40 ΠΑΝΕΛΛΗΝΙΟ ΣΥΝΕΔΡΙΟ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΕ ΔΙΕΘΝΗ ΣΥΜΜΕΤΟΧΗ

ΠΡΑΚΤΙΚΑ
ΕΙΣΗΓΗΣΕΩΝ
ΤΟΜΟΣ 1

ΧΟΡΗΓΟΣ ΕΚΔΟΣΗΣ COMPUTER
ΤΟ ΕΛΛΗΝΙΚΟ ΠΕΡΙΟΔΙΚΟ ΓΙΑ BUSINESS COMPUTING

Υπο την αιγίδα του Υπουργείου Προεδρείας της Κυβερνήσεως.

ΕΡΕΥΝΑ
ΑΝΑΠΤΥΞΗ
ΕΦΑΡΜΟΓΕΣ
Βελτίωση της απόδοσης λογικών προγραμμάτων με τη χρήση μετασχηματισμών
Μανώλης Γεργατσούλης  Μαρία Κατζουράκη
Ινστιτούτο Πληροφορικής & Τηλεπικοινωνιών
ΕΚΕΦΕ 'Δημόκριτος', 153 10 Αγία Παρασκευή Αττικής
e-mail: manolis@iit.nrcps.ariadne-t.gr

Περίληψη
Η εργασία αυτή αναφέρεται στη βελτίωση της απόδοσης λογικών προγραμμάτων με τη χρήση μετασχηματισμών. Παρουσιάζεται ένα σύστημα μετασχηματισμών διπλών/ξεδιπλών και εξετάζονται μορφές νέων ορισμών και στρατηγικές εφαρμογής των μετασχηματισμών, εστιάζοντας στη μετατροπή προγραμμάτων που περιλαμβάνουν προτάσεις με μη γραμμική αναδρομή σε ισοδύναμα με γραμμική αναδρομή. Οι προτεινόμενοι νέοι ορισμοί οδηγούν στη γενικότερη λύσας και την εισαγωγή συσσωρευτών.

Δέξεις Κλειδιά: μετασχηματισμοί προγραμμάτων, γενικότερη λύσας, λογικά προγραμματισμός

Abstract
This paper concerns the use of program transformations to improve the efficiency of logic programs. We present a fold/unfold program transformation system and discuss the forms of the new definitions and the transformation strategies, focussing on the transformation of a common class of logic programs which include clauses with nonlinear recursion into equivalent but linear recursive programs. The proposed new definitions and strategies result in tupling generalization and in accumulator introduction.

I. ΕΙΣΑΓΩΓΗ
Οι στόχοι της ανάπτυξης προγραμμάτων τα οποία να είναι από τη μια μεριά σωστά, κατανοητά, και εύχρηστα ομοιοποιήσιμα, και από την άλλη να έχουν μικρές σπατηθίσεις σε μικρή και να είναι αποδοτικά, έχουν συχνά σε αντίθεση μεταξύ τους. Επομένως, ο προγραμματισμός αναγκάζεται πολλές φορές να θυσάστε κόστος από τους στόχους αυτούς για να πετύχετε τους υπόλοιπους. Αυτό έχει σαν συνέπεια να αναπτύσσονται δυναστεία και πολύπλοκα προγράμματα, να αυξάνεται η πιθανότητα λάθων, και να δυσκολεύεται η διόρθωση τους, προκειμένου αυτά τα προγράμματα να είναι αποδοτικά. Η αντίφαση αυτή παρατηρείται τόσο στην ανάπτυξη προγραμμάτων με τις γλώσσες του κλασικού αλγοριθμικού προγραμματισμού και τις συναρτησιακές γλώσσες όσο και στον λογικό προγραμματισμό (logic programming), [29,30] στον οποίο αναφέρεται αυτή η εργασία. Μια βασική μεθοδολογία που στοχεύει στην αντιμετώπιση του προβλήματος αποτελούν οι μετασχηματισμοί προγραμμάτων (program transformations). Με βάση τη μεθοδολογία αυτή, ο προγραμματισμός ανακύκλωσε το πρόγραμμα, συγκεντρώνοντας την προσοχή του στη συνθήκη και την ευκολία κατανόησής του και αδιαφορώντας για τις σπατηθίσεις του σε μικρή και χρόνο. Στη συνέχεια, ένα σύστημα μετασχηματισμών που διατηρούν την ορθότητα (correctness preserving program transformations) αναλαμβάνει τη μετατροπή του προγράμματος, αν είναι δυνατόν με αυτόματο ή ημιαυτόματο τρόπο, σε ισοδύναμο πρόγραμμα το οποίο έχει χαλαρότερη υπολογιστική συμπεριφορά από το αρχικό. Οι μετασχηματισμοί μπορούν να εφαρμοστούν σε διάφορα επίπεδα. Στο επίπεδο του πηγαίου κώδικα (source level), στο οποίο αναφέρομαστε στην εργασία αυτή,
στο επίπεδο χάκποιου ενδιάμεσου κώδικα, ή στο επίπεδο του τελικού κώδικα.

Αξιολογήθηκε κατηγορία μετασχηματισμών είναι οι μετασχηματισμοί διπλών/ξεδιπλών (fold/unfold) οι οποίοι αναπτύχθηκαν υποκείμενα από τους R. Burstall και J. Darlington [7] για συναρτησιακά προγράμματα και στη συνέχεια διατυπώθηκαν από τους H. Tamaki και T. Sato [27,26] για λογικά προγράμματα, έτσι ώστε να διατηρηθούν τη μοντελοθεωρητική σημασία των προγραμμάτων (model theoretic semantics) [15]. Το συζαμαρότερο πρόβλημα στη χρήση των μετασχηματισμών αυτών, οι οποίοι αποτελούνται από ένα πολύ μικρό αλλά δυνατό σύνολο κανόνων, βρίσκεται στο σημαντικό βαθμό "εφιαλτική" που απαιτείται για την εσωτερική νέων ορισμών και την επιλογή της κατάλληλης στρατηγικής για την εφαρμογή τους, αφού η ανεξέλεγκτη χρήση τους οδηγεί σε συνδιαστική έκρηξη. Οι λόγοι αυτοί καθιστούν δύσκολη την πλήρη αυτοματοποίηση τέτοιων συστημάτων. Παράλληλα αυτά, οι μετασχηματισμοί διπλών/ξεδιπλών αποτελούν βασικό συστατικό πολλών εργασιών που αναφέρονται σε βελτίωση της απόδοσης λογικών προγραμμάτων [17,4,5,9,24], μερικό υπολογισμό (partial evaluation) και εξειδίκευση προγράμματος (program specialization) [16,2], ενσωμάτωση πληροφορίας ελέγχου σε λογικά προγράμματα [6] και σύνθεση προγραμμάτων από προδιαγραφές σε λογική 1ες τάξεις [22,13,14,10]. Ανάλογες εργασίες αναφέρονται και για συναρτησιακά και αλγοριθμικά προγράμματα [1,7,28].

Στην εργασία αυτή εξετάζουμε τα αίτια της μειωμένης αποδοτικότητας των λογικών προγραμμάτων (ανάτομα II) και στη συνέχεια εξισώνουμε ένα σύνολο μετασχηματισμών διπλών/ξεδιπλών (ανάτομα III). Οι μετασχηματισμοί αυτοί χρησιμοποιούνται στη συνέχεια (ανάτομα IV έως VII) για να μετασχηματίσουμε προγράμματα που ανήκουν σε μια συγκεκριμένη κατηγορία. Στόχος μας είναι να περιορίζουμε μη-γραμμική αναδρομή που εμφανίζουν τα προγράμματα αυτά. Αυτό οδηγεί συγκάτοιο στην αύξηση της αποδοτικότητας των προγραμμάτων. Σημαντικός για τους μετασχηματισμούς αποδεικνύονται κάποιες διόδιτες που έχουν πολλά κατηγορήματα όπως είναι η ύπαρξη μοναδικών στοιχείων (identity elements) και η προσεκαμνητικότητα (associativity). Βασικό σημείο της προσέγγισης που σχολολουθεί με σχετικά χαρακτηριστικά της αποδοτικότητας με βάση την εφαρμογή των μετασχηματισμών. Τελικώς, στην ενότητα 8 παρουσιάζονται τα σημερισμένα και γίνεται αναφορά σε συναφείς εργασίες.

II. ΣΧΕΤΙΚΑ ΜΕ ΤΗΝ ΑΠΟΔΟΤΙΚΟΤΗΤΑ ΤΩΝ ΛΟΓΙΚΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ

Η σύνταξη λογικών προγραμμάτων με στόχο την αρμότητα και την ευκολία κατανόησης οδηγεί συχνά στη μειωμένη αποδοτικότητα των προγραμμάτων αυτών. Η μικρή αποδοτικότητα οφείλεται και σε άλλους λόγους οι οποίοι σχετίζονται με τα χαρακτηριστικά του λογικού προγραμματισμού. Ανάμεσα στα αίτια του προβλήματος σημαντικά είναι:

- Η διάχυση δομών δεδομένων περισσότερες φορές από όσες αυτές είναι αναγκαίες, καθώς και η δόμηση μη αναγκαίων ενδιάμεσων τιμών.
- Η πολλαπλός υπολογισμός του διών υποστόχου
- Η ύπαρξη πλεονεκτών υπολογισμών (redundant computations) και γενικότερα υπολογισμών που θα μπορούσαν να γίνουν σε χρόνο μετάφρασης
- Η ύπαρξη διαδικασιών της μορφής: λύσε(...) — γέννησε...πωλήσε...λύσε(...) ελέγξε...οφέλησε...λύσε(...)ελέγξε...οφέλησε...λύσε(...)
- Η ουσιαστική θέληση τρόπων έκφρασης επανάληψης όπως οι while και for άλλων γλώσσας, με αποτέλεσμα η επανάληψη να εκφράζεται σχεδόν αποκλειστικά μέσω της αναδρομής
- Η πολύ μικρή δυνατότητα που παρέχεται στον προγραμματιστή να επιδρά στον ελέγχο, καθώς και η "τυφλή" οπισθοδρομή χωρίς εξέταση των αιτιών της αποτυχίας
- Η ιδιότητα των λογικών μεταβλητών να παίρνουν μια μόνο τιμή (single assignment)
Στη βελτίωση της απόδοσης των λογικών προγραμμάτων και τη μείωση των απαιτήσεων τους σε μικρή μπορούν να συμβάλλουν οι μετασχηματισμοί περιορίζοντας αρκετές από τις ατείς που προκαλούν τα προβλήματα. Ετσι, με τη χρήση μετασχηματισμών γίνεται προσπάθεια να αποφευχθεί η κοιλαπλή διάσχιση δομών δεδομένων (συγγραφή ανακουφίσεων) [19,20], να περιοριστούν οι πλανοδικές και ενδιαμέσες μεταβλητές [20], να περιοριστεί ο περίπτως εξαναπολογισμός του διώκτη υποστήριξη [5], για ενσωματώθηκε πληροφορία ελέγχου σε δηλωτικά προγράμματα [6], να πολολογούνται τιμήματα του προγράμματος για τα οποία υπάρχουν αρκετά δεδομένα σε χρόνο μετάφασης [21,16], να αμφισβητηθεί ο ελέγχος μέσα στη διάδοση παραγωγής για προβλήματα του σχήματος δημιουργήστε-άποφευξτε-ορθότητα [24], να μετατραπεί η αναδρομή σε αναδρομή ουράς [9], να αντικατασταθούν δομές δεδομένων που χρησιμοποιεί το πρόγραμμα με άλλες οι οποίες μπορούν να χρησιμοποιηθούν πιο αδυνατικά [12].

Εναλλακτική προσέγγιση στη χρήση μετασχηματισμών αποτελεί η εκτέλεση του προγράμματος με τη χρήση συστημάτων που ασκούν "πια έξυπνο ελέγχο" από ότι η στάνταρτ Prolog, είναι αυτόματα είτε με βάση οδηγίες ελέγχου που διατυπώνονται από το χρήστη. Τέτοια συστήματα περιλαμβάνουν ισχυρές ισχυρότερες δυνατότητες όπως ήσυγκράφηση εκτέλεσης (coroutines), [8], εξυπηρέτηση εισαγωγής (intelligent backtracking) χρήση μετα-καταγράφησης και μετα-χανόνων για ελέγχο [11] χ.λ.π.

III. ΟΙ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΙ ΔΙΠΛΩΝ/ΞΕΔΙΠΛΩΝ


1) Νέοι ορισμοί (Definitions): Ο ορισμός μιας νέας διαδικασίας εισάγεται στο πρόγραμμα. Ο ορισμός αυτός εισάγεται μαζί με ό,τι και καταγράφονται το πρόγραμμα. Ο νέος ορισμός μπορεί να είναι αναδρομικός.

2) Ξεδιπλώμα (Unfolding): Εστώ η ακόλουθη πρόταση e' c: A ← A1, ..., An, A1, A1, ..., Ak
c1, ..., cm, ή οι προτάσεις ενός προγράμματος P, των σκοπών η κεφαλή του πρόγραμμας είναι με τον αποκλειστικό τύπο A1 με περιγράφοντας το πρόγραμμα των τύπων δεδομένων τους θ1, ..., θm, αντίστοιχα. Το ξεδιπλώμα της c στο A1 συνιστάται στην αντικατάσταση της c από τις προτάσεις c1', ..., cm' έτσι ώστε να έχει:
cj: Bj ← Bj, ..., Bnj, και Bj, θj = Aj, θj, για όλα τα j: 1 ≤ j ≤ m

3) Αντιστρέψιμο διπλώμα (reversible folding): Εστώ η ακόλουθη πρόταση c' b: B ← B1, ..., Bk, ..., Bn,

Εστώ ότι υπάρχει πρόταση c1 στο πρόγραμμα P

4) Κανόνας αντικατάστασης (replacement rule) είναι ένας άρθρος της μορφής
\[ \exists X_1, \ldots, X_n M_1, \ldots, M_k \Rightarrow \exists Y_1, \ldots, Y_m N_1, \ldots, N_h \]

όπου όλες οι μεταβλητές που περιλαμβάνονται στα \( M_1, \ldots, M_k \) εκτός από τις \( X_1, \ldots, X_n \) περιλαμβάνονται στα \( N_1, \ldots, N_h \).

Ενας κανόνας αντικατάστασης αντικαταστάσεις κάποιους στόχους στο σώμα μιας πρότασης από άλλους. Ενας τέτοιος κανόνας είναι ορθός (correct) ας προς ένα πρόγραμμα \( P \) αν για κάθε στιγμιότυπο χωρίς μεταβλητές του κανόνα ισχύει ότι: η σύζευξη των στόχων του δεξιού μέρους έπεται από το πρόγραμμα αν και μόνο αν η σύζευξη των στόχων του αριστερού του μέρους έπεται από το πρόγραμμα.

Βιδικές περιπτώσεις του κανόνα αντικατάστασης είναι η διαγραφή στόχου (goal deletion), η συνχώνευση ταυτόσημων στόχων (goal merging), και η εισαγωγή στόχων (goal addition). Ακόμη, εξαιρετικά χρήσιμοι κανόνες αντικατάστασης, όπως θα δούμε, είναι αυτοί που αναφέρονται σε ιδιότητες των καθηγοριμάτων. Στη συνέχεια θα χρησιμοποιήσουμε τρεις τέτοιους κανόνες αντικατάστασης. Ο πρώτος αναφέρεται στην προσεταιριστική ιδιότητα (associativity) ενός κατηγορήματος και εφαρμόζεται από:

\[ \exists R_1 (qa(X, Y, R_1), qa(R_1, Z, R)) \Rightarrow \exists R_2 (qa(Y, Z, R_2), qa(X, R_2, R)) \]

Οι επόμενοι δύο αναφέρονται στην υπάρξη δεξιού μοναδιαίου στοιχείου \( i_r \) (right identity element) και αριστερού μοναδιαίου στοιχείου \( i_l \) (left identity element) αντίστοιχα:

\[ qa(X, i_r, Y) \Rightarrow X = Y \]
\[ qa(i_l, X, Y) \Rightarrow Y = X \]

Η εφαρμογή των παραπάνω κανόνων μετασχηματισμού αποδεικνύεται [26] ότι διατηρεί τη σημασία (meaning) των προγραμμάτων, εφόσον τηρηθούν κάποιες προϋποθέσεις κατά την εφαρμογή τους. Σαν σημείο \( M(P) \) ενός προγράμματος \( P \) ορίζεται το σύνολο \( M(P) = \{ G : G \) ατομικός τύπος χωρίς μεταβλητές (ground) που μπορεί να αποδειχθεί από το πρόγραμμα \( P \). Η εφαρμογή των μετασχηματισμών που παρουσιάζουμε συνάντα δύο σοβαρά προβλήματα.

- Τη συνδιαστική έκρηξη που προκαλείται από την ανεξέλεγκτη εφαρμογή των μετασχηματισμών (κύρια του ξεδιπλώματος).
- Τη δυσκολία εύρεσης των καταλλήλων νέων ορισμών, οι οποίοι γι'αυτό συναφείανται και Ευρήκα (Eureka) [7], δουλειά που ασκείται αρχετή εφύοι και δυσκολεύει την αυτοματοποίηση.

Το πρόβλημα της εύρεσης νέων ορισμών καθώς και της εύρεσης αλγορίθμου για την εφαρμογή των μετασχηματισμών έχει αποδειχθεί μη επιλύσιμο (unsolvable) [18], ακόμα και για αρκετά απλές μορφές προγραμμάτων. Ετσι, πολλοί ερευνητές καταφέρουν στην χρήση εικονικών κανόνων και στρατηγικών για την αντιμετώπιση των προβλημάτων αυτών [19,20,9,5]. Οι στρατηγικές που έχουν προτεθεί, γενικά αφορούν την εισαγωγή μη αναδρομικών νέων ορισμών. Μια τέτοια περίπτωση θα δούμε στη συνέχεια.

IV. ΕΙΣΑΓΩΓΗ ΣΤΣΟΩΡΕΤΩΝ

Η περίπτωση της γραμμικής αναδρομής

Δίνεται η παρακάτω διαδικασία \( e.\sum() \) η οποία υπολογίζει το άθροισμα των στοιχείων μιας λίστας.

1. \( e.\sum([\ ]), 0) \).
2. \( e.\sum([X \mid Xs], S) \leftarrow e.\sum(Xs, S), S = S1 + X \).

Στόχος μας είναι να μετατρέψουμε τη διαδικασία αυτή σε ισοδύναμη με αναδρομή ουράς. Εισάγουμε τον σχάλιου το νέο ορισμό (Eureka) :

3. \( e.\sum.(L, Acc, S) \leftarrow e.\sum(L, S), S = S1 + Acc \).

Δίνοντας την τιμή 0 στη μεταβλητή \( Acc \) στην (D) παίρνουμε:

4. \( e.\sum.(L, 0, S) \leftarrow e.\sum(L, S), S = S1 + 0 \).

Επειδή το 0 είναι μοναδιαίο στοιχείο της πράξης +, η (3) μετατρέπεται στην

5. \( e.\sum.(L, 0, S) \leftarrow e.\sum(L, S) \).
Από την οποία παίρνουμε την (5)
(5) \( e_{sum}(L, S) \leftarrow e_{sum. acc}(L, 0, S) \).

Τώρα προσπαθούμε να βρούμε έναν αναδρομικό ορισμό για τη διαδικασία \( e_{sum. acc} \). Ετσι, ξεκινάμε να παίρνουμε την (D) στo \( e_{sum} \) με τις προτάσεις (1) και (2). Παίρνουμε έτσι τις προτάσεις
(6) \( e_{sum. acc}([] , Acc, S) \leftarrow S \) is 0 + Acc.
(7) \( e_{sum. acc}([X | X] , Acc, S) \leftarrow e_{sum}(Xs, S Xs), S1 is S Xs + X, S is S1 + Acc. \)

Η πρόταση (6) απλοποιείται δύοντας την
(8) \( e_{sum. acc}([ ] , S, S) \).

ενώ η (7) μετατρέπεται στην (9) αξιοποιώντας την προστασιακότητα της +, και αναδιατάσσοντας τις χλέσεις στο σώμα της πρότασης
(9) \( e_{sum. acc}([X | X] , Acc, S) \leftarrow Acc1 is X + Acc, e_{sum}(X, S Xs), S is S Xs + Acc1. \)

Διπλώνοντας την (9) με τη βοήθεια της (D) παίρνουμε
(10) \( e_{sum. acc}([X | X] , Acc, S) \leftarrow Acc1 is X + Acc, e_{sum. acc}(Xs, Acc1, S). \)

Ετσι, το νέο πρόγραμμα για τη διαδικασία \( e_{sum} \) σαμπτείται από τις προτάσεις {5, 8, 10} και είναι:
(11) \( e_{sum1}(L, S) \leftarrow e_{sum. acc}(L, 0, S) \).
(12) \( e_{sum. acc}([], S, S) \).
(13) \( e_{sum. acc}([X | X] , Acc, S) \leftarrow Acc1 is X + Acc, e_{sum. acc}(Xs, Acc1, S). \)

Το πρόγραμμα αυτό είναι ισοδύναμο με το {1, 2} και εμφανίζει αναδρομική ουρά. Η μεταβλητή Acc παίζει το ρόλο συσσωρευτή (accumulator) ο οποίος συσσωρεύει τα μερικά αποτελέσματα.

Εισαγωγή συσσωρευτών σε μη-γραμμικές αναδρομικές διαδικασίες

Εισαγωγή συσσωρευτών γίνεται με παρόμοιο τρόπο, και σε μη-γραμμικές αναδρομικές διαδικασίες, όπως προκύπτει από το παράδειγμα που οκολουθεί. Το πρόγραμμα flatree μαζεύει σε μια λίστα τις πληροφορίες των κόμβων ενός διαδικού δέντρου με ετικέτες (labeled binary tree).
(1) \( flatree(\text{void}, []) \).
(2) \( flatree(t(E, LT, RT), [E | L]) \leftarrow flatree(LT, LL), flatree(RT, RL), append(LL, RL, L). \)
(3) \( append([], X, X) \).
(4) \( append([X | X], Y, [X | Z]) \leftarrow append(Xs, Y, Z). \)

Για το μεταχειρισμό του προγράμματος εισάγουμε το νέο ορισμό (D) ο οποίος είναι παρόμοιος με τον ορισμό της προηγούμενης παραγράφου.
(5) \( flat_{acc}(T, Acc, L) \leftarrow flat_{acc}(T, L1), append(L1, Acc, L). \)

Δίνοντας τη μεταβλητή Acc στην (D) τη τιμή [] και απολογούντας (αξιοποιούμε το γεγονός ότι το [] είναι δεξιό μοναδιαίο στοιχείο της σχέσης append) παίρνουμε
(6) \( flat_{acc}(T, [], L) \leftarrow flat_{acc}(T, L1). \)

Από τη (5) προκύπτει η
(7) \( flat_{acc}(T, L1) \leftarrow flat_{acc}(T, [], L). \)

Και εδώ αναζητούμε έναν αναδρομικό ορισμό για τη διαδικασία \( flat_{acc} \). Ετσι, ξεκινάμε να παίρνουμε τη (D) στην flatree με τις προτάσεις (1) και (2), παίρνοντας έτσι τις προτάσεις
(8) \( flat_{acc}(\text{void}, Acc, L) \leftarrow append([], Acc, L). \)
(9) \( flat_{acc}(t(E, LT, RT), Acc, L) \leftarrow flat_{acc}(E, LT, LL), flat_{acc}(RT, RL), append(LL, LR, L1), append(L1, Acc, L). \)

Τώρα, ξεκινάμε να παίρνουμε την (7) στο append με τη πρόταση (3) και παίρνουμε
(10) \( flat_{acc}(E, LT, RT), Acc, L) \leftarrow flat_{acc}(E, LT, LL), flat_{acc}(RT, RL), append(LL, LR, L1), append(L1, Acc, L). \)

279
Αξιοποιώντας την προσεταιριστικότητα της append και αναδιατάζοντας τους στόχους, η (10) γίνεται

\[(11) \text{flat} \_\text{acc}(E, LT, RT), \text{Acc}, [E|L]) \leftarrow \text{flat} \_\text{tree}(RT, LR), \text{append}(LR, \text{Acc}, \text{Acc}1), \text{flat} \_\text{tree}(LT, LL), \text{append}(LL, \text{Acc}1, L).\]

Τέλος, διαπλάνουμε (2 φορές) την (11) χρησιμοποιώντας τη (D). Ετσι παίρνουμε τη

\[(12) \text{flat} \_\text{acc}(E, LT, RT), \text{Acc}, [E|L]) \leftarrow \text{flat} \_\text{tree}(RT, LR), \text{append}(LR, \text{Acc}, \text{Acc}1), \text{flat} \_\text{acc}(LT, LL), \text{Acc}1, L).\]

Οι προτάσεις \{6,9,12\} αποτελούν το νέο πρόγραμμα για το flat\_tree. Το πρόγραμμα αυτό εξακολουθεί να έχει μη γραμμική ανάδρομη, είναι ίμως πιο αποδοτικό από ότι το αρχικό αφού η κλήση στη διαδικασία append δεν υπάρχει και. Γενικεύοντας την παραπάνω συζήτηση μπορούμε να δούμε την ακόλουθη στρατηγική αναδρομών προγραμμάτων που ορίζουν ένα κατηγοριακό R με γραμμική ανάδρομη τα οποία στις κλήσεις των αναδρομών τους προτάσεων περιλαμβάνουν την κλήση μιας διαδικασίας Q η οποία είναι προσεταιριστική και έχει μοναδιαίο στοιχείο.

1) Εκατέρωθεν ένα μη αναδρομικό νέο ορισμό της μορφής
\[(\text{Def}) \text{R}(...) \leftarrow \text{R}(...), Q(...)\]

Οι όροι των κατηγοριατίκων είναι μεταβλητές. Οι κοινές μεταβλητές των R και Q αντανακλούν τη σύνδεση των κλήσεων στην αναδρομική διαδικασία του προγράμματος.

2) Αξιοποιώντας το μοναδιαίο στοιχείο της Q παίρνουμε από τη (Def) μια πρόταση της μορφής \(R(...) \leftarrow R(...)\)

3) Στη συνέχεια αναζητούμε αναδρομικό ορισμό για το \(R\) εφαρμόζοντας τα βήματα 3.1.

3.1) Εκκίνησης τη διάδρομη (Def) στo \(R\). Διπλώνει κάθε πρόταση που προέκυπτε με τη (Def) εφαρμόζοντας πρώτα, αν χρειάζεται, τον κανόνα για την προσεταιριστικότητα και κάνοντας απολογίσεις.

V. ΓΕΝΙΚΕΥΣΗ ΔΙΣΤΑΣ

Στόχος μας εδώ είναι να μετατρέψουμε το πρόγραμμα { 1, 2, 3, 4} της προηγουμένης παραγράφου σε άλλο συστάματο με γραμμική ανάδρομη. Εκατέρωθεν γι’αυτό τον ακόλουθο νέο (αναδρομικό) ορισμό.

\[(D1) \text{f} \_\text{ipl}([ ], [ ]).\]

\[(D2) \text{f} \_\text{ipl}([T|Ts], L) \leftarrow \text{flat} \_\text{tree}(T, FT), \text{f} \_\text{ipl}(Ts, LTs), \text{append}(FT, LTs, L).\]

Η συμπλήρωση του κατηγοριατικού (completion) [15] \(f\_ipl\) είναι

\[(C) \text{f} \_\text{ipl}(TL, L) \leftarrow (TL = [ ], L = [ ]) \text{or } (TL = [T|Ts], \text{flat} \_\text{tree}(T, FT), \text{f} \_\text{ipl}(Ts, LTs), \text{append}(FT, LTs, L))\]

Δίνοντας τη τιμή [ ] στη μεταβλητή Ts στη (C) παίρνουμε

\[(5) \text{f} \_\text{ipl}(TL, L) \leftarrow \text{false or } (TL = [T], \text{flat} \_\text{tree}(T, FT), \text{f} \_\text{ipl}([ ], LTs), \text{append}(FT, LTs, L))\]

Η (5) αποκλείεται διότι δίνουμε την (6)

\[(6) \text{f} \_\text{ipl}([T], L) \leftarrow \text{flat} \_\text{tree}(T, FT), \text{append}(FT, [ ], L).\]

Χρησιμοποιούμε το μοναδιαίο στοιχείο ([ ]) της append και παίρνουμε το ένα μέρος της συνεπαγώγης:

\[(7) \text{flat} \_\text{tree}(T, L) \leftarrow \text{f} \_\text{ipl}([T], L).\]

Προσπαθούμε τώρα να απλεφίσουμε την κλήση της flat\_tree από τη (D2). Γιατί εξετάλουμε τη (D2) στην flat\_tree χρησιμοποιούμε τις προτάσεις (1) και (2). Παίρνουμε έτσι τις προτάσεις

\[(8) \text{f} \_\text{ipl}([\text{void}|Ts], L) \leftarrow \text{f} \_\text{ipl}(Ts, LTs), \text{append}([ ], LTs, L).\]

\[(9) \text{f} \_\text{ipl}([T(E, LT, RT)|Ts], L) \leftarrow \text{flat} \_\text{tree}(LT, LL), \text{flat} \_\text{tree}(RT, RL), \text{append}(LL, RL, R), \text{f} \_\text{ipl}(Ts, LTs), \text{append}([E|R], LTs, L).\]

Έκδιπλώνοντας την (8) στο append με τη (4) παίρνοντας έτσι

\[(10) \text{f} \_\text{ipl}([\text{void}|Ts], L) \leftarrow \text{f} \_\text{ipl}(Ts, L).\]

Ακολουθούμε, εξετάλουμε την (9) στο append([E|R], LTs, L) με την (4) παίρνοντας την (11)

\[(11) \text{f} \_\text{ipl}([T(E, LT, RT)|Ts], [E|L]) \leftarrow \text{flat} \_\text{tree}(LT, LL), \text{flat} \_\text{tree}(RT, RL), \text{append}(LL, RL, R), \text{f} \_\text{ipl}(Ts, LTs), \text{append}(R, LTs, L).\]

280
Η αναδιάταξή των στόχων και η αξιοποίηση της προσεταριστικότητας του `append` στην (11) δίνει τη

(12) \( f_{\text{t}}([E, LT, RT][T]\{E[L]\}) \leftarrow \text{flatten}(LT, LL), \text{flatten}(RT, RL), f_{\text{t}}(Ts, LTs), \text{append}(RL, LTs, W), \text{append}(LL, W, L). \)

Διπλώνοντας τη (12) με τη (D2) παίρνουμε τη (13)

(13) \( f_{\text{t}}([t(E, LT, RT)[T]\{E[L]\}) \leftarrow \text{flatten}(LT, LL), f_{\text{t}}([RT][T]\{E[L]\}), \text{append}(LL, W, L). \)

Ενα ακόμη δίπλωμα με τη (D2) δίνει

(14) \( f_{\text{t}}([t(E, LT, RT)[T]\{E[L]\}) \leftarrow f_{\text{t}}([LT, RT][T]\{E[L]\}). \)

Ετσι παίρνουμε τον τελικό ορισμό για την \( f_{\text{t}} \), που είναι ένα ακόλουθος:

(15) \( \text{flatten}(T, L) \leftarrow f_{\text{t}}([T], L). \)

(16) \( f_{\text{t}}([ ]), [ ]). \)

(17) \( f_{\text{t}}([\text{void}, T], L) \leftarrow f_{\text{t}}(T, L). \)

(18) \( f_{\text{t}}([t(E, LT, RT)[T]\{E[L]\}) \leftarrow f_{\text{t}}([LT, RT][T]\{E[L]\}). \)

Το πρόγραμμα αυτό έχει γραμμική αναδρομή (και μάλιστα αναδρομή ουράς). Ακόμη, η ασαλειφή της κλήσης στη διαδικασία `append` συμβάλει στην αύξηση της αποδοτικότητας του προγράμματος. Γενικεύοντας το παράδειγμα αυτό εξήγουμε την ακόλουθη στρατηγική,

1. Εξήγησις ενός νέου αναδρομικού ορισμού της μορφής
   (D1) \( R_{\text{t}}([], t) \)
   (D2) \( R_{\text{t}}([X|Y], Z) \leftarrow R(X, X X), R_{\text{t}}(Y, Z), Q(X X, Z1, Z) \)

2. Βρες τη σχέση που συνδέει την \( R \) με τη \( R_{\text{t}} \) χρησιμοποιώντας το μονοδιαδικ στοιχείο της \( Q \).

3. Προσπάθησε να απαλείψεις την \( R \) από τη (D2) όπως έχεις (3.1) Εξειδικεύουμε τη (D2) στην \( . \) 3.2

Προσπάθησε να διπλώσεις τις προτάσεις που προέκυπταν στο 3.1 με τη (D2). Χρησιμοποιήσε αν χρειαστεί εξειδικεύσεις και κανόνες αντικατάστασης.

VI. ΣΥΝΔΙΑΖΟΝΤΑΣ ΤΗ ΓΕΝΙΚΕΥΣΗ ΔΙΣΤΑΣ ΜΕ ΤΗΝ ΕΙΣΑΓΩΓΗ ΣΤΣΣΩΡΕΥΤΗ

Στην επόμενη παράδειγμα έχουμε τη διαδικασία `sumnodes` η οποία υπολογίζει το άθροισμα των τιμών των φύλλων ενός διαδικ οδήγησης ετικέττων ( unlabeled binary tree).

(1) \( \text{sumnodes}(\text{leaf}(N), N). \)
(2) \( \text{sumnodes}(\text{t}(LT, RT), S) \leftarrow \text{sumnodes}(LT, SL), \text{sumnodes}(RT, SR), S \text{ is } SL + SR. \)

Όπως και προηγουμένως ορίζουμε

(D1) \( \text{sum}_{\text{t}}([], 0). \)
(D2) \( \text{sum}_{\text{t}}([T][T], S) \leftarrow \text{sumnodes}(T, ST), \text{sum}_{\text{t}}(Ts, STs), S \text{ is } ST + STs. \)

Παίρνοντας και εδώ τη συμπλήρωση ( completion) του κατηγορήματος \( \text{sum}_{\text{t}} \), δίνοντας την τιμή [ ] στη μεταβλητή \( Ts \), και απλοποιώντας όπως και προηγούμενος, παίρνουμε

(3) \( \text{sum}_{\text{t}}([T], S) \leftarrow \text{sumnodes}(T, S). \)

Εξειδικεύοντας τη (D2) στη `sumnodes` με τις προτάσεις (1) και (2) παίρνουμε

(4) \( \text{sum}_{\text{t}}([\text{leaf}(N)]T Ts, S) \leftarrow \text{sum}_{\text{t}}(Ts, STs), S \text{ is } N + STs. \)
(5) \( \text{sum}_{\text{t}}([t(LT, RT)]T Ts, S) \leftarrow \text{sumnodes}(LT, SL), \text{sumnodes}(RT, SR), ST \text{ is } SL + SR, \text{sum}_{\text{t}}(Ts, STs), S \text{ is } ST + STs. \)

Δόγμα της προσεταριστικότητας της πρόσθεσης (+), παίρνουμε από την (6)

(6) \( \text{sum}_{\text{t}}([t(LT, RT)]T Ts, S) \leftarrow \text{sumnodes}(LT, SL), \text{sumnodes}(RT, SR), \text{sum}_{\text{t}}(Ts, STs), W \text{ is } SR + STs, S \text{ is } SL + W. \)

Διπλώνοντας τα υπογραμμισμένα άτομα με τη (D2) παίρνουμε

(7) \( \text{sum}_{\text{t}}([t(LT, RT)]T Ts, S) \leftarrow \text{sumnodes}(LT, SL), \text{sum}_{\text{t}}([RT]Ts, W), S \text{ is } SL + W. \)

Ενα ακόμη δίπλωμα δίνει

281
(8) \( \text{sum}_\text{tpl}([t(LT, RT)|Ts], S) \leftarrow \text{sum}_\text{tpl}([LT, RT|Ts], S) \).

Ετσι, το τελικό πρόγραμμα, το οποίο είναι ισοδύναμο με το αρχικό είναι

(9) \( \text{sumnodes}(T, S) \leftarrow \text{sum}_\text{tpl}([T], S) \).

(10) \( \text{sum}_\text{tpl}([], 0) \).

(11) \( \text{sum}_\text{tpl}([\text{leaf}(N)|Ts], S) \leftarrow \text{sum}_\text{tpl}(Ts, STs), S \text{ is } N + STs \).

(12) \( \text{sum}_\text{tpl}([t(LT, RT)|Ts], S) \leftarrow \text{sum}_\text{tpl}([LT, RT|Ts], S) \).

Το πρόγραμμα αυτό έχει γραμμική αναδρομή, όχι όμως αναδρομή ουράς (λόγω της πρότασης (11)). Είναι όμως δυνατόν να μετασχηματιστεί σε ισοδύναμο πρόγραμμα με αναδρομή ουράς, αν εσάγουμε ένα συστατικό με τη βοήθεια της στρατηγικής της ενότητας 4.2. Γιατί έσαγουμε τον ορισμό:

(D3) \( \text{s}_\text{tpl}_\text{acc}(L, Acc, S) \leftarrow \text{sum}_\text{tpl}(L, S1), S \text{ is } Acc + S1 \).

Δίνοντας την τιμή 0 στη μεταβλητή Acc και την τιμή \([T]\) στη μεταβλητή \(L\) στη (D3), και απλοποιώντας παίρνουμε

(13) \( \text{s}_\text{tpl}_\text{acc}([T], 0, S) \leftarrow \text{sum}_\text{tpl}([T], S) \).

Η (13) σε συνδιασμό με την (9) δίνει

(14) \( \text{sumnodes}(T, S) \leftarrow \text{s}_\text{tpl}_\text{acc}([T], 0, S) \).

Ας αναζητήσουμε τώρα έναν αναδρομικό ορισμό για την \( \text{s}_\text{tpl}_\text{acc} \). Ξεκινώντας ξεκινώντας την \( \text{sum}_\text{tpl} \) στη (D3) χρησιμοποιώντας τις προτάσεις (10), (11) και (12). Παίρνουμε έτσι

(15) \( \text{s}_\text{tpl}_\text{acc}([], Acc, S) \leftarrow S \text{ is } Acc + 0 \).

(16) \( \text{s}_\text{tpl}_\text{acc}([\text{leaf}(N)|Ts], Acc, S) \leftarrow \text{sum}_\text{tpl}(Ts, STs), S1 \text{ is } N + STs, S \text{ is } Acc + S1 \).

(17) \( \text{s}_\text{tpl}_\text{acc}([t(LT, RT)|Ts], Acc, S) \leftarrow \text{sum}_\text{tpl}([LT, RT|Ts], S1), S \text{ is } Acc + S1 \).

Η (15) απλοποιείται δίνοντας τη

(18) \( \text{s}_\text{tpl}_\text{acc}([], S, S) \).

Αλλά της προστατευτικότητας της πρόσθεσης (+) και με αναδάταξη των στόχων, η (16) δίνει την

(19) \( \text{s}_\text{tpl}_\text{acc}([\text{leaf}(N)|Ts], Acc, S) \leftarrow Acc1 \text{ is } Acc + N, \text{sum}_\text{tpl}(Ts, STs), S \text{ is } Acc1 + STs \).

Διπλώνοντας την (19) με τη βοήθεια της (D3) παίρνουμε

(20) \( \text{s}_\text{tpl}_\text{acc}([\text{leaf}(N)|Ts], Acc, S) \leftarrow Acc1 \text{ is } Acc + N, \text{sum}_\text{tpl}_\text{acc}(Ts, Acc1, S) \).

Τώρα διπλώνουμε τη (17) με τη (D3) παίρνοντας

(21) \( \text{s}_\text{tpl}_\text{acc}([t(LT, RT)|Ts], Acc, S) \leftarrow \text{sum}_\text{tpl}_\text{acc}([LT, RT|Ts], Acc, S) \).

Ετσι παίρνουμε τον νέο πρόγραμμα για τη \text{sumnodes} το οποίο είναι

(22) \( \text{sumnodes}(T, S) \leftarrow \text{s}_\text{tpl}_\text{acc}([T], 0, S) \).

(23) \( \text{s}_\text{tpl}_\text{acc}([], S, S) \).

(24) \( \text{s}_\text{tpl}_\text{acc}([\text{leaf}(N)|Ts], Acc, S) \leftarrow Acc1 \text{ is } Acc + N, \text{sum}_\text{tpl}_\text{acc}(Ts, Acc1, S) \).

(25) \( \text{s}_\text{tpl}_\text{acc}([t(LT, RT)|Ts], Acc, S) \leftarrow \text{sum}_\text{tpl}_\text{acc}([LT, RT|Ts], Acc, S) \).

VII. ΜΙΑ ΆΛΛΗ ΜΟΡΦΗ ΝΕΩΝ ΟΡΙΣΜΩΝ ΓΙΑ ΓΕΝΙΚΕΥΣΗ ΛΙΣΤΑΣ

Η διαδικασία \text{maztree} που ακολουθεί βρίσκει τη μεγαλύτερη από τις τιμές που υπάρχουν στα φύλλα ενός δυαδικού δέντρου χωρίς ετικέτες (unlabeled binary tree). Οι τιμές αυτές πραγματικοί αριθμοί.

(1) \text{maztree}(\text{leaf}(N), N).

(2) \text{maztree}(t(LT, RT), M) \leftarrow \text{maztree}(LT, M1), \text{maztree}(RT, M2), \text{mazof2}(M1, M2, M).

(3) \text{mazof2}(M, M2, M) \leftarrow M >= M2.

(4) \text{mazof2}(M1, M, M) \leftarrow M >= M1.

Η \text{mazof} είναι προστατευτική δεν έχει όμως μοναδιαίο στοιχείο. Αυτό μας εμποδίζει να εσάγουμε νέο ορισμό της μορφής της ενότητας 5. Ετσι θα δώσει μια άλλη μορφή νέων ορισμών. Εσάγουμε

(D1) \text{maz}_\text{tpl}(T, [], M) \leftarrow \text{maztree}(T, M).

(D2) \text{maz}_\text{tpl}(T, [H|Hs], M) \leftarrow \text{maztree}(T, M1), \text{maz}_\text{tpl}(H, Hs, M2), \text{mazof2}(M1, M2, L).  

282
Η συμπλήρωση (completion) του $\text{max}$. $\text{pl}$ είναι

\[(C)\]  
$\text{max}.\text{pl}(T, L, M) \leftrightarrow \text{maxtree}(T, M1)$,  

\[(L = [1], M = M1) \text{ or } (L = [H|Hs], \text{max}.\text{pl}(H, Hs, M2), \text{mazof}2(M1, M2, L)).\]

Δίνοντας την υμ [ ] στη μεταβλητή $L$ παίρνουμε από τη (C)

\[(5)\]  
$\text{maxtree}(T, M) \leftrightarrow \text{max}.\text{pl}(T, [ ], M).$

Σε αναλογία με την ενότητα 5, προσπαθούμε να απαλέψουμε τη χλήση $\text{maxtree}$ από τον ορισμό \{D1,D2\}. Ξεκινάμε γιατί τις (D1) και (D2) στη $\text{maxtree}$ με τις προτάσεις (1) και (2) παίρνουμε

\[(6)\]  
$\text{max}.\text{pl}(\text{leaf}(N), [ ], N).$

\[(7)\]  
$\text{max}.\text{pl}(t(LT, RT), [ ], M) \leftrightarrow \text{maxtree}(LT, ML), \text{maxtree}(RT, MR), \text{mazof}2(ML, MR, M).$

\[(8)\]  
$\text{max}.\text{pl}(\text{leaf}(N), [H|Hs], M) \leftrightarrow \text{max}.\text{pl}(H, Hs, M2), \text{mazof}2(N, M2, M).$

\[(9)\]  
$\text{max}.\text{pl}(t(LT, RT), [H|Hs], M) \leftrightarrow \text{maxtree}(LT, ML), \text{maxtree}(RT, MR), \text{mazof}2(ML, MR, M1), \text{max}.\text{pl}(H, Hs, M2), \text{mazof}2(M1, M2, M).$

Διπλώνοντας την χλήση $\text{maxtree}(RT, MR)$ στην (7) με τη (D1) παίρνουμε

\[(10)\]  
$\text{max}.\text{pl}(t(LT, RT), [ ], M) \leftrightarrow \text{maxtree}(LT, ML), \text{max}.\text{pl}(RT, [ ], MR), \text{mazof}2(ML, MR, M).$

Τώρα διπλώνουμε την (10) με τη βοήθεια της (D2). Εστια παίρνουμε

\[(11)\]  
$\text{max}.\text{pl}(t(LT, RT), [ ], M) \leftrightarrow \text{max}.\text{pl}(LT, [RT], M).$

Εφαρμόζοντας στην (9) το κανόνα αντικατάστασης για τη προστασία ταυτότητας της $\text{mazof}2$ παίρνουμε

\[(12)\]  
$\text{max}.\text{pl}(t(LT, RT), [H|Hs], M) \leftrightarrow \text{maxtree}(LT, ML), \text{maxtree}(RT, MR), \text{mazof}2(H, Hs, M2), \text{mazof}2(ML, MR, M2, L1), \text{mazof}2(ML, L1, M).$

Διπλώνοντας τους υπογραμμισμένους στόχους στη (12) με τη (D2) παίρνουμε

\[(13)\]  
$\text{max}.\text{pl}(t(LT, RT), [H|Hs], M) \leftrightarrow \text{maxtree}(LT, ML), \text{max}.\text{pl}(RT, [H|Hs], L1), \text{mazof}2(ML, L1, M).$

Διπλώνοντας άλλη μια φορά με τη (D2) παίρνουμε

\[(14)\]  
$\text{max}.\text{pl}(t(LT, RT), [H|Hs], M) \leftrightarrow \text{max}.\text{pl}(LT, [RT, H|Hs], M).$

Το πρόγραμμα που προκύπτει συνοψίζεται στις προτάσεις:

\[(15)\]  
$\text{maxtree}(T, M) \leftrightarrow \text{max}.\text{pl}(T, [ ], M).$

\[(16)\]  
$\text{max}.\text{pl}(\text{leaf}(N), [ ], N).$

\[(17)\]  
$\text{max}.\text{pl}(\text{leaf}(N), [H|Hs], M) \leftrightarrow \text{max}.\text{pl}(H, Hs, M2), \text{mazof}2(N, M2, M).$

\[(18)\]  
$\text{max}.\text{pl}(t(LT, RT), [ ], M) \leftrightarrow \text{max}.\text{pl}(LT, [RT], M).$

\[(19)\]  
$\text{max}.\text{pl}(t(LT, RT), [H|Hs], M) \leftrightarrow \text{max}.\text{pl}(LT, [RT, H|Hs], M).$

Το πρόγραμμα αυτό έχει γραμμική ανάδρομη, όχι όμως ανάδρομη σωρία (λόγω της πρότασης (17)). Προσπαθούμε να εισάγουμε ένα συστατικό με τη στρατηγική της ενότητας 4.2 βεβαιώνοντας πάλι αντιμέτωπο με την θέλεισή μοναδιαίο στοιχείο για τη $\text{mazof}$ πράγμα που μας αναγκάζει να διαφοροποιήσουμε τη στρατηγική μας. Εστια, εισάγουμε τον επόμενο νέο ορισμό.

\[(D3)\]  
$\text{max}.\text{pl}.\text{acc}(H, Hs, N, M) \leftrightarrow \text{max}.\text{pl}(H, Hs, M2), \text{mazof}2(N, M2, M).$

Χρησιμοποιούμε τη (D3) διπλώνουμε τη (17) παίρνοντας

\[(20)\]  
$\text{max}.\text{pl}(\text{leaf}(N), [H|Hs], M) \leftrightarrow \text{max}.\text{pl}.\text{acc}(H, Hs, N, M).$

Αναλογία με και άλλο έναν ανάδρομο ορισμό για τη $\text{max}.\text{pl}.\text{acc}$. Εστια, εξελικώνουμε τη (D3) στη $\text{max}.\text{pl}$ με τις προτάσεις (16-19). Παίρνουμε έτσι

\[(21)\]  
$\text{max}.\text{pl}.\text{acc}(\text{leaf}(N1), [ ], N, M) \leftrightarrow \text{mazof}2(N, N1, M).$

\[(22)\]  
$\text{max}.\text{pl}.\text{acc}(\text{leaf}(N1), [H|Hs], N, M) \leftrightarrow \text{max}.\text{pl}(H, Hs, M22), \text{mazof}2(N1, M22, M2), \text{mazof}2(N, M2, M).$

\[(23)\]  
$\text{max}.\text{pl}.\text{acc}(t(LT, RT), [ ], N, M) \leftrightarrow \text{max}.\text{pl}(LT, [RT], M2), \text{mazof}2(N, M2, M).$
\[(24) \max_{t,\text{acc}(t(LT, RT), [H1|H1s], N, M)} = \max_{\text{acc}(L(T, RT, H1|H1s), M2), \maxof(N, M, 2, M)}\]

Αξιοποιώντας την προστατευτικότητα της \(\maxof2\) και αναδιατάζοντας τους στόχους, η (22) γίνεται
\[(25) \max_{\text{acc}(t(\text{leaf}(N1), [H|Hs], N, M)} = \maxof(N, N1, W), \max_{\text{acc}(H, Hs, M22), \maxof2(W, M22, M)}\]

Τώρα διπλώνουμε τη (25) χρησιμοποιώντας τη (D3). Ετσι παίρνουμε (26)
\[(26) \max_{\text{acc}(t(\text{leaf}(N1), [H|Hs], N, M)} = \maxof(N, N1, W), \max_{\text{acc}(H, Hs, W, M)}\]

\[\Delta\]
\[\Delta\]
\[\Delta\]
\[\Delta\]

Μαζεύοντας όλες τις προτάσεις παίρνουμε το σκόλουμ πρόγραμμα για το \(\text{maztree}\).
\[(29) \text{maztree}(T, M) \leftarrow \max_{\text{ipl}(T, \text{[, } M)}\]

\[(30) \max_{\text{ipl}(\text{leaf}(N1), [\text{[, } N])\]

\[(31) \max_{\text{ipl}(\text{leaf}(N), [H|Hs], M) \leftarrow \max_{\text{ipl}(H, Hs, N, M)}\]

\[(32) \max_{\text{ipl}(t(LT, RT), [\text{[, } M) \leftarrow \max_{\text{ipl}(LT, RT), M}\]

\[(33) \max_{\text{ipl}(t(LT, RT), [H|Hs], M) \leftarrow \max_{\text{ipl}(LT, RT, H|Hs), M)}\]

\[(34) \max_{\text{ipl}(\text{leaf}(N1), [\text{[, } N, M) \leftarrow \maxof2(N, N1, M)}, \max_{\text{ipl}(H, Hs, W, M)}\]

\[(35) \max_{\text{ipl}(\text{leaf}(N1), [H|Hs], N, M) \leftarrow \maxof2(N, N1, W), \max_{\text{ipl}(H, Hs, W, M)}\]

\[(36) \max_{\text{ipl}(t(LT, RT), [\text{[, } N, M) \leftarrow \max_{\text{ipl}(LT, RT), N, M}\]

\[(37) \max_{\text{ipl}(t(LT, RT), [H1|H1s], N, M) \leftarrow \max_{\text{ipl}(LT, RT, H1|H1s), N, M)}\]

Η διαδικασία μετασχηματισμού του παραδείγματος αυτού μπορεί να γενικεύεται και να διατυπωθεί σαν στρατηγική μετασχηματισμού. Αυτό όμως δεν θα μας απασχολήσει άλλο λόγο ήθελης χώρου.

VIII. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στόχος της εργασίας αυτής είναι η διερεύνηση της χρησιμότητας της εισαγωγής αναδρομικών νέων ορισμών και η ανάπτυξη στρατηγικών εφαρμογών των μετασχηματισμών διπλών/εξελίξλων. Ετσι, παρουσιάζουμε δύο διαφορετικές μορφές αναδρομικών νέων ορισμών κατάλληλες για μετασχηματισμό μη γραμμικά αναδρομικών διαδικασιών οι οποίες συνδέονται τα αποτελέσματα των αναδρομικών κλήσεων τους μέσω προστατευτικών διαδικασιών. Ακόμη παρουσιάζουμε στρατηγικές για την εφαρμογή των μετασχηματισμών καθώς και τη χρήση των αναδρομικών νέων ορισμών για τη μετατροπή των διαδικασιών σε αναδρομικές σειρές. Οποιος είναι γνωστό, τα προγράμματα που παρουσιάζουμε αναδρομική ουρά εκτελούν μειωμένες απαιτήσεις σε μηχανή. Η μετατροπή των μη γραμμικών διαδικασιών σε ισοδύναμες γραμμικές καθώς ευκολοπεύεται την παραπάνω διαδικασία μετασχηματισμού. Οι στρατηγικές εφαρμογής των μετασχηματισμών που προτείνουμε οδηγούν σε ισοδύναμα προγράμματα αφού όμως μπορεί να αποδειχθεί προοδεύουν εκ τέκμερης στην εργασία των Tamaki & Sato [26].

Το δίπλωμα με αναδρομικές προτάσεις αναφέρεται υπόλοιπο στην εργασία [26], χωρίς όμως να εξετάζονται παθητικές μορφές νέων αναδρομικών ορισμών ή να δίνονται στρατηγικές για την εφαρμογή των μετασχηματισμών. Γενικά, η σημασία των αναδρομικών νέων ορισμών δεν έχει επαρκώς εκτιμηθεί γεγονός που αποδεικνύεται από το ότι στη συντριπτική τελευταία των εργασιών [27, 12, 24, 19, 20, 18] που αναφέρονται σε μετασχηματισμούς διπλών/εξελίξλων επιτρέπονται μόνο μη αναδρομικοί νέοι ορισμοί που αποτελούνται από μια μόνο πρόταση για κάθε νέο καταγράφημα. Η μοναδική εργασία που γνωρίζουμε η οποία (έμμεσα) εισάγει αναδρομικούς νέους ορισμούς είναι η [21] που αναφέρεται στη μετατροπή λογικών προγραμμάτων σε μορφή περάσματος της συνέχειας (Continuation Passing Style).

Για την εφαρμογή των μετασχηματισμών σημαντικές αποδειχθαν οι διάστες των κατηγοριών όπως είναι η ύπαρξη μοναδικών στοιχείων και η προστατευτικότητα. Η χρήση της προστατευτικότητας σε μετασχηματισμούς προγραμμάτων αναφέρεται και σε άλλες εργασίες [4].

284
Κοινό χαρακτηριστικό των ορισμών που προτείνουμε αποτελεί το γεγονός ότι γενικέυσουν διαδικασίες που ήδη ορίζονται στο πρόγραμμα. Έχουμε συζητήσει δύο είδη γενικέυσης. Το πρώτο αφορά την εισαγωγή εικονέλων ορισμάτων που παίζουν το ρόλο συσσωρευτή και αποτελεί ένα είδος υπολογιστικής γενικέυσης (computational generalization) [10], ενώ το δεύτερο τη γενικέυση μιας διαδικασίας που χειρίζεται όρους, έτσι ώστε να χειρίζεται λίστα όρων. Η γενικέυση αυτή ονομάζεται γενικέυση λίστας (tupling generalization) [31] και αποτελεί ένα είδος δομικής γενικέυσης (structural generalization) [10].

Η έννοια της γενικέυσης δεν είναι κανονική. Μέθοδοι γενικέυσης χρησιμοποιούνται στα μαθηματικά για την απόδειξη θεωρημάτων. Στην επιστήμη των υπολογιστών μεθόδοι γενικέυσης αναφέρονται στη σύνθεση προγραμμάτων από παραδείγματα [25], στη μηχανική εκμάθηση (machine learning), την αυτόματη απόδειξη θεωρημάτων (automatic theorem proving) [3], τη σύνθεση προγραμμάτων από προ-διαγραφές [10] και την επιλέξιμη προγραμμάτων (program verification) [23].

IX. ΒΙΒΛΙΟΓΡΑΦΙΑ


