      {
         countnoNF++;
      }
      else
      {
         if (Sredcount > maxred)
         {
            maxred = Sredcount;
            sprintf(maxterms, "%d", total);

         }
         else if (Sredcount == maxred)
         {
            assert(strlen(maxterms) <= CHECKLEN);
           sprintf(&maxterms[strlen(maxterms)], " %d", total);
         }
      }
      BTreeFree(p);
      p = SreadInfix();
      if (p == NULL) break;
      readsemi();
   }

   printf("maxreds= %d,  maxterms = %s\n", maxred, maxterms);

   return 0;
}


void readsemi(void)
{
```

```
   int c;
   while((c = getchar()) != EOF)
   {
      if ((c == ' ') || (c == '\n') || (c == '\t'))
      {
         continue;
      }
      if (c == ';')
      {
         break;
      }
   }
}
```

## Πρόγραμμα maxlen

Το πρόγραμμα χρησιμοποιείται για την συμπλήρωση του πίνακα 13.3 στην σελίδα 150. Ο πηγαίος κώδικας βρίσκεται στο αρχείο maxlen.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "bintree.h"
#include "sterm.h"
#include "string.h"

void readsemi(void);

int main(void)
{
   BTreePtr p;
   int total, countnoNF, maxlen, curlength;

   #define STRINGLEN 10050
   #define CHECKLEN  10000

   char maxterms[STRINGLEN];
   total = 0;
   countnoNF = 0;
   maxlen = 0;
```

```
   p = SreadInfix();
   readsemi();
   while (p != NULL)
   {
      total++;
      Snormalize(p);
      if (p->label == NONORMAL)
      {
         countnoNF++;
      }
      else
      {
         curlength = Slength(p);
         if (curlength > maxlen)
         {
            maxlen = curlength;
            sprintf(maxterms, "%d", total);

         }
         else if (curlength == maxlen)
         {
            assert(strlen(maxterms) <= CHECKLEN);
           sprintf(&maxterms[strlen(maxterms)], " %d", total);
         }
      }
      BTreeFree(p);
      p = SreadInfix();
      if (p == NULL) break;
      readsemi();
   }

   printf("maxlen= %d,   terms: %s\n", maxlen, maxterms);
   return 0;
}

void readsemi(void)
{
   int c;
   while((c = getchar()) != EOF)
   {
      if ((c == ' ') || (c == '\n') || (c == '\t'))
```

```
      {
         continue;
      }
      if (c == ';')
      {
         break;
      }
   }
}
```

# Παράρτημα Β

# Τεχνολόγηση και μετατροπή με eli για $S$-όρους

Η μελέτη των $S$-όρων απαιτεί συχνά την μετατροπή μεταξύ των διαφόρων αναπαραστάσεων (προθεματική, απλή ενθεματική, ενθεματική με τις συντομογραφίες για τους όρους $SSS = A$ και $S(SS) = B$, γραφικές αναπαραστάσεις ως δένδρα). Όπως έχουμε πει, κάθε αναπαράσταση έχει ανάλογα με την περίπτωση την χρησιμότητά της. Για παράδειγμα, τα περισσότερα προγράμματα του προηγούμενου παραρτήματος ενεργούν επί όρων σε προθεματική αναπαράσταση. Όμως, ο άνθρωπος γενικά προτιμά να διαβάζει όρους σε ενθεματική αναπαράσταση και μάλιστα με τις συντομογραφίες $A$, $B$. Τέλος, για μεγάλους σε μήκος $S$-όρους, η δενδρική αναπαράσταση επιτρέπει μία γενικότερη εποπτεία και αντίληψη, αφού ο ίδιος όρος απλώνεται στις δύο διαστάσεις σε αντίθεση με την μονοδιάστατη αναπαράσταση της συμβολοσειράς.

Επιπλέον, χρειάζεται πληροφορίες για έναν $S$-όρο, όπως το μήκος αυτού, να προκύπτουν από τις διάφορες αναπαραστάσεις αυτού και αυτό επιτυγχάνεται με τεχνολόγηση (parsing).

## Β.1 Το χρησιμοποιηθέν σύστημα : eli

Στόχος είναι να κατασκευαστούν προγράμματα τα οποία να πραγματοποιούν τις μετατροπές μεταξύ των παραπάνω αναπαραστάσεων και να υπολογίζουν διάφορες πληροφορίες για $S$-όρους.

Δεδομένου ότι διαδικασίες που σχετίζονται με «μετάφραση» και τεχνολόγηση είναι αρκετά συνηθισμένες στον κόσμο των υπολογιστών (βλέπε για παράδειγμα τους μεταγλωττιστές γλωσσών προγραμματισμού), προγράμματα όπως αυτά που θέλουμε να κατασκευάσουμε συνήθως δεν προγραμματίζονται σε μία συνηθισμένη γλώσσα προγραμματισμού (χαμηλό επίπεδο), αλλά

χρησιμοποιούνται εργαλεία κάπως υψηλοτέρου επιπέδου, οι λεγόμενοι μετα-μεταγλωττιστές (meta-compilers). Συγκεκριμένα, οι μετα-μεταγλωττιστές είναι εργαλεία που λαμβάνουν ως είσοδο μία περιγραφή (σε μία δικιά τους γλώσσα υψηλότερου επιπέδου) της διαδικασίας μετάφρασης και τεχνολόγησης και δίνουν ως έξοδο ένα πρόγραμμα που επιτελεί την προδιαγραφείσα μετάφραση και τεχνολόγηση.

Ένα τέτοιο εργαλείο είναι και το eli (βλέπε [35]). Με τα ευρέως διαδεδομένα εργαλεία lex και yacc (ή τις εξελίξεις τους flex και bison αντιστοίχως), μπορούν να περιγραφούν δηλωτικά (declarative specification) με την βοήθεια κανονικών παραστάσεων (regular expressions) και γραμματικών χωρίς συμφραζόμενα (context-free grammars) αντιστοίχως ο λεκτικός αναλυτής και ο τεχνολογητής. Για τα υπόλοιπα στάδια της διαδικασίας μετάφρασης πρέπει να γραφεί κώδικας, ή αλλιώς τα στάδια αυτά να περιγραφούν λειτουργικά (operational specification). Το eli πάει ένα βήμα παραπέρα, αφού για αρκετά από τα υπόλοιπα αυτά στάδια, παρέχει εργαλεία με τα οποία μπορούν οι αντίστοιχες λειτουργίες να περιγραφούν επίσης δηλωτικά. Φυσικά, για τις πολύ εξειδικευμένες λειτουργίες οι οποίες ενδεχομένως απαιτούνται από ένα πρόγραμμα μετάφρασης, το eli μας επιτρέπει να συμπεριλάβουμε και κώδικα χαμηλού επιπέδου που έχουμε γράψει.

Στο eli, για κάθε πρόγραμμα τεχνολόγησης ή μετάφρασης που θέλουμε να φτιάξουμε, δημιουργούμε ένα κατάλληλο αρχείο γενικών προδιαγραφών με κατάληξη .specs. Σε αυτό περιλαμβάνονται τα ονόματα όλων των αρχείων με τις προδιαγραφές των επιμέρους τμημάτων του προγράμματος μετάφρασης. Δίνουμε παρακάτω τους τύπους αρχείων που χρησιμοποιήσαμε και οι οποίοι ξεχωρίζουν βάσει της κατάληξής τους :

- .gla για λεκτικό αναλυτή (για περισσότερες λεπτομέρειες βλέπε [36, Lexical analysis])
- .con για τεχνολογητή (γραμματική χωρίς συμφραζόμενα) (για περισσότερες λεπτομέρειες βλέπε [36, Syntax analysis])
- .head για βοηθητικές δηλώσεις τύπων C που ενδεχομένως χρειάζονται
- .lido για υπολογισμούς επί του συντακτικού δένδρου (για περισσότερες λεπτομέρειες βλέπε [36, LIDO, Computation in trees])
- .ptg για έξοδο κειμένου (για περισσότερες λεπτομέρειες βλέπε [36, PTG Pattern-based Text Generator])

Πρακτικά, σε ένα πρόγραμμα μετάφρασης πρώτα περιγράφουμε τον λεκτικό αναλυτή (.gla), κατόπιν τον τεχνολογητή (.con), ο οποίος κατασκευάζει ένα δένδρο, στο οποίο ενδεχομένως γίνεται επεξεργασία (.lido) και βάσει του οποίου παρέχεται κάποια έξοδος (.ptg).

Για την διαχείριση των αλλαγών στις προδιαγραφές των τμημάτων, μετά από διαδοχικές δημιουργίες του εκάστοτε προγράμματος μετάφρασης, το eli

χρησιμοποιεί το σύστημα Odin (βλέπε [78]) που διαχειρίζεται τα ενδιάμεσα αποτελέσματα των σταδίων κατασκευής του προγράμματος μετάφρασης. Ουσιαστικά, το Odin έχει παρόμοια λειτουργικότητα με το make των συστημάτων Unix, αλλά διαφορετική προσέγγιση στην επίλυση του προβλήματος. Όπως, αναφέρεται και στο [36, eli user interface reference manual] :

> Eli focuses a user's attention on the information required to specify a text processor, rather than the tools implementing it, by automating the process of tool invocation. Desired products, such as an executable program or the result of a test run, are regarded as derived objects. Eli responds to a request for a derived object by invoking the minimum number of tools necessary to produce that object. Derived objects are automatically stored for re-use in future derivations, thereby significantly shortening the time required to satisfy requests.

**Β.1.1 Παράδειγμα.** Για να τρέξουμε το σύστημα eli δίνουμε στην γραμμή εντολών του Unix :

```
> eli
```

οπότε εμφανίζεται το prompt του eli :

```
Eli Version 4.3.1 (? for help, ^D to exit)
->
```

και αναμένει την είσοδο του χρήστη.

Ας υποθέσουμε ότι έχουμε το αρχείο προδιαγραφών pre2in.specs και θέλουμε να κατασκευάσουμε το εκτελέσιμο πρόγραμμα μετάφρασης pre2in.exec. Τότε, δίνουμε :

```
-> pre2in.specs :exe > pre2in.exec
```

Μπορούμε αντί για εκτελέσιμο πρόγραμμα να κατασκευάσουμε κώδικα C για το πρόγραμμα μετάφρασης ως εξής :

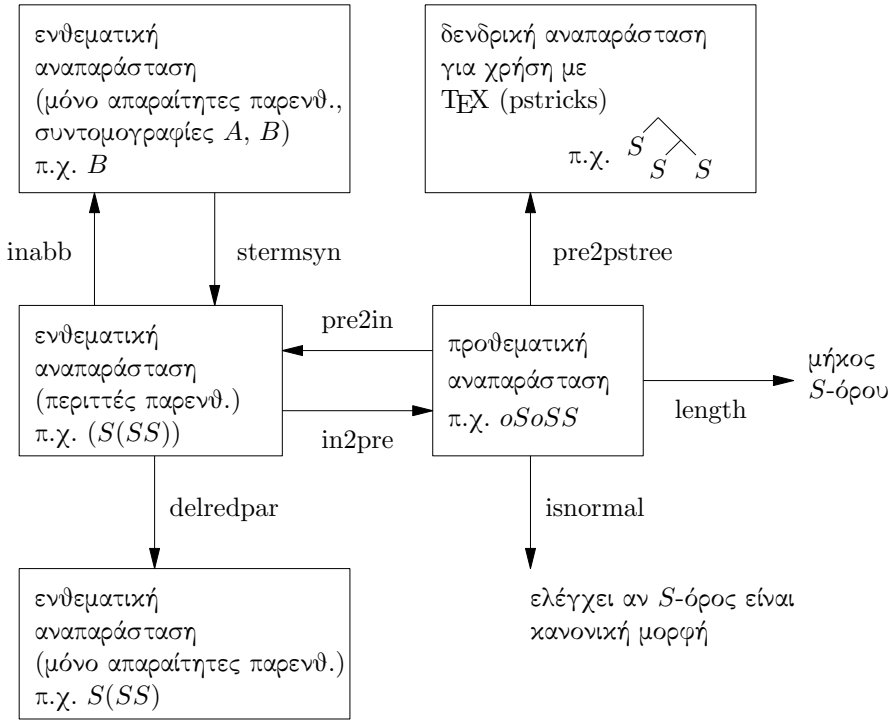
```
-> pre2in.specs :source > pre2insrc/
```

Το παραπάνω θα τοποθετήσει τα αρχεία κώδικα (μαζί με το αντίστοιχο Makefile) στον κατάλογο (directory) `pre2insrc`.

Για περισσότερες πληροφορίες σχετικά με τις δυνατές εντολές, βλέπε [36, eli user interface reference manual].

## Β.2 Οι μετατροπείς και τεχνολογητές συγκεντρωτικά

Στο διάγραμμα του σχήματος Β.1 φαίνεται πώς ενεργούν τα προγράμματα eli που περιμένουν ως είσοδο έναν $S$-όρο σε διάφορες αναπαραστάσεις και πώς γίνονται οι μετατροπές μεταξύ των αναπαραστάσεων ή τι πληροφορίες δίνονται για κάθε $S$-όρο.

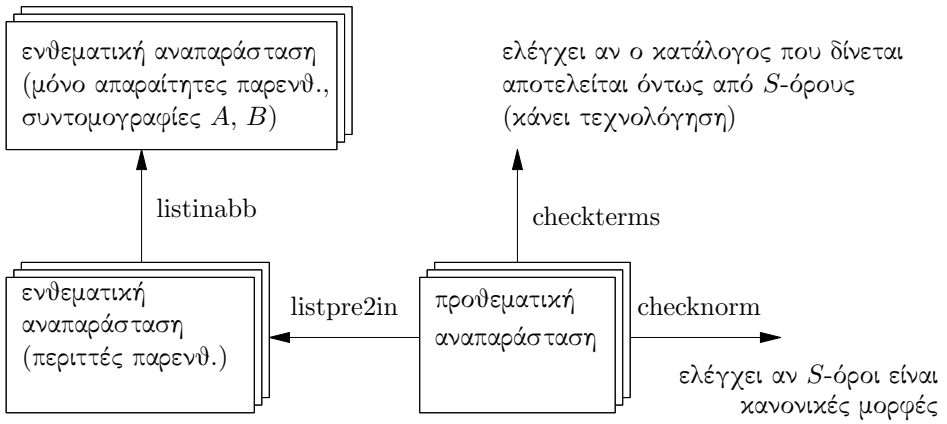



Σχήμα Β.1 – Προγράμματα eli που ενεργούν επί ενός $S$-όρου

Πέραν των προγραμμάτων που επενεργούν επί ενός $S$-όρου, χρειαζόμαστε και προγράμματα που επεξεργάζονται καταλόγους $S$-όρων. Οι κατάλογοι αυτοί είναι παρόμοιοι με αυτούς που βγάζει το πρόγραμμα fineprod (βλέπε ενότητα Α.5, στην σελίδα 222) : Κάθε $S$-όρος ακολουθείται από ένα semicolon, προκειμένου να έχουμε σαφή διαχωρισμό από τον επόμενο $S$-όρο στον κατάλογο.

Στο διάγραμμα του σχήματος Β.2 φαίνονται τα αντίστοιχα προγράμματα που δέχονται ως είσοδο έναν κατάλογο $S$-όρων.

Για κάθε πρόγραμμα προτιθέμεθα να δώσουμε το αρχικό αρχείο προδιαγραφής που περιέχει τα επιμέρους αρχεία προδιαγραφής (για τον λεκτικό αναλυτή, την γραμματική κτλ.). Όμως, πολλά προγράμματα έχουν κοινό κάποιο




Σχήμα B.2 – Προγράμματα eli που ενεργούν επί καταλόγου $S$-όρων

επιμέρους τμήμα τους (για παράδειγμα, όλα τα προγράμματα που δέχονται ως είσοδο όρους σε προθεματική αναπαράσταση έχουν τον ίδιο λεκτικό αναλυτή ή όλα τα προγράμματα που επενεργούν επί καταλόγου $S$-όρων έχουν κοινό το τμήμα τεχνολόγησης που διαχωρίζει τους $S$-όρους). Πριν δώσουμε, λοιπόν, τις προδιαγραφές των προγραμμάτων, θα δώσουμε κάποιες κοινές προδιαγραφές : λεκτικούς αναλυτές, γραμματικές και την ptgfunc.head. Κατόπιν για κάθε πρόγραμμα εκτός από την συνολική προδιαγραφή του θα δίνουμε και τις προδιαγραφές των ιδιαιτέρων επιμέρους τμημάτων του.

## B.3 Προδιαγραφές λεκτικών αναλυτών

Έχουμε έναν λεκτικό αναλυτή για κάθε δυνατή αναπαράσταση εισόδου ενός $S$-όρου. Οι κατάλογοι $S$-όρων αντιμετωπίζονται σε επίπεδο τεχνολόγησης και όχι λεκτικής ανάλυσης (βλέπε ενότητα B.14).

### Προθεματική αναπαράσταση : prefix.gla

Οι μόνες λεκτικές μονάδες στην περίπτωση της προθεματικής αναπαράστασης είναι τα σύμβολο για τον συνδυαστή ($S$) και για τον τελεστή εφαρμογής ($o$) :

```
Sconst:         $"S"
Application:    $"o"
```

### Απλή ενθεματική αναπαράσταση : sterm.gla

Στην ενθεματική αναπαράσταση δεν έχουμε πλέον τον τελεστή εφαρμογής, αλλά έχουμε τις παρενθέσεις ως λεκτικές μονάδες :

```
Sconst:         $"S"

ParOpen:        $"("
ParClose:       $")"
```

### Αναπαράσταση με συντομογραφίες : stermsyn.gla

Εδώ έχουμε επιπλέον λεκτικές μονάδες για τις συντομογραφίες $A$ και $B$ ($A = SSS$ και $B = S(SS)$ αντιστοίχως) :

```
Sconst:         $"S"
Aconst:         $"A"
Bconst:         $"B"

ParOpen:        $"("
ParClose:       $")"
```

## Β.4 Προδιαγραφές γραμματικών για τεχνολόγηση

Εδώ δίνονται γραμματικές χωρίς συμφραζόμενα για την τεχνολόγηση. Οι γραμματικές είναι σε LALR(1) μορφή, όπως απαιτείται από το eli.

### Προθεματική αναπαράσταση : prefix.con

```
/* S-Term grammar, prefix notation */

Axiom:
   PrefixTerm .

PrefixTerm:
   Sconst /
   Application PrefixTerm PrefixTerm .
```

### Απλή ενθεματική αναπαράσταση : sterm.con

```
/* normal S-Term grammar,
   infix (application not shown) notation,
   parentheses override left association */

Axiom:
   STerm .

STerm:
   STerm ParOpen STerm ParClose /  /* override left assoc. */
   STerm Sconst /                  /* left association */
   ParOpen STerm ParClose /        /* redundant parenthesis */
   Sconst .
```

### Αναπαράσταση με συντομογραφίες : stermsyn.con

```
/* normal S-Term grammar,
   infix (application not shown) notation,
   parentheses override left association
   Abkurzungen:    A = SSS      B = S(SS)
*/

Axiom:
   STerm .

STerm:
   STerm ParOpen STerm ParClose /  /* override left assoc. */
   STerm Const /                  /* left association */
   ParOpen STerm ParClose /        /* redundant parenthesis */
   Const .

Const:
   Sconst / Aconst / Bconst .
```

## B.5 Προδιαγραφή ptgfunc.head

Σε αυτό το αρχείο ορίζουμε επώνυμα έναν τύπο C που χρησιμοποιούμε συχνά κατά την έξοδο της διαδικασίας μετάφρασης, ούτως ώστε να αναφερόμαστε πιο σύντομα σε αυτόν :

```
#include "ptg_gen.h"
typedef PTGNode (*PTGFunc)();
```

## Β.6 Πρόγραμμα pre2in

Αυτό το πρόγραμμα μετατρέπει έναν $S$-όρο από προθεματική σε ενθεματική αναπαράσταση. Ο όρος που δίνει ως έξοδο το πρόγραμμα πιθανόν να έχει περιττές παρενθέσεις.

### Προδιαγραφή pre2in.specs

```
prefix.gla /* lexical analysis*/
prefix.con  /* grammar of terms */

$/Output/LeafPtg.fw
$/Output/PtgCommon.fw

/* prefix to infix notation conversion */
ptgfunc.head
pre2in.lido
pre2in.ptg

/* beware! redundant parentheses are created! */
/* use delredpar to eliminate them */
```

### pre2in.lido

```
/* S-Term prefix to infix notation conversion */

ATTR Infix: PTGNode;

SYMBOL Axiom
COMPUTE
   PTGOut(THIS.Infix);
END;

RULE: Axiom ::= PrefixTerm
COMPUTE
   Axiom.Infix = PrefixTerm.Infix;
END;
```

```
RULE: PrefixTerm ::= Application PrefixTerm PrefixTerm
COMPUTE
   PrefixTerm[1].Infix =
      PTGParenthesize(
         PTGInfixApply(PrefixTerm[2].Infix,
                       PrefixTerm[3].Infix));
END;

RULE: PrefixTerm ::= Sconst
COMPUTE
   PrefixTerm.Infix = PTGSconst();
END;
```

### pre2in.ptg

```
Sconst:        "S "
InfixApply:    $ $
Parenthesize:  "(" $ ") "
```

Παράδειγμα χρήσης :

```
> echo ooSSS | ./pre2in
((S S ) S )
```

## B.7 Πρόγραμμα in2pre

Αυτό το πρόγραμμα μετατρέπει έναν $S$-όρο από ενθεματική σε προθεματική αναπαράσταση. (Είναι αντίστροφο του pre2in.)

### Προδιαγραφή in2pre.specs

```
sterm.gla
sterm.con

$/Output/LeafPtg.fw
$/Output/PtgCommon.fw
/* infix to prefix notation conversion */
ptgfunc.head
in2pre.lido
in2pre.ptg
```

## in2pre.lido

```
/* S-Term infix to prefix notation conversion */

ATTR Prefix: PTGNode;

SYMBOL Axiom
COMPUTE
   PTGOut(THIS.Prefix);
END;

RULE: Axiom ::= STerm
COMPUTE
   Axiom.Prefix = STerm.Prefix;
END;

RULE: STerm ::= STerm ParOpen STerm ParClose
COMPUTE
   STerm[1].Prefix = PTGPrefixApply(STerm[2].Prefix,
                                    STerm[3].Prefix);
END;

RULE: STerm ::= STerm Sconst
COMPUTE
   STerm[1].Prefix = PTGPrefixApply(STerm[2].Prefix,
                                    PTGSconst());
END;

RULE: STerm ::= ParOpen STerm ParClose
COMPUTE
   STerm[1].Prefix = STerm[2].Prefix;
END;

RULE: STerm ::= Sconst
COMPUTE
   STerm.Prefix = PTGSconst();
END;
```

## in2pre.ptg

```
Sconst:          " S"
```

```
PrefixApply:     "o" $ $
```

Παράδειγμα χρήσης :

```
> echo 'S(S(SS)S)' | ./in2pre
o Soo So S S S
```

## B.8 Πρόγραμμα delredpar

Αυτό το πρόγραμμα επενεργεί επί ενός $S$-όρου σε ενθεματική αναπαράσταση που πιθανώς έχει περιττές παρενθέσεις και αφαιρεί αυτές τις περιττές παρενθέσεις αφήνοντας μόνον τις απαραίτητες.

### Προδιαγραφή delredpar.specs

```
sterm.gla
sterm.con

$/Output/LeafPtg.fw
$/Output/PtgCommon.fw

/* delete redundant parentheses */
ptgfunc.head
delredpar.lido
delredpar.ptg
```

### delredpar.lido

```
/* S-Term delete redundant parentheses */

ATTR Infix: PTGNode;

ATTR IsSconst: int;

SYMBOL STerm
COMPUTE
   SYNT.IsSconst = 0;
END;

SYMBOL Axiom
COMPUTE
```

```
   PTGOut(THIS.Infix);
END;

RULE: Axiom ::= STerm
COMPUTE
   Axiom.Infix = STerm.Infix;
END;

RULE: STerm ::= STerm ParOpen STerm ParClose
COMPUTE
   STerm[1].Infix =
     PTGInfixApply(
       STerm[2].Infix,
       IF(STerm[3].IsSconst,
          STerm[3].Infix,
          PTGParenthesize(STerm[3].Infix)));
END;

RULE: STerm ::= STerm Sconst
COMPUTE
   STerm[1].Infix = PTGInfixApply(STerm[2].Infix,
                                  PTGSconst());
END;

RULE: STerm ::= ParOpen STerm ParClose
COMPUTE
   STerm[1].Infix = STerm[2].Infix;
END;

RULE: STerm ::= Sconst
COMPUTE
   STerm.Infix = PTGSconst();
   STerm.IsSconst = 1;
END;
```

## delredpar.ptg

```
Sconst:         "S "
InfixApply:     $ $
Parenthesize:   "(" $ ") "
```

Παράδειγμα χρήσης :

```
> echo '((SSS)S)(SSS)' | ./delredpar
S S S S (S S S )
```

## B.9 Πρόγραμμα inabb

Αυτό το πρόγραμμα βρίσκει σε έναν $S$-όρο σε ενθεματική αναπαράσταση τους υποόρους $SSS$ και $S(SS)$ και τους αντικαθιστά με τις συντομογραφίες $A$ και $B$ αντιστοίχως. Επιπλέον αφαιρούνται και οι περιττές παρενθέσεις.

### Προδιαγραφή inabb.specs

```
sterm.gla
sterm.con

$/Output/LeafPtg.fw
$/Output/PtgCommon.fw

/* abbreviate:   A = SSS,   B = S(SS)  */
ptgfunc.head
inabb.lido
inabb.ptg
```

### inabb.lido

```
/* S-Term delete redundant parentheses */

ATTR Infix: PTGNode;

ATTR IsSconst: int;

ATTR IsSS: int;

ATTR IsConst: int; /* yes if term one of S, A, B */

SYMBOL STerm
COMPUTE
   SYNT.IsSconst = 0;
   SYNT.IsSS     = 0;
```

```
   SYNT.IsConst  = 0;
END;

SYMBOL Axiom
COMPUTE
   PTGOut(THIS.Infix);
END;

RULE: Axiom ::= STerm
COMPUTE
   Axiom.Infix = STerm.Infix;
END;


RULE: STerm ::= STerm ParOpen STerm ParClose
COMPUTE
   /* maybe this term is SS, if yes, label it */
   STerm[1].IsSS  =
     IF(AND(STerm[2].IsSconst, STerm[3].IsSconst), 1, 0);

   /* the Sterm[1] can be B */

   STerm[1].IsConst =
     IF(AND(STerm[2].IsSconst, STerm[3].IsSS), 1, 0);

   STerm[1].Infix =
     IF(AND(STerm[2].IsSconst, STerm[3].IsSS),
        PTGBconst(),
        PTGInfixApply(
         STerm[2].Infix,
         IF(STerm[3].IsConst,
            STerm[3].Infix,
            PTGParenthesize(STerm[3].Infix))));
END;


RULE: STerm ::= STerm Sconst
COMPUTE
   /* maybe this term is SS, if yes, label it */
   STerm[1].IsSS  = IF(STerm[2].IsSconst, 1, 0);
```

```
   /* the Sterm[1] can be A */

   /* if yes it is a constant */
   STerm[1].IsConst = IF(STerm[2].IsSS, 1, 0);

   STerm[1].Infix =
     IF(STerm[2].IsSS,
        PTGAconst(),
        PTGInfixApply(STerm[2].Infix, PTGSconst()));
END;


RULE: STerm ::= ParOpen STerm ParClose
COMPUTE
   STerm[1].IsSconst = STerm[2].IsSconst;
   STerm[1].IsSS     = STerm[2].IsSS;
   STerm[1].IsConst  = STerm[2].IsConst;
   STerm[1].Infix    = STerm[2].Infix;
END;

RULE: STerm ::= Sconst
COMPUTE
   STerm.Infix = PTGSconst();
   STerm.IsSconst = 1;
   STerm.IsConst  = 1;
END;
```

## inabb.ptg

```
Sconst:         "S "
Aconst:         "A "
Bconst:         "B "
InfixApply:     $ $
Parenthesize:   "(" $ ") "
```

Παράδειγμα χρήσης :

```
> echo 'SSS(S(SS)S)' | ./inabb
A (B S )
```

## Β.10 Πρόγραμμα stermsyn

Αυτό το πρόγραμμα αναπτύσσει σε έναν $S$-όρο σε ενθεματική αναπαράσταση τα $A$ και $B$ στους υποόρους $SSS$ και $S(SS)$ αντιστοίχως. (Είναι το αντίστροφο του inabb.) Ας σημειωθεί ότι είναι σχεδόν βέβαιο ότι θα υπάρχουν περιττές παρενθέσεις στον όρο εξόδου (τοποθετούνται πάντοτε παρενθέσεις γύρω από τις αναπτύξεις των $A$ και $B$).

### Προδιαγραφή stermsyn.specs

```
stermsyn.gla
stermsyn.con

$/Output/LeafPtg.fw
$/Output/PtgCommon.fw

/* substitute Abkurzungen */
stermsyn.head
stermsyn.lido
stermsyn.ptg
```

### stermsyn.lido

```
ATTR Infix: PTGNode;

ATTR IsSconst: int;

SYMBOL STerm
COMPUTE
   SYNT.IsSconst = 0;
END;

SYMBOL Axiom
COMPUTE
   PTGOut(THIS.Infix);
END;

RULE: Axiom ::= STerm
COMPUTE
   Axiom.Infix = STerm.Infix;
END;
```

```
RULE: STerm ::= STerm ParOpen STerm ParClose
COMPUTE
   STerm[1].Infix =
     PTGInfixApply(
       STerm[2].Infix,
       IF(STerm[3].IsSconst,
          STerm[3].Infix,
          PTGParenthesize(STerm[3].Infix)));
END;

RULE: STerm ::= STerm Const
COMPUTE
   STerm[1].Infix = PTGInfixApply(STerm[2].Infix,
                                  Const.Infix);
END;

RULE: STerm ::= ParOpen STerm ParClose
COMPUTE
   STerm[1].Infix = STerm[2].Infix;
END;

RULE: STerm ::= Const
COMPUTE
   STerm.Infix = Const.Infix;
END;

RULE: Const ::= Sconst
COMPUTE
   Const.Infix = PTGSconst();
END;

RULE: Const ::= Aconst
COMPUTE
   Const.Infix = PTGAconst();
END;

RULE: Const ::= Bconst
COMPUTE
   Const.Infix = PTGBconst();
END;
```

### stermsyn.ptg

```
Sconst:          "S "
Aconst:          "(SSS) "
Bconst:          "(S(SS)) "
InfixApply:      $ $
Parenthesize:    "(" $ ") "
```

Παράδειγμα χρήσης :

```
> echo 'SABS(AS)' | ./stermsyn
S (SSS) (S(SS)) S ((SSS) S )
```

## Β.11 Πρόγραμμα pre2pstree

Αυτό το πρόγραμμα μετατρέπει έναν όρο σε προθεματική μορφή σε μία περιγραφή δενδρικής αναπαράστασης. Αυτή η περιγραφή δενδρικής αναπαράστασης είναι εντολές από το πακέτο pst-tree που βασίζεται στα λεγόμενα pstricks (βλέπε [86]), μία λύση για χρήση γραφικών PostScript σε LaTeX. Με την βοήθεια ενός πακέτου του eli (Indent) γίνεται κατάλληλη στοίχιση στις εντολές για κάθε επίπεδο του δένδρου.

### Προδιαγραφή pre2pstree.specs

```
/* pre2pstree: prefix to PStree */

/* *****
input:  an S-term in prefix notation
output: nicely indented pstree macros
        for inclusion in LaTeX code
******* */

prefix.gla /* lexical analysis */
prefix.con  /* grammar of terms */

$/Output/LeafPtg.fw
$/Output/PtgCommon.fw
$/Output/Indent.fw

/* prefix to PSTree notation conversion */
```

```
ptgfunc.head
pre2pstree.lido
pre2pstree.ptg
```

## pre2pstree.lido

```
/* S-Term prefix to PSTree notation conversion */

ATTR PSTree: PTGNode;

SYMBOL Axiom
COMPUTE
   IndentSetStep(3);
   PTGOut(THIS.PSTree);
END;

RULE: Axiom ::= PrefixTerm
COMPUTE
   Axiom.PSTree = PTGAxiomInit(PrefixTerm.PSTree);
END;

RULE: PrefixTerm ::= Application PrefixTerm PrefixTerm
COMPUTE
   PrefixTerm[1].PSTree =
      PTGApply(
         PrefixTerm[2].PSTree,
         PrefixTerm[3].PSTree);
END;

RULE: PrefixTerm ::= Sconst
COMPUTE
   PrefixTerm.PSTree = PTGSconst();
END;
```

## pre2pstree.ptg

```
AxiomInit:
   "\\begingroup" [IndentNewLine]
   "\\psset{levelsep=2ex}" [IndentNewLine]
   $ [IndentNewLine]
   "\\endgroup" [IndentNewLine]
```

```
Sconst:
   "\\TR{S}\%" /* [IndentNewLine] */

Apply:
   "\\pstree{\\Tp}{\%" [IndentIncr] [IndentNewLine]
      $                [IndentNewLine]
      $ [IndentDecr]   [IndentNewLine]
   "}\%"
```

Παράδειγμα χρήσης :

```
> echo oSoSS | ./pre2pstree
\begingroup
\psset{levelsep=2ex}
\pstree{\Tp}{%
   \TR{S}%
   \pstree{\Tp}{%
      \TR{S}%
      \TR{S}%
   }%
}%
\endgroup
```

## Β.12 Πρόγραμμα isnormal

Αυτό το πρόγραμμα ελέγχει αν ένας όρος (σε προθεματική αναπαράσταση) που δίνεται είναι κανονική μορφή, δηλαδή το πρόγραμμα απλώς κάνει τεχνολόγηση (και καμμία μετατροπή).

### Προδιαγραφή isnormal.specs

```
prefix.gla   /* lexical analysis */
normal.con   /* grammar of normal form terms */
```

### Γραμματική normal.con

```
/* normal form S-term grammar, prefix notation */

Axiom:
   NormalTerm .
```

```
NormalTerm:
   Sconst /
   Application Sconst NormalTerm /
   Application Application Sconst NormalTerm NormalTerm .
```

## B.13 Πρόγραμμα length

Αυτό το πρόγραμμα επιστρέφει το μήκος (βλέπε ορισμό 11.2.1, στην σελίδα 123) ενός $S$-όρου σε προθεματική αναπαράσταση.

### Προδιαγραφή length.specs

```
/* calculate length of an S-term */
/* length is number of S's */

prefix.gla
prefix.con

length.lido
```

### length.lido

```
/* lenght of an S-Term in prefix notation  */

ATTR Length: int;

SYMBOL Axiom
COMPUTE
   printf("%d ", THIS.Length);
END;

RULE: Axiom ::= PrefixTerm
COMPUTE
   Axiom.Length = PrefixTerm.Length;
END;

RULE: PrefixTerm ::= Application PrefixTerm PrefixTerm
COMPUTE
   PrefixTerm[1].Length =
```

```
      ADD(PrefixTerm[2].Length, PrefixTerm[3].Length);
END;

RULE: PrefixTerm ::= Sconst
COMPUTE
   PrefixTerm.Length = 1;
END;
```

Παράδειγμα χρήσης :

```
> echo ooSSoSoSoSoSS | ./length
7
```

## Β.14 Τεχνολόγηση καταλόγων $S$-όρων

Θα δώσουμε τρία αρχεία που χρησιμοποιούνται από τα προγράμματα που επεξεργάζονται καταλόγους $S$-όρων. Έχουν σχεδιαστεί έτσι ώστε να συνδυάζονται με τις υπάρχουσες προδιαγραφές των προγραμμάτων που επεξεργάζονται έναν $S$-όρο, ούτως ώστε να προσφέρεται παρόμοια λειτουργικότητα από το πρόγραμμα με αυτήν του αντίστοιχου προγράμματος για έναν όρο, με τις ελάχιστες προδιαγραφές (βλέπε για παράδειγμα το πρόγραμμα listpre2in στην ενότητα B.15 με το αντίστοιχο pre2in).

### Γραμματική listsemi.con

```
ListAxiom:
  List .

List:
  /* empty */                /
  Axiom /* term */ ';' List  .
```

### listsemi.lido

```
ATTR Infix: PTGNode;

RULE: ListAxiom ::= List
COMPUTE
   PTGOut(List.Infix);
END;
```

```
RULE: List ::= Axiom ';' List
COMPUTE
   List[1].Infix = PTGSemiSeq(Axiom.Infix, List[2].Infix);
END;

RULE: List ::= /* empty */
COMPUTE
   List.Infix = PTGNULL;
END;
```

### listsemi.ptg

```
SemiSeq:     $ " ;\n" $
```

## B.15 Πρόγραμμα listpre2in

Το πρόγραμμα μετατρέπει έναν κατάλογο $S$-όρων, ο καθένας από τους οποίους είναι σε προθεματική αναπαράσταση στον αντίστοιχο κατάλογο, όπου ο κάθε όρος είναι σε ενθεματική αναπαράσταση, εν γένει με περιττές παρενθέσεις.

### Προδιαγραφή listpre2in.specs

```
pre2in.specs

listsemi.con
listsemi.lido
listsemi.ptg
```

Παρατηρήσατε ότι οι προδιαγραφές βασίζονται σε ήδη υπάρχοντα αρχεία και κυρίως στο αντίστοιχο αρχείο προδιαγραφών του προγράμματος που επιτελεί την αντίστοιχη επεξεργασία αλλά επί ενός μόνον όρου (pre2in.specs).

Παράδειγμα χρήσης :

```
> ./fine 3 | ./listpre2in
((S S ) S )  ;
(S (S S ) )  ;
```

## Β.16 Πρόγραμμα listinabb

Το πρόγραμμα μετατρέπει έναν κατάλογο $S$-όρων, ο καθένας από τους οποίους είναι σε ενθεματική αναπαράσταση στον αντίστοιχο κατάλογο, όπου ο κάθε όρος είναι πάλι σε ενθεματική αναπαράσταση, αλλά χωρίς περιττές παρενθέσεις και επιπλέον οι όροι $SSS$ και $S(SS)$ έχουν αντικατασταθεί από τις συντομογραφίες $A$ και $B$ αντιστοίχως.

### Προδιαγραφή listinabb.specs

```
inabb.specs

listsemi.con
listsemi.lido
listsemi.ptg
```

Παρομοίως, οι προδιαγραφές βασίζονται σε ήδη υπάρχοντα αρχεία και κυρίως στο αντίστοιχο αρχείο προδιαγραφών του προγράμματος που επιτελεί την αντίστοιχη επεξεργασία αλλά επί ενός μόνον όρου (inabb.specs).

Παράδειγμα χρήσης :

```
> ./fine 3 | ./listpre2in | ./listinabb
A  ;
B  ;
```

## Β.17 Πρόγραμμα checkterms

Το πρόγραμμα ελέγχει αν ο κατάλογος που δίνεται αποτελείται όντως από $S$-όρους σε προθεματική αναπαράσταση. Το πρόγραμμα απλώς εκτελεί τεχνολόγηση και δεν κάνει καμμία μετατροπή.

### Προδιαγραφή checkterms.specs

```
prefix.gla
prefix.con
listsemi.con
```

Παράδειγμα χρήσης :

```
> echo 'oSSS ;' | ./checkterms
"stdin", line 1:4 ERROR: Syntax error
"stdin", line 1:4 NOTE: Parsing resumed here
```

## B.18 Πρόγραμμα checknorm

Το πρόγραμμα ελέγχει αν ο κατάλογος που δίνεται αποτελείται από $S$-όρους σε προθεματική αναπαράσταση που είναι κανονικές μορφές. Το πρόγραμμα απλώς εκτελεί τεχνολόγηση και δεν κάνει καμμία μετατροπή.

### Προδιαγραφή checknorm.specs

```
isnormal.specs
listsemi.con
```

Οι προδιαγραφές βασίζονται σε ήδη υπάρχοντα αρχεία και κυρίως στο αντίστοιχο αρχείο προδιαγραφών του προγράμματος που επιτελεί την αντίστοιχη επεξεργασία αλλά επί ενός μόνον όρου (isnormal.specs).

Παράδειγμα χρήσης :

```
> echo 'ooo S S S S ;' | ./checknorm
"stdin", line 1:3 ERROR: Syntax error
"stdin", line 1:3 NOTE: Parsing resumed here
"stdin", line 1:11 ERROR: Syntax error
"stdin", line 1:11 NOTE: Parsing resumed here
```

# Παράρτημα Γ

# Γραμματική χωρίς συμφραζόμενα κανονικοποιήσιμων $S$-όρων

Ο Waldmann, όπως αναφέραμε, έχει δώσει κανονική γραμματική δένδρων για τους όρους που κανονικοποιούνται (βλέπε [105]). Μάλιστα στην ιστοσελίδα του [106] δίνει και ένα πρόγραμμα για την κατασκευή αυτομάτων δένδρων που αναγνωρίζουν την γλώσσα που αντιστοιχεί σε μία τέτοια γραμματική. Το πρόγραμμα αυτό, όμως, είναι γραμμένο στην γλώσσα συναρτησιακού προγραμματισμού Haskel.

Από την παραπάνω γραμματική δένδρων μπορεί εύκολα να προκύψει μία γραμματική συμβολοσειρών χωρίς συμφραζόμενα (context free), αν θεωρήσουμε την προθεματική παράσταση των $S$-όρων.[1] Για παράδειγμα :

$$T ::= Ap(S, S) \quad \text{γίνεται} \quad T ::= oSS$$

Μένει να κατασκευαστεί ένας τεχνολογητής (parser) για την εν λόγω γλώσσα συμβολοσειρών. Για τον σκοπό αυτό θα χρησιμοποιηθεί ένας κατασκευαστής τεχνολογητών (parser generator). Οι κανόνες που προκύπτουν, όμως, δεν είναι σε κάποια από τις μορφές, όπως LALR(1), που απαιτούν οι οι συνήθεις κατασκευαστές τεχνολογητών (π.χ. : yacc, bison, eli, antlr). Ο πιο απλός τρόπος επίλυσης του παραπάνω προβλήματος είναι να χρησιμοποιηθεί κάποιος γενικός κατασκευαστής τεχνολογητών, που να υποστηρίζει οπισθοδρόμηση (backtracking).

Τελικά, χρησιμοποιήθηκε ο κατασκευαστής τεχνολογητών btyacc (backtracking yacc), έκδοση 3.0, από το [10].

1. Βλέπε και πρόταση 9.6.7 στην σελίδα 112.

# Γ.1 Τεχνολόγηση κανονικοποιήσιμων $S$-όρων (normzin.y)

Η περιγραφή της γραμματικής δίνεται στο αρχείο normzin.y. Επιπλέον, στο τέλος της ενότητας, δίνουμε και την περιγραφή ενός λεκτικού αναλυτή (αρχείο sterm.lex, σε μορφή κατανοητή από τον κατασκευαστή λεκτικών αναλυτών flex) για την αναγνώριση των συμβόλων «$o$» και «$S$» της προθεματικής παράστασης $S$-όρων. Το παρακάτω script κατασκευάζει τον τεχνολογητή :

```
#!/bin/sh
rm -f y_tab.c y_tab.h lex.yy.c
flex sterm.lex
btyacc -d normzin.y
g++ -o btnzin y_tab.c -lfl
```

Το εκτελέσιμο πρόγραμμα btnzin δέχεται ως είσοδο μία συμβολοσειρά που αναπαριστά έναν $S$-όρο σε προθεματική παράσταση και δίνει έξοδο 0 αν ο $S$-όρος κανονικοποιείται και 1 αν ο $S$-όρος δεν κανονικοποιείται, π.χ. :

```
> echo "SSASA" | ./stermsyn | ./in2pre | ./btnzin
0
> echo "AAA" | ./stermsyn | ./in2pre | ./btnzin
1
```

Τα περιεχόμενα του αρχείου normzin.y είναι :

```
%{
#define YYDELETEVAL(x,y)
#define YYDELETEPOSN(x,y)
#define TRUE ((1 == 1))

#include "lex.yy.c"

%}

%token TOKENOAPPLY
%token TOKENSCOMB

%%

/* grammar of normalizing S-terms */
/*
```

```
   this grammar is from Waldmann's paper:
   The Combinator S,
   Johannes Waldmann,
   Information and Computation 159, 2-21 (2000)
*/



/* Axiom */
NormalizingTerm:
   pred_N ;



/* Waldmann abbreviates: T = SS */

Tterm:
   Application Sconst Sconst ;

/* {\cal P}_2 is S + T */
P2:
   Sconst | Tterm ;


/* Abbrev: A = SSS */
Aterm:
   Application Tterm Sconst ;



pred_N:
   Sconst                          |
   Application   pred_N0   pred_N  |
   Application   pred_L0   pred_N01 |
   Application   pred_L1   pred_L0  |
   Application   pred_N01  pred_N0  |
   Application   predS_N   Sconst   |
   Application   predT_N   Tterm
   ;

pred_N0:
```

```
   Sconst                          |
   Application   Sconst    pred_N  |
   Application   Tterm     pred_N0 |
   Application   predS_N0  Sconst  |
   Application   predT_N0  Tterm
   ;


pred_N01:
   Sconst                                  |
   Application   Sconst    pred_N          |
   Application
     Application Sconst
                 Application Sconst Tterm
     P2                                    |
   Application Application Sconst P2
               pred_N01                    |
   Application   predS_N01 Sconst          |
   Application   predT_N01 Tterm           ;


predS_L0:
   RepeatingT
      Application Sconst Application Sconst P2 ;

pred_L0:
   RepeatingT
      Application predS_L0 Sconst ;

predT_L1:
   Application Sconst Tterm | Aterm ;

predS_L1:
   RepeatingT
     Application
      RepeatingT Application Sconst Tterm
      Sconst
   ;

pred_L1:
   RepeatingT
```

```
      restpred_L1
   ;

restpred_L1:
   Application Application Sconst Tterm
               Application Sconst Tterm   |
   Application predT_L1 Tterm             |
   Application predS_L1 Sconst
   ;


N00:
   Sconst |
   Application  Tterm  N00    |
   Application  N00    Sconst ;


predT_N0:
   P2 ;

predS_N0:
   N00 ;

predT_N01:
   N00 | predS_L0 ;


predS_N01:
   predT_N01 |
   RepeatingT
     Application RepeatingT
                   Application Sconst N00
                 Sconst
   ;


predT_N:
   Sconst                                  |
   Application   Sconst      pred_N        |
   Application   predT_N01   P2            |
   Application Application Sconst P2 predT_N  |
```

```
   Application predSpredT_N Sconst            |
   Application predTpredT_N Tterm
   ;


predSpredT_N:
   predS_N01 |
   RepeatingT Application Sconst predT_N01 ;


predTpredT_N:
   RepeatingT Application Sconst predT_N01 ;


predTpredS_N:
   RepeatingT
     Application
       Application
         Sconst
         Application Sconst Tterm
       Tterm                        |
   predS_L1
   ;




/* for notational convenience:
   one   = predS(N),     two  = predS(one),
   three = predS(two),   four = predS(three)
 */

predS_N: one ;


one:
   Application Application Sconst N00 one ;

two:
   Application Tterm two ;
```

```
three:
   Application Tterm three ;

four:
   Application Tterm four ;


one:
   Application two Sconst ;

two:
   Application three Sconst ;

three:
   Application four Sconst ;


four:
   N00 ;

three:
   N00 ;

two:
   N00 ;

one:
   N00 ;


four:
   Application Sconst Aterm |
   Application Application Sconst Aterm Tterm ;


three:
   Application Sconst N00                    |
   Application Sconst
               RepeatingT
                 Application Sconst Tterm    |
   Application Application Sconst Aterm N00  |
```

```
   Application Application Aterm Sconst Aterm |
   Application Application Tterm Tterm  Aterm
   ;

two:
   Application Application Sconst N00 N00   |
   Application Application Application
      Sconst Tterm Tterm Tterm              |
   Application Sconst predS_N01             |
   Application
     Sconst
     RepeatingT
       Application Sconst predT_N01         |
   Application Application Aterm Sconst N00 |
   Application Application Tterm Tterm  N00
   ;

one:
   Application Sconst pred_N |
   Application Application Sconst predS_L0 predS_N01 |
   Application Application Sconst predS_L1 predS_L0  |
   Application RepeatingT
                 Application Sconst predS_N01
               predS_N0                            |
   Application predTpredS_N Tterm                  |
   Application N00 N00                             |
   Application
     RepeatingT
        Application Application Sconst Tterm Tterm
     N00                                           |
   Application Application Aterm Sconst predS_L0     |
   Application Application Tterm Tterm  predS_L0     |
   Application Aterm Application Sconst Tterm
   ;


RepeatingT:
   /* empty or ... */   |
   Application Tterm RepeatingT
   ;
```

```
Application:
   TOKENOAPPLY ;

Sconst:
   TOKENSCOMB ;


%%


int main(void)
{
   int retval;

   retval = yyparse();
   printf("%d\n", retval);
   return retval;
}


int yyerror(char *s)
{
   return 2;
}
```

Απομένει να δώσουμε την περιγραφή του λεκτικού αναλυτή (sterm.lex) :

```
%{
#include "y_tab.h"
%}

%%

o         return TOKENOAPPLY;
S         return TOKENSCOMB;
[ \t\n]+  /* ignore whitespace */

%%
```

## Γ.2 Μετρητής κανονικοποιήσιμων $S$-όρων (countercf.y)

Δίνουμε και ένα πρόγραμμα, το ccf, το οποίο δέχεται ως είσοδο έναν κατάλογο $S$-όρων στον οποίο κάθε $S$-όρος ακολουθείται από ένα ελληνικό ερωτηματικό (semicolon) « ;» και δίνει τρεις αριθμούς : το πλήθος των $S$-όρων, πόσοι από αυτούς είναι κανονικοποιήσιμοι και πόσοι δεν είναι κανονικοποιήσιμοι. Το εκτελέσιμο πρόγραμμα ccf βασίζεται στο αρχείο πηγαίου κώδικα countercf.y και κατασκευάζεται από το script makecou :

```
#!/bin/sh
rm y_tab.c
btyacc countercf.y
g++ -o ccf y_tab.c -lfl -g
```

Για παράδειγμα, το πλήθος των $S$-όρων μήκους 8 που δεν κανονικοποιούνται είναι ίσο με 41, όπως επιβεβαιώνεται και από κατάλληλη εκτέλεση του προγράμματος :

```
> ./fine 8 | ./ccf
       429         388          41
```

Η γραμματική που χρησιμοποιείται είναι ίδια όπως στο normzin.y. Σε αυτήν την περίπτωση, όμως, ο λεκτικός αναλυτής (συνάρτηση yylex) είναι κατασκευασμένος «με το χέρι» και όχι με κάποιο εργαλείο όπως το flex. Πρακτικά, ο λεκτικός αναλυτής φροντίζει αφού αναγνωριστεί ή όχι ένας όρος ως κανονικοποιήσιμος, να βρίσκει το επόμενο « ;» και να συνεχίζει με τον επόμενο όρο στην είσοδο. Τα περιεχόμενα του αρχείου countercf.y είναι :

```
%{
#define YYDELETEVAL(x,y)
#define YYDELETEPOSN(x,y)
#define TRUE ((1 == 1))

#include <assert.h>

static int lexglobal;

%}

%token TOKENOAPPLY
%token TOKENSCOMB
```

```
%%

/* grammar of normalizing S-terms */
/*
   this grammar is from Waldmann's paper:
   The Combinator S,
   Johannes Waldmann,
   Information and Computation 159, 2-21 (2000)
*/



/* Axiom */
NormalizingTerm:
   pred_N ;



/* Waldmann abbreviates: T = SS */

Tterm:
   Application Sconst Sconst ;

/* {\cal P}_2 is S + T */
P2:
   Sconst | Tterm ;


/* Abbrev: A = SSS */
Aterm:
   Application Tterm Sconst ;



pred_N:
   Sconst                              |
   Application   pred_N0   pred_N      |
   Application   pred_L0   pred_N01    |
   Application   pred_L1   pred_L0     |
   Application   pred_N01  pred_N0     |
```

```
   Application   predS_N   Sconst   |
   Application   predT_N   Tterm
   ;

pred_N0:
   Sconst                          |
   Application   Sconst    pred_N  |
   Application   Tterm     pred_N0 |
   Application   predS_N0  Sconst  |
   Application   predT_N0  Tterm
   ;


pred_N01:
   Sconst                                  |
   Application   Sconst    pred_N          |
   Application
     Application Sconst
                 Application Sconst Tterm
     P2                                    |
   Application Application Sconst P2
               pred_N01                    |
   Application   predS_N01 Sconst          |
   Application   predT_N01 Tterm           ;


predS_L0:
   RepeatingT
      Application Sconst Application Sconst P2 ;

pred_L0:
   RepeatingT
      Application predS_L0 Sconst ;

predT_L1:
   Application Sconst Tterm | Aterm ;

predS_L1:
   RepeatingT
     Application
      RepeatingT Application Sconst Tterm
```

```
      Sconst
   ;

pred_L1:
   RepeatingT
      restpred_L1
   ;

restpred_L1:
   Application Application Sconst Tterm
               Application Sconst Tterm   |
   Application predT_L1 Tterm             |
   Application predS_L1 Sconst
   ;


N00:
   Sconst |
   Application  Tterm  N00    |
   Application  N00    Sconst ;


predT_N0:
   P2 ;

predS_N0:
   N00 ;

predT_N01:
   N00 | predS_L0 ;


predS_N01:
   predT_N01 |
   RepeatingT
     Application RepeatingT
                   Application Sconst N00
                 Sconst
   ;
```

```
predT_N:
   Sconst                                        |
   Application   Sconst      pred_N              |
   Application   predT_N01   P2                  |
   Application Application Sconst P2 predT_N     |
   Application predSpredT_N Sconst               |
   Application predTpredT_N Tterm
   ;


predSpredT_N:
   predS_N01 |
   RepeatingT Application Sconst predT_N01 ;


predTpredT_N:
   RepeatingT Application Sconst predT_N01 ;


predTpredS_N:
   RepeatingT
     Application
       Application
         Sconst
         Application Sconst Tterm
       Tterm                          |
   predS_L1
   ;




/* for notational convenience:
   one   = predS(N),     two  = predS(one),
   three = predS(two),   four = predS(three)
 */

predS_N: one ;


one:
```

```
   Application Application Sconst N00 one ;

two:
   Application Tterm two ;

three:
   Application Tterm three ;

four:
   Application Tterm four ;


one:
   Application two Sconst ;

two:
   Application three Sconst ;

three:
   Application four Sconst ;


four:
   N00 ;

three:
   N00 ;

two:
   N00 ;

one:
   N00 ;


four:
   Application Sconst Aterm |
   Application Application Sconst Aterm Tterm ;


three:
```

```
   Application Sconst N00                      |
   Application Sconst
               RepeatingT
                 Application Sconst Tterm      |
   Application Application Sconst Aterm N00    |
   Application Application Aterm Sconst Aterm  |
   Application Application Tterm Tterm  Aterm
   ;

two:
   Application Application Sconst N00 N00    |
   Application Application Application
      Sconst Tterm Tterm Tterm               |
   Application Sconst predS_N01              |
   Application
     Sconst
     RepeatingT
       Application Sconst predT_N01          |
   Application Application Aterm Sconst N00  |
   Application Application Tterm Tterm  N00
   ;

one:
   Application Sconst pred_N |
   Application Application Sconst predS_L0 predS_N01 |
   Application Application Sconst predS_L1 predS_L0  |
   Application RepeatingT
                 Application Sconst predS_N01
               predS_N0                              |
   Application predTpredS_N Tterm                    |
   Application N00 N00                               |
   Application
     RepeatingT
        Application Application Sconst Tterm Tterm
     N00                                             |
   Application Application Aterm Sconst predS_L0     |
   Application Application Tterm Tterm  predS_L0     |
   Application Aterm Application Sconst Tterm
   ;
```

```
RepeatingT:
   /* empty or ... */   |
   Application Tterm RepeatingT
   ;

Application:
   TOKENOAPPLY ;

Sconst:
   TOKENSCOMB ;


%%


int main(void)
{
   extern int lexglobal;
   int retval;
   int c;
   int total, countNF, countNoNF;
   total = countNF = countNoNF = 0;

   lexglobal = 0;

   do {

      retval =  yyparse();
      total++;
      if (retval == 1)
         countNoNF++;
      else
         countNF++;

      do
      {
         c = getchar();
         if (lexglobal != ';')
         {
            if (c == EOF)
               break;
```

```
            if (c == 'S' || c == 'o')
            {
               c = ' ';
               continue;
            }
         }
      }
      while (c == ' ' || c == '\t' || c == '\n');

      assert(c != ' ' && c != '\t' && c != '\n');

      if (c == EOF)
      {
         break;
      }
      else
      {
         if (c != ';')
            ungetc(c, stdin);
      }
   }
   while (1);

   printf("%10d %10d %10d\n", total, countNF, countNoNF);

   return 0;
}


int yylex(void)
{
   extern int lexglobal;
   int c;
   do {
      c = getchar();

      lexglobal = c;

      if (c == ' ' || c == '\t' || c == '\n')
      {
         continue;
```

```
      }

      if (c == ';')
      {
          lexglobal = ';';
          return 0;
      }


      if (c == 'o')
         return TOKENOAPPLY;

      if (c == 'S')
         return TOKENSCOMB;

      return c;
   }
   while (1);
}

int yyerror(char *s)
{
   return 2;
}
```

# Παράρτημα Δ

# Κατάλογος μερικών μη κανονικοποιήσιμων $S$-όρων

Σε αυτό το παράρτημα σκοπεύουμε να δώσουμε όλους τους μη κανονικοποιήσιμους όρους μήκους μέχρι 10. Από τον πίνακα 13.1 (σελίδα 149) έχουμε ότι υπάρχουν 2 τέτοιοι όροι μήκους 7, 41 μήκους 8, 276 μήκους 9 και 1481 μήκους 10. Η παράθεση αυτών των όρων έχει νόημα, επειδή, όπως έχει αναφερθεί, προς το παρόν η μελέτη των $S$-όρων βασίζεται σημαντικά στην παρατήρησή τους, αφού ακόμα δεν έχει βρεθεί ένα ενοποιητικό πλαίσιο.

Προκειμένου να κατασκευάσουμε τους παραπάνω καταλόγους όρων χρησιμοποιούμε το παρακάτω script (nonflist) :

```
#!/bin/sh
if [ $# -eq 1 ]
then
   for i in `./fine $1 | ./numcf`
   do
      echo -n "\hspace*{2.5em}\llap{$i.}\ "
      echo "\(`./printh $1 $i | ./pre2in | ./inabb`\)\par"
   done
else
   echo "Usage: $0 length"
fi
```

το οποίο παίρνει ως όρισμα το μήκος για το οποίο θέλουμε να βρούμε τους μη κανονικοποιήσιμους $S$-όρους. Το παραπάνω script χρησιμοποιεί τα προγράμματα fine, printh, pre2in, inabb, τα οποία έχουν εξηγηθεί στα παραρτήματα Α και Β και επιπλέον το πρόγραμμα numcf, το οποίο δίνει τον αύξοντα αριθμό κάθε μη κανονικοποιήσιμου $S$-όρου συγκεκριμένου μήκους στην κωδικοποίηση που δόθηκε στον ορισμό 11.6.1 (σελίδα 127). Το πρόγραμμα numcf είναι

παρόμοιο με το πρόγραμμα ccf (βλέπε ενότητα Γ.2). Η περιγραφή numcf.y σε είσοδο κατανοητή από το btyacc για το πρόγραμμα numcf είναι ίδια με αυτήν του ccf, την countercf.y, και απλώς αλλάζουμε την συνάρτηση main ως εξής :

```
int main(void)
{
   extern int lexglobal;
   int retval;
   int c;
   int total, countNF, countNoNF;
   total = countNF = countNoNF = 0;
   int number = 0;
   lexglobal = 0;

   do {
      number++;
      retval =  yyparse();
      total++;
      if (retval == 1)
      {
         countNoNF++;
         printf("%d\n", number);
      }
      else
      {
         countNF++;
      }

      do {
         c = getchar();
         if (lexglobal != ';')
         {
            if (c == EOF)
               break;
            if (c == 'S' || c == 'o')
            {
               c = ' ';
               continue;
            }
         }
      } while (c == ' ' || c == '\t' || c == '\n');
```

```
      if (c == EOF)
         break;
      else
      {
         if (c != ';')
            ungetc(c, stdin);
      }
   } while (1);

   return 0;
}
```

# Δ.1 Μη κανονικοποιήσιμοι όροι μήκους 7

2. $BSSSS$
4. $A(SS)SS$

# Δ.2 Μη κανονικοποιήσιμοι όροι μήκους 8

2. $BSSSSS$
4. $A(SS)SSS$
9. $SBSSSS$
10. $B(SS)SSS$
11. $BS(SS)SS$
12. $BSS(SS)S$
13. $BSSS(SS)$
19. $AASS$
20. $ABSS$
21. $A(SS)(SS)S$
22. $A(SS)S(SS)$
29. $S(BS)SSS$
32. $SAS(SS)S$
34. $S(SA)SSS$
35. $S(SB)SSS$
36. $SB(SS)SS$
37. $SBS(SS)S$
38. $SBSS(SS)$
40. $BBSS$
42. $B(SS)S(SS)$
43. $BSAS$
44. $BSBS$
45. $BS(SS)(SS)$
63. $A(BS)S$
65. $AA(SS)$
66. $A(SA)S$
67. $A(SB)S$
68. $AB(SS)$
77. $S(BSS)SS$
79. $S(A(SS))SS$
83. $S(SBS)SS$
84. $S(B(SS))SS$
86. $S(BS)S(SS)$
94. $SASA$
95. $SASB$
97. $S(S(BS))SS$
99. $S(SA)(SS)S$
100. $S(SA)S(SS)$
101. $S(S(SA))SS$
299. $S(BSSSS)$
301. $S(A(SS)SS)$

# Δ.3 Μη κανονικοποιήσιμοι όροι μήκους 9

2. $BSSSSSS$
4. $A(SS)SSSS$
5. $AS(SS)SSS$
6. $ASS(SS)SS$
9. $SASSSSS$
10. $SBSSSSS$
11. $B(SS)SSSS$
12. $BS(SS)SSS$
13. $BSS(SS)SS$
14. $BSSS(SS)S$
15. $BSSSS(SS)$
17. $SSBSSSS$
18. $SS(SS)(SS)SSS$
19. $SS(SS)S(SS)SS$
22. $AASSS$
23. $ABSSS$
24. $A(SS)(SS)SS$
25. $A(SS)S(SS)S$
26. $A(SS)SS(SS)$
28. $ASBSS$
29. $AS(SS)(SS)S$
30. $AS(SS)S(SS)$
32. $ASSBS$
36. $S(AS)SSSS$
37. $S(BS)SSSS$
38. $S(SS(SS))SSSS$
40. $SAS(SS)SS$
41. $SASS(SS)S$
42. $SASSS(SS)$
43. $S(SA)SSSS$
44. $S(SB)SSSS$
45. $SB(SS)SSS$
46. $SBS(SS)SS$
47. $SBSS(SS)S$
48. $SBSSS(SS)$
49. $BASSS$
50. $BBSSS$
52. $B(SS)S(SS)S$
53. $B(SS)SS(SS)$
54. $BSASS$
55. $BSBSS$
56. $BS(SS)(SS)S$
57. $BS(SS)S(SS)$
58. $BSSAS$
59. $BSSBS$
60. $BSS(SS)(SS)$
61. $BSSSA$
62. $BSSSB$
64. $SS(BS)SSS$
66. $SSA(SS)SS$
71. $SSB(SS)SS$
75. $SS(SS)BSS$
76. $SS(SS)(SS)(SS)S$
77. $SS(SS)(SS)S(SS)$
79. $SS(SS)SBS$
83. $A(AS)SS$
84. $A(BS)SS$
85. $A(SS(SS))SS$
86. $AA(SS)S$
87. $AAS(SS)$
88. $A(SA)SS$
89. $A(SB)SS$
90. $AB(SS)S$
91. $ABS(SS)$
92. $A(SS)AS$
93. $A(SS)BS$
94. $A(SS)(SS)(SS)$
95. $A(SS)SA$
96. $A(SS)SB$
98. $AS(BS)S$
100. $ASA(SS)$
101. $AS(SA)S$
103. $ASB(SS)$
112. $S(BSS)SSS$
114. $S(A(SS))SSS$
116. $S(AS)S(SS)S$
118. $S(SAS)SSS$
119. $S(SBS)SSS$
120. $S(B(SS))SSS$
121. $S(BS)(SS)SS$
122. $S(BS)S(SS)S$
123. $S(BS)SS(SS)$
127. $S(SS(SS))S(SS)S$
130. $SABSS$
131. $SA(SS)(SS)S$
132. $SA(SS)S(SS)$
133. $SASAS$
134. $SASBS$
135. $SAS(SS)(SS)$
138. $S(S(AS))SSS$
139. $S(S(BS))SSS$
140. $S(S(SS(SS)))SSS$
141. $S(SA)(SS)SS$
142. $S(SA)S(SS)S$
143. $S(SA)SS(SS)$
144. $S(S(SA))SSS$
145. $S(S(SB))SSS$
146. $S(SB)(SS)SS$
147. $S(SB)S(SS)S$
148. $S(SB)SS(SS)$
149. $SBASS$
150. $SBBSS$
152. $SB(SS)S(SS)$
153. $SBSAS$
154. $SBSBS$
155. $SBS(SS)(SS)$
156. $SBSSA$
157. $SBSSB$
159. $B(BS)SS$
161. $BA(SS)S$
162. $BAS(SS)$
163. $B(SA)SS$
164. $B(SB)SS$
165. $BB(SS)S$
166. $BBS(SS)$
168. $B(SS)BS$
169. $B(SS)(SS)(SS)$
170. $B(SS)SA$
171. $B(SS)SB$
172. $BS(AS)S$
173. $BS(BS)S$
174. $BS(SS(SS))S$
175. $BSA(SS)$
176. $BS(SA)S$
177. $BS(SB)S$
178. $BSB(SS)$
179. $BS(SS)A$
180. $BS(SS)B$
187. $SS(BSS)SS$
189. $SS(A(SS))SS$
202. $SSABS$
216. $SSBBS$
221. $SS(SS)(BS)S$
223. $SS(SS)A(SS)$
224. $SS(SS)(SA)S$
226. $SS(SS)B(SS)$
235. $A(BSS)S$
237. $A(A(SS))S$
238. $A(AS)(SS)$
239. $A(SAS)S$
240. $A(SBS)S$
241. $A(B(SS))S$
242. $A(BS)(SS)$
244. $A(SSB)S$
245. $A(SS(SS))(SS)$
246. $AAA$
247. $AAB$
248. $A(S(AS))S$
249. $A(S(BS))S$
250. $A(S(SS(SS)))S$
251. $A(SA)(SS)$
252. $A(S(SA))S$
253. $A(S(SB))S$
254. $A(SB)(SS)$
255. $ABA$
256. $ABB$
277. $S(BSSS)SS$
279. $S(A(SS)S)SS$
280. $S(AS(SS))SS$
283. $S(SASS)SS$
284. $S(SBSS)SS$
285. $S(B(SS)S)SS$
286. $S(BS(SS))SS$
287. $S(BSS)(SS)S$
288. $S(BSS)S(SS)$
291. $S(SS(SS)(SS))SS$
294. $S(AA)SS$
295. $S(AB)SS$
296. $S(A(SS))(SS)S$
297. $S(A(SS))S(SS)$
301. $S(AS)SA$
302. $S(AS)SB$
304. $S(S(BS)S)SS$
306. $S(SA(SS))SS$
308. $S(SAS)S(SS)$
309. $S(S(SA)S)SS$
310. $S(S(SB)S)SS$
311. $S(SB(SS))SS$
312. $S(SBS)(SS)S$
313. $S(SBS)S(SS)$
314. $S(BA)SS$
315. $S(BB)SS$
316. $S(B(SS))(SS)S$
317. $S(B(SS))S(SS)$
320. $S(BS)(SS)(SS)$
321. $S(BS)SA$

322. $S(BS)SB$
335. $S(SS(SS))SA$
336. $S(SS(SS))SB$
340. $SAA(SS)$
343. $SAB(SS)$
344. $SA(SS)A$
345. $SA(SS)B$
346. $SAS(AS)$
347. $SAS(BS)$
348. $SAS(SS(SS))$
349. $SAS(SA)$
350. $SAS(SB)$
352. $S(S(BSS))SS$
354. $S(S(A(SS)))SS$
355. $S(S(AS))(SS)S$
356. $S(S(AS))S(SS)$
357. $S(S(SAS))SS$
358. $S(S(SBS))SS$
359. $S(S(B(SS)))SS$
360. $S(S(BS))(SS)S$
361. $S(S(BS))S(SS)$
364. $S(S(SS(SS)))(SS)S$
365. $S(S(SS(SS)))S(SS)$
366. $S(SA)AS$
367. $S(SA)BS$
368. $S(SA)(SS)(SS)$
369. $S(SA)SA$
370. $S(SA)SB$
371. $S(S(S(AS)))SS$
372. $S(S(S(BS)))SS$
373. $S(S(S(SS(SS))))SS$
374. $S(S(SA))(SS)S$
375. $S(S(SA))S(SS)$
376. $S(S(S(SA)))SS$
377. $S(S(S(SB)))SS$
378. $S(S(SB))(SS)S$
379. $S(S(SB))S(SS)$
383. $S(SB)SA$
384. $S(SB)SB$
388. $SBA(SS)$
389. $SB(SA)S$
391. $SBB(SS)$
431. $BS(SAS)$
442. $SS(BSSS)S$
444. $SS(A(SS)S)S$
574. $S(BSSSS)S$
576. $S(A(SS)SS)S$
871. $SS(BSSSS)$
873. $SS(A(SS)SS)$
1003. $S(BSSSSS)$
1005. $S(A(SS)SSS)$
1010. $S(SBSSSS)$
1011. $S(B(SS)SSS)$
1012. $S(BS(SS)SS)$
1013. $S(BSS(SS)S)$
1014. $S(BSSS(SS))$
1020. $S(AASS)$
1021. $S(ABSS)$
1022. $S(A(SS)(SS)S)$
1023. $S(A(SS)S(SS))$
1030. $S(S(BS)SSS)$
1033. $S(SAS(SS)S)$
1035. $S(S(SA)SSS)$
1036. $S(S(SB)SSS)$
1037. $S(SB(SS)SS)$
1038. $S(SBS(SS)S)$
1039. $S(SBSS(SS))$
1041. $S(BBSS)$
1043. $S(B(SS)S(SS))$
1044. $S(BSAS)$
1045. $S(BSBS)$
1046. $S(BS(SS)(SS))$
1064. $S(A(BS)S)$
1066. $S(AA(SS))$
1067. $S(A(SA)S)$
1068. $S(A(SB)S)$
1069. $S(AB(SS))$
1078. $S(S(BSS)SS)$
1080. $S(S(A(SS))SS)$
1084. $S(S(SBS)SS)$
1085. $S(S(B(SS))SS)$
1087. $S(S(BS)S(SS))$
1095. $S(SASA)$
1096. $S(SASB)$
1098. $S(S(S(BS))SS)$
1100. $S(S(SA)(SS)S)$
1101. $S(S(SA)S(SS))$
1102. $S(S(S(SA))SS)$
1300. $S(S(BSSSS))$
1302. $S(S(A(SS)SS))$

# Δ.4 Μη κανονικοποιήσιμοι όροι μήκους 10

2. $BSSSSSSS$
4. $A(SS)SSSSS$
5. $AS(SS)SSSS$
6. $ASS(SS)SSS$
7. $ASSS(SS)SS$
10. $SASSSSSS$
11. $SBSSSSSS$
12. $B(SS)SSSSS$
13. $BS(SS)SSSS$
14. $BSS(SS)SSS$
15. $BSSS(SS)SS$
16. $BSSSS(SS)S$
17. $BSSSSS(SS)$
19. $SSBSSSSS$
20. $SS(SS)(SS)SSSS$
21. $SS(SS)S(SS)SSS$
22. $SS(SS)SS(SS)SS$
25. $AASSSS$
26. $ABSSSS$
27. $A(SS)(SS)SSS$
28. $A(SS)S(SS)SS$
29. $A(SS)SS(SS)S$
30. $A(SS)SSS(SS)$
32. $ASBSSS$
33. $AS(SS)(SS)SS$
34. $AS(SS)S(SS)S$
35. $AS(SS)SS(SS)$
36. $ASSASS$
37. $ASSBSS$
38. $ASS(SS)(SS)S$
39. $ASS(SS)S(SS)$
41. $ASSSBS$
45. $S(AS)SSSSS$
46. $S(BS)SSSSS$
47. $S(SS(SS))SSSSS$
49. $SAS(SS)SSS$
50. $SASS(SS)SS$
51. $SASSS(SS)S$
52. $SASSSS(SS)$
53. $S(SA)SSSSS$
54. $S(SB)SSSSS$
55. $SB(SS)SSSS$
56. $SBS(SS)SSS$
57. $SBSS(SS)SS$
58. $SBSSS(SS)S$
59. $SBSSSS(SS)$
60. $BASSSS$
61. $BBSSSS$
62. $B(SS)(SS)SSS$
63. $B(SS)S(SS)SS$
64. $B(SS)SS(SS)S$
65. $B(SS)SSS(SS)$
66. $BSASSS$
67. $BSBSSS$
68. $BS(SS)(SS)SS$
69. $BS(SS)S(SS)S$
70. $BS(SS)SS(SS)$
71. $BSSASS$
72. $BSSBSS$
73. $BSS(SS)(SS)S$
74. $BSS(SS)S(SS)$
75. $BSSSAS$
76. $BSSSBS$
77. $BSSS(SS)(SS)$
78. $BSSSSA$
79. $BSSSSB$

81. $SS(BS)SSSS$
83. $SSA(SS)SSS$
84. $SSAS(SS)SS$
88. $SS(SB)SSSS$
89. $SSB(SS)SSS$
90. $SSBS(SS)SS$
91. $SSBSS(SS)S$
92. $SSBSSS(SS)$
94. $SS(SS)BSSS$
95. $SS(SS)(SS)(SS)SS$
96. $SS(SS)(SS)S(SS)S$
97. $SS(SS)(SS)SS(SS)$
98. $SS(SS)SASS$
99. $SS(SS)SBSS$
100. $SS(SS)S(SS)(SS)S$
101. $SS(SS)S(SS)S(SS)$
103. $SS(SS)SSBS$
107. $A(AS)SSS$
108. $A(BS)SSS$
109. $A(SS(SS))SSS$
110. $AA(SS)SS$
111. $AAS(SS)S$
112. $AASS(SS)$
113. $A(SA)SSS$
114. $A(SB)SSS$
115. $AB(SS)SS$
116. $ABS(SS)S$
117. $ABSS(SS)$
118. $A(SS)ASS$
119. $A(SS)BSS$
120. $A(SS)(SS)(SS)S$
121. $A(SS)(SS)S(SS)$
122. $A(SS)SAS$
123. $A(SS)SBS$
124. $A(SS)S(SS)(SS)$
125. $A(SS)SSA$
126. $A(SS)SSB$
128. $AS(BS)SS$
130. $ASA(SS)S$
131. $ASAS(SS)$
132. $AS(SA)SS$
133. $AS(SB)SS$
134. $ASB(SS)S$
135. $ASBS(SS)$
136. $AS(SS)AS$
137. $AS(SS)BS$
138. $AS(SS)(SS)(SS)$
139. $AS(SS)SA$
140. $AS(SS)SB$
142. $ASS(BS)S$
144. $ASSA(SS)$
145. $ASS(SA)S$
146. $ASS(SB)S$
147. $ASSB(SS)$
155. $S(ASS)SSSS$
156. $S(BSS)SSSS$
157. $S(SS(SS)S)SSSS$
158. $S(A(SS))SSSS$
159. $S(AS)(SS)SSS$
160. $S(AS)S(SS)SS$
161. $S(AS)SS(SS)S$
162. $S(AS)SSS(SS)$
163. $S(SAS)SSSS$
164. $S(SBS)SSSS$
165. $S(B(SS))SSSS$
166. $S(BS)(SS)SSS$
167. $S(BS)S(SS)SS$
168. $S(BS)SS(SS)S$
169. $S(BS)SSS(SS)$
170. $S(SSA)SSSS$
171. $S(SSB)SSSS$
172. $S(SS(SS))(SS)SSS$
173. $S(SS(SS))S(SS)SS$
174. $S(SS(SS))SS(SS)S$
175. $S(SS(SS))SSS(SS)$
177. $SABSSS$
178. $SA(SS)(SS)SS$
179. $SA(SS)S(SS)S$
180. $SA(SS)SS(SS)$
181. $SASASS$
182. $SASBSS$
183. $SAS(SS)(SS)S$
184. $SAS(SS)S(SS)$
185. $SASSAS$
186. $SASSBS$
187. $SASS(SS)(SS)$
188. $SASSSA$
189. $SASSSB$
190. $S(S(AS))SSSS$
191. $S(S(BS))SSSS$
192. $S(S(SS(SS)))SSSS$
193. $S(SA)(SS)SSS$
194. $S(SA)S(SS)SS$
195. $S(SA)SS(SS)S$
196. $S(SA)SSS(SS)$
197. $S(S(SA))SSSS$
198. $S(S(SB))SSSS$
199. $S(SB)(SS)SSS$
200. $S(SB)S(SS)SS$
201. $S(SB)SS(SS)S$
202. $S(SB)SSS(SS)$
203. $SBASSS$
204. $SBBSSS$
205. $SB(SS)(SS)SS$
206. $SB(SS)S(SS)S$
207. $SB(SS)SS(SS)$
208. $SBSASS$
209. $SBSBSS$
210. $SBS(SS)(SS)S$
211. $SBS(SS)S(SS)$
212. $SBSSAS$
213. $SBSSBS$
214. $SBSS(SS)(SS)$
215. $SBSSSA$
216. $SBSSSB$
217. $B(AS)SSS$
218. $B(BS)SSS$
219. $B(SS(SS))SSS$
220. $BA(SS)SS$
221. $BAS(SS)S$
222. $BASS(SS)$
223. $B(SA)SSS$
224. $B(SB)SSS$
225. $BB(SS)SS$
226. $BBS(SS)S$
227. $BBSS(SS)$
228. $B(SS)ASS$
229. $B(SS)BSS$
230. $B(SS)(SS)(SS)S$
231. $B(SS)(SS)S(SS)$
232. $B(SS)SAS$
233. $B(SS)SBS$
234. $B(SS)S(SS)(SS)$
235. $B(SS)SSA$
236. $B(SS)SSB$
237. $BS(AS)SS$
238. $BS(BS)SS$
239. $BS(SS(SS))SS$
240. $BSA(SS)S$
241. $BSAS(SS)$
242. $BS(SA)SS$
243. $BS(SB)SS$
244. $BSB(SS)S$
245. $BSBS(SS)$
246. $BS(SS)AS$
247. $BS(SS)BS$
248. $BS(SS)(SS)(SS)$
249. $BS(SS)SA$
250. $BS(SS)SB$
251. $BSS(AS)S$
252. $BSS(BS)S$
253. $BSS(SS(SS))S$
254. $BSSA(SS)$
255. $BSS(SA)S$
256. $BSS(SB)S$
257. $BSSB(SS)$
258. $BSS(SS)A$
259. $BSS(SS)B$
260. $BSSS(AS)$
261. $BSSS(BS)$
262. $BSSS(SS(SS))$
263. $BSSS(SA)$
264. $BSSS(SB)$
266. $SS(BSS)SSS$
268. $SS(A(SS))SSS$
269. $SS(AS)(SS)SS$
273. $SS(SBS)SSS$
274. $SS(B(SS))SSS$
275. $SS(BS)(SS)SS$
276. $SS(BS)S(SS)S$
277. $SS(BS)SS(SS)$
280. $SS(SS(SS))(SS)SS$
283. $SSAASS$
284. $SSABSS$
285. $SSA(SS)(SS)S$
286. $SSA(SS)S(SS)$
288. $SSASBS$
293. $SS(S(BS))SSS$
295. $SS(SA)(SS)SS$
296. $SS(SA)S(SS)S$
298. $SS(S(SA))SSS$
299. $SS(S(SB))SSS$
300. $SS(SB)(SS)SS$
301. $SS(SB)S(SS)S$
302. $SS(SB)SS(SS)$
303. $SSBASS$

304. $SSBBSS$
305. $SSB(SS)(SS)S$
306. $SSB(SS)S(SS)$
307. $SSBSAS$
308. $SSBSBS$
309. $SSBS(SS)(SS)$
313. $SS(SS)(BS)SS$
315. $SS(SS)A(SS)S$
316. $SS(SS)AS(SS)$
317. $SS(SS)(SA)SS$
318. $SS(SS)(SB)SS$
319. $SS(SS)B(SS)S$
320. $SS(SS)BS(SS)$
321. $SS(SS)(SS)AS$
322. $SS(SS)(SS)BS$
323. $SS(SS)(SS)(SS)(SS)$
324. $SS(SS)(SS)SA$
325. $SS(SS)(SS)SB$
327. $SS(SS)S(BS)S$
329. $SS(SS)SA(SS)$
330. $SS(SS)S(SA)S$
331. $SS(SS)S(SB)S$
332. $SS(SS)SB(SS)$
340. $A(ASS)SS$
341. $A(BSS)SS$
342. $A(SS(SS)S)SS$
343. $A(A(SS))SS$
344. $A(AS)(SS)S$
345. $A(AS)S(SS)$
346. $A(SAS)SS$
347. $A(SBS)SS$
348. $A(B(SS))SS$
349. $A(BS)(SS)S$
350. $A(BS)S(SS)$
351. $A(SSA)SS$
352. $A(SSB)SS$
353. $A(SS(SS))(SS)S$
354. $A(SS(SS))S(SS)$
355. $AAAS$
356. $AABS$
357. $AA(SS)(SS)$
358. $AASA$
359. $AASB$
360. $A(S(AS))SS$
361. $A(S(BS))SS$
362. $A(S(SS(SS)))SS$
363. $A(SA)(SS)S$
364. $A(SA)S(SS)$
365. $A(S(SA))SS$
366. $A(S(SB))SS$
367. $A(SB)(SS)S$
368. $A(SB)S(SS)$
369. $ABAS$
370. $ABBS$
371. $AB(SS)(SS)$
372. $ABSA$
373. $ABSB$
374. $A(SS)(AS)S$
375. $A(SS)(BS)S$
376. $A(SS)(SS(SS))S$
377. $A(SS)A(SS)$
378. $A(SS)(SA)S$
379. $A(SS)(SB)S$
380. $A(SS)B(SS)$
381. $A(SS)(SS)A$
382. $A(SS)(SS)B$
383. $A(SS)S(AS)$
384. $A(SS)S(BS)$
385. $A(SS)S(SS(SS))$
386. $A(SS)S(SA)$
387. $A(SS)S(SB)$
389. $AS(BSS)S$
391. $AS(A(SS))S$
392. $AS(AS)(SS)$
393. $AS(SAS)S$
394. $AS(SBS)S$
395. $AS(B(SS))S$
396. $AS(BS)(SS)$
399. $AS(SS(SS))(SS)$
400. $ASAA$
401. $ASAB$
402. $AS(S(AS))S$
403. $AS(S(BS))S$
404. $AS(S(SS(SS)))S$
405. $AS(SA)(SS)$
406. $AS(S(SA))S$
407. $AS(S(SB))S$
408. $AS(SB)(SS)$
409. $ASBA$
410. $ASBB$
412. $AS(SS)(BS)$
431. $S(BSSS)SSS$
433. $S(A(SS)S)SSS$
434. $S(AS(SS))SSS$
436. $S(ASS)S(SS)S$
438. $S(SASS)SSS$
439. $S(SBSS)SSS$
440. $S(B(SS)S)SSS$
441. $S(BS(SS))SSS$
442. $S(BSS)(SS)SS$
443. $S(BSS)S(SS)S$
444. $S(BSS)SS(SS)$
446. $S(SSBS)SSS$
447. $S(SS(SS)(SS))SSS$
449. $S(SS(SS)S)S(SS)S$
451. $S(AA)SSS$
452. $S(AB)SSS$
453. $S(A(SS))(SS)SS$
454. $S(A(SS))S(SS)S$
455. $S(A(SS))SS(SS)$
457. $S(AS)BSS$
458. $S(AS)(SS)(SS)S$
459. $S(AS)(SS)S(SS)$
460. $S(AS)SAS$
461. $S(AS)SBS$
462. $S(AS)S(SS)(SS)$
465. $S(S(AS)S)SSS$
466. $S(S(BS)S)SSS$
467. $S(S(SS(SS))S)SSS$
468. $S(SA(SS))SSS$
469. $S(SAS)(SS)SS$
470. $S(SAS)S(SS)S$
471. $S(SAS)SS(SS)$
472. $S(S(SA)S)SSS$
473. $S(S(SB)S)SSS$
474. $S(SB(SS))SSS$
475. $S(SBS)(SS)SS$
476. $S(SBS)S(SS)S$
477. $S(SBS)SS(SS)$
478. $S(BA)SSS$
479. $S(BB)SSS$
480. $S(B(SS))(SS)SS$
481. $S(B(SS))S(SS)S$
482. $S(B(SS))SS(SS)$
483. $S(BS)ASS$
484. $S(BS)BSS$
485. $S(BS)(SS)(SS)S$
486. $S(BS)(SS)S(SS)$
487. $S(BS)SAS$
488. $S(BS)SBS$
489. $S(BS)S(SS)(SS)$
490. $S(BS)SSA$
491. $S(BS)SSB$
493. $S(SS(BS))SSS$
496. $S(SSA)S(SS)S$
498. $S(SS(SA))SSS$
499. $S(SS(SB))SSS$
500. $S(SSB)(SS)SS$
501. $S(SSB)S(SS)S$
502. $S(SSB)SS(SS)$
504. $S(SS(SS))BSS$
505. $S(SS(SS))(SS)(SS)S$
506. $S(SS(SS))(SS)S(SS)$
507. $S(SS(SS))SAS$
508. $S(SS(SS))SBS$
509. $S(SS(SS))S(SS)(SS)$
513. $SA(BS)SS$
515. $SAA(SS)S$
516. $SAAS(SS)$
517. $SA(SA)SS$
518. $SA(SB)SS$
519. $SAB(SS)S$
520. $SABS(SS)$
521. $SA(SS)AS$
522. $SA(SS)BS$
523. $SA(SS)(SS)(SS)$
524. $SA(SS)SA$
525. $SA(SS)SB$
526. $SAS(AS)S$
527. $SAS(BS)S$
528. $SAS(SS(SS))S$
529. $SASA(SS)$
530. $SAS(SA)S$
531. $SAS(SB)S$
532. $SASB(SS)$
533. $SAS(SS)A$
534. $SAS(SS)B$
536. $SASS(BS)$
540. $S(S(ASS))SSS$
541. $S(S(BSS))SSS$
542. $S(S(SS(SS)S))SSS$
543. $S(S(A(SS)))SSS$
544. $S(S(AS))(SS)SS$
545. $S(S(AS))S(SS)S$
546. $S(S(AS))SS(SS)$
547. $S(S(SAS))SSS$

548. $S(S(SBS))SSS$
549. $S(S(B(SS)))SSS$
550. $S(S(BS))(SS)SS$
551. $S(S(BS))S(SS)S$
552. $S(S(BS))SS(SS)$
553. $S(S(SSA))SSS$
554. $S(S(SSB))SSS$
555. $S(S(SS(SS)))(SS)SS$
556. $S(S(SS(SS)))S(SS)S$
557. $S(S(SS(SS)))SS(SS)$
558. $S(SA)ASS$
559. $S(SA)BSS$
560. $S(SA)(SS)(SS)S$
561. $S(SA)(SS)S(SS)$
562. $S(SA)SAS$
563. $S(SA)SBS$
564. $S(SA)S(SS)(SS)$
565. $S(SA)SSA$
566. $S(SA)SSB$
567. $S(S(S(AS)))SSS$
568. $S(S(S(BS)))SSS$
569. $S(S(S(SS(SS))))SSS$
570. $S(S(SA))(SS)SS$
571. $S(S(SA))S(SS)S$
572. $S(S(SA))SS(SS)$
573. $S(S(S(SA)))SSS$
574. $S(S(S(SB)))SSS$
575. $S(S(SB))(SS)SS$
576. $S(S(SB))S(SS)S$
577. $S(S(SB))SS(SS)$
578. $S(SB)ASS$
579. $S(SB)BSS$
580. $S(SB)(SS)(SS)S$
581. $S(SB)(SS)S(SS)$
582. $S(SB)SAS$
583. $S(SB)SBS$
584. $S(SB)S(SS)(SS)$
585. $S(SB)SSA$
586. $S(SB)SSB$
587. $SB(AS)SS$
588. $SB(BS)SS$
589. $SB(SS(SS))SS$
590. $SBA(SS)S$
591. $SBAS(SS)$
592. $SB(SA)SS$
593. $SB(SB)SS$
594. $SBB(SS)S$
595. $SBBS(SS)$
596. $SB(SS)AS$
597. $SB(SS)BS$
598. $SB(SS)(SS)(SS)$
599. $SB(SS)SA$
600. $SB(SS)SB$
601. $SBS(AS)S$
602. $SBS(BS)S$
603. $SBS(SS(SS))S$
604. $SBSA(SS)$
605. $SBS(SA)S$
606. $SBS(SB)S$
607. $SBSB(SS)$
608. $SBS(SS)A$
609. $SBS(SS)B$
610. $SBSS(AS)$
611. $SBSS(BS)$
612. $SBSS(SS(SS))$
613. $SBSS(SA)$
614. $SBSS(SB)$
616. $B(BSS)SS$
618. $B(A(SS))SS$
619. $B(AS)(SS)S$
620. $B(AS)S(SS)$
621. $B(SAS)SS$
622. $B(SBS)SS$
623. $B(B(SS))SS$
624. $B(BS)(SS)S$
625. $B(BS)S(SS)$
627. $B(SSB)SS$
628. $B(SS(SS))(SS)S$
629. $B(SS(SS))S(SS)$
630. $BAAS$
631. $BABS$
632. $BA(SS)(SS)$
633. $BASA$
634. $BASB$
635. $B(S(AS))SS$
636. $B(S(BS))SS$
637. $B(S(SS(SS)))SS$
638. $B(SA)(SS)S$
639. $B(SA)S(SS)$
640. $B(S(SA))SS$
641. $B(S(SB))SS$
642. $B(SB)(SS)S$
643. $B(SB)S(SS)$
644. $BBAS$
645. $BBBS$
646. $BB(SS)(SS)$
647. $BBSA$
648. $BBSB$
650. $B(SS)(BS)S$
652. $B(SS)A(SS)$
653. $B(SS)(SA)S$
654. $B(SS)(SB)S$
655. $B(SS)B(SS)$
656. $B(SS)(SS)A$
657. $B(SS)(SS)B$
658. $B(SS)S(AS)$
659. $B(SS)S(BS)$
660. $B(SS)S(SS(SS))$
661. $B(SS)S(SA)$
662. $B(SS)S(SB)$
663. $BS(ASS)S$
664. $BS(BSS)S$
665. $BS(SS(SS)S)S$
666. $BS(A(SS))S$
667. $BS(AS)(SS)$
668. $BS(SAS)S$
669. $BS(SBS)S$
670. $BS(B(SS))S$
671. $BS(BS)(SS)$
672. $BS(SSA)S$
673. $BS(SSB)S$
674. $BS(SS(SS))(SS)$
675. $BSAA$
676. $BSAB$
677. $BS(S(AS))S$
678. $BS(S(BS))S$
679. $BS(S(SS(SS)))S$
680. $BS(SA)(SS)$
681. $BS(S(SA))S$
682. $BS(S(SB))S$
683. $BS(SB)(SS)$
684. $BSBA$
685. $BSBB$
686. $BS(SS)(AS)$
687. $BS(SS)(BS)$
688. $BS(SS)(SS(SS))$
689. $BS(SS)(SA)$
690. $BS(SS)(SB)$
692. $BSS(BSS)$
694. $BSS(A(SS))$
695. $BSS(SAS)$
697. $BSS(B(SS))$
706. $SS(BSSS)SS$
708. $SS(A(SS)S)SS$
713. $SS(SBSS)SS$
714. $SS(B(SS)S)SS$
715. $SS(BS(SS))SS$
716. $SS(BSS)(SS)S$
717. $SS(BSS)S(SS)$
723. $SS(AA)SS$
724. $SS(AB)SS$
725. $SS(A(SS))(SS)S$
726. $SS(A(SS))S(SS)$
728. $SS(AS)BS$
733. $SS(S(BS)S)SS$
736. $SS(SAS)(SS)S$
738. $SS(S(SA)S)SS$
739. $SS(S(SB)S)SS$
740. $SS(SB(SS))SS$
741. $SS(SBS)(SS)S$
742. $SS(SBS)S(SS)$
744. $SS(BB)SS$
746. $SS(B(SS))S(SS)$
747. $SS(BS)AS$
748. $SS(BS)BS$
749. $SS(BS)(SS)(SS)$
762. $SS(SS(SS))BS$
767. $SSA(BS)S$
769. $SSAA(SS)$
770. $SSA(SA)S$
771. $SSA(SB)S$
772. $SSAB(SS)$
781. $SS(S(BSS))SS$
783. $SS(S(A(SS)))SS$
787. $SS(S(SBS))SS$
788. $SS(S(B(SS)))SS$
790. $SS(S(BS))S(SS)$
796. $SS(SA)BS$
798. $SS(SA)SA$
799. $SS(SA)SB$
801. $SS(S(S(BS)))SS$
803. $SS(S(SA))(SS)S$
804. $SS(S(SA))S(SS)$
805. $SS(S(S(SA)))SS$
810. $SS(SB)BS$
815. $SSB(BS)S$

817. $SSBA(SS)$
818. $SSB(SA)S$
819. $SSB(SB)S$
820. $SSBB(SS)$
829. $SS(SS)(BSS)S$
831. $SS(SS)(A(SS))S$
832. $SS(SS)(AS)(SS)$
833. $SS(SS)(SAS)S$
834. $SS(SS)(SBS)S$
835. $SS(SS)(B(SS))S$
836. $SS(SS)(BS)(SS)$
839. $SS(SS)(SS(SS))(SS)$
840. $SS(SS)AA$
841. $SS(SS)AB$
842. $SS(SS)(S(AS))S$
843. $SS(SS)(S(BS))S$
844. $SS(SS)(S(SS(SS)))S$
845. $SS(SS)(SA)(SS)$
846. $SS(SS)(S(SA))S$
847. $SS(SS)(S(SB))S$
848. $SS(SS)(SB)(SS)$
849. $SS(SS)BA$
850. $SS(SS)BB$
852. $SS(SS)(SS)(BS)$
871. $A(BSSS)S$
873. $A(A(SS)S)S$
874. $A(AS(SS))S$
875. $A(ASS)(SS)$
876. $A(SASS)S$
877. $A(SBSS)S$
878. $A(B(SS)S)S$
879. $A(BS(SS))S$
880. $A(BSS)(SS)$
882. $A(SSBS)S$
883. $A(SS(SS)(SS))S$
884. $A(SS(SS)S)(SS)$
885. $A(AA)S$
886. $A(AB)S$
887. $A(A(SS))(SS)$
888. $A(AS)A$
889. $A(AS)B$
890. $A(S(AS)S)S$
891. $A(S(BS)S)S$
892. $A(S(SS(SS))S)S$
893. $A(SA(SS))S$
894. $A(SAS)(SS)$
895. $A(S(SA)S)S$
896. $A(S(SB)S)S$
897. $A(SB(SS))S$
898. $A(SBS)(SS)$
899. $A(BA)S$
900. $A(BB)S$
901. $A(B(SS))(SS)$
902. $A(BS)A$
903. $A(BS)B$
905. $A(SS(BS))S$
907. $A(SSA)(SS)$
908. $A(SS(SA))S$
909. $A(SS(SB))S$
910. $A(SSB)(SS)$
911. $A(SS(SS))A$
912. $A(SS(SS))B$
913. $AA(AS)$
914. $AA(BS)$
915. $AA(SS(SS))$
916. $AA(SA)$
917. $AA(SB)$
918. $A(S(ASS))S$
919. $A(S(BSS))S$
920. $A(S(SS(SS)S))S$
921. $A(S(A(SS)))S$
922. $A(S(AS))(SS)$
923. $A(S(SAS))S$
924. $A(S(SBS))S$
925. $A(S(B(SS)))S$
926. $A(S(BS))(SS)$
927. $A(S(SSA))S$
928. $A(S(SSB))S$
929. $A(S(SS(SS)))(SS)$
930. $A(SA)A$
931. $A(SA)B$
932. $A(S(S(AS)))S$
933. $A(S(S(BS)))S$
934. $A(S(S(SS(SS))))S$
935. $A(S(SA))(SS)$
936. $A(S(S(SA)))S$
937. $A(S(S(SB)))S$
938. $A(S(SB))(SS)$
939. $A(SB)A$
940. $A(SB)B$
941. $AB(AS)$
942. $AB(BS)$
943. $AB(SS(SS))$
944. $AB(SA)$
945. $AB(SB)$
947. $A(SS)(BSS)$
949. $A(SS)(A(SS))$
950. $A(SS)(SAS)$
952. $A(SS)(B(SS))$
1003. $S(BSSSS)SS$
1005. $S(A(SS)SS)SS$
1006. $S(AS(SS)S)SS$
1007. $S(ASS(SS))SS$
1010. $S(SASSS)SS$
1011. $S(SBSSS)SS$
1012. $S(B(SS)SS)SS$
1013. $S(BS(SS)S)SS$
1014. $S(BSS(SS))SS$
1015. $S(BSSS)(SS)S$
1016. $S(BSSS)S(SS)$
1018. $S(SSBSS)SS$
1019. $S(SS(SS)(SS)S)SS$
1020. $S(SS(SS)S(SS))SS$
1023. $S(AAS)SS$
1024. $S(ABS)SS$
1025. $S(A(SS)(SS))SS$
1026. $S(A(SS)S)(SS)S$
1027. $S(A(SS)S)S(SS)$
1028. $S(ASA)SS$
1029. $S(ASB)SS$
1030. $S(AS(SS))(SS)S$
1031. $S(AS(SS))S(SS)$
1035. $S(ASS)SA$
1036. $S(ASS)SB$
1037. $S(S(AS)SS)SS$
1038. $S(S(BS)SS)SS$
1039. $S(S(SS(SS))SS)SS$
1040. $S(SA(SS)S)SS$
1041. $S(SAS(SS))SS$
1042. $S(SASS)(SS)S$
1043. $S(SASS)S(SS)$
1044. $S(S(SA)SS)SS$
1045. $S(S(SB)SS)SS$
1046. $S(SB(SS)S)SS$
1047. $S(SBS(SS))SS$
1048. $S(SBSS)(SS)S$
1049. $S(SBSS)S(SS)$
1050. $S(BAS)SS$
1051. $S(BBS)SS$
1052. $S(B(SS)(SS))SS$
1053. $S(B(SS)S)(SS)S$
1054. $S(B(SS)S)S(SS)$
1055. $S(BSA)SS$
1056. $S(BSB)SS$
1057. $S(BS(SS))(SS)S$
1058. $S(BS(SS))S(SS)$
1059. $S(BSS)AS$
1060. $S(BSS)BS$
1061. $S(BSS)(SS)(SS)$
1062. $S(BSS)SA$
1063. $S(BSS)SB$
1065. $S(SS(BS)S)SS$
1067. $S(SSA(SS))SS$
1071. $S(SS(SB)S)SS$
1072. $S(SSB(SS))SS$
1074. $S(SSBS)S(SS)$
1075. $S(SS(SS)A)SS$
1076. $S(SS(SS)B)SS$
1077. $S(SS(SS)(SS))(SS)S$
1078. $S(SS(SS)(SS))S(SS)$
1082. $S(SS(SS)S)SA$
1083. $S(SS(SS)S)SB$
1084. $S(A(AS))SS$
1085. $S(A(BS))SS$
1086. $S(A(SS(SS)))SS$
1087. $S(AA)(SS)S$
1088. $S(AA)S(SS)$
1089. $S(A(SA))SS$
1090. $S(A(SB))SS$
1091. $S(AB)(SS)S$
1092. $S(AB)S(SS)$
1093. $S(A(SS))AS$
1094. $S(A(SS))BS$
1095. $S(A(SS))(SS)(SS)$
1096. $S(A(SS))SA$
1097. $S(A(SS))SB$
1101. $S(AS)A(SS)$
1104. $S(AS)B(SS)$
1105. $S(AS)(SS)A$
1106. $S(AS)(SS)B$
1107. $S(AS)S(AS)$
1108. $S(AS)S(BS)$
1109. $S(AS)S(SS(SS))$
1110. $S(AS)S(SA)$
1111. $S(AS)S(SB)$
1113. $S(S(BSS)S)SS$

1115. $S(S(A(SS))S)SS$
1116. $S(S(AS)(SS))SS$
1118. $S(S(AS)S)S(SS)$
1119. $S(S(SAS)S)SS$
1120. $S(S(SBS)S)SS$
1121. $S(S(B(SS))S)SS$
1122. $S(S(BS)(SS))SS$
1123. $S(S(BS)S)(SS)S$
1124. $S(S(BS)S)S(SS)$
1126. $S(S(SSB)S)SS$
1127. $S(S(SS(SS))(SS))SS$
1129. $S(S(SS(SS))S)S(SS)$
1130. $S(SAA)SS$
1131. $S(SAB)SS$
1132. $S(SA(SS))(SS)S$
1133. $S(SA(SS))S(SS)$
1135. $S(SAS)BS$
1136. $S(SAS)(SS)(SS)$
1137. $S(SAS)SA$
1138. $S(SAS)SB$
1139. $S(S(S(AS))S)SS$
1140. $S(S(S(BS))S)SS$
1141. $S(S(S(SS(SS)))S)SS$
1142. $S(S(SA)(SS))SS$
1143. $S(S(SA)S)(SS)S$
1144. $S(S(SA)S)S(SS)$
1145. $S(S(S(SA))S)SS$
1146. $S(S(S(SB))S)SS$
1147. $S(S(SB)(SS))SS$
1148. $S(S(SB)S)(SS)S$
1149. $S(S(SB)S)S(SS)$
1150. $S(SBA)SS$
1151. $S(SBB)SS$
1152. $S(SB(SS))(SS)S$
1153. $S(SB(SS))S(SS)$
1154. $S(SBS)AS$
1155. $S(SBS)BS$
1156. $S(SBS)(SS)(SS)$
1157. $S(SBS)SA$
1158. $S(SBS)SB$
1159. $S(B(AS))SS$
1160. $S(B(BS))SS$
1161. $S(B(SS(SS)))SS$
1162. $S(BA)(SS)S$
1163. $S(BA)S(SS)$
1164. $S(B(SA))SS$
1165. $S(B(SB))SS$
1166. $S(BB)(SS)S$
1167. $S(BB)S(SS)$
1168. $S(B(SS))AS$
1169. $S(B(SS))BS$
1170. $S(B(SS))(SS)(SS)$
1171. $S(B(SS))SA$
1172. $S(B(SS))SB$
1174. $S(BS)(BS)S$
1176. $S(BS)A(SS)$
1177. $S(BS)(SA)S$
1179. $S(BS)B(SS)$
1180. $S(BS)(SS)A$
1181. $S(BS)(SS)B$
1182. $S(BS)S(AS)$
1183. $S(BS)S(BS)$
1184. $S(BS)S(SS(SS))$
1185. $S(BS)S(SA)$
1186. $S(BS)S(SB)$
1188. $S(SS(BSS))SS$
1190. $S(SS(A(SS)))SS$
1194. $S(SS(SBS))SS$
1195. $S(SS(B(SS)))SS$
1197. $S(SS(BS))S(SS)$
1205. $S(SSA)SA$
1206. $S(SSA)SB$
1208. $S(SS(S(BS)))SS$
1210. $S(SS(SA))(SS)S$
1211. $S(SS(SA))S(SS)$
1212. $S(SS(S(SA)))SS$
1219. $S(SSB)SA$
1220. $S(SSB)SB$
1224. $S(SS(SS))A(SS)$
1227. $S(SS(SS))B(SS)$
1228. $S(SS(SS))(SS)A$
1229. $S(SS(SS))(SS)B$
1230. $S(SS(SS))S(AS)$
1231. $S(SS(SS))S(BS)$
1232. $S(SS(SS))S(SS(SS))$
1233. $S(SS(SS))S(SA)$
1234. $S(SS(SS))S(SB)$
1239. $SA(AS)(SS)$
1243. $SA(BS)(SS)$
1246. $SA(SS(SS))(SS)$
1247. $SAAA$
1248. $SAAB$
1252. $SA(SA)(SS)$
1255. $SA(SB)(SS)$
1256. $SABA$
1257. $SABB$
1258. $SA(SS)(AS)$
1259. $SA(SS)(BS)$
1260. $SA(SS)(SS(SS))$
1261. $SA(SS)(SA)$
1262. $SA(SS)(SB)$
1263. $SAS(ASS)$
1264. $SAS(BSS)$
1265. $SAS(SS(SS)S)$
1266. $SAS(A(SS))$
1267. $SAS(SAS)$
1268. $SAS(SBS)$
1269. $SAS(B(SS))$
1270. $SAS(SSA)$
1271. $SAS(SSB)$
1272. $SAS(S(AS))$
1273. $SAS(S(BS))$
1274. $SAS(S(SS(SS)))$
1275. $SAS(S(SA))$
1276. $SAS(S(SB))$
1278. $S(S(BSSS))SS$
1280. $S(S(A(SS)S))SS$
1281. $S(S(AS(SS)))SS$
1282. $S(S(ASS))(SS)S$
1283. $S(S(ASS))S(SS)$
1284. $S(S(SASS))SS$
1285. $S(S(SBSS))SS$
1286. $S(S(B(SS)S))SS$
1287. $S(S(BS(SS)))SS$
1288. $S(S(BSS))(SS)S$
1289. $S(S(BSS))S(SS)$
1291. $S(S(SSBS))SS$
1292. $S(S(SS(SS)(SS)))SS$
1293. $S(S(SS(SS)S))(SS)S$
1294. $S(S(SS(SS)S))S(SS)$
1295. $S(S(AA))SS$
1296. $S(S(AB))SS$
1297. $S(S(A(SS)))(SS)S$
1298. $S(S(A(SS)))S(SS)$
1299. $S(S(AS))AS$
1300. $S(S(AS))BS$
1301. $S(S(AS))(SS)(SS)$
1302. $S(S(AS))SA$
1303. $S(S(AS))SB$
1304. $S(S(S(AS)S))SS$
1305. $S(S(S(BS)S))SS$
1306. $S(S(S(SS(SS))S))SS$
1307. $S(S(SA(SS)))SS$
1308. $S(S(SAS))(SS)S$
1309. $S(S(SAS))S(SS)$
1310. $S(S(S(SA)S))SS$
1311. $S(S(S(SB)S))SS$
1312. $S(S(SB(SS)))SS$
1313. $S(S(SBS))(SS)S$
1314. $S(S(SBS))S(SS)$
1315. $S(S(BA))SS$
1316. $S(S(BB))SS$
1317. $S(S(B(SS)))(SS)S$
1318. $S(S(B(SS)))S(SS)$
1319. $S(S(BS))AS$
1320. $S(S(BS))BS$
1321. $S(S(BS))(SS)(SS)$
1322. $S(S(BS))SA$
1323. $S(S(BS))SB$
1325. $S(S(SS(BS)))SS$
1327. $S(S(SSA))(SS)S$
1328. $S(S(SSA))S(SS)$
1329. $S(S(SS(SA)))SS$
1331. $S(S(SSB))(SS)S$
1332. $S(S(SSB))S(SS)$
1333. $S(S(SS(SS)))AS$
1334. $S(S(SS(SS)))BS$
1335. $S(S(SS(SS)))(SS)(SS)$
1336. $S(S(SS(SS)))SA$
1337. $S(S(SS(SS)))SB$
1338. $S(SA)(AS)S$
1339. $S(SA)(BS)S$
1340. $S(SA)(SS(SS))S$
1341. $S(SA)A(SS)$
1342. $S(SA)(SA)S$
1343. $S(SA)(SB)S$
1344. $S(SA)B(SS)$
1345. $S(SA)(SS)A$
1346. $S(SA)(SS)B$
1347. $S(SA)S(AS)$
1348. $S(SA)S(BS)$
1349. $S(SA)S(SS(SS))$
1350. $S(SA)S(SA)$
1351. $S(SA)S(SB)$
1352. $S(S(S(ASS)))SS$
1353. $S(S(S(BSS)))SS$

1354. $S(S(S(SS(SS)S)))SS$
1355. $S(S(S(A(SS))))SS$
1356. $S(S(S(AS)))(SS)S$
1357. $S(S(S(AS)))S(SS)$
1358. $S(S(S(SAS)))SS$
1359. $S(S(S(SBS)))SS$
1360. $S(S(S(B(SS))))SS$
1361. $S(S(S(BS)))(SS)S$
1362. $S(S(S(BS)))S(SS)$
1363. $S(S(S(SSA)))SS$
1364. $S(S(S(SSB)))SS$
1365. $S(S(S(SS(SS))))(SS)S$
1366. $S(S(S(SS(SS))))S(SS)$
1367. $S(S(SA))AS$
1368. $S(S(SA))BS$
1369. $S(S(SA))(SS)(SS)$
1370. $S(S(SA))SA$
1371. $S(S(SA))SB$
1372. $S(S(S(S(AS))))SS$
1373. $S(S(S(S(BS))))SS$
1374. $S(S(S(S(SS(SS)))))SS$
1375. $S(S(S(SA)))(SS)S$
1376. $S(S(S(SA)))S(SS)$
1377. $S(S(S(S(SA))))SS$
1378. $S(S(S(S(SB))))SS$
1379. $S(S(S(SB)))(SS)S$
1380. $S(S(S(SB)))S(SS)$
1381. $S(S(SB))AS$
1382. $S(S(SB))BS$
1383. $S(S(SB))(SS)(SS)$
1384. $S(S(SB))SA$
1385. $S(S(SB))SB$
1389. $S(SB)A(SS)$
1390. $S(SB)(SA)S$
1392. $S(SB)B(SS)$
1393. $S(SB)(SS)A$
1394. $S(SB)(SS)B$
1395. $S(SB)S(AS)$
1396. $S(SB)S(BS)$
1397. $S(SB)S(SS(SS))$
1398. $S(SB)S(SA)$
1399. $S(SB)S(SB)$
1401. $SB(BSS)S$
1403. $SB(A(SS))S$
1404. $SB(AS)(SS)$
1406. $SB(SBS)S$
1407. $SB(B(SS))S$
1408. $SB(BS)(SS)$
1411. $SB(SS(SS))(SS)$
1412. $SBAA$
1413. $SBAB$
1414. $SB(S(AS))S$
1415. $SB(S(BS))S$
1416. $SB(S(SS(SS)))S$
1417. $SB(SA)(SS)$
1418. $SB(S(SA))S$
1419. $SB(S(SB))S$
1420. $SB(SB)(SS)$
1421. $SBBA$
1422. $SBBB$
1424. $SB(SS)(BS)$
1432. $SBS(SAS)$
1443. $B(BSSS)S$
1445. $B(A(SS)S)S$
1486. $BA(BS)$
1519. $B(SS)(BSS)$
1521. $B(SS)(A(SS))$
1522. $B(SS)(SAS)$
1524. $B(SS)(B(SS))$
1533. $BS(BSSS)$
1535. $BS(A(SS)S)$
1538. $BS(SBSS)$
1539. $BS(B(SS)S)$
1540. $BS(BS(SS))$
1544. $BS(AA)$
1545. $BS(AB)$
1546. $BS(S(AS)S)$
1547. $BS(S(BS)S)$
1548. $BS(S(SS(SS))S)$
1549. $BS(SA(SS))$
1550. $BS(S(SA)S)$
1551. $BS(S(SB)S)$
1575. $SS(BSSSS)S$
1577. $SS(A(SS)SS)S$
1582. $SS(SBSSS)S$
1583. $SS(B(SS)SS)S$
1584. $SS(BS(SS)S)S$
1585. $SS(BSS(SS))S$
1586. $SS(BSSS)(SS)$
1592. $SS(AAS)S$
1593. $SS(ABS)S$
1594. $SS(A(SS)(SS))S$
1595. $SS(A(SS)S)(SS)$
1602. $SS(S(BS)SS)S$
1605. $SS(SAS(SS))S$
1607. $SS(S(SA)SS)S$
1608. $SS(S(SB)SS)S$
1609. $SS(SB(SS)S)S$
1610. $SS(SBS(SS))S$
1611. $SS(SBSS)(SS)$
1613. $SS(BBS)S$
1615. $SS(B(SS)S)(SS)$
1616. $SS(BSA)S$
1617. $SS(BSB)S$
1618. $SS(BS(SS))(SS)$
1636. $SS(A(BS))S$
1638. $SS(AA)(SS)$
1639. $SS(A(SA))S$
1640. $SS(A(SB))S$
1641. $SS(AB)(SS)$
1650. $SS(S(BSS)S)S$
1652. $SS(S(A(SS))S)S$
1656. $SS(S(SBS)S)S$
1657. $SS(S(B(SS))S)S$
1659. $SS(S(BS)S)(SS)$
1667. $SS(SAS)A$
1668. $SS(SAS)B$
1670. $SS(S(S(BS))S)S$
1672. $SS(S(SA)(SS))S$
1673. $SS(S(SA)S)(SS)$
1674. $SS(S(S(SA))S)S$
1872. $A(BSSSS)$
1874. $A(A(SS)SS)$
2004. $S(BSSSSS)S$
2006. $S(A(SS)SSS)S$
2012. $S(SBSSSS)S$
2013. $S(B(SS)SSS)S$
2014. $S(BS(SS)SS)S$
2015. $S(BSS(SS)S)S$
2016. $S(BSSS(SS))S$
2017. $S(BSSSS)(SS)$
2024. $S(AASS)S$
2025. $S(ABSS)S$
2026. $S(A(SS)(SS)S)S$
2027. $S(A(SS)S(SS))S$
2028. $S(A(SS)SS)(SS)$
2039. $S(S(BS)SSS)S$
2042. $S(SAS(SS)S)S$
2045. $S(S(SA)SSS)S$
2046. $S(S(SB)SSS)S$
2047. $S(SB(SS)SS)S$
2048. $S(SBS(SS)S)S$
2049. $S(SBSS(SS))S$
2052. $S(BBSS)S$
2054. $S(B(SS)S(SS))S$
2056. $S(BSAS)S$
2057. $S(BSBS)S$
2058. $S(BS(SS)(SS))S$
2086. $S(A(BS)S)S$
2088. $S(AA(SS))S$
2090. $S(A(SA)S)S$
2091. $S(A(SB)S)S$
2092. $S(AB(SS))S$
2114. $S(S(BSS)SS)S$
2116. $S(S(A(SS))SS)S$
2121. $S(S(SBS)SS)S$
2122. $S(S(B(SS))SS)S$
2124. $S(S(BS)S(SS))S$
2135. $S(SASA)S$
2136. $S(SASB)S$
2141. $S(S(S(BS))SS)S$
2143. $S(S(SA)(SS)S)S$
2144. $S(S(SA)S(SS))S$
2146. $S(S(S(SA))SS)S$
2576. $S(S(BSSSS))S$
2578. $S(S(A(SS)SS))S$
2873. $B(BSSSS)$
2875. $B(A(SS)SS)$
3005. $SS(BSSSSS)$
3007. $SS(A(SS)SSS)$
3012. $SS(SBSSSS)$
3013. $SS(B(SS)SSS)$
3014. $SS(BS(SS)SS)$
3015. $SS(BSS(SS)S)$
3016. $SS(BSSS(SS))$
3022. $SS(AASS)$
3023. $SS(ABSS)$
3024. $SS(A(SS)(SS)S)$
3025. $SS(A(SS)S(SS))$
3032. $SS(S(BS)SSS)$
3035. $SS(SAS(SS)S)$
3037. $SS(S(SA)SSS)$
3038. $SS(S(SB)SSS)$
3039. $SS(SB(SS)SS)$
3040. $SS(SBS(SS)S)$
3041. $SS(SBSS(SS))$

3043. $SS(BBSS)$
3045. $SS(B(SS)S(SS))$
3046. $SS(BSAS)$
3047. $SS(BSBS)$
3048. $SS(BS(SS)(SS))$
3066. $SS(A(BS)S)$
3068. $SS(AA(SS))$
3069. $SS(A(SA)S)$
3070. $SS(A(SB)S)$
3071. $SS(AB(SS))$
3080. $SS(S(BSS)SS)$
3082. $SS(S(A(SS))SS)$
3086. $SS(S(SBS)SS)$
3087. $SS(S(B(SS))SS)$
3089. $SS(S(BS)S(SS))$
3097. $SS(SASA)$
3098. $SS(SASB)$
3100. $SS(S(S(BS))SS)$
3102. $SS(S(SA)(SS)S)$
3103. $SS(S(SA)S(SS))$
3104. $SS(S(S(SA))SS)$
3302. $SS(S(BSSSS))$
3304. $SS(S(A(SS)SS))$
3434. $S(BSSSSSS)$
3436. $S(A(SS)SSSS)$
3437. $S(AS(SS)SSS)$
3438. $S(ASS(SS)SS)$
3441. $S(SASSSSS)$
3442. $S(SBSSSSS)$
3443. $S(B(SS)SSSS)$
3444. $S(BS(SS)SSS)$
3445. $S(BSS(SS)SS)$
3446. $S(BSSS(SS)S)$
3447. $S(BSSSS(SS))$
3449. $S(SSBSSSS)$
3450. $S(SS(SS)(SS)SSS)$
3451. $S(SS(SS)S(SS)SS)$
3454. $S(AASSS)$
3455. $S(ABSSS)$
3456. $S(A(SS)(SS)SS)$
3457. $S(A(SS)S(SS)S)$
3458. $S(A(SS)SS(SS))$
3460. $S(ASBSS)$
3461. $S(AS(SS)(SS)S)$
3462. $S(AS(SS)S(SS))$
3464. $S(ASSBS)$
3468. $S(S(AS)SSSS)$
3469. $S(S(BS)SSSS)$
3470. $S(S(SS(SS))SSSS)$
3472. $S(SAS(SS)SS)$
3473. $S(SASS(SS)S)$
3474. $S(SASSS(SS))$
3475. $S(S(SA)SSSS)$
3476. $S(S(SB)SSSS)$
3477. $S(SB(SS)SSS)$
3478. $S(SBS(SS)SS)$
3479. $S(SBSS(SS)S)$
3480. $S(SBSSS(SS))$
3481. $S(BASSS)$
3482. $S(BBSSS)$
3484. $S(B(SS)S(SS)S)$
3485. $S(B(SS)SS(SS))$
3486. $S(BSASS)$
3487. $S(BSBSS)$
3488. $S(BS(SS)(SS)S)$
3489. $S(BS(SS)S(SS))$
3490. $S(BSSAS)$
3491. $S(BSSBS)$
3492. $S(BSS(SS)(SS))$
3493. $S(BSSSA)$
3494. $S(BSSSB)$
3496. $S(SS(BS)SSS)$
3498. $S(SSA(SS)SS)$
3503. $S(SSB(SS)SS)$
3507. $S(SS(SS)BSS)$
3508. $S(SS(SS)(SS)(SS)S)$
3509. $S(SS(SS)(SS)S(SS))$
3511. $S(SS(SS)SBS)$
3515. $S(A(AS)SS)$
3516. $S(A(BS)SS)$
3517. $S(A(SS(SS))SS)$
3518. $S(AA(SS)S)$
3519. $S(AAS(SS))$
3520. $S(A(SA)SS)$
3521. $S(A(SB)SS)$
3522. $S(AB(SS)S)$
3523. $S(ABS(SS))$
3524. $S(A(SS)AS)$
3525. $S(A(SS)BS)$
3526. $S(A(SS)(SS)(SS))$
3527. $S(A(SS)SA)$
3528. $S(A(SS)SB)$
3530. $S(AS(BS)S)$
3532. $S(ASA(SS))$
3533. $S(AS(SA)S)$
3535. $S(ASB(SS))$
3544. $S(S(BSS)SSS)$
3546. $S(S(A(SS))SSS)$
3548. $S(S(AS)S(SS)S)$
3550. $S(S(SAS)SSS)$
3551. $S(S(SBS)SSS)$
3552. $S(S(B(SS))SSS)$
3553. $S(S(BS)(SS)SS)$
3554. $S(S(BS)S(SS)S)$
3555. $S(S(BS)SS(SS))$
3559. $S(S(SS(SS))S(SS)S)$
3562. $S(SABSS)$
3563. $S(SA(SS)(SS)S)$
3564. $S(SA(SS)S(SS))$
3565. $S(SASAS)$
3566. $S(SASBS)$
3567. $S(SAS(SS)(SS))$
3570. $S(S(S(AS))SSS)$
3571. $S(S(S(BS))SSS)$
3572. $S(S(S(SS(SS)))SSS)$
3573. $S(S(SA)(SS)SS)$
3574. $S(S(SA)S(SS)S)$
3575. $S(S(SA)SS(SS))$
3576. $S(S(S(SA))SSS)$
3577. $S(S(S(SB))SSS)$
3578. $S(S(SB)(SS)SS)$
3579. $S(S(SB)S(SS)S)$
3580. $S(S(SB)SS(SS))$
3581. $S(SBASS)$
3582. $S(SBBSS)$
3584. $S(SB(SS)S(SS))$
3585. $S(SBSAS)$
3586. $S(SBSBS)$
3587. $S(SBS(SS)(SS))$
3588. $S(SBSSA)$
3589. $S(SBSSB)$
3591. $S(B(BS)SS)$
3593. $S(BA(SS)S)$
3594. $S(BAS(SS))$
3595. $S(B(SA)SS)$
3596. $S(B(SB)SS)$
3597. $S(BB(SS)S)$
3598. $S(BBS(SS))$
3600. $S(B(SS)BS)$
3601. $S(B(SS)(SS)(SS))$
3602. $S(B(SS)SA)$
3603. $S(B(SS)SB)$
3604. $S(BS(AS)S)$
3605. $S(BS(BS)S)$
3606. $S(BS(SS(SS))S)$
3607. $S(BSA(SS))$
3608. $S(BS(SA)S)$
3609. $S(BS(SB)S)$
3610. $S(BSB(SS))$
3611. $S(BS(SS)A)$
3612. $S(BS(SS)B)$
3619. $S(SS(BSS)SS)$
3621. $S(SS(A(SS))SS)$
3634. $S(SSABS)$
3648. $S(SSBBS)$
3653. $S(SS(SS)(BS)S)$
3655. $S(SS(SS)A(SS))$
3656. $S(SS(SS)(SA)S)$
3658. $S(SS(SS)B(SS))$
3667. $S(A(BSS)S)$
3669. $S(A(A(SS))S)$
3670. $S(A(AS)(SS))$
3671. $S(A(SAS)S)$
3672. $S(A(SBS)S)$
3673. $S(A(B(SS))S)$
3674. $S(A(BS)(SS))$
3676. $S(A(SSB)S)$
3677. $S(A(SS(SS))(SS))$
3678. $S(AAA)$
3679. $S(AAB)$
3680. $S(A(S(AS))S)$
3681. $S(A(S(BS))S)$
3682. $S(A(S(SS(SS)))S)$
3683. $S(A(SA)(SS))$
3684. $S(A(S(SA))S)$
3685. $S(A(S(SB))S)$
3686. $S(A(SB)(SS))$
3687. $S(ABA)$
3688. $S(ABB)$
3709. $S(S(BSSS)SS)$
3711. $S(S(A(SS)S)SS)$
3712. $S(S(AS(SS))SS)$
3715. $S(S(SASS)SS)$
3716. $S(S(SBSS)SS)$
3717. $S(S(B(SS)S)SS)$
3718. $S(S(BS(SS))SS)$

3719. $S(S(BSS)(SS)S)$
3720. $S(S(BSS)S(SS))$
3723. $S(S(SS(SS)(SS))SS)$
3726. $S(S(AA)SS)$
3727. $S(S(AB)SS)$
3728. $S(S(A(SS))(SS)S)$
3729. $S(S(A(SS))S(SS))$
3733. $S(S(AS)SA)$
3734. $S(S(AS)SB)$
3736. $S(S(S(BS)S)SS)$
3738. $S(S(SA(SS))SS)$
3740. $S(S(SAS)S(SS))$
3741. $S(S(S(SA)S)SS)$
3742. $S(S(S(SB)S)SS)$
3743. $S(S(SB(SS))SS)$
3744. $S(S(SBS)(SS)S)$
3745. $S(S(SBS)S(SS))$
3746. $S(S(BA)SS)$
3747. $S(S(BB)SS)$
3748. $S(S(B(SS))(SS)S)$
3749. $S(S(B(SS))S(SS))$
3752. $S(S(BS)(SS)(SS))$
3753. $S(S(BS)SA)$
3754. $S(S(BS)SB)$
3767. $S(S(SS(SS))SA)$
3768. $S(S(SS(SS))SB)$
3772. $S(SAA(SS))$
3775. $S(SAB(SS))$
3776. $S(SA(SS)A)$
3777. $S(SA(SS)B)$
3778. $S(SAS(AS))$
3779. $S(SAS(BS))$
3780. $S(SAS(SS(SS)))$
3781. $S(SAS(SA))$
3782. $S(SAS(SB))$
3784. $S(S(S(BSS))SS)$
3786. $S(S(S(A(SS)))SS)$
3787. $S(S(S(AS))(SS)S)$
3788. $S(S(S(AS))S(SS))$
3789. $S(S(S(SAS))SS)$
3790. $S(S(S(SBS))SS)$
3791. $S(S(S(B(SS)))SS)$
3792. $S(S(S(BS))(SS)S)$
3793. $S(S(S(BS))S(SS))$
3796. $S(S(S(SS(SS)))(SS)S)$
3797. $S(S(S(SS(SS)))S(SS))$
3798. $S(S(SA)AS)$
3799. $S(S(SA)BS)$
3800. $S(S(SA)(SS)(SS))$
3801. $S(S(SA)SA)$
3802. $S(S(SA)SB)$
3803. $S(S(S(S(AS)))SS)$
3804. $S(S(S(S(BS)))SS)$
3805. $S(S(S(S(SS(SS))))SS)$
3806. $S(S(S(SA))(SS)S)$
3807. $S(S(S(SA))S(SS))$
3808. $S(S(S(S(SA)))SS)$
3809. $S(S(S(S(SB)))SS)$
3810. $S(S(S(SB))(SS)S)$
3811. $S(S(S(SB))S(SS))$
3815. $S(S(SB)SA)$
3816. $S(S(SB)SB)$
3820. $S(SBA(SS))$
3821. $S(SB(SA)S)$
3823. $S(SBB(SS))$
3863. $S(BS(SAS))$
3874. $S(SS(BSSS)S)$
3876. $S(SS(A(SS)S)S)$
4006. $S(S(BSSSS)S)$
4008. $S(S(A(SS)SS)S)$
4303. $S(SS(BSSSS))$
4305. $S(SS(A(SS)SS))$
4435. $S(S(BSSSSS))$
4437. $S(S(A(SS)SSS))$
4442. $S(S(SBSSSS))$
4443. $S(S(B(SS)SSS))$
4444. $S(S(BS(SS)SS))$
4445. $S(S(BSS(SS)S))$
4446. $S(S(BSSS(SS)))$
4452. $S(S(AASS))$
4453. $S(S(ABSS))$
4454. $S(S(A(SS)(SS)S))$
4455. $S(S(A(SS)S(SS)))$
4462. $S(S(S(BS)SSS))$
4465. $S(S(SAS(SS)S))$
4467. $S(S(S(SA)SSS))$
4468. $S(S(S(SB)SSS))$
4469. $S(S(SB(SS)SS))$
4470. $S(S(SBS(SS)S))$
4471. $S(S(SBSS(SS)))$
4473. $S(S(BBSS))$
4475. $S(S(B(SS)S(SS)))$
4476. $S(S(BSAS))$
4477. $S(S(BSBS))$
4478. $S(S(BS(SS)(SS)))$
4496. $S(S(A(BS)S))$
4498. $S(S(AA(SS)))$
4499. $S(S(A(SA)S))$
4500. $S(S(A(SB)S))$
4501. $S(S(AB(SS)))$
4510. $S(S(S(BSS)SS))$
4512. $S(S(S(A(SS))SS))$
4516. $S(S(S(SBS)SS))$
4517. $S(S(S(B(SS))SS))$
4519. $S(S(S(BS)S(SS)))$
4527. $S(S(SASA))$
4528. $S(S(SASB))$
4530. $S(S(S(S(BS))SS))$
4532. $S(S(S(SA)(SS)S))$
4533. $S(S(S(SA)S(SS)))$
4534. $S(S(S(S(SA))SS))$
4732. $S(S(S(BSSSS)))$
4734. $S(S(S(A(SS)SS)))$

# Αναφορές


[1] F. Baader και Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, Cambridge, U.K., 1998.

[2] Leo Bachmair, Nachum Dershowitz, και Jieh Hsiang. Orderings for equational proofs. Στο *Proceedings of the 1st IEEE Symposium on Logic in Computer Science, Cambridge, MA, USA*, σελίδες 346–357, Ιούνιος 1986.

[3] Chris Barker. Ιστοσελίδα "Iota and Jot : the simplest languages ?". `http://ling.ucsd.edu/~barker/Iota/`, 2000.

[4] Lewis Denver Baxter. A practically linear unification algorithm. Τεχνική αναφορά CS-76-13, University of Waterloo, Canada, 1976.

[5] Garrett Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31 :433–454, 1935.

[6] Garrett Birkhoff. *Lattice Theory.* American Mathematical Society, 1948.

[7] Bruno Bogaert και Sophie Tison. Equality and disequality constraints on direct subterms in tree automata. Στο Alain Finkel και Matthias Jantzen, επιμελητές, *Proceedings of 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS 92)*, τόμος 577 του *Lecture Notes in Computer Science*, σελίδες 161–171, Cachan, France, 13–15 Φεβρουαρίου 1992. Springer.

[8] Béla Bollobás. *Modern Graph Theory.* Springer-Verlag, New York, 1998.

[9] Walter S. Brainerd. Tree generating regular systems. *Information and Computation*, 14 :217–231, 1969.

[10] Ιστοσελίδα btyacc 3.0. `http://www.siber.org/btyacc/`, 1999.

[11] J. R. Büchi. On a decision method in a restricted second order arithmetic. Στο *Proceedings of International Congress on Logic, Methodology and Philosophy of Science*, 1960.

[12] Πρότυπο για την γλώσσα προγραμματισμού C, ISO/IEC 9899 :1990(E), 1990.

[13] Eugène Charles Catalan. Note sur une Équation aux différences finies. *Journal de Mathématiques Pures et Appliquées*, 3 :508–516, 1838.

[14] A. K. Chandra, D. C. Kozen, και L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1) :114–133, Ιανουάριος 1981.

[15] Gary Chartrand. *Introductory Graph Theory*. Dover Publications, New York, 1985.

[16] Philippe Le Chénadec. Canonical forms in finitely presented algebras. Στο *Conference on Automated Deduction*, σελίδες 142–165, 1984.

[17] Alonso Church. Logic, arithmetic, automata. Στο *Proceedings of International Mathematical Congress*, 1962.

[18] Alonzo Church. The calculi of lambda conversion. *Annals of Mathematics Studies*, 6, 1941.

[19] Hubert Comon. Sequentiality, second-order monadic logic and tree automata. Στο *Proceedings of the 10th IEEE Symposium on Logic in Computer Science (LICS'95), San Diego, CA, USA*, σελίδες 508–517, Ιούνιος 1995.

[20] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, και Marc Tommasi. *Tree automata : techniques and applications*. Διαθέσιμο στην διεύθυνση : `http://www.grappa.univ-lille3.fr/tata`, 1999.

[21] J.-L. Coquidé και Rémi Gilleron. Proofs and reachability problem for ground rewrite systems. Στο *Proceedings of the 6th International Meeting of Young Computer Scientists*, τόμος 464 του *Lecture Notes in Computer Science*, σελίδες 120–129, Berlin, 1990. Springer-Verlag.

[22] Haskell B. Curry και Robert Feys. *Combinatory Logic*. North Holland-Elsevier Science Publishers, Amsterdam, New York, Oxford, 1958.

[23] Max Dauchet. Simulation of Turing machines by a regular rule. *Theoretical Computer Science*, 103(2) :409–420, 1992.

[24] Max Dauchet, T. Heuillard, Pierre Lescanne, και Sophie Tison. Decidability of the confluence of ground term rewriting systems. Στο *Proceedings of the Second Symposium on Logic in Computer Science (Lille, France)*, σελίδες 353–359, Ithaca, NY, Ιούνιος 1987. IEEE.

[25] Max Dauchet και Sophie Tison. Tree automata and decidability in ground term rewriting systems. Στο *Proceedings FCT*, τόμος 199 του *Lecture Notes in Computer Science*, σελίδες 80–84, Berlin, 1985. Springer-Verlag.

[26] Max Dauchet και Sophie Tison. The theory of ground rewrite systems is decidable. Στο *Proceedings of the 5th IEEE Symposium on Logic in Computer Science, Philadelphia, PA, USA*, σελίδες 242–248, Ιούνιος 1990.

[27] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3) :279–301, 1982.

[28] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1) :69–116, 1987. Παροράματα στο 4(3) :409–410.

[29] Nachum Dershowitz και Charles Hoot. Natural termination. *Theoretical Computer Science*, 142(2) :179–207, 1995.

[30] Nachum Dershowitz και Jean-Pierre Jouannaud. *Rewrite systems*, τόμος B, σελίδες 243–320. North Holland-Elsevier Science Publishers, Amsterdam, New York, Oxford, 1990.

[31] Nachum Dershowitz και David A. Plaisted. Rewriting. τόμος I, κεφάλαιο 9, σελίδες 535–610. North Holland-Elsevier Science Publishers, Amsterdam, New York, Oxford, 2001.

[32] Reinhard Diestel. *Graph Theory*. Springer-Verlag, New York, δεύτερη έκδοση, 2000.

[33] J. E. Doner. Decidability of the weak second-order theory of two succesors. *Notices of the American Mathematical Society*, 12 :365–468, Μάρτιος 1965.

[34] J. E. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4 :406–451, 1970.

[35] Eli. Ιστοσελίδα.
`http://www.cs.colorado.edu/~eliuser`, 2000.

[36] Eli. Τεκμηρίωση.
`http://www.cs.colorado.edu/~eliuser/elionline4.3/`, 2000.

[37] Trevor Evans. On multiplicative systems defined by generators and relations, i. *Proceedings of the Cambridge Philosophical Society*, 47 :637–649, 1951.

[38] Robert W. Floyd. Algorithm 97 : shortest path. *Communications of the ACM*, 5(6) :345, 1962.

[39] Jeroen Fokker. The systematic construction of a one-combinator basis for lambda-terms. *Formal Aspects of Computing*, 2 :776–780, 1992.

[40] Thom Frühwirth, Ehud Shapiro, Moshe Y. Vardi, και Eyal Yardeni. Logic programs as types for logic programs. Στο *Proceedings, 6th Annual IEEE Symposium on Logic in Computer Science*, σελίδες 300–309, 1991.

[41] Z. Fülöp και H. Vögler. *Formal models based on tree transducers.* Monographs in Theoretical Computer Science. Springer-Verlag, 1998.

[42] J. H. Gallier και R. V. Book. Reductions in tree replacement systems. *Theoretical Computer Science*, 37(2) :123–150, 1985.

[43] F. Gécseg και Magnus Steinby. *Tree automata.* Akademiai Kiado, 1984.

[44] F. Gécseg και Magnus Steinby. *Tree languages*, τόμος 3, σελίδες 1–68. Springer-Verlag, 1996.

[45] Alfons Geser. *Relative termination.* Διδακτορική διατριβή, Universität Ulm, 1990.

[46] Alfons Geser. An improved general path order. Τεχνική αναφορά MIP-9407, Universität Passau, Ιούνιος 1994.

[47] Rémi Gilleron και Sophie Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24 :157–176, 1995.

[48] Bernhard Gramlich. *Termination and confluence properties of structured rewrite systems.* Διδακτορική διατριβή, Universität Kaiserslautern, 1996.

[49] Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society (3)*, 2(7) :326–336, 1952.

[50] J. Roger Hindley. *The Church-Rosser property and a result in Combinatory Logic.* Διδακτορική διατριβή, University of Newcastle-upon-Tyne, 1964.

[51] J.E. Hopcroft και J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, USA, 1979.

[52] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4) :797–821, Οκτώβριος 1980.

[53] Gérard Huet. A complete proof of correctness of the Knuth-Bendix completion algorithm. *Journal of Computer and System Sciences*, 23(1) :11–21, 1981.

[54] Gérard Huet και Dallas S. Lankford. On the uniform halting problem for term rewriting systems. Rapport laboria 283, Institut de Recherche en Informatique et en Automatique, Le Chesnay, France, Μάρτιος 1978.

[55] Gérard Huet και D. C. Oppen. Equations and rewrite rules : a survey. Στο *Formal Language Theory : Perspectives and Open Problems*, σελίδες 349–405. Academic Press, New York, 1990.

[56] Florent Jacquemard. Decidable approximations of term rewriting systems. Στο Harald Ganziger, επιμελητής, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, τόμος 1103 του *Lecture Notes in Computer Science*, σελίδες 362–376, Berlin, 1996. Springer-Verlag.

[57] Thomas J. Jech. About the Axiom of Choice. κεφάλαιο B.2, σελίδες 345–370. North Holland-Elsevier Science Publishers, Amsterdam, New York, Oxford, 1977.

[58] Stephen Cole Kleene. Representation of events in nerve nets and finite automata. Στο C. E. Shannon και M. McCarthy, επιμελητές, *Automata Studies*, σελίδες 3–41. Princeton University Press, 1956.

[59] Jan Willem Klop. *Term rewriting systems*, τόμος 2, σελίδες 2–116. Clarendon Press, Oxford, 1992.

[60] Donald E. Knuth και Peter B. Bendix. Simple word problems in universal algebras. Στο J. Leech, επιμελητής, *Computational Problems in Abstract Algebra*, σελίδες 263–297. Pergamon Press, Oxford, U. K., 1970. Ανατύπωση στο *Automation of Reasoning 2*, Springer-Verlag, Berlin, σελίδες 342–376 (1983).

[61] Dexter Kozen. On the Myhill-Nerode theorem for trees. *Bulletin of the European Association for Theoretical Computer Science*, 47 :170–173, 1992.

[62] Joseph B. Kruskal. Well-quasi-ordering, the tree theorem, and vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95(2) :210–225, 1960.

[63] Gregory Kucherov. On relationship between term rewriting systems and regular tree languages. Στο *Proceedings of the 4th International Conference on Rewriting Techniques and Applications*, τόμος 488 του *Lecture Notes in Computer Science*, σελίδες 299–311, Berlin, 1991. Springer-Verlag.

[64] Orna Kupferman, Moshe Y. Vardi, και Pierre Wolper. An automata-theoretic approach to branching-time model checking. Αδημοσίευτο, 1998.

[65] Masahito Kurihara και Ikuo Kaji. Modular term rewriting systems and the termination. *Information Processing Letters*, 34 :1–4, Φεβρουάριος 1990.

[66] Dallas S. Lankford. Some approaches to equality for computational logic : a survey and assessment. Memo ATP-36, University of Texas, Math. Dept., Automatic Theorem Proving Project, Austin, Texas, USA, 1977.

[67] A. Martelli και U. Montanari. An efficient unification algorithm. *Transactions on Programming Languages and Systems*, 4(2) :258–282, Απρίλιος 1982.

[68] Robert F. McNaughton και H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9(1) :39–47, 1960.

[69] Aart Middeldorp. *Modular properties of term rewriting systems.* Διδακτορική διατριβή, Vrije Universiteit, Amsterdam, 1990.

[70] J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG.* Διδακτορική διατριβή, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, 1981.

[71] Yannis N. Moschovakis. *Notes on Set Theory.* Springer-Verlag, Berlin, 1994. Στα ελληνικά : *Σημειώσεις στην Συνολοθεωρία*, Εκδόσεις Νεφέλη, Αθήνα, 1993.

[72] D.E. Muller, A. Saoudi, και P.E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. Στο *Proceedings of the 3rd IEEE Symposium on Logic in Computer Science*, σελίδες 422–427, Edinburgh, Ιούλιος 1988.

[73] D.E. Muller και P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54 :267–276, 1987.

[74] J. Myhill. Finite automata and the representationm of events. Τεχνική αναφορά 57-264, WADC, 1957.

[75] Crispin St. John Alvah Nash-Williams. On well-quasi-ordering finite trees. *Proceedings of the Cambridge Philosophical Society*, 59 :833–835, 1963.

[76] A. Nerode. Linear automata transformations. *Proceedings of the American Mathematical Society*, 9 :541–544, 1958.

[77] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(3) :223–243, 1942.

[78] Odin. Διανομή αρχείων. `ftp://ftp.cs.colorado.edu/pub/distribs/odin/`, 2000.

[79] M. J. O'Donnell. *Computing in systems described by equations*, τόμος 58 του *Lecture Notes in Computer Science*. Springer-Verlag, 1977.

[80] Enno Ohlebusch. *Modular properties of composable term rewriting systems*. Διδακτορική διατριβή, Universität Bielefeld, 1994.

[81] M. Oyamaguchi. The Church-Rosser property for ground term rewriting systems is decidable. *Theoretical Computer Science*, 49(1) :43–79, 1987.

[82] Christos Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, USA, 1994.

[83] M. S. Paterson και M. N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16 :158–167, 1978.

[84] David A. Plaisted. *Equational Reasoning and Term rewriting systems*, τόμος 1, σελίδες 273–364. Clarendon Press, Oxford, 1992.

[85] G. Plotkin. Building in equational theories. Στο B. Meltzer και D. Michie, επιμελητές, *Machine Intelligence 7*, σελίδες 73–90, Edinburgh, Scotland, 1972. Edinburgh University Press.

[86] PStricks. Γραφικά PostScript σε LaTeX. `ftp://ftp.ntua.gr/pub/tex/graphics/pstricks/`, 2001.

[87] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141 :1–35, 1969.

[88] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1) :23–41, Ιανουάριος 1965.

[89] B. K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20 :160–187, 1973.

[90] Kai Salomaa. Deterministic tree pushdown automata and monadic tree rewriting systems. *Journal of Computer and System Sciences*, 37 :367–394, 1988.

[91] Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92 :305–316, 1924.

[92] N.J.A. Sloane. *A Handbook of Integer Sequences*. Academic Press, New York, 1973.

[93] Giora Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41 :305–318, 1985.

[94] Raymond Smullyan. *To Mock a Mockingbird and Other Logic Puzzles Including an Amazing Adventure in Combinatory Logic*. Alfred A. Knopf, New York, 1985.

[95] Murray R. Spiegel. *Schaum's Outline Series : Mathematical Handbook of Formulas and Tables*. McGraw-Hill, New York, 1968.

[96] Richard P. Stanley. *Enumerative Combinatorics*, τόμος 2. Cambridge University Press, Cambridge, U.K., 1999.

[97] Joachim Steinbach. *Termination of rewriting — Extensions, comparison and automatic generation of simplification orderings*. Διδακτορική διατριβή, Universität Kaiserslautern, 1994.

[98] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5 :285–309, 1955.

[99] J. W. Thatcher. Generalized sequential machines. *Journal of Computer and System Sciences*, 4 :339–367, 1970.

[100] J. W. Thatcher. Tree automata : an informal survey. Στο A. V. Aho, επιμελητής, *Currents in the Theory of Computing*, σελίδες 143–178. Prentice-Hall, 1973.

[101] Axel Thue. Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln. *Skrifter utgit av Videnskapsselskapet i Kristiania, Mat.-Nat. Klasse*, 10 :34 κ.ε., 1914.

[102] Boris Trakhtenbrot. Origins and metamorphoses of the trinity : Logic, nets, automata. Στο *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, New York, 1995. IEEE Press.

[103] Vincent van Oostrom. *Confluence for abstract and higher-order rewriting*. Διδακτορική διατριβή, Vrije Universiteit, Amsterdam, 1994.

[104] Johannes Waldmann. *The combinator S*. Διδακτορική διατριβή, University of Jena, 1998.

[105] Johannes Waldmann. The combinator $S$. *Information and Computation*, 159 :2–21, 2000.

[106] Johannes Waldmann. Προσωπική ιστοσελίδα. `http://www.informatik.uni-leipzig.de/~joe/`, 2000.

[107] S. Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1) :11–12, 1962.

[108] Stathis Zachos. *Kombinatorische Logik und S-Terme*. Διδακτορική διατριβή, ETH Zürich, 1978.

[109] Stathis Zachos. *Μοντέλα υπολογισμού, τυπικές γλώσσες, θεωρία αυτομάτων*. Τυπογραφείο Ε.Μ.Π., 1995.

[110] Stathis Zachos. A decision algorithm for $S$-term normalization. Στο *Proceedings of the 3rd Panhellenic Logic Symposium (PLS3)*, Ανώγεια, Κρήτη, 17–21 Ιουλίου 2001.

[111] Stathis Zachos, Stavros Routzounis, και Panos Hilaris. Deciding normalization and computing normal forms for $S$-terms. Στο *Proceedings of the 8th Panhellenic Conference in Informatics*, τόμος 1, σελίδες 187–196, Λευκωσία, Κύπρος, 8–10 Νοεμβρίου 2001.

[112] Hans Zantema. Termination of term rewriting : interpretation and type elimination. *Journal of Symbolic Computation*, 17 :23–50, 1994.

# Μετάφραση όρων

αιτιοκρατικό (σύστημα) : deterministic (system)
αλυσίδα : chain
αναγνωρίσιμος : recognizable
αναγωγή : reduction
ανάγωγο : irreducible
ανάξιμο : redex
αντικατάσταση : substitution
απόγονος : descendant
αποκρισιμότητα : decidability
αριστερογραμμικό (σύστημα μεταγραφής όρων) : left-linear (trs)
αριστερογραμμικός : left-linear
αριστερότερη (στρατηγική) : leftmost (strategy)
ασθενής κανονικοποίηση : weak normalization
αυτοδύναμη (σχέση) : idempotent (relation)
αφηρημένο σύστημα μεταγραφής : abstract rewriting system

βρόχος : loop

γραμματική χωρίς συμφραζόμενα : context-free grammar
γραμμική διάταξη : linear order
γραμμικός : linear

δεξιά αναλλοίωτη (σχέση) : right invariant (relation)
δεξιογραμμικός : right-linear
διαγράφων : erasing
διάδοχος : successor
διάζευξη : disjunction
διακρίσιμη (κατάσταση) : distinguishable (state)
διαμέριση : partition
διάταξη αναγωγής : reduction ordering
διάταξη απλοποίησης : simplification ordering
διάταξη μεταγραφής : rewrite ordering
διάταξη : order

διπλασιάζων : duplicating
δομή : structure

εκτέλεση : run
ελαχιστικό στοιχείο : minimal element
έλεγχος υπέρθεσης : superposition test
εμφάνιση : occurence
εναλλαγή : alternation
ενοποιητής : unifier
εξωτερικότερη (στρατηγική) : outermost (strategy)
επαλήθευση : verification
επιθεματικός (τελεστής) : postfix (operator)
επικάλυψη : cover
εσωτερικότερη (στρατηγική) : innermost (strategy)

ζευξιμότητα : joinability

θεμελιώδες (σύστημα μεταγραφής όρων) : ground (trs)
θέση : position

ιδιότητα μοναδικής κανονικής μορφής : unique normal form property
ιδιότητα της κανονικής μορφής : normal form property
ισχυρή κανονικοποίηση : strong normalization
ισχυρή συμβολή : strong confluence

καλώς εδραιωμένη απεικόνιση : well-founded mapping
καλώς εδραιωμένη επαγωγή : well founded induction
καλώς εδραιωμένη σχέση : well founded relation
κανονική μορφή : normal form
κανονική παράσταση : regular expression
κανονικοποιημένη γραμματική : normalized grammar
κανονικοποιούσα (στρατηγική) : normalizing (strategy)
καταρρέων : collapsing
κλείσιμο (σχέσης) : closure (of a relation)
κλείσιμο : closure
κλειστό ως προς το πρόθεμα σύνολο : prefix closed set
κορυφή : peak
κύκλωμα : circuit

λεκτικό πρόβλημα : word problem
λήμμα μετάντλησης : pumping lemma

μέγεθος : size
μειωμένη γραμματική : reduced grammar
μερική διάταξη : partial order

μεταγραφή : rewriting
μετα-μεταγλωττιστής : meta-compiler
μετατρεψιμότητα : convertibility
μετατροπέας (δένδρων) : (tree) transducer
μετονομασία : renaming
μονοτονικότητα : monotonicity

οκνηρή αποτίμηση : lazy evaluation
ολική διάταξη : total order
ομοδύναμη (σχέση) : idempotent (relation)
ομοιόμορφη συμβολή : uniform confluence
ομοιότητα : literal similarity
οπή : hole

παραγωγή : derivation
παραγωγικό : productive
περιβάλλον : context
περικλείω : encompass, contain
πλήθος ορισμάτων (συναρτησιακού συμβόλου) : arity
πλήρους αντικατάστασης (στρατηγική) : full substitution (rule)
πρόβλημα τερματισμού : halting problem
πρόγονος : ancestor
πρόθεμα : prefix
πρόθυμη αποτίμηση : eager evaluation
προκάτοχος : predecessor
προσβάσιμο : reachable
πρότυπο : pattern

ρομβοειδές λήμμα : diamond lemma
ρομβοειδής ιδιότητα : diamond property

σταθερότητα : stability
στιγμιότυπο : instance
σύζευξη : conjunction
συμβατότητα : compatibility
συμβολή : confluence
συμβολοσειρά : string
συμπερίληψη : subsumption
συμπλήρωση : completion
συμφραζόμενο : context
συναντησιμότητα : meetability
συνδυαστική λογική : combinatorial logic
συνεσταλμένο : contractum

συντακτικό δένδρο : parse tree
σχεδόν διάταξη : quasi ordering
σχέση παραγωγής : derivation relation

ταξινόμηση εισαγωγής : insertion sort
τέρας : monster
τερματισμός : termination
τεχνολόγηση : parsing
τεχνολογητής : parser
τοπική συμβολή : local confluence

υποαντιμεταθετικότητα : subcommutativity
υποόρος : subterm
ύψος : height

ψαρόνι : starling

# Ευρετήριο

Στο παρόν ευρετήριο προτάσσονται οι όροι ή οι συγγραφείς με ελληνικούς χαρακτήρες και ακολουθούν αυτοί με λατινικούς χαρακτήρες. Τα σημεία στα οποία δίνονται οι ορισμοί σημειώνονται με έντονη γραφή.