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

Απόδειξη Ορθότητας του Μετασχηματισμού Διακλαδιζόμενων Διαστάσεων στο Coq

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΗΤΡΙΟΣ ΒΕΚΡΗΣ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Επίκ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιανουάριος 2009

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

Απόδειξη Ορθότητας του Μετασχηματισμού Διακλαδιζόμενων Διαστάσεων στο Coq

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΗΤΡΙΟΣ ΒΕΚΡΗΣ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Επίκ. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 7η Ιανουαρίου 2009.

Νικόλαος Παπασπύρου Επίκ. Καθηγητής Ε.Μ.Π.	Κωστής Σαγώνας Αν. Καθηγητής Ε.Μ.Π.	Παναγιώτης Ροντογιάννης Αν. Καθηγητής Ε.Κ.Π.Α.

Αθήνα, Ιανουάριος 2009

Δημήτριος Βεκρής

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της διπλωματικής είναι η κωδικοποίηση της απόδειξης ορθότητας του μετασχηματισμού που προτείνεται από τους *Π. Ροντογιάννη* και *W. W. Wadge*, από προγράμματα συναρτησιακών γλωσσών ανώτερης τάξης σε σημασιολογικά ισοδύναμα πολυδιάστατα προγράμματα νοηματικού προγραμματισμού που έχουν μόνο συναρτήσεις μηδενικής τάξης, δηλαδή κανένα όρισμα.

Ο αλγόριθμος αυτός καταργεί τις συναρτήσεις του πηγαίου προγράμματος κατά αυστηρό και συστηματικό τρόπο με την εισαγωγή κατάλληλων τελεστών χειρισμού των περιβαλλόντων που προκύπτουν σε κάθε στάδιο του μετασχηματισμού. Ο μετασχηματισμός πραγματοποιείται σε M βήματα και σε κάθε στάδιο ο βαθμός του προγράμματος μειώνεται κατά 1. Το αποτέλεσμα είναι ένα M-διάστατο πρόγραμμα νοηματικού προγραμματισμού μηδενικής τάξης. Η μεθοδολογία που αναπτύχθηκε δίνει τις κατευθυντήριες γραμμές για ανάπτυξη νέων τεχνικών υλοποίησης συναρτησιακών γλωσσών προγραμματισμού.

Λέξεις κλειδιά

Συναρτησιακός προγραμματισμός, νοηματικός προγραμματισμός, μετασχηματισμός προγραμμάτων, διακλαδιζόμενες διαστάσεις, απόδειξη ορθότητας, Coq.

Abstract

The aim of this paper is the mechanical proof of correctness of the transformation proposed by *P. Rondogiannis* and *W. W. Wadge*, in which a broad class of higher-order functional programs can be transformed into semantically equivalent multidimensional intensional programs that contain only nullary variable definitions.

The proposed algorithm systematically eliminates user-defined functions from the source program, by appropriately introducing context manipulation (i.e. intensional) operators. The transformation takes place in M steps, where M is the order of the initial functional program. During each step the order of the program is reduced by one, and the final outcome of the algorithm is an M-dimensional intensional program of order zero.As the resulting intensional code can be executed in a purely tagged-dataflow way, the proposed approach offers a promising new technique for the implementation of higher-order functional languages. The proof was written in Coq.

Key words

Functional programming, intensional programming, program transformation, branching dimensions correctness proof, Coq.

Ευχαριστίες

Ευχαριστώ τους γονείς μου για την στήριξή που μου παρείχαν καθ'όλη τη διάρκεια των σπουδών μου στο ΕΜΠ, τον καθηγητή μου κ.Παπασπύρου για την καθοδήγησή του και το χρόνο που διέθεσε για τη συγκεκριμένη εργασία, τον υποψήφιο διδάκτορα Γιώργο Φουρτούνη για τη βοήθεια του στην χρήση του εργαλείου Coq και την Κωνσταντίνα που συμμετείχε στη μετάφραση αποσπασμάτων της εργασίας των *Ροντογιάννη* και *Wadge*.

Δημήτριος Βεκρής,

Αθήνα, 7 Ιανουαρίου 2009.

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-1-09, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Ιανουάριος 2009.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός της εργασίας

Η εργασία των *Π. Ροντογιάννη* και *W. W. Wadge* (1997) προέκυψε από τη διδακτορική διατριβή του πρώτου, στην οποία προτάθηκε ένας μετασχηματισμός από συναρτησιακά προγράμματα πρώτης τάξης σε προγράμματα νοηματικού προγραμματισμού μηδενικής τάξης. Ο συνολικός μετασχηματισμός αντιστοιχεί σε επιμέρους μετασχηματισμούς σε μία ενδιάμεση γλώσσα, η οποία διαφέρει από αυτή του προηγούμενου σταδίου στο ότι έχουν αντικατασταθεί οι μεταβλητές μέγιστης τάξης και έχουν προκύψει νέοι ορισμοί οι οποίοι θα δοθούν αναλυτικά στη συνέχεια. Στη διατριβή εξετάστηκε και αποδείχθηκε η ορθότητα του μετασχηματισμού.

Υπάρχει μεγάλη συνάφεια μεταξύ γλωσσών νοηματικού προγραμματισμού και του *tagged-data flow* μοντέλου υπολογισμού. Σε αυτό συμβάλλει καθοριστικά ο ρόλος του *context* (περιβάλλον) αλλού εμφανίζεται ως *world*. Οι *tagged-dataflow* μηχανές αποτελούν το κατάλληλο hardware για την εκτέλεση προγραμμάτων νοηματικού προγραμματισμού. Συνεπώς η διατριβή εισάγει τη δυνατότητα υλοποίησης γλωσσών συναρτησιακού προγραμματισμού με ένα καθαρά *dataflow* τρόπο. Η τεχνική βέβαια μπορεί να υλοποιηθεί σε συμβατικές αρχιτεκτονικές, αλλά σίγουρα αποτελεί μία εναλλακτική της κλασσικής μεθόδου υλοποίησης, γνωστής και ως *graph reduction method*.

Στην παρούσα εργασία γίνεται αρχικά μια ανασκόπηση της μεθόδου για προγράμματα τάξης 1. Για πιο ενδελεχή μελέτη και κατανόηση της μεθόδου ο αναγνώστης παραπέμπεται στην εν λόγω εργασία. Στη συνέχεια παρουσιάζεται ο μετασχηματισμός για προγράμματα ανώτερης τάξης. Ακολουθεί η παρουσίαση των μαθηματικών συμβολισμών που χρησιμοποιήθηκαν καθ'όλη τη διάρκεια της εργασίας, η διατύπωση της γλώσσας συναρτησιακού προγραμματισμού FL (σύνταξη, τύποι μεταβλητών και κλασσική σημασιολογία), οι γλώσσες IL και $NVIL$ (μηδενικής τάξης) με τη συγχρονική σημασιολογία τους (synchronic denotational semantics). Η εργασία καταλήγει με τη διατύπωση του μετασχηματισμού και την απόδειξη του. Σε καθε στάδιο της εργασίας θα παρουσιάζεται ο κώδικας της υλοποίησης μας παράλληλα με τους μαθηματικούς συμβολισμούς της θεωρητικής απόδειξης. Παράλληλα γίνεται μία αναφορά στη σημασία της απόδειξης, της τεχνικής που αναπτύχθηκε, τις εφαρμογές που προκύπτουν και μελλοντικές εργασίες που μπορούν να γίνουν πάνω σε αυτό το αντικείμενο.

1.2 Λίγα λόγια για την υλοποίηση και το βοηθητικό εργαλείο *Coq*

Η υλοποίηση μας έγινε σε *Coq*, που είναι ένα βοηθητικό εργαλείο (*proof assistant*) απόδειξης θεωρημάτων της μαθηματικής λογικής και όχι μόνο. Το *Coq* επιτρέπει τη διατύπωση εικασιών, προτάσεων κτλ. των μαθηματικών και ελέγχει την απόδειξη τους κατά αυστηρό και μηχανικό τρόπο. Όταν ολοκληρώσει την εν λόγω διαδικασία εξάγει μία κατασκευαστική απόδειξη (constructive proof) των προδιαγραφών που έχουν τεθεί από το χρήστη, η οποία στην ουσία αποτελεί τη «διαδρομή» μέχρι τη λύση. Η αποδεικτική διαδικασία είναι κατασκευαστική υπό την έννοια ότι εκφράζει την ύπαρξη ενός αντικειμένου που αντιστοιχεί στην πρόταση προς απόδειξη και παράγει το πρόγραμμα που αντιστοιχεί στην απόδειξη αυτή. Η εν λόγω προσέγγιση αναφέρεται στη βιβλιογραφία ως *constructive* ή *intuitionistic* και διαφέρει από την κλασική λογική στην οποία ισχύει η αρχή : $A \vee$ (*not* A) = *True*, γνωστή και ως κανόνας του αποκλεισμένου μέσου *(excluded middle rule)*. Ωστόσο μέχρι σήμερα έχουν

αναπτυχθεί βιβλιοθήκες που δίνουν τη δυνατότητα εφαρμογής της συγκεκριμένης αρχής στα διάφορα στάδια των αποδείξεων.

Επίσης διαθέτει ισχυρό διαδραστικό σύστημα αλληλεπίδρασης με το χρήστη *interactive theorem proving* ,ενώ παράλληλα παρέχει ένα πλήρες σύνολο εντολών *tactics* για την βήμα προς βήμα απόδειξη των θεωρημάτων που διατυπώνει ο χρήστης. Στις προδιαγραφές/δυνατότητες του *Coq* περιλαμβάνεται σύστημα εξαρτημένων τύπων (*dependent types*) και επαγωγικών ορισμών (*inductive definitions*).

Το σύνολο των φυσικών αριθμών με χρήση επαγωγικού ορισμού γράφεται :

Inductive $nat : Set :=$
$0 : nat \mid S : nat \rightarrow nat.$

Για παράδειγμα dependent type, μπορούμε να αναφέρουμε τη δομή *tuple*, η οποία προέρχεται από τον κώδικά μας.Ένα αντικείμενο τύπου *tuple F l* αντιστοιχεί στη λίστα που προήλθε από εφαρμογή της *F* στα στοιχεία της *l* (λειτουργία παρόμοια με *map*).

Variable $A : Type.$
Variable $F : A \rightarrow Type.$
$Function\ tuple\ (l : list\ A) : Type :=$
$match\ l\ with$
$\mid nil \Rightarrow unit$
$\mid a :: l \Rightarrow (F\ a \times tuple\ l)\%type$
$end.$

Το *Coq* αναπτύχθηκε από το ινστιτούτο *INRIA* της Γαλλίας και επεκτείνεται και στις μέρες μας από ανεξάρτητους προγραμματιστές.

1.3 Curry-Howard

Η όλη φιλοσοφία ανάπτυξης του συγκεκριμένου εργαλείου βασίζεται στον ισομορφισμό των *Curry-Howard*, ο οποίος εκφράζει την αντιστοιχία προγραμμάτων και μαθηματικών αποδείξεων. Ο ισομορφισμός *Curry-Howard* καθιερώνει μια εκπληκτική απεικόνιση-αναλογία μεταξύ των συστημάτων της τυπικής λογικής, όπως συναντώνται στη θεωρία αποδείξεων (*αποδείξεις*) και των συστημάτων υπολογισμού της θεωρίας τύπων (*προγράμματα*). Έτσι με εντυπωσιακό τρόπο συνδέονται δύο φαινομενικά ξένες περιοχές, η λογική (*θεωρία αποδείξεων*) και η πληροφορική (*θεωρία τύπων*) και ανοίγει ο δρόμος για μια συνεκτική θεωρητική προσέγγιση του μοντέρνου συναρτησιακού προγραμματισμού.

Ο ισομορφισμός των *Curry-Howard* έχει τις ρίζες του στις παρακάτω ιστορικές συγκυρίες :

1. Στην παρατήρηση του *Curry* το 1934 ότι τα θεμελιώδη αντικείμενα του λάμβδα λογισμού (*οι μεταβλητές, οι αφαιρέσεις και οι εφαρμογές*) μπορούν να ειδωθούν ως απλά σχήματα αξιωμάτων για την κατασκευστική *intuitionistic* λογική.

2. Στην παρατήρηση του *Curry* το 1954 ότι μία κατηγορία από *proof systems* γνωστά και ως *συστήματα επαγωγής Hilbert* συμπίπτουν κατά μεγάλο βαθμό με ένα σύνηθες μοντέλο υπολογισμού που έχει τύπους και λέγεται *συνδυαστική λογική*.

3. Στην παρατήρηση του *Howard* το 1969 ότι ένα υφηλού επιπέδου σύστημα αποδείξεων μπορεί να ερμηνευθεί σαν μία παραλλαγή του λάμβδα λογισμού με τύπους.

Με άλλα λόγια, ο ισομοφισμός των *Curry-Howard* είναι επί της ουσίας η απλή διαπίστωση ότι δύο φαινομενικά ασυσχέτιστοι φορμαλισμοί, των συστημάτων απόδειξης από τη μία και των μοντέλων υπολογισμού από την άλλη είναι δομικά ισοδύναμοι.

Εάν τώρα κάποιος παραλείψει τις ιδιαιτερότητες των δύο φορμαλισμών, καταλήγει στο συμπέρασμα ότι μία απόδειξη αντιστοιχεί σε ένα πρόγραμμα και το θεώρημα που αποδεικνύει είναι ο τύπος του προγράμματος. Εναλλακτικά μπορεί να διατυπωθεί ότι ο τύπος αποτελέσματος μίας συνάρτησης,

δηλαδή ο τύπος των αποτελεσμάτων που επιστρέφει είναι ανάλογος με ένα θεώρημα της λογικής που έχει υποθέσεις αντίστοιχες των τύπων που έχουν οι παράμετροι της συνάρτησης. Επίσης, ο κώδικας για τον υπολογισμό της συνάρτησης αυτής είναι ανάλογος με την απόδειξη για το θεώρημα αυτό.

Ο ισομοφισμός των *Curry-Howard* αποτέλεσε το σημείο εκκίνησης για σοβαρή έρευνα στο χώρο της λογικής. Καρπός των συγκεκριμένων ερευνητικών προσπαθειών ήταν η εμφάνιση μίας νέας κλάσης τυπικών συστημάτων σχεδιασμένων να λειτουργούν είτε ως συστήματα αποδείξεων είτε ως γλώσσες προγραμματισμού. Σε αυτά περιλαμβάνονται το σύστημα τύπων (*type sytem*) Martin-Löf και ο λογισμός του *Coquand*.

Τέτοια συστήματα σαν και αυτά που προέκυψαν από μελέτες πάνω στον ισομορφισμό οδήγησαν σε λογισμικό (βλ. *Coq*) που επιτρέπει την τυποποίηση αποδείξεων σε μορφή προγραμμάτων που μπορούν να ελεγχθούν και να τρέξουν.

Ο ισομοφισμός των *Curry-Howard* εγείρε και νέα ερωτήματα που αφορούν το υπολογιστικό περιεχόμενο των αποδείξεων που δεν είχε καλυφθεί από το έργο των *Curry-Howard* εξ' αρχής. Ενδεχομένως να οδηγήσει και στην ενοποίηση μεταξύ μαθηματικής λογικής και της επίστημης της πληροφορικής. Τα *συστήματα επαγωγής Hilbert* εντάσσονται σε μία ευρύτερη κατηγορία φορμαλισμών. Εναλλακτικοί φορμαλισμοί περιλαμβάνουν δίκτυα αποδείξεων (*proof nets*), λογισμούς κατασκευών (*calculus of structures*) κ.ά. Αν κάποιος θεωρήσει ότι ο ισομοφισμός των *Curry-Howard* είναι ισοδύναμος με τη γενική αρχή ότι κάθε απόδειξη κρύβει μέσα της ένα μοντέλο υπολογισμού τότε είναι εφικτή η ανάπτυξη μίας θεωρίας γύρω από την υπολογιστική συμπεριφορά των συγκεκριμένων συστημάτων απόδειξης. Το επόμενο ερώτημα είναι αν μπορεί να βρεθεί κάτι ενδιαφέρον από μαθηματική σκοπιά για τη συμπεριφορά αυτή.

Στην πιο γενική του μορφή ο ισομορφισμός των *Curry-Howard* είναι μία αντιστοίχηση μεταξύ του λογισμού αποδείξεων (*proof calculi*) και των συστημάτων τύπων (*type systems*) για μοντέλα υπολογισμού. Στην ουσία χωρίζεται σε δύο αντιστοιχίες. Η μία εντοπίζεται στο επίπεδο των εξισώσεων και των τύπων που είναι ανεξάρτητα του συστήματος αποδείξεων ή του μοντέλου υπολογισμού (*computational model*) που χρησιμοποιείται και η άλλη αφορά στις αποδείξεις και στα προγράμματα που αυτή τη φορά εξαρτώνται από το μοντέλο αποδείξεων.

Στο επίπεδο των εξισώσεων και των τύπων ο ισομορφισμός δείχνει ότι η λογική συνεπαγωγή συμπεριφέρεται κατά τον ίδιο τρόπο με τον τύπο συνάρτησης, η τομή με τον τύπο γινομένου (*product*), η ένωση με τον τύπο αθροίσματος (*sum*), η ψευδής (*false*) πρόταση με τον κενό τύπο και η αληθής πρόταση (*true*) με τον μοναδικό (*singleton*) τύπο. Αν θεωρήσει κανείς ότι κάθε βήμα της απόδειξης είναι υπολογιστό (*computational*) τότε ο καθολικός ποσοδείκτης (*universal quantifier*) ερμηνεύεται ως γινόμενο (*dependent product*) και ο υπαρξιακός ποσοδείκτης (*existential quantifier*) ως άθροισμα (*dependent sum*). Αυτή η θεώρηση αναφέρεται στην εκδοχή του *λ-λογισμού* που έδωσε ο Church (*σύστημα a la Church*). Εναλλακτικά αν κάποιος θεωρήσει ότι κάποιο βήμα (*computational step*) της απόδειξης μπορεί να παραλειφθεί τότε ο καθολικός ποσοδείκτης (*universal quantifier*) ερμηνεύεται ως τύπος τομής (*intersection type*) και ο υπαρξιακός ποσοδείκτης (*existential quantifier*) ως τύπος ένωσης (*union type*). Πρόκειται για το σύστημα *a la Curry*. Στον παρακάτω πίνακα συνοψίζουμε τις αντιστοιχίες :

Logic side	Programming side
universal quantification	1) intersection type (Curry), 2) dependent product type (Howard)
existential quantification	1)union type (Curry), 2)dependent sum type (Howard)
implication	function type
conjunction	product type
disjunction	sum type
true formula	unit type
false formula	bottom type

1.4 Hilbert

Στο επίπεδο των συστημάτων αποδείξεων και των μοντέλων υπολογισμού ο ισομορφισμός δίνει την αντιστοιχία μεταξύ των συστημάτων Hilbert (*Hilbert-style deduction systems*) και της συνδυαστικής λογικής (*combinatory logic*), ενώ δευτερευόντος την αντιστοιχία φυσικής επαγωγής (*natural deduction*) και του λάμβδα λογισμού (*lambda calculus*).

Κεφάλαιο 2

Η περίπτωση πρώτης τάξης

Προτού ασχοληθούμε με τα προγράμματα υψηλής τάξης, σκιαγραφούμε τη μέθοδο που υιοθετούμε για την περίπτωση της πρώτης τάξης. Αυτή η διαδικασία αναπτύχθηκε αρχικά στο [Yagh84] και τυποποιήθηκε στο [Rond97b]. Ο αλγόριθμος μετατρέπει ένα πρόγραμμα πρώτης τάξεως σε ένα σύνολο από μηδενικής τάξεως ορισμούς. Η σημασιολογία του τελικού κώδικα βασίζεται στο [Yagh84]. Ομοίως, οι τελικοί ορισμοί χαρακτηρίζονται ως *intensional*(νοηματικοί) ορισμοί. Ο αλγόριθμος μετασχηματισμού μπορεί να περιγραφεί ως ακολούθως (βλέπε [Yagh84] για πιο λεπτομερή ανάλυση).

Για κάθε συνάρτηση f που συναντάται στο αρχικό συναρτησιακό πρόγραμμα :

- Αριθμούμε τις εμφανίσεις των κλήσεων της **f** στο πρόγραμμα ξεκινώντας από το 0 συμπεριλαμβανομένων των κλήσεων στο σώμα του ορισμού της **f**.
- Αντικαθιστούμε την i-οστή εμφάνιση της **f** στο πρόγραμμα με $\texttt{call}_i\,(\mathbf{f})$. Αφαιρούμε τις τυπικές παραμέτρους από τον ορισμό της **f**,ώστε η **f** να ορίζεται πλέον σαν μία απλή μεταβλητή.
- Εισάγουμε ένα νέο ορισμό για κάθε τυπική παράμετρο της **f**. Το δεξί μέλος του ορισμού βρίσκεται ο τελεστής `actuals` εφαρμοσμένος σε μία λίστα από `actuals` παραμέτρους που αντικαθιστούν την τυπική παράμετρο στο πρόγραμμα και στη σειρά που εμφανίζονται.

Για να φανεί η λειτουργία του αλγορίθμου εξετάζουμε το ακόλουθο συναρτησιακό πρόγραμμα πρώτης τάξεως :

```
result  ≐  f(4)+f(5)
f(x)    ≐  g(x+1)
g(y)    ≐  y
```

Ο αλγόριθμος μετασχηματισμού παράγει το ακόλουθο νοηματικό πρόγραμμα :

```
result  ≐  call0(f)+call1(f)
f       ≐  call0(g)
g       ≐  y
x       ≐  actuals(4,5)
y       ≐  actuals(x+1)
```

Για την εκτέλεση του προγράμματος θεωρούμε ότι οι σημασιολογίες για τους τελεστές $\texttt{call}_i$ και `actuals` αντιστοιχούν σε πράξεις πάνω σε πεπερασμένες λίστες φυσικών αριθμών οι οποίες θα αναφέρονται από εδώ και στο εξής ως *tags* ή *contexts*). Η εκτέλεση ξεκινάει ζητώντας την τιμή της μεταβλητής `result` του αρχικού προγραμματος με κενό tag [ ]. Ο τελεστής $\texttt{call}_i$ εισάγει το i στο tag w. Από την άλλη το `actuals` αφαιρεί το i από το tag και το χρησιμοποιεί για να επιλέξει την i-οστή παράμετρο. Πιο αυστηρά, με δεδομένα τα $a, a_0, \ldots, a_{n-1}$, και θεωρώντας ότι ο τελεστής ":" εισάγει το στοιχείο αριστερά του στη λίστα δεξιά του, οι σημασιολογικές εξισώσεις όπως εισάγονται στο [Thom74] έχουν ώς εξής :

$$\begin{aligned}(call_i(a))(w) &= a(i:w)\\ (actuals(a_0,\ldots,a_{n-1}))(i:w) &= (a_i)(w)\end{aligned}$$

Ακολουθώντας τους ανωτέρω σημασιολογικούς κανόνες, το παραπάνω νοηματικό πρόγραμμα μπορεί να εκτελεστεί ως ακολούθως :

$$
\begin{aligned}
& EVAL(\texttt{call}_0\texttt{(f)+call}_1\texttt{(f)}, [\,]) \\
= \;& EVAL(\texttt{call}_0\texttt{(f)}, [\,]) + EVAL(\texttt{call}_1\texttt{(f)}, [\,]) \\
= \;& EVAL(\texttt{f}, [0]) + EVAL(\texttt{f}, [1]) \\
= \;& EVAL(\texttt{call}_0\texttt{(g)}, [0]) + EVAL(\texttt{call}_0\texttt{(g)}, [1]) \\
= \;& EVAL(\texttt{g}, [0,0]) + EVAL(\texttt{g}, [0,1]) \\
= \;& EVAL(\texttt{y}, [0,0]) + EVAL(\texttt{y}, [0,1]) \\
= \;& EVAL(\texttt{actuals(x+1)}, [0,0]) + EVAL(\texttt{actuals(x+1)}, [0,1]) \\
= \;& EVAL(\texttt{x+1}, [0]) + EVAL(\texttt{x+1}, [1]) \\
= \;& EVAL(\texttt{x}, [0]) + EVAL(\texttt{1}, [0]) + EVAL(\texttt{x}, [1]) + EVAL(\texttt{1}, [1]) \\
= \;& EVAL(\texttt{x}, [0]) + 1 + EVAL(\texttt{x}, [1]) + 1 \\
= \;& EVAL(\texttt{actuals(4,5)}, [0]) + 1 + EVAL(\texttt{actuals(4,5)}, [1]) + 1 \\
= \;& EVAL(\texttt{4}, [\,]) + 1 + EVAL(\texttt{5}, [\,]) + 1 \\
= \;& 4 + 1 + 5 + 1 \\
= \;& 11
\end{aligned}
$$

Η περιγραφείσα τεχνική χρησιμοποιήθηκε ευρέως τόσο στις εφαρμογές της [Wadg85] όσο και σε άλλες συναρτησιακές γλώσσες και συστήματα [Du90a, Du90b]. Στα ακόλουθα θα χρησιμοποιήσουμε την παραπάνω τεχνική ελαφρώς τροποποιημένη. Αυτό θα καταστήσει ευκολότερη την αντιμετώπιση των συναρτήσεων υψηλής τάξεως. Για παράδειγμα, θεωρώντας τον ορισμό :

$$\texttt{x} \doteq \texttt{actuals(4,5)}$$

στο ανωτέρω μετασχηματισμένο πρόγραμμα, θα γράψουμε εκ νέου τον ορισμό ως ακολούθως :

$$\texttt{x} \doteq \texttt{case(actuals}_0\texttt{(4),actuals}_1\texttt{(5))}$$

Ο νέος τελεστής `case` που εισάγαμε παραπάνω επιτρέπει τη χρήση μίας νέας οικογένειας από τελεστές `case` που αντιστοιχούν στο τελεστή `call`. Η σημασιολογία των νέων τελεστών δίνεται παρακάτω :

$$
\begin{aligned}
case(a_0, \ldots, a_{n-1})(i : w) &= (a_i)(i : w) \\
(actuals_i(a))(i : w) &= (a)(w)
\end{aligned}
$$

Ο παραπάνω φορμαλισμός είναι ισοδύναμος με τον προηγούμενο, αλλά έχει ένα επιπλέον πλεονέκτημα : κάθε τελεστής $\texttt{actuals}_i$ δέχεται τώρα ένα μόνο όρισμα, όπως στην περίπτωση με τον τελεστή $\texttt{call}_i$. Αυτό θα μας βοηθήσει να δώσουμε με ένα πιο κομψό τρόπο ορισμένες από τις ιδιότητες του μετασχηματισμού για προγράμματα ανώτερης τάξης.

Κεφάλαιο 3

Η περίπτωση ανώτερης τάξης

Η βασική ιδέα για τη γενίκευση της τεχνικής για προγράμματα ανώτερης τάξης γενικεύθηκε και τυποποιήθηκε στο [Wadg91] και από τότε έχει επεκταθεί και τυποποιηθεί στο [Rond94a, Rond94b]. Διαισθητικά, η τεχνική μπορεί να χειριστεί προγράμματα ανώτερης τάξης στα οποία :

1. Τα ονόματα των συναρτήσεων μπορούν να περαστούν ως παράμετροι αλλά όχι να επιστραφούν από άλλες συναρτήσεις.

2. Τα σύμβολα των τελεστών είναι πρώτης τάξης.

Οι παραπάνω συνθήκες σίγουρα επιβάλλουν ορισμένους περιορισμούς στη χρήση συναρτήσεων ανώτερης τάξης. Όμως, η κλάση των προγραμμάτων παραμένει μεγάλη.Πολλά συναρτησιακά συστήματα που χρησιμοποιούν τέτοιες συναρτήσεις μπορούν να εκτελεστούν με τη χρήση του αλγορίθμου που εξετάζεται στην παρούσα εργασία.

Η πρώτη συνθήκη ικανοποιείται από πολλές ευρέως χρησιμοποιούμενες συναρτήσεις όπως η `map` και η `foldr`. Το παρακάτω πιθανό πρόγραμμα κάνει χρήση της `map` :

```
result        ≐ map(inc,[1,2,3])
map(f,x:xs)   ≐ if (x eq []) then [] else f(x):map(f,xs)
inc(y)        ≐ y+1
```

Επιπλέον, πολλές συναρτήσεις που δεν υπακούουν στην πρώτη συνθήκη μπορούν να τροποποιηθούν ώστε η συνθήκη να ικανοποιείται.

Προσέξτε ότι η γλώσσα που χρησιμοποιούμε, παρ'ότι ανώτερης τάξης, ακόμα χρησιμοποιεί συμβατική μαθηματική σημειολογία. Οι παράμετροι μίας συνάρτησης εμφανίζονται σαν μία λίστα αμέσως μετά το όνομα της συνάρτησης.Συνεπώς γράφουμε `map(f,xs)` αντί για `(map f xs)` το συμβολισμό δηλαδή που εισήγαγε ο Curry και ο οποίος είναι πιο συνηθισμένος.Χρησιμοποιούμε τον πρώτο συμβολισμό γιατί ο δεύτερος ερμηνεύεται ως `((map f) xs)`, που σημαίνει ότι η `map` επιστρέφει μία συνάρτηση ,κάτι το οποίο η γλώσσα μας δεν επιτρέπει.

Ο δεύτερος περιορισμός δεν είναι και τόσο σημαντικός γιατί τα πιο συνηθισμένα σύμβολα τελεστών όπως +, `if-then-else`, `eq` κτλ. είναι συνήθως πρώτης τάξης.

Στο υπόλοιπο μέρος του κεφαλαίου κάνουμε μία διαισθητική εισαγωγή στην προτεινόμενη τεχνική μετασχηματισμού. Η βασική ιδέα του γενικευμένου μετασχηματισμού είναι ότι ένα πρόγραμμα M-τάξης πρέπει πρώτα να μετασχηματιστεί σε ένα νοηματικό πρόγραμμα $(M-1)$ τάξης, με τη χρήση μίας παρόμοιας τεχνικής με αυτή που χρησιμοποιήθηκε για την περίπτωση πρώτης τάξεως. Η ίδια διαδικασία μπορεί να εφαρμοστεί για το νέο πρόγραμμα μέχρι να καταλήξουμε σε πρόγραμμα μηδενικής τάξεως.

Η ιδέα των tags είναι ακόμα πιο γενική : για ένα πρόγραμμα τάξης M tag καλείται μία M-ακολουθία (μία ακολουθία μήκους M) από λίστες, κάθε λίστα της οποίας αντιστοιχεί σε μία διάσταση/τάξη του προγράμματος. Οι τελεστές είναι τώρα πιο γενικοί γιατί πρέπει να χειρίζονται νέα και πιο πολύπλοκα tags.

Επειδή ο μετασχηματισμός για την περίπτωση ανώτερης τάξης αποτελείται από αρκετά στάδια, χρησιμοποιούμε ένα διαφορετικό σύνολο από τελεστές για κάθε στάδιο. Για το πρώτο στάδιο χρησιμοποιούμε τους τελεστές $\texttt{case}^{M-1}$, $\texttt{actuals}_i^{M-1}$ και $\texttt{call}_i^{M-1}$, όπου κυμαίνεται όπως στην

περίπτωση πρώτης τάξης. Για το δεύτερο βήμα τους $\texttt{case}^{M-2}$, $\texttt{actuals}_i^{M-2}$ και $\texttt{call}_i^{M-2}$ και ούτως καθ'εξής.

Ο κώδικας που προκύπτει από το μετασχηματισμό μπορεί να εκτελεστεί ακολουθώντας τις ίδιες βασικές αρχές όπως στην περίπτωση πρώτης τάξης. Στη συνέχεια της ενότητας παρουσιάζουμε σε διαισθητικό επίπεδο το μετασχηματισμό και περιγράφουμε τη σημασιολογία των γενικευμένων τελεστών.

3.1 Ένα παράδειγμα εφαρμογής του μετασχηματισμού

Θεωρήστε το παρακάτω πρόγραμμα 2ης τάξης:

```
result      ≐ apply(inc,8)+apply(dec,5)
apply(f,x)  ≐ f(x)
inc(y)      ≐ y+1
dec(a)      ≐ a-1
```

Η συνάρτηση `apply` είναι 2ης τάξης γιατί το πρώτο όρισμά της είναι πρώτης τάξης. Ο γενικευμένος μετασχηματισμός στο 1ο στάδιο καταργεί το 1ο όρισμα της `apply` :

```
result      ≐ call_0^1(apply)(8)+call_1^1(apply)(5)
apply(x)    ≐ f(x)
inc(y)      ≐ y+1
dec(a)      ≐ a-1
f           ≐ case^1(actuals_0^1(inc),actuals_1^1(dec))
```

Το $\texttt{call}_0^1\texttt{(apply)(8)}$ είναι ισοδύναμο με το πιο συνηθισμένο συμβολισμό $\texttt{call}_0^1$ `apply 8` που δίνει $\texttt{((call}_0^1\ \texttt{apply) 8)}$.

Διαπιστώνουμε ότι το τελικό πρόγραμμα περιέχει μόνο 1ης τάξης συναρτήσεις ορισμένες από το χρήστη. Η μόνη εξαίρεση είναι ο ορισμός `f`, που είναι μία ισότητα μεταξύ συναρτήσεων. Μπορούμε εύκολα να το αλλάξουμε αυτό εισάγοντας μία βοηθητική μεταβλητή `z`:

```
result      ≐ call_0^1(apply)(8)+call_1^1(apply)(5)
apply(x)    ≐ f(x)
inc(y)      ≐ y+1
dec(a)      ≐ a-1
f(z)        ≐ case^1(actuals_0^1(inc)(z),actuals_1^1(dec)(z))
```

Αξίζει να προσέξουμε ότι στο παραπάνω πρόγραμμα οι συναρτήσεις είναι όλες 1ης τάξης (έχουν παραμέτρους μηδενικής τάξης). Το βασικό χαρακτηριστικό του νέου προγράμματος είναι ότι έχει κλήσεις σε συναρτήσεις της μορφής $\mathbf{q}\,(\mathbf{f})\,(\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$ [1], όπου $\mathbf{q}$ είναι ένας νοηματικός τελεστής (όπως οι κλήσεις $\texttt{call}_0^1\texttt{(apply)(8)}$, $\texttt{call}_1^1\texttt{(apply)(5)}$, $\texttt{actuals}_0^1\texttt{(inc)(z)}$ και $\texttt{actuals}_1^1\texttt{(dec)(z)}$).

Ακολουθεί το τελευταίο στάδιο του μετασχηματισμού το οποίο θα οδηγήσει σε πρόγραμμα μηδενικής τάξης. Προχωρούμε όπως προηγουμένως με τη μόνη διαφορά ότι χρησιμοποιούμε μία νέα διάσταση και αντίστοιχους τελεστές. Προσέξτε τη χρήση του τελεστή "$\cdot$" για σύνθεση συναρτήσεων. Συγκεκριμένα, μία έκφραση της μορφής $\mathbf{q}_1 \cdot \mathbf{q}_2\,(\mathbf{f})$ είναι ισοδύναμη με τη $\mathbf{q}_1\,(\mathbf{q}_2\,(\mathbf{f}))$ (η σύνθεση έχει μεγαλύτερη προτεραιότητα από την εφαρμογή συνάρτησης).

```
result  ≐ call_0^1·call_0^0(apply)+call_1^1·call_1^0(apply)
apply   ≐ call_0^0(f)
```

[1] Αυτό είναι ισοδύναμο με το να γράψουμε $(\mathbf{q}\ \mathbf{f})\ (\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$, που ισοδυναμεί, στον πιο συνηθισμένο συμβολισμό για τις συναρτησιακές γλώσσες, με το $(\mathbf{q}\ \mathbf{f}\ \mathbf{E}_0\ \cdots\ \mathbf{E}_{n-1})$.

$$
\begin{array}{lcl}
\texttt{inc} & \doteq & \texttt{y+1} \\
\texttt{dec} & \doteq & \texttt{a-1} \\
\texttt{f} & \doteq & \texttt{case}^1\texttt{(actuals}_0^1 \cdot \texttt{call}_0^0\texttt{(inc),actuals}_1^1 \cdot \texttt{call}_0^0\texttt{(dec))} \\
\texttt{z} & \doteq & \texttt{case}^0\texttt{(actuals}_0^0\texttt{(x))} \\
\texttt{y} & \doteq & \texttt{case}^0\texttt{(actuals}_0^0 \cdot \texttt{call}_0^1\texttt{(z))} \\
\texttt{a} & \doteq & \texttt{case}^0\texttt{(actuals}_0^0 \cdot \texttt{call}_1^1\texttt{(z))} \\
\texttt{x} & \doteq & \texttt{case}^0\texttt{(actuals}_0^0 \cdot \texttt{actuals}_0^1\texttt{(8),actuals}_1^0 \cdot \texttt{actuals}_1^1\texttt{(5))}
\end{array}
$$

Ο μετασχηματισμός είναι παρόμοιος με αυτόν της περίπτωσης 1ης τάξης. Η βασική διαφορά αφορά το χειρισμό κλήσεων της μορφής $\mathbf{q}\,(\mathbf{f})\,(\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$. Θεωρήστε π.χ. την κλήση $\texttt{call}_0^1\texttt{(apply) (8)}$ και παρατηρήστε την έκφραση $\texttt{actuals}_0^0 \cdot \texttt{actuals}_0^1\texttt{(8)}$ που εμφανίζεται στο τελικό πρόγραμμα και αντιστοιχεί στην παράμετρο 8. Το καινούριο εδώ είναι η εμφάνιση του τελεστή $\texttt{actuals}_0^1$ που θα καλέσουμε το αντίστροφο (*inverse*) του τελεστή $\texttt{call}_0^1$ που υπήρχε στην αρχική κλήση. Γενικά, το αντίστροφο της $\texttt{call}_i^m$ είναι το $\texttt{actuals}_i^m$ και αντίστροφα.

Για δεύτερο παράδειγμα, θεωρήστε $\texttt{actuals}_0^1\texttt{(inc) (z)}$. Ο μετασχηματισμός δίνει για την παράμετρο z την έκφραση $\texttt{actuals}_0^0 \cdot \texttt{call}_0^1\texttt{(z)}$ γιατί το $\texttt{call}_0^1$ είναι το αντίστροφο (*inverse*) του $\texttt{actuals}_0^1$. Οι παραπάνω ιδέες θα τυποποιηθούν σε επόμενες παραγράφους και θα γενικευθούν για την περίπτωση κλήσεων συναρτήσεων της μορφής $\mathbf{Q}\,(\mathbf{f})\,(\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$, όπου $\mathbf{Q}$ είναι η σύνθεση ενός αριθμού από νοηματικούς τελεστές.

Ο αλγόριθμος για την περίπτωση ανώτερης τάξης συνίσταται σε επανάληψη των παραπάνω βημάτων μέχρι να προκύψει πρόγραμμα μηδενικής τάξης. Για κάθε συνάρτηση **f** της τρέχουσας μέγιστης τάξης m:

1. Αριθμούμε τις εμφανίσεις των κλήσεων της **f** στο πρόγραμμα, ξεκινώντας από το 0.

2. Αφαιρούμε από την i-οστή κλήση στην **f** όλες τις actuals παραμέτρους τάξης $(m-1)$. Εισάγουμε στην αρχή της κλήσης στη **f** το $\texttt{call}_i^{m-1}$.

3. Αφαιρούμε από τον ορισμό της **f** όλες τις τυπικές παραμέτρους τάξης $(m-1)$.

4. Για κάθε τυπική παράμετρο **x** της **f** που αφαιρέθηκε , εισάγουμε ένα ορισμό $\texttt{case}^{m-1}$. Ο τελεστής $\texttt{case}^{m-1}$ παίρνει τόσες παραμέτρους όσες είναι οι κλήσεις στην **f** στο πρόγραμμα. Πιο συγκεκριμένα, η i-οστή παράμετρος του $\texttt{case}^{m-1}$ αντιστοιχεί στην i-οστή κλήση της **f** στο πρόγραμμα και είναι μία έκφραση που ξεκινάει από $\texttt{actuals}_i^{m-1}$. Επίσης, αν η συγκεκριμένη κλήση στη **f** είναι της μορφής $\mathbf{Q}\,(\mathbf{f})\,(\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$, όπου $\mathbf{Q}$ είναι η "συντακτική" σύνθεση ενός αριθμού από νοηματικούς τελεστές, το αντίστροφο του $\mathbf{Q}$ πρέπει να ληφθεί υπόψη κατά τη δημιουργία των υποεκφράσεων της $\texttt{case}^{m-1}$. Περισσότερες λεπτομέρειες πάνω στον ορισμό θα δοθούν παρακάτω.

Δοθέντος ενός πηγαίου συναρτησιακού προγράμματος τάξης M, το μοντέλο εκτέλεσης για το τελικό πρόγραμμα μηδενικής τάξης που προκύπτει απ'το μετασχηματισμό απαιτεί τα tags να είναι ακολουθίες μεγέθους M από λίστες από φυσικούς αριθμούς, όπου κάθε λίστα θα αντιστοιχεί σε κάθε τάξη του αρχικού προγράμματος ή ισοδύναμα ένα διαφορετικό στάδιο του μετασχηματισμού. Θα χρησιμοποιήσουμε το συμβολισμό $\langle w_0, ..., w_{M-1}\rangle$ όταν αναφερόμαστε σε κάποιο tag. Οι τελεστές $\texttt{call}_i^m$ και $\texttt{actuals}_i^m$ θα επιδρούν πάνω στα tags. Δεδομένου του tag , το m του $\texttt{call}_i^m$ επιλέγει την αντίστοιχη λίστα από το tag. Στη συνέχεια εισάγεται ένα i στην αρχή της λίστας και επιστρέφεται στο tag. Από την άλλη, το $\texttt{actuals}_i^m$ παίρνει από το tag τη λίστα που αντιστοιχεί στο m, ελέγχει αν η κεφαλή της λίστας είναι ίση με το i και επιστρέφει την ουρά της λίστας στο tag. Οι νέες σημασιολογικές εξισώσεις για το τους τελεστές που εμφανίζονται στα τελικά μηδενικής τάξης προγράμματα είναι :

$$
\begin{array}{lcl}
(call_i^m(a))\langle w_0,\ldots,w_m,\ldots,w_{M-1}\rangle & = & a\langle w_0,\ldots,(i:w_m),\ldots,w_{M-1}\rangle \\
(actuals_i^m(a))\langle w_0,\ldots,(i:w_m),\ldots,w_{M-1}\rangle & = & a\langle w_0,\ldots,w_m,\ldots,w_{M-1}\rangle \\
(case^m(a_0,\ldots,a_{n-1}))\langle w_0,\ldots,(i:w_m),\ldots,w_{M-1}\rangle & = & a_i\langle w_0,\ldots,(i:w_m),\ldots,w_{M-1}\rangle
\end{array}
$$

Σημειώνουμε ότι οι νοηματικοί τελεστές που εμφανίζονται σε ενδιάμεσα στάδια του μετασχηματισμού είναι γενικά ανώτερης τάξης. Προσέξτε επίσης ότι στην περίπτωση του τελεστή $actuals_i^m$, η σημασιολογική συνάρτηση δεν δίνει τι συμβαίνει αν ο έλεγχος αποτύχει. Το αποτέλεσμα σ' αυτή την περίπτωση είναι απροσδιόριστο. Παρ'όλα αυτά, ο έλεγχος ποτέ δεν αποτυγχάνει για προγράμματα που που προέρχονται από τον μετασχηματισμό.

Η εκτέλεση του προγράμματος ξεκινά από μία ακολουθία M κενών λιστών, μία για κάθε τάξη.Η εκτέλεση συνεχίζεται όπως στην περίπτωση της πρώτης τάξης και η διαφορά είναι ότι η κατάλληλη λίστα προσπελαύνεται κάθε φορά.

Το τελικό πρόγραμμα μηδενικής τάξης που προκύπτει από το μετασχηματισμό μπορεί να εκτελεστεί με τη χρήση της συνάρτησης $EVAL$. Η συνάρτηση $EVAL$ δεν ορίζεται αυστηρά, απλά μπορεί να θεωρηθεί σαν ένας απλός διερμηνέας που ακολουθεί τις σημασιολογικούς κανόνες των νοηματικών τελεστών. Επίσης η $EVAL$ επιτελεί μία απλή μορφή αντικατάστασης : κάθε φορά που χρειάζεται να αποτιμήσει μία μεταβλητή μηδενικής τάξης υπό συγκεκριμένο περιβάλλον, απλά αντικαθιστά τη μεταβλητή με την ισοδύναμη έκφρασή της και συνεχίζει την αποτίμηση. Αξίζει να αναφέρουμε ότι η $EVAL$ δε μπορεί να χρησιμοποιηθεί για να αποτιμήσει την τιμή ενός προγράμματος που οδηγεί σε ενδιάμεσο στάδιο του μετασχηματισμού γιατί τέτοια προγράμματα διαθέτουν ακόμα συναρτήσεις ορισμένες απο το χρήστη. Τα τελικά προγράμματα είναι ακόμα απλούστερα και η σημασιολογία των τελεστών που περιγράφηκαν παραπάνω αρκούν για την εύρεση της εξόδου του προγράμματος.

Η εκτέλεση ξεκινά ζητώντας την τιμή του `result` με κενό tag :

$$\begin{aligned}
& EVAL(\texttt{result},\ \langle[\,],[\,]\rangle) \\
& EVAL(\texttt{call}_0^1 \cdot \texttt{call}_0^0\texttt{(apply)} + \texttt{call}_1^1 \cdot \texttt{call}_1^0\texttt{(apply)},\ \langle[\,],[\,]\rangle) \\
= \ & EVAL(\texttt{call}_0^1\texttt{(call}_0^0\texttt{(apply))} + \texttt{call}_1^1\texttt{(call}_1^0\texttt{(apply))},\ \langle[\,],[\,]\rangle)
\end{aligned}$$

Αυτό μπορεί να αποτιμηθεί εκτελώντας ανεξάρτητα τις πράξεις για τους δύο προσθετέους και προσθέτοντας στη συνέχεια τα επιμέρους αποτελέσματα. Οι δύο υπολογισμοί είναι παρόμοιοι στη φύση. Δείχνουμε την πλήρη διαδικασία μόνο για το δεύτερο όρο (ο πρώτος δίνει 9):

$$\begin{aligned}
& EVAL(\texttt{call}_1^1\texttt{(call}_1^0\texttt{(apply))},\ \langle[\,],[\,]\rangle) \\
= \ & EVAL(\texttt{call}_1^0\texttt{(apply)},\ \langle[\,],[1]\rangle) \\
= \ & EVAL(\texttt{apply},\ \langle[1],[1]\rangle) \\
= \ & EVAL(\texttt{call}_0^0\texttt{(f)},\ \langle[1],[1]\rangle) \\
= \ & EVAL(\texttt{f},\ \langle[0,1],[1]\rangle) \\
= \ & EVAL(\texttt{case}^1\texttt{(actuals}_0^1 \cdot \texttt{call}_0^0\texttt{(inc),actuals}_1^1 \cdot \texttt{call}_0^0\texttt{(dec))},\ \langle[0,1],[1]\rangle) \\
= \ & EVAL(\texttt{actuals}_1^1 \cdot \texttt{call}_0^0\texttt{(dec)},\ \langle[0,1],[1]\rangle) \\
= \ & EVAL(\texttt{actuals}_1^1\texttt{(call}_0^0\texttt{(dec))},\ \langle[0,1],[1]\rangle) \\
= \ & EVAL(\texttt{call}_0^0\texttt{(dec)},\ \langle[0,1],[\,]\rangle) \\
= \ & EVAL(\texttt{dec},\ \langle[0,0,1],[\,]\rangle) \\
= \ & EVAL(\texttt{a-1},\ \langle[0,0,1],[\,]\rangle) \\
= \ & EVAL(\texttt{a},\ \langle[0,0,1],[\,]\rangle) - EVAL(1,\ \langle[0,0,1],[\,]\rangle) \\
= \ & EVAL(\texttt{a},\ \langle[0,0,1],[\,]\rangle) - 1 \\
= \ & EVAL(\texttt{case}^0\texttt{(actuals}_0^0 \cdot \texttt{call}_1^1\texttt{(z))},\ \langle[0,0,1],[\,]\rangle) - 1 \\
= \ & EVAL(\texttt{actuals}_0^0 \cdot \texttt{call}_1^1\texttt{(z)},\ \langle[0,0,1],[\,]\rangle) - 1 \\
= \ & EVAL(\texttt{actuals}_0^0\texttt{(call}_1^1\texttt{(z))},\ \langle[0,0,1],[\,]\rangle) - 1 \\
= \ & EVAL(\texttt{call}_1^1\texttt{(z)},\ \langle[0,1],[\,]\rangle) - 1 \\
= \ & EVAL(\texttt{z},\ \langle[0,1],[1]\rangle) - 1 \\
= \ & EVAL(\texttt{case}^0\texttt{(actuals}_0^0\texttt{(x))},\ \langle[0,1],[1]\rangle) - 1 \\
= \ & EVAL(\texttt{actuals}_0^0\texttt{(x)},\ \langle[0,1],[1]\rangle) - 1 \\
= \ & EVAL(\texttt{x},\ \langle[1],[1]\rangle) - 1 \\
= \ & EVAL(\texttt{case}^0\texttt{(actuals}_0^0 \cdot \texttt{actuals}_0^1\texttt{(8),actuals}_1^0 \cdot \texttt{actuals}_1^1\texttt{(5))},\ \langle[1],[1]\rangle) - 1 \\
= \ & EVAL(\texttt{actuals}_1^0 \cdot \texttt{actuals}_1^1\texttt{(5)},\ \langle[1],[1]\rangle) - 1
\end{aligned}$$

$$
\begin{aligned}
&= EVAL(\texttt{actuals}_1^0\texttt{(actuals}_1^1\texttt{(5))},\ \langle[1],[1]\rangle) - 1\\
&= EVAL(\texttt{actuals}_1^1\texttt{(5)},\ \langle[\,],[1]\rangle) - 1\\
&= EVAL(\texttt{5},\ \langle[\,],[\,]\rangle) - 1\\
&= \texttt{5} - \texttt{1}\\
&= \texttt{4}
\end{aligned}
$$

Το τελικό αποτέλεσμα είναι το άθροισμα των αποτελεσμάτων για τους δύο υπολογισμούς, $9 + 4 = 13$. Προσέξτε ότι παρ' ότι ο παραπάνω υπολογισμός δείχνει κάπως μεγάλος, κάθε υπολογισμός που λαμβάνει χώρα σε κάθε στάδιο είναι πρωταρχικός και μπορεί να υλοποιηθεί πολύ αποδοτικά. Επίσης, πρέπει να παρατηρηθεί ότι είναι εύκολο να κάνει κανείς απλές βελτιοποιήσεις για να βελτιώσει την απόδοση του παραγομενου κώδικα.

3.2 Ένα παράδειγμα που αφορά αναδρομή

Θεωρήστε το παρακάτω αναδρομικό πρόγραμμα 2ης τάξης που υπολογίζει τη συνάρτηση *factorial*

```
result    ≐ ffac(sq,2)
ffac(h,n) ≐ if (n<=1) then 1 else h(n)*ffac(h,n-1)
sq(a)     ≐ a∗a
```

Το πρώτο στάδιο του αλγορίθμου δίνει το εξής :

$$
\begin{aligned}
&\texttt{result} \doteq \texttt{call}_0^1\texttt{(ffac)(2)}\\
&\texttt{ffac(n)} \doteq \texttt{if (n<=1) then 1 else h(n)*call}_1^1\texttt{(ffac)(n-1)}\\
&\texttt{h} \doteq \texttt{case}^1\texttt{(actuals}_0^1\texttt{(sq),actuals}_1^1\texttt{(h))}\\
&\texttt{sq(a)} \doteq \texttt{a}*\texttt{a}
\end{aligned}
$$

Εισάγοντας μία μεταβλητή z σε κάθε πλευρά του ορισμού της h, παίρνουμε το παρακάτω 1ης τάξης πρόγραμμα :

$$
\begin{aligned}
&\texttt{result} \doteq \texttt{call}_0^1\texttt{(ffac)(2)}\\
&\texttt{ffac(n)} \doteq \texttt{if (n<=1) then 1 else h(n)*call}_1^1\texttt{(ffac)(n-1)}\\
&\texttt{h(z)} \doteq \texttt{case}^1\texttt{(actuals}_0^1\texttt{(sq)(z),actuals}_1^1\texttt{(h)(z))}\\
&\texttt{sq(a)} \doteq \texttt{a}*\texttt{a}
\end{aligned}
$$

Μπορούμε να συνεχίσουμε παρακάτω παίρνοντας το ακόλουθο μηδενικής τάξης νοηματικό πρόγραμμα :

$$
\begin{aligned}
&\texttt{result} \doteq \texttt{call}_0^1 \cdot \texttt{call}_0^0\texttt{(ffac)}\\
&\texttt{ffac} \doteq \texttt{if (n<=1) then 1 else call}_0^0\texttt{(h)*call}_1^1 \cdot \texttt{call}_1^0\texttt{(ffac)}\\
&\texttt{h} \doteq \texttt{case}^1\texttt{(actuals}_0^1 \cdot \texttt{call}_0^0\texttt{(sq),actuals}_1^1 \cdot \texttt{call}_1^0\texttt{(h))}\\
&\texttt{sq} \doteq \texttt{a}*\texttt{a}\\
&\texttt{n} \doteq \texttt{case}^0\texttt{(actuals}_0^0 \cdot \texttt{actuals}_0^1\texttt{(2),actuals}_1^0 \cdot \texttt{actuals}_1^1\texttt{(n-1))}\\
&\texttt{z} \doteq \texttt{case}^0\texttt{(actuals}_0^0\texttt{(n),actuals}_1^0 \cdot \texttt{call}_1^1\texttt{(z))}\\
&\texttt{a} \doteq \texttt{case}^0\texttt{(actuals}_0^0 \cdot \texttt{call}_0^1\texttt{(z))}
\end{aligned}
$$

Η έξοδος του παραπάνω προγράμματος μπορεί εύκολα να υπολογιστεί όπως προηγουμένως με χρήση των σημασιολογικών κανόνων για τους νοηματικούς τελεστές.

Κεφάλαιο 4

Μαθηματικά Σύμβολα

Το σύνολο των φυσικών αριθμών συμβολίζεται με N. Το σύνολο των συναρτήσεων από A σε B συμβολίζεται με $A \rightarrow B$. Το αποτέλεσμα της εφαρμογής της συνάρτηση f στο όρισμα a θα γράφεται $f(a)$ ή σε εξαιρετικές περιπτώσεις f_a. Ακόμα και εδώ απόφεύγεται το "Currying".

Θα χρησιμοποιηθεί ένας τελεστής σύνθεσης "$\cdot$" με σημασιολογία ανάλογη αυτή του τελεστή "$\cdot$" στην αντικείμενη γλώσσα.

Για σημειολογική απλότητα, συμβολίζουμε μία τούπλα $\langle z_0, z_1, \ldots, z_{n-1} \rangle$ με $\vec{z}$. Ακολουθείται η ακόλουθη γενίκευση για γινόμενα συνόλων : Εάν I σύνολο και A_i είναι σύνολο για κάθε $i \in I$:

$$\prod_{i \in I} A_i = \{f : I \rightarrow \bigcup_{i \in I} A_i \mid \forall i \in I, f(i) \in A_i\}$$

Ορίζουμε την πράξη perturbation για μία συνάρτηση αναφορικά με μία άλλη συνάρτηση :

Ορισμός 4.1: Θεωρούμε $f : A \rightarrow B$ και $g : S \rightarrow B$, όπου $S \subseteq A$. Τότε το *perturbation* $f \oplus g$ του f αναφορικά με τη g ορίζεται παρακάτω :

$$(f \oplus g)(x) = \begin{cases} g(x) & \text{if } x \in S \\ f(x) & otherwise \end{cases}$$

Δεδομένης συνάρτησης $g = \{\langle x_0, b_0 \rangle, \ldots, \langle x_{n-1}, b_{n-1} \rangle\}$, συχνά θα χρησιμοποιούμε τον εναλλακτικό συμβολισμό $f[x_0/b_0, \ldots, x_{n-1}/b_{n-1}]$ αντί για $f \oplus g$.

Θεωρούμε ένα σύνολο L . Γράφουμε $List(L)$ για το σύνολο από λίστες αντικειμένων της L.Οι συνήθεις πράξεις στη λίστα $head$, $tail$ και $cons$ θεωρούνται γνωστές. Ο συμβολισμός ":" θα χρησιμοποιείται αντί για $cons$.

Κάποια εξοικείωση με τις βασικές αρχές της θεωρίας πεδίων (domain theory) και της δηλωτικής σημασιολογίας (denotational semantics) [Stoy77, R91, Gunt92] κρίνεται απαραίτητη. Δεδομένου πεδίου D, η μερική διάταξη (*partial order*) και το ελάχιστο στοιχείο του D συμβολίζονται με $\sqsubseteq_D$ και $\perp_D$ αντίστοιχα. Ο δείκτης D θα παραλείπεται όποτε είναι προφανής. Αν A, B είναι πεδία , $[A \rightarrow B]$ είναι το σύνολο από συνεχείς συναρτήσεις από το A στο B.

Στη συνέχεια, υιοθετούμε συγκεκριμένες τυπογραφικές συμβάσεις που δίνονται συνοπτικά παρακάτω. Στοιχεία της αντικείμενης γλώσσας , όπως για παράδειγμα ο κώδικας των προγραμμάτων συμβολίζονται με τη χρήση τυπογραφικού φόντου (π.χ. $\mathtt{f}, \mathtt{x}, \ldots$). Στοιχεία της μετα-γλώσσας χωρίζονται σε δύο κατηγορίες : σε εκείνα τα οποία χρησιμοποιούνται για να συμβολίσουν συνηθισμένα μαθηματικά αντικείμενα, όπως συναρτήσεις, σύνολα και για τα οποία χρησιμοποιούμε το φόντο : $f, x, \mathcal{E}, \mathcal{A}, \ldots$) και σε εκείνα που χρησιμοποιούμε ώστε να μιλήσουμε για το συντακτικό της αντικείμενης γλώσσας και για τα οποία χρησιμοποιούμε το εξής φόντο : $\mathbf{f}, \mathbf{x}, \mathbf{P}, \mathbf{E}, \ldots$)

Τα τελευταία χρόνια, έγινε μεγάλη πρόοδος στην κατεύθυνση του εμπλουτισμού των γλωσσών προγραμματισμού με μία ευρεία γκάμα τύπων δεδομένων. Οι τύποι επιβάλλουν *a priori* συντακτικούς περιορισμούς στο ποιές δομές της γλώσσας μπορούν να συνδυαστούν, βοηθώντας με αυτόν τον τρόπο τον προγραμματιστή να αποφύγει κώδικα λανθασμένο ή χωρίς νόημα. Σε αυτή την ενότητα, ορίζουμε τη σύνταξη και τη σημασιολογία των τύπων που χρειάζονται για τις ανάγκες αυτής της εργασίας.

Ορισμός 4.2: Το σύνολο $STyp$ από *simple types* και το σύνολο Typ από τύπους ορίζονται αναδρομικά ως εξής :

$$\begin{array}{rcl} \sigma & ::= & \iota \\ & | & (\sigma_0, \ldots, \sigma_{n-1}) \rightarrow \iota \\ \tau & ::= & \sigma \\ & | & \sigma \rightarrow \sigma \end{array}$$

Ο τύπος ι είναι απλός *ground* τύπος. Προσέξτε ότι όλα τα μέλη από το σύνολο $STyp$ έχουν τύπο αποτελέσματος ι. Όπως θα περιγραφεί παρακάτω, οι γλώσσες που εξετάζονται σε αυτή την εργασία υπακούουν σε αυτό τον περιορισμό ,υπό την έννοια ότι όλες οι συναρτήσεις ορισμένες σε αυτές πρέπει να έχουν ένα τύπο που ανήκει στο σύνολο $STyp$. Από την άλλη, οι γλώσσες που συζητούμε θα έχουν νοηματικούς τελεστές (`call` και `actuals`) με τύπους της μορφής $\sigma \rightarrow \sigma$.

Σε επίπεδο κώδικα, ο τύπος δεδομένων $STyp$ δίνεται από τον παρακάτω επαγωγικό ορισμό ο οποίος δίνει αναδρομικά το $STyp$ από λίστες ομοειδών αντικειμένων $STyp$.

```
Inductive STyp : Type :=
  | st_arrow : list STyp → STyp.
```

Επίσης υιοθετήθηκε η σύμβαση ο απλός *ground* τύπος να αντιστοιχεί στην κενή λίστα :

```
Definition st_iota := st_arrow nil.
```

Το Typ δίνεται εύκολα ως εξής :

```
Inductive Typ : Type :=
  | t_simple : STyp → Typ
  | t_arrow : STyp → STyp → Typ.
```

Ορισμός 4.3: Η τάξη *order* ενός απλού τύπου ορίζεται αναδρομικά ως εξής :

$$\begin{array}{rcl} order(\iota) & = & 0 \\ order((\sigma_0, \ldots, \sigma_{n-1}) \rightarrow \iota) & = & 1 + max(\{order(\sigma_i) \mid 0 \leq i \leq n-1\}) \end{array}$$

και στη γλώσσα του *theorem assistant* γράφουμε εύκολα :

```
Function order (s : STyp) : nat :=
  match s with
    | st_arrow sl ⇒ match sl with
                      | nil ⇒ 0
                      | _ ⇒ 1 + max_list (map order sl)
                    end
  end.
```

Ορισμός 4.4: Η σημασιολογία ενός τύπου σε σχέση με ένα δοθέν πεδίο D δίνεται αναδρομικά από μία συνάρτηση $[\![\cdot]\!]_D$, όπου το D συχνά θα παραλείπεται ως εξής :

- $[\![\iota]\!]_D = D$
- $[\![(\sigma_0, \ldots, \sigma_{n-1}) \rightarrow \iota]\!]_D = [([\![\sigma_0]\!]_D, \ldots, [\![\sigma_{n-1}]\!]_D) \rightarrow [\![\iota]\!]_D]$
- $[\![\sigma_1 \rightarrow \sigma_2]\!]_D = [[\![\sigma_1]\!]_D \rightarrow [\![\sigma_2]\!]_D]$

Παρατηρούμε ότι η σημασιολογία του ι ισούται με το σύνολο D. Η σημασιολογία του τύπου μία συνάρτησης με τύπο παραμέτρων $sigma_0, \ldots, \sigma_{n-1}$ και τύπο αποτελέσματος ι ισούται με το σύνολο των σημασιολογικών συναρτήσεων που έχουν πεδίο ορισμού το $([\![\sigma_0]\!]_D, \ldots, [\![\sigma_{n-1}]\!]_D)$ και τύπο αποτελέσματος το σύνολο D. Στη γενική περίπτωση η συνάρτηση με τύπο $\sigma_1 \rightarrow \sigma_2$ έχει σημασιολογία το σύνολο των συναρτήσεων από το $[\![\sigma_1]\!]_D$ στο $[\![\sigma_2]\!]_D$.

Θεωρούμε την ύπαρξη ενός συνόλου Σ από σύμβολα σταθερών διαφόρων τύπων του $STyp$. Στοιχεία του Σ έχουν τύπους που προέκυψαν από συνάρτηση ανάθεσης $\theta : \Sigma \rightarrow STyp$. Οι σταθερές συμβολίζονται ως $\mathbf{c}$.

Για την απόδειξη θεωρήσαμε δύο απλές περιπτώσεις σταθερών : τους φυσικούς αριθμούς και τα αθροίσματα φυσικών.

```
Inductive Sigma : Type :=
    | constNat : nat → Sigma
    | constPlus : Sigma.
Definition result : Var := mkVar "result".

Parameter theta : Sigma → STyp.
```

Επίσης θεωρούμε την ύπαρξη ενός συνόλου Var από σύμβολα μεταβλητών διαφόρων τύπων. Στοιχεία του Var έχουν τύπους που προέκυψαν από την συνάρτηση ανάθεσης $\pi : Var \rightarrow STyp$. Οι μεταβλητές συμβολίζονται ως $\mathbf{f}, \mathbf{g}, \mathbf{x}, \ldots$. Συγκεκριμένα, χρησιμοποιούμε το Var_0 για να αναφερόμαστε σε μεταβλητές με τύπο ι. Μεταβλητές ή σταθερές τύπου ι καλούνται επίσης μηδενικές (*nullary*) ή ατομικές (*individual*) μεταβλητές.

```
Inductive Var : Type :=
    | mkVar : string → Var.
```

Κεφάλαιο 5

Η συναρτησιακή γλώσσα FL

Σε αυτή την ενότητα, ορίζουμε το συντακτικό και τη δηλωτική σημασιολογία μίας συναρτησιακής γλώσσας FL ανώτερης τάξης.

Ορισμός 5.1: Το συντακτικό της συναρτησιακής γλώσσας FL ορίζεται αναδρομικά και τα $\mathbf{E}, \mathbf{E}_i$ αντιστοιχούν σε εκφράσεις, τα $\mathbf{F}, \mathbf{F}_i$ σε ορισμούς και τα $\mathbf{P}$ σε προγράμματα.

$$
\begin{array}{lcl}
\mathbf{E} & ::= & \mathbf{f} \in \mathit{Var} \\
 & | & \mathbf{c}\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,,\ \mathbf{c} \in \Sigma,\ n \geq 0 \\
 & | & \mathbf{f}\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,,\ \mathbf{f} \in \mathit{Var},\ n \geq 0 \\
\mathbf{F} & ::= & \mathbf{f}\,(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{E},\ \mathbf{f}, \mathbf{x}_i \in \mathit{Var},\ n \geq 0 \\
\mathbf{P} & ::= & \{\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}\}
\end{array}
$$

Το συντακτικό της συναρτησιακής γλώσσας FL, όπως ορίστηκε στον κώδικά μας έχει ως εξής (είναι αρκετά απλό και δεν χρειάζεται επεξηγήσεις) :

```
Inductive Expr : Type :=
  | f_expr : Var → Expr
  | c_expr : Sigma → list Expr → Expr
  | a_expr : Var → list Expr → Expr.

Inductive Def : Type :=
  | def : Var → list Var → Expr → Def.

Definition Prog : Type := list Def.
```

Προσέξτε ότι το συντακτικό επιτρέπει σταθερές μηδενικής τάξης, που σε αυτή την περίπτωση θα γραφούν $\mathbf{c}\,()$ ή απλά $\mathbf{c}$. Το συντακτικό επιτρέπει επίσης μεταβλητές μηδενικής τάξης ή ακόμα και ορισμούς μηδενικής τάξης. Και στις δύο περιπτώσεις το κενό σύνολο τν παρενθέσεων που ακολουθεί θα παραλείπεται. Το συντακτικό της FL θα περιοριστεί ακόμα περισσότερο από κανόνες που ορίζουν πότε οι τύποι είναι καλά ορισμένοι.

Δεδομένου του ορισμού $\mathbf{f}\,(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{E}$, οι μεταβλητές $\mathbf{x}_i$ είναι *τυπικοί παράμετροι* του $\mathbf{f}$ και $\mathbf{E}$ είναι η έκφραση ή το σώμα *body* της $\mathbf{f}$.

Ορισμός 5.2: Θεωρούμε πρόγραμμα $\mathbf{P} = \{\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}\}$: Κάνουμε τις ακόλουθες υποθέσεις.

1. Ακριβώς ένα από τα $\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}$ ορίζει μία ατομική μεταβλητή **result**, η οποία δεν εμφανίζεται στο σώμα οποιουδήποτε εκ των ορισμών της $\mathbf{P}$.

2. Κάθε σύμβολο μεταβλητής στο $\mathbf{P}$ ορίζεται ή εμφανίζεται ως τυπική παράμετρος στον ορισμό της συνάρτησης, ακριβώς μία φορά σε ολόκληρο το πρόγραμμα.

3. Οι τυπικές παράμετροι του ορισμού της συνάρτησης στο $\mathbf{P}$ μπορούν να εμφανιστούν μόνο στο σώμα του συγκεκριμένου ορισμού.

4. Οι μόνες μεταβλητές που μπορούν να εμφανιστούν στο $\mathbf{P}$ είναι εκείνες που ορίζονται στο $\mathbf{P}$ και τις τυπικές τους παραμέτρους.

Το σύνολο των μεταβλητών του προγράμματος $\mathbf{P}$ συμβολίζεται $func(\mathbf{P})$. Οι κανόνες ελέγχου τύπων για τη γλώσσα δίνονται στη μορφή $\mathbf{E} : \sigma$, το οποίο δείχνει ότι η έκφραση $\mathbf{E}$ είναι καλά ορισμένη με τύπο σ και οι μεταβλητές-σταθερές που χρησιμοποιούνται στη $\mathbf{E}$ έχουν τύπους ορισμένους από τα π και θ αντίστοιχα. Ο παραπάνω ορισμός αντανακλά τους δύο περιορισμούς που αναφέρθηκαν παραπάνω, ότι τα ονόματα συναρτήσεων μπορούν να περάσουν ως παράμετροι αλλά όχι να επιστραφούν ως αποτελέσματα και ότι τα σύμβολα των τελεστών είναι πρώτης τάξης.

```
Definition func (p : Prog) : list Var :=
  map lhs_def p.
```

Ορισμός 5.3: Το σύνολο των εκφράσεων με τύπους που είναι καλά ορισμένοι δίνεται παρακάτω :

$$\frac{\pi(\mathbf{f}) = \sigma}{\mathbf{f} : \sigma}$$

$$\frac{(\theta(\mathbf{c}) = (\iota, \ldots, \iota) \rightarrow \iota) \wedge (\forall i \in \{0, \ldots, n-1\}\ (\mathbf{E}_i : \iota))}{\mathbf{c}\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1}) : \iota}$$

$$\frac{(\mathbf{f} : (\sigma_0, \ldots, \sigma_{n-1}) \rightarrow \iota) \wedge (\forall i \in \{0, \ldots, n-1\}\ (\mathbf{E}_i : \sigma_i))}{\mathbf{f}\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1}) : \iota}$$

```
Inductive typExpr : Expr → STyp → Type :=
  | f_typ :
    ∀ (f : Var),
      typExpr (f_expr f) (pi f)
  | c_typ :
    ∀ (c : Sigma) (el : list Expr) (n : nat),
      theta c = st_arrow (repeat_list st_iota n) →
      typExprList el (repeat_list st_iota n) →
        typExpr (c_expr c el) st_iota
  | a_typ :
    ∀ (f : Var) (el : list Expr) (sl : list STyp),
    pi f = st_arrow sl →
    sl ≠ nil →
    typExprList el sl →
      typExpr (a_expr f el) st_iota
  with typExprList : list Expr → list STyp → Type :=
  | nil_typl : typExprList nil nil
  | cons_typl (e : Expr) (s : STyp) (el : list Expr) (sl : list STyp) :
    typExpr e s → typExprList el sl → typExprList (e :: el) (s :: sl).
```

Ο επαγωγικός ορισμός typExpr αντιστοιχεί στο λεγόμενο typing των εκφράσεων της συναρτησιακής γλώσσας FL. Παίρνει δύο ορίσματα : την έκφραση και τον τύπο της. Στην περίπτωση των μεταβλητών $\mathbf{f}$ ο ορισμός εξασφαλίζει μέσω του *typExpr* (*f_expr f*) (*pi f*) ότι ο τύπος της $\mathbf{f}$ είναι αυτός που ανατίθεται από τη συνάρτηση *pi*. Για την περίπτωση των σταθερών θα πρέπει, καθ'ότι πρόκειται για εφαρμογή ενός συμβόλου σταθεράς σε ορίσματα, ο τύπος της έκφρασης να είναι *st_iota*, το οποίο εκφράζεται από την πρόταση *typExpr* (*c_expr c el*) *st_iota* ενώ θα πρέπει οι τύποι των ορισμάτων να ανατίθενται από τη *theta* και να ισούνται με *st_iota*, το οποίο εξασφαλίζεται από την : *theta c* = *st_arrow* (*repeat_list st_iota n*). Παράλληλα, θα πρέπει αναδρομικά να υπολογίζεται το typing για όλα τα actual ορίσματα της *c*. Αυτό το ρόλο παίζει η *typExprList el* (*repeat_list st_iota n*) →, όπου το *typExprList* είναι εμφωλευμένος επαγωγικός ορισμός που δίνει τα typing μίας λίστας εκφράσεων *el*. Παρόμοια πράγματα μπορούν να ειπωθούν για την περίπτωση της εφαρμογής συνάρτησης f σε λίστα ορισμάτων. Ο επιπλέον περιορισμός εδώ είναι ότι δεν μπορεί η λίστα αυτή των ορισμάτων να μην είναι κενή (*sl* ≠ *nil*).

Ορισμός 5.4: Ένας ορισμός $\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{E}$ με $\mathbf{f} : (\sigma_0, \ldots, \sigma_{n-1}) \to \iota$ έχει τύπο καλά ορισμένο αν $\mathbf{E} : \iota$ και για κάθε $i \in \{0, \ldots, n-1\}$, $\mathbf{x}_i : \sigma_i$.

```
Inductive typDef : Def → Type :=
  | d_typ : ∀ (f : Var) (xl : list Var) (e : Expr),
              NoDup xl → pi f = st_arrow (map pi xl) → typExpr e st_iota → typDef (def f xl e).
```

Ο ορισμός των *Def* της συναρτησιακής γλώσσας FL ικανοποιεί τον περιορισμό να μην υπάρχουν διπλά ονόματα για τυπικές παραμέτρους της *f*. Επίσης πληροί την προυπόθεση ο τύπος που αναθέτει η συνάρτηση *pi* στην f να ισούται με τη λίστα των επιμέρους τύπων για τις παραμέτρους *xl* :

pi f = st_arrow (*map pi xl*) και ο τύπος της έκφρασης στο δεξί μέλος του ορισμού να είναι *st_iota* :

typExpr e st_iota.

Ορισμός 5.5: Ένα πρόγραμμα $\{\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}\}$ έχει τύπο καλά ορισμένο αν $\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}$ ορισμοί με καλά ορισμένους τύπους.

```
Inductive typProg : Prog → Type :=
  | p_nil : typProg nil
  | p_cons : ∀ (d : Def) (Hd : typDef d) (p : Prog) (Hp : typProg p),
               ¬In (lhs_def d) (func p) → typProg (d :: p).
```

Για το typing του προγράμματος αρκεί να έχουμε ένα σύνολο από καλά ορισμένα *Def*. Στα επόμενα , συχνά αναφερόμαστε σε προγράμματα μηδενικής τάξης , 1ης τάξης κτλ. Ο ακόλουθος ορισμός τυποποιεί τις παραπάνω αρχές :

Ορισμός 5.6: Θεωρούμε **P** ένα FL πρόγραμμα. Η τάξη του **P** ορίζεται ως εξής :

$$Order(\mathbf{P}) = max(\{order(\pi(\mathbf{f})) \mid \mathbf{f} \in func(\mathbf{P})\})$$

Χωρίς να έχουμε να σχολιάσουμε κάτι πάνω σε αυτό οι τάξεις των ορισμών και του προγράμματος αντίστοιχα δίνονται από τα παρακάτω τμήματα κώδικα :

```
Definition order_of_Def (pi : Var → STyp) (d : Def_IL) : nat :=
  order (pi (lhs_def_IL d)).
    Definition order_of_P (pi : Var → STyp) (p : Prog_IL) : nat :=
  max_list (map (fun (d : Def_IL) ⇒ order (pi (lhs_def_IL d))) p).
```

Δίνεται το πεδίο D. Η σημασιολογία των σταθερών συμβόλων της FL σε σχέση με το D προσδιορίζεται από τη συνάρτηση $\mathcal{C}$, η οποία αναθέτει σε κάθε σταθερά τύπου σ μία τιμή στο $[\![\sigma]\!]_D$. Έστω Exp_σ το σύνολο όλων των εκφράσεων **E** της FL τέτοιο ώστε $\mathbf{E} : \sigma$. Έστω Env_π το σύνολο των π-συμβατών περιβαλλόντων ,τα περιβάλλοντα που για όλα τα $\mathbf{f} \in Var$ να ισχύει $u(\mathbf{f}) \in [\![\pi(\mathbf{f})]\!]_D$. Αυτό το σύνολο ορίζεται από το $Env_\pi = \prod_{\mathbf{f} \in Var} [\![\pi(\mathbf{f})]\!]_D$. Τότε η σημσιολογία της FL ορίζεται με τη χρήση συναρτήσεων αποτίμησης $[\![\cdot]\!]^\sigma_D : Exp_\sigma \to [Env_\pi \to [\![\sigma]\!]_D]$, (όπου οι δείκτες και εκθέτες D, σ π θα παραληφθούν όποτε είναι προφανείς από το περιβάλλον).

Για τα παραπάνω χρειάστηκε καταρχάς μία δομή, η οποία να αποθηκεύει τη σημασιολογία για μία λίστα από τύπους. Αυτή η δομή είναι η *tuple* και ο κώδικάς της είναι :

```
Section Hetero.

  Variable A : Type.
  Variable F : A → Type.

  Function tuple (l : list A) : Type :=
    match l with
      | nil ⇒ unit
```

```
    | a :: l ⇒ (F a × tuple l)%type
  end.
```

End *Hetero*.

Την ανάθεση των σημασιολογιών για κάθε τύπο μίας λίστας *sl* αναλαμβάνει η συνάρτηση *SDom* :

```
Fixpoint SDom (s : STyp) : Type :=
  match s with
    | st_arrow sl ⇒ tuple SDom sl → D
  end.
```

Στη συνέχεια ορίζουμε μεταβατική και αυτοπαθή σχέση για αντικείμενα τύπου *SDom s* για οποιοδήποτε τύπο *s*.

```
Definition SDomrel (s : STyp) : relation (SDom s) :=
  match s return relation (SDom s) with
    | st_arrow sl ⇒ fun (f1 f2 : tuple SDom sl → D) ⇒
                          ∀ t : tuple SDom sl, Drel (f1 t) (f2 t)
  end.
```

και παίρνουμε ελάχιστο στοιχείο για τη σχέση αυτή το :

```
Fixpoint SDombot (s : STyp) : SDom s :=
  match s return SDom s with
    | st_arrow sl ⇒ fun dl : tuple SDom sl ⇒ Dbot
  end.
```

Η συνάρτηση απόδοσης της σημασιολογίας των σταθερών *C* θα είναι :

```
Parameter C : ∀ (c : Sigma), SDom (theta c).
```

Κατά παρόμοιο τρόπο δίνεται ένας πολυμορφικός ορισμός για το περιβάλλον u, ο οποίος θα χρησιμοποιηθεί και στην περίπτωση της IL γλώσσας, για τις σχέσεις μεταξύ περιβαλλόντων και για το ελάχιστο στοιχείο για τη σχέση αυτή.

```
Definition Env := ∀ f : Var, SDom (pi f).
Definition Erel (u1 u2 : Env) := ∀ (f : Var), SDomrel (pi f) (u1 f) (u2 f).
Definition Ebot : Env := fun (f : Var) ⇒ SDombot (pi f).
```

Ορισμός 5.7: Η σημασιολογία των εκφράσεων της FL σε σχέση με το $u \in Env$ ορίζεται αναδρομικά ως εξής :

$$[\![\mathbf{f}]\!](u) = u(\mathbf{f})$$
$$[\![\mathbf{c}\ (\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\ ]\!](u) = \mathcal{C}(\mathbf{c})([\![\mathbf{E}_0]\!](u), \ldots, [\![\mathbf{E}_{n-1}]\!](u))$$
$$[\![\mathbf{f}\ (\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\ ]\!](u) = u(\mathbf{f})([\![\mathbf{E}_0]\!](u), \ldots, [\![\mathbf{E}_{n-1}]\!](u))$$

Πρέπει να αναφερθεί ότι το $[\![\cdot]\!]$ όπως ορίστηκε παραπάνω είναι πολυμορφικό. Πιο ακριβής ορισμός θα ενέπλεκε τη χρήση $[\![\cdot]\!]^{\sigma}_{D}$, που χρησιμοποιούνται για διαφορετικούς τύπους εκφράσεων. Για λόγους σαφήνειας αποφύγαμε να κάνουμε το συμβολισμό πιο πολύπλοκο.

Στον κώδικά μας, η σημασιολογία της κάθε έκφρασης θα δίνεται από τον ορισμό *sem_fl*. Η *sem_fl_expr_list* ορίζει αφενός τη σημασιολογία για μία έκφραση δεδομένου του typing της και του περιβάλλοντος στο οποίο θα αποτιμηθεί και αφετέρου τη σημασιολογία για μία λίστα από εκφράσεις με αντίστοιχες παραμέτρους. Αξίζει να σημειωθεί ότι ο τελευταίος ορισμός έγινε σε αποδεικτική μορφή *proof mode*, εξαιτίας τς πολυπλοκότητας που θα ενείχε η κλασική διατύπωση. Η περιπτωσιολογία πάνω στο *typing* της εκάστοτε έκφρασης έγινε με τη βοήθεια του ορισμού *typing_expr_list_mutrec*, ο οποίος παραλείπεται λόγω της πολυπλοκότητας και της δυσκολίας στην κατανόηση που τον χαρακτηρίζει. Η *sem_fl_args* προσδιορίζει τη σημασιολογία για μία λίστα από εκφράσεις.

```
Definition sem_fl_expr_list :
  (∀ (e : Expr) (s : STyp), typExpr pi e s → Env_FL → SDom s) ×
  (∀ (el : list Expr) (sl : list STyp), typExprList pi el sl → Env_FL → tuple SDom sl).
Proof.
  apply typing_expr_list_mutrec.
    intros f u.
      exact (u f).
    intros c el n H1 H2 dl u _.
    apply (cast (C_FL c) H1).
    exact (dl u).
    intros f el sl H1 H2 H3 dl u _.
      apply (cast (u f) H1).
      exact (dl u).
      intro u.
      exact tt.
      intros e s el sl H1 d H2 dl u.
      exact (d u, dl u).
Defined.

Definition sem_fl : ∀ (e : Expr) (s : STyp), typExpr pi e s → Env_FL → SDom s :=
  fst (sem_fl_expr_list).
Definition sem_fl_args :
  ∀ (el : list Expr) (sl : list STyp), typExprList pi el sl → Env_FL → tuple SDom sl :=
  snd (sem_fl_expr_list).
```

Θεώρημα 5.1: [R91, page 97] Για όλες τις εκφράσεις **E**, η $[\![\mathbf{E}]\!]$ είναι συνεχής και μονότονη.

Για τον ορισμό της συνέχειας χρησιμοποιήσαμε κώδικα, ο οποίος περιέχεται στο *function.v* και αποδίδεται στον Adam Chlipala. Αν μία συνάρτηση *c*: *nat* → *A* ορίζει αλυσίδα και το ελάχιστο άνω φράγμα αυτής είναι το *l* τότε για συνεχή συνάρτηση *f*:*A* → *B* θα ισχύει ότι το ελάχιστο άνω φράγμα για την *f c* ισούται με *l c*.

```
Definition continuous (f:A → B) :=
  ∀ (c: nat → A), chain R c →        ∀ (l:A), lub R l c →        lub R' (f l) (fun n ⇒ f (c n)).
```

Η αλυσίδα (*chain*) ορίζει σχέση *R* για διαδοχικές τιμές της *f*:*nat*→*A* και δίνεται παρακάτω :

```
Definition chain (f:nat→A) := ∀ n, R (f n)(f (S n)).
```

Για λόγους πληρότητας παραθέτουμε τον ορισμούς του άνω φράγματος και του ελαχίστου άνω φράγματος :

```
Definition upper_bound (x:A)(f:nat → A) :=
  ∀ n, R (f n) x.

Definition lub (x:A)(f:nat → A) :=
  upper_bound x f ∧ ∀ y, upper_bound y f → R x y.
```

Συνεπώς το θεώρημα 5.1 μπορεί να διατυπωθεί ως εξής :

```
Theorem th_5_1 :
  ∀ (e : Expr) (s : STyp) (H : typExpr pi e s),
    continuous Erel_FL (SDomrel s) (sem_fl e H).
Proof.
Admitted.
```

Εξαιτίας της εξάρτησης του από θεωρία πάνω στη σημασιολογία των εκφράσεων που διατυπώθηκε από τον Tennent και επειδή η απόδειξη του παραλείφθηκε από τη διατριβή των *P.Rondogiannis* και *W.W.Wadge*, ομοίως δεν ασχοληθήκαμε και εμείς με την απόδειξη του.

Ορισμός 5.8: Η σημασιολογία του προγράμματος $\mathbf{P} = \{\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}\}$ της FL ορίζεται ως το αποτέλεσμα της $\widetilde{u}(\mathbf{result})$, όπου $\widetilde{u}$ είναι το ελάχιστο (*least*) περιβάλλον τ.ώ. για κάθε $\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{E}$ στο $\mathbf{P}$ με $\mathbf{f} : (\sigma_0, \ldots, \sigma_{n-1}) \to \iota$, και για όλα τα $d_0 \in [\![\sigma_0]\!]_D, \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]_D$, $\widetilde{u}(\mathbf{f})(d_0, \ldots, d_{n-1}) = [\![\mathbf{E}]\!](\widetilde{u}[\mathbf{x}_0/d_0, \ldots, \mathbf{x}_{n-1}/d_{n-1}])$.

Ο παραπάνω ορισμός δεν ξεκαθαρίζει πως κατασκευάζεται το ελάχιστο περιβάλλον $\widetilde{u}$. Το παρακάτω θεώρημα δίνει ότι το $\widetilde{u}$ είναι το ελάχιστο άνω φράγμα από μία αλυσίδα περιβαλλόντων, που μπορούν να θεωρηθούν ως διαδοχικές προσεγγίσεις του $\widetilde{u}$.

Οι βασικοί ορισμοί για τον ορισμό της σημασιολογίας προγράμματος $\mathbf{P}$ είναι ο *sem_prog_aux* και ο *sem_prog*. Ο πρώτος προσδιορίζει το ελάχιστο περιβάλλον, όπως προκύπτει από τον ορισμό του *Tarski_fix* και ο δεύτερος τη σημασιολογία της έκφρασης *result*, η οποία δίνει τη σημασιολογική τιμή του προγράμματος.

```
Definition sem_prog_aux (p : Prog) (H : typProg pi p) : Env_FL :=

  Tarski_fix (fun (u : Env_FL) ⇒ sem_update_prog u p H Ebot_FL ).

Definition sem_prog (p : Prog) (H : typProg pi p) : SDom st_iota :=
  sem_fl (f_expr result) (typ_result pi) (sem_prog_aux p H).
```

Για τον ορισμό του *sem_update_prog* χρειάστηκαν οι *sem_def_FL* και *sem_update_def*. Ο πρώτος αποδίδει τη σημασιολογική τιμή για το *Def* στο περιβάλλον *u*, βρίσκοντας αρχικά τη σημασιολογία της έκφρασης *e* του δεύτερου μέλους στο περιβάλλον *u'*. Το *u'* (δε χρησιμοποιείται στον κώδικα παρακάτω, απλά αναφέρεται εδώ για να διευκολυνθεί ο αναγνώστης στην κατανόηση) είναι το *u* στο οποίο έχουν προστεθεί για σημασιολογικές τιμές των πραγματικών παραμέτρων *xl* οι τιμές *dl* με τύπους συμβατούς ως προς το (*pi f*). Το περιβάλλον *u'* δίνεται από το τμήμα κώδικα *env_update_list_FL xl* (*cast dl Hpar*) *u*) *tt* και κάνει χρήση της συνάρτησης *env_update_list_FL*, η οποία ανανεώνει τις σημασιολογικές τιμές των μεταβλητών *xl* αποδίδοντάς τους τις τιμές *dl*. Ο κώδικας της *env_update_list* είναι σε *proof mode* και κάνει χρήση της *env_update*, που έχει προφανή λειτουργία. Ο *sem_update_prog* παίρνει ως παράμετρο το *u*, το οποίο στη συνέχεια θα προσδιοριστεί από τη συνάρτηση *Tarski*. Επίσης παίρνει παράμετρο ένα άλλο περιβάλλον *e*, το οποίο συγκεντρώνει σταδιακά όλες τις σημασιολογικές τιμές για κάθε ορισμό *def* καθώς η *sem_update_prog* "διατρέχει" τους ορισμούς του προγράμματος. Αυτές προστίθενται στο επαγωγικό βήμα (*sem_update_def u d Hd* (*sem_update_prog u p H e*) οδηγώντας κατόπιν εφαρμογής της συνάρτησης *Tarski* στο ζητούμενο ελάχιστο περιβάλλον. Το περιβάλλον που "περνά" ως παράμετρος στη συνάρτηση *sem_prog_aux* αρχικοποείται στο *bottom* περιβάλλον, *Ebot_FL*.

```
Definition env_update (f : Var) (d : SDom (pi f)) (u : Env) : Env :=

  fun f' : Var ⇒
    match var_eq_dec f f' with
      | left Heq ⇒ cast (A:=STyp) d (f_equal _ Heq)
      | right _ ⇒ u f'
    end.

Definition env_update_list : ∀ xl : list Var, tuple SDom (map pi xl) → Env → Env.
Proof.
  intros xl dl.
  induction xl as [ | x xl ].
  exact (fun u : Env ⇒ u).
  exact (fun u : Env ⇒ env_update x (fst dl) (IHxl (snd dl) u)).
Defined.

Definition sem_def_FL (d : Def) (H : typDef pi d) (u : Env_FL) : SDom (pi (lhs_def d)) :=

  match H in typDef _ d return SDom (pi (lhs_def d)) with
    | d_typ f xl e _ Hpar Ht ⇒
```

```
        func_make (fun dl : dom_args (pi f) ⇒
          sem_fl e Ht (env_update_list_FL xl (cast dl Hpar) u) tt)
  end.

Definition sem_update_def (u : Env_FL) (d : Def) (H : typDef pi d) : Env_FL → Env_FL.
Proof.
  intros u d H.
  exact (env_update_FL (lhs_def d) (sem_def_FL d H u)).
Defined.

Definition sem_update_prog (u : Env_FL) (p : Prog) (H : typProg pi p) (e : Env_FL) : Env_FL.
Proof.
  fix 3.
  intros.
  destruct H.
  exact e.
  exact (sem_update_def u d Hd (sem_update_prog u p H e)).
Defined.
```

Θεώρημα 5.2: [R91, page 96] Έστω $\mathbf{P}$ και $\widetilde{u}$ όπως στον ορισμό 5.8. Τότε, $\widetilde{u}$ είναι το ελάχιστο άνω φράγμα για την ακολουθία $\widetilde{u}_k$ που ορίζεται παρακάτω :

1. Για κάθε $\mathbf{f} \in Var$ με $\mathbf{f} \notin func(\mathbf{P})$, $\widetilde{u}_k(\mathbf{f}) = \bot$, για κάθε $k \in N$.
2. Για κάθε $\mathbf{f}\,(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{E}$ στο $\mathbf{P}$, με $\mathbf{f} : (\sigma_0, \ldots, \sigma_{n-1}) \to \iota$, και για κάθε $d_0 \in [\![\sigma_0]\!]_D, \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]_D$,

$$\begin{aligned} \widetilde{u}_0(\mathbf{f})(d_0, \ldots, d_{n-1}) &= \bot \\ \widetilde{u}_{k+1}(\mathbf{f})(d_0, \ldots, d_{n-1}) &= [\![\mathbf{E}]\!](\widetilde{u}_k[\mathbf{x}_0/d_0, \ldots, \mathbf{x}_{n-1}/d_{n-1}]) \end{aligned}$$

Ο παραπάνω ορισμός των διαδοχικών προσεγγίσεων του *uk* αντανακλάται στον κώδικά μας :

```
Fixpoint uk (p : Prog) (H : typProg pi p) (k : nat) {struct k} : Env_FL :=
  match k return Env_FL with
    | 0 ⇒ fun f : Var ⇒ SDombot (pi f)
    | S n ⇒ sem_update_prog (uk p H n) p H Ebot_FL
  end.
```

Το θεώρημα 5.2 διατυπώθηκε εύκολα στον κώδικά μας ως εξής. Συγκεκριμένα, χωρίς να υπεισέλθουμε σε λεπτομέρειες της απόδειξης το *th_5_2_1_aux* δηλώνει ότι για μεταβλητή f που δεν ανήκει στο σύνολο *func* του προγράμματος η σημασιολογία του δίνεται από το αρχικό περιβάλλον *e* που εισάγεται ως παράμετρος στη συνάρτηση *sem_update_prog*.

```
Lemma th_5_2_1_aux :
  ∀ (p : Prog) (Hp : typProg pi p) (f : Var) (u e : Env_FL),
    ¬In f (func p) →
      sem_update_prog u p Hp e f = e f.
  Qed.
```

Το θεώρημα *th5_2_1* εκφράζει την ιδιότητα του περιβάλλοντος *uk* ότι για κάθε f που δεν ανήκει στο σύνολο *func* του προγράμματος η σημασιολογική τιμή της f ισούται με το *bottom* στοιχείο του συνόλου *SDom* (*pi f*).

```
Lemma th5_2_1:
  ∀ (p : Prog) (H : typProg pi p) (k : nat) (f : Var),
    ¬In f (func p) →
```

uk p H k f = SDombot (*pi f*).

Το θεώρημα *th5_2_2* δηλώνει ότι το *uk* για *k=0* δίνει σημασιολογική τιμή σε κάθε μεταβλητή *f* του προγράμματος ίση με το bottom στοιχείο του συνόλου *SDom* (*pi f*).

Lemma *th5_2_2*:
∀ (*p* : *Prog*) (*Hp* : *typProg pi p*) (*f* : *Var*) (*xl* : *list Var*) (*e* : *Expr*)
(*Heq* : *pi f = st_arrow* (*map pi xl*)) (*args* : *dom_args* (*pi f*)),
In (*def f xl e*) *p* →
func_apply (*uk p Hp* 0 *f*) *args = Dbot*.

Το επαγωγικό βήμα του ορισμού για το περιβάλλον *uk* δίνεται από το θεώρημα *th5_2_3* :

Lemma *th5_2_3*:
∀ (*p* : *Prog*) (*Hp* : *typProg pi p*) (*k* : *nat*) (*f* : *Var*) (*xl* : *list Var*)
(*e* : *Expr*) (*He* : *typExpr pi e st_iota*) (*Hnd* : *NoDup xl*)
(*Heq* : *pi f = st_arrow* (*map pi xl*)) (*args* : *dom_args* (*pi f*)),
In (*def f xl e*) *p* →
func_apply (*uk p Hp* (*S k*) *f*) *args* =
sem_fl e He (*env_update_list_FL xl* (*cast args Heq*) (*uk p Hp k*)) *tt*.

Το ότι ο *Tarski* δίνει το ελάχιστο περιβάλλον για το πρόγραμμα *Prog* εξάγεται από το παρακάτω θεώρημα :

Theorem *th5_2_a* (*p* : *Prog*) (*Hp* : *typProg pi p*) : *lub Erel_FL* (*sem_prog_aux p Hp*) (*uk p Hp*).

Επίσης, για κάθε $k \in N$, θα είναι $\widetilde{u}_k(\mathbf{f}) \sqsubseteq \widetilde{u}_{k+1}(\mathbf{f})$.

Theorem *th5_2_b* (*p* : *Prog*) (*Hp* : *typProg pi p*) (*f* : *Var*) (*k* : *nat*) :
SDomrel (*pi f*) (*uk p Hp k f*) (*uk p Hp* (*S k*) *f*).

Το ακόλουθο λήμμα είναι μία άμεση συνέπεια του θεωρήματος 6.1.2 :

Λήμμα 5.1: Έστω **P** και $\widetilde{u}$ όπως στον ορισμό 5.8. Τότε για κάθε $\mathbf{f}\,(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{E}$ στο **P** με $\mathbf{f} : (\sigma_0, \ldots, \sigma_{n-1}) \to \iota$,

$$\widetilde{u}_k(\mathbf{f})(d_0, \ldots, d_{n-1}) \quad \sqsubseteq \quad [\![\mathbf{E}]\!](\widetilde{u}_k[\mathbf{x}_0/d_0, \ldots, \mathbf{x}_{n-1}/d_{n-1}])$$

Lemma *lm5_1*:
∀ (*p* : *Prog*) (*Hp* : *typProg pi p*) (*k* : *nat*) (*f* : *Var*) (*xl* : *list Var*)
(*e* : *Expr*) (*He* : *typExpr pi e st_iota*) (*Hnd* : *NoDup xl*)
(*Heq* : *pi f = st_arrow* (*map pi xl*)) (*args* : *dom_args* (*pi f*)),
In (*def f xl e*) *p* →
Drel (*func_apply* (*uk p Hp k f*) *args*)
(*sem_fl e He* (*env_update_list_FL xl* (*cast args Heq*) (*uk p Hp k*)) *tt*).
Proof.

Κεφάλαιο 6

Οι νοηματικές γλώσσες IL και NVIL

Σε αυτή την ενότητα , ορίζουμε το συντακτικό της νοηματικής γλώσσας IL και $NVIL$ που χρησιμοποιούνται στον αλγόριθμο μετασχηματισμού (η IL παριστάνει τη *νοηματική γλώσσα* και η $NVIL$ τη *νοηματική γώσσα μηδενικής τάξης*). Η γλώσσα IL είναι νοηματική ανώτερης τάξης και τα προγράμματα που εμφανίζονται στα ενδιάμεσα στάδια του μετασχηματισμού ανήκουν στην IL. Από την άλλη, τα τελικά προγράμματα μηδενικής τάξης που προκύπτουν από τον αλγόριθμο μετασχηματισμού ανήκουν στη νοηματική γλώσσα $NVIL$ που είναι απλούστερη στη δομή από την IL και η οποία παρουσιάζεται ανεξάρτητα.

Η διαφορά μεταξύ IL και $NVIL$ είναι στην παρουσία νοηματικών τελεστών. Εξαιτίας της φύσης του μετασχηματισμού, οι νοηματικοί τελεστές σε προγράμματα της IL εμφανίζονται με συγκεκριμένο τρόπο. Θεωρήστε για παράδειγμα το παρακάτω πρόγραμμα ως αποτέλεσμα του πρώτου σταδίου του μετασχηματισμού (section 3):

$$
\begin{array}{lcl}
\texttt{result} & \doteq & \texttt{call}_0^1\texttt{(apply)(8)+call}_1^1\texttt{(apply)(5)} \\
\texttt{apply(x)} & \doteq & \texttt{f(x)} \\
\texttt{inc(y)} & \doteq & \texttt{y+1} \\
\texttt{dec(a)} & \doteq & \texttt{a-1} \\
\texttt{f(z)} & \doteq & \texttt{case}^1\texttt{(actuals}_0^1\texttt{(inc)(z),actuals}_1^1\texttt{(dec)(z))}
\end{array}
$$

Αν εξετάσουμε τις κλήσεις σε συναρτήσεις στο πρόγραμμα, διαπιστώνουμε ότι ορισμένες από αυτές είναι της μορφής $\mathbf{q}\,(\mathbf{f})\,(\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$. Γενικά , οι κλήσεις σε συναρτήσεις που εμφανίζονται στα ενδιάμεσα προγράμματα του μετασχηματισμού θα έχουν τη μορφή $\mathbf{Q}\,(\mathbf{f})\,(\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$, όπου $\mathbf{Q}$ είναι μία πιθανά κενή ακολουθία από συνδυασμούς νοηματικών τελεστών.

Τα τελικά προγράμματα μηδενικής τάξης που αποτελούν το αποτέλεσμα του μετασχηματισμού, έχουν απλούστερο συντακτικό από τα προγράμματα που εμφανίζονται στα ενδιάμεσα στάδια του μετασχηματισμού. Επιπλέον, σε αυτά τα προγράμματα, οι νοηματικοί τελεστές μπορούν να εφαρμοστούν όχι μόνο σε συναρτήσεις αλλά και σε άλλες εκφράσεις, κάτι το οποίο δε συμβαίνει στα ενδιάμεσα προγράμματα του μετασχηματισμού (π.χ. θεωρούμε την έκφραση $\texttt{actuals}_0^0 \cdot \texttt{actuals}_0^1\texttt{(8)}$ στο τελικό πρόγραμμα του πρώτου παραδείγματος της ενότητας 3). Για αυτούς τους λόγους, ορίζουμε ανεξάρτητα το συντακτικό για τις γλώσσες IL και $NVIL$.

Ο ακόλουθος ορισμός δίνει το συντακτικό για ακολουθίες νοηματικών τελεστών. Το ϵ αναφέρεται στην κενή ακολουθία.

Ορισμός 6.1: Το σύνολο $ISeq$ των ακολουθιών από νοηματικούς τελεστές $\mathbf{Q}$ ορίζεται αναδρομικά ως εξής:

$$
\begin{array}{lcl}
\mathbf{Q} & ::= & \epsilon \\
 & | & \texttt{call}_i^m,\ i,m \geq 0 \\
 & | & \texttt{actuals}_i^m,\ i,m \geq 0 \\
 & | & \texttt{call}_i^m \cdot \mathbf{Q},\ i,m \geq 0, \mathbf{Q} \neq \epsilon \\
 & | & \texttt{actuals}_i^m \cdot \mathbf{Q},\ i,m \geq 0, \mathbf{Q} \neq \epsilon
\end{array}
$$

Το αντίστοιχο τμήμα του κώδικά μας είναι απλό στην κατανόηση :

Inductive Q : *Type* :=

| *nothing*
| *complex* (_: *Q_aux*)
with Q_aux : *Type* :=
| *call* (_ : *nat*) (_ : *nat*)
| *actuals* (_ : *nat*) (_ : *nat*)
| *call_complex* (_ : *nat*) (_ : *nat*) (_ : *Q_aux*)
| *actuals_complex* (_ : *nat*) (_ : *nat*) (_ : *Q_aux*)
.

Προσέξτε ότι τα στοιχεία του συνόλου $ISeq$ είναι συντακτικά αντικείμενα. Σε κάθε $\mathbf{Q} \in ISeq$ αντιστοιχεί μία συνάρτηση Q η οποία είναι η σύνθεση των σημασιολογιών των αντικειμένων της ακολουθίας (λεπτομέρειες στην ενότητα 6.1).

Λαμβάνοντας υπόψη τις παραπάνω παρατηρήσεις, έχουμε τον ακόλουθο ορισμό που αφορά το συντακτικό της νοηματικής γλώσσας IL:

Ορισμός 6.2: Το συντακτικό της νοηματικής γλώσσας IL ορίζεται αναδρομικά με τους ακόλουθους κανόνες, στους οποίους τα $\mathbf{E}, \mathbf{E}_i$ παριστάνουν *εκφράσεις*, το **B** παριστάνει το *σώμα* ενός ορισμού, τα $\mathbf{F}, \mathbf{F}_i$ τους ορισμούς και το **P** παριστάνει το *πρόγραμμα*:

$$
\begin{array}{lcl}
\mathbf{E} & ::= & \mathbf{f} \in \mathit{Var} \\
 & | & \mathbf{c}\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,,\ \mathbf{c} \in \Sigma,\ n \geq 0 \\
 & | & \mathbf{Q}\,(\mathbf{f})\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,,\ \mathbf{f} \in \mathit{Var},\ \mathbf{Q} \in ISeq,\ n \geq 0 \\
\mathbf{B} & ::= & \mathbf{E} \\
 & | & \mathtt{case}^m\,(\mathbf{E}_0, \ldots, \mathbf{E}_{r-1})\,,\ r, m \geq 0 \\
\mathbf{F} & ::= & \mathbf{f}\,(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B},\ \mathbf{f}, \mathbf{x}_i \in \mathit{Var},\ n \geq 0 \\
\mathbf{P} & ::= & \{\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}\}
\end{array}
$$

Inductive *Expr_IL* : *Type* :=
| *f_expr_IL* : *Var* → *Expr_IL*
| *c_expr_IL* : *Sigma* → *list Expr_IL* → *Expr_IL*
| *a_expr_IL* : *Q* → *Var* → *list Expr_IL* → *Expr_IL*
| *q_expr_IL* : *Q* → *Expr_IL* → *Expr_IL*.

Inductive *B_IL* : *Type*:=
| *E_IL* : *Expr_IL* → *B_IL*
| *case_IL* : *nat* → *list Expr_IL* → *B_IL*.

Inductive *Def_IL* : *Type* :=
| *def_IL* : *Var* → *list Var* → *B_IL* → *Def_IL*.

Definition *Prog_IL* := *list Def_IL*.

Προσέξτε ότι οι συνηθισμένες κλήσεις σε συναρτήσεις είναι επιτρεπτές στην περίπτωση που το **Q** είναι η κενή ακολουθία. Σε αυτή την περίπτωση, οι παρενθέσεις γύρω από το **f** θα παραλείπονται.

Οι έννοιες των ορισμών και των προγραμμάτων με καλά ορισμένες τύπους, όπως και η έννοια της τάξης του προγράμματος εφαρμόζονται με ανάλογο τρόπο και εδώ στα 5.4, 5.5 and 5.6. Επιπλέον υιοθετούνται οι υποθέσεις του 5.2 για τα προγράμματα IL.

Τα τελικά προγράμματα μηδενικής τάξης, που προκύπτουν από το μετασχηματισμό, ανήκουν στη γλώσσα $NVIL$. Το συντακτικό της $NVIL$ ορίζεται παρακάτω :

Ορισμός 6.3: Το συντακτικό της νοηματικής γλώσσας $NVIL$ ορίζεται αναδρομικά με τους ακόλουθους κανόνες, όπου τα $\mathbf{E}, \mathbf{E}_i$ αναφέρονται σε *εκφράσεις*, το **B** παριστάνει *σώμα*, τα $\mathbf{F}, \mathbf{F}_i$ *ορισμούς* και το **P** παριστάνει το *πρόγραμμα*:

$$
\begin{array}{lcl}
\mathbf{E} & ::= & \mathbf{f} \in \mathit{Var}_0 \\
 & | & \mathbf{c}\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,,\ \mathbf{c} \in \Sigma,\ n \geq 0 \\
 & | & \mathbf{Q}\,(\mathbf{E})\,,\ \mathbf{Q} \in ISeq \\
\mathbf{B} & ::= & \mathbf{E} \\
 & | & \mathtt{case}^m\,(\mathbf{E}_0, \ldots, \mathbf{E}_{r-1})\,,\ r, m \geq 0 \\
\mathbf{F} & ::= & \mathbf{f} \doteq \mathbf{B},\ \mathbf{f} \in \mathit{Var}_0 \\
\mathbf{P} & ::= & \{\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}\}
\end{array}
$$

Ο ορισμός της $NVIL$ δε δόθηκε ξεχωριστά στον κώδικά μας, γιατί μπορεί να ενσωματωθεί με αυτόν της IL, οπότε δόθηκε σαν σύνολο παραπάνω. Εξάλλου, οι διαφορές τους εντοπίζονται στον όρο *q_expr_IL*, ο οποίος επί της ουσίας αφορά μόνον την $NVIL$, αφού οι παράμετροι των μεταβλητών-συναρτήσεων έχουν καταργηθεί. Η $NVIL$ διαθέτει μόνο μεταβλητές μηδενικής τάξης.

Προσέξτε ότι η γλώσσα $NVIL$ είναι παρόμοια με αυτή που ορίστηκε στο [Rond97b]. Η διαφορά είναι στους τελεστές που εδώ είναι πολυδιάστατοι. Αναφορικά με το typing των στοιχείων της γλώσσας θα έχουμε τα παρακάτω :

Variable *pi* : *Var* → *STyp*.
Inductive *typExpr_IL* : *Expr_IL* → *STyp* → *Type* :=
| *f_typ_IL* :
 ∀ (*f* : *Var*),
 typExpr_IL (*f_expr_IL f*) (*pi f*)
| *c_typ_IL* :
 ∀ (*c* : *Sigma*) (*el* : *list Expr_IL*) (*n* : *nat*),
 theta c = *st_arrow* (*repeat_list st_iota n*) →
 typExprList_IL el (*repeat_list st_iota n*) →
 typExpr_IL (*c_expr_IL c el*) *st_iota*
| *a_typ_IL* :
 ∀ (*q* : *Q*) (*f* : *Var*) (*el* : *list Expr_IL*) (*sl* : *list STyp*),
 pi f = *st_arrow sl* →
 sl ≠ *nil* →
 typExprList_IL el sl →
 typExpr_IL (*a_expr_IL q f el*) *st_iota*
| *q_typ_IL* :
 ∀ (*q* : *Q*) (*e* : *Expr_IL*),
 typExpr_IL e st_iota →
 typExpr_IL (*q_expr_IL q e*) *st_iota*
with *typExprList_IL* : *list Expr_IL* → *list STyp* → *Type* :=
| *nil_typl_IL* : *typExprList_IL nil nil*
| *cons_typl_IL* (*e* : *Expr_IL*) (*s* : *STyp*) (*el* : *list Expr_IL*) (*sl* : *list STyp*) :
 typExpr_IL e s → *typExprList_IL el sl* → *typExprList_IL* (*e* :: *el*) (*s* :: *sl*).

Οι διαφορές σε σχέση με το typing της συναρτησιακής FL έγκεινται αφενός στην εμφάνιση της *q_typ_IL*, που δίνει ground type στην εν λόγω έκφραση (στοιχείο της VIL, όπως ειπώθηκε παραπάνω) και αφετέρου στη χρήση του *pi*, το οποίο διαφέρει ανάλογο με το στάδιο του μετασχηματισμού, οπότε κατά τη χρήση της επαγωγικής δομής *typExpr_IL* θα πρέπει να δίνεται ως παράμετρος.

Inductive *typB_IL* : *B_IL* → *Type* :=
| *E_typ_IL* :
 ∀ *e* : *Expr_IL*,
 typExpr_IL e st_iota → *typB_IL* (*E_IL e*)
| *case_typ_IL* :
 ∀ (*n m* : *nat*) (*el* : *list Expr_IL*),

typExprList_IL el (*repeat_list st_iota n*) → *typB_IL* (*case_IL m el*).

Ο ορισμός του *typB_IL* λαμβάνει υπόψη τον ορισμό του σώματος στο δεύτερο μέλος, που εδώ μπορεί να είναι και *case*.

Inductive *typDef_IL* : *Def_IL* → *Type* :=
| *d_typ_IL* : ∀ (*f* : *Var*) (*xl* : *list Var*) (*b* : *B_IL*),
NoDup xl → *pi f* = *st_arrow* (*map pi xl*) →
typB_IL b → *typDef_IL* (*def_IL f xl b*).

Σε σχέση με το *Def_IL* ισχύουν παρόμοια πράγματα.

Inductive *typProg_IL* : *Prog_IL* → *Type* :=
| *p_nil_IL* : *typProg_IL nil*
| *p_cons_IL* : ∀ (*d* : *Def_IL*) (*Hd* : *typDef_IL d*) (*p* : *Prog_IL*) (*Hp* : *typProg_IL p*),
¬*In* (*lhs_def_IL d*) (*func_IL p*) → *typProg_IL* (*d* :: *p*).

και ο ορισμός του typing του προγράμματος των IL και $NVIL$ είναι πανομοιότυπος με αυτόν της FL.

6.1 Οι νοηματικές γλώσσες IL και $NVIL$: Συγχρονική σημασιολογία

Σε αυτή την ενότητα ορίζουμε τη δηλωτική σημασιολογία των νοηματικών γλωσσών IL και $NVIL$. Το σύνολο των πιθανών κόσμων για τις δύο γλώσσες είναι το σύνολο των ακολουθιών από λίστες φυσικών αριθμών το οποίο συμβολίζουμε $N \rightarrow List(N)$. Προσέξτε ότι, όπως δόθηκε στην ενότητα 3, για το μετασχηματισμό ενός συναρτησιακού προγράμματος M-τάξης , τα contexts πρέπει να είναι ακολουθίες M στοιχείων από λίστες από φυσικούς αριθούς. Ωστόσο, θα θέλαμε η σημασιολογία να δοθεί κάπως γενικά για να μπορεί να εφαρμοστεί για προγράμματα ανεξάρτητα της τάξης τους. Επίσης δεν υπάρχει κάποιο πρόβλημα αν θεωρήσουμε ότι τα contexts είναι ακολουθίες από άπειρες λίστες επειδή ούτως ή άλλως μόνο ένας πεπερασμένος αριθμός λιστών θα χρησμοποιηθεί. Συνεπώς :

Ορισμός 6.4: Το σύνολο W των πιθανών κόσμων των IL και $NVIL$ είναι το σύνολο $N \rightarrow List(N)$.

Δεδομένου του παραπάνω συνόλου W πιθανών κόσμων, μπορούμε να ορίσουμε το σύνολο από πιθανές σημασιολογίες με τύπο σ, ως εξής:

Ορισμός 6.5: Θεωρούμε ένα πεσίο D. Το σύνολο των πιθανών σημασιολογιών για $\sigma \in STyp$ σε σχέση με το W και το D ορίζεται :

$$[\![\sigma]\!]^*_D = W \rightarrow [\![\sigma]\!]_D$$

Με άλλα λόγια, τα στοιχεία της IL με τύπο σ είναι οικογένειες με δείκτη W από συναρτήσεις με τύπο σ πάνω στο D. *Δεν* είναι συναρτήσεις με τύπο σ πάνω στο $W \rightarrow D$, που είναι ένα πιο πολύπλοκο πεδίο. Κατά τον ορισμό της σημασιολογίας των IL και $NVIL$, ακολουθούμε τη μέθοδο που ακολουθήθηκε από τον Montague για τη σημασιολογία των νοηματικών γλωσσών ανώτερης τάξης [Dowt81, Gall75]. Επειδή αυτή η τεχνική διαφέρει από τις συνηθισμένες σημασιολογίες των συναρτησιακών γλωσσών, θα αναφερόμαστε από εδώ και στο εξής σε αυτή με τον όρο *συγχρονική* σημασιολογία για λόγους που θα φανούν παρακάτω.

Έστω D ένα πεδίο. Τότε η σημασιολογία των σταθερών συμβόλων της IL (και $NVIL$) σε σχέση με το D, δίνεται από τη συνάρτηση μετάφρασης $\mathcal{C}^*$, η οποία αναθέτει σε κάθε σταθερά τύπου σ μία συνάρτηση στο $[\![\sigma]\!]^*_D$. Επειδή οι γλώσσες IL και $NVIL$ θα χρησιμοποιηθούν στη διαδικασία μετασχηματισμού των προγραμμάτων της FL, η συνάρτηση $\mathcal{C}^*$ ορίζεται σε σχέση με τη συνάρτηση μετάφρασης $\mathcal{C}$ για την FL. Πιο ειδικά :

Ορισμός 6.6: Για κάθε $\mathbf{c} \in \Sigma$ και για κάθε $w \in W$, $\mathcal{C}^*(\mathbf{c})(w) = \mathcal{C}(\mathbf{c})$.

Τα παραπάνω συνοψίζονται στον απλό κώδικα της απόδειξής μας :

```
Definition World := nat → list nat.

Definition SDom_Star (s : STyp) : Type := World → SDom s.

Definition C_Star (c : Sigma) (w : World) := C_FL c.
```

Η σημασιολογία των νοηματικών τελεστών των γλωσσών IL και $NVIL$ δίνεται από τον ακόλουθο ορισμό :

Ορισμός 6.7: Έστω $w \in (N \to List(N))$ και $a, a_0, \ldots, a_{n-1} \in [\![\sigma]\!]^*$. Η σημασιολογία των νοηματικών τελεστών $call$, $actuals$ και $case$ δίνεται παρακάτω :

$$
\begin{array}{lcl}
call_i^m(a)(w) & = & a(w[m/(i : w_m)]) \\
actuals_i^m(a)(w) & = & \begin{cases} a(w[m/tail(w_m)]) & \text{if } head(w_m) = i \\ undefined & \text{otherwise} \end{cases} \\
case^m(a_0, \ldots, a_{n-1})(w) & = & a_{head(w_m)}(w)
\end{array}
$$

Η *call_aux* επιδρά πάνω στον "κόσμο" *old* και βάσει των προδιαγραφών εισάγει στη *m*-οστή θέση του πίνακα τον ακέραιο *i*, ενώ παράλληλα αφήνει τα υπόλοιπα στοιχεία του πίνακα ως έχουν. Η *call_sem* επιδρά στη σημασιολογική τιμή *a* αποδίδοντας τη σημασιολογία του τελεστή *call*.

```
Definition call_aux (i m : nat) (old : World) : World :=
  fun n : nat ⇒
    match (eq_nat m n) with
      | left _ ⇒ cons i (old m)
      | right _ ⇒ old n
    end.

Definition call_sem (i m : nat) (s : STyp) (a : SDom_Star s) : SDom_Star s :=
  fun w : World ⇒ a (call_aux i m w).
```

Η *actuals_aux* επιδρά πάνω στον "κόσμο" *old* και βάσει των προδιαγραφών αφαιρεί από τη *m*-οστή θέση του πίνακα τον ακέραιο στην κορυφή της λιστας, ενώ παράλληλα αφήνει τα υπόλοιπα στοιχεία του πίνακα ως έχουν. Ο έλεγχος αν υπάρχει ταύτιση του στοιχείου αυτού με την τιμή *i* δεν πραγματοποιείται. Για καλά ορισμένα προγράμματα ο περιορισμός αυτός ικανοποιείται. Η *actuals_sem* επιδρά στη σημασιολογική τιμή *a* αποδίδοντας τη λειτουργία του τελεστή *actuals*.

```
Definition actuals_aux (i m : nat) (old : World) : World :=
  fun n : nat ⇒
    match (eq_nat m n) with
      | left _ ⇒ match (old m) with
                   | nil ⇒ nil
                   | cons i xs ⇒ xs
                 end
      | right _ ⇒ old n
    end.

Definition actuals_sem (i m : nat) (s : STyp) (a : SDom_Star s) : SDom_Star s :=
  fun w : World ⇒ a (actuals_aux i m w).
```

Η *case_sem* υλοποιεί τη λειτουργία του τελεστή *case* προσέχοντας ο ακέραιος στην κεφαλή της λίστας να είναι μικρότερος από τον αριθμό των στοιχείων της λίστας των παραμέτρων του *case*.

```
Definition case_sem (n m : nat) (s : STyp) (dl : tuple SDom_Star (repeat_list s n))
    : SDom_Star s :=
```

```
fun w : World ⇒
  match w m with
    | nil ⇒ sorry
    | h :: _ ⇒
        match lt_ge_dec h n with
          | left Hlt ⇒
              eq_rect (nth h (repeat_list s n) sorry) SDom
            (tuple_nth h (l:=repeat_list s n) dl sorry w) s (nth_of_repeat STyp n h s sorry Hlt)
          | right _ ⇒ sorry
        end
  end.
```

Δεδομένης μίας ακολουθίας $\mathbf{Q} \in ISeq$, ορίζουμε τη σημασία του $\mathbf{Q}$ ως τη σύνθεση των σημασιών των νοηματικών τελεστών που απαρτίζουν το $\mathbf{Q}$. Θα αναφερόμαστε με το Q στη σημασία της ακολουθίας $\mathbf{Q}$. Παρόμοια, θα γράφουμε συχνά $\mathbf{q}$ όταν αναφερόμαστε σε ένα μόνο νοηματικό τελεστή, και με q για τη σημασία του $\mathbf{q}$. Η *Q_sem_aux* αντανακλά την επίδραση των τελεστών στον "κόσμο" αποκλειστικά, ενώ η *Q_sem* επιδεικνύει την επίδραση στην σημασιολογική τιμή *a* γενικότερα.

```
Fixpoint Q_sem_aux (q : Q_aux) (w : World)
                   {struct q} : World :=
  match q with
      | call i m ⇒ call_aux i m w
      | actuals i m ⇒ actuals_aux i m w
      | call_complex i m q ⇒ Q_sem_aux q (call_aux i m w)
      | actuals_complex i m q ⇒ Q_sem_aux q (actuals_aux i m w)
  end.
```

```
Fixpoint Q_sem (q : Q_aux) (s : STyp) (a : SDom_Star s)
                   {struct q} : SDom_Star s :=
  fun w : World ⇒
    match q with
      | call i m ⇒ call_sem i m s a w
      | actuals i m ⇒ actuals_sem i m s a w
      | call_complex i m q ⇒ Q_sem q s a (call_aux i m w)
      | actuals_complex i m q ⇒ Q_sem q s a (actuals_aux i m w)
    end.
```

Μπορούμε τώρα να προχωρήσουμε στη σημασιολογία των εκφράσεων της IL. Έστω Exp_τ το σύνολο των εκφράσεων $\mathbf{B}$ της IL τέτοια ώστε $\mathbf{B} : \sigma$. Έστω Env^*_π το σύνολο των π-συμβατών συγχρονικών περιβαλλόντων ορισμένων από τη σχέση $Env^*_\pi = \prod_{\mathbf{f} \in Var} [\![\pi(\mathbf{f})]\!]^*_D$. Τότε η συγχρονική σημασιολογία της γλώσσας IL ορίζεται από τη συνάρτηση αποτίμησης $[\![\cdot]\!]^* : Exp_\sigma \to [Env^*_\pi \to [\![\sigma]\!]^*_D]$, ως εξής:

Ορισμός 6.8: Η *συγχρονική* ερμηνεία των εκφράσεων της IL σε σχέση με το $u \in Env^*_\pi$ ορίζεται αναδρομικά για κάθε $w \in W$, όπως παρακάτω :

$$
\begin{array}{l}
[\![\mathbf{f}]\!]^*(u)(w) = u(\mathbf{f})(w) \\
[\![\mathbf{c}\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,]\!]^*(u)(w) = \mathcal{C}^*(\mathbf{c})(w)([\![\mathbf{E}_0]\!]^*(u)(w), \ldots, [\![\mathbf{E}_{n-1}]\!]^*(u)(w)) \\
[\![\mathbf{Q}\,(\mathbf{f})\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,]\!]^*(u)(w) = Q(u(\mathbf{f}))(w)([\![\mathbf{E}_0]\!]^*(u)(w), \ldots, [\![\mathbf{E}_{n-1}]\!]^*(u)(w)) \\
[\![\mathtt{case}^m\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,]\!]^*(u)(w) = case^m([\![\mathbf{E}_0]\!]^*(u), \ldots, [\![\mathbf{E}_{n-1}]\!]^*(u))(w)
\end{array}
$$

Μπορεί να φανεί από τον παραπάνω ορισμό ότι η σημασιολογική συνάρτηση για τη γλώσσα L δεν είναι η συνήθης. Εμπλέκει την εύρεση της σημασιολογίας των υποεκφράσεων υπό το τρέχον context w.

Η βασική αρχή είναι ότι η τιμή του $\mathbf{f}\,(\mathbf{E})$ στον κόσμο w είναι η τιμή του $\mathbf{f}$ στον κόσμο w εφαρμοσμένη στην τιμή του $\mathbf{E}$ στον ίδιο κόσμο w . Αν θεωρήσουμε το w σαν μια "χρονική στιγμή", διαπιστώνουμε ότι η τιμή του $\mathbf{f}\,(\mathbf{E})$ τη "στιγμή" w εξαρτάται από την τιμή του $\mathbf{E}$ την ίδια στιγμή w. Για αυτό υιοθετήθηκε ο όρος "συγχρονικός".

Για τον ορισμό της σημασιολογικής συνάρτησης χρειάστηκε η βοηθητική συνάρτηση *tuple_apply*, η οποία για μία λίστα από αντικείμενα τύπου *SDom_Star s* με το *s* μεταβαλλόμενο εφαρμόζει τον κόσμο *w* σε αυτά.

```
Fixpoint tuple_apply (l : list STyp) (w : World) : tuple SDom_Star l → tuple SDom l :=
  match l return tuple SDom_Star l → tuple SDom l with
    | nil ⇒ fun (_ : tuple SDom_Star nil) ⇒ tt
    | x :: l ⇒ fun (tup : tuple SDom_Star (x::l)) ⇒ (fst tup w, tuple_apply l w (snd tup))
  end.
```

Η διαδικασία προσδιορισμού της είναι παρόμοια με αυτή που ακολουθήθηκε στην ενότητα, όπου ορίστηκε η σημασιολογία για τη γλώσσα FL. Στην ουσία ορίζεται ταυτόχρονα η σημασιολογία για μία έκφραση *e* με καλά ορισμένο τύπο *s* και για μία λίστα *el* με επίσης καλά ορισμένους τύπους σε ένα περιβάλλον.

```
Definition sem_il_expr_list (pi : Var → STyp) :
  (∀ (e : Expr_IL) (s : STyp), typExpr_IL pi e s → Env pi SDom_Star→
      SDom_Star s) ×
  (∀ (el : list Expr_IL) (sl : list STyp), typExprList_IL pi el sl →
      Env pi SDom_Star → tuple SDom_Star sl).
Proof.
  intros .
  apply typing_expr_IL_list_mutrec .
  intros f u.
  exact (u f).
  intros c el n H1 H2 dl u w _.
  apply (cast (P := SDom_Star) (C_Star c ) H1) .
  apply w. exact (tuple_apply w (dl u)).
  intros q f el sl H1 H2 H3 dl u w _.
  apply (cast (u f) H1).
  apply w . exact (tuple_apply w (dl u)).
  intros q e t dl u.
  auto.
  intro u.
  exact tt.
  intros e s el sl H1 d H2 dl u.
  exact (d u, dl u).
Defined.
```

Ξεχωρίζοντας την τούπλα της *sem_il_expr_list* παίρνουμε τις *sem_il* και *sem_il_args*.

```
Definition sem_il (pi : Var → STyp) : ∀ (e : Expr_IL) (s : STyp),
  typExpr_IL pi e s → Env pi SDom_Star → SDom_Star s :=
    fst (sem_il_expr_list pi).

Definition sem_il_args (pi : Var → STyp) : ∀ (el : list Expr_IL) (sl : list STyp),
  typExprList_IL pi el sl → Env pi SDom_Star → tuple SDom_Star sl :=
    snd (sem_il_expr_list pi).
```

Στην περίπτωση της $NVIL$, έχουμε την παρακάτω σημασιολογική εξίσωση, η οποία σε επίπεδο κώδικα δε θα μας απασχόλησε ιδιαίτερα :

$$[\![\mathbf{Q}\,(\mathbf{E})\,]\!]^*(u)(w) = Q([\![\mathbf{E}]\!]^*(u))(w)$$

Πριν εισάγουμε τη σημασιολογία των προγραμμάτων , ο ακόλουθος ορισμός είναι απαραίτητος :

Ορισμός 6.9: Έστω $d \in [\![\sigma]\!]_D$. Τότε, d^∞ είναι η συνάρτηση στο W της οποίας η τιμή σε κάθε $w \in W$ είναι ίση με d.

Η d^∞ αντιστοιχεί στη συνάρτησή *tuple_abstract* :

```
Fixpoint tuple_abstract (l : list STyp) : tuple SDom l → tuple SDom_Star l :=
  match l return tuple SDom l → tuple SDom_Star l with
    | nil ⇒ fun (_ : tuple SDom nil) ⇒ tt
    | x :: l ⇒ fun (tup : tuple SDom (x::l)) ⇒
        (fun (w : World) ⇒ fst tup, tuple_abstract l (snd tup))
  end.
```

Μπορούμε τώρα να εισάγουμε τη σημασιολογία της IL. Προσέξτε ότι οι παρακάτω ορισμοί και θεωρήματα μπορούν να εφαρμοστούν σε $NVIL$ προγράμματα (η διαφορά είναι ότι τα $NVIL$ προγράμματα επιτρέπουν ορισμούς με μεταβλητές μηδενικής τάξης στο αριστερό μέρος των ορισμών).

Ορισμός 6.10: Η συγχρονική σημασιολογία των προγραμμάτων $\mathbf{P} = \{\mathbf{F}_0, \ldots, \mathbf{F}_{n-1}\}$ της IL (ή της $NVIL$) ορίζεται $\widetilde{u}(\mathbf{result})$, όπου $\widetilde{u}$ είναι το *ελάχιστο* περιβάλλον τέτοια ώστε για κάθε ορισμό $\mathbf{f}\,(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B}$ στο $\mathbf{P}$ με $\mathbf{f} : (\sigma_0, \ldots, \sigma_{n-1}) \to \iota$ για όλα τα $d_0 \in [\![\sigma_0]\!]_D, \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]_D$, και για $w \in W$,
$\widetilde{u}(\mathbf{f})(w)(d_0, \ldots, d_{n-1}) = [\![\mathbf{B}]\!]^*(\widetilde{u}[\mathbf{x}_0/d_0^\infty, \ldots, \mathbf{x}_{n-1}/d_{n-1}^\infty])(w)$.

Ο παραπάνω ορισμός δεν διευκρινίζει πώς κατασκευάζεται το ελάχιστο περιβάλλον $\widetilde{u}$. Το παρακάτω θεώρημα ορίζει ότι $\widetilde{u}$ είναι το ανώτερο άνω φράγμα μίας αλυσίδας από περιβάλλοντα που μπορούμε να θεωρήσουμε ως διαδοχικές προσεγγίσεις του $\widetilde{u}$.

Η σημασιολογία ενός σώματος *b* δίνεται με ανάλυση στον τύπο του σώματος, δηλαδή αν πρόκειται γιά απλή έκφραση ή για *case*.

```
Definition sem_B_IL (pi : Var → STyp) (b : B_IL) (H : typB_IL pi b)
                          (u : Env pi SDom_Star) : SDom_Star st_iota :=
  match H in typB_IL _ b return SDom_Star st_iota with
| E_typ_IL e typ ⇒
      sem_il e typ u
| case_typ_IL n m el typlist ⇒
      case_sem m (sem_il_args el typlist u)
  end.
```

Η ανάλυση από εδώ και στο εξής είναι όμοια με την περίπτωση της γλώσσας FL με μόνη διαφοροποίηση στη διάσταση που εισάγει ο εκάστοτε "κόσμος" w. Αν ανατρέξει κανείς στην ενότητα με τη σημασιολογία της FL θα βρει τη σημασία των παρακάτω συναρτήσεων :

```
Definition sem_def_IL (pi : Var → STyp) (d : Def_IL) (H : typDef_IL pi d)
                          (u : Env pi SDom_Star) : SDom_Star (pi (lhs_def_IL d)) :=
  match H in typDef_IL _ d return SDom_Star (pi (lhs_def_IL d)) with
    | d_typ_IL f xl b _ Hpar Ht ⇒
        fun w ⇒
          func_make (fun dl : dom_args (pi f) ⇒
            (sem_B_IL pi b Ht
              (env_update_list pi SDom_Star xl (tuple_abstract (cast dl Hpar)) u) w) tt)
  end.

Definition sem_update_def_IL (pi : Var → STyp) (u : Env pi SDom_Star) (d : Def_IL)
                                 (H : typDef_IL pi d) : Env pi SDom_Star → Env pi SDom_Star.
Proof.
```

```
  intros pi0 u d H.
  exact (env_update pi0 SDom_Star (lhs_def_IL d) (sem_def_IL pi0 d H u)).
Defined.

Definition sem_update_prog_IL (pi : Var → STyp) (u : Env pi SDom_Star ) (p : Prog_IL)
                              (H : typProg_IL pi p) (e : Env pi SDom_Star) : Env pi SDom_Star.
Proof.
  fix 4.
  intros.
  destruct H.
  exact e.
  exact (sem_update_def_IL u d Hd (sem_update_prog_IL pi u p H e)).
Defined.
```

Η αντίστοιχη συνάρτηση της *SDombot* με την επιπλέον διάσταση *World*

```
Fixpoint SDombot_Star (s : STyp) : SDom_Star s :=
  match s return SDom_Star s with
    | st_arrow sl ⇒ fun (w : World) (dl : tuple SDom sl) ⇒ Dbot
  end.
```

και το ελάχιστο περιβάλλον για την περίπτωση της IL γλώσσας *sem_prog_aux_IL*, ενώ η σημασιολογία του προγράμματος δίνεται από τη συνάρτηση *sem_prog_IL*.

```
Definition Ebot_Star (pi : Var → STyp) := Ebot pi SDom_Star SDombot_Star .

Definition sem_prog_aux_IL (pi : Var → STyp)(p : Prog_IL) (H : typProg_IL pi p)
    : Env pi SDom_Star :=
  Tarski_fix (fun (u : Env pi SDom_Star) ⇒ sem_update_prog_IL u p H (Ebot_Star pi)).

Definition sem_prog_IL (pi : Var → STyp)(p : Prog_IL) (H : typProg_IL pi p)
    : SDom_Star st_iota :=
  sem_il (f_expr_IL result) (typ_result_IL pi) (sem_prog_aux_IL p H).
```

Θεώρημα 6.1: Έστω $\mathbf{P}$ και $\widetilde{u}$ όπως στον ορισμό 6.10. Τότε $\widetilde{u}$ είναι το ελάχιστο ανώτερο φράγμα των περιβαλλόντων $\widetilde{u}_k$, $k \in N$, τα οποία ορίονται παρακάτω :

1. Για κάθε $\mathbf{f} \in Var$ με $\mathbf{f} \notin func(\mathbf{P})$, για κάθε $w \in W$, $\widetilde{u}_k(\mathbf{f})(w) = \bot$, για όλα $k \in N$.
2. Για κάθε $\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B}$ στο $\mathbf{P}$, με $\mathbf{f} : (\sigma_0, \ldots, \sigma_{n-1}) \to \iota$, για επίσης για όλα τα $d_0 \in [\![\sigma_0]\!]_D, \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]_D$, και όλα τα $w \in W$,

$$\begin{aligned} \widetilde{u}_0(\mathbf{f})(w)(d_0, \ldots, d_{n-1}) &= \bot \\ \widetilde{u}_{k+1}(\mathbf{f})(w)(d_0, \ldots, d_{n-1}) &= [\![\mathbf{B}]\!]^*(\widetilde{u}_k[\mathbf{x}_0/d_0^\infty, \ldots, \mathbf{x}_{n-1}/d_{n-1}^\infty])(w) \end{aligned}$$

```
Fixpoint uk_IL (pi : Var → STyp)(p : Prog_IL) (H : typProg_IL pi p) (k : nat) {struct k}
    : Env pi SDom_Star :=
  match k return Env pi SDom_Star with
    | 0 ⇒ fun f : Var ⇒ SDombot_Star (pi f)
    | S n ⇒ sem_update_prog_IL (uk_IL pi p H n) p H (Ebot_Star pi)
  end.
```

Επίσης, για κάθε $k \in N$, $\widetilde{u}_k(\mathbf{f}) \sqsubseteq \widetilde{u}_{k+1}(\mathbf{f})$.

Απόδειξη : Ανάλογο με την απόδειξη του Θεωρήματος 5.2. ∎

Το παρακάτω λήμμα είναι μία άμεση συνέπεια του παραπάνω θεωρήματος :

Λήμμα 6.1: Εστω **P** και $\widetilde{u}$ όπως στον Ορισμό 6.10. Τότε για κάθε $\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B}$ στο **P** με $\mathbf{f} : (\sigma_0, \ldots, \sigma_{n-1}) \to \iota$, για κάθε $d_0 \in [\![\sigma_0]\!]_D, \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]_D$ και για όλα $w \in W$,

$$\widetilde{u}_k(\mathbf{f})(w)(d_0, \ldots, d_{n-1}) \quad \sqsubseteq \quad [\![\mathbf{B}]\!]^*(\widetilde{u}_k[\mathbf{x}_0/d_0^\infty, \ldots, \mathbf{x}_{n-1}/d_{n-1}^\infty])(w)$$

```
Lemma lemma_6_1 :
 ∀ (p : Prog_IL)(pi : Var → STyp)(Hp : typProg_IL pi p) (k : nat)
          (d : Def_IL) (Hd : typDef_IL pi d) (xl : list Var)(H :NoDup xl)
          (w : World)(Heq : pi (lhs_def_IL d) = st_arrow (map pi xl))
          (args : dom_args (pi (lhs_def_IL d))),
    In d p →
     Drel (func_apply ((uk_IL pi p Hp (S k) (lhs_def_IL d)) w) args)
          (func_apply ((sem_def_IL pi d Hd (uk_IL pi p Hp k )) w) args) .
Proof.
Admitted.
```

Το παρακάτω θεώρημα θα χρησιμοποιηθεί σε επόμενες ενότητες :

Θεώρημα 6.2: Όλες τις εκφράσεις $\mathbf{B} \in Exp_\sigma$, $[\![\mathbf{B}]\!]^*$ είναι μονότονες και συνεχείς. Επίσης, όταν $\sigma \neq \iota$, $[\![\mathbf{B}]\!]^*(u)(w)$ είναι μονότονη και συνεχής , για όλα τα $u \in Env_\pi^*$ και $w \in W$.

```
Variable SDomrel_Star : ∀ s : STyp, relation (SDom_Star s).
Definition Erel_Star (pi : Var → STyp) := Erel pi SDom_Star SDomrel_Star.

Theorem th_6_2 :
  ∀ (p : Prog_IL)(pi : Var → STyp)(Hp : typProg_IL pi p) (k : nat)
          (e : Expr_IL)(s : STyp) (typE : typExpr_IL pi e s) ,
 continuous (Erel_Star pi) (SDomrel_Star s)(sem_il e typE).
Proof.
Admitted.
```

6.2 Ιδιότητες της συγχρονικής σημασιολογίας

Σε αυτή την υποενότητα εξετάζουμε ορισμένες ιδιότητες της συγχρονικής σημασιολογίας. Αρχικά, θεωρούμε τα προγράμματα της IL που δεν περιέχουν τους τελεστές `call`, `actuals` και `case`. Προσέξτε ότι προγράμματα αυτού του υποσυνόλου είναι στην ουσία προγράμματα της FL για τα οποία έχουμε ορίσει ήδη την standard δηλωτική σημασιολογία (δείτε τον ορισμό 5.8). Το παρακάτω θεώρημα εισάγει τη σχέση μεταξύ της standard και της συγχρονικής σημασιολογίας για προγράμματα του παραπάνω υποσυνόλου :

Θεώρημα 6.3: Έστω **P** ένα πρόγραμμα IL που δεν περιέχει τους τελεστές `call`, `actuals` και `case`, και έστω u και $\widehat{u}$ τα ελάχιστα περιβάλλοντα που ικανοποιούν τους ορισμούς της **P** υπό τη standard και τη συγχρονική προσέγγιση αντίστοιχα . Τότε για κάθε $w \in W$, ισχύει $[\![\mathbf{P}]\!]^*(\widehat{u})(w) = [\![\mathbf{P}]\!](u)$.

Απόδειξη : Αρκεί να δείξουμε ότι για κάθε ορισμό $\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B_f}$ στο **P**, με την υπόθεση $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$,

$$\widehat{u}(\mathbf{f})(w)(d_0, \ldots, d_{n-1}) = u(\mathbf{f})(d_0, \ldots, d_{n-1})$$

για όλα τα $d_0 \in [\![\sigma_0]\!]_D, \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]_D$, και όλα τα $w \in W$. Αυτό μπορεί να αποδειχθεί με μία διπλή επαγωγή : μία εξωτερική επαγωγή στις προσεγγίσεις του $\widehat{u}$ και του u και μία εσωτερική επαγωγή στο σώμα της **f**. ∎

Θεωρήστε τώρα τα προγράμματα της $NVIL$. Για αυτά τα προγράμματα, μία standard δηλωτική σημασιολογία $[\![\cdot]\!]_{(W \to D)}$ μπορεί να οριστεί εύκολα, το οποίο έγινε στο [Rond97b][page 85]. Το επόμενο θεώρημα δείχνει ότι η standard και η συγχρονική σημασιολογία συμπίπουν στην περίπτωση της $NVIL$.

Θεώρημα 6.4: Έστω **P** ένα $NVIL$ πρόγραμμα και έστω u και $\widehat{u}$ τα ελάχιστα περιβάλλοντα που ικανοποιούν τους ορισμούς του **P** υπό τη standard και τη συχρονική προσέγγιση αντίστοιχα. Τότε $[\![\mathbf{P}]\!]^*(\widehat{u}) = [\![\mathbf{P}]\!]_{(W \to D)}(u)$.

Απόδειξη : (Περίληψη) Αρκεί να δείξουμε ότι για κάθε συνάρτηση μηδενικής τάξης **f** που έχει κάποιο ορισμό στο **P**,ισχύει $\widehat{u}(\mathbf{f}) = u(\mathbf{f})$. Αυτό προκύπτει άμεσα με μία διαδικασία παρόμοια αυτής στο θεώρημα 6.3. ■

Προκύπτει από τα δύο θεωρήματα παραπάνω , ότι για να δείξουμε την ορθότητα του μετασχηματισμού αρκεί να περιοριστούμε στη συγχρονική προσέγγιση. (βλέπε 8.5 παρακάτω).

Ο κώδικας μας δεν έχει προχωρήσει σε σημείο που να μπορούμε να επιδείξουμε κάτι αξιόλογο για την ενότητα αυτή.

Κεφάλαιο 7

Τυπικός Ορισμός του αλγόριθμου μετασχηματισμού

Ο σκοπός της ενότητας αυτής είναι να οριστεί με τυπικό τρόπο ο αλγόριθμος μετασχηματισμού από συναρτησιακά προγράμματα ανώτερης τάξης σε νοηματικά μηδενικής τάξης. Ο αλγόριθμος αποτελείται από ένα αριθμό από βήματα. Σε κάθε βήμα, η τάξη του αρχικού προγράμματος μειώνεται κατά ένα.Ο αλγόριθμος ολοκληρώνεται όταν αποκτάται νοηματικό πρόγραμμα μηδενικής τάξης. Πιο ειδικά, η είσοδος στον αλγόριθμο είναι ένα πρόγραμμα M-τάξης με M>0. Μετά από το πρώτο στάδιο του μετασχηματισμού, παράγεται ένα $(M-1)$-τάξης πρόγραμμα. Μετά από M βήματα, το μηδενικής τάξης $NVIL$ πρόγραμμα είναι το ζητούμενο.

Συνεπώς, αρκεί να περιγράψουμε ένα μόνο στάδιο του μετασχηματισμού, δηλαδή τη διαδικασία που απαιτείται για το μετασχηματισμό του νοηματικού προγράμματος m-τάξης για $(1 \leq m \leq M)$ σε πρόγραμμα $(m-1)$-τάξης. Παρατηρήστε ότι αυτή η διαδικασία εφαρμόζεται και για το πρώτο στάδιο του μετασχηματισμού, επειδή μπορούμε να θεωρήσουμε το πηγαίο FL πρόγραμμα σαν ένα πρόγραμμα IL το οποίο τυχαίνει να μην έχει νοηματικούς τελεστές.

Ένα στάδιο του αλγορίθμου μπορεί διαισθητικά να περιγραφεί παρακάτω : δεδομένου ενός προγράμματος εισόδου m-τάξης, ξεκινάμε θεωρώντας τις m-τάξης συναρτήσεις που που ορίζονται σε αυτό. Ο στόχος είναι να μειωθεί η τάξη των συναρτήσεων καταργώντας τις τυπικές παραμέτρους $(m-1)$-τάξης, επεξεργαζόμενοι κάθε στιγμή όλες τις κλήσεις σε κάθε **f** του προγράμματος. Για κάθε τυπική παράμετρο που αφαιρείται από τη λίστα των τυπικών παραμέτρων της **f**, ένας νέος ορισμός δημιουργείται και εισάγεται στο πρόγραμμα. Κάθε ορισμός συγκεντρώνει όλες τις πραγματικές παραμέτρους που αντιστοιχούν στη συγκεκριμένη τυπική παράμετρο και εμφανίζονται στις κλήσεις της **f** μέσα στο πρόγραμμα. Αυτή η λειτουργία επιτυγχάνεται με τη χρήση των τελεστών `case` και `actuals`.

Κατ'αυτό τον τρόπο, η είσοδος του αλγορίθμου,το m-τάξης πρόγραμμα μετασχηματίστηκε σε ένα $(m-1)$-τάξης πρόγραμμα. Η διαδικασία που περιγράφηκε μπορεί να χρησιμοποιηθεί επαναληπτικά μέχρις ότου όλες οι τυπικές παράμετροι εξαλειφθούν. Το τελικό πρόγραμμα θα είναι ένα πρόγραμμα που αποτελείται από ένα σύνολο από νοηματικούς ορισμούς μηδενικής τάξης.

7.1 Προκαταρκτκοί Ορισμοί

Σε αυτή την υποενότητα παρέχουμε ορισμένους προκαταρκτικούς ορισμούς που θα μας φανούν χρήσιμοι για τον ορισμό του αλγόριθμου μετασχηματισμού.

Ορισμός 7.1: Έστω **q** ένας νοηματικός τελεστής. Ο αντίστροφος του νοηματικού τελεστή **q** συμβολίζεται $\mathbf{q}^{-1}$ και ορίζεται παρακάτω :

$$\mathbf{q}^{-1} \quad = \quad \begin{cases} \texttt{call}_i^m & \text{if } \mathbf{q} = \texttt{actuals}_i^m \\ \texttt{actuals}_i^m & \text{if } \mathbf{q} = \texttt{call}_i^m \end{cases}$$

Δεδομένου του **q**, συχνά θα γράφουμε q για τη σημασιολογία του **q** και q^{-1} για τη σημασιολογία του $\mathbf{q}^{-1}$. Μπορεί εύκολα να φανεί ότι όποτε η σύνθεση του q και του q^{-1} εμφανίζεται σε οποιαδήποτε σειρά στο πρόγραμμα, τότε μπορεί να αντικατασταθεί από την ταυτοτική συνάρτηση.

Ορισμός 7.2: Έστω $\mathbf{Q} = \mathbf{q}_0 \cdot \mathbf{q}_1 \cdot \ldots \cdot \mathbf{q}_{r-1} \in ISeq$. Τότε, η *αντίστροφη ακολουθία* του **Q** είναι η $\mathbf{Q}^{-1} = \mathbf{q}_{r-1}^{-1} \cdot \ldots \cdot \mathbf{q}_1^{-1} \cdot \mathbf{q}_0^{-1}$.

Η συνάρτηση *Q_aux_list* μετατρέπει μία δομή *q_aux* σε λίστα απλών τελεστών *q_aux* αντεστραμμένων, δηλαδή τελεστών *call* στη θέση των *actuals* και τελεστών *actuals* στη θέση των *call*.

```
Fixpoint Q_aux_list (q_aux : Q_aux) : list Q_aux :=
  match q_aux with
  | call m i ⇒ cons ( actuals m i) nil
  | actuals m i ⇒ cons (call m i) nil
  | call_complex m i q ⇒ cons (actuals m i) (Q_aux_list q)
  | actuals_complex m i q ⇒ cons (call m i) (Q_aux_list q)
  end.
```

Η συνάρτηση *Q_aux_to_Q_aux_list* μετατρέπει μία δομή *q_aux* σε λίστα απλών τελεστών *q_aux*, όπως εμφανίζονται στη δομή *q_aux*.

```
Fixpoint Q_aux_to_Q_aux_list (q_aux : Q_aux) : list Q_aux :=
  match q_aux with
  | call m i ⇒ cons ( call m i) nil
  | actuals m i ⇒ cons (actuals m i) nil
  | call_complex m i q ⇒ cons (call m i) (Q_aux_to_Q_aux_list q)
  | actuals_complex m i q ⇒ cons (actuals m i) (Q_aux_to_Q_aux_list q)
  end.
```

Η *Q_aux_list2* είναι η αντίστροφη της *Q_aux_to_Q_aux_list*, δηλαδή παράγει μία δομή *q_aux* από λίστα απλών τελεστών *q_aux*.

```
Fixpoint Q_aux_list2 (l : list Q_aux) : Q_aux :=
  match l with
  | nil ⇒ sorry | cons q_aux ls ⇒ match q_aux, ls with
                          | call m i , nil ⇒ call m i
                          | actuals m i , nil ⇒ actuals m i
                          | call m i , ls' ⇒ call_complex m i (Q_aux_list2 ls')
                          | actuals m i, ls' ⇒ actuals_complex m i (Q_aux_list2 ls')
                          | _ , ls' ⇒ sorry                          end
  end.
```

Η *Q_invert* αντιστρέφει τη σύνθεση τελεστών, όπως ορίστηκε παραπάνω.

```
Fixpoint Q_invert (l : Q) : Q :=
  match l with
  | nothing ⇒ nothing
  | complex q_aux ⇒ complex ( Q_aux_list2 (rev (Q_aux_list q_aux)))
  end.
```

Έστω **P** πρόγραμμα m-τάξης. Στα παρακάτω, θα θεωρούμε ότι η διάταξη των ορισμών είναι λεξικογραφική στο **P**. Αυτό θα μας επιτρέπει να μιλάμε για σειρά εμφάνισης της κλήσης στο πρόγραμμα. Έστω $Sub(\mathbf{P})$ το σύνολο των υποεκφράσεων του **P**. Υιοθετούμε τις ακόλουθες συμβάσεις.

- Έστω **f** μία συνάρτηση m-τάξης ορισμένης στο $\mathbf{P}$. Το σύνολο των κλήσεων στην **f** μέσα στο **P** ορίζεται:

$$calls(\mathbf{P}, \mathbf{f}) = \{\mathbf{Q}\ (\mathbf{f})\ (\mathbf{E}_0, \ldots, \mathbf{E}_{n-1}) \in Sub(\mathbf{P})\}$$

  Επειδή για τον ορισμό της calls χρειαζόμαστε μόνον αντικείμενα της μορφής *a_expr_IL* ορίζουμε τη δομή *Calls_data_type*, η οποία κρατά ένα σύνολο τέτοιων αντικειμένων.

```
Inductive Calls_data_type : Type :=
 | nil_calls : Calls_data_type
```

```
  | cons_calls : Call → Calls_data_type → Calls_data_type
with Call : Type :=
| call_def : Q → Var → list Expr_IL → Call
.
```

Η συνάρτηση που συνενώνει δύο τέτοιες δομές :

```
Definition append_calls (c1 : Calls_data_type)(c2 : Calls_data_type) : Calls_data_type.
Proof.
intros.
induction c1;[assumption|constructor].
Defined.
```

Η *concat_calls* έχει λειτουργία παρόμοια με αυτή της *concat* για παραδοσιακές λίστες :

```
Definition concat_calls (l : list Calls_data_type) : Calls_data_type.
Proof.
intros.
induction l;[exact nil_calls|inversion a;[assumption|apply cons_calls;assumption]].
Defined.
```

Με την παρακάτω αλληλουχία συναρτήσεων προσδιορίζουμε το σύνολο των υποεκφράσεων *sub* για ένα πρόγραμμα της γλώσσας IL.

```
Fixpoint sub_expr (e : Expr_IL) : Calls_data_type :=
  match e with
    | f_expr_IL f ⇒ nil_calls
    | c_expr_IL c el ⇒ concat_calls ( map sub_expr el )
    | a_expr_IL q f el ⇒ append_calls ( cons_calls (call_def q f el) nil_calls )
                                                    ( concat_calls (map sub_expr el) )
    | q_expr_IL q e' ⇒ sub_expr e'
  end.


Definition sub_body (b : B_IL) : Calls_data_type :=
  match b with
    | E_IL e ⇒ sub_expr e
    | case_IL m el ⇒ concat_calls (map sub_expr el)
  end.


Definition sub_def (d : Def_IL) : Calls_data_type :=
  match d with
    | def_IL f xl b ⇒ sub_body b
  end.


Definition sub (p : Prog_IL) : Calls_data_type := concat_calls (map sub_def p).
```

Η *is_call_of* ελέγχει αν η έκφραση είναι κλήση στη συνάρτηση *f*.

```
Definition is_call_of (f : Var) (e : Expr_IL) : bool :=
  match e with
    | f_expr_IL f' ⇒
        false       | a_expr_IL q f' el ⇒
        match var_eq_dec f f' with
```

```
        | left _ ⇒ true
        | right _ ⇒ false
        end
    | _ ⇒ false
  end.
```

Η *calls_aux* από το σύνολο *c* επιλέγει μόνον εκείνα τα *calls*, που είναι κλήσεις στην *f* με απώτερο στόχο τον ορισμό της συνάρτησης *calls*.

```
Fixpoint calls_aux (c : Calls_data_type) (f : Var) {struct c}: Calls_data_type :=
match c with
| nil_calls ⇒ nil_calls
| cons_calls (call_def q v el) calls_rest ⇒
    if (is_call_of f (a_expr_IL q v el)) then
        cons_calls (call_def q v el) (calls_aux calls_rest f)
    else calls_aux calls_rest f
end.

Definition calls (P : Prog_IL)(f : Var) := calls_aux (sub P) f.
```

- Έστω **f** μία συνάρτηση m-τάξης ορισμένης στο $\mathbf{P}$. Έστω $\mathbf{C}_0, \ldots, \mathbf{C}_{r-1}$ οι κλήσεις στην **f** σε σειρά ίδια με αυτή που εμφανίζονται στο πρόγραμμα **P**. Η συνάρτηση $label$ αναθέτει φυσικούς αριθμούς στις κλήσεις της **f** κατά τέτοιο τρόπο ώστε διαφορετικές κλήσεις να δέχονται διαφορετικές ταμπέλες (labels):

$$label(\mathbf{P}, \mathbf{f}, \mathbf{C}_i) \quad = \quad \min\{j | \mathbf{C}_i = \mathbf{C}_j\}$$

Κατ'αυτό τον τρόπο, οι κλήσεις σε συναρτήσεις αριθμούνται με σειρά αυτή των εμφανίσεών τους στο **P**, εκτός από τις κλήσεις που έχουν περισσότερες από μία εμφανίσεις στο **P**, οι οποίες αποκτούν το label της πρώτης εμφάνισής τους [1].

Η *In_calls* ελέγχει αν η κλήση *t* ανήκει στο σύνολο *c*.

```
Fixpoint In_calls (c : Calls_data_type)(t : Call){struct c} : Prop :=
match c with
| nil_calls ⇒ False
| cons_calls t' c' ⇒
    match eq_Call t t' with
    | left _ ⇒ t = t'
    | right _ ⇒ In_calls c' t
    end
end.
```

Η *fold_right_calls* έχει παρόμοια λειτουργία με την *fold_right* για λίστες.

```
Fixpoint fold_right_calls (A : Type)(c : Calls_data_type)(a0 : A)(f : A → Call → A)
{struct c} : A :=
  match c with
  | nil_calls ⇒ a0
  | cons_calls c calls_rest ⇒ fold_right_calls A calls_rest (f a0 c) f
  end.
```

[1] Αυτός είναι ένας ελαφρά διαφορετικός τρόπος αρίθμησης από αυτόν που δόθηκε στην πρόχειρη παρουσίαση του μετασχηματισμού στην ενότητα 3. Παρ'όλα αυτά και οι δύο τρόποι δουλεύουν εξίσου καλά στην πράξη.

Η *calls_length* προσδιορίζει το πλήθος των αντικειμένων της δομής *c*

```
Fixpoint calls_length_aux (c : Calls_data_type)(i : nat) {struct c} : nat :=
 match c with
 | nil_calls ⇒ i
 | cons_calls cal calls_rest ⇒ calls_length_aux calls_rest (i+1)
 end.

Definition calls_length (c : Calls_data_type) := calls_length_aux c 0.
```

για να φθάσουμε στον ορισμό της συνάρτησης *label* :

```
Definition label (P : Prog_IL) (f : Var) (Ci : Expr_IL) : nat :=
  let ln := calls P f in
    snd (fold_right_calls ln (calls_length ln, calls_length ln)
          (fun (pr : (nat×nat)%type) (e : Call) ⇒
            let (counter, rest) := pr in
              match e with
              | call_def q v el ⇒
                  match expr_eq_dec (a_expr_IL q v el) Ci with
                  | left _ ⇒ (pred counter, counter)
                  | right _ ⇒ (pred counter, rest)
                  end
              end) ).
```

- Έστω $\mathbf{f}$ μία συνάρτηση ορισμένη στο πρόγραμμα $\mathbf{P}$. Η λίστα με τις θέσεις των τυπικών παραμέτρων της $\mathbf{f}$ με τάξη μικρότερη του $(m-1)$, συμβολίζεται με $low(\mathbf{f}, m)$. Η λίστα είναι ταξινομημένη με αύξουσα σειρά. Για παράδειγμα, αν η μηδενική και τρίτη παράμετρος της $\mathbf{f}$ έχουν τάξη μικρότερη από $(m-1)$, τότε θα έχουμε $low(\mathbf{f}, m) = [0, 3]$.

- Έστω $\mathbf{f}$ μία συνάρτηση ορισμένη στο πρόγραμμα $\mathbf{P}$ και έστω $\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}$ οι τυπικές παράμετροι $\mathbf{f}$. Τότε το σύνολο των τυπικών παραμέτρων της $\mathbf{f}$ που έχουν τάξη ίση με $(m-1)$, παριστάνεται με $high(\mathbf{f}, m)$. Για παράδειγμα , αν η πρώτη και τέταρτη παράμετρος της $\mathbf{f}$ είναι τάξης $(m-1)$, τότε $high(\mathbf{f}, m) = \{\mathbf{x}_1, \mathbf{x}_4\}$. Προσέξτε ότι η $high$ έχει διαφορετικό τύπο αποτελέσματος από τη low που ορίστηκε πιο πάνω.

  Διαφοροποιηθήκαμε στον κώδικά μας σε σχέση με τους ορισμούς low. Στη θέση τους χρησιμοποιήθηκε η υπόθεση που αφορά τις παραμέτρους μίας οποιαδήποτε μεταβλητής f του προγράμματος και η οποία χρησιμοποιείται σαν βασική υπόθεση των θεωρημάτων της απόδειξης σε παρακάτω ενότητες. Συγκεκριμένα, ισχύουν τα εξής : Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B_f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ και υπάρχει $0 \leq l \leq n-1$ τ.ώ. $order(\sigma_0) = (m-1), \ldots, order(\sigma_{l-1}) = (m-1)$ και $order(\sigma_l) < (m-1), \ldots, order(\sigma_{n-1}) < (m-1)$.

  Ο προσδιορισμός του l για κάθε παράμετρο γίνεται με τη βοήθεια της *f*. Η λειτουργία της *find_l_2* είναι παρόμοια με μόνη διαφορά ότι στη θέση της f δίνεται η λίστα των τύπων *sl* που αντιστοιχούν στις τυπικές παραμέτρους της*f*.

```
Fixpoint find_l_aux (m : nat) (sl : list STyp) (l : nat) {struct sl} : nat :=
  match sl with
    | nil ⇒ l
    | s::sl ⇒
        match eq_nat (order s) (m-1) with
          | left _ ⇒ find_l_aux m sl (l+1)
          | right _ ⇒ l              end
```

end.

Definition *find_l* (*m* : *nat*) (*f* : *Var*)
:= *find_l_aux m* (*type_args* (*pi f*)) 0.

Definition *find_l_2* (*m* : *nat*) (*sl* : *list STyp*)
:= *find_l_aux m sl* 0.

Fixpoint *high_aux* (*i* : *nat*)(*vl* : *list Var*) {*struct vl*}: *list* (*Var* × *nat*):=
match vl return list (*Var* × *nat*) *with*
| *nil* ⇒ *nil*
| *v* :: *vl* ⇒ (*v*,*i*):: (*high_aux* (*S i*)(*vl*))
end.

Definition *high* (*v* : *Var*)(*vl* :*list Var*)(*m* : *nat*) : *list* (*Var* × *nat*) :=
high_aux 0 (*take* (*find_l m v*) *vl*).

- Έστω $\mathbf{x} \in Vars(\mathbf{P})$ με $\mathbf{x} : (\sigma_0, \ldots, \sigma_{k-1}) \rightarrow \iota$. Τότε η $Form(\mathbf{P}, \mathbf{x})$ είναι μία λίστα από μεταβλητές με διάσταση k (ή βαθμού k), οι οποίες ικανοποιούν τα παρακάτω :
  - Καμία μεταβλητή στη λίστα δεν εμφανίζεται στο πρόγραμμα $\mathbf{P}$.
  - Δεδομένου $\mathbf{y} \neq \mathbf{x}$,η $Form(\mathbf{P}, \mathbf{x})$ και η $Form(\mathbf{P}, \mathbf{y})$ δεν έχουν κοινά στοιχεία .

  Διαισθητικά, υπάρχουν φρέσκες μεταβλητές που θα προσαρτηθούν και στις δύο πλευρές των νέων ορισμών που εμφανίζονται κατά την εκτέλεση του αλγορίθμου.

  Parameter *string_of_nat* : *nat* → *string*.

  Fixpoint *name_of_var* (*f* : *Var*) :=
  match f with
  | *mkVar n* ⇒ *n*
  end.
  Definition *Form* (*P* : *Prog_IL*) (*f* : *Var*) : *list string* :=
  let arity := *arity_of_styp* (*pi f*) *in*
  let name := *name_of_var f in*
  map (*fun i* : *nat* ⇒ "_" ++ *name* ++ "_" ++ *string_of_nat i*)%*string*
  (*from_num_to_list arity*).

Βασισμένοι στους παραπάνω ορισμούς μπορούμε τώρα να παρουσιάσουμε τον αλγόριθμο μετασχηματισμού βήμα-βήμα.

7.2 Επεξεργασία των εκφράσεων

Ξεκινά με τον ορισμό της συνάρτησης που επεξεργάζεται τις εκφράσεις του πηγαίου προγράμματος. Πιο ειδικά, η εξάλειψη των παραμέτρων $(m-1)$-τάξης για κάθε κλήση συνάρτησης στο πρόγραμμα εξασφαλίζεται από τη συνάρτηση $\mathcal{E}_{\mathbf{P},m}$ που ορίζεται παρακάτω :

$$\frac{\mathbf{E} = \mathbf{f}}{\mathcal{E}_{\mathbf{P},m}(\mathbf{E}) = \mathbf{f}}$$

$$\frac{\mathbf{E} = \mathbf{c}\,(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})}{\mathcal{E}_{\mathbf{P},m}(\mathbf{E}) = \mathbf{c}\,(\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_0), \ldots, \mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{n-1}))}$$

$$\frac{\mathbf{E} = \mathbf{Q}\,(\mathbf{f})\ (\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,,\ order(\mathbf{f}) = m,\ low(\mathbf{f}, m) = [i_0, \ldots, i_{k-1}],\ label(\mathbf{P}, \mathbf{f}, \mathbf{E}) = i}{\mathcal{E}_{\mathbf{P},m}(\mathbf{E}) = \mathbf{Q} \cdot \mathtt{call}_i^{m-1}\,(\mathbf{f})\ (\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{i_0}), \ldots, \mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{i_{k-1}}))}$$

$$\frac{\mathbf{E} = \mathbf{Q}\,(\mathbf{f})\ (\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})\,,\ order(\mathbf{f}) < m}{\mathcal{E}_{\mathbf{P},m}(\mathbf{E}) = \mathbf{Q}\,(\mathbf{f})\ (\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_0), \ldots, \mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{n-1}))}$$

Ο πρώτος κανόνας είναι για την περίπτωση των μεταβλητών που εμφανίζονται κατά τη διάρκεια του μετασχηματισμού. Σε αυτή την περίπτωση, η έκφραση δεν επηρεάζεται από τον αλγόριθμο μετασχηματισμού. Ο δεύτερος κανόνας αφορά τις σταθερές και ο αλγόριθμος επιδρά στις παραμέτρους των σταθερών. Ο τρίτος κανόνας είναι για την περίπτωση της κλήσης σε συνάρτηση και η αντίστοιχη συνάρτηση είναι m-τάξης. Οι παράμετροι με τάξη $(m-1)$ αφαιρούνται και η κλήση παίρνει μπροστά τον κατάλληλο νοηματικό τελεστή. Ο τέταρτος κανόνας αφορά τις συναρτήσεις που έχουν τάξη μικρότερη από m. Σε αυτή την περίπτωση ο μετασχηματισμός προχωρά με τις πραγματικές παραμέτρους χωρίς να εξαλείφεται καμία.

Η συνάρτηση μετασχηματισμού για τις εκφράσεις *e* της γλώσσας L δεν έχει κάποια ιδιαιτερότητα, πέραν της περίπτωσης *a_expr_IL*, όπου χρησιμοποιείται η συνάρτηση *map* για να εφαρμόσει το μετασχηματισμό σε όλη τη λίστα των εκφράσεων *el* και στη συνέχεια με εφαρμογή της παραπάνω υπόθεσης αναφορικά με τη διάταξη των παραμέτρων της *f*, εξαφείλονται οι υψηλή τάξης παράμετροι με τάξη $m-1$ (συνάρτηση *drop*, η κλασική για λίστες). Επίσης γίνεται έλεγχος αν φθάσαμε στο τελικό στάδιο του μετασχηματισμού, οπότε παράγεται έκφραση *q_expr_IL*.

```
Fixpoint transform_7_2 (m : nat) (P : Prog_IL) (e : Expr_IL) {struct e} : Expr_IL :=
  match e with
    | f_expr_IL f ⇒
        e
    | c_expr_IL c el ⇒
        c_expr_IL c (map (transform_7_2 m P) el)
    | a_expr_IL q f el ⇒
        if le_lt_dec m (order (pi f)) then                    let i := label P f e in
              let q' := match q with
                              | nothing ⇒
                                    complex (call i (m-1))
                              | complex q_aux ⇒
                                    complex (Q_aux_list2 (Q_aux_to_Q_aux_list q_aux ++
                                                          call i (m-1) :: nil))
                        end in
              let el' := map (transform_7_2 m P) el in
              let el'' := drop (find_l m f) el' in
              let sl := drop (find_l m f) (type_args (pi f)) in
                 if eq_nil sl then
                          a_expr_IL q' f el''
                 else
                          q_expr_IL q' (f_expr_IL f)
        else
              a_expr_IL q f (map (transform_7_2 m P) el)
      | q_expr_IL q e' ⇒
          e
  end.
```

Επειδή η συνάρτηση *transform_7_2* αναφέρεται στο μετασχηματισμό του συντακτικού της γλώσσας και όχι στον μετασχηματισμό του τύπου κάθε έκφρασης κρίθηκε απαραίτητος ο ορισμός μίας συνάρτησης, η οποία να δείχνει πώς μετασχηματίζεται ο τύπος της εκάστοτε έκφρασης παρακολουθώντας ταυτόχρονα το μετασχηματισμό της. Ο ορισμός της *transform_typing_expr_IL_aux* έγινε σε

$proofmode$ και χρησιμοποιεί λήμματα/βοηθητικές συναρτήσεις. Για λόγους οικονομίας χώρου κρίναμε σωστό να μην το παραθέσουμε εδώ. Ο ενδιαφερόμενος μπορεί να ανατρέξει στον κώδικα για περισσότερες λεπτομέρειες.

```
Definition transform_typing_expr_IL_aux :
  ∀ (m : nat) (P : Prog_IL),
    (∀ (e : Expr_IL) (s: STyp) (t : typExpr_IL pi e s),
       typExpr_IL (transform_pi m) (transform_7_2 m P e) (transform_type m s)) ×
    (∀ (el : list Expr_IL) (sl: list STyp) (tlist : typExprList_IL pi el sl),
       typExprList_IL (transform_pi m) (map (transform_7_2 m P) el) (map (transform_type m) sl)).

Definition transform_typing_expr_IL (m : nat)(P : Prog_IL) :=
 fst(transform_typing_expr_IL_aux m P).

Definition transform_typing_exprlist_IL (m : nat)(P : Prog_IL) :=
 snd(transform_typing_expr_IL_aux m P).
```

7.3 Εξαλείφοντας τις τυπικές παραμέτρους ανώτερης τάξης

Η συνάρτηση $\mathcal{D}_m$ χρησιμοποιείται για την επεξεργασία ορισμών του προγράμματος στο **P**, με την αφαίρεση των παραμέτρων $(m-1)$-τάξης. Προσέξτε ότι την ίδια στιγμή, το σώμα κάθε ορισμού τροποποιείται με τη βοήθεια της συνάρτησης $\mathcal{E}_{\mathbf{P},m}$. Ο ορισμός της $\mathcal{D}_m$ δίνεται παρακάτω :

$$\mathcal{D}_m(\mathbf{P}) = \bigcup_{\mathbf{F}\in\mathbf{P}} \mathcal{D}^*_m(\mathbf{F})$$

$$\frac{\mathbf{F} = \mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1})\ \doteq \mathbf{E},\ low(\mathbf{f}, m) = [i_0, \ldots, i_{k-1}]}{\mathcal{D}^*_m(\mathbf{F}) = \{\mathbf{f}\ (\mathbf{x}_{i_0}, \ldots, \mathbf{x}_{i_{k-1}}) \doteq \mathcal{E}_{\mathbf{P},m}(\mathbf{E})\}}$$

$$\frac{\mathbf{F} = \mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1})\ \doteq \mathtt{case}^m\ (\mathbf{E}_0, \ldots, \mathbf{E}_{r-1})\ ,\ low(\mathbf{f}, m) = [i_0, \ldots, i_{k-1}]}{\mathcal{D}^*_m(\mathbf{F}) = \{\mathbf{f}\ (\mathbf{x}_{i_0}, \ldots, \mathbf{x}_{i_{k-1}}) \doteq \mathtt{case}^m\ (\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_0), \ldots, \mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{r-1}))\ \}}$$

Προσέξτε ότι έχουμε προσθέσει δύο κανόνες, μία για την περίπτωση που το σώμα ξεκινά με τον τελεστή `case` και μία για την περίπτωση που το σώμα είναι μία συνηθισμένη έκφραση.

Η *transform_7_3_B_IL* αναφέρεται στο μετασχηματισμό του σώματος για κάποιο ορισμό *F* :*Def_IL*). Στην περίπτωση της απλής έκφρασης εφαρμόζει το μετασχηματισμό σε αυτή, ενώ στην περίπτωση του *case* εφαρμόζει το μετασχηματισμό σε όλες τις παραμέτρους του *case*.

```
Definition transform_7_3_B_IL (m : nat) (P : Prog_IL) (b : B_IL): B_IL :=
  match b with
    | E_IL e ⇒
        E_IL (transform_7_2 m P e)
    | case_IL _ el ⇒
        case_IL m (map (transform_7_2 m P) el)
  end.
```

Η συνάρτηση μετασχηματίζει τον ορισμό *F* :*Def_IL*, καταργώντας τις παραμέτρους υψηλής τάξης για τη συνάρτηση *v* του αριστερού μέλους.

```
Definition transform_7_3_aux (m : nat) (P : Prog_IL) (F : Def_IL): Def_IL :=
  match F with
  | def_IL v vl b ⇒
      def_IL v (drop (find_l m v) vl) (transform_7_3_B_IL m P b)
```

end.

Για το μετασχηματισμό του typing του σώματος *b* : *B_IL* χρειάστηκε μία συνάρτηση που να μετασχηματίζει το *pi*, το οποίο ορίστηκε σε προηγούμενη ενότητα λαμβάνοντας υπόψη την υπόθεση για τη διάταξη των παραμέτρων της εκάστοτε συνάρτησης *f* του προγράμματος.

```
Definition transform_pi (m : nat) : (Var → STyp) :=
  fun (f : Var) ⇒
    st_arrow (drop (find_l m f)(type_args (pi f))).
```

Επίσης χρειάστηκε το βοηθητικό θεώρημα *transform_st_iota_sanity*, με το οποίο επιβεβαιώνεται ότι ο ground τύπος μετασχηματιζόμενος οδηγεί σε ground τύπο.

```
Lemma transform_st_iota_sanity ( m : nat):
    transform_type m st_iota = st_iota.
Proof.
intros.
simpl;reflexivity.
Qed.
```

και η επιβεβαίωση του παρακάτω θεωρήματος :

```
Lemma transform_formals_sanity
  (f : Var)
  (xl : list Var)
  (m : nat)
  (e : pi f = st_arrow (map pi xl)):
  transform_pi m f = st_arrow (map (transform_pi m) (drop (find_l m f) xl)).
```

Η *map_of_repeat* κάνει το γνωστό *map* σε μία λίστα από ίδια αντικείμενα *repeat_list*. Ο κώδικας της, καθώς και μία σειρά άλλων βοηθητικών συναρτήσεων μπορεί να αναζητηθεί στο κείμενο του κώδικα της παρούσας διπλωματικής.

```
Definition transform_typing_B_IL (m : nat) (P : Prog_IL) (b : B_IL)
  (t : typB_IL pi b) : typB_IL (transform_pi m)(transform_7_3_B_IL m P b).
Proof.
intros.
induction t ; unfold transform_7_3_B_IL .
constructor ; rewrite ← transform_st_iota_sanity with (m := m).
eapply transform_typing_expr_IL ; assumption.
eapply case_typ_IL with (n := n).
rewrite ← transform_st_iota_sanity with (m := m).
rewrite ← map_of_repeat with (f := transform_type m).
eapply transform_typing_exprlist_IL ; assumption.
Qed.
```

Για την *transform_typing_def_IL* χρειαστήκαμε το λήμμα *NoDup_sublist*, το οποίο πολύ απλά υποδηλώνει ότι για λίστα με μη κοινά στοιχεία μεταξύ τους η υπολίστα που προκύπτει αν αφαιρέσουμε κάποια στοιχεία από την αρχή της έχει πάλι μη κοινά στοιχεία.

```
Lemma NoDup_sublist (l : nat)(A : Type)(xl :list A)(H : NoDup xl) :NoDup (drop l xl).
Proof.
    Admitted.

Definition transform_typing_def_IL (m : nat) (P : Prog_IL) (d : Def_IL)
  (t : typDef_IL pi d) : typDef_IL (transform_pi m)(transform_7_3_aux m P d).
Proof.
  intros.
```

inversion t ;unfold transform_7_3_aux.
constructor.
apply NoDup_sublist ; assumption.
apply transform_formals_sanity;assumption.
apply transform_typing_B_IL ; assumption.
Qed.

Το μετασχηματισμό του προγράμματος στο σύνολό του αναλαμβάνει η *transform_7_3*.

Definition *transform_7_3* (*m* : *nat*)(*P* : *Prog_IL*) : *Prog_IL* :=
map (*transform_7_3_aux m P*) *P*.

Ο ορισμός *transform_typing_Prog_IL_1* δίνει το πώς μετασχηματίζεται το typing του προγράμματος *Prog_IL* χωρίς να λαμβάνονται υπόψη οι ορισμοί που προστίθενται στο αρχικό πρόγραμμα (επόμενη παράγραφος).

Definition *transform_typing_Prog_IL_1* (*m* : *nat*) (*P* : *Prog_IL*)
(*t* : *typProg_IL pi P*) : *typProg_IL* (*transform_pi m*)(*transform_7_3 m P*).
Proof.
intros.
induction t;*unfold transform_7_3*;*simpl*;*constructor.*
apply transform_typing_def_IL;*assumption.*
replace (*map* (*transform_7_3_aux m* (*d* :: *p*)) *p*) *with* (*map* (*transform_7_3_aux m p*) *p*) *by admit.*
unfold transform_7_3 in IHt; *assumption.*
apply In_func; *assumption.*
Defined.

7.4 Νέοι Ορισμοί

Σε αυτό το στάδιο του αλγορίθμου μετασχηματισμού , ένας νέος ορισμός δημιουργείται για κάθε τυπική παράμετρο $(m-1)$-τάξης που υπήρχε στο πρόγραμμ $\mathbf{P}$. Πριν ορίσουμε τυπικά τη συνάρτηση $\mathcal{A}_m$ που κάνει ακριβώς αυτό , χρειάζεται να ορίσουμε ορισμένες βοηθητικές συναρτήσεις . Έστω $\mathbf{f}$ μία συνάρτηση m-τάξης ορισμένη στο πρόγραμμα $\mathbf{P}$, και έστω $\mathbf{C}_0, \ldots, \mathbf{C}_{r-1}$ οι διαφορετικές κλήσεις σε συναρτήσεις $\mathbf{f}$ του $\mathbf{P}$, με σειρά που υπακούει στα labels τους (δηλαδή $label(\mathbf{P}, \mathbf{f}, \mathbf{C}_0) = 0, \ldots, label(\mathbf{P}, \mathbf{f}, \mathbf{C}_{r-1}) = r-1$). Έστω $\mathbf{x}$ η j-οστή τυπική παράμετρος της $\mathbf{f}$.

- Έστω $\mathbf{C}_i = \mathbf{Q}\,(\mathbf{f})\;(\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})$, $0 \leq i \leq r-1$, μία από τις κλήσεις στην $\mathbf{f}$. Η έκφραση $inv(\mathbf{C}_i, \mathbf{x})$ ορίζεται ως εξής (θυμηθείτε ότι η $\mathbf{x}$ είναι η j-οστή τυπική παράμετρος της $\mathbf{f}$, $0 \leq j \leq n-1$):

$$inv(\mathbf{C}_i, \mathbf{x}) = \mathtt{actuals}_i^{m-1} \cdot \mathbf{Q}^{-1}\,(\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_j))$$

  Η παραπάνω έκφραση θα χρησιμοποιηθεί από το $\mathcal{A}_m$ για τη δημιουργία του σώματος νέων ορισμών .

```
Definition inv (P : Prog_IL) (m : nat) (j : nat) (Ci : Call): (Q × Expr_IL ) :=
match Ci with
  | call_def q v el ⇒
     (complex( Q_aux_list2
                          (match q with
                           | nothing ⇒
                                actuals (label P v (a_expr_IL q v el))(m-1) ::nil
                           | complex q_aux ⇒
                                actuals (label P v (a_expr_IL q v el))(m-1) ::
```

```
                                    Q_aux_list (match Q_invert (complex q_aux) with
                                                  | nothing ⇒ sorry
                                                  | complex q_aux' ⇒ q_aux'
                                                  end)
                                end )) ,
        transform_7_2 m P (nth j el sorry) ) end.
```

- Η συνάρτηση $params(\mathbf{P}, \mathbf{f}, \mathbf{x})$ ορίζεται ως εξής :

$$params(\mathbf{P}, \mathbf{f}, \mathbf{x}) = [inv(\mathbf{C}_0, \mathbf{x}), \ldots, inv(\mathbf{C}_{r-1}, \mathbf{x})]$$

  Με άλλα λόγια, η συνάρτηση $params$ συγκεντρώνει όλες τις inv εκφράσεις που αντιστοιχούν στην τυπική παράμετρο **x** της **f**.

```
Fixpoint map_calls (A : Type)(f : Call → A)(c : Calls_data_type){ struct c} : list A:=
  match c with
  | nil_calls ⇒ nil
  | cons_calls c calls_rest ⇒ f c :: map_calls A f calls_rest
  end.

Definition params (m : nat)(P : Prog_IL )(f : Var)(j : nat) := map_calls (inv P m j) (calls P f).
```

Η συνάρτηση $\mathcal{A}_m$ παράγει νέο ορισμό για κάθε τυπική παράμετρο $(m-1)$-τάξης στο πρόγραμμα **P**. Έαν η τυπική παράμετρος **x** της **f** είναι $(m-1)$-τάξης, τότε η συνάρτηση $\mathcal{A}_{\mathbf{f},\mathbf{x},m}$ επιστρέφει ένα σύνολο που περιέχει ένα ορισμό για αυτή την παράμετρο . Ο τυπικός ορισμός της $\mathcal{A}_m$ δίνεται παρακάτω :

$$\mathcal{A}_m(\mathbf{P}) = \bigcup_{\mathbf{f} \in func(\mathbf{P})} \bigcup_{\mathbf{x} \in high(\mathbf{f},m)} \mathcal{A}_{\mathbf{f},\mathbf{x},m}(\mathbf{P})$$

$$\frac{params(\mathbf{P}, \mathbf{f}, \mathbf{x}) = [\mathbf{A}_0, \ldots, \mathbf{A}_{r-1}],\ \ Form(\mathbf{P}, \mathbf{x}) = \vec{\mathbf{z}}}{\mathcal{A}_{\mathbf{f},\mathbf{x},m}(\mathbf{P}) = \{\mathbf{x}\,(\vec{\mathbf{z}})\ \doteq \mathtt{case}^{m-1}\,(\mathbf{A}_0\,(\vec{\mathbf{z}})\,, \ldots, \mathbf{A}_{r-1}\,(\vec{\mathbf{z}})\,)\,\}}$$

```
Definition transform_7_4_aux (P : Prog_IL )(f : Var)(m : nat)(x : Var)(j : nat) : Def_IL :=
  let el := map (fun (h : (Q × Expr_IL)% type) ⇒
                   match (snd h) with
                   | f_expr_IL v ⇒
                         a_expr_IL (fst h) v (map f_expr_IL (map mkVar (Form P x)) )
                   | _ ⇒
                         sorry                          end ) (params m P f j) in
    def_IL f (map mkVar (Form P x)) (case_IL m el)
  .

Definition transform_7_4 (m : nat)(P : Prog_IL) : Prog_IL :=
  concat (map (fun def ⇒
                match def with
                | def_IL v vl b ⇒
                    map (fun (pr :(Var × nat)% type) ⇒
                           transform_7_4_aux P v m (fst pr) (snd pr)) (high v vl m)
                end ) P).
```

7.5 Ολόκληρος ο μετασχηματισμός

Ο μετασχηματισμός του προγράμματος IL m-τάξης σε ένα $(m-1)$-τάξης, πραγματοποιείται από τη συνάρτηση $Step_m$ παρακάτω :

$$Step_m(\mathbf{P}) = \mathcal{D}_m(\mathbf{P}) \cup \mathcal{A}_m(\mathbf{P})$$

```
Definition step (P : Prog_IL) (m : nat) : Prog_IL :=
  transform_7_3 m P ++ transform_7_4 m P.
```

Το typing των νέων ορισμών που προστίθενται στο πρόγραμμα δίνεται από την παρακάτω συνάρτηση :

```
Definition transform_typing_Prog_IL_2 (m : nat) (P : Prog_IL) (t : typProg_IL pi P) :
                                        typProg_IL (transform_pi m)(transform_7_4 m P).
Proof.
intros.
induction t;[constructor|unfold transform_7_4;simpl;apply append_typing].
inversion d ; unfold transform_7_4_aux.
simpl.
admit.
unfold transform_7_4_aux.
unfold params.
Admitted.
```

Η σύνθεση των μετασχηματισμών (μετασχηματισμός αρχικού προγράμματος + παραγόμενοι ορισμοί) για το typing του προγράμματος υλοποιείται με χρήση του βοηθητικού ορισμού *append_typing* από τη *transform_typing_prog_IL_overall*.

```
Definition append_typing (pi : Var → STyp)(p1 : Prog_IL)(p2 : Prog_IL)(H1 : typProg_IL pi p1)
    (H2 : typProg_IL pi p2) : typProg_IL pi (p1 ++ p2).
Proof.
intros.
induction p1;[simpl; assumption|rewrite lemma33;inversion H1;constructor].
assumption.
apply IHp1 ; assumption.
admit.
Qed.

Definition transform_typing_prog_IL_overall (m : nat) (P : Prog_IL) (t : typProg_IL pi P) :
                                        typProg_IL (transform_pi m) (step P m).
Proof.
intros.
unfold step.
apply append_typing .
apply transform_typing_Prog_IL_1 ; assumption.
apply transform_typing_Prog_IL_2 ; assumption.
Qed.
```

Τέλος, δεδομένου του προγράμματος FL M-τάξης, ο συνολικός μετασχηματισμός του προγράμματος **P** σε ένα νοηματικό πρόγραμμα μηδενικής τάξης περιγράφεται από τη συνάρτησ $Trans_M$, που δίνεται παρακάτω :

$$Trans_M(\mathbf{P}) = Step_1(\cdots(Step_M(\mathbf{P}))\cdots)$$

```
Fixpoint trans (M : nat) (P : Prog_IL) {struct M} : Prog_IL :=
```

match M with
| 0 ⇒ *nil*
| *S n* ⇒ *trans n* (*step P* (*S n*))
end.

Αυτό ολοκληρώνει την τυπική περιγραφή του μετασχηματισμού. Μπορεί εύκολα να διαπιστωθεί ότι τα προγράμματα που προκύπτουν από τα ενδιάμεσα στάδια του μετασχηματισμού είναι συντακτικά έγκυρα IL προγράμματα, ενώ το τελικό πρόγραμμ είναι $NVIL$.

Κεφάλαιο 8

Απόδειξη Ορθότητας

Σε αυτή την ενότητα θα παρουσιάσουμε την απόδειξη ορθότητας του μετασχηματισμού. Πρώτα θα κάνουμε μία υπόθεση η οποία θα μας βοηθήσει να απλοποιήσουμε το συμβολισμό. Στη συνέχεια η απόδειξη της ορθότητας του μετασχηματισμού θα παρουσιαστεί λεπτομερώς .

Όπως συζητήθηκε σε προηγούμενες ενότητες, έστω $\mathbf{P}_M$, $M > 0$, το M-τάξης πηγαίο πρόγραμμα συναρτησιακού προγραμματισμού πάνω στο οποίο θα εφαρμοστεί ο αλγόριθμος μετασχηματισμού. Τα προγράμματα που προκύπτουν στα διαδοχικά στάδια του μετασχηματισμού είναι τα $\mathbf{P}_{M-1}, \ldots, \mathbf{P}_0$.

Για να απλοποιήσουμε το συμβολισμό, στα παρακάτω θεωρούμε ότι όλες οι συναρτήσεις ορισμένες στο $\mathbf{P}_M$, έχουν τυπικές παραμέτρους ανώτερης τάξης μπροστά στον κώδικα για παράδειγμα αν $\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B_f}$ είναι μια συνάρτηση ορισμένη στο $\mathbf{P}_M$, τότε θα είναι $order(\mathbf{x}_0) \geq order(\mathbf{x}_1) \geq \cdots \geq order(\mathbf{x}_{n-1})$. Αυτό μας βοηθάει να αποφύγουμε πολύπλοκους συμβολισμούς που θα προέκυπταν αν θεωρούσαμε ότι η σειρά των τυπικών παραμέτρων μπορεί να είναι οποιαδήποτε. Προσέξτε ότι αυτή η ιδιότητα διατηρείται από το μετασχηματισμό, το οποίο σημαίνει ότι αν ισχύει για το $\mathbf{P}_M$ θα ισχύει για όλα τα $\mathbf{P}_m$, $0 \leq m \leq M$.

Μπορεί εύκολα να ελεγχθεί ότι η παραπάνω υπόθεση δεν επιδρά στη γενικότητα της απόδειξης. Γενικεύοντας την υπόθεση δεν επιδρούμε στη λογική της απόδειξης, απλά εισάγει ένα επίπεδο πολυπλοκότητας στο συμβολισμό.

Ορισμός 8.1: Έστω $w \in W$. Η συνάρτηση $\Downarrow$: $(W, ISeq) \rightarrow W$ ορίζεται αναδρομικά ως εξής :

$$\begin{array}{lcl}
(w \Downarrow \epsilon) & = & w \\
(w \Downarrow \texttt{call}_i^m) & = & w[m/(i : w_m)] \\
(w \Downarrow \texttt{actuals}_i^m) & = & \begin{cases} w[m/tail(w_m)] & \text{if } head(w_m) = i \\ undefined & \text{otherwise} \end{cases} \\
(w \Downarrow (\mathbf{q}_0 \cdot \mathbf{q}_1 \cdot \ldots \cdot \mathbf{q}_{n-1})) & = & (w \Downarrow \mathbf{q}_0) \Downarrow (\mathbf{q}_1 \cdot \ldots \cdot \mathbf{q}_{n-1})
\end{array}$$

Το παρακάτω λήμμα μπορεί εύκολα να αποδειχθεί :

Λήμμα 8.1: Έστω $w \in W$, $\mathbf{Q} \in ISeq$ και $a \in [\![\sigma]\!]^*$. Τότε :

$$\begin{array}{rcl}
(w \Downarrow \mathbf{Q}) \Downarrow \mathbf{Q^{-1}} & = & w \\
Q(a)w & = & a(w \Downarrow \mathbf{Q})
\end{array}$$

όποτε το $(w \Downarrow \mathbf{Q})$ ορίζεται.

Θεωρήστε τώρα το πρόγραμμα $\mathbf{P}_m$ ($0 < m \leq M$). Το παρακάτω θεώρημα εισάγει τη σχέση μεταξύ της σημασιολογίας των συναρτήσεν στο $\mathbf{P}_m$ και της σημασιολογίας των συναρτήσεων στο $\mathbf{P}_{m-1}$.

Θεώρημα 8.1: Έστω u και $\widehat{u}$ τα ελάχιστα περιβάλλοντα που ικανοποιούν κάτω από τη συγχρονική προσέγγιση τους ορισμούς στο $\mathbf{P}_m$ και στο $\mathbf{P}_{m-1}$ αντίστοιχα . Τότε :

- Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B_f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ και υπάρχει $0 \leq l \leq n-1$ τ.ώ. $order(\sigma_0) = (m-1), \ldots, order(\sigma_{l-1}) = (m-1)$ και $order(\sigma_l) <$

$(m-1),\ldots,order(\sigma_{n-1})<(m-1)$, τότε για κάθε κλήση $\mathbf{E}=\mathbf{Q}\,(\mathbf{f})\;(\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$ στην $\mathbf{f}$ μέσα στο $\mathbf{P}_m$, για όλα τα $d_l\in[\![\sigma_l]\!],\ldots,d_{n-1}\in[\![\sigma_{n-1}]\!]$ και για όλα τα $w\in W$,

$$Q\cdot call_i^{m-1}(\widehat{u}(\mathbf{f}))(w)(d_l,\ldots,d_{n-1})\sqsubseteq$$
$$Q(u(\mathbf{f}))(w)([\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_0)]\!]^*(\widehat{u})(w),\ldots,[\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{l-1})]\!]^*(\widehat{u})(w),d_l,\ldots,d_{n-1})$$

όπου $i=label(\mathbf{P},\mathbf{f},\mathbf{E})$.

- Για κάθε ορισμό $(\mathbf{f}\;(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1})\doteq\mathbf{B}_\mathbf{f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0:\sigma_0,\ldots,\mathbf{x}_{n-1}:\sigma_{n-1}$ και επίσης $order(\sigma_0)<(m-1),\ldots,order(\sigma_{n-1})<(m-1)$, τότε για όλα τα $d_0\in[\![\sigma_0]\!],\ldots,d_{n-1}\in[\![\sigma_{n-1}]\!]$ και για όλα τα $w\in W$,

$$\widehat{u}(\mathbf{f})(w)(d_0,\ldots,d_{n-1})\sqsubseteq u(\mathbf{f})(w)(d_0,\ldots,d_{n-1})$$

Απόδειξη : Η απόδειξη γίνεται με μία μακροσκελή αλλά εύκολη στη σύλληψη επαγωγή πάνω στα στάδια της κατασκευής του $\widehat{u}$ (και *όχι* του u). Πιο ειδικά , αρκεί να δείξουμε ότι οι παραπάνω θέσεις ισχύουν για όλες τις προσεγγίσεις $\widehat{u}_k, k\in N$, του περιβάλλοντος $\widehat{u}$. Με άλλα λόγια , αρκεί να δείξουμε τις παρακάτω δύο προτάσεις :

- Για κάθε ορισμό $(\mathbf{f}\;(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1})\doteq\mathbf{B}_\mathbf{f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0:\sigma_0,\ldots,\mathbf{x}_{n-1}:\sigma_{n-1}$ και υπάρχει $0\le l\le n-1$ τ.ώ. $order(\sigma_0)=(m-1),\ldots,order(\sigma_{l-1})=(m-1)$ και $order(\sigma_l)<(m-1),\ldots,order(\sigma_{n-1})<(m-1)$, τότε για κάθε κλήση $\mathbf{E}=\mathbf{Q}\,(\mathbf{f})\;(\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$ στην $\mathbf{f}$ μέσα στο $\mathbf{P}_m$, για όλα τα $d_l\in[\![\sigma_l]\!],\ldots,d_{n-1}\in[\![\sigma_{n-1}]\!]$ και για όλα τα $w\in W$,

$$Q\cdot call_i^{m-1}(\widehat{u}_k(\mathbf{f}))(w)(d_l,\ldots,d_{n-1})\sqsubseteq$$
$$Q(u(\mathbf{f}))(w)([\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_0)]\!]^*(\widehat{u}_k)(w),\ldots,[\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{l-1})]\!]^*(\widehat{u}_k)(w),d_l,\ldots,d_{n-1})$$

όπου $i=label(\mathbf{P},\mathbf{f},\mathbf{E})$.

- Για κάθε ορισμό $(\mathbf{f}\;(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1})\doteq\mathbf{B}_\mathbf{f})$ μέσα στο $\mathbf{P}_m$, αν $\mathbf{x}_0:\sigma_0,\ldots,\mathbf{x}_{n-1}:\sigma_{n-1}$ και $order(\sigma_0)<(m-1),\ldots,order(\sigma_{n-1})<(m-1)$, τότε για όλα τα $d_0\in[\![\sigma_0]\!],\ldots,d_{n-1}\in[\![\sigma_{n-1}]\!]$ και για όλα τα $w\in W$,

$$\widehat{u}_k(\mathbf{f})(w)(d_0,\ldots,d_{n-1})\sqsubseteq u(\mathbf{f})(w)(d_0,\ldots,d_{n-1})$$

Παρουσιάζουμε τα παραπάνω με επαγωγή στο k. Για $k=0$, έχουμε το $\widehat{u}_0$. Τα παραπάνω ισχύουν γιατί το αριστερό μέρος κάθε πρότασης είναι ίσο με την ελάχιστη τιμή (bottom value). Υποθέστε ότι η υπόθεση ισχύει για $k\ge 0$. Αποδεικνύουμε τη πρόταση για $k+1$. Δηλαδή δείχνουμε ότι :

- Για κάθε ορισμό $(\mathbf{f}\;(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1})\doteq\mathbf{B}_\mathbf{f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0:\sigma_0,\ldots,\mathbf{x}_{n-1}:\sigma_{n-1}$ και υπάρχουν $0\le l\le n-1$ τ.ώ $order(\sigma_0)=(m-1),\ldots,order(\sigma_{l-1})=(m-1)$ και $order(\sigma_l)<(m-1),\ldots,order(\sigma_{n-1})<(m-1)$, τότε για κάθε κλήση $\mathbf{E}=\mathbf{Q}\,(\mathbf{f})\;(\mathbf{E}_0,\ldots,\mathbf{E}_{n-1})$ στην $\mathbf{f}$ με $\mathbf{P}_m$, για όλα τα $d_l\in[\![\sigma_l]\!],\ldots,d_{n-1}\in[\![\sigma_{n-1}]\!]$ και για όλα τα $w\in W$,

$$Q\cdot call_i^{m-1}(\underline{\widehat{u}_{k+1}}(\mathbf{f}))(w)(d_l,\ldots,d_{n-1})\sqsubseteq$$
$$Q(u(\mathbf{f}))(w)([\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_0)]\!]^*(\underline{\widehat{u}_{k+1}})(w),\ldots,[\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{l-1})]\!]^*(\underline{\widehat{u}_{k+1}})(w),d_l,\ldots,d_{n-1})$$

όπου $i=label(\mathbf{P},\mathbf{f},\mathbf{E})$.

- Για κάθε ορισμό $(\mathbf{f}\;(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1})\doteq\mathbf{B}_\mathbf{f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0:\sigma_0,\ldots,\mathbf{x}_{n-1}:\sigma_{n-1}$ και επίσης $order(\sigma_0)<(m-1),\ldots,order(\sigma_{n-1})<(m-1)$, τότε για όλα τα $d_0\in[\![\sigma_0]\!],\ldots,d_{n-1}\in[\![\sigma_{n-1}]\!]$ και για όλα $w\in W$,

$$\underline{\widehat{u}_{k+1}}(\mathbf{f})(w)(d_0,\ldots,d_{n-1})\sqsubseteq u(\mathbf{f})(w)(d_0,\ldots,d_{n-1})$$

Με χρήση της σημασιολογίας των $call_i^{m-1}$ και Q, οι δύο παραπάνω προτάσεις μπορούν να γραφούν ως εξής :

- Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B_f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ στο $0 \leq l \leq n-1$ τ.ώ. $order(\sigma_0) = (m-1), \ldots, order(\sigma_{l-1}) = (m-1)$ και $order(\sigma_l) < (m-1), \ldots, order(\sigma_{n-1}) < (m-1)$, τότε για κάθε κλήση $\mathbf{E} = \mathbf{Q}\ (\mathbf{f})\ (\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})$ στην $\mathbf{f}$ στο $\mathbf{P}_m$, για όλα τα $d_l \in [\![\sigma_l]\!], \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]$ και για όλα τα $w \in W$,

$$\underline{\widehat{u}_{k+1}(\mathbf{f})(w \Downarrow (\mathbf{Q} \cdot \mathtt{call}_i^{m-1}))(d_l, \ldots, d_{n-1}) \sqsubseteq}$$
$$\underline{u(\mathbf{f})(w \Downarrow \mathbf{Q})([\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_0)]\!]^*(\widehat{u}_{k+1}))(w), \ldots, [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{l-1})]\!]^*(\widehat{u}_{k+1})(w), d_l, \ldots, d_{n-1})}$$

  όπου $i = label(\mathbf{P}, \mathbf{f}, \mathbf{E})$.

- Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B_f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ και επίσης $order(\sigma_0) < (m-1), \ldots, order(\sigma_{n-1}) < (m-1)$, τότε για όλα τα $d_0 \in [\![\sigma_0]\!], \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]$ και για όλα τα $w \in W$,

$$\widehat{u}_{k+1}(\mathbf{f})(w)(d_0, \ldots, d_{n-1}) \sqsubseteq u(\mathbf{f})(w)(d_0, \ldots, d_{n-1})$$

Θυμηθείτε ότι τώρα $\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B_f}$ στο $\mathbf{P}_m$ και επίσης $\mathbf{f}\ (\mathbf{x}_l, \ldots, \mathbf{x}_{n-1}) \doteq \mathcal{E}_{\mathbf{P},m}(\mathbf{B_f})$ στο $\mathbf{P}_{m-1}$. Η ιδέα είναι να πάρουμε ισοδύναμες προτάσεις που αφορούν το σώμα της συνάρτησης $\mathbf{f}$. Πιο ειδικά χρησιμοποιούμε τον ορισμό 6.10 για να αντικαταστήσουμε τα δεξιά μέρη της πρότασης και το θεώρημα 6.1 για να αντικαταστήσουμε το αριστερά μέρη των προτάσεων (προσέξτε ότι στις προτάσεις που παίρνουμε παρακάτω , για λόγους απλότητας χρησιμοποιούμε το συμβολισμό της συνάρτησης perturbation $\oplus$ αντί για το συμβολισμό $[\cdots]$ ο οποίος χρησιμοποιείται για το 6.10 και το θεώρημα 6.1).

Συνεπώς αρκεί να δείξουμε τα παρακάτω :

- Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B_f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ και υπάρχει $0 \leq l \leq n-1$ τ.ώ $order(\sigma_0) = (m-1), \ldots, order(\sigma_{l-1}) = (m-1)$ και $order(\sigma_l) < (m-1), \ldots, order(\sigma_{n-1}) < (m-1)$, τότε για κάθε κλήση $\mathbf{E} = \mathbf{Q}\ (\mathbf{f})\ (\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})$ στη $\mathbf{f}$ $\mathbf{P}_m$, για όλα τα $d_l \in [\![\sigma_l]\!], \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]$ και για όλα τα $w \in W$,

$$\underline{[\![\mathcal{E}_{\mathbf{P},m}(\mathbf{B_f})]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow (\mathbf{Q} \cdot \mathtt{call}_i^{m-1}))} \sqsubseteq \underline{[\![\mathbf{B_f}]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q})}$$

  όπου $\underline{\sigma(\mathbf{x}_j) = d_j^\infty}, \underline{l \leq j \leq n-1}$, και $\underline{\widehat{\rho}_{k+1}(\mathbf{x}_j) = ([\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_j)]\!]^*(\widehat{u}_{k+1})(w))^\infty}, \underline{0 \leq j \leq l-1}$.

- Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B_f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ και επίσης $order(\sigma_0) < (m-1), \ldots, order(\sigma_{n-1}) < (m-1)$, τότε για όλα τα $d_0 \in [\![\sigma_0]\!], \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]$ και για όλα τα $w \in W$,

$$\underline{[\![\mathcal{E}_{\mathbf{P},m}(\mathbf{B_f})]\!]^*(\widehat{u}_k \oplus \sigma)(w)} \sqsubseteq \underline{[\![\mathbf{B_f}]\!]^*(u \oplus \sigma)(w)}$$

  $\underline{\text{όπου } \sigma(\mathbf{x}_j) = d_j^\infty, 0 \leq j \leq n-1.}$

Στα παρακάτω , δίνουμε την απόδειξη για την πρώτη από τις δύο προτάσεις . Η απόδειξη για τη δεύτερη πρόταση είναι απλούστερη και μπορεί να δοθεί με παρόμοιο τρόπο . Προσέξτε ότι η απόδειξη για κάθε μία από τις παραπάνω προτάσεις , χρησιμοποιεί σε κάποιο σημείο την επαγωγική υπόθεση της άλλης πρότασης .

Για να αποδείξουμε την πρώτη από τις προτάσεις , θεωρούμε κάθε συνάρτηση $\mathbf{f}$ στο πρόγραμμα που ικανοποιεί τους περιορισμούς που θέτει η πρόταση . Προχωρούμε διακρίνοντας δύο περιπτώσεις , ανάλογα με το αν ο ορισμός της $\mathbf{f}$ ξεκινά με τον τελεστή `case` ή όχι . Θα δείξουμε την απόδειξη μόνο για τη δεύτερη περίπτωση (η απόδειξη για την πρώτη είναι παρόμοια).

Η απόδειξη μπορεί να γίνει με δομική επαααγωγή πάνω στο σώμα της συνάρτησης , δηλαδή για κάθε υποέκφραση $\mathbf{S}$ του $\mathbf{B_f}$:

$$[\![\mathcal{E}_{\mathbf{P},m}(\mathbf{S})]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow (\mathbf{Q} \cdot \mathtt{call}_i^{m-1})) \sqsubseteq [\![\mathbf{S}]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q})$$

Στα παρακάτω , για λόγους απλότητας ταυτίζουμε την ακολουθία $\mathbf{Q} \cdot \mathtt{call}_i^{m-1}$ με το $\widehat{\mathbf{Q}}$.

Βάση Δομικής Επαγωγής:
Περίπτωση 1: Το $\mathbf{S}$ είναι ίσο με μία μεταβλητή $\mathbf{x}_j \in \{\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}\}$ και η $\mathbf{x}_j$ είναι $(m-1)$-τάξης. Τότε ένας ορισμός της μορφής :

$$\mathbf{x}_j\,(\vec{\mathbf{z}}) \doteq \mathtt{case}^{m-1}\,(\mathbf{A}_0\,(\vec{\mathbf{z}})\,, \ldots, \mathbf{A}_{r-1}\,(\vec{\mathbf{z}})\,)$$

δημιουργήθηκε στο $\mathbf{P}_{m-1}$, όπου τα $\mathbf{A}_0, \ldots, \mathbf{A}_{r-1}$ βγαίνουν από τη συνάρτηση $params(\mathbf{P}, \mathbf{f}, \mathbf{x}_j)$. Πρώτα παρουσιάζουμε τα ακόλουθα :

$$\widehat{u}_k(\mathbf{x}_j)(w \Downarrow \widehat{\mathbf{Q}}) \sqsubseteq \widehat{\rho}_{k+1}(\mathbf{x}_j)(w \Downarrow \mathbf{Q}) \tag{8.1}$$

Θεωρήστε ότι η $\vec{\mathbf{z}}$ είναι της μορφής $(\mathbf{z}_0, \ldots, \mathbf{z}_{k-1})$ και $\mathbf{z}_0 : \sigma'_0, \ldots, \mathbf{z}_{k-1} : \sigma'_{k-1}$. Έστω τότε $\vec{e} = (e_0, \ldots, e_{k-1})$ μία k-τούπλα τ.ώ. $e_0 \in [\![\sigma'_0]\!], \ldots, e_{k-1} \in [\![\sigma'_{k-1}]\!]$. Επίσης , έστω $\phi(\mathbf{z}_i) = e_i^{\infty}, 0 \leq i \leq k-1$. Η απόδειξη του (1) έχει ως εξής :

$$\begin{array}{ll}
 & \widehat{u}_k(\mathbf{x}_j)(w \Downarrow \widehat{\mathbf{Q}})(\vec{e}) = \\
\sqsubseteq & [\![\mathtt{case}^{m-1}\,(\mathbf{A}_0\,(\vec{\mathbf{z}})\,, \ldots, \mathbf{A}_{r-1}\,(\vec{\mathbf{z}})\,)\,]\!]^*(\widehat{u}_k \oplus \phi)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Ορισμός του } \mathbf{x}_j \text{ και λήμμα 6.1)} \\
= & [\![\mathbf{A}_i\,(\vec{\mathbf{z}})\,]\!]^*(\widehat{u}_k \oplus \phi)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Σημασιολογία του } \mathtt{case}^{m-1}\text{)} \\
= & [\![\mathtt{actuals}_i^{m-1} \cdot \mathbf{Q}^{-1}\,(\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_j))\,(\vec{\mathbf{z}})\,]\!]^*(\widehat{u}_k \oplus \phi)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Ορισμός του } \mathbf{A}_i \text{ βάσει του ορισμού της συνάρτησης } params(\mathbf{P}, \mathbf{f}, \mathbf{x}_j)\text{)} \\
= & [\![\mathbf{Q}^{-1}\,(\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_j))\,]\!]^*(\widehat{u}_k \oplus \phi)(w \Downarrow \mathbf{Q})(\vec{e}) \\
 & \text{(Σημασιολογία του } \mathtt{actuals}_i^{m-1} \text{ και εφαρμογή)} \\
= & [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_j)]\!]^*(\widehat{u}_k \oplus \phi)(w)(\vec{e}) \\
 & \text{(Λήμμα 8.1)} \\
= & [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_j)]\!]^*(\widehat{u}_k)(w)(\vec{e}) \\
 & \text{(Το } \mathbf{E}_j \text{ δεν περιέχει κανένα } \mathbf{z}_i\text{)} \\
\sqsubseteq & [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_j)]\!]^*(\widehat{u}_{k+1})(w)(\vec{e}) \\
 & \text{(Μονοτονία της } [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_j)]\!]^*\text{)} \\
= & \widehat{\rho}_{k+1}(\mathbf{x}_j)(w \Downarrow \mathbf{Q})(\vec{e}) \\
 & \text{(Ορισμός του } \widehat{\rho}_{k+1}\text{)}
\end{array}$$

Το παραπάνω αποτέλεσμα χρησιμοποιείται για την απόδειξη που δίνεται παρακάτω : Το αριστερό μέρος της πρότασης που θέλουμε να δείξουμε μπορεί να γραφεί :

$$\begin{array}{ll}
 & [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{S})]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}) = \\
= & [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{x}_j)]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Επειδή } \mathbf{S} = \mathbf{x}_j\text{)} \\
= & [\![\mathbf{x}_j]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Ορισμός του } \mathcal{E}_{\mathbf{P},m}\text{)} \\
= & \widehat{u}_k(\mathbf{x}_j)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Η μεταβλητή } \mathbf{x}_j \text{ είναι } (m-1)\text{-order)} \\
\sqsubseteq & \widehat{\rho}_{k+1}(\mathbf{x}_j)(w \Downarrow \mathbf{Q}) \\
 & \text{(Λόγω του ορισμού (1) παρακάτω)} \\
= & [\![\mathbf{x}_j]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q}) \\
 & \text{(Η μεταβλητή } \mathbf{x}_j \text{ παίρνει τιμή από } \widehat{\rho}_{k+1}\text{)} \\
= & [\![\mathbf{S}]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q}) \\
 & \text{(Λόγω } \mathbf{S} = \mathbf{x}_j\text{)}
\end{array}$$

Περίπτωση 2:Το $\mathbf{S}$ είναι ίσο με μία μεταβλητή $\mathbf{x}_j \in \{\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}\}$, και $\mathbf{x}_j$ έχει τάξη μικρότερη από $(m-1)$. Πρέπει να θυμηθούμε ότι το d^∞ είναι ανεξάρτητη του world και συνεπώς η τιμή της δεν ποικίλλει από περιβάλλον σε περιβάλλον . Το αριστερό μέρος της πρότασης που θέλουμε να δείξουμε μπορεί να γραφεί ως εξής :

$$
\begin{array}{ll}
 & [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{S})]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}) = \\
= & [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{x}_j)]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Επειδή } \mathbf{S} = \mathbf{x}_j\text{)} \\
= & [\![\mathbf{x}_j]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Ορισμός του } \mathcal{E}_{\mathbf{P},m}\text{)} \\
= & \sigma(\mathbf{x}_j)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Επειδή η } \mathbf{x}_j \text{ έχει τάξη μικρότερη του } (m-1)\text{.)} \\
= & \sigma(\mathbf{x}_j)(w \Downarrow \mathbf{Q}) \\
 & \text{(Επειδή } \sigma(\mathbf{x}_j) = d_j^\infty\text{)} \\
= & [\![\mathbf{x}_j]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q}) \\
 & \text{(Η μεταβλητή } \mathbf{x}_j \text{ παίρνει τιμή από το } \sigma\text{)} \\
= & [\![\mathbf{S}]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q}) \\
 & \text{(Επειδή } \mathbf{S} = \mathbf{x}_j\text{)}
\end{array}
$$

Περίπτωση 3: Το $\mathbf{S}$ είναι ίσο με μία σταθερά μηδενικής τάξης **c**. Η απόδειξη σε αυτή την περίπτωση είναι εύκολη επειδή η σημασιολογία της **c** είναι ανεξάρτητη του περιβάλλοντος .

Περίπτωση 4: Το $\mathbf{S}$ είναι ίσο με $\mathbf{h}$, όπου το **h** δεν είναι μία τυπική παράμετρος της **f**. Σε αυτή την περίπτωση , η τάξη του **h** είναι αυστηρά μικρότερη από m (επειδή οι συναρτήσεις m-τάξης έχουν μόνο πλήρεις εφαρμογές (εννοεί με ground τύπο) στο $\mathbf{P}_m$). Θυμηθείτε τώρα ότι η εξωτερική επαγωγική υπόθεση για συναρτήσεις μικρότερης τάξης από m, ξεκαθαρίζει ότι $\widehat{u}_k(\mathbf{h}) \sqsubseteq u(\mathbf{h})$. Χρησιμοποιώντας αυτό και το γεγονός ότι το $u(\mathbf{h})$ δεν εξαρτάται από την m-οστή διάσταση , παίρνουμε το επιθυμητό αποτέλεσμα .

Δομικό Βήμα Επαγωγής.
Περίπτωση 1: $\mathbf{S} = \mathbf{x}_j\,(\mathbf{S}_0, \ldots, \mathbf{S}_{r-1})$ όπου $\mathbf{x}_j \in \{\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}\}$, and $\mathbf{x}_j$ είναι $(m-1)$-τάξης. Η απόδειξη χρησιμοποιεί το παρακάτω γεγονός το οποίο χρησιμοποιήθηκε επίσης στην επαγωγική υπόθεση :

$$\widehat{u}_k(\mathbf{x}_j)(w \Downarrow \widehat{\mathbf{Q}}) \sqsubseteq \widehat{\rho}_{k+1}(\mathbf{x}_j)(w \Downarrow \mathbf{Q}) \tag{8.2}$$

Στα παρακάτω, προσέξτε ότι κανένα από τα ορίσματα της $\mathbf{x}_j$ δεν καταργείται κατά τη διάρκεια του μετασχηματισμού γιατί όλες έχουν τάξη μικρότερη από $(m-1)$. Η απόδειξη δίνεται παρακάτω :

$$
\begin{array}{ll}
 & [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{S})]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}) = \\
= & [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{x}_j\,(\mathbf{S}_0, \ldots, \mathbf{S}_{r-1})\,)]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Επειδή } \mathbf{S} = \mathbf{x}_j\,(\mathbf{S}_0, \ldots, \mathbf{S}_{r-1})\,\text{)} \\
= & [\![\mathbf{x}_j\,(\mathcal{E}_{\mathbf{P},m}(\mathbf{S}_0), \ldots, \mathcal{E}_{\mathbf{P},m}(\mathbf{S}_{r-1}))\,]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}) \\
 & \text{(Ορισμός του } \mathcal{E}_{\mathbf{P},m}\text{)} \\
= & \widehat{u}_k(\mathbf{x}_j)(w \Downarrow \widehat{\mathbf{Q}})([\![\mathcal{E}_{\mathbf{P},m}(\mathbf{S}_0)]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}), \ldots, \\
 & \quad [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{S}_{r-1})]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}})) \\
 & \text{(Η μεταβλητή } \mathbf{x}_j \text{ είναι } (m-1)\text{-τάξης)} \\
\sqsubseteq & \widehat{\rho}_{k+1}(\mathbf{x}_j)(w \Downarrow \mathbf{Q})([\![\mathcal{E}_{\mathbf{P},m}(\mathbf{S}_0)]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}}), \ldots, \\
 & \quad [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{S}_{r-1})]\!]^*(\widehat{u}_k \oplus \sigma)(w \Downarrow \widehat{\mathbf{Q}})) \\
 & \text{(Επειδή } \widehat{u}_k(\mathbf{x}_j)(w \Downarrow \widehat{\mathbf{Q}}) \sqsubseteq \widehat{\rho}_{k+1}(\mathbf{x}_j)(w \Downarrow \mathbf{Q})\text{)}
\end{array}
$$

$$
\begin{aligned}
\sqsubseteq\ & \widehat{\rho}_{k+1}(\mathbf{x}_j)(w \Downarrow \mathbf{Q})([\![\mathbf{S}_0]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q}), \ldots, \\
& \quad [\![\mathbf{S}_{r-1}]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q})) \\
& \text{(Χρησιμοποιώντας δομική επαγωγική υπόθεση και μονοτονία)} \\
=\ & [\![\mathbf{x}_j\ (\mathbf{S}_0, \ldots, \mathbf{S}_{r-1})\ ]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q}) \\
& \text{(Η μεταβλητή } \mathbf{x}_j \text{ παίρνει τιμή από } \widehat{\rho}_{k+1}) \\
=\ & [\![\mathbf{S}]\!]^*(u \oplus \sigma \oplus \widehat{\rho}_{k+1})(w \Downarrow \mathbf{Q}) \\
& \text{(Επειδή } \mathbf{S} = \mathbf{x}_j\ (\mathbf{S}_0, \ldots, \mathbf{S}_{r-1})\ \text{)}
\end{aligned}
$$

Περίπτωση 2: $\mathbf{S} = \mathbf{x}_j\ (\mathbf{S}_0, \ldots, \mathbf{S}_{r-1})$ όπου $\mathbf{x}_j \in \{\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}\}$, και έχει τάξη $\mathbf{x}_j$ μικρότερη από $(m-1)$. Τότε , η $\mathbf{x}_j$ παίρνει τιμή από τη σ και στις δύο πλευρές της πρότασης που θέλουμε να δείξουμε . Προσέξτε επίσης ότι $\sigma(\mathbf{x}_j) = d_j^{\infty}$ και η τιμή της είναι ανεξάρτητη από το περιβάλλον στο οποίο βρίσκεται . Η απόδειξη είναι ίδια σε δομή με αυτή για την παραπάνω περίπτωση .

Περίπτωση 3: $\mathbf{S} = \mathbf{c}\ (\mathbf{S}_0, \ldots, \mathbf{S}_{r-1})$. Η απόδειξη γι' αυτή την περίπτωση είναι απλή και χρησιμοποιεί το γεγονός ότι $\mathcal{C}^*(\mathbf{c})$ έχει σταθερή σημασιολογία . .

Περίπτωση 4: $\mathbf{S} = \mathbf{\Phi}\ (\mathbf{g})\ (\mathbf{S}_0, \ldots, \mathbf{S}_{r-1})$ όπου η $\mathbf{\Phi}$ είναι μία ακολουθία από νοηματικούς τελεστές και η $\mathbf{g}$ είναι η συνάρτηση ορισμένη στο $\mathbf{P}_m$. Η απόδειξη είναι παρόμοια με προηγουμένως (πρέπει να θεωρήσουμε δύο περιπτώσεις : μία για την $\mathbf{g}$ ούσα m-τάξης και μία για μικρότερης τάξης).

Αυτό συμπληρώνει την απόδειξη του θεωρήματος . ∎

Θεώρημα 8.2: Έστω u και $\widehat{u}$ τα ελάχιστα περιβάλλοντα που ικανοποιούν υπό τη συγχρονική προσέγγιση τους ορισμούς στο $\mathbf{P}_m$ και $\mathbf{P}_{m-1}$ αντίστοιχα . Τότε :

- Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1})\ \doteq \mathbf{B}_{\mathbf{f}})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ και επειδή υπάρχει $0 \leq l \leq n-1$ τ.ώ. $order(\sigma_0) = (m-1), \ldots, order(\sigma_{l-1}) = (m-1)$ και $order(\sigma_l) < (m-1), \ldots, order(\sigma_{n-1}) < (m-1)$, τότε για όλες τις κλήσεις $\mathbf{E} = \mathbf{Q}\ (\mathbf{f})\ (\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})$ στην $\mathbf{f}$ μέσα στο $\mathbf{P}_m$, για όλα τα $d_l \in [\![\sigma_l]\!], \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]$ και για όλα τα $w \in W$,

  $$
  \begin{aligned}
  & Q \cdot call_i^{m-1}(\widehat{u}(\mathbf{f}))(w)(d_l, \ldots, d_{n-1}) \sqsupseteq \\
  & Q(u(\mathbf{f}))(w)([\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_0)]\!]^*(\widehat{u})(w), \ldots, [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{l-1})]\!]^*(\widehat{u})(w), d_l, \ldots, d_{n-1})
  \end{aligned}
  $$

  where $i = label(\mathbf{P}, \mathbf{f}, \mathbf{E})$.

- Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1})\ \doteq \mathbf{B}_{\mathbf{f}})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ και $order(\sigma_0) < (m-1), \ldots, order(\sigma_{n-1}) < (m-1)$, τότε για όλα τα $d_0 \in [\![\sigma_0]\!], \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]$ και για όλα τα $w \in W$,

  $$\widehat{u}(\mathbf{f})(w)(d_0, \ldots, d_{n-1}) \sqsupseteq u(\mathbf{f})(w)(d_0, \ldots, d_{n-1})$$

Απόδειξη : Ακολουθώντας τις ίδιες ιδέες όπως στην απόδειξη για το θεώρημα 8.1 (αλλά τώρα με τη χρήση υπολογιστικής επαγωγής στις προσεγγίσεις της u). ∎

Η *map_sem_il* δεδομένης μίας λίστας εκφράσεων *el*, στις οποίες αντιστοιχούν οι τύποι *sl*, βρίσκει την ακολουθία σημασιολογικών τιμών για αυτή τη λίστα στο περιβάλλον *u*. Γίνεται σε $proofmode$ και επαγωγικά ως προς το typing της λίστας των εκφράσεων, *tlist*.

```
Definition map_sem_il (pi :Var → STyp) (el : list Expr_IL) (sl : list STyp) (w : World)
                      (u : Env pi SDom_Star) (tlist : typExprList_IL pi el sl) :
                        dom_args (st_arrow sl).
Proof.
  intros.
  induction tlist.
  exact tt.
```

```
  unfold dom_args , type_args ; simpl.
  exact (sem_il e t u w, IHtlist).
Defined.
```

Η *append_dom_args* συνενώνει δύο ακολουθίες σημασιολογικών τιμών *dl1* και *dl2*.

```
Definition append_dom_args (sl1 : list STyp) (dl1 : dom_args (st_arrow sl1))
                                    (sl2 : list STyp) (dl2 : dom_args (st_arrow sl2)) :
                                      dom_args (st_arrow (sl1 ++ sl2)).
Proof.
intros.
induction sl1.
simpl.
exact dl2.
assert (H : (a::sl1) ++ sl2 = a ::(sl1 ++ sl2)).
apply lemma33.
rewrite H ; unfold dom_args ,type_args , tuple .
exact (fst dl1 , IHsl1 (snd dl1)).
Defined.
```

Η *transform_type* μετασχηματίζει τον τύπο *s*, λαμβάνοντας υπόψη την υπόθεση για τη διάταξη των υποτύπων του *s* και καταργώντας τους υποτύπους ανώτερης τάξης.

```
Definition transform_type (m : nat) (s : STyp) : STyp :=
  match s with
    | st_arrow sl ⇒
        st_arrow (drop (find_l_2 m sl) sl)
  end.
```

Το λήμμα *preserve_types* εκφράζει το γεγονός ότι είτε μετασχηματίσουμε τη λίστα υποτύπων που συνθέτουν την *pi f* και μετά κρατήσουμε τα πρώτα *l* στοιχεία της μετασχηματισμένης ακολουθίας είτε κρατήσουμε εξ'αρχής τα πρώτα *l* στοιχεία της αρχικής ακολουθίας είναι το ίδιο, καθ'ότι η τάξη των πρώτων *l* στοιχείων της ακολουθίας, βάσει της γνωστής σύμβασης, είναι $m-1$, οπότε δεν επηρεάζονται από το μετασχηματισμό.

```
Definition preserve_types(m : nat)(f : Var)(pi :Var → STyp) :
  take (find_l pi m f)(map (transform_type m) (type_args (pi f )))=
  take (find_l pi m f)(type_args (pi f )).
Proof.
Admitted.
```

Η *take_trans_typ* επιλέγει, δεδομένης της *tlist*, τη λίστα από τα typing των *l* πρώτων στοιχείων της *tlist*μετασχηματισμένων.

```
Definition take_trans_typ (m : nat)(f : Var)(el : list Expr_IL)(pi :Var → STyp)(P : Prog_IL)
    (tlist : typExprList_IL pi el (type_args (pi f )))

    : typExprList_IL (transform_pi pi m)
      (take (find_l pi m f ) (map (transform_7_2 pi m P) el))
      (take (find_l pi m f ) (map (transform_type m) (type_args (pi f )))).
Proof.
intros.
apply take_typing.
apply transform_typing_exprlist_IL ; assumption.
Qed.
```

Η *sem_il_for_every_expr* με τη βοήθεια των λημμάτων/ορισμών που δόθηκαν παραπάνω παράγει τη λίστα των σημασιολογικών τιμών του δεύτερου μέλους της αποδεικτέας πρότασης.

```
Definition sem_il_for_every_expr (pi :Var → STyp)(m : nat)(f : Var)(el : list Expr_IL)
    (w : World)(u : Env (transform_pi pi m) SDom_Star)
    (dl2 : dom_args ((transform_pi pi m) f))(P : Prog_IL)
    (tlist : typExprList_IL pi el (type_args (pi f))) : dom_args (pi f) .
Proof.
intros.
unfold dom_args.
rewrite ← take_drop_sanity with (l := type_args (pi f))(n:=find_l pi m f) .
unfold transform_pi in dl2.
assert (tlist_trans : typExprList_IL (transform_pi pi m)
                                                    (map (transform_7_2 pi m P) el)
                                                    (map (transform_type m) (type_args (pi f)))).
apply transform_typing_exprlist_IL ; assumption.

assert (take_tlist : typExprList_IL (transform_pi pi m)
                                                (take (find_l pi m f)(map (transform_7_2 pi m P) el))
                                                (take (find_l pi m f)(map (transform_type m)(type_args (pi f))))).
apply take_trans_typ;assumption.
assert (dl1 : dom_args (st_arrow (take (find_l pi m f)
                                (map (transform_type m)(type_args (pi f)))))).
apply (map_sem_il w u take_tlist).
rewrite ←preserve_types.
apply (append_dom_args dl1 dl2).
Qed.
```

Το *sem_il_bot* εκφράζει ακριβώς την ιδιότητα ότι το bottom περιβάλλον *Ebot_Star* αναθέτει bottom σημασιολογία σε κάθε συνάρτηση *f*.

```
Theorem sem_il_bot (f : Var)(pi :Var → STyp):
    sem_il (f_expr_IL f) (f_typ_IL pi f) (Ebot_Star pi) = SDombot_Star (pi f).
intros.
unfold sem_il , sem_il_expr_list.
simpl.
unfold Ebot_Star.
unfold Ebot.
unfold SDombot_Star.
unfold typing_expr_IL_list_rec.
case (pi f); simpl; auto.
Qed.
```

Η *sem_il_sanity* εκφράζει προφανή ιδιότητα της σημασιολογικής συνάρτησης *sem_il* αναφορικά με μία έκφραση τύπου *f_expr_IL*.

```
Theorem sem_il_sanity (f : Var)(pi : Var → STyp) (u : Env pi SDom_Star):
    sem_il (f_expr_IL f) (f_typ_IL pi f) u = u f .
Proof.
intros.
unfold sem_il , sem_il_expr_list;simpl;unfold typing_expr_IL_list_rec ;auto.
Qed.
```

Τα βοηθητικά λήμματα *eq_env* και *q_sem_dom* χρησιμοποιούνται στην απόδειξη του βασικού θεωρήματος *theorem_8_1_a* :

```
Theorem eq_env (pi :Var → STyp)(m : nat) :
 (fun (f0 : Var)(w : World)(dl : tuple SDom (drop (find_l pi m f0) (type_args (pi f0)))) ⇒ Dbot)
 = Ebot_Star (transform_pi pi m).
```

```
intros.
unfold Ebot_Star , transform_pi.
unfold Ebot,SDom_Star,SDombot_Star;reflexivity.
Qed.

Theorem q_sem_dom (pi :Var → STyp)(f : Var) (m : nat)(q_aux' : Q_aux)(w : World)
    (dl : dom_args (transform_pi pi m f)):
  Q_sem q_aux' (SDombot_Star (transform_pi pi m f)) w dl= SDombot ((transform_pi pi m) f) dl.
Proof.
fix 4.
intros.
unfold Q_sem.
induction q_aux';[unfold call_sem ;simpl;reflexivity|unfold actuals_sem;simpl; reflexivity| | ].
fold Q_sem in ×; apply q_sem_dom.
fold Q_sem in ×; apply q_sem_dom.
Qed.
```

Η διατύπωση του θεωρήματος 9.1.1 και ένα κομμάτι τς απόδειξης που αφορά κυρίως την περίπτωση μηδενικού κ. Το *pi_m_1* αναφέρεται στο *pi* για το στάδιο *m_1*, το *typ_P_m_1* στο typing του προγράμματος για το στάδιο *m_1*, το *u_m* στο ελάχιστο περιβάλλον για το στάδιο *m*, το *dl'* στην ακολουθία των σημασιολογικών τιμών όπως προσδιορίζεται από τον ορισμό *sem_il_for_every_expr* και τα *f_typ*,*f_typ΄* στα typing της συνάρτησης *f* για τα δύο στάδια.

```
Definition theorem_8_1_a
        (m : nat)(pi :Var → STyp)(Pm : Prog_IL)(typ_P_m : typProg_IL pi Pm)(w : World)
        (f : Var)(xl : list Var) (b : B_IL) (q_aux : Q_aux)(el : list Expr_IL)


        (typb : typB_IL pi b)

          (Heq : pi f = st_arrow (map pi xl))
        (t_list : typExprList_IL pi el (type_args (pi f))) (k : nat)
        (dl : dom_args ((transform_pi pi m) f))
        (i : nat)
        (H1 : i = label Pm f (a_expr_IL (complex q_aux) f el))
        (q_aux' : Q_aux)
        (H2 : q_aux' = Q_aux_list2 ((Q_aux_to_Q_aux_list q_aux) ++ call i (m-1)::nil)) :

    let pi_m_1 := transform_pi pi m in
    let typ_P_m_1 := transform_typing_prog_IL_overall pi m Pm typ_P_m in
    let u_m := sem_prog_aux_IL Pm typ_P_m in
    let u_k' := uk_IL (transform_pi pi m) (step pi Pm m) typ_P_m_1 k in
    let dl' := sem_il_for_every_expr pi m f el w u_k' dl Pm t_list in
    let f_typ := f_typ_IL pi f in
    let f_typ' := f_typ_IL pi_m_1 f in

    In (def_IL f xl b) Pm →

    In_calls (calls Pm f) (call_def (complex q_aux) f el) →
          Drel (func_apply (Q_sem q_aux'(sem_il (pi := pi_m_1) (f_expr_IL f) f_typ' u_k') w) dl)
         (func_apply (Q_sem q_aux (sem_il (pi := pi) (f_expr_IL f) f_typ u_m) w) dl') .
Proof.
intros.
```

induction k.
unfold func_apply;simpl.
simpl in u_k'; subst u_k' .
rewrite eq_env ; subst f_typ'; rewrite sem_il_bot.
rewrite q_sem_dom with (pi := pi)(m := m).
simpl;apply D_is_bot.
subst f_typ' ; rewrite sem_il_sanity.
subst f_typ0; rewrite sem_il_sanity.
rewrite Q_sem_sanity.
rewrite Q_sem_sanity.
unfold u_k' ; unfold pi_m_1.
assert (typb' : typB_IL (transform_pi pi m) (transform_7_3_B_IL pi m Pm b)).
apply transform_typing_B_IL.
assumption.
assert (Heq': transform_pi pi m f =
st_arrow (map (transform_pi pi m)(drop (find_l pi m f) xl))).
admit.
rewrite th6_1_IL with (pi := transform_pi pi m) (Hb := typb')(Heq := Heq').
unfold u_m.
assert (H11 : func_apply (s:=pi f) (sem_prog_aux_IL Pm typ_P_m f (Q_sem_aux q_aux w)) dl' =
(func_apply (s:=st_iota) (sem_B_IL pi b typb
(env_update_list pi SDom_Star xl (tuple_abstract (cast dl' Heq))
(sem_prog_aux_IL Pm typ_P_m))(Q_sem_aux q_aux w)) tt)).
admit.
rewrite H11.
Admitted.

Θεώρημα 8.3: Έστω u και $\widehat{u}$ τα ελάχιστα περιβάλλοντα που ικανοποιούν υπό τη συγχρονική προσέγγιση τους ορισμούς στο $\mathbf{P}_m$ και $\mathbf{P}_{m-1}$ αντίστοιχα . Τότε :

- Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B}_\mathbf{f})$ μέσα στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ και υπάρχει $0 \leq l \leq n-1$ τ.ώ. $order(\sigma_0) = (m-1), \ldots, order(\sigma_{l-1}) = (m-1)$ και $order(\sigma_l) < (m-1), \ldots, order(\sigma_{n-1}) < (m-1)$, και κάθε κλήση $\mathbf{E} = \mathbf{Q}\ (\mathbf{f})\ (\mathbf{E}_0, \ldots, \mathbf{E}_{n-1})$ στην $\mathbf{f}$ μέσα στο $\mathbf{P}_m$, τότε για όλα τα $d_l \in [\![\sigma_l]\!], \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]$ και για όλα τα $w \in W$,

  $$Q \cdot call_i^{m-1}(\widehat{u}(\mathbf{f}))(w)(d_l, \ldots, d_{n-1}) =$$
  $$Q(u(\mathbf{f}))(w)([\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_0)]\!]^*(\widehat{u})(w), \ldots, [\![\mathcal{E}_{\mathbf{P},m}(\mathbf{E}_{l-1})]\!]^*(\widehat{u})(w), d_l, \ldots, d_{n-1})$$

  όπου $i = label(\mathbf{P}, \mathbf{f}, \mathbf{E})$.

- Για κάθε ορισμό $(\mathbf{f}\ (\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) \doteq \mathbf{B}_\mathbf{f})$ στο $\mathbf{P}_m$, αν $\mathbf{x}_0 : \sigma_0, \ldots, \mathbf{x}_{n-1} : \sigma_{n-1}$ και $order(\sigma_0) < (m-1), \ldots, order(\sigma_{n-1}) < (m-1)$, τότε $d_0 \in [\![\sigma_0]\!], \ldots, d_{n-1} \in [\![\sigma_{n-1}]\!]$ και για όλα τα $w \in W$,

  $$\widehat{u}(\mathbf{f})(w)(d_0, \ldots, d_{n-1}) = u(\mathbf{f})(w)(d_0, \ldots, d_{n-1})$$

Η διατύπωση του θεωρήματος :

Theorem *theorem_8_1_b*
(*m* : *nat*)(*pi* :*Var* → *STyp*)(*Pm* : *Prog_IL*)(*typ_P_m* : *typProg_IL pi Pm*)(*w* : *World*)
(*f* : *Var*)(*xl* : *list Var*) (*b* : *B_IL*) (*q* : *Q*) (*el* : *list Expr_IL*)(*dl* : *dom_args* (*pi f*))
(*t_list* : *typExprList_IL pi el* (*type_args* (*pi f*)))(*H2* : *find_l pi m f* = 0) :

let u_m := *sem_prog_aux_IL Pm typ_P_m in*

let typ_P_m_1 := transform_typing_prog_IL_overall pi m Pm typ_P_m in
let u_m_1 := sem_prog_aux_IL (step pi Pm m) typ_P_m_1 in

let pi_m_1 := transform_pi pi m in

let f_typ := f_typ_IL pi f in
let f_typ' := f_typ_IL pi_m_1 f in
let dl' := cast_dl f pi m dl H2 in

Drel (func_apply (sem_il (pi := pi_m_1) (f_expr_IL f) f_typ' u_m_1 w) dl')
(func_apply (sem_il (pi := pi) (f_expr_IL f) f_typ u_m w) dl).

Proof.
Admitted.

Απόδειξη : Άμεση συνέπεια των Θεωρημάτων 8.1 και 8.2. ∎

Προσέξτε ότι παρ'όλο που στα επόμενα θα χρησιμοποιήσουμε μόνο τη δεύτερη από τις προτάσεις του παραπάνω Θεωρήματος , η πρώτη θα είναι απαραίτητη (η απόδειξη του βήματος του επαγωγικού βήματος της δεύτερης πρότασης χρησιμοποιεί σε κάποιο σημείο την επαγωγική υπόθεση της πρώτης).

Το ακόλουθο θεώρημα δίνει ότι τα προγράμματα $\mathbf{P}_m$ και $\mathbf{P}_{m-1}$ είναι σημασιολογικά ισοδύναμα υπό τη συγχρονική προσέγγιση .

Θεώρημα 8.4: Έστω u και $\widehat{u}$ τα ελάχιστα περιβάλλοντα που ικανοποιούν υπό τη συγχρονική προσέγγιση τους ορισμούς στα $\mathbf{P}_m$ και $\mathbf{P}_{m-1}$ αντίστοιχα . Τότε, $[\![\mathbf{P}_m]\!]^*(u) = [\![\mathbf{P}_{m-1}]\!]^*(\widehat{u})$.

Απόδειξη : Εύκολο αν εφαρμόσουμε τη δεύτερη πρόταση του Θεωρήματος 8.3 στη μεταβλητή **result** των προγραμμάτων $\mathbf{P}_m$ και $\mathbf{P}_{m-1}$. ∎

Theorem *theorem_8_4*
(Pm : Prog_IL)(pi : Var → STyp)(typ_P_m : typProg_IL pi Pm)(m : nat) (w : World) :
let Pm_1 := step pi Pm m in
let typ_P_m_1 := transform_typing_prog_IL_overall pi m Pm typ_P_m in
let pi_m_1 := transform_pi pi m in
sem_prog_IL (pi := pi) Pm typ_P_m w = sem_prog_IL (pi := pi_m_1) Pm_1 typ_P_m_1 w .
Proof.
Admitted.

Απομένει να δείξουμε ότι το αρχικό συναρτησιακό πρόγραμμα $\mathbf{P}_M$, έχει την ίδια δηλωτική σημασιολογία για το τελικό πρόγραμμα μηδενικής τάξης $\mathbf{P}_0$. Αυτό δίνεται με το παρακάτω θεώρημα :

Θεώρημα 8.5: Έστω $\mathbf{P}_M$ ένα M-τάξης FL πρόγραμμα και έστω $\mathbf{P}_{M-1}, \ldots, \mathbf{P}_0$ τα νοηματικά ενδιάμεσα προγράμματα του αλγορίθμου μετασχηματισμού. Έστω u_M και u_0 τα ελάχιστα περιβάλλοντα που ικανοποιούν τους ορισμούς των $\mathbf{P}_M$ και $\mathbf{P}_0$ υπό τη standard προσέγγιση . Τότε για κάθε $w \in W$,

$$[\![\mathbf{P}_M]\!]_D(u_M) = [\![\mathbf{P}_0]\!]_{(W \to D)}(u_0)(w)$$

Απόδειξη : Έστω $\widehat{u}_M, \ldots, \widehat{u}_0$ τα ελάχιστα περιβάλλοντα που ικανοποιούν τους ορισμούς στα προ-

γράμματα $\mathbf{P}_M, \ldots, \mathbf{P}_0$ υπό τη συγχρονική προσέγγιση. Τότε για όλα τα $w \in W$:

$$
\begin{array}{ll}
 & [\![\mathbf{P}_M]\!]_D(u_M) = \\
= & [\![\mathbf{P}_M]\!]^*_D(\widehat{u}_M)(w) \\
 & \text{(Theorem 6.3)} \\
= & [\![\mathbf{P}_{M-1}]\!]^*_D(\widehat{u}_{M-1})(w) \\
 & \text{(Theorem 8.4)} \\
 & \ldots \\
= & [\![\mathbf{P}_0]\!]^*_D(\widehat{u}_0)(w) \\
 & \text{(Theorem 8.4)} \\
= & [\![\mathbf{P}_0]\!]^*_{(W \to D)}(u_0)(w) \\
 & \text{(Theorem 6.4)}
\end{array}
$$

■

Η θεωρητική απόδειξη ορθότητας που δόθηκε παραπάνω δεν πιστοποίησε απλά την ορθότητα του μετασχηματισμού, αλλά κατέδειξε αλλαγές που έπρεπε να γίνουν στον αρχικό αλγόριθμο που προτάθηκε στο [Wadg91]. Προσέξτε ότι ο μετασχηματισμός για προγράμματα ανώτερης τάξης είναι πολύ πιο σοφιστικέ από αυτό της πρώτης τάξης στο [Rond97b]. Βέβαια κάθε διαισθητική προσέγγιση σε ένα θέμα πρέπει να συνοδεύεται από αυστηρή απόδειξη των προτεινόμενων ιδεών.

Αναφορικά με την υλοποίηση της απόδειξης σε Coq, υπάρχει δρόμος ακόμα. Η ενασχόληση μας με την απόδειξη κατέδειξε ότι κάτι το οποίο είναι διαισθητικά ορθό και το οποίο από θεωρητικής άποψης μπορεί να προσεγγιστεί εύκολα, ενδέχεται να είναι εκπληκτικά δύσκολο να διατυπωθεί στη γλώσσα του εκάστοτε theorem prover. Μία αντικειμενική δυσκολία στη χρήση του Coq έγκειται στην έλλειψη αυτοματισμού του που πολλές φορές αναγκάζει το χρήστη να αποδείξει θεωρήματα προφανή ή low-level. Αυτός είναι και ο λόγος που χρησιμοποιείται κυρίως ο όρος ”theorem assistant” και όχι ”prover” για να περιγράψει το Coq.

Επίσης, η μαθηματική αυστηρότητα που πρέπει να χαρακτηρίζει τον προγραμματισμό σε Coq επιβάλλει να είμαστε ακριβείς στη διατύπωσή μας. Βέβαια μέσω της αυστηρότητας αυτής διευκολύνει στην εύρεση λαθών της θεωρίας ή ανακριβειών γενικότερα. Το βέβαιο είναι ότι υπάρχει ακόμα πεδίο βελτίωσης των theorem assistants για τη δημιουργία πιο προσιτών εργαλείων απόδειξης θεωρημάτων και ορθότητας προγραμμάτων.

Κεφάλαιο 9

Συμπεράσματα

9.1 Σημασία Αποτελεσμάτων

Τα παρουσιαζόμενα εδώ αποτελέσματα έχουν σημαντική πρακτική αξία. Εισάγουν μία εναλλακτική μέθοδο υλοποίησης συναρτησιακών γλωσσών υψηλής τάξεως, διαφορετική από τις γνωστές σήμερα συμβατικές τεχνικές μεθόδους. Η τεχνική αυτή μπορεί να εφαρμοστεί τόσο σε συμβατικές αρχιτεκτονικές με μόνο ένα επεξεργαστή, όσο και σε δικτυακά dataflow συστήματα. Ο αναγνώστης μπορεί να ανατρέξει στο [Ashc95] που περιγράφει την eduction-based υλοποίηση της πολυδιάστατης γλώσσας GLU. Επιπροσθέτως, η τεχνική αυτή μπορεί να αποβεί εξίσου χρήσιμη για Web εφαρμογές, όπως για παράδειγμα στην Intentional HTML [Wadg98]. Τέλος, η προτεινόμενη μέθοδος είναι δυνατόν να χρησιμοποιηθεί στην υλοποίηση προγραμμάτων υψηλής τάξεως για μηχανές fine-grain dataflow, εφόσον αυτές χρησιμοποιηθούν για πρακτικούς λόγους (τα ζητήματα αυτά θα αναλυθούν λεπτομερώς στο επόμενο κεφάλαιο).

Τα αποτελέσματα της παρούσας μελέτης έχουν επιπλέον σημαντική θεωρητική αξία καθώς παρουσιάζουν μια νέα όψη της σχέσης που είναι δυνατόν να αναπτυχθεί μεταξύ του συναρτησιακού προγραμματισμού και της νοηματικής λογικής. Η νοηματική λογική έχει ήδη αποβεί ιδιαιτέρως χρήσιμη στην ανάπτυξη της επιστήμης των ηλεκτρονικών υπολογιστών, ιδίως στο πεδίο της τεχνητής νοημοσύνης και της επαλήθευσης/verification (temporal logics). Συνεπώς, μία σχετικά απλή νοηματική λογική μπορεί να δώσει μία νέα προοπτική στους υπολογισμούς που αφορούν την αποτίμηση εκφράσεων ανώτερης τάξης.

Ακόμη, τα αποτελέσματα της έρευνας αυτής μπορούν να ρίξουν φως στη σχέση μεταξύ της επανάληψης και της αναδρομής, με άλλα λόγια, το άλφα και το ωμέγα τη επιστήμης των ηλεκτρονικών υπολογιστών. Πράγματι, η παρούσα μελέτη μπορεί να εκληφθεί ως γενίκευση της γνωστής υλοποίησης της μεθόδου tail recursion με απλές επαναλήψεις. Στην περίπτωση πρώτης τάξεως μπορούμε να αντιληφθούμε την μετάφραση ως μία τεχνική μείωσης της πρώτης τάξης αναδρομής σε branching-time επανάληψη. Η παρουσιαζόμενη εδώ γενίκευση είναι δυνατόν να γίνει αντιληπτή ως μία τεχνική μείωσης της αναδρομής ανώτερης τάξης σε πολυδιάστατες *branching-time* επαναλήψεις.

Τέλος, τα ανωτέρω αποτελέσματα στοχεύουν στη δημιουργία μίας στενότερης σχέσης μεταξύ της έννοιας του *higher-order* και του *multidimensional*. Το *multidimensional* διαδραματίζει σημαντικό ρόλο σε ποικίλους τομείς της επιστήμης των ηλεκτρονικών υπολογιστών (για παράδειγμα, OLAP/πολυδιάστατες τεχνολογίες βάσεων δεδομένων) και, μέχρι σήμερα, απουσιάζει από τις συναρτησιακές γλώσσες. Δείξαμε επιπλέον ότι μια γλώσσα υψηλής τάξεως περιλαμβάνει ένα είδος πολυδιάστασης (*multidimensionality*) στο υπολογιστικό της μοντέλο. Αυτό υποδηλώνει σε γενικές γραμμές πως το *multidimensional* μπορεί να προστεθεί στις συναρτησιακές γλώσσες (και αντιστρόφως) χωρίς να προκληθούν σημαντικά προβλήματα.

9.2 Ζητήματα Υλοποίησης

Ο αλγόριθμος μετασχηματισμού που αναπτύχθηκε σε αυτή τη μελέτη γενικεύει τον αλγόριθμο μετασχηματισμού για τα προγράμματα πρώτης τάξεως που τυποποιήθηκε από τους Rondogiannis και Wadge στο[Rond97b]. Όπως είδαμε στα προηγούμενα κεφάλαια, ο αλγόριθμος μετασχηματίζει μια

σημαντική κατηγορία υψηλής τάξεως προγραμμάτων σε πολυδιάστατα μηδενικής τάξεως νοηματικά προγράμματα. Ο τελικός νοηματικός κώδικας μπορεί να εκτελεστεί σε συμβατικές αρχιτεκτονικές χρησιμοποιώντας τις ίδιες βασικές αρχές που έχουν περιγραφεί στο [Rond97b][section 12]. Η μόνη διαφορά συνίσταται στο ότι το περιβάλλον είναι τώρα πολυδιάστατο.

Στο [Rond97b] είδαμε ότι οι λίστες των φυσικών αριθμών που δημιουργούνται κατά τη διάρκεια της εκτέλεσης μπορούν να κωδικοποιηθούν ως μικροί φυσικοί αριθμοί, χρησιμοποιώντας τη γνωστή *hash-consing* τεχνική. Η ίδια τεχνική βρίσκει εφαρμογή και εδώ: δεδομένου ενός συναρτησιακού προγράμματος M-τάξεως, το απαιτούμενο περιβάλλον για την εκτέλεσή του είναι M-συνέχειες από λίστες φυσικών αριθμών. Χρησιμοποιώντας hash-consing, το περιβάλλον μετατρέπεται σε M-ακολουθίες από φυσικούς αριθμούς τις οποίες μπορούμε πολύ πιο εύκολα να χειριστούμε. Η κατασκευή του warehouse είναι όμοια με την προηγούμενη. Η μόνη διαφορά έγκειται στο ότι τα tags είναι τώρα πολυδιάστατα.

Με άλλα λόγια, η υλοποίηση της προτεινόμενης τεχνικής συνίσταται σε τρία συστατικά :

- Η *μηχανή εκτέλεσης* που υλοποιεί τη συνάρτηση $EVAL$ που χρησιμοποιήθηκε στην ενότητα 3.
- Η *List Store*, που είναι ένας πίνακας κατακερματισμού για κωδικοποίηση λιστών με μικρούς φυσικούς αριθμούς.
- Το *Warehouse* που χρησιμοποιείται για τη διατήρηση ήδη υπολογισμένων αποτελεσμάτων (αναγνωριστικά μαζί με τις τιμές τους σε συγκεκριμένα περιβάλλοντα).

Ο αλγόριθμος μετασχηματισμού που προτείνεται σε αυτή τη μελέτη έχει τεθεί σε εφαρμογή και έχει παράγει σημαντικά αποτελέσματα [Rond93, Rond94a, Rond95].

Στο [Rond93], η τεχνική αυτή χρησιμοποιείται με σκοπό να δημιουργηθεί ένας compiler για μία ISWIM-like συναρτησιακή γλώσσα (που χρησιμοποιεί υψηλής τάξεως συναρτήσεις, όπως παρουσιάζονται σε αυτή τη μελέτη). Ο compiler υλοποιήθηκε σε C και παράγει κώδικα C σαν έξοδο. Σε αυτή την υλοποίηση η List Store είναι ένας πίνακας κατακερματισμού και οι συγκρούσεις επιλύονται με *linear probing* [Knut75, Algorithm L]. Με άλλα λόγια, όταν πραγματοποιείται μια σύγκρουση, η επόμενη θέση του table εξετάζεται. Η υλοποίηση Warehouse χρησιμοποιεί ένα ανοιχτό σχήμα [Knut75, pages 513–514] στο οποίο επιλύονται οι συγκρούσεις με τη χρήση chaining. Παρά το γεγονός ότι ο compiler δεν χρησιμοποιεί ιδιαίτερες βελτιστοποιήσεις, παρατηρείται πως λειτουργεί με τρόπο συγκρίσιμο με τις γνωστές εφαρμογές που βασίζονται στη μέθοδο graph-reduction.

Μία διαφορετική υλοποίηση δίνεται στο [Rond94a, Rond95]. Η βασική ιδέα είναι ότι το List Store και το Warehouse μπορούν να συγχωνευτούν σε μία δομή. Αυτό μπορεί να επιτευφχθεί αν ενσωματωθούν πληροφορίες σχετικές με το περιβάλλον σε παραδοσιακά εγγραφήματα δραστηριοποίησης. Αυτή η τεχνική φαίνεται να είναι πιο πολλά υποσχόμενη σε σύγκριση με την προηγούμενη καθώς αποφεύγει αρκετά από τα overheads που συνδέονται με το hashing.

Στην παρούσα εργασία δε θα ασχοληθούμε άλλο με ζητήματα υλοποίησης. Σαν ένα τελευταίο σχόλιο, πρέπει να παρατηρηθεί ότι η νοηματική προσέγγιση για την υλοποίηση συναρτησιακών γλωσσών φέρνει μία σειρά από ενδιαφέροντα προβλήματα από οποία απαιτούν επιπλέον μελέτη. Ένα από αυτά είναι ο εύρεση της πολυδιάστασης των μεταβλητών που εμφανίζονται στο τελικό νοηματικό πρόγραμμα . Ειδικότερα, είναι πιθανό μία μεταβλητή σε κάποιο ενδιάμεσο πρόγραμμα να μην εξαρτάται από τη διάσταση του εν λόγω προγράμματος. Σε αυτή την περίπτωση χρειαζόμαστε μόνο μία εγγραφή στο warehouse μαζί με μία ένδειξη ότι η μεταβλητή δεν εξαρτάται από τη διάσταση του προγράμματος. Κατ' αυτό τον τρόπο κερδίζουμε σε χώρο και χρόνο και εξασφαλίζουμε πιο αποδοτικές υλοποιήσεις. Μια πολλά υποσχόμενη προσέγγιση για πολυδιάστατη ανάλυση παρουσιάζεται στο [Dodd96], η οποία βέβαια εφαρμόζεται για διαφορετική κλάση πολυδιάστατων προγραμμάτων.

9.3 Σχετικές Μελέτες

Η μελέτη μας συνδέεται στενά με την πρόσφατη έρευνα σε *higher-order removal* [Chin96] και στο *firstification* [Nela91], των οποίων σκοπός είναι ο μετασχηματισμός ενός συναρτησιακού προγράμ-

ματος υψηλής τάξης σε πρόγραμμα πρώτης τάξης. Το πρακτικό αποτέλεσμα των δύο αυτών τεχνικών μεθόδων έγκειται στο ότι τα τελικά προγράμματα πρώτης τάξεως μπορούν να εκτελεστούν με τρόπο περισσότερο αποτελεσματικό συγκριτικά με τα αρχικά προγράμματα υψηλής τάξεως. Ο μετασχηματισμός των Chin και Darlington τυποποιείται με τη χρήση κανόνων unfold/fold ενώ ο Nelan παίρνει μία πιο άμεση προσέγγιση στο πρόβλημα του *firstification* .

Η μελέτη μας διαφέρει από τις παραπάνω δύο προσεγγίσεις επειδή το αποτέλεσμα των μετασχηματισμών μας είναι ένα πολυδιάστατο νοηματικό πρόγραμμα από μεταβλητές μηδενικής τάξης. Επιπλέον, στόχος μας είναι να μετασχηματίσουμε το αρχικό πρόγραμμα σε ένα form που μπορεί να εκτελεστεί κατά ένα dataflow τρόπο. Αντιθέτως, σκοπός τόσο του firstification όσο και του higher order removal είναι να αποτελέσει ένα μηχανισμό βελτιστοποίησης για προγράμματα ανώτερης τάξης.

Παρά τις διαφορές τους, η περαιτέρω σύγκριση μεταξύ του firstification, του higher order removal και της νοηματικής τεχνικής μπορεί να αποβεί χρήσιμη, καθώς είναι δυνατόν να αποκαλύψει την προοπτική κάθε προσέγγισης. Θεωρούμε ότι μία τέτοιου είδους σύγκριση πρέπει να βασιστεί σε δύο βασικά κριτήρια:

1. Στο *μέγεθο του κώδικα*. Προκειμένου να λάβει χώρα μία τέτοια σύγκριση, θα πρέπει κάποιος να ταυτοποιήσει τις παραμέτρους που διαδραματίζουν σημαντικό ρόλο σε κάθε μία από αυτές τις τεχνικές. Για παράδειγμα, η τάξη του αρχικού προγράμματος φαίνεται να αποτελεί σημαντικό παράγοντα, ο οποίος επηρεάζει το τελικό μέγεθος του κώδικα στην περίπτωση της νοηματικής προσέγγισης. Πιστεύουμε πως η νοηματική προσέγγιση μπορεί περιορίσει το μέγεθος του κώδικα, καθώς συγκεντρώνει σε ένα και μοναδικό ορισμό όλες τις υπάρχουσες παραμέτρους που αντιστοιχούν σε μία formal παράμετρο μίας λειτουργίας.

2. *Ταχύτητα εκτέλεσης και προαπαιτούμενα μνήμης* . Η σύγκριση αυτή είναι πολύ πιο δύσκολο να πραγματοποιηθεί εξαιτίας τις διαφορετικής φιλοσοφίας που ενέχει κάθε τεχνική. Η firstification και η higher order removal οδηγούν σε συμβατικά συναρτησιακά προγράμματα. Αντιθέτως, η νοηματική προσέγγιση οδηγεί σε πολυδιάστατα μηδενικής τάξεως προγράμματα που μπορούν να εκτελεστούν με τεχνικές dataflow . Άλλωστε, μία τέτοια σύγκριση θα απαιτούσε την ύπαρξη αυτόνομων υλοποιήσεων των παραπάνω τριών τεχνικών.

Θα πρέπει επιπλέον να αναφερθεί, ως περιορισμός των τριών παραπάνω προσεγγίσεων, ότι καμία από τις προσεγγίσεις αυτές δεν έχει επεκταθεί έτσι ώστε να εφαρμόζεται σε μία πλήρως υψηλής τάξεως συναρτησιακή γλώσσα. Αυτό υποδηλώνει ότι οι συναρτήσεις υψηλής τάξεως είναι θεμελιώδους φύσεως και ότι η εξάλλειψή τους από τα συναρτησιακά προγράμματα είναι δυνατόν να απαιτεί περισσότερο εξελιγμένες επεκτάσεις των τριών παραπάνω τεχνικών. Επιπλέον, στόχος της τεχνικής που αρχικά προτάθηκε από τον Reynolds [Ashc72] είναι η μείωση της τάξης του αρχικού προγράμματος. Μολαταύτα, προκειμένου αυτό να επιτευχθεί, θα πρέπει να εισαχθούν ορισμένες δομές δεδομένων στο πρόγραμμα. Επιπλέον, ο τελικός κώδικας μιμείται την runtime συμπεριφορά του αρχικού προγράμματος. Συνεπώς, η τεχνική του Reynolds, αν και προωθημένη, δεν εξυπηρετεί τους ίδιους σκοπούς με την τεχνική που προτείνεται στην παρούσα μελέτη.

9.4 Παρατηρήσεις και μελλοντική έρευνα

Σε αυτή τη μελέτη παρουσιάσαμε και τυποποιήσαμε μία τεχνική μετασχηματισμού μίας συγκεκριμένης κατηγορίας υψηλής τάξεως συναρτησιακών προγραμμάτων σε μηδενικής τάξεως πολυδιάστατα προγράμματα. Ο μετασχηματισμός που προτείνουμε παρουσιάζει πρακτικό ενδιαφέρον, καθώς είναι δυνατόν να χρησιμοποιηθεί στην υλοποίηση συναρτησιακών γλωσσών με tagged dataflow τρόπο. Η σύνταξη των συναρτησιακών γλωσσών που ελήφθη υπόψη στη μελέτη αυτή επιβάλλει κάποιους περιορισμούς στη χρήση υψηλής τάξεως συναρτησιακών προγραμμάτων. Πιο συγκεκριμένα, τα μόνα μερικώς εφαρμοζόμενα αντικείμενα που είναι δυνατόν να εμφανιστούν στο πρόγραμμα είναι τα ονό-

ματα συναρτήσεων. Ας αναφέρουμε για παράδειγμα το ακόλουθο πρόγραμμα :

```
result      ≐ g(8)
g(x)        ≐ twice(add(x),x)
twice(f,y)  ≐ f(f(y))
add(a)(b)   ≐ a+b
```

Αυτό είναι προφανώς ένα μη έγκυρο πρόγραμμα για τη γλώσσα FL. Η κλήση στη συνάρτηση `twice` έχει μία πραγματική παράμετρο, την μερικώς εφαρμοζόμενη κλήση `add(x)` . Στα ακόλουθα, καταδεικνύουμε τα προβλήματα που αντιμετωπίζουμε όταν επιχειρούμε να εφαρμόσουμε την τεχνική που αναπτύσσεται στην παρούσα μελέτη σε προγράμματα όπως τα παραπάνω. Η υψηλής τάξεως τυπική παράμετρος στο πρόγραμμα αυτό είναι η f της συνάρτησης `twice`. Αν προσπαθήσουμε, ως συνήθως, να εξαλείψουμε την παράμετρο αυτή οδηγούμαστε στο παρακάτω αποτέλεσμα :

```
result      ≐ g(8)
g(x)        ≐ call¹₀(twice)(x)
twice(y)    ≐ f(f(y))
add(a)(b)   ≐ a+b
f           ≐ case¹(actuals¹₀(add(x)))
```

Παρατηρούμε τώρα ότι η μεταβλητή `x` εμφανίζεται ελεύθερα στον ορισμό της **f**, ενώ είναι δεσμευμένη στον ορισμό της g. Το τελικό πρόγραμμα δεν μπορεί να είναι σημασιολογικά ισοδυναμο με το αρχικό. Συνεπώς, ο μετασχηματισμός θα πρέπει να πραγματοποιηθεί με διαφορετικό τρόπο. Εικάζουμε ότι ο γενικευμένος μετασχηματισμός πρέπει πρώτα απ ' όλα να αντιμετωπίσει τις μεταβλητές εκείνες που προκαλούν προβλήματα (όπως η formal παράμετρος **x** της **g** που αναφέρθηκε παραπάνω) . Οι μελετητές ερευνούν τεχνικές εφαρμογής του μετασχηματισμού σε γενικά υψηλής τάξεως προγράμματα.

Ένα άλλο εξίσου ενδιαφέρον πρόβλημα που χρίζει περαιτέρω έρευνας είναι η αντιμετώπιση των πολυδιάστατων νοηματικών γλωσσών ως γλωσσών προγραμματισμού (και όχι απλά ως γλωσσών σχετιζόμενων με το μετασχηματισμό) καθώς επίσης και η έρευνα άλλων δυνατών εφαρμογών που είναι δυνατόν να έχουν. Μία προσέγγιση προς αυτή την κατεύθυνση πραγματοποιήθηκε στο [Rond97a] για την περίπτωση των νοηματικών γλωσσών προγραμματισμού. Θεωρούμε πως μία τέτοια δυνατότητα υπάρχει και στο πεδίο του νοηματικού προγραμματισμού.

Βιβλιογραφία

[Ashc72] E. A. Ashcroft, A. A. Faustini, R. Jagannathan and W. W. Wadge, "Definitional Interpreters for Higher-Order Programming Languages", in *Proceedings of the 25th ACM National Conference*, pp. 717–740, 1972.

[Ashc95] E. A. Ashcroft, A. A. Faustini, R. Jagannathan and W. W. Wadge, *Multidimensional programming*, Oxford University Press, 1995.

[Chin96] W. Chin and J. Darlington, *A Higher-Order Removal Method*, Oxford University Press, 1996.

[Dodd96] C. Dodd, *Rank Analysis in the GLU Compiler*, World Scientific, 1996.

[Dowt81] D. Dowty, R. Wall and S. Peters, *Introduction to Montague Semantics*, Reidel Publishing Company, 1981.

[Du90a] W. Du and W. W. Wadge, *Spreadsheet Based on Intensional Logic*, IEEE Software, 1990.

[Du90b] W. Du and W. W. (1990b). Wadge, *The Eductive Implementation of a Three-dimensional Spreadsheet*, Software-Practice and Experience, 1990.

[Gall75] D. Gallin, *Intensional and Higher-Order Modal Logic*, North-Holland, 1975.

[Gunt92] C. Gunter, *Semantics of Programming Languages*, The MIT Press, 1992.

[Knut75] D. E. Knuth, *The Art of Computer Programming (Sorting and Searching)*, vol. Vol. 3., Addison-Wesley, 1975.

[Nela91] G. Nelan, *Firstification*, Ph.D. thesis, Department of Computer Science, Arizona State University, U.S.A., 1991.

[R91] Tennent R., *Semantics of Programming Languages*, Prentice Hall, 1991.

[Rond93] P. Rondogiannis and W. W. Wadge, "A Dataflow Implementation Technique for Lazy Typed Functional Languages", in *Proceedings of the Sixth International Symposium on Lucid and Intensional Programming*, pp. 23–42, 1993.

[Rond94a] P. Rondogiannis, *Higher-Order Functional Languages and Intensional Logic*, Ph.D. thesis, Department of Computer Science, University of Victoria, Canada., 1994.

[Rond94b] P. Rondogiannis and W. W. Wadge, "Compiling Higher-Order Functions for Tagged-Dataflow", in *Proceedings of the IFIP International Conference on Parallel Architectures and Compilation Techniques*, pp. 269–278, North-Holland, 1994.

[Rond95] P. Rondogiannis and W. W. Wadge, "Higher-Order Dataflow and its Implementation on Stock Hardware", in *Proceedings of the ACM Symposium on Applied Computing*, pp. 431–435, ACM Press, 1995.

[Rond97a] P. Rondogiannis, M. Gergatsoulis and T. Panayiotopoulos, "Cactus: A Branching-Time Logic Programming Language", in *Proceedings of the International Joint Conference on Qualitative and Quantitative Practical Reasoning*, pp. 511–524, Springer Verlag, 1997.

[Rond97b] P. Rondogiannis and W. W. Wadge, *First-Order Functional Languages and Intensional Logic*, Springer Verlag, 1997.

[Stoy77] J. Stoy, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1977.

[Thom74] R. (ed). Thomason, *Formal Philosophy, Selected Papers of R. Montague*, Yale University Press, 1974.

[Wadg85] W. W. Wadge and E. A. Ashcroft, *Lucid, the Dataflow Programming Language*, Academic Press, 1985.

[Wadg91] W. W. Wadge, "Higher-Order Lucid", in *Proceedings of the Fourth International Symposium on Lucid and Intensional Programming*, 1991.

[Wadg98] W.W. Wadge, G.D. Brown, m.c. schraefel and T. Yildirim, "Intensional HTML", in *Proceedings of the Fourth International Workshop on Principles of Digital Document Processing (PODDP '98)*, pp. 128–139, Springer Verlag, 1998.

[Yagh84] A. A. Yaghi, *The Intensional Implementation Technique for Functional Languages*, Ph.D. thesis, Department of Computer Science, University of Warwick, Coventry, UK., 1984.