Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και
Υπολογιστων

Ανάπτυξη Βιβλιοθήκης Θεωριών για την Απόδειξη Ιδιοτήτων Πιστοποιημένου Κώδικα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΟΔΩΡΗΣ Γ. ΤΣΟΚΟΣ

Επιβλέπων : Νίκος Παπασπύρου
Λέκτορας Ε.Μ.Π.

Αθήνα, Οκτώβριος 2005

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και
Υπολογιστων

Ανάπτυξη Βιβλιοθήκης Θεωριών για την Απόδειξη Ιδιοτήτων Πιστοποιημένου Κώδικα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΘΟΔΩΡΗΣ Γ. ΤΣΟΚΟΣ

Επιβλέπων : Νίκος Παπασπύρου
Λέκτορας Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20ή Οκτωβρίου 2005.

.....	.....	.....
Νίκος Παπασπύρου Λέκτορας Ε.Μ.Π.	Στάθης Ζάχος Καθηγητής Ε.Μ.Π.	Γιώργος Κολέτσος Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2005

Θοδωρής Γ. Τσόκος

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

Περίληψη

Σκοπός αυτής της διπλωματικής εργασίας είναι η μελέτη συστημάτων λάμβδα λογισμού, η αντιστοιχία τους με συστήματα μαθηματικής λογικής, και στη συνέχεια η καταγραφή και απόδειξη χρήσιμων θεωριών σε μια παραλλαγή του λογισμού των επαγωγικών κατασκευών (Calculus of Inductive Constructions — CIC), τη γλώσσα τύπων nflint. Η βιβλιοθήκη θεωριών που αναπτύχθηκαν μπορούν να ενσωματωθούν στην ήδη υπάρχουσα δουλειά, στην προσπάθεια κατασκευής ενός συστήματος παραγωγής πιστοποιημένου εκτελέσιμου κώδικα.

Ο λάμβδα-λογισμός είναι ένα πλήρες υπολογιστικό σύστημα με μεγάλη εκφραστικότητα. Από τις αρχές της δεκαετίας του 1930, όταν Alonso Church όρισε τον κλασσικό λάμβδα-λογισμό χωρίς τύπους, αναπτύχθηκαν πολλές παραλλαγές του με τύπους οι οποίες αντιστοιχούν, μέσω του ισομορφισμού Curry-Howard, σε συστήματα λογικής. Μέσω αυτής της αντιστοίχισης, ο λαμβδα λογισμός αποτελεί λοιπόν βασικό εργαλείο για τη καταγραφή και την απόδειξη θεωριών.

Η δύναμη της εκφραστικότητας της γλώσσας τύπων nflint, σε συνδυασμό με την τυπικότητα και το βάθος του συστήματος λογικής που αντιστοιχεί στο λογισμό των επαγωγικών κατασκευών, αποτελούν σήμερα μία ελπιδοφόρα πρόταση στο ζήτημα της παραγωγής πιστοποιημένου και ασφαλούς εκτελέσιμου κώδικα, ένα ζήτημα που έχει απασχολήσει πολύ τελευταία, όταν μεγάλα συστήματα λογισμικού έδειξαν αδυναμία στην ασφάλεια, προκαλώντας έτσι και οικονομικά προβλήματα. Έτσι, με βάση μία γλώσσα προγραμματισμού με ισχυρό σύστημα τύπων, καταγράφονται και αποδεικνύονται χρήσιμες ιδιότητες του εκτελέσιμου κώδικα, πιστοποιώντας έτσι τη σωστή και ασφαλή λειτουργία του.

Σε ένα τέτοιο πλαίσιο πιστοποίησης της ασφάλειας του εκτελέσιμου κώδικα, η γλώσσα τύπων nflint είναι το μέσο για την καταγραφή των τυπικών συλλογισμών. Όμως, απαιτείται ένα σύνολο θεωριών και αποδείξεων της λογικής που θα χρησιμοποιηθούν για την απόδειξη των ιδιοτήτων που θα καταγραφούν με τη χρήση της nflint. Επειδή το σύστημα υποστήριξης αποδείξεων Coq χρησιμοποιεί και αυτό μια παραλλαγή του λογισμού των επαγωγικών κατασκευών και διαθέτει μια πλούσια βιβλιοθήκη θεωριών, κρίθηκε ουσιαστικό και αποτελεσματικό, η ανάπτυξη ενός συνόλου βιβλιοθηκών για τη γλώσσα τύπων να ακολουθήσει τη δομή της βιβλιοθήκης του συστήματος Coq. Έτσι, με ημιαυτόματο τρόπο, αναπτύχθηκε στο πλαίσιο αυτής της διπλωματικής εργασίας ένα σύνολο από θεωρίες και αποδείξεις της λογικής — βασικοί τύποι, προτασιακή και κατηγορηματική λογική, αριθμητική Peano — στη γλώσσα τύπων nflint.

Λέξεις κλειδιά

Συστήματα τύπων, λάμβδα λογισμός, λογισμός των επαγωγικών κατασκευών, προγραμματισμός με αποδείξεις, ασφάλεια εκτελέσιμου κώδικα, γλώσσα τύπων nflint, Coq.

Abstract

The purpose of this diploma dissertation is the study of lambda calculi, their connection to systems of mathematical logic, as well as the definition and proof of useful theories in a variation of the Calculus of Inductive Constructions (CIC), the nflint type language. The library of theories that has been developed can be incorporated in existing work, in an overall effort of constructing a system for generating certified executable code.

Lambda calculus is a complete system of computation with great expressiveness. Since the early 1930s, when Alonso Church defined the classic untyped lambda calculus, many typed variations have been developed which correspond, via the Curry-Howard isomorphism, to systems of logic. Through this correspondence, lambda calculus can be used as a tool for the development and proof of theories.

The expressive power of the nflint type language in combination with the formality and depth of the logic system that corresponds to the calculus of inductive constructions form nowadays a promising proposal towards building certifiably safe executable code. This issue has gained interest lately, as a failure in the safety of large software systems can lead to significant financial problems. According to this proposal, based on a programming language with a strong type system, it is possible to state and prove useful properties of executable code, thus certifying its correct and safe operation.

In such a framework for certifying the safety of executable code, the nflint type language provides the means of formal reasoning. However, a set of theories and their proofs is needed, which will be used to facilitate the proof of program properties. As the Coq proof assistant uses a variation of the calculus of inductive constructions and possesses a rich library of theories, the library of theories for nflint was chosen to follow the structure of Coq's. In a semi-automatic manner, a set of theories were stated and proved throughout this diploma dissertation — basic types, propositional and predicate logic, Peano arithmetic — in the nflint type language.

Key words

Type systems, lambda calculus, calculus of inductive contructions, programming with proofs, security of executable code, type language nflint, Coq.

Ευχαριστίες

Κατ' αρχήν, θα ήθελα να ευχαριστήσω τον κ. Νίκο Παπασπύρου, επιβλέποντα καθηγητή της διπλωματικής μου, για την προσοχή και υπομονή που έδειξε εξ αρχής, την αμέριστη προσπάθεια τον τελευταίο καιρό πριν την παρουσίαση, αλλά και γιατί με ενέπνευσε να ασχοληθώ με αυτό το κομμάτι της επιστήμης των υπολογιστών. Θέλω επίσης να ευχαριστήσω το συνάδελφο Μιχάλη Παπακυριάκου για την καθοδήγηση και τις συμβουλές του, όπως επίσης και τον καθηγητή κ. Γιώργο Κολέτσο για τη διδασκαλία του στον τομέα της μαθηματικής λογικής. Θα ήθελα να ευχαριστήσω τους φίλους Ελένη, Νίκο, Θοδωρή, Φαίδωνα, Μήτσο, Χρυσάνθη, Κώστα, γιατί όλα αυτά τα χρόνια στην Αθήνα μεγάλωσα μαζί τους και τους χρωστάω πολλά. Επίσης, ο Αστέριος Λάμπρου με την παθιασμένη του δίψα για επιστημονική αναζήτηση, ακόμη και για ζητήματα της καθημερινότητας, μου έδωσε άλλη οπτική για την έννοια του διαβάσματος. Τέλος, ευχαριστώ τους γονείς μου, γιατί πάντα με στήριζαν στις δύσκολες στιγμές και μου έδειξαν εξ αρχής τον απαραίτητο, όμορφο αλλά όχι αγχωτικό, δρόμο της γνώσης.

Θοδωρής Γ. Τσόκος,

Αθήνα, 20 Οκτωβρίου 2005.

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-5-05, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Οκτώβριος 2005.

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

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός της εργασίας

Σκοπός της εργασίας είναι η ανάπτυξη βιβλιοθήκης θεωριών για την απόδειξη ιδιοτήτων πιστοποιημένου κώδικα. Έτσι, υλοποιήθηκε ένας μετατροπέας θεωρημάτων και αποδείξεων, από το σύστημα διαχείρισης τυπικών αποδείξεων *Coq* στη γλώσσα τύπων *nflint*. Ο μετατροπέας αυτός, μπορεί να αποδώσει στο 70% περίπου των θεωρημάτων, ορισμών και λημμάτων που εξετάσαμε, ενώ η έξοδός του προσέγγιζε από 60% έως 95%, το ζητούμενο αποτέλεσμα. Στη συνέχεια, διορθώσεις αλλά και καταγραφή των θεωριών που δεν πέρασαν από την εφαρμογή, έγιναν με το χέρι.

Η γλώσσα *nflint* είναι εκφραστική ούτως ώστε να μπορεί να χρησιμοποιηθεί για την κωδικοποίηση κατηγορηματικής λογικής υψηλής τάξης. Αυτό αποδεικνύεται από το σύνολο των θεωρημάτων που μεταφράστηκαν στη γλώσσα αυτή.

1.1.1 Αποδείξεις θεωρημάτων

Το ζήτημα που απασχόλησε την έρευνα στο παρελθόν είναι πώς ένα σύστημα τύπων μπορεί να παράγει ένα σύστημα αποδείξεων και πώς ένα σύστημα τύπων μπορεί να αντιστοιχηθεί σε ένα σύστημα λογικής. Στο εξής, θα ασχοληθούμε με το σύστημα τύπων του λ-λογισμού με τύπους και τις επεκτάσεις του, και πιο συγκεκριμένα με το *σύστημα κατασκευής επαγωγικών τύπων* (Calculus of Inductive Constructions).

Οι προτάσεις πάνω σε αυτό το ζήτημα είναι οι αποδείξεις να παριστάνονται ως τύποι και οι αποδείξεις των τύπων είναι όροι της γλώσσας τύπων. Αυτό παρουσίασε και η πρώτη και πιο διαδεδομένη περιγραφή της αναπαράστασης αποδείξεων με συστήματα τύπων, από τους Curry και Howard, πιο γνωστή και ως *ισομορφισμός Curry-Howard*. Η πρόταση αυτή δίνει μία τυπική έκφραση των αποδείξεων σε λ-όρους με τύπους. Έτσι, για τη συνεπαγωγή $A \supset B$ και για τον καθολικό ποσοδείκτη $\forall x{\in}A.\, P(x)$, αντιστοιχώ τον κανόνα εισαγωγής στην αφαίρεση του λ-λογισμού και τον κανόνα της απαλοιφής στην εφαρμογή

Θα θεωρούμε τους τύπους–προτάσεις ως σύνολα και τους όρους–αποδείξεις ως στοιχεία των συνόλων, στη συγκεκριμένη αναπαράσταση αποδείξεων με λ-όρους. Τότε, για να αποδείξουμε έναν τύπο–πρόταση, αρκεί το σύνολο να είναι κατοικημένο, δηλαδή στο πεδίο του λ-λογισμού να υπάρχει κάποιος όρος που να έχει αυτόν τον τύπο. Για παράδειγμα

- Τύπος–πρόταση: $\Pi x{:}A.\, P(x)$ Σύνολο: A

- Ερμηνεία: "Για κάθε στοιχείο x του συνόλου A, ισχύει $P(x)$".

- Απόδειξη της πρότασης: $\lambda x{:}A.\, p(x)$, με τύπο τον παραπάνω όρο.

- Ερμηνεία: "Έστω ένα στοιχείο x του συνόλου A. Αποδεικνύεται ότι ισχύει $P(x)$", με την προϋπόθεση το $p(x)$ να είναι μία απόδειξη της πρότασης $P(x)$.

1.1.2 Ανασκόπηση Συστημάτων Αποδείξεων

Κάποια γνωστά σημερινά συστήματα αποδείξεων είναι τα εξής:

1. Coq `http://coq.inria.fr/`
   Το *Coq*, το οποίο θα αποτελέσει στη συνέχεια και τη βάση για την καταγραφή θεωρημάτων και αποδείξεων στη γλώσσα τύπων, είναι ένα σύστημα αποδείξεων που αναπτύχθηκε από το *Γαλλικό Ινστιτούτο Έρευνας για την Επιστήμη των Υπολογιστών* (INRIA - `http://www.inria.fr/`), ως λογισμικό ανοιχτού κώδικα. Περιλαμβάνει ένα μηχανισμό για αυτοματοποιημένη παραγωγή πιστοποιημένων προγραμμάτων από τις αποδείξεις των προδιαγραφών τους, ένα εύχρηστο γραφικό περιβάλλον, ένα μεγάλο σύνολο θεωριών με αποδείξεις. Το *Coq* θα αναλυθεί και θα παρουσιαστεί ο τρόπος που αποτέλεσε βάση για την καταγραφή των θεωριών στη γλώσσα τύπων στο κεφάλαιο 4.

2. HOL `http://hol.sourceforge.net/`
   Το HOL είναι ένα αυτοματοποιημένο σύστημα αποδείξεων για υψηλής τάξης λογική, δηλαδή ένα προγραμματιστικό περιβάλλον στο οποίο μπορούν να αποδειχθούν θεωρήματα και να υλοποιηθούν εργαλεία αποδείξεων, που με αυτόματες διαδικασίες μπορούν να αποδείξουν αρκετά απλά θεωρήματα. Επιπλέον, μέσω ενός μηχανισμού της Oracle δίνει τη δυνατότητα προσπέλασης από εξωτερικά προγράμματα όπως μηχανές SAT και BDD. Τα τελευταία χρόνια το HOL αναπτύχθηκε ευρέως από το πανεπιστήμιο του Cambridge (HOL88), από τα Calgary and Bell Labs (HOL90) και από συνεργασία των πανεπιστημίων του Cambridge, της Γλασκόβης και της Utah. Η σημερινή έκδοση (HOL4) αναπτύσσεται από ομάδα πανεπιστημίων, στα πλαίσια του προγράμματος PROSPER (`http://www.dcs.gla.ac.uk/prosper/`), ενώ διατίθεται ως λογισμικό ανοικτού κώδικα.

3. Isabelle `http://www.cl.cam.ac.uk/Research/HVG/Isabelle/`
   Είναι κι αυτό ένα σύστημα αποδείξεων. Μαθηματικές φόρμουλες εκφράζονται σε μία τυπική γλώσσα, ενώ το σύστημα παρέχει εργαλεία για την απόδειξη αυτών των θεωρημάτων. Το βασικό του κομμάτι είναι η κανονικοποίηση μαθηματικών εννοιών και η τυπική τους πιστοποίηση, που βρίσκει εφαρμογή στην απόδειξη της ορθότητας είτε λογισμικού είτε υλικού και η απόδειξη ιδιοτήτων γλωσσών προγραμματισμού και πρωτοκόλλων Διαφέρει από τα υπόλοιπα συστήματα αποδείξεων στο ότι είναι μπορεί να δεχτεί πολλές τυπικές λογικές, εκτός της βασικής λογικής υψηλής τάξης. Υπεύθυνοι για το Isabelle είναι το πανεπιστήμιο του Cambridge και το Τεχνολογικό Πανεπιστήμιο του Μονάχου.

4. Lego `http://www.dcs.ed.ac.uk/home/lego/`
   Το Lego είναι ένα διαδραστικό σύστημα ανάπτυξης αποδείξεων, το οποίο έχει σχεδιάσει και υλοποιήσει ο Randy Pollack στο Εδιμβούργο, χρησιμοποιώντας τη γλώσσα προγραμματισμού New Jersey ML. Και αυτό δέχεται περισσότερα από ένα συστήματα τύπων - το Edinburgh Logical Framework (LF), το λογισμό των επαγωγικών κατασκευών (CC), το γενικευμένο λογισμό των κατασκευών (GCC) και την ενοποιημένη θεωρία των εξαρτώμενων τύπων (UTT).

5. Nuprl `http://www.cs.cornell.edu/Info/Projects/NuPrl/`
   Το Nuprl (η "Nu" ήταν μία παλαιότερη εξέλιξη των εκδόσεων του συστήματος, ενώ PRL από το Progam Refinement Logic) είναι ένα σύστημα που ξεκίνησε να αναπτύσσεται από το πανεπιστήμιο του Cornell το 1979 από τον Joseph L. Bates και τον Robert L. Constable. Βασικός του στόχος όπως διατείνονται οι κατασκευαστές είναι να φτιάξουν ένα σύστημα αποδείξεων τόσο εύκολο και εύκολο για προγραμματιστές και μαθηματικούς, όσο εύκολοι και έξυπνοι είναι σήμερα οι επεξεργαστές κειμένου.

6. PVS `http://pvs.csl.sri.com/`
   Το PVS (Prototype Verification System), κατασκευασμένο από το 1993 στο εργαστήριο

επιστήμης υπολογιστών του ερευνητικού ινστιτούτου του Stanford, είναι ένα σύστημα πιστοποίησης, δηλαδή μία γλώσσα προδιαγραφών με ενσωματωμένα βοηθητικά εργαλεία και ένα σύστημα απόδειξης θεωρημάτων. Έχει βρει πρακτική εφαρμογή σε αλγόριθμους και αρχιτεκτονικές για ανοχή σφαλμάτων σε συστήματα ελέγχου πτήσεων, όπως και σε προβλήματα σχεδιασμού συστημάτων υλικού και συστημάτων πραγματικού χρόνου.

7. Twelf `http//www.cs.cmu.edu/~twelf/`
   Το Twelf αναπτύχθηκε στο Τμήμα Επιστήμης των Υπολογιστών στο Πανεπιστήμιο του Carnegie Mellon, από τους Frank Pfenning και Carsten Sch΄urmann, ενώ υποστηρίχθηκε στη συνέχεια και από τους Brigitte Pientka, Roberto Virga και Kevin Watkins. Το Twelf παρέχει μία ενοποιημένη μεταγλώσσα για την περιγραφή, την υλοποίηση και την απόδειξη ιδιοτήτων γλωσσών προγραμματισμού και λογικών. Συνδυάζει το λογικό πλαίσιο LF, τη γλώσσα λογικού προγραμματισμού με περιορισμούς *Elf* και ένα - σε πολύ αρχικό στάδιο - επαγωγικό σύστημα απόδειξης μετα-θεωρημάτων για το πλαίσιο LF.

1.1.3 Ασφάλεια εκτελέσιμου κώδικα

Η ασφάλεια εκτελέσιμου κώδικα είναι κάτι που απασχόλησε και απασχολεί την έρευνα. Η ασφάλεια και η συμβατότητα μεγάλων συστημάτων ήταν αναγκαία από παλιά, για οικονομικούς και πολιτικούς λόγους. Σήμερα, από τη στιγμή που τα μεγάλα συστήματα λογισμικού εξελίσσονται ως αυτόνομα μικρότερα κομμάτια, η ασφάλεια αποκτά πιο ευρεία έννοια. Διαφορετικές ομάδες ανάπτυξης λογισμικού παράγουν μικρότερα λογισμικά συστήματα τα οποία απαιτείται να είναι ασφαλή και διαλειτουργικά, ανεξαρτήτως της πλατφόρμας που θα συνεργαστούν. Μέσω διαδικτύου, από απόσταση ή από κοντά, για τον έλεγχο συσκευών σπιτιού ή για πανεπιστημιακά εργαστήρια, τα κομμάτια αυτά θα πρέπει να συνεργαστούν μέσω διαπροσωπειών που παρέχουν τους κανόνες για τη συνολική λειτουργία του συστήματος. Επιπλέον, μέσω διαδικτύου, ανά πάσα στιγμή μπορεί να βρεθούν αυτόνομα συστήματα λογισμικού - άγνωστης πιστοποίησης και ασφάλειας - που μπορούν να χρησιμοποιηθούν σε μεγαλύτερα συστήματα χωρίς να αποδεικνύεται η ασφάλειά τους με κάποιον γενικά αντικειμενικό και αξιόπιστο τρόπο.

Ένας επιπλέον παράγοντας που κάνει τα σημερινά συστήματα πιο ευάλωτα σε ζητήματα ασφάλειας είναι το ότι κατανέμονται σε περισσότερες υπολογιστικές συσκευές. Σπάζοντας τον εκτελέσιμο κώδικα σε περισσότερα κομμάτια ώστε το καθένα να εκτελεστεί ξεχωριστά ανεβάζει περισσότερο την πιθανότητα λαθών, αφού και το σύστημα πλέον είναι πιο ευάλωτο σε επιθέσεις σε κομμάτια κώδικα, όπου πχ σαν αυτόνομα κομμάτια δεν παρέχουν ασφάλεια όπως παρείχαν στο ενιαίο σύστημα, αλλά και η κατανεμημένη εκτέλεση του κώδικα δεν έχει ελεγχθεί διεξοδικά όπως στο ενιαίο σύστημα, ανεβάζοντας έτσι την πιθανότητα λαθών.

Οι πιο γνωστές τεχνικές αυτή τη στιγμή είναι δύο. Με την πρώτη, αφού ο μεταγλωττιστής ελέγξει το πρόγραμμα, μία νοητή μηχανή και ένας μεταγλωττιστής της τελευταίας στιγμής (Just-In-Time Compiler ή JIT) αναλαμβάνει να ελέγξει σε βάθος την πιστοποίηση του δεδομένου προγράμματος. Αυτήν την τεχνική εφαρμόζουν υλοποιήσεις της γλώσσας προγραμματισμού *Java* και της πλατφόρμας *.NET*. Παρ' όλ' αυτά, επειδή ο έλεγχος γίνεται σε χρόνο εκτέλεσης, η απόδοση του συστήματος περιορίζεται, ενώ και το αποτέλεσμα δεν είναι πάντα ικανοποιητικό. Η δεύτερη τεχνική, που τελευταία αναπτύσσεται ιδιαίτερα, είναι η πιστοποίηση κώδικα μέσω της ταυτοποίησης με τον παραγωγό του. Αυτή η τεχνική μπορεί να γίνει δυνατή με την αντιστοίχηση, μέσω κρυπτογραφικών μεθόδων, του κώδικα με τον παραγωγό, κάτι που βέβαια προφανώς προϋποθέτει την εμπιστοσύνη στον τελευταίο. Και οι δύο τεχνικές, όμως όπως φαίνεται είτε περιορίζουν την απόδοση και λειτουργία του συστήματος, όπως η πρώτη, είτε είναι τελικά πολύ περιορισμένης κλίμακας αφού δεν πιστοποιούν κώδικα αλλά παραγωγούς, δηλαδή προγραμματιστές, όπως η δεύτερη.

Η πιο φιλόδοξη σκέψη είναι ότι με βάση ένα γενικό πλαίσιο, μπορεί να είναι δυνατή η αναπαράσταση σύνθετων προτάσεων και των αποδείξεών τους, σε γλώσσες χαμηλού επιπέδου με ισχυρά συστήματα τύπων. Έτσι, το πρόγραμμα είναι ένα αρχείο πιστοποιημένου κώδικα ο

τύπος του οποίου παρέχει το σύνολο των ιδιοτήτων που έχει το πρόγραμμα. Ο ελεγκτής τύπων τώρα, μπορεί να ελέγξει στατικά και αρκετά εύκολα, αν το δεδομένο αρχείο είναι συνεπές, και εφόσον είναι, το πρόγραμμα είναι ασφαλές να εκτελεστεί.

Με βάση την παραπάνω σκέψη, στόχος είναι η κατασκευή ενός πλαισίου-συστήματος που θα εξασφαλίζει:

1. Αξιοπιστία εκτελέσιμου κώδικα.

2. Ασφάλεια και απόρρητο στα επίπεδα της εφαρμογής/υποδομής.

που από τις ειδικές περιπτώσεις, μπορεί να αναχθεί σε ένα γενικευμένο πλαίσιο, τυπικά ορισμένο, ικανό να επαληθεύσει αυθαίρετες ιδιότητες, εκφρασμένες ως προδιαγραφές υψηλού επιπέδου σε μία κατάλληλη τυπική λογική.

Η λογική που θα χρησιμοποιηθεί πρέπει να είναι ισχυρή ώστε να μπορεί να εκφράσει και να επαληθεύσει αυθαίρετες ιδιότητες συστημάτων λογισμικού. Με βάση την αρχή των "Προτάσεων ως Τύπων" (propositions-as-types), η κατηγορηματική λογική μπορεί να χρησιμοποιηθεί επιτυχώς, ενσωματωμένη σε συστήματα τύπων.

Από την πλευρά του συνολικού πλαισίου–συστήματος τώρα, τα επιθυμητά χαρακτηριστικά του θα πρέπει να είναι τα εξής:

1. *Ευελιξία στο πλήθος των γλωσσών προγραμματισμού.* Θα πρέπει να επιτρέπει σε προγραμματιστές που χρησιμοποιούν γλώσσες υψηλού επιπέδου να μπορούν να χρησιμοποιήσουν διάφορες γλώσσες και κάθε φορά να μπορούν να βγάλουν συμπεράσματα για το πρόγραμμα τους. Επιπλέον, και για γλώσσες χαμηλού επιπέδου το σύστημα θα πρέπει να είναι ευέλικτο, ώστε τόσο για τις υπάρχουσες υπολογιστικές συσκευές, όσο και για τις μελλοντικές, οι προγραμματιστές να μπορούν να γράφουν προγράμματα αποδοτικά, και στη συνέχεια να μπορούν να ελέγχουν την ασφάλειά τους.

2. *Ευελιξία στο ζήτημα της ασφάλειας.* Για διαφορετικές περιοχές εφαρμογών, έχοντας διαφορετικούς αποδέκτες κώδικα, ο κάθε παραλήπτης ορίζει με διαφορετικό τρόπο την ασφάλεια. Η πολιτική της ασφάλειας σε αυτές τις περιπτώσεις θα ήταν ήταν το σύνολο των προδιαγραφών ασφάλειας να μπορεί να παραμετροποιηθεί από τον αποδέκτη. Κάτι τέτοιο θα είναι πολύ πιο αποδοτικό αν γίνει δυναμικά, δηλαδή ο τελικός χρήστης να μπορεί να παραμετροποιεί απόλυτα, χωρίς την παρουσία κάποιας υπηρεσίας πιστοποίησης, αλλά μόνο με τις δικιές του επιλογές.

Έτσι, ακολουθώντας τα παραπάνω, και η γλώσσα τύπων που χρησιμοποιήθηκε σε αυτήν την εργασία για την καταγραφή και ανάλυση θεωριών και αποδείξεων συνοδεύεται από μία κατάλληλη, καλά ορισμένη γλώσσα υπολογισμών.

1.1.4 Ανασκόπηση Συστημάτων Ανάπτυξης Πιστοποιημένου Κώδικα

Ένας proof-assistant είναι ένα "interface" για τον χρήστη, που αποτελείται από δύο κομμάτια, το proof-development και το proof-checking. Τα proof assistants είναι προγράμματα — αλληλεπιδρά ο χρήστης με τη μηχανή — που τρέχουν σε έναν υπολογιστή και βοηθούν το χρήστη να αποκτήσει επαληθευμένες προτάσεις, με δοσμένα κάποιο μαθηματικό σύστημα. Αυτή η επαλήθευση μπορεί να γίνει με δύο τρόπους. Αυτομάτως, με κάποιον *μηχανικό αποδείκτη θεωρημάτων*, ή με τη βοήθεια του χρήστη, ο οποίος μαζί με τη μαθηματική πρόταση δίνει και την απόδειξη η οποία ελέγχεται από το μηχάνημα.

Το πρόβλημα "μία θεωρούμενη απόδειξη, είναι πράγματι απόδειξη;", είναι μετρήσιμο. Το πρόβλημα "ένα θεωρούμενο θεώρημα, είναι πράγματι θεώρημα;", είναι μη μετρήσιμο. Επομένως, ένας αμιγής ελεγκτής απόδειξης — στον οποίο θα πρέπει να εισάγουμε μία πλήρως φορμαλοποιημένη απόδειξη, μία αρκετά δύσκολη διαδικασία — δεν είναι πρακτικός, γιατί είναι πολύ δύσκολο να κάνουμε αυτήν την φορμαλοποίηση, να παρέχουμε δηλαδή αντικείμενα–απόδειξης. Από την άλλη, ένας αμιγής αυτοματοποιημένος αποδείκτης θεωρήματος — δηλαδή

ένα πρόγραμμα που βρίσκει μία απόδειξη για μία πρόταση A, και επίσης ελέγχει ότι δεν υπάρχει απόδειξη που να αντικρούει την πρόταση — είναι πραγματικά αδύνατος, αφού δεν μπορεί να υλοποιηθεί ούτε για θεωρήματα μιας απλής θεωρίας, όπως είναι η κατηγορηματική λογική. Επομένως, μπορούμε να πούμε ότι ο έλεγχος απόδειξης δεν μπορεί να γίνει με μία απολύτως αυτοματοποιημένη διαδικασία, εκτός των περιπτώσεων που έχουμε έναν μερικό αλγόριθμο, ο οποίος τρέχει μέχρι να βρει μία απόδειξη για μία πρόταση, χωρίς όμως να μπορεί να δείξει ότι δεν υπάρχει άλλη που να την αντικρούει. Σε κάποιες απλές θεωρίες, όπως είναι πχ η στοιχειώδης γεωμετρία, μπορεί να βρεθεί ένας συνολικός αλγόριθμος. Σχεδόν πάντα όμως, ο χρήστης πρέπει να δώσει κάποια στοιχεία στο πρόγραμμα σαν βοήθεια. Αυτόν το ρόλο έχει ένας ελεγκτής απόδειξης, να αποτελέσει ένα εύχρηστο μέσο για να μπορεί να δεχθεί το πρόγραμμα κωδικοποιημένα κάποια βοήθεια από τον χρήστη. Μαζί με το σύστημα ανάπτυξης απόδειξης, θα αποτελέσει όπως αναφέραμε και στην εισαγωγή του κεφαλαίου, το σύστημα του proof assistant.

Τα συστήματα ανάπτυξης πιστοποιημένου κώδικα που έχουν παρουσιαστεί τελευταία, σε σχέση με τα συστήματα λογικής που υποστηρίζουν, αναλύονται στη συνέχεια.

- *Ενδιάμεση γλώσσα προγραμματισμού με Τύπους (TIL), Ενδιάμεση γλώσσα μηχανής με Τύπους (TAL).* Αποτελεί την πρώτη προσπάθεια επέκτασης ενδιάμεσης γλώσσας και γλώσσας μηχανής με ισχυρά συστήματα τύπων για πιστοποίηση κώδικα. Με τη χρήση μίας τέτοιας γλώσσας, μπορούν να αποδοθούν απλές ιδιότητες ασφάλειας και με τη χρήση ενός κατάλληλου ελεγκτή τύπων, να πιστοποιηθούν πριν την τελική εκτέλεση του κώδικα.

- *Το σύστημα Proof-Carrying Code (PCC).* Προτάθηκε από τον Necula και χρησιμοποιεί βασικές τεχνικές της μαθηματικής λογικής και της σημασιολογίας γλωσσών προγραμματισμού. Το PCC απαιτεί κοινή πολιτική ασφαλείας από το χρήστη και τον προγραμματιστή, εκφρασμένη σε κατάλληλη τυπική λογική. Ο προγραμματιστής περνάει τον κώδικα από έναν μεταγλωττιστή, που παράγει πιστοποιημένο κώδικα, ένα σύνολο από τον τελικό κώδικα και την αναπαράσταση της απόδειξης ότι πράγματι αυτός ο κώδικας πληροί τις ζητούμενες προϋποθέσεις ασφαλείας. Ο χρήστης από την πλευρά του απλά διασφαλίζει τον εκτελέσιμο κώδικα που παραλαμβάνει, με βάση την απόδειξη που παίρνει μαζί. Χρησιμοποιεί μία γενικής εφαρμογής κατηγορηματική λογική πρώτης τάξης και μπορεί αν εκφράσει πιο πολύπλοκες ιδιότητες ασφάλειας, σε σχέση με το TAL. Το μειονέκτημα του συστήματος αυτού όπως και του TAL, είναι ότι η πολιτική ασφάλειας πρέπει να εκφραστεί με βάση τη γλώσσα και το σύστημα τύπων του συστήματος. Με άλλα λόγια, οι κανόνες της πολιτικής ασφαλείας, ο ελεγκτής τύπων και ο μεταφραστής στην ενδιάμεση γλώσσα δεν πρέπει να έχουν ελάττωμα.

- *Foundational Proof-Carrying Code* (FPCC), παρουσιάστηκε από τους Appel και Felty προσπαθεί να λύσει το παραπάνω πρόβλημα, χρησιμοποιώντας όσο το δυνατό μικρότερου μεγέθους τμήματα που θεωρούνται πιστοποιημένα χωρίς απόδειξη. Χρησιμοποιεί έναν γενικής χρήσης κατηγορηματικό λογισμό υψηλής τάξης και μερικά αξιώματα από την αριθμητική. Ορίζει με αυτήν τη γλώσσα μία πολιτική ασφάλειας, ένα σύστημα τύπων και σημασιολογία για τη γλώσσα προγραμματισμού. Ο πιστοποιημένος κώδικας είναι ένα σύνολο από τον τελικό κώδικα μαζί με μία αναπαράσταση της απόδειξης ασφαλείας, η οποία πρέπει να είναι καλά ορισμένη, δηλαδή όλες οι ιδιότητες της πολιτικής ασφαλείας να αποδεικνύονται με βάση μαθηματικούς ορισμούς.

Όπως φαίνεται λοιπόν, το FPCC είναι πολύ πιο ευέλικτο από τα TAL και PCC διότι δεν περιορίζεται σε κάποια συγκεκριμένη γλώσσα προγραμματισμού ή κάποιο υποσύνολο λογικής ή μαθηματικών, αλλά αντίθετα οι αποδείξεις ξεκινούν από θεμελιώδεις σχέσεις και φτάνουν μέχρι τον ορισμό νέων συστημάτων τύπων για την τελική γλώσσα ή και νέων ιδιοτήτων για την απόδειξη της ασφάλειας του συστήματος. Το FPCC είναι πιο ασφαλές γιατί δέχεται

μικρότερα τμήματα ως ασφαλή, κάτι που βέβαια κάνει την κατασκευή αποδείξεων πιο δύσκολη και πολύπλοκη.

Σύγκριση proof-assistants

Όλα τα συστήματα που αναλύσαμε λειτουργούν με *πλάνα απόδειξης*, ένα σύνολο από *τακτικές* που χρειάζονται ώστε ο proof assistant να επαληθεύσει την εγκυρότητα μίας πρότασης. Θα χωρίσουμε τα proof assistants σε δύο κατηγορίες. Σε αυτά που λειτουργούν με *αντικείμενα–απόδειξης*, και αυτά που λειτουργούν χωρίς τέτοια αντικείμενα. Για την πρώτη κατηγορία, ένα script παράγει και αποθηκεύει έναν όρο, ο οποίος είναι μία απόδειξη. Η απόδειξη στη συνέχεια ελέγχεται από έναν *ελεγκτή απόδειξης*. Στη δεύτερη περίπτωση, proof assistants χωρίς αντικείμενα απόδειξης, έχουμε δύο υποκατηγορίες. Στην πρώτη υποκατηγορία, το σύστημα μεταφράζει το πλάνο απόδειξης σε αντικείμενο απόδειξης το οποίο στο τέλος περνάει για επαλήθευση και από έναν απλό ελεγκτή. Μπορούμε να πούμε δηλαδή, ότι με αυτή τη διαδικασία, το σύστημα φορμαλοποιεί τα πλάνα απόδειξης σε αντικείμενα απόδειξης, γιαυτό και σε αυτήν την περίπτωση τα πλάνα απόδειξης ονομάζονται και *μη πρότυπα αντικείμενα απόδειξης*. Στη δεύτερη υποκατηγορία, τα συστήματα είναι τέτοια όπου δεν είναι εφικτό να εισαχθούν με αξιόπιστο τρόπο αντικείμενα απόδειξης. Σε αυτή την περίπτωση λοιπόν απλά ο χρήστης εμπιστεύεται την απόφαση του proof assistant, και αυτό που συναντάται είναι ότι αυτά τα συστήματα είναι πολύ εύχρηστα και με μεγάλες δυνατότητες αυτόματης διαδικασίας αναγωγών και υποπεριπτώσεων. Παραδείγματα για όλες τις περιπτώσεις είναι τα συστήματα Coq, Lego, όπου ανήκουν στην πρώτη κατηγορία — δέχονται δηλαδή αντικείμενα απόδειξης στην είσοδο. Τα συστήματα Nuprl, HOL, Isabelle, ανήκουν στην πρώτη υποκατηγορία, όπου τους παρέχονται πλάνα απόδειξης και τα φορμαλοποιούν, ενώ στην τελευταία υποκατηγορία ανήκουν το σύστημα PVS.

1.2 Σύνοψη

Η συνέχεια της εργασίας έχει την ακόλουθη δομή:

Κεφάλαιο 2. Περιγραφή του λ-λογισμού και των παραλλαγών του με τύπους. Αντιστοίχιση συστημάτων τύπων σε συστήματα λογικής.

Κεφάλαιο 3. Ορισμός και περιγραφή της γλώσσας τύπων που χρησιμοποιήθηκε για την καταγραφή θεωριών και αποδείξεων.

Κεφάλαιο 4. Αναλυτική περιγραφή της βιβλιοθήκης θεωριών: βασικοί τύποι, προτασιακή και κατηγορηματική λογική, αριθμητική Peano.

Κεφάλαιο 5. Συμπεράσματα και κατευθύνσεις μελλοντικής έρευνας.

Κεφάλαιο 2

Θεωρία Τύπων και Λογική

2.1 Λάμβδα λογισμός

Σε αυτό το κεφάλαιο, θα γίνει μια πρώτη εισαγωγή στον λ-λογισμό *(ή λ-calculus)*, χωρίς τύπους. Στο κομμάτι αυτό, επίσης, θα παρουσιαστούν λ-λογισμοί με τύπους και "ο κύβος του λ-λογισμού με τύπους" *(Cube of typed λ-calculi)*. Εδώ ανακύπτει η ανάγκη να διαχωρίσουμε τον μαθηματικό τύπο *(formula)* με τον τύπο της λογικής *(Type)* που ασχολούμασταν μέχρι τώρα. Έτσι, θα ακολουθήσουμε την αγγλική ωρολογία, και ο μαθηματικός τύπος θα αναφέρεται ως "φόρμουλα", ενώ ο τύπος της λογικής απλά ως "τύπος".

2.1.1 λ-Λογισμός χωρίς τύπους

Ο λάμβδα λογισμός, αποτελεί ένα πλήρες υπολογιστικό μοντέλο, που αναπτύχθηκε από τον Alonso Church το 1930 και αποτελεί μέρος μιας γενικότερης θεωρίας. Σε αυτό το σημείο, αναφερόμενοι στον λ-λογισμό θα υπονοούμε τον λ-λογισμό χωρίς τύπους, όπως άλλωστε αναπτύχθηκε από τον Church αρχικά.

Εφαρμογή και αφαίρεση

Θα αρχίσουμε λίγο ανάποδα. Ας ορίσουμε πρώτα — χωρίς ακόμη να έχουμε ορίσει και περιγράψει τον λ-λογισμό — τις βασικές λειτουργίες του. Συνοπτικά:

- **Εφαρμογή:** η έκφραση
$$F\ A$$
δηλώνει ότι το στοιχείο F μπορεί να θεωρηθεί αλγόριθμος και το στοιχείο A είσοδος του αλγορίθμου.

- **Αφαίρεση:** Αν $M \equiv M[x]$ είναι μια έκφραση που εξαρτάται από το x, τότε ο συμβολισμός $\lambda x.M[x]$ υποδηλώνει τη συνάρτηση
$$x \mapsto M[x].$$

Για να γίνει πιο αντιληπτό, ας δείξουμε με ένα παράδειγμα πώς συνεργάζονται η εφαρμογή και η αφαίρεση. Έστω, λοιπόν:
$$(\lambda x.\, x^2 + 1)3 = 3^2 + 1 = 10.$$
Δηλαδή, ο συμβολισμός $(\lambda x.\, x^2 + 1)3$ δηλώνει τη συνάρτηση $x \mapsto x^2 + 1$, η οποία εφαρμόζεται στο όρισμα 3 και δίνει $3^2 + 1$, δηλαδή αποτέλεσμα 10. Πιο γενικά, ο συμβολισμός θα είναι ο εξής:
$$(\lambda x.\, M[x])N = M[N]$$
ή θα συναντάται — ειδικά σε ορισμούς και θεωρήματα — και ως εξής:
$$(\lambda x.\, M)N = M\{x \mapsto N\}.$$

Διαφαίνεται λοιπόν, από τις πρώτες εικόνες του λ-λογισμού, τί αρχικά "κάνει" ως υπολογιστικό σύστημα με τις δύο βασικές λειτουργίες. Χρησιμοποιούμε ήδη έναν όρο, τις μεταβλητές — το x στον παραπάνω "μαθηματικό τύπο". Ας δούμε λίγο το ρόλο που έχουν αυτές οι μεταβλητές στον λ-λογισμό. Π.χ. από ποιους όρους εξαρτώνται, πότε μπορώ να εφαρμόσω έναν άλλο όρο σε μια μεταβλητή, κτλ. Αυτός ο διαχωρισμός είναι η διαφορά *δεσμευμένης* και *ελεύθερης* μεταβλητής. Ας δούμε το αντίστοιχο από τα μαθηματικά. Έστω το ολοκλήρωμα $\int_a^b f(x,y)dx$. Είναι προφανές, ότι μία αντικατάσταση $x = 7$, δηλαδή το $\int_a^b f(7,y)d7$ δεν θα είχε κανένα νόημα, αντίθετα η αντικατάσταση $y = 7$, δηλαδή $\int_a^b f(x,7)dx$ θα είχε νόημα. Στο παράδειγμα, η x είναι δεσμευμένη μεταβλητή, ενώ η y ελεύθερη. Αν επανέλθουμε τώρα στο λ-λογισμό, κι εδώ κατά αντιστοιχία, πχ στον όρο $(yx(\lambda x.\, x))$, το y και το εξωτερικό x είναι ελεύθερη μεταβλητή, ενώ το εσωτερικό x είναι δεσμευμένη. Επομένως, *η αφαίρεση δεσμεύει τις ελεύθερες μεταβλητές*. Για παράδειγμα, στην παρακάτω αντικατάσταση, αντικαθιστώ μόνο στις ελεύθερες εμφανίσεις του x :

$$yx(\lambda x.\, x)[x := N] = yN(\lambda x.\, x).$$

Μπορούμε να δούμε τρεις ιδιότητες του λ-λογισμού για τις πιο "πολύπλοκες" εφαρμογές ή αφαιρέσεις, που θα συναντάμε βέβαια πολύ συχνά στη συνέχεια.

1. Ισχύει ο προς τα αριστερά προσεταιρισμός σε επαναλαμβανόμενες εφαρμογές, δηλαδή:

$$F\, M_1\, M_2 \ldots M_n \quad \equiv \quad (\ldots((F\, M_1)M_2) \ldots M_n).$$

2. Ισχύει ο προς τα δεξιά προσεταιρισμός σε επαναλαμβανόμενες αφαιρέσεις, δηλαδή:

$$\lambda x_1 \ldots x_n.\, f(x_1, \ldots, x_n) \quad \equiv \quad \lambda x_1.\, (\lambda x_2.\, (\ldots(\lambda x_n.\, f(x_1, \ldots, x_n))\ldots)).$$

3. Και ο συνδυασμός των δύο παραπάνω δίνει

$$(\lambda x_1 \ldots x_n.\, f(x_1, \ldots, x_n))x_1 \ldots x_n \;=\; f(x_1, \ldots, x_n).$$

Ένα ενδιαφέρον σημείο του λ-λογισμού, που θα μας βοηθήσει επιπλέον να κατανοήσουμε το εκφραστικό του βάθος είναι οι συναρτήσεις πολλών ορισμάτων. Ας θεωρήσουμε τις συναρτήσεις:

$$\begin{aligned} F_x &= \lambda y.\, f(x,y) \\ F &= \lambda x.\, F_x \end{aligned}$$

Τότε έχω

$$(Fx)y = F_x y = f(x,y).$$

ενώ βλέπουμε επίσης ότι με βάση την αρχική σκέψη

$$F = \lambda xy.\, f(x,y)$$

και πάλι έχω τελικά το ίδιο αποτέλεσμα δηλαδή

$$(\lambda xy.\, f(x,y))xy = f(x,y).$$

Ορισμός λ-λογισμού

Ορισμός 2.1 Το σύνολο των λ–όρων *(λ-terms)* που συμβολίζεται με Λ, αποτελείται από ένα άπειρο σύνολο μεταβλητών $V = u, u', u'', \ldots$ πάνω στο οποίο γίνεται — σύμφωνα με τα παραπάνω — εφαρμογή και αφαίρεση, δηλαδή

$$\begin{aligned} x \in V &\Rightarrow x \in \Lambda, \\ M, N \in \Lambda &\Rightarrow (MN) \in \Lambda, \\ M \in \Lambda, x \in V &\Rightarrow (\lambda x M) \in \Lambda. \end{aligned}$$

Ορισμός 2.2 Το σύνολο των ελεύθερων μεταβλητών του M συμβολίζεται με $FV(M)$ και ορίζεται επαγωγικά ως εξής:

$$\begin{aligned} FV(x) &= \{x\}; \\ FV(MN) &= FV(M) \cup FV(N); \\ FV(\lambda x.M) &= FV(M) - \{x\}. \end{aligned}$$

Ο λ-όρος M καλείται κλειστός αν $FV(M) = \oslash$. Το σύνολο των κλειστών λ-όρων συμβολίζεται με Λ^0.

Ορισμός 2.3 Το αποτέλεσμα της αντικατάστασης του N *όπου* x *στο* M, δηλαδή ο συμβολισμός $M\{x \mapsto N\}$ ορίζεται ως εξής :

$$\begin{aligned} x\{x \mapsto N\} &\equiv N; \\ y\{x \mapsto N\} &\equiv y, \text{ για } x \neq y; \\ (PQ)\{x \mapsto N\} &\equiv (P\{x \mapsto N\})(Q\{x \mapsto N\}); \\ (\lambda y.\, P)\{x \mapsto N\} &\equiv \lambda y.\, (P\{x \mapsto N\}), \text{ για } x \neq y; \\ (\lambda x.\, P)\{x \mapsto N\} &\equiv (\lambda x.\, P). \end{aligned}$$

Πρόταση 2.1 (Λήμμα αντικατάστασης) Έστω $M, N, L \in \Lambda$ και $x \neq y, x \notin FV(L)$. Τότε

$$M[x := N][y := L] \equiv M[y := L][x := N[y := L]].$$

Ορισμός 2.4 Το πρωτεύον αξίωμα του λ-λογισμού είναι $(\lambda x.\, M)N = M\{x \mapsto N\}$ για κάθε $M, N \in \Lambda$. Το παραπάνω αξίωμα, καλείται επίσης β-μετατροπή *(βλέπε παρακάτω για τις μετατροπές).*

Ορισμός 2.5 Λογικά αξιώματα και κανόνες του λ-λογισμού είναι οι:

$$\begin{aligned} M &= M; \\ M = N &\Rightarrow N = M; \\ M = N, N = L &\Rightarrow M = L; \\ M = M' &\Rightarrow MZ = M'Z; \\ M = M' &\Rightarrow ZM = ZM'; \\ M = M' &\Rightarrow \lambda x.\, M = \lambda x.\, M'. \end{aligned}$$

Θεώρημα 2.1 (Θεώρημα του Σταθερού σημείου) 1. Για κάθε λ-όρο F, υπάρχει ένας άλλος λ-όρος X τέτοιος ώστε $\lambda \vdash F\,X = X$. [1] Δηλαδή

$$\forall F\, \exists X\, FX = X.$$

2. Υπάρχει ο λεγόμενος *τελεστής σταθερού σημείου:*

$$\mathsf{Y} \equiv \lambda f.\, (\lambda x.\, f(xx))(\lambda x.\, f(xx))$$

τέτοιος ώστε

$$\forall F\ F(\mathsf{Y}F)\, \mathsf{Y}F.$$

[1] Ο συμβολισμός $A \vdash B$ που θα χρησιμοποιείται στο εξής εννοεί ότι το A πλαίσιο υποδεικνύει το B

Μετατροπές α, β και η

Στο σημείο αυτό θα ορίσουμε τυπικά τις τρεις μετατροπές — τις α, β, και η — τις οποίες χρησιμοποιούμε ήδη, στη μέχρι τώρα ανάλυση.

▷ *α-μετατροπή*

$\lambda x.\, M = \lambda y.\, M\,[x := y]$ και συμβολίζω με $\lambda x.\, M \to_\alpha \lambda y.\, M\,[x := y]$, αν $y \notin FV(M)$.

▷ *β-μετατροπή*

$$(\lambda x.\, M)N = M\{x \mapsto N\} \quad \text{και συμβολίζω με} \quad (\lambda x.\, M)N \to_\beta M\{x \mapsto N\}$$

▷ *η-μετατροπή*

$$\lambda x.\, Mx \;=\; M \quad \text{και συμβολίζω με} \quad \lambda x.\, Mx \;\to_\eta\; M$$

Θεώρημα Church-Rosser

Πριν ασχοληθούμε με το θεώρημα Church-Rosser, πρέπει να ορίσουμε και τα εξής:

Ορισμός 2.6 Η δυαδικές σχέσεις $\to_\beta, \twoheadrightarrow_\beta$ και $=_\beta$ στο Λ ορίζονται επαγωγικά ως εξής:

1. (α) $(\lambda x.\, M)N \leftarrow_\beta M\{x \mapsto N\}$;
   (β) $M \leftarrow_\beta N \Rightarrow ZM \leftarrow_\beta ZN, MZ \leftarrow_\beta NZ$ και $\lambda x.\, M \leftarrow_\beta \lambda x.\, N$.
2. (α) $M \twoheadrightarrow_\beta M$;
   (β) $M \leftarrow_\beta N \Rightarrow M \twoheadrightarrow_\beta N$;
   (γ) $M \twoheadrightarrow_\beta N, N \twoheadrightarrow_\beta L \Rightarrow M \twoheadrightarrow_\beta L$.
3. (α) $M \twoheadrightarrow_\beta N \Rightarrow M =_\beta N$;
   (β) $M =_\beta N \Rightarrow N =_\beta M$;
   (γ) $M =_\beta N, N =_\beta L \Rightarrow M =_\beta L$.

Για να κατανοήσουμε την "πρακτική" σημασία των τριών — $\twoheadrightarrow_\beta, \leftarrow_\beta, =_\beta$ — παραπάνω σχέσεων, μπορούμε να σκεφτούμε ότι υπονοούν κάτι τέτοιο:

$M \twoheadrightarrow_\beta N$: M με β-αναγωγή σε N. *(σχέση αναγωγής).*
$M \leftarrow_\beta N$: M με β-αναγωγή σε N, σε ένα βήμα *(σχέση συμβατότητας).*
$M =_\beta N$: M είναι β-μετατρέψιμο σε N. *(ταυτοτική σχέση).*

κάτι που μας δίνει πιο διαισθητικά το περιεχόμενό τους.

Πρόταση 2.2 $M =_\beta N \Leftrightarrow \lambda \vdash M = N$.

Ορισμός 2.7 1. Ένα *β-redex* είναι ένας όρος της μορφής $(\lambda x M)N$. Σε αυτήν την περίπτωση το $M\{x \mapsto N\}$ είναι το contractum του *β-redex*.

2. Ένας λ-όρος M είναι σε *β-κανονική μορφή (ή β-nf)* εάν δεν έχει κάποιο *β-redex* σαν υποέκφραση.

3. Ένας όρος M έχει β-κανονική μορφή αν $M =_\beta N$ και ο N είναι σε β-nf, για κάποιο N.

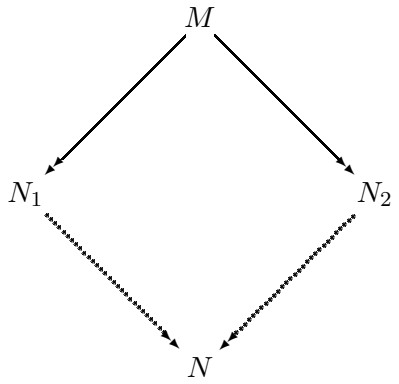
Λήμμα 2.1 Έστω $M, M', N, L \in \lambda$.

1. Έστω ότι ο M είναι σε β-κανονική μορφή. Τότε

$$M \twoheadrightarrow_\beta N \Rightarrow N \equiv M.$$

2. Άν $M \leftarrow_\beta M'$, τότε $M\{x \mapsto N\} \leftarrow_\beta M'\{x \mapsto N\}$.

Θεώρημα 2.2 (Church-Rosser) Έστω ότι ισχύουν οι σχέσεις $M \twoheadrightarrow_\beta N_1, M \twoheadrightarrow_\beta N_2$ για $M, N \in \Lambda$. Τότε θα υπάρχει κάποιο $N_3 \in \Lambda$ τέτοιο ώστε να ισχύει $N_1 \twoheadrightarrow_\beta N_3$ και $N_2 \twoheadrightarrow_\beta N_3$; Η παραπάνω πρόταση φαίνεται και στο παρακάτω διάγραμμα:



Η απόδειξη του θεωρήματος 2.2 είναι αρκετά πολύπλοκη και εκτενής. Πιο χρήσιμο θα ήταν να εξεταστούν τα ακόλουθα πορίσματα

Πόρισμα 2.1 Εάν M,N και L όροι και $M =_\beta N$ τότε το L είναι τέτοιο ώστε $M \twoheadrightarrow_\beta N$ και $N \twoheadrightarrow_\beta M$

Πόρισμα 2.2 1. Εάν ο όρος M έχει τον όρο N σαν β-nf, τότε $M \twoheadrightarrow_\beta N$.

2. Ένας λ-όρος έχει το πολύ έναν β-nf.

και συμπεράσματα.

1. Υπάρχει τρόπος να κωδικοποιηθούν ως λ-όροι οι αληθοτιμές, οι φυσικοί αριθμοί, κ.λπ. κατά τέτοιο τρόπο ώστε αυτή η κωδικοποίηση να είναι συνεπής. Επομένως, ο λ-λογισμός είναι συνεπές σύστημα, πχ $\lambda \vdash \textbf{\textit{true}} \neq \textbf{\textit{false}}$, όπου $\mathsf{true} \equiv \lambda xy.\, x$, $\mathsf{false} \equiv \lambda xy.\, y$.

2. Το $\Omega \equiv (\lambda x.\, xx)(\lambda x.\, xx)$ δεν έχει β-nf.

3. Για την εύρεση ενός β-nf σε έναν λ-όρο, οι υποεκφράσεις του τελευταίου μπορεί να αναχθούν με διάφορους τρόπους, όμως εάν βρεθεί ένας β-nf τότε είναι μοναδικός.

2.1.2 λ-Λογισμός με τύπους à la Church

Τα περισσότερα συστήματα à la Church αποτελούνται από ένα σύνολο τύπων $\mathbb{T}$ και από μία ανάθεση τύπων $\sigma \in \mathbb{T}$ σε όρους $M \in \Lambda_\mathbb{T}$.

Ορισμός 2.8 Έστω $\mathbb{T}$ ένα σύνολο τύπων. Με $\Lambda_\mathbb{T}$ θεωρώ το σύνολο των $\mathbb{T}$-*σημειωμένων λ-όρων — ή αλλιώς ψευδο-όρων* — που ορίζεται ως εξής:

$$\Lambda_\mathbb{T} \;=\; V \,|\, \Lambda_\mathbb{T}\Lambda_\mathbb{T} | \lambda x : \mathbb{T}.\, \Lambda_\mathbb{T}$$

όπου με V συμβολίζω το σύνολο των όρων μεταβλητών. Δηλαδή, μπορούμε να πούμε ότι οι ψευδο-όροι περιέχουν επιπλέον πληροφορία, από αυτήν των όρων, *τον τύπο του όρου*.

Το σύστημα $\lambda\!\rightarrow$-Church

Ορισμός 2.9 Ο λ-λογισμός με τύπους $\lambda\!\rightarrow$-Church ορίζεται ως εξής:

1. Τύποι $\quad \mathbb{T} = \mathbb{V} \mid \mathbb{T}\!\rightarrow\!\mathbb{T}$,
2. Ψευδοόροι $\quad \Lambda_{\mathbb{T}} = V \mid \Lambda_{\mathbb{T}}\Lambda_{\mathbb{T}} \mid \lambda V : \mathbb{T}.\,\Lambda$,
3. Πλαίσιο $\quad \Gamma = x_1 : A_1, ..., x_n : A_n$,
   όπου όλα τα x_i είναι διαφορετικά και όλα τα $A_i \in \mathbb{T}$,
4. Κανόνας συστολής $\quad (\lambda x : A.\,M)N \rightarrow_\beta M[x := N]$,
5. Ανάθεση τύπων $\quad$ Το $\Gamma \vdash M : A$ ορίζεται ως εξής:

   a. Κανόνας εκκίνησης $\quad \dfrac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$,

   b. $\rightarrow$-ελάττωση $\quad \dfrac{\Gamma \vdash M : (A \rightarrow B) \qquad \Gamma \vdash N : A}{\Gamma \vdash (MN) : B}$,

   c. $\rightarrow$-εισαγωγή $\quad \dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A.\,M) : (A \rightarrow B)}$.

Ορισμός 2.10 Το σύνολο των *νόμιμων $\lambda\!\rightarrow$-όρων*, που συμβολίζεται με $\Lambda(\lambda\!\rightarrow)$, ορίζεται ως

$$\Lambda(\lambda\!\rightarrow) = \{M \in \Lambda_{\mathbb{T}} \mid \exists \Gamma, \sigma\ \Gamma \vdash M : \sigma\}.$$

Παράδειγμα 2.1 Μερικά παραδείγματα για να κατανοήσουμε λίγο καλύτερα τον λ-λογισμό με τύπους à la Church.

1. $\vdash (\lambda x : \sigma.\,x) : (\sigma\!\rightarrow\!\sigma)$,
2. $\vdash (\lambda x : \sigma.\,\lambda y : \tau.\,x) : (\sigma\!\rightarrow\!\tau\!\rightarrow\!\sigma)$,
3. $x : \sigma \vdash (\lambda y : \tau.\,x) : (\tau\!\rightarrow\!\sigma)$.

□

Πρόταση 2.3 1. *(Βασικό λήμμα για το σύστημα $\lambda\!\rightarrow$-Church:)* Έστω Γ ένα πλαίσιο.

- Εάν $\Gamma' \supseteq \Gamma$ είναι ένα άλλο πλαίσιο, τότε $\Gamma \vdash M : \sigma \;\Rightarrow\; \Gamma' \vdash M : \sigma$.
- $\Gamma \vdash M : \sigma \;\Rightarrow\; FV(M) \subseteq dom(\Gamma)$. [2]
- $\Gamma \vdash M : \sigma \;\Rightarrow\; \Gamma \restriction FV(M) \vdash M : \sigma$, όπου $\Gamma \restriction FV(M)$ δηλώνει ότι $\{(x : \sigma) \in \Gamma \mid x \in FV(M)\}$.

2. *(Λήμμα παραγωγής για το σύστημα $\lambda\!\rightarrow$-Church:)* Έστω Γ ένα πλαίσιο.

- $\Gamma \vdash x : \sigma \;\Rightarrow\; (x : \sigma) \in \Gamma$
- $\Gamma \vdash MN : \tau \;\Rightarrow\; \exists \sigma\,[\Gamma \vdash M : (\sigma\!\rightarrow\!\tau) \text{ και } \Gamma \vdash N : \sigma]$,
- $\Gamma \vdash M : (\lambda x : \sigma.\,M) : \rho \;\Rightarrow\; \exists \tau[\rho = (\sigma\!\rightarrow\!\tau) \text{ και } \Gamma, x : \sigma \vdash M : \tau]$.

3. *(Υπολογισιμότητα των υποόρων για το σύστημα $\lambda\!\rightarrow$-Church:)* Εάν M έχει ένα τύπο, τότε κάθε υποόρος του M έχει τύπο.

4. *(Λήμμα αντικατάστασης για το σύστημα $\lambda\!\rightarrow$-Church:)*

- $\Gamma \vdash M : \sigma \;\Rightarrow\; \Gamma[a := \tau] \vdash M[a := \tau] : \sigma[a := \tau]$.
- Έστω ότι $\Gamma, x : \sigma \vdash M : \tau$ και $\Gamma \vdash N : \sigma$, τότε $\Gamma \vdash M[x := N] : \tau$.

[2] Με $dom(A)$ συμβολίζεται το πεδίο ορισμού της συνάρτησης A.

Church–Curry

Εκτός από το λ-λογισμό με τύπους à la Church, υπάρχει και η εκδοχή à la Curry. Δεν θα επεκταθούμε σε αυτήν την εκδοχή, αλλά μπορούμε να δείξουμε την βασική διαφορά μεταξύ τους, με το εξής παράδειγμα:

$$\vdash_{Curry} (\lambda x.\, x) : (\sigma \to \sigma),$$

και από την άλλη το:

$$\vdash_{Church} (\lambda x : \sigma.\, x) : (\sigma \to \sigma).$$

Δηλαδή, στην à la Church εκδοχή, ο λ-όρος "εμπλουτίζεται" με τον τύπο της δεσμευμένης μεταβλητής, αυτό που πριν ονομάσαμε ψευδοόρο, σε αντίθεση με την à la Curry εκδοχή, όπου η σύνταξη των λ-όρων είναι η ίδια με αυτή του λ-λογισμού χωρίς τύπους. Χονδρικά το πρόβλημα που προκύπτει σε αυτήν τη δεύτερη εκδοχή είναι το ότι δεν είναι πάντα υπολογίσιμο το πρόβλημα: "αυτός ο όρος έχει αυτόν τον τύπο;".

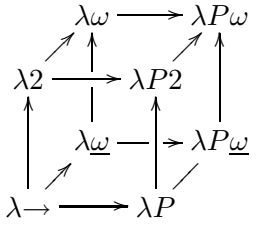
2.1.3 Κύβος λ-λογισμού με τύπους

Ο κύβος του λ-λογισμού είναι ένας κύβος που αποτελείται από οκτώ συστήματα λ-λογισμού με τύπους. Τα εξής:

$$\lambda\to, \lambda 2, \lambda\underline{\omega}, \lambda\omega, \lambda P, \lambda P2, \lambda P\underline{\omega}, \lambda P\omega.$$

Αυτός ο λ-κύβος — αν και θα φανεί η σημασία του περισσότερο στην επόμενη παράγραφο με τα pure type systems — καταφέρνει να δημιουργήσει ένα φυσικό πλαίσιο στο οποίο πολλά γνωστά συστήματα (τύπων) à la Church — όπως τα προαναφερθέντα $\lambda\to, \lambda 2, \lambda\underline{\omega}$ και λP — δίνονται με έναν ομοειδή τρόπο. Με τον λ-κύβο, παρουσιάζεται ένα "ενιαίο" οικοδόμημα του λεγόμενου "λογισμού των κατασκευών", το οποίο είναι το ισχυρότερο σύστημα στον κύβο.

Παρακάτω, ακολουθεί το σχήμα του λ-cube *(κάθε ακμή → περιγράφει τη σχέση ⊆)*:



Ακολουθούν μερικά παραδείγματα από τα συστήματα *(à la Church)* $\lambda\to, \lambda 2, \lambda\omega, \lambda P$ και $\lambda P\omega$ του λ-κύβου. Θα χρησιμοποιήσουμε το συμβολισμό $\Box$ και $*$. Αν και θα φανεί στην επόμενη παράγραφο η χρησιμότητα του παραπάνω συμβολισμού, ουσιαστικά το $*$ αποτελεί ένα σύνολο προτάσεων και υποσυνόλων προτάσεων, ενώ το $\Box$ αποτελεί ένα σύνολο τύπων. Έτσι, μπορούμε στα παραδείγματα που ακολουθούν να εφαρμόσουμε τον παραπάνω συμβολισμό και να αναλύσουμε κάποιες αναγωγές σε αυτά τα συστήματα λ-λογισμού με τύπους.

Παράδειγμα 2.2 ($\lambda\to$) Με βάση το σύστημα $\lambda\to$, μπορούμε να εξάγουμε τα εξής [3]:

$$\begin{aligned}
A : * &\vdash (\Pi x : A.\, A) : *;\\
A : * &\vdash (\lambda a : A.\, a) : (\Pi x : A.\, A);\\
A : *, B : *, b : B &\vdash (\lambda a : A.\, b) : (\Pi x : A.\, B);\\
A : *, b : A &\vdash ((\lambda a : A.\, a) b) : A;\\
A : *, B : *, c : A, b : B &\vdash ((\lambda a : A.\, b) c) : B;\\
A : *, B : * &\vdash (\lambda a : A\, \lambda b : B.\, a) : (A \to (B \to A)) : *.
\end{aligned}$$

[3] Για την τρίτη περίπτωση, προφανώς

$$(\Pi x : A.\, B) \equiv (A \to B)$$

□

Παράδειγμα 2.3 ($\lambda 2$) Με βάση το σύστημα $\lambda 2$, μπορούμε να εξάγουμε τα εξής [4]:

$$\begin{aligned} a:* &\vdash (\lambda a:a.\,a):(a\to a);\\ &\vdash (\lambda a:*\,\lambda a:a.\,a):(\Pi a:*.\,(a\to a)):*;\\ A:* &\vdash (\lambda a:*\,\lambda a:a.\,a)A:(A\to A);\\ A:*, b:A &\vdash (\lambda a:*\,\lambda a:a.\,a)Ab:A. \end{aligned}$$

□

Παράδειγμα 2.4 ($\lambda\omega$) Με βάση το σύστημα $\lambda\omega$, μπορούμε να εξάγουμε τα εξής:

- Έστω $\alpha\&\beta \equiv \Pi\gamma:*.\,(\alpha\to\beta\to\gamma)\to\gamma$, τότε

$$\alpha:*, \beta:* \vdash \alpha\&\beta : *$$

- Έστω $AND \equiv \lambda a:*.\,\lambda b:*.\,a\&b$ και $K \equiv \lambda a:*.\,\lambda b:*.\,\lambda x:a.\,\lambda y:b.\,x$, τότε

$$\begin{aligned} &\vdash AND : (*\to *\to *)\\ &\vdash K : (\Pi a:*.\,\Pi b:*.\,a\to b\to a). \end{aligned}$$

□

Παράδειγμα 2.5 ($\lambda P\omega = \lambda C$) Με βάση το σύστημα $\lambda P\omega$, μπορούμε να εξάγουμε τα εξής:

$$\vdash (\lambda A:*.\,\lambda P:A\to *.\,\lambda a:A.\,Pa\to\bot) : (\Pi A:*.\,(A\to *)\to(A\to *)) : \Box$$

□

2.2 Αμιγή Συστήματα Τύπων

Το ερώτημα που οδήγησε στον ορισμό του PTS [5] είναι το εξής: μπορεί να υπάρξει μία γενική μέθοδος που θα "κατασκευάζει" τα αναγκαία συστήματα λ-λογισμού στον λ-κύβο; Στο προηγούμενο κεφάλαιο, με κάποια παραδείγματα μπορέσαμε να δούμε ότι τα διάφορα συστήματα του λ-κύβου έχουν πολλά κοινά. Με βάση αυτά τα κοινά, θα μπορέσουμε με μία γενική περιγραφή, χωρίς ειδικούς κανόνες να τα περιγράψουμε. Αυτά είναι τα PTS, όπως αναπτύχθηκαν από τους Berardi(1989) και Terlouw(1989). Ας δούμε πώς.

2.2.1 Ορισμός και περιγραφή του PTS

Θα ορίσουμε το PTS για τον λ-κύβο, με βάση ένα σύνολο ψευδο-όρων, το $\mathcal{T}$, το οποίο αναλύθηκε και προηγουμένως. Ας το επαναλάβουμε, σε αφηρημένο συντακτικό:

$$\mathcal{T} = V \mid C \mid \mathcal{T}\mathcal{T} \mid \lambda V:\mathcal{T}\mathcal{T} \mid \Pi V:\mathcal{T}\mathcal{T}$$

όπου V και C είναι μη πεπερασμένες συλλογές (όχι σύνολα) μεταβλητών και σταθερών αντίστοιχα. Διαχωρισμός μεταξύ μεταβλητών-τύπων (type-variables) και μεταβλητών-όρων (term-variables) δε γίνεται. Επίσης, οι έννοιες της β-μετατροπής και β-αναγωγής στο σύνολο $\mathcal{T}$, ορίζονται από τον παρακάτω κανόνα:

[4] Για την τελευταία, αν κάνουμε με βάση τους γνωστούς κανόνες την αναγωγή, φαίνεται ακόμη καλύτερα γιατί καταλήγει η εξαγωγή του τύπου A:

$$(\lambda a:A\,\lambda a:a.\,a)Ab \to (\lambda a:A.\,a)b \to b.(\text{όπου b:A})$$

[5] Στο εξής θα χρησιμοποιείται με τα αρχικά ο όρος αυτός, τα οποία υποδηλώνουν την έννοια Pure Type System — Αμιγές Σύστημα Τύπων. Η χρήση του όρου αμιγές (pure) θα γίνει αντιληπτή αμέσως παρακάτω.

$$(\lambda x : A.\, B)C \to B\{x \mapsto C\}.$$

Τέλος, θα συναντήσουμε στη συνέχεια τα εξής:

- *τη δήλωση:* που θα έχει τη μορφή $A : B$, όπου $A, B \in \mathcal{T}$. Το A ονομάζεται υποκείμενο και το B κατηγόρημα στο $A : B$. Θα συναντάμε δηλαδή τη δήλωση $x : A$, με το $A \in \mathcal{T}$ και το x είναι μία μεταβλητή. Επίσης, ένα *ψευδο-πλαίσιο* είναι μία πεπερασμένη και διατεταγμένη ακολουθία από δηλώσεις, όλες με διακριτά υποκείμενα. Μία κενή τέτοια ακολουθία συμβολίζεται με $\langle\,\rangle$, ενώ εάν $\Gamma = \langle x_1 : A_1, x_2 : A_2, ..., x_n : A_n\rangle$, τότε

  $$\Gamma, x : B = \langle x_1 : A_1, x_2 : A_2, ..., x_n : A_n, x : B\rangle.$$

  αν και συνήθως το κενό δεν γράφεται καν ως $\langle\rangle$, απλά εννοείται.

- *την φόρμουλα:*

  $$\Gamma \vdash A \,:\, B$$

  όπου με βάση τους κανόνες ανάθεσης τύπων (βλ. παρακάτω), υποδηλοί ότι η δήλωση $A : B$ μπορεί να εξαχθεί από την ακολουθία *(ψευδο-πλαίσιο)* Γ. Σε αυτήν την περίπτωση τα A και B καλούνται νόμιμες εκφράσεις και το Γ νόμιμο πλαίσιο.

Ορισμός 2.11 Η προδιαγραφή ενός PTS γίνεται με βάση την τριάδα $S = (\mathcal{S}, \mathcal{A}, \mathcal{R})$, όπου:

1. Το $\mathcal{S}$ είναι ένα υποσύνολο του C, το οποίο καλείται "είδη" *(sorts)*.
2. Το $\mathcal{A}$ είναι ένα σύνολο αξιωμάτων της μορφής

   $$c : s,$$

   όπου $c \in \mathcal{C}$ και $s \in \mathcal{S}$.
3. Το $\mathcal{R}$ είναι το σύνολο των κανόνων, και είναι της μορφής:

   $$(s_1, s_2, s_3),$$

   όπου $s_1, s_2, s_3 \in \mathcal{S}$.

Με βάση την προδιαγραφή $S = (\mathcal{S}, \mathcal{A}, \mathcal{R})$, θα ορίσουμε το σύστημα (PTS), το οποίο συμβολίζεται με

$$\lambda S = \lambda(\mathcal{S}, \mathcal{A}, \mathcal{R}).$$

Έχουμε λοιπόν τα εξής αξιώματα και τους εξής κανόνες [6]:

[6] στο παρακάτω θεωρώ ότι το s παίρνει τιμές από το σύνολο $\mathcal{S}$, το σύνολο των "ειδών" όπως προαναφέρθηκε και ότι το x είναι μεταβλητή.

$\lambda(\mathcal{S},\mathcal{A},\mathcal{R})$

(αξιώματα)	$\langle\rangle \vdash c : s,$	εάν$(c : s) \in \mathcal{A}$;
(κανόνας εκκίνησης)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A},$	εάν $x \equiv {}^s\mathrm{x} \notin \Gamma$;
(κανόνας αποδυνάμωσης)	$\dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}$	εάν $x \equiv {}^s\mathrm{x} \notin \Gamma$;
(παραγωγή)	$\dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.\, B) s_3},$	εάν$(s_1, s_2, s_3) \in \mathcal{R}$;
(εφαρμογή)	$\dfrac{\Gamma \vdash F : (\Pi x : A.\, B) \qquad \Gamma \vdash a : A}{\Gamma \vdash\ Fa\ :\ B\{x \mapsto a\}}$	;
(αφαίρεση)	$\dfrac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (\Pi x : A.\, B) : s}{\Gamma \vdash (\lambda x : A.\, b)\ :\ (\Pi x : A.\, B)}$	;
(μετατροπή)	$\dfrac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s \qquad B =_\beta B'}{\Gamma \vdash A : B'}$	

Στον παραπάνω ορισμό και ειδικά στους κανόνες εκκίνησης και αντικατάστασης, ο συμβολισμός $x \equiv {}^s\mathrm{x} \notin \Gamma$, σημαίνει ότι το x *δεν μπορεί να είναι από τα* x *είδους* s *τα οποία ανήκουν στο πλαίσιο* Γ.

Ορισμός 2.12 1. Ειδικά για τον λ-κύβο που θα ασχοληθούμε, ο κανόνας (s_1, s_2, s_3) θα γράφεται για συντομία (s_1, s_2). Στα παραδείγματα που ακολουθούν θα φανεί γιατί μπορεί να γίνει αυτή η συντόμευση.

2. Τα οκτώ συστήματα του λ-κύβου μπορούν να ορισθούν ως εξής, ως PTS:

Σύστημα λ-λογισμού	Σύνολο ειδικών κανόνων			
$\lambda\!\!\rightarrow$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda P\omega = \lambda C$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

3. Ένα PTS, $\lambda(\mathcal{S},\mathcal{A},\mathcal{R})$, καλείται *πλήρες* εάν

$$\mathcal{R} = \{(s_1, s_2) \mid s_1, s_2 \in \mathcal{S}\},$$

δηλαδή όταν κάθε κανόνας της μορφής (s_1, s_2) του συστήματος, προϋποθέτει ότι τα s_1, s_2 είναι "είδη".

Όπως φαίνεται και από τον παραπάνω ορισμό, αυτό που καταφέρνουμε με την κατασκευή ενός PTS, είναι με τη χρήση ενός αξιώματος και 6 κανόνων, να κατασκευάσουμε τα οκτώ συστήματα του λ-κύβου, χωρίς άλλους ειδικούς κανόνες ή ορισμούς ή συμβολισμούς για κάθε σύστημα διαφορετικά. Κάθε σύστημα θα διαφοροποιηθεί ανάλογα με το τί σύνολα $\mathcal{S},\mathcal{A},\mathcal{R}$ θα χρησιμοποιήσει. Ας δούμε επιλεκτικά για μερικά συστήματα λ-λογισμού με τύπους, που ανήκουν στον λ-κύβο, πώς μπορούν να περιγραφούν ως PTS.

Παράδειγμα 2.6 ($\lambda\rightarrow$) Το σύστημα $\lambda\rightarrow$ του λ-λογισμού είναι το εξής PTS:

$\lambda\rightarrow$

$\mathcal{S}$	$*, \Box$
$\mathcal{A}$	$* : \Box$
$\mathcal{R}$	$(*,*)$

□

Παράδειγμα 2.7 ($\lambda 2$) Το σύστημα $\lambda 2$ του λ-λογισμού είναι το εξής PTS:

$\lambda 2$

$\mathcal{S}$	$*, \Box$
$\mathcal{A}$	$* : \Box$
$\mathcal{R}$	$(*,*), (\Box,*)$

□

Παράδειγμα 2.8 ($\lambda\omega$) Το σύστημα $\lambda\omega$ του λ-λογισμού, είναι το εξής PTS:

$\lambda\omega$

$\mathcal{S}$	$*, \Box$
$\mathcal{A}$	$* : \Box$
$\mathcal{R}$	$(*,*), (\Box,*), (\Box,\Box)$

□

Παράδειγμα 2.9 (λC) Το σύστημα λC του λ-λογισμού — που είναι το ίδιο σύστημα με το $\lambda P\omega$ — είναι το εξής PTS και μάλιστα πλήρες PTS:

λC

$\mathcal{S}$	$*, \Box$
$\mathcal{A}$	$* : \Box$
$\mathcal{R}$	$(*,*), (\Box,*), (*,\Box), (\Box,\Box)$

□

2.2.2 Ιδιότητες ενός αυθαίρετου PTS

Με βάση την πιο αυθαίρετη μορφή PTS που αναλύσαμε, την $\lambda S = \lambda(\mathcal{S}, \mathcal{A}, \mathcal{R})$, μερικές ιδιότητες των PTSs.

Ορισμός 2.13 Έστω Γ ένα ψευδο-πλαίσιο και A ένας ψευδο-όρος [7]:

1. Το Γ καλείται *νόμιμο* πλαίσιο εάν $\exists P, Q \in \mathcal{T}\ \Gamma \vdash P : Q$.
2. Ο όρος A καλείται *Γ-όρος* εάν $\exists B \in \mathcal{T}\ [\Gamma \vdash A : B$ ή $\Gamma \vdash B : A]$.
3. Ο όρος A καλείται *Γ-τύπος* εάν $\exists s \in \mathcal{S}\ [\Gamma \vdash A : s]$.
4. Εάν $\Gamma \vdash A : s$, τότε το A καλείται *Γ-τύπος του "είδους" s*.
5. Ο όρος A καλείται *Γ-στοιχείο* εάν $\exists B \in \mathcal{T}, \exists s \in \mathcal{S}\ [\Gamma \vdash A : B : s]$.
6. Εάν $\Gamma \vdash A : B : s$ τότε το A καλείται *Γ-στοιχείο με τύπο B και του "είδους" s*.

[7] Από εδώ και στο εξής, θεωρώ τις εξής συμβάσεις στους συμβολισμούς:

1. $\Gamma \vdash A : B : C$ υπονοεί το ισοδύναμο $\Gamma \vdash A : B \,\&\, \Gamma \vdash B : C$.
2. Έστω $\triangle \equiv u_1 : B_1, u_2 : B_2, ..., u_n : B_n, n > 0$. Το $\triangle$ είναι ψευδο-πλαίσιο. Τότε, το $\Gamma \vdash \triangle$ σημαίνει $\Gamma \vdash u_1 : B_1 \,\&\, \Gamma \vdash u_2 : B_2 \,\&\, ... \,\&\, \Gamma \vdash u_n : B_n$.

7. Το $A \in \mathcal{T}$ είναι *νόμιμο* εάν $\exists \Gamma, B\, [\Gamma \vdash A : B \ \text{ή} \ \Gamma \vdash B : A]$.

Ορισμός 2.14 Έστω ότι το $\Gamma \equiv x_1 : A_1, ..., x_n : A_n$ και το $\triangle \equiv y_1 : B_1, ..., y_m : B_m$ είναι ψευδο-πλαίσια.

1. Μία δήλωση του τύπου "το x:A είναι *μέσα* στο Γ",συμβολίζεται $(x : A) \in \Gamma$, εάν $x \equiv x_i$ και $A \equiv A_i$ για κάποιο i, $1 \leq i \leq n$.

2. Το Γ είναι "μέρος" του $\triangle$, συμβολίζεται με $\Gamma \subseteq \triangle$, εάν κάθε $(x : A)$ στο Γ είναι επίσης και στο $\triangle$.

3. Έστω $1 \leq i \leq n+1$. Τότε ο *περιορισμός* (restriction) του Γ στο i, συμβολίζεται με $\Gamma \lceil i$, και είναι το $x_1 : A_1, ..., x_{i-1} : A_{i-1}$.

4. Το Γ είναι ένα "αρχικό τμήμα" του $\triangle$ και συμβολίζεται με $\Gamma \leq \triangle$, εάν για κάποιο $j \leq m+1$ ισχύει $\Gamma \equiv \triangle \lceil j$.

Λήμμα 2.2 (Λήμμα ελεύθερης μεταβλητής για τα PTS's) Έστω ότι το $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ είναι νόμιμο ψευδο-πλαίσιο και ότι $\Gamma \vdash B : C$. Τότε ισχύουν τα εξής:

1. Τα $x_1, x_2, ..., x_n$ είναι όλα διακριτά μεταξύ τους.

2. $FV(B), FV(C) \subseteq \{x_1, ..., x_n\}$.

3. $FV(A_i) \subseteq \{x_1, ..., x_{i-1}\}$ για $1 \leq i \leq n$.

Λήμμα 2.3 (Λήμμα εκκίνησης για τα PTS's) Έστω ότι το Γ είναι *νόμιμο* πλαίσιο. Τότε

1. $(c : s)$ είναι ένα αξίωμα $\Rightarrow \quad \Gamma \vdash c : s$;

2. $(x : A) \in \Gamma \quad \Rightarrow \quad \Gamma \vdash x : A$;

Λήμμα 2.4 (Λήμμα μεταβατικότητας για τα PTS's) Έστω Γ και $\triangle$ πλαίσια από τα οποία το Γ είναι νόμιμο. Τότε

$$[\Gamma \vdash \triangle \quad \& \quad \triangle \vdash A : B] \Rightarrow A : B.$$

Λήμμα 2.5 (Λήμμα αντικαταστάσης για τα PTS's) Ας υποθέσουμε ότι

$$\Gamma, x : A, \triangle \vdash B : C$$

και

$$\Gamma \vdash D : A.$$

Τότε

$$\Gamma, \triangle[x := D] \vdash B\{x \mapsto D\} : C\{x \mapsto D\}.$$

Λήμμα 2.6 (Λήμμα εξασθένισης *(thinning lemma)* **για τα PTS's)** Έστω ότι τα Γ και $\triangle$ είναι νόμιμα πλαίσιο τέτοια ώστε $\Gamma \subseteq \triangle$. Τότε

$$\Gamma \vdash A : B \rightarrow \triangle \vdash A : B.$$

Λήμμα 2.7 (Λήμμα παραγωγής για τα PTS's) [8]

[8] Το παρακάτω λήμμα μας παρουσιάζει τον πλέον γενικό τρόπο με τον οποίο η ανάθεση τύπου $\Gamma \vdash A : B$ μπορεί να αποκτηθεί είτε αν το A είναι μεταβλητή, είτε αν είναι σταθερά, μία εφαρμογή, μία λ-αφαίρεση ή μία Π-αφαίρεση.

1. $\Gamma \vdash c : C \quad \Rightarrow \quad \exists s \in \mathcal{S}\ [C =_\beta s \ \ \& \ \ (c : s)]$ είναι αξίωμα.
2. $\Gamma \vdash x : C \quad \Rightarrow \quad \exists s \in \mathcal{S}\ \exists B =_\beta C\ [\Gamma \vdash B : s \ \ \& \ \ (x : B) \in \Gamma$
   $\& \quad x \equiv {}^s x]$.
3. $\Gamma \vdash (\Pi x : A.\, B) : C \quad \Rightarrow \quad \exists (s_1, s_2, s_3) \in \mathcal{R}\,[\Gamma \vdash A : s_1 \ \ \& \ \ \Gamma, x : A \vdash B : s_2$
   $\& \quad c =_\beta s_3]$.
4. $\Gamma \vdash (\lambda x : A.\, b) : C \quad \Rightarrow \quad \exists s \in \mathcal{S}\ \exists B\ [\Gamma \vdash (\Pi x : A.\, B) : s \ \ \& \ \ \Gamma, x : A \vdash b : B$
   $\& \quad C =_\beta (\Pi x : A.\, B)]$.
5. $\Gamma \vdash (Fa) : C \quad \Rightarrow \quad \exists A, B\,[\Gamma \vdash F : (\Pi x : A.\, B) \ \ \& \ \ \Gamma \vdash a : A$
   $\& \quad C =_\beta B\{x \mapsto a\}]$.

Θεώρημα 2.3 (Θεώρημα αναγωγής υποκειμένου για τα PTS's)

$$\Gamma \vdash A : B \ \ \& \ \ A \twoheadrightarrow_\beta A' \quad \Rightarrow \quad \Gamma \vdash A' : B.$$

Λήμμα 2.8 (Λήμμα πύκνωσης *(condensing)* **για τα PTS's)** Σε κάθε PTS ισχύει το εξής [9]:

$$\Gamma, x : A, \triangle \vdash B : C \ \ \& \ \ x \notin \triangle, B, C \quad \Rightarrow \quad \Gamma, \triangle \vdash B : C.$$

2.3 Λογική

Η αντιστοίχιση συστημάτων λ-λογισμού μεταξύ τους δεν είναι η μόνη, αφού μπορεί να υπάρξει αντιστοίχιση και λογικών. Έτσι, ανάλογα με τον λ-κύβο, ορίζουμε τον κύβο της λογικής, έναν κύβο με οκτώ διαφορετικές λογικές στις κορυφές τους.

2.3.1 Κατηγορηματική λογική με πολλά είδη

Στην ανάλυση που ακολουθεί, η κατηγορηματική λογική θα θεωρηθεί στην απλούστερη μορφή της, όπου οι τύποι σχηματίζονται από "άτομα" και με τη χρήση μόνο των λογικών τελεστών $\Rightarrow, \forall$.

Ορισμός 2.15 1. **Δομή με πολλά είδη:** ας περιγράψουμε αυτήν τη δομή με ένα παράδειγμα:

$$\mathcal{A} = \langle A, B, f, g, P, Q, c \rangle,$$

όπου

A, B είναι μη κενά σύνολα (ισοδύναμα τα "είδη" του $\mathcal{A}$ ή "τύποι" του $\mathcal{A}$)
$f : (A \to (A \to A))$ και $g : A \to B$ είναι συναρτήσεις
$P \subseteq A$ και $Q \subseteq A \times B$ είναι σχέσεις
$c \subseteq A$ είναι σταθερά.

2. Η **υπογραφή** του $\mathcal{A}$ είναι η:

$$\langle 2; \langle 1,1,1\rangle, \langle 1,2\rangle; \langle 1\rangle, \langle 1,2\rangle; 1\rangle.$$

Ας την αναλύσουμε λίγο. Αρχικά, με τον πρώτο αριθμό (2) υποδεικνύει ότι το $\mathcal{A}$ έχει δύο "είδη",τα A,B και αντιστοιχεί τον αριθμό 1 στο A και τον αριθμό 2 στο B. Στη συνέχεια, με τα $< 1,1,1 >, < 1,2 >$, υποδηλοί ότι έχουμε δύο συναρτήσεις, τις f,g. Η πρώτη έχει, αντίστοιχα, την υπογραφή $< 1,1,1 >$, όπου με βάση τον όλο συλλογισμό, ο αριθμός 1 αντιστοιχεί στην εμφάνιση της μεταβλητής A, με 2 εισόδους και μία έξοδο $(f : (A \to (A \to A)))$. Ομοίως, η $< 1,2 >$ είναι η υπογραφή της συνάρτησης g,όπου οι εμφανίσεις των αριθμών 1,2 υποδηλώνουν την εμφάνιση των A,B στον τύπο της g

[9] Ο συμβολισμός $x \notin \triangle$ δηλοί ότι το x δεν είναι *ελεύθερο* μέσα στο $\triangle$.

$(g : A \to B)$, δηλαδή είσοδο στοιχείο με τον πρώτο τύπο (A) και έξοδο με το δεύτερο (B). Αν συνεχίσουμε θα δούμε ότι στην υπογραφή αναπαριστούμε με $< 1 >$ την πρώτη σχέση ($P \subseteq A$) και με $< 1, 2 >$ τη δεύτερη ($Q \subseteq A \times B$). Τέλος, η σταθερά c τύπου A, συμπεριλαμβάνεται με την τελευταία εμφάνιση του 1.

Ορισμός 2.16 Δεδομένης μιας λογικής με πολλά είδη $\mathcal{A}$, ορίζω τη γλώσσα $L_{\mathcal{A}}$ της *(ελάχιστης) κατηγορηματικής λογικής με πολλά είδη* πάνω στο $\mathcal{A}$ ως εξής:

1. $L_{\mathcal{A}}$ έχει τα εξής ειδικά σύμβολα:
   - $\boldsymbol{A}, \boldsymbol{B}$ σύμβολα ειδών,
   - $\boldsymbol{f}, \boldsymbol{g}$ σύμβολα συναρτήσεων,
   - $\boldsymbol{P}, \boldsymbol{Q}$ σύμβολα σχέσεων,
   - $\boldsymbol{c}$ σύμβολα σταθεράς.
2. Το σύνολο των μεταβλητών της $L_{\mathcal{A}}$ είναι το:
$$V = \{x^{\boldsymbol{A}} \mid \text{ το } x \text{ μεταβλητή}\} \cup \{x^{\boldsymbol{B}} \mid \text{ το } x \text{ μεταβλητή}\}$$
3. Τα σύνολα των όρων με είδος A και B, συμβολίζονται με $\mathsf{Term}_{\boldsymbol{A}}$ και $\mathsf{Term}_{\boldsymbol{B}}$ αντίστοιχα και ορίζονται ως εξής:
   - $x^{\boldsymbol{A}} \in \mathsf{Term}_{\boldsymbol{A}}$, $x^{\boldsymbol{B}} \in \mathsf{Term}_{\boldsymbol{B}}$,
   - $\boldsymbol{c} \in \mathsf{Term}_{\boldsymbol{A}}$,
   - $s \in \mathsf{Term}_{\boldsymbol{A}}$ και $t \in \mathsf{Term}_{\boldsymbol{A}} \quad \Rightarrow \quad \boldsymbol{f}(s,t) \in \mathsf{Term}_{\boldsymbol{A}}$,
   - $s \in \mathsf{Term}_{\boldsymbol{A}} \quad \Rightarrow \quad g(s) \in \mathsf{Term}_{\boldsymbol{B}}$.
4. Το σύνολο των φορμουλών της $L_{\mathcal{A}}$ συμβολίζεται με Form και ορίζεται ως εξής:
   - $s \in \mathsf{Term}_{\boldsymbol{A}} \quad \Rightarrow \quad \boldsymbol{P}(S) \in \mathsf{Form}$,
   - $s \in \mathsf{Term}_{\boldsymbol{A}}, t \in \mathsf{Term}_{\boldsymbol{B}} \quad \Rightarrow \quad \boldsymbol{Q}(s,t) \in \mathsf{Form}$,
   - $\phi \in \mathsf{Form}, \psi \in \mathsf{Form} \quad \Rightarrow \quad (\phi \to \psi) \in \mathsf{Form}$,
   - $\phi \in \mathsf{Form} \quad \Rightarrow \quad (\forall x^{\boldsymbol{A}} .\phi) \in \mathsf{Form}$ και $(\forall x^{\boldsymbol{B}} .\phi) \in \mathsf{Form}$

Ορισμός 2.17 Έστω $\mathcal{A}$ μία δομή πολλών ειδών. Η *(ελάχιστη) κατηγορηματική λογική πολλών ειδών* — τη συμβολίζω με $PRED = PRED_{\mathcal{A}}$ — θα ορισθεί ως εξής: εάν το Δ είναι ένα σύνολο φορμουλών, αποτελεί δηλαδή ένα πλαίσιο, τότε γράφω $\Delta \vdash \phi$, που σημαίνει ότι το ϕ μπορεί να παραχθεί με βάση τις θεωρήσεις του Δ.

$$\begin{aligned} \phi \in \Gamma &\Rightarrow \Gamma \vdash \phi \\ \Gamma \vdash \phi \to \psi, \Gamma \vdash \phi &\Rightarrow \Gamma \vdash \psi \\ \Gamma, \phi \vdash \psi &\Rightarrow \Gamma \vdash \phi \to \psi \\ \Gamma \vdash \forall x^{\boldsymbol{C}} .\phi, t \in \mathsf{Term}_{\boldsymbol{C}} &\Rightarrow \Gamma \vdash \phi[x := t] \\ \Gamma \vdash \phi, x^{\boldsymbol{C}} \notin FV(\Gamma) &\Rightarrow \Gamma \vdash \forall x^{\boldsymbol{C}} .\phi \end{aligned}$$

2.3.2 Κατηγορηματική λογική υψηλής τάξης

Ορισμός 2.18 Η γλώσσα της **HOL** — Higher Order Logic ή (Μαθηματική) Λογική Υψηλής Τάξης [10] — ορίζεται ως εξής:

[10] Οι όροι κατηγορηματική λογική υψηλής τάξης, higher order predicate logic, higher order logic, HOPL ή HOP είναι ισοδύναμοι. Στα εξής, θα αναφερόμαστε σε αυτούς με έναν μοναδικό όρο, *HOPL*, για αποφυγή σύγχυσης

→ Ένα σύνολο *πεδίων τιμών* , D ορίζεται ως εξής:

$$D ::= B|\Omega|D \to D.$$

όπου B αποτελεί το λεγόμενο *Βασικό Πεδίο* και το Ω αποτελεί το *πεδίο ορισμού των προτάσεων.*

→ Για κάθε $\sigma \in D$, το σύνολο των όρων με *τύπο* σ, δηλαδή το Term_σ, ορίζεται επαγωγικά ως εξής [11]:

1. έχω τις σταθερές: $c_1^\sigma, c_2^\sigma, ... \in \mathrm{Term}_\sigma$,
2. έχω τις μεταβλητές: $c_1^\sigma, c_2^\sigma, ... \in \mathrm{Term}_\sigma$,
3. εάν $\phi : \Omega$ και x^σ είναι μεταβλητή τότε $(\forall x_\sigma . \phi) : \Omega$,
4. εάν $\phi : \Omega$ και $\psi : \Omega$ είναι μεταβλητή τότε $(\phi \Rightarrow \psi) : \Omega$,
5. εάν $M : \sigma \to \tau$ και $N : \sigma$ τότε $(M\ N) : \tau$,
6. εάν $M : \tau$ και x^σ είναι μεταβλητή τότε $(\lambda x^\sigma . M) : \sigma \to \tau$. [12]

→ Το σύνολο των όρων της **HOL**, Term, ορίζεται ως $\mathsf{Term} := \cup_{\sigma \in D} \mathrm{Term}_\sigma$.

→ Το σύνολο των (μαθηματικών) τύπων της **HOL**, form, ορίζεται ως $\mathsf{form} := \mathrm{Term}_\Omega$.

Στο εξής, θα γίνεται αναφορά στην έννοια του πεδίου ορισμού, όπως και στο πεδίο τιμών. Αυτή η αναφορά είναι ίσως λίγο παρακινδυνευμένη, όπως όταν αναφερόμαστε σε τύπους. Έτσι, για έναν όρο που έχει τιμή έναν τύπο A, το πεδίο τιμών αφορά αυτήν ακριβώς την έννοια, του τύπου του, και όχι κάποια άλλη αριθμητική.

Ένα πολύ ενδιαφέρον σημείο στην λογική που εισάγουμε, είναι ότι δεν υπάρχουν πεδία τιμών τύπου καρτεσιανού γινομένου, $D \times D$. Αναπαριστούμε συναρτήσεις μεγαλύτερου arity κάνοντας χρήση του λεγόμενου *Currying*: μία δυαδική συνάρτηση στο D, αναπαρίσταται ως όρος στο πεδίο ορισμού $D \to (D \to D)$. Ένα κατηγόρημα αναπαρίσταται ως συνάρτηση στο Ω, με βάση την ιδέα ότι ένα κατηγόρημα μπορεί να θεωρηθεί συνάρτηση αν παίρνει ως είσοδο μία τιμή και επιστρέφει ένα μαθηματικό τύπο . Κατ' αντιστοιχία, μία δυαδική σχέση πάνω στο D, δεν θα είναι τίποτε άλλο από έναν όρο, με πεδίο ορισμού το $D \to (D \to D)$. Αν το αναλύσουμε αυτό θα δούμε ότι όντως αν

$$R : D \to (D \to \Omega) \ \text{ και } \ t, q : D \quad \text{τότε πράγματι} \quad ((Rt)q) : \Omega.$$

Ας δούμε ακόμη πιο τυπικά αυτήν την γλώσσα, με τους κανόνες παραγωγής της:

[11] Όπως και σε όλην την εργασία, συνεχίζουμε να ακολουθούμε τον συμβολισμό $t : \sigma$ για να δηλώσουμε ότι το t είναι ένας όρος με τύπο σ.

[12] Σε αυτό το σημείο να αναφέρουμε ότι για την HOPL ισχύουν οι γνωστές από πριν έννοιες της δεσμευμένης και ελεύθερης μεταβλητής, της αντικατάστασης, της β-αναγωγής και β-μετατροπής.

HOL

(αξίωμα)	$\dfrac{}{\Gamma \vdash \phi}$	εάν $\phi \in \Gamma$,
(κανόνας εισαγωγής για το $\Rightarrow$)	$\dfrac{\Gamma \cup \phi \vdash \psi}{\Gamma \vdash \phi \Rightarrow \psi}$,	
(κανόνας εξάλειψης για το $\Rightarrow$)	$\dfrac{\Gamma \vdash \phi \qquad \Gamma \vdash \phi \ \Rightarrow \psi}{\Gamma \vdash \psi}$	
(κανόνας εισαγωγής για το$\forall$)	$\dfrac{\Gamma \vdash \phi}{\Gamma \vdash \forall x_\sigma .\phi}$	εάν $x_\sigma \notin FV(\Gamma)$
(κανόνας εξάλειψης για το$\forall$)	$\dfrac{\Gamma \vdash \forall x_\sigma .\phi}{\Gamma \vdash \phi[t/x_\sigma]}$	εάν $t : \sigma$
(μετατροπή)	$\dfrac{\Gamma \vdash \phi}{\Gamma \vdash \psi}$	εάν $\phi =_\beta \psi$

Τώρα μπορούμε να αναλύσουμε μερικά απλά παραδείγματα και να αντιληφθούμε καλύτερα την HOPL. Θα θεωρήσουμε ότι μας δίνονται τα πεδία τιμών $\mathbb{N}$ και A, η σχέση-σταθεράς $>: \mathbb{N} \to \mathbb{N} \to \Omega$, η σχέση-μεταβλητών $R, Q : A \to A \to \Omega$ και οι συναρτήσεις-σταθερές $0 : \mathbb{N}$ και $S : \mathbb{N} \to \mathbb{N}$.

Παράδειγμα 2.10 ► Μπορούμε με την γλώσσα που αναλύουμε, να εκφράσουμε το κατηγόρημα "είναι μεγαλύτερο από το 0" ως όρο: $\lambda x^{\mathbb{N}}. x > 0 : \mathbb{N} \to \Omega$.

► Ας δούμε πώς θα εκφράσουμε την επαγωγή πάνω στο $\mathbb{N}$:

$$\forall P^{\mathbb{N}\to\Omega}.(P0) \ \Rightarrow \ (\forall x^{\mathbb{N}}.(Px \ \Rightarrow \ P(Sx))) \ \Rightarrow \ \forall x^{\mathbb{N}}.Px.$$

Στο εξής, η επαγωγή όπως ορίστηκε παραπάνω θα συμβολίζεται και με τη φόρμουλα ind.

► Η φόρμουλα trans(R), δηλώνει αν η R είναι μεταβατική, δηλαδή trans : $(A \to A \to \Omega) \to \Omega$. Ας την ορίσουμε:

$$\forall x^A.\forall y^A.\forall z^A\,(Rxy \ \Rightarrow \ Ryz \ \Rightarrow \ Rxz).$$

► Μπορούμε να ορίσουμε τον όρο $\subseteq$, όπου $\subseteq: (A \to A \to \Omega) \to (A \to A \to \Omega) \to \Omega$, ως εξής:

$$R \subseteq Q := \forall x^A y^A.(Rxy \ \Rightarrow \ Qxy).$$

► Ας δούμε κάτι πιο πολύπλοκο, το μεταβατικό κλείσιμο της σχέσης R:

$$\lambda x^A.\,\lambda y^A.\,(\forall Q^{A\to A\to\Omega}.\mathsf{trans}(Q) \ \Rightarrow \ (R \subseteq Q) \ \Rightarrow \ Qxy)).$$

► Ας δούμε πώς χρησιμοποιώντας μόνο τους τελεστές $\Rightarrow$ και $\forall$ μπορούμε να ορίσουμε τους συνδέσμους. Έστω $\phi, \psi : \Omega$. Έχω, λοιπόν:

$$\begin{aligned}
\phi\&\psi &:= \forall x^\Omega.(\phi \ \Rightarrow \ \psi \ \Rightarrow \ x) \ \Rightarrow x,\\
\phi \vee \psi &:= \forall x^\Omega.(\phi \ \Rightarrow \ x) \ \Rightarrow \ (\psi \ \Rightarrow \ x) \ \Rightarrow \ x,\\
\bot &:= \forall x^\Omega.x,\\
\neg\phi &:= \phi \Rightarrow \neg,\\
\exists x^\sigma.\phi &:= \forall z^\Omega.(\forall x^\sigma.(\phi \ \Rightarrow \ z)) \ \Rightarrow \ z.
\end{aligned}$$

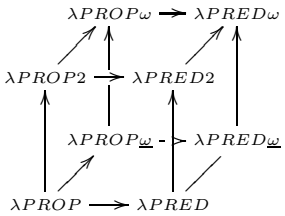
□

2.3.3 Κύβος λογικής

Κατά αναλογία με τον κύβο του λ-λογισμού που αναλύθηκε σε προηγούμενο κεφάλαιο, μπορούμε να φτάσουμε και στον κύβο της λογικής (logic-cube). Αυτή τη φορά στις κορυφές του δε θα βρίσκονται οκτώ συστήματα λ-λογισμού, αλλά οκτώ συστήματα λογικών, τεσσάρων προτασιακών και τεσσάρων κατηγορηματικών.

$PROP$	Προτασιακή Λογική
$PROP2$	Δεύτερης τάξης προτασιακή λογική
$PROP\underline{\omega}$	Αδύναμη προτασιακή λογική υψηλής τάξης
$PROP\omega$	Προτασιακή λογική υψηλής τάξης
$PRED$	Κατηγορηματική Λογική
$PRED2$	Δεύτερης τάξης Κατηγορηματική λογική
$PRED\underline{\omega}$	Αδύναμη Κατηγορηματική λογική υψηλής τάξης
$PRED\omega$	Κατηγορηματική λογική υψηλής τάξης

Όλες οι παραπάνω λογικές θεωρούνται στην ελάχιστη μορφή τους, όπου υπάρχουν μόνο οι λογικοί τελεστές $\rightarrow, \forall$.



Τί είναι όμως, διαισθητικά, ο κύβος της λογικής, κατά αναλογία με τον κύβο του λ-λογισμού; Πρώτα από όλα, να αναφέρουμε ότι κάθε σύστημα λογικής στον κύβο-λογικής, αντιστοιχεί ως προς τη θέση του με το σύστημα λ-λογισμού του λ-κύβου. Έστω ένας τύπος (με τη μαθηματική έννοια, δηλαδή μία λογική φόρμουλα) A, σε μία λογική L_i του κύβου λογικής. Τότε, αυτή ο λογική φόρμουλα μπορεί να μεταφερθεί στο αντίστοιχο σύστημα λ-λογισμού του λ-κύβου ως τύπος $[[A]]$. Αυτή η μετατροπή $(A \mapsto [[A]])$ είναι που θα εξεταστεί αναλυτικά στο επόμενο κεφάλαιο και καλείται προτάσεις ως τύποι (propositions as types) μετατροπή, όπως αρχικά προτάθηκε από τους de Bruijn (1970) και Howard(1980) και στη συνέχεια επεκτάθηκε πλήρως από τον Martin-Löf (1984). Αυτή η PAT μετατροπή εξασφαλίζει το εξής: Αν το A είναι αποδείξιμο στην λογική PRED, τότε το $[[A]]$ χρησιμοποιείται από όρους ως σύνολο τύπων στο λP. Το αντίθετο, έχει αποδειχθεί μέχρι στιγμής μόνο για κάποιες συγκεκριμένες λογικές.

2.4 Προτάσεις ως Τύποι

Σε αυτό το κεφάλαιο θα αναλυθούν αρκετά εκτενώς οι προτάσεις ως τύποι *(Propositions as Types)* Πριν ξεκινήσουμε την ανάλυση των Προτάσεων ως Τύπων, θα εξετάσουμε λίγο πιο διαισθητικά, και λιγότερο τυπικά, τον έλεγχο ή επαλήθευση, απόδειξης, που αποτελούν προβλήματα που βασίζονται στις προτάσεις ως τύπους. Έχει ως βάση μία διαδικασία τυπικής "απεικόνισης" των μαθηματικών στη μηχανή [13], όπου αρχικά προχωράμε σε μία φορμαλοποίηση των βασικών εννοιών — ορισμών, αξιωμάτων και αποδείξεων — και στη συνέχεια — με βάση τη δοσμένη μαθηματική λογική του συγκεκριμένου δοθέντος συστήματος — ελέγχουμε τους ορισμούς για το αν έχουν διατυπωθεί σωστά, όπως και τις αποδείξεις για την ορθότητά τους. Όλη δε η παραπάνω αυτοματοποιημένη διαδικασία, ουσιαστικά επαληθεύει την ζητούμενη μαθηματική θεωρία. Αφού βέβαια επιλεγεί μία μαθηματική λογική όπως προαναφέρθηκε, πρέπει

[13] Είναι πιο σωστό να χρησιμοποιήσουμε τον όρο μηχανή παρά υπολογιστής, αφού ουσιαστικά αναφερόμαστε σε μία γενική μηχανική διαδικασία. Δεν ενδιαφερόμαστε αν έχουμε μπροστά μας έναν προσωπικό υπολογιστή, ένα τερματικό, ή κάτι άλλο. Αναζητούμε κάτι πιο γενικό, π.χ. μία μηχανή Turing, με τη βοήθεια της οποίας θα επεξεργαστούμε τα μοντελοποιημένα μαθηματικά

να κτίσουμε τη μαθηματική θεωρία που αναζητούμε. Σε αυτό το σημείο χρησιμεύει η θεωρία τύπων. Η θεωρία τύπων, παρέχει ένα πολύ δυνατό τυπικό σύστημα που περιλαμβάνει τόσο την έννοια του *υπολογισμού* από τη μεριά του μαθηματικού συστήματος (ανάγοντας προγράμματα σε λ-λογισμό με τύπους), όσο και την έννοια της απόδειξης (με τις προτάσεις ως τύποι, όπου οι τύποι μπορούν να θεωρηθούν προτάσεις και οι όροι ως αποδείξεις). Βέβαια πρέπει να ορισθεί και μία μηχανική διαδικασία με την οποία γίνεται αυτή η μοντελοποίηση της θεωρίας. Αυτή η διαδικασία είναι το λεγόμενο *σύστημα ανάπτυξης απόδειξης — proof-development system* — που ουσιαστικά είναι ένα περιβάλλον που βοηθά τον άνθρωπο να "κατασκευάσει" αποδείξεις. Δίπλα σε αυτό το σύστημα βρίσκεται ο *ελεγκτής απόδειξης — proof checker* — που είναι ένα μικρό πρόγραμμα, το οποίο θα ελέγξει την ορθότητά της θεωρίας. Το σύνολο και των δύο, του ελεγκτή απόδειξης και του συστήματος ανάπτυξης αποδείξεων, αποτελεί το λεγόμενο *βοηθό-απόδειξης — proof-assistant*. Πρέπει να τονίσουμε ότι ένας proof-assistant δεν ελέγχει την εγκυρότητα ενός θεωρήματος, παράγοντάς το μέσω ενός τυφλοσούρτη και ενός συνόλου αλγορίθμων, μηχανικά. Αποτελεί, αντίθετα ένα **βοηθό παραγωγής αποδείξεων σε αλληλεπίδραση με το χρήστη**. [14]

Για να φτάσουμε όμως, στην κατασκευή των παραπάνω προγραμμάτων — proof checker, proof development, proof assistant — πρέπει να ορίσουμε μία μαθηματική λογική, αλλά και να κατανοήσουμε την έννοια των προτάσεων ως τύπους και πώς αυτή την αναπαράσταση μπορούμε να την ενσωματώσουμε στη λογική. Όλα τα παραπάνω βέβαια, με τη χρήση των αμιγών συστημάτων τύπων.

2.4.1 Υψηλής τάξης λ-λογισμός με τύπους

Γενικά

Από αυτό το σημείο θα εισάγουμε τις έννοιες των "προτάσεων ως τύπους" και "αποδείξεων ως όρους", όπως αυτές προτάθηκαν από τους Curry, Howard και de Bruijn. Η εφαρμογή τους στη θεωρία τύπων είναι η εξής: μπορώ με τη χρήση τους **να ερμηνεύσω μία φόρμουλα ως τον τύπο της απόδειξής της**. Αυτό θα το κατασκευάσω με κάποιο σύστημα λ-λογισμού, που θα βοηθήσει για μια τέτοια αναπαράσταση. Ας δούμε λοιπόν αρχικά, περιγραφικά το σύστημα λ-λογισμού που χρειαζόμαστε για αυτό το σκοπό. Στη συνέχεια θα το ορίσουμε και πιο τυπικά:

- ⇝ Θα βασίζεται στους κανόνες της **HOPL** και προφανώς χρειάζεται να είναι ένα σύστημα λ-λογισμού με τύπους. Θα το συμβολίσουμε λοιπόν με λHOPL.

- ⇝ Πρέπει εδώ με κάποιο τρόπο να διαχωρίσουμε την έννοια του *τύπου*, αφού παρουσιάζεται η εξής ασάφεια: Στον λHOPL, φόρμουλες όπως η $\phi \Rightarrow \psi$ ή η $\forall x^A.\phi$, *θα αποτελούν τύπους*, όπως τύπος για μία μεταβλητή είναι και το σύνολο των ακεραίων $\mathbb{N}$. Πρέπει κάπως να διαχωρίσω αυτά τα δύο "είδη" τύπων. Έτσι λοιπόν, για την πρώτη περίπτωση, οι "προτασιακοί τύποι" θα συμβολίζονται με Prop και για τη δεύτερη περίπτωση οι "τύποι συνόλου", δηλαδή $\mathbb{N}, \mathbb{R}$, κτλ, θα συμβολίζονται με Type.

- ⇝ Οι αρχικές υποθέσεις της λογικής, αναπαρίστανται στον λHOPL σαν λ-όροι με τύπους. Αντιστοιχούμε εδώ τις εξής ενέργειες: τη διαδικασία εκκίνησης από μία υπόθεση και την ανάλυσή της με την *λ-αφαίρεση*, ενώ τον κανόνα *modus ponens* σαν τη γνωστή *εφαρμογή* από το λ-λογισμό.

- ⇝ Θα συναντήσουμε τη μορφή
$$\Gamma \vdash M : A,$$
στις προτάσεις της λογικής. Ας την αναλύσουμε λίγο. Το $\mathbf{\Gamma}$ θα είναι ένα πλαίσιο και έχει τη μορφή $x_1 : A_1, x_2 : A_2, ..., x_n : A_n$, όπου τα $x_1, x_2, ..., x_n$ είναι μεταβλητές και τα

[14] Στο εξής, θα χρησιμοποιούμε τους αγγλικούς όρους για τα παραπάνω, δηλαδή proof-cheking, proof-development και proof-assistant.

$A_1, A_2, ..., A_n$ όροι. Τα **M** και **A** της αρχικής φόρμουλας είναι όροι. Αν ο όρος 'A' είναι προτασιακός τύπος (δηλαδή $\Gamma \vdash A : \mathsf{Prop}$), θεωρούμε ότι **το M είναι η απόδειξη του A**. Αλλιώς, αν το A είναι όρος με τύπο συνόλου (δηλαδή $\Gamma \vdash A : \mathsf{Type}$), θεωρούμε ότι **το M είναι ένα στοιχείο του A**.

Ορισμός 2.19 Το σύστημα του λ-λογισμού λHOPL, ο οποίος αναπαριστά κατηγορηματική λογική υψηλής τάξης, ορίζεται ως εξής:

- Το σύνολο των ψευδοόρων του λHOPL είναι το:

$$\mathcal{T} ::= \mathsf{Prop} \mid \mathsf{Type} \mid \mathsf{Type'} \mid \mathcal{V} \mid \Pi\mathcal{V} : \mathcal{T}.\mathcal{T} \mid \mathcal{T}\mathcal{T},$$

όπου $\mathcal{V}$ είναι το σύνολο των μεταβλητών και $\mathcal{S}$ είναι το σύνολο των *ειδών*, δηλαδή {Prop, Type, Type'}.

- Το σύνολο των κανόνων τύπων:

(αξίωμα)	$\mathsf{Prop}:\mathsf{Type}$	$\vdash \mathsf{Type}:\mathsf{Type'}$
(var)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	
(weak)	$\dfrac{\Gamma \vdash A : s \qquad \Gamma \vdash M : C}{\Gamma, x : A \vdash M : C}$	
(Π)	$\dfrac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.B : s_2}$	εάν$(s_1, s_2) \in \{(\mathsf{Type},\mathsf{Type}),$ $(\mathsf{Prop},\mathsf{Prop}),$ $(\mathsf{Type},\mathsf{Prop})\}$
(λ)	$\dfrac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.B : s}{\Gamma \vdash \lambda x : A.M \ : \ \Pi x : A.B}$	
(app)	$\dfrac{\Gamma \vdash M \ : \ \Pi x : A.B \qquad \Gamma \vdash N \ : \ A}{\Gamma \vdash M\,N \ : \ B[N/x]}$	
(conv)	$\dfrac{\Gamma \vdash M \ : \ A \qquad \Gamma \vdash B \ : \ s}{\Gamma \vdash M \ : \ B}$	εάν$A =_\beta B$

Εδώ μπορούμε να εντοπίσουμε αυτό που συχνά αναφέρεται ότι δεν υπάρχει πλέον διαφοροποίηση μεταξύ τύπων και όρων. Όπως φαίνεται και στους παραπάνω κανόνες, πρώτα σχηματίζονται οι τύποι και στη συνέχεια με βάση τους τύπους που έχουν προκύψει, σχηματίζονται οι όροι. Ένας ψευδο-όρος A, θα καλείται *όρος με ορθό τύπο* εφόσον υπάρχει ένα πλαίσιο Γ και ένας ψευδο-όρος B τέτοια ώστε ή το $\Gamma \vdash A : B$ ή το $\Gamma \vdash B : A$ να μπορεί να παραχθεί. Το σύνολο αυτών των *σωστά διατυπωμένων όρων* [15] του λHOPL θα συμβολίζεται

[15] Στο εξής θα χρησιμοποιηθεί ως μετάφραση των αγγλικών όρων *well-typed* — ορθά διατυπωμένο — και *well-formed* — σωστά διαρθρωμένο, αντίστοιχα

με Term(λHOPL). Αντίστοιχα ένα πλαίσιο Γ θα είναι σωστά διαρθρωμένο εάν μπορεί να χρησιμοποιηθεί σε προτάσεις που μπορούν να παραχθούν, πχ εάν υπάρχουν όροι M και A, ένα Γ είναι σωστά διαρθρωμένο εάν το $\Gamma \vdash M : A$ μπορεί να παραχθεί.

Όπως φαίνεται από τα παραπάνω, ο μόνος τελεστής που διαμορφώνει τους τύπους θα είναι το Π, το οποίο είτε εξαρτάται από τον τύπο του πεδίου ορισμού, δηλαδή το A στο $\Pi x : A.\, B$, είτε από τον τύπο του πεδίου τιμών, δηλαδή το B στο $\Pi x : A.\, B$. Έτσι, μπορούμε να δούμε το αποτέλεσμα του τύπου που μας δίνει ο τελεστής Π σαν ένα σύνολο συναρτήσεων. Π.χ. $\Pi x : A.\, B$ είναι ο *τύπος εξαρτημένης συνάρτησης*, των συναρτήσεων που παίρνουν έναν όρο με τύπο A σαν είσοδο και αποδίδει σαν αποτέλεσμα έναν όρο τύπου B, στο οποίο το x έχει αντικατασταθεί από την είσοδο. Αντίθετα, εφόσον $x \notin FV(B)$, γράφουμε $A \to B$ υπονοώντας το $\Pi x : A.\, B$, και ο τύπος είναι αυτός της *μη-εξαρτώμενης συνάρτησης*. Ας δούμε μερικά παραδείγματα.

Παράδειγμα 2.11 1. Χρησιμοποιώντας το συνδυασμό (Type,Type), μπορούμε να κατασκευάσουμε το συναρτησιακό τύπο $A \to B$ όπου A, B : Type. Μπορούμε να φτιάξουμε και πιο σύνθετους τύπος, όπως $A \to$ Prop, $A \to A \to$ Prop, $(A \to A \to \text{Prop}) \to$ Prop.

$\rightsquigarrow$ Εάν $\Gamma \vdash A$: Type και $\Gamma, x : A \vdash B$: Type, τότε στο σύστημα λHOPL ισχύει $x \notin FV(B)$, δηλαδή όσοι τύποι προκύπτουν από το συνδυασμό (Type,Type) είναι τύποι μη-δεσμευμένων συναρτήσεων.

2. Χρησιμοποιώντας το συνδυασμό (Prop,Prop), μπορούμε να σχηματίσουμε τον προτασιακό τύπο $\phi \to \psi$ για ϕ, ψ : Prop. Το τελευταίο θα το καλούμε ως *εμπλεκόμενη φόρμουλα*.

$\rightsquigarrow$ Εάν $\Gamma \vdash \phi$: Prop και $\Gamma, x : \phi \vdash \psi$: Prop, τότε στο σύστημα λHOPL ισχύει $x \notin FV(\psi)$, δηλαδή όσοι τύποι προκύπτουν από το συνδυασμό (Prop,Prop) είναι μη-δεσμευμένοι τύποι.

3. Χρησιμοποιώντας το συνδυασμό (Type,Prop), μπορούμε να σχηματίσουμε τον εξαρτημένο προτασιακό τύπο $\Pi x : A.\, \phi$ για A : Type, ϕ : Prop. Η τελευταία θα καλείται ως *καθολικά ποσοτικοποιημένη φόρμουλα* — universally quantified formula.

$\rightsquigarrow$ Εάν $\Gamma \vdash A$: Type και $\Gamma, x : A \vdash \phi$: Prop τότε είναι πιθανόν να ισχύει $x \in FV(\phi)$ για το λHOPL.

□

Για τα παρακάτω παραδείγματα να υπενθυμίσουμε κάποιες συμβάσεις στους συμβολισμούς, κυρίως ως προς τον τρόπο προσεταιρισμού των μεταβλητών:

- Το Rab υποδηλοί το $((R\, a)\, b)$.
- Το $Rab \to Rbc$ υποδηλοί το $(Rab) \to (Rbc)$
- Το $\lambda x : Rab.\, M$ υποδηλοί το $\lambda x : (Rab).\, M$
- Το $A \to A \to$ Prop υποδηλοί το $A \to (A \to$ Prop$)$.

Ας δούμε λοιπόν το παράδειγμα.

Παράδειγμα 2.12 1. $\mathbb{N}$: Type , $0 : \mathbb{N}$, $>: \mathbb{N} \to \mathbb{N} \to$ Prop $\vdash \lambda x : \mathbb{N}.\, x > 0 : \mathbb{N} \to$ Prop. Εδώ βλέπουμε πώς με τη μέθοδο της λ-αφαίρεσης μπορούμε να ορίσουμε κατηγορήματα.

2. $\mathbb{N}$: Type , $0 : \mathbb{N}$, $S : \mathbb{N} \to \mathbb{N} \vdash \Pi P : \mathbb{N} \to$ Prop$.\, (P0) \to (\Pi x : \mathbb{N}.\, (Px \to P(Sx))) \to \Pi x : \mathbb{N}.\, Px$: Prop. Η παραπάνω φόρμουλα αφορά στην στην αναγωγή, για την λHOPL, και την παρουσιάζει σαν όρο με τύπο Prop.

□

Τέλος, για να ολοκληρώσουμε την περιγραφή του συστήματος λHOPL, όπως και στον HOPL, θα δούμε πώς μπορούμε να ορίσουμε τους κοινούς συνδέσμους $\&, \vee, \bot, \neg, \exists$. Έστω ϕ, ψ : Prop. Έχω λοιπόν:

$$\begin{array}{rcl} \phi \,\&\, \psi & := & \Pi\alpha : \mathsf{Prop}.\, (\phi \to \psi \to \alpha) \to \alpha, \\ \phi \vee \psi & := & \Pi\alpha : \mathsf{Prop}.\, (\phi \to \alpha) \to (\psi \to \alpha) \to \alpha, \\ \bot & := & \Pi\alpha : \mathsf{Prop}.\, \alpha, \\ \neg\phi & := & \phi \to \bot, \\ \exists x : A.\phi & := & \Pi\alpha : \mathsf{Prop}.\, (\Pi x : A.\, (\phi \to \alpha)) \to \alpha. \end{array}$$

Προτάσεις ως τύποι στην HOPL

Η κατά προτάσεις-ως-τύποι μετάφραση από μία υψηλής τάξης κατηγορηματική λογική (**HOPL**) στο σύστημα λογισμού λHOPL *απεικονίζει μία μαθηματική φόρμουλα σε ένα τύπο και μία απόδειξη μίας άλλης μαθηματικής φόρμουλας ϕ σε έναν όρο — πχ έναν λ-όρο με τύπο — που ο τύπος του συσχετίζεται με το ϕ.* Ας δούμε λίγο τί σημαίνει το παραπάνω. Έστω Σ μία απόδειξη μίας φόρμουλας και ψ μία φόρμουλα. Η μετάφραση που ζητείται θα μετατρέψει τη φόρμουλα ψ σε τύπο, που συμβολίζεται με $[\psi]$, και την απόδειξη Σ σε λ-όρο, που συμβολίζεται με $[\![\Sigma]\!]$. Συμβολικά:

$$\begin{array}{c} \boxed{\Sigma} \\ \psi \end{array} \quad \mapsto \quad [\![\Sigma]\!] \,:\, ([\psi])$$

Έλεγχος τύπων

Στα συστήματα τύπων, κάτι που απασχολεί είναι η λεγόμενη *υπολογισιμότητα των τύπων*, δηλαδή αν μας δοθεί ένα πλαίσιο Γ και ένας όρος M, υπολόγισε με κάποιο μηχανικό τρόπο ένα τύπο A, τέτοιο ώστε να ισχύει: $\Gamma \vdash M : A$, αλλιώς αν δεν υπάρχει τέτοιο A, επίστρεψε 'false'. Αυτό το πρόβλημα, καλείται *Πρόβλημα σύνθεσης τύπου* — Type Synthesis Problem (TSP) ή απλά typability. Κατ' επέκταση, ένα δεύτερο πρόβλημα που σχετίζεται με το TSP είναι το TCP — *Type Checking Problem ή απλά type checking, Πρόβλημα ελέγχου του τύπου* [16]. Έχει ως εξής: δοθέντος ενός πλαισίου Γ, ενός όρου M και ενός τύπου A, με μηχανικό τρόπο να αποφασισθεί εάν μπορεί να ισχύει $\Gamma \vdash M : A$. Τα δύο προβλήματα σχετίζονται άμεσα, γιατί για τη λύση του TCP πρώτα πρέπει να λυθεί το TSP. Τέλος, ένα τρίτο πρόβλημα είναι το εξής: δοθέντος ενός τύπου A και ενός πλαισίου Γ, μπορεί να υπάρξει όρος M τέτοιος ώστε $\Gamma \vdash M : A$ *(πρόβλημα inhabitation)*; Ας δούμε δύο παραδείγματα.

Παράδειγμα 2.13 Έστω ότι μας δίνεται το $MN : C$ και πρέπει να καταλήξουμε αν ισχύει. Πρώτα πρέπει να θεωρήσουμε ένα τύπο για το N, έστω A, και ένα τύπο για το M, έστω D. Στη συνέχεια πρέπει να δούμε αν μπορεί ο τύπος του M να είναι "προτασιακός" με μορφή πρακτικά $A \to B$, όπου B ένας ενδιάμεσος τυχαίος τύπος. Αυτή η μορφή για το M, αρκετά πιο αυστηρά είναι η εξής: $D =_\beta \Pi x : A.\, B$. Αυτό το B τώρα, θέλουμε να ξέρουμε ότι με είσοδο με τύπο N, μπορεί να έχει έξοδο με τύπο N. Δηλαδή: $B[N/x] =_\beta C$. Αν θυμηθούμε τον κανόνα εφαρμογής — τον 5ο από τους κανόνες τύπων του λHOPL — θα δούμε ότι με τα παραπάνω αποδεικνύουμε το ζητούμενο. Πρώτα λοιπόν λύσαμε το TSP βρίσκοντας τους αναγκαίους τύπους για τα M, N, και στη συνέχεια το TCP. □

Για να λύσουμε επομένως το TSP, θέλουμε έναν αλγόριθμο — συμβολικά Type_(-) — ο οποίος θα παίρνει ως εισόδους ένα πλαίσιο Γ και έναν όρο M, τέτοιο ώστε:

$$Type_\Gamma(M) =_\beta A \quad \Leftrightarrow \quad \Gamma \vdash M : A.$$

[16] ή συχνά υπολογισημότητα του Τύπου

Ουσιαστικά ο ζητούμενος αλγόριθμος *θα επιλύει μία β-ισότητα*. Αποδεικνύεται, με βάση ακόμη και το λ-λογισμό χωρίς τύπους, ότι *κάθε αναγωγή που ξεκινά από έναν σωστά διατυπωμένο όρο, τερματίζει*. Με άλλα λόγια, αποδεικνύεται ότι η εύρεση επίλυσης της β-ισότητας είναι υπολογίσιμος αλγόριθμος. Με βάση αυτά τα δεδομένα, μπορούμε να ορίσουμε τυπικά, δύο αλγορίθμους, τους OK και Type, ως εξής:

Ορισμός 2.20 Έστω ο αλγόριθμός OK(-) ο οποίος δέχεται ως είσοδο ένα πλαίσιο και ως έξοδο δίνει 'true' ή 'false' — δηλαδή επιλύει το TCP — και ένας αλγόριθμος Type_(-) ο οποίος δέχεται ως είσοδο ένα πλαίσιο και έναν όρο και επιστρέφει έναν άλλο όρο ή 'false' — δηλαδή επιλύει το TSP. Έστω x μία μεταβλητή. Οι παραπάνω αλγόριθμοι έχουν ως εξής:

$$
\begin{array}{rcl}
OK(\langle\rangle) & = & \textbf{true} \text{ , το κενό πλαίσιο} \\
OK(\Gamma, x:A) & = & Type_\Gamma(A) \in \{\text{Prop,Type,Type'}\} \\
Type_\Gamma(x) & = & \text{εάν } OK(\Gamma) \text{ και } x:A \in \Gamma \text{ τότε } A \text{ αλλιώς } \textbf{false} \\
Type_\Gamma(\text{Prop}) & = & \text{εάν } OK(\Gamma) \text{ τότε Type αλλιώς } \textbf{false} \\
Type_\Gamma(\text{Type}) & = & \text{εάν } OK(\Gamma) \text{ τότε Type' αλλιώς } \textbf{false} \\
Type_\Gamma(\text{Type'}) & = & \textbf{false} \\
Type_\Gamma(M\,N) & = & \text{if}(Type_\Gamma(M) = C) \text{ και } (Type_\Gamma(N) = D) \\
 & & \text{then} \\
 & & \quad \text{if } C \twoheadrightarrow_\beta \Pi x:A.\,B \text{ και } A =_\beta D \\
 & & \quad \text{then } B[N/x] \text{ else } \textbf{false} \\
 & & \text{else } \textbf{false} \\
Type_\Gamma(\lambda x:A.\,M) & = & \text{if}(Type_{\Gamma,x:A}(M) = B) \\
 & & \text{then} \\
 & & \quad \text{if } Type_\Gamma(\Pi x:A.\,B) \in \{\text{Prop, Type, Type'}\} \\
 & & \quad \text{then } \Pi x:A.\,B \text{ else } \textbf{false} \\
 & & \text{else } \textbf{false} \\
Type_\Gamma(\Pi x:A.\,B) & = & \text{if}(Type_\Gamma(A) = s_1) \text{ και } (Type_{\Gamma,x:A}(B) = s_2) \\
 & & \qquad \text{και } s_1, s_2 \in \{\text{Prop, Type, Type'}\} \\
 & & \text{then} \\
 & & \quad \text{if } (s_1, s_2) \in \{\text{ (Type,Type),(Type,Prop), (Prop,Prop)}\} \\
 & & \quad \text{then } s_2 \text{ else } \textbf{false} \\
 & & \text{else } \textbf{false}
\end{array}
$$

2.4.2 Ενσωμάτωση της HOPL στον λογισμό των κατασκευών

Να υπενθυμίσουμε ότι τα συστήματα λHOPL και $\lambda PRED_\omega$ σε μορφή PTS είναι αντίστοιχα:

$$
\begin{array}{rcl}
\mathcal{S} & = & \text{Prop, Type, Type'}, \\
\mathcal{A} & = & \text{Prop:Type, Type:Type'}, \\
\mathcal{R} & = & \text{(Type, Type), (Prop, Prop), (Type,Prop).}
\end{array}
$$

και

$$
\begin{array}{rcl}
\mathcal{S} & = & \text{Prop, Set, Type}^p\text{, Type}^s, \\
\mathcal{A} & = & \text{Prop:Type}^p\text{, Set:Type}^s, \\
\mathcal{R} & = & \text{(Set,Set),(Set,Type}^p\text{),(Type}^p\text{,Set),(Type}^p\text{,Type}^p\text{),} \\
 & & \text{(Prop,Prop),(Set,Prop),(Type}^p\text{,Prop).}
\end{array}
$$

όπου Prop είναι το σύνολο των προτάσεων, τα Set και Type^p αποτελούν το σύνολο των πλαισίων — όπου το Set είναι της μορφής $A_1 \to A_2 \to ... \to A_n \to \alpha$, με το α μεταβλητή, και το Type^p είναι της μορφής $A_1 \to A_2 \to ... \to A_n \to \Omega$ όπου το Ω θεωρώ ότι είναι το *βασικό πλαίσιο*, ο τύπος των προτάσεων — και τέλος το Type^p είναι το σύνολο των μεταβλητών με τύπο Set. Επιπλέον, το σύστημα του λογισμού των κατασκευών σε αμιγές σύστημα τύπων είναι το εξής

$$\begin{aligned}\mathcal{S} &= \mathsf{Prop}, \mathsf{Type},\\ \mathcal{A} &= \mathsf{Prop}{:}\mathsf{Type},\\ \mathcal{R} &= (\mathsf{Prop},\mathsf{Prop}),(\mathsf{Prop},\mathsf{Type}),(\mathsf{Type},\mathsf{Type}),(\mathsf{Type},\mathsf{Prop}).\end{aligned}$$

Ορισμός 2.21 Η κατά προτάσεις-ως-τύποι ενσωμάτωση από το σύστημα $\lambda PRED_\omega$ στο λογισμό των κατασκευών, είναι το μόρφωμα ως αμιγές-σύστημα-τύπων H με

$$\begin{aligned}H(\mathsf{Prop}) &= \mathsf{Prop},\\ H(\mathsf{Set}) &= \mathsf{Prop},\\ H(\mathsf{Type}^p) &= \mathsf{Type},\\ H(\mathsf{Type}^s) &= \mathsf{Type}.\end{aligned}$$

Ορισμός 2.22 Ο λ-λογισμός με τύπους που αντιστοιχεί στην υψηλής τάξης προτασιακή λογική ($\lambda PROP\omega$) είναι το αμιγές σύστημα τύπων με

$$\begin{aligned}\mathcal{S} &= \mathsf{Prop}, \mathsf{Type},\\ \mathcal{A} &= \mathsf{Prop}{:}\mathsf{Type},\\ \mathcal{R} &= (\mathsf{Prop},\mathsf{Prop}),(\mathsf{Type},\mathsf{Prop}),(\mathsf{Type},\mathsf{Type}).\end{aligned}$$

Συνέπεια

Ορισμός 2.23 Θα ορίσουμε την απεικόνιση $[\ -\ ] : TERM(CC) \to TERM(\lambda PROP\omega)$, ως εξής:

$$\begin{aligned}[\mathsf{Type}] &= \mathsf{Type},\\ [\mathsf{Prop}] &= \mathsf{Prop},\\ [x] &= x, \text{ όπου } x \text{ μεταβλητή.}\\ [\Pi x : A.\, B] &= [B], \text{ εάν } A : \mathsf{Prop}, B : \mathsf{Type},\\ &= \text{ αλλιώς } \Pi x : [A].\, [B],\\ [\lambda x : A.\, M] &= [M] \text{ εάν } A : \mathsf{Prop}, M : B : \mathsf{Type}, (\text{για κάποιο } B),\\ &= \text{ αλλιώς } \lambda x : [A].\, [B],\\ [PM] &= [P] \text{ εάν } A : \mathsf{Prop}, \Pi : B : \mathsf{Type}, (\text{για κάποια } A, B),\\ &= \text{ αλλιώς } [P][M].\end{aligned}$$

και με βάση την παραπάνω απεικόνιση μπορούμε να δούμε και το εξής:

Πρόταση 2.4 *[Paulin 1989, Berardi 1988]* Αν $\vdash_{CC}$ δηλοί ότι μπορεί να παραχθεί στο λογισμό τον κατασκευών και $\vdash_{\lambda PROP\omega}$ αντίστοιχα ότι μπορεί να παραχθεί στον $\lambda PROP\omega$, τότε έχω το εξής:

$$\Gamma \vdash_{CC} M : A \ \Rightarrow\ [\Gamma] \vdash_{\lambda PROP\omega} [M] : [A]$$

Πόρισμα 2.3 *[Paulin 1989, Berardi 1988]* Ο λογισμός των κατασκευών είναι συντηρητικός στο σύστημα $\lambda PROP\omega$.

Με βάση την παραπάνω απεικόνιση και το παραπάνω πόρισμα, θα χρησιμοποιήσουμε το σύστημα $\lambda PROP\omega$, θα αποδείξουμε ότι είναι συνεπές, και προφανώς αφού από αυτό μπορούμε να ανάξουμε το λογισμό των κατασκευών, θα αποδειχθεί ότι και ο λογισμός των κατασκευών είναι συνεπής. Δύο χαρακτηριστικά του συστήματος $\lambda PROP\omega$ που θα χρειαστούμε είναι τα:

- Το σύνολο των τύπων του $\lambda PROP\omega$ θα καλείται K, όπου

$$K ::= \mathsf{Prop} | K \Rightarrow K.$$

- Οι μεταβλητές-προτάσεις δεν μπορούν να είναι υποόροι προτάσεων, δηλαδή

$$\Gamma \vdash M : A : \mathsf{Type} \ \Rightarrow\ \Gamma' \vdash M : A : \mathsf{Type},$$

όπου Γ' αποτελείται από τις δηλώσεις $x : B \in \Gamma$ για τις οποίες $\Gamma \vdash B : \mathsf{Type}$.

Θα χωρίσουμε τις μεταβλητές σε δύο σύνολα, V^{Prop} για τις μεταβλητές-αποδείξεις και V^{Type} για τις μεταβλητές-κατασκευαστές, όπου ρ θα είναι μία αποτίμηση για το πρώτο σύνολο και ξ θα είναι μία αποτίμηση για το δεύτερο, αντίστοιχα. Θα χρησιμοποιήσουμε μόνο τη δεύτερη αποτίμηση για απλότητα στην αποδεικτική διαδικασία, αφού για την πρώτη θα θεωρήσουμε ότι είναι 0. Επίσης, θα χωρίσουμε και τα πλαίσια του $\lambda PROP\omega$, σε Γ_1 που θα περιέχει τις δηλώσεις των μεταλητών-κατασκευαστών και σε Γ_2 που θα περιέχει τις δηλώσεις των μεταβλητών-αποδείξεων. Θα λέμε ότι η αποτίμηση ξ *ικανοποιεί* το πλαίσιο Γ_1 (και θα συμβολίζω με $\xi \models \Gamma_1$, εάν για κάθε $a : A \in \Gamma_1, \xi a$ αποτελεί μέρος της ερμηνείας του συστήματος $\lambda PROP\omega$.

Ορισμός 2.24 Για $\Gamma \vdash M : A$ ορίζουμε μία *ερμηνευτική συνάρτηση* $[\![-]\!] : \mathsf{TERM}(\lambda PROP\omega) \to Sets$ ως εξής:

1. Για τύπους έχω, $[\![\mathsf{Prop}]\!] = 2$ και $[\![k_1 \to k_2]\!] = [\![k_1]\!] \to [\![k_2]\!]$, για $k_1, k_2 \in K$.
2. Για κατασκευαστές, έστω ξ μία αποτίμηση μεταβλητών-κατασκευαστών, τέτοια ώστε $\xi \models \Gamma$. Έχω

$$\begin{array}{rcl} [\![a]\!]_\xi & = & \xi(a), \\ [\![\Pi x : A.\, B]\!]_\xi & = & 1, \text{ εάν } \forall a \in [\![A]\!], [\![B]\!]_{\xi(x:=a)} = 1, \\ & = & \text{ αλλιώς } 0, \text{ για } A : \mathsf{Type}, B : \mathsf{Prop}, \\ [\![A \to B]\!]_\xi & = & [\![A]\!]_\xi \to [\![B]\!]_\xi, \text{ για } A, B : \mathsf{Prop}, \\ [\![PQ]\!]_\xi & = & [\![P]\!]_\xi [\![Q]\!]_\xi, \\ [\![\lambda a : A.\, P]\!]_\xi & = & \mathbf{\lambda} a \in [\![A]\!]_\xi \,.\, [\![P]\!]_{\xi(x:=a)}. \end{array}$$

3. Όλες οι αποδείξεις ερμηνεύονται με 0.

όπου $\mathbf{\lambda}\in U.V(a)$ υποδηλοί συνολοθεωρητική συνάρτηση.

Πρόταση 2.5 Εάν $\Gamma \vdash M : A$ τότε για κάθε αποτίμηση ξ με $\xi \models \Gamma$, ισχύει $[\![M]\!]_\xi \in [\![A]\!]_\xi$.

Πόρισμα 2.4 Το σύστημα $\lambda PROP\omega$ είναι συνεπές, και επομένως και ο λογισμός των κατασκευών είναι συνεπής.

Απόδειξη Για κάθε αποτίμηση ξ ισχύει $[\![\bot]\!]_\xi = 0$. Όλες οι αποτιμήσεις ικανοποιούν το κενό πλαίσιο, επομένως εάν $\vdash M : \bot$, τότε $0 \in 0$. *ο.ε.δ.* □

Μη-πληρότητα

Σε αυτό κομμάτι, θα αποδείξουμε ότι η ενσωμάτωση κατηγορηματικής λογικής υψηλής τάξης στο λογισμό των κατασκευών σαν προτάσεις-ως-τύποι είναι μη πλήρες πρόβλημα. Επιπλέον, θα δούμε ότι είναι δυνατόν να υπάρχουν προβλήματα που, ενώ στη λογική δεν είναι δυνατόν να αποδειχθούν, όταν μεταφερθούν στο λογισμό των κατασκευών, μπορούν.

Πρόταση 2.6 Η ενσωμάτωση κατηγορηματικής λογικής υψηλής τάξης στο λογισμό των κατασκευών σαν προτάσεις-ως-τύποι είναι μη πλήρες πρόβλημα.

Απόδειξη [Geuvers 1989] Θεωρώ δεδομένο ότι ένα $x \notin FV(\phi)$, τότε οι προτάσεις $\forall x : A\,.\phi$ και $A \to \phi$ δεν μπορούν να διαχωριστούν στο λογισμό των κατασκευών. Ας θεωρήσουμε λοιπόν τη πλαίσιο

$$\Gamma \;:=\; A : \mathsf{Set}, a : A, \phi : \mathsf{Prop}, \alpha : \mathsf{Prop} \to \mathsf{Prop}, z : P(\forall x : A\,.\phi).$$

Αν προσπαθήσουμε να βρούμε την απόδειξη t για το αν ισχύει $\exists \beta : \mathsf{Prop}.P(\beta \to \phi)$, θα αποτύχουμε γιατί τέτοιο t δεν υπάρχει. Στον λογισμό των κατασκευών όμως, επειδή τα πλαίσια και η προτάσεις δεν διαχωρίζονται, μπορούμε να πάρουμε ένα πλαίσιο A στη θέση του

β. Δηλαδή, με βάση το Γ πλαίσιο, έστω Γ' το πλαίσιο του λογισμού των κατασκευών, $\Gamma' = A : \mathsf{Prop}, a : A, \phi : \mathsf{Prop}, \alpha : \mathsf{Prop} \to \mathsf{Prop}, z : P(\Pi x : A.\, \phi)$, τέτοιο ώστε τελικά:

$$\Gamma' \vdash \lambda\gamma : \mathsf{Prop}.\, \lambda h : (\Pi\beta : \mathsf{Prop}.\, P(\gamma \to \phi) \to \beta).\, hAz \;:\; \exists\beta : \mathsf{Prop}.P(\beta \to \phi).$$

□

Λήμμα 2.9 Στη λογική λHOPL, για $x \notin FV(\phi)$,

$$P : A \to \mathsf{Prop}, \phi : \mathsf{Prop} \;\not\vDash ((\exists x : A.Px) \Rightarrow (\forall x : A.\phi) \Rightarrow \phi),$$

αλλά στο λογισμό των κατασκευών, μπορεί να υπάρξει όρος M τέτοιος ώστε:

$$A : \mathsf{Prop}, P : A \to \mathsf{Prop}, \phi : \mathsf{Prop} \;\vdash\; M \;:\; (\exists x : A.Px) \to (A \to \phi) \to \phi).$$

Απόδειξη Στο πλαίσιο της πρώτης περίπτωσης, για τη λογική λHOPL δηλαδή, δεν έχω ορίσει μεταβλητή με τύπο A, επομένως δεν μπορώ να κατασκευάσω έναν όρο με τύπο A. Επομένως, δεν μπορώ να έχω και απόδειξη. Αντίθετα, στο λογισμό των κατασκευών, έστω ο όρος $M \equiv \lambda z : (\exists x : A.Px).\, \lambda y : (A \to \phi).\, y(px)$, όπου

$$p \;:\; (\exists x : A.\phi) \to A \qquad \text{και} \qquad p :\equiv \lambda z : (\exists x : A.\phi).\, zA(\lambda x : A.\, \lambda y : \phi.\, x).$$

Δηλαδή εάν μπορώ να αποδείξω ότι ισχύει το $\exists x : A.\phi$, τότε μπορώ να κατασκευάσω όρο με τύπο A, με την εφαρμογή του p.

□

Κεφάλαιο 3

Η γλώσσα των τύπων

3.1 Ορισμός

Σε BNF μορφή, ο ορισμός της γλώσσας τύπων είναι ο εξής:

$$
\begin{array}{lcl}
s & ::= & \mathsf{Set} \mid \mathsf{Type} \mid \mathsf{Ext} \\
X & ::= & z \mid k \mid t \\
A, B & ::= & s \mid X \mid \Pi X{:}A.\, B \mid \lambda X{:}A.\, B \mid A\, B \\
 & \mid & \mathsf{Ind}(X{:}A)\{\underline{A}\} \mid \mathsf{Constr}(n, A) \mid \mathsf{Elim}[A'](A{:}B\,\vec{B})\{\underline{A}\}
\end{array}
$$

Όπως θα αναλύθηκε και στο κεφάλαιο 2, η γλώσσα τύπων δεν κάνει συντακτική διάκριση μεταξύ των όρων και των τύπων. Η γλώσσα ορίζει τρία είδη (Sorts): Set, Type - τα οποία αντίστοιχούν στα $*$ και $\Box$ - και Ext. Πιο αναλυτικα, και με βάση τη θεωρία του λ-λογισμού, Set είναι ο τύπος των τύπων, Type ο τύπος των πολυμορφικών τύπων και Ext ο τύπος του Type.

Το σύνολο των ελεύθερων μεταβλητών ενός όρου A συμβολίζεται με $\mathsf{FV}(A)$ και ορίζεται ως εξής:

$$
\begin{array}{lcl}
\mathsf{FV}(s) & = & \emptyset \\
\mathsf{FV}(X) & = & \{\, X \,\} \\
\mathsf{FV}(\Pi X{:}A.\, B) & = & \mathsf{FV}(A) \cup (\mathsf{FV}(B) - \{\, X \,\}) \\
\mathsf{FV}(\lambda X{:}A.\, B) & = & \mathsf{FV}(A) \cup (\mathsf{FV}(B) - \{\, X \,\}) \\
\mathsf{FV}(A\, B) & = & \mathsf{FV}(A) \cup \mathsf{FV}(B) \\
\mathsf{FV}(\mathsf{Ind}(X{:}A)\{\underline{A}\}) & = & \mathsf{FV}(A) \cup (\mathsf{FV}(\underline{A}) - \{\, X \,\}) \\
\mathsf{FV}(\mathsf{Constr}(n, A)) & = & \mathsf{FV}(\vec{A}) \\
\mathsf{FV}(\mathsf{Elim}[A'](A{:}B\,\vec{B})\{\underline{A}\}) & = & \mathsf{FV}(A') \cup \mathsf{FV}(A) \cup \mathsf{FV}(B) \cup \mathsf{FV}(\vec{B}) \cup \mathsf{FV}(\underline{A}) \\
\mathsf{FV}(\underline{A}) & = & \cup_i \mathsf{FV}(A_i)
\end{array}
$$

Επίσης, ορίζω τους εξής συμβολισμούς:

$$
\begin{array}{lcll}
A \to B & \equiv & \Pi X{:}A.\, B & \text{αν } X \notin \mathsf{FV}(B) \\
B\,\vec{A} & \equiv & B\ A_1 A_2 \ldots A_n & \\
\Pi \vec{X}{:}\vec{A}.\, B & \equiv & \Pi X_1{:}A_1.\, \Pi X_2{:}A_2.\, \ldots \Pi X_n{:}A_n.\, B & \\
\lambda \vec{X}{:}\vec{A}.\, B & \equiv & \lambda X_1{:}A_1.\, \lambda X_2{:}A_2.\, \ldots \lambda X_n{:}A_n.\, B &
\end{array}
$$

ενώ με $\underline{\mathrm{A}}$ συμβολίζουμε την ακολουθία από όρους.

Με βάση τα παραπάνω και τον λογισμό των επαγωγικών κατασκευών που αναλύθηκε στο προηγούμενο κεφάλαιο, τα υπόλοιπα στοιχεία ορίζονται ως εξής:

- $\Pi x{:}A.\, B$ που ορίζει τύπους, εξαρτώμενους από όρους [1]
- $\lambda x{:}A.\, B$ που ορίζει συναρτήσεις

[1] Ως όροι αναφέρονται όλα τα στοιχεία της γλώσσας, δηλαδή και οι τύποι.

- AB που ορίζει την εφαρμογή ενός όρου σε όρο
- $\mathsf{Ind}(X:A)\{\vec{A}\}$ που ορίζει έναν επαγωγικό τύπο
- $\mathsf{Constr}(n, A)$ που ορίζει έναν κατασκευαστή τύπου A
- $\mathsf{Elim}[A'](A:B\,\vec{B})\{\underline{A}\}$ που ορίζει την αναδρομική κλήση στον κατασκευαστή ενός στοιχείου.

Επίσης, ορίζω την αντικατάσταση ως εξής:

$$
\begin{array}{lcl}
s[\underline{Z} \mapsto \underline{R}] & \equiv & s \\
X[\underline{Z} \mapsto \underline{R}] & \equiv & \begin{cases} X & \text{αν } X \neq Z_i, \text{ για κάθε } i \\ R_i & \text{αν } X = Z_i, \text{ για κάποιο } i \end{cases} \\
(\Pi X:A.\,B)[\underline{Z} \mapsto \underline{R}] & \equiv & \begin{cases} \Pi X:A[\underline{Z} \mapsto \underline{R}].\,B[\underline{Z} \mapsto \underline{R}] & \text{, περ. (i)} \\ \Pi Y:A[\underline{Z} \mapsto \underline{R}].\,B[X \mapsto Y][\underline{Z} \mapsto \underline{R}] & \text{, περ. (ii)} \end{cases} \\
(\lambda X:A.\,B)[\underline{Z} \mapsto \underline{R}] & \equiv & \begin{cases} \lambda X:A[\underline{Z} \mapsto \underline{R}].\,B[\underline{Z} \mapsto \underline{R}] & \text{, περ. (i)} \\ \lambda Y:A[\underline{Z} \mapsto \underline{R}].\,B[X \mapsto Y][\underline{Z} \mapsto \underline{R}] & \text{, περ. (ii)} \end{cases} \\
(A\,B)[\underline{Z} \mapsto \underline{R}] & \equiv & A[\underline{Z} \mapsto \underline{R}]\,B[\underline{Z} \mapsto \underline{R}] \\
(\mathsf{Ind}(X:A)\{\underline{A}\})[\underline{Z} \mapsto \underline{R}] & \equiv & \begin{cases} \mathsf{Ind}(X:A[\underline{Z} \mapsto \underline{R}])\{\underline{A}[\underline{Z} \mapsto \underline{R}]\} & \text{, περ. (i)} \\ \mathsf{Ind}(Y:A[\underline{Z} \mapsto \underline{R}])\{\underline{A}[X \mapsto Y][\underline{Z} \mapsto \underline{R}]\} & \text{, περ. (ii)} \end{cases} \\
(\mathsf{Constr}(n, A))[\underline{Z} \mapsto \underline{R}] & \equiv & \mathsf{Constr}(n, A[\underline{Z} \mapsto \underline{R}]) \\
(\mathsf{Elim}[A'](A:B\,\underline{B})\{\underline{A}\})[\underline{Z} \mapsto \underline{R}] & \equiv & \mathsf{Elim}[A'[\underline{Z} \mapsto \underline{R}]](A[\underline{Z} \mapsto \underline{R}]:B[\underline{Z} \mapsto \underline{R}]\,\underline{B}[\underline{Z} \mapsto \underline{R}]) \\
 & & \quad \{\underline{A}[\underline{Z} \mapsto \underline{R}]\}
\end{array}
$$

ενώ οι δύο περιπτώσεις που διαχωρίζονται παραπάνω είναι οι

$$
\begin{array}{ll}
\text{περίπτωση (i)} & X \neq Z_i, \text{ για κάθε } i, \text{ και } X \notin \mathsf{FV}(\underline{R}) \\
\text{περίπτωση (ii)} & X = Z_i, \text{ για κάποιο } i, \text{ ή } X \in \mathsf{FV}(\underline{R})
\end{array}
$$

και Y είναι μια καινούργια μεταβλητή, που θεωρούμε οτι δεν συναντάται σε κανέναν όρο.

3.2 Λειτουργική σημασιολογία

Σκοπός μας είναι η γλώσσα τύπων να είναι εφαρμόσιμη, επομένως πρέπει να αναλύσουμε και να ορίσουμε τις προϋποθέσεις κάτω από τις οποίες γίνεται αποδεκτή μία σχέση μεταξύ όρων, ώστε τελικά να μπορούμε να αποτιμήσουμε όρους της γλώσσας σε απλούστερες εκφράσεις. Επιπλέον, θα ορίσουμε τις β, η και ι αναγωγές, όπως και τη σχέση ισότητας μεταξύ των όρων.

3.2.1 Συμβατές σχέσεις

Υποθέτω μία σχέση $\sim$ μεταξύ όρων της γλώσσας. Κατ' επέκταση ορίζουμε τη σχέση $\sim^*$ μέσα σε ένα σύνολο όρων, ως την ελάχιστη που ικανοποιεί για κάθε όρο A , A' και σύνολα όρων $\underline{B}$, $\underline{B}'$, τα παρακάτω:

$$
\begin{array}{lcl}
A \sim A' & \Rightarrow & A;\underline{B} \sim^* A';\underline{B} \\
\underline{B} \sim \underline{B}' & \Rightarrow & A;\underline{B} \sim^* A;\underline{B}'
\end{array}
$$

Για να είναι αποδεκτή στη γλώσσα τύπων μία σχέση $\sim$, αν X μεταβλητές, A, A', B, B', Q, Q' όροι και $\underline{A}$, $\underline{B}$ σύνολα από όρους, πρέπει:

$$
\begin{array}{lcl}
A \sim A' & \Rightarrow & \Pi X{:}A.\, B \sim \Pi X{:}A'.\, B \\
B \sim B' & \Rightarrow & \Pi X{:}A.\, B \sim \Pi X{:}A.\, B' \\
A \sim A' & \Rightarrow & \lambda X{:}A.\, B \sim \Pi X{:}A'.\, B \\
B \sim B' & \Rightarrow & \lambda X{:}A.\, B \sim \Pi X{:}A.\, B' \\
A \sim A' & \Rightarrow & A\, B \sim A'\, B \\
B \sim B' & \Rightarrow & A\, B \sim A\, B' \\
A \sim A' & \Rightarrow & \mathsf{Ind}(X{:}A)\{\underline{A}\} \sim \mathsf{Ind}(X{:}A')\{\underline{A}\} \\
\vec{A} \sim^* \vec{A}' & \Rightarrow & \mathsf{Ind}(X{:}A)\{\underline{A}\} \sim \mathsf{Ind}(X{:}A)\{\underline{A}'\} \\
A \sim A' & \Rightarrow & \mathsf{Constr}(i, A) \sim \mathsf{Constr}(i, A') \\
Q \sim Q' & \Rightarrow & \mathsf{Elim}[Q](A{:}B\,\vec{B})\{\underline{A}\} \sim \mathsf{Elim}[Q'](A{:}B\,\vec{B})\{\underline{A}\} \\
A \sim A' & \Rightarrow & \mathsf{Elim}[Q](A{:}B\,\vec{B})\{\underline{A}\} \sim \mathsf{Elim}[Q](A'{:}B\,\vec{B})\{\underline{A}\} \\
B \sim B' & \Rightarrow & \mathsf{Elim}[Q](A{:}B\,\vec{B})\{\underline{A}\} \sim \mathsf{Elim}[Q](A{:}B'\,\vec{B})\{\underline{A}\}
\end{array}
$$

3.2.2 Αναγωγές

Οι β, η και ι αναγωγές είναι οι μικρότερες συμβατές σχέσεις, τέτοιες ώστε:

$$
\begin{array}{ll}
(\lambda X{:}A.\, B)\, A' & \to_\beta \quad B[X \mapsto A'] \\
\lambda X{:}A.\, B\, X & \to_\eta \quad B \\
\quad (\alpha\nu\ X \notin \mathsf{FV}(B)) & \\
\mathsf{Elim}[A'](\mathsf{Constr}(I, B)\, \vec{A'}{:}B\,\vec{B})\{\underline{A}\} & \to_\iota \\
\qquad \Phi_{X,I}(C_i, A_i, \lambda \vec{X}{:}\vec{B'}.\, \lambda Y{:}B\,\vec{X}.\, \mathsf{Elim}[A'](Y{:}B\,\vec{X})\{\underline{A}\})\, \vec{A'} & \\
\quad (\textit{όπου}\ B = \mathsf{Ind}(X{:}B')\{\underline{C}\}\ \textit{και}\ B' = \Pi \vec{X}{:}\vec{B'}.\, \mathsf{Set}) &
\end{array}
$$

Η συνάρτηση Φ ορίζεται ως:

$$
\begin{array}{lcl}
\Phi_{X,I}(X\, \vec{A'}, A, B) & \equiv & A \\
\Phi_{X,I}(\Pi X'{:}A'.\, B', A, B) & \equiv & \lambda X'{:}A'.\, \Phi_{X,I}(B', A\, X', B) \\
\Phi_{X,I}((\Pi \vec{X'}{:}\vec{A'}.\, X\, \vec{B'}) \to B', A, B) & \equiv & \lambda Y{:}(\Pi \vec{X'}{:}\vec{A'}.\, I\, \vec{B'}). \\
 & & \qquad \Phi_{X,I}(B', A\, Y\, (\lambda \vec{X'}{:}\vec{A'}.\, B\, \vec{B'}\, (Y\, \vec{X'})), B)
\end{array}
$$

Ορίζουμε επίσης τα εξής:

1. Τη σχέση $\twoheadrightarrow_\chi$ ως το ανακλαστικό και μεταβατικό κλείσιμο της σχέσης $\to_\chi$ για $\chi \in \{\beta, \eta, \iota\}$.
2. Το $\twoheadrightarrow_{\beta\eta\iota}$ είναι η ένωση των $\to_\beta$, $\to_\eta$ και $\to_\iota$.
3. Το $\twoheadrightarrow_{\beta\eta\iota}$ είναι το ανακλαστικό και μεταβατικό κλείσιμο της παραπάνω ένωσης.
4. Με τον συμβολισμό $\to$ και $\twoheadrightarrow$ εννοούμε τα $\to_{\beta\eta\iota}$ και $\twoheadrightarrow_{\beta\eta\iota}$ αντίστοιχα.

3.2.3 Ισότητα

Ορίζουμε την σχέση ισότητας $=_\chi$, η οποία προέρχεται από την σχέση $\to_\chi$, όπου $\chi \in \{\beta, \eta, \iota\}$, και τη σχέση $=$ ως την ένωση τριών προηγούμενων.

3.3 Κανόνες τύπων

Σε αυτό το σημείο θα ορίσουμε τους κανόνες τύπων της γλώσσας τύπων. Θα χρησιμοποιήσουμε κάποιες βοηθητικές συναρτήσεις, που ορίζονται παρακάτω, και στη συνέχεια θα αναλύσουμε τις προϋποθέσεις κάτω από τις οποίες ένας όρος της γλώσσας είναι αποδεκτός.

3.3.1 Βοηθητικές συναρτήσεις

- Θετικές εμφανίσεις (Positive Occurrences).

$$\frac{X \neq Y_i \quad X \notin \mathsf{FV}(A_i) \quad X \notin \mathsf{FV}(B_i) \quad \textit{(για κάθε } i\textit{)}}{\mathsf{pos}_X(\Pi \vec{Y} : \vec{A}.\, X\ \vec{B})}$$

- Καλά ορισμένοι κατασκευαστές (Well Founded Constructors).

$$\frac{X \notin \mathsf{FV}(A_i) \quad \textit{(για κάθε } i\textit{)}}{\mathsf{wfc}_X(X\ \vec{A})} \qquad \frac{X \neq X' \quad X \notin \mathsf{FV}(A') \quad \mathsf{wfc}_X(B)}{\mathsf{wfc}_X(\Pi X' : A'.\, B)}$$

$$\frac{\mathsf{pos}_X(A) \quad \mathsf{wfc}_X(B)}{\mathsf{wfc}_X(A \to B)}$$

- Μικρός τύπος κατασκευαστή (Small constructor type)

  Η συνάρτηση small δέχεται ως παράμετρο μία μεταβλητή. Αν ο όρος είναι ακολουθία από γινόμενα - κάθε μεταβλητή του γινομένου πρέπει να έχει τύπο Set - και το σώμα κάθε γινομένου είναι μία ακολουθία από εφαρμογές - η εφαρμογή θα ξεκινάει από την ελεύθερη μεταβλητή — παράμετρο που δώσαμε στην συνάρτηση - τότε η συνάρτηση επιστρέφει αληθή τιμή.

$$\frac{\Delta \vdash A'_i : \mathsf{Set} \quad \textit{(για κάθε } i\textit{)}}{\Delta \vdash \mathsf{small}_X(\Pi \vec{X'} : \vec{A'}.\, X\ \vec{A})}$$

- Συνάρτηση ψ.

  Χρησιμοποιείται για τον έλεγχο της αποσύνθεσης επαγωγικά ορισμένων τύπων.

$$\begin{array}{lcl}
\Psi_{X,I}(X\ \vec{A'}, A, B) & \equiv & A\ \vec{A'}\ B \\
\Psi_{X,I}(\Pi X' : A'.\, B', A, B) & \equiv & \Pi X' : A'.\, \Psi_{X,I}(B', A, B\ X') \\
\Psi_{X,I}((\Pi \vec{X'} : \vec{A'}.\, X\ \vec{B'}) \to B', A, B) & \equiv & \Pi Y : (\Pi \vec{X'} : \vec{A'}.\, I\ \vec{B'}).\, \Pi \vec{X'} : \vec{A'}. \\
 & & \quad A\ \vec{B'}\ (Y\ \vec{X'}) \to \Psi_{X,I}(B', A, B\ Y)
\end{array}$$

- Πολλαπλές εκφράσεις

  Αν και δεν αποτελούν στοιχείο της γλώσσας, είναι μια χρήσιμη συντόμευση.

$$\frac{}{\Delta \vdash \epsilon : (\epsilon : \epsilon)} \qquad \frac{\Delta \vdash A : B \quad \Delta \vdash \vec{A} : (\vec{X} : \vec{B}[X \mapsto A])}{\Delta \vdash A; \vec{A} : (X; \vec{X} : B; \vec{B})}$$

3.3.2 Μεταβλητές και σταθερές

$$\frac{(X : A) \in \Delta}{\Delta \vdash X : A} \qquad \frac{}{\Delta \vdash \mathsf{Set} : \mathsf{Type}} \qquad \frac{}{\Delta \vdash \mathsf{Type} : \mathsf{Ext}}$$

3.3.3 Ισοδυναμία τύπων

$$\frac{\Delta \vdash X : A \quad A = A'}{\Delta \vdash X : A'}$$

3.3.4 Γινόμενο, αφαίρεση και εφαρμογή

$$\frac{\Delta \vdash A : s_1 \quad \Delta, X : A \vdash B : s_2 \quad (s_1, s_2) \in \mathcal{R}}{\Delta \vdash \Pi X{:}A.\, B : s_2}$$

$$\frac{\Delta, X : A \vdash B : B' \quad \Delta \vdash \Pi X{:}A.\, B' : s}{\Delta \vdash \lambda X{:}A.\, B : \Pi X{:}A.\, B'} \qquad \frac{\Delta \vdash A : \Pi X{:}B'.\, A' \quad \Delta \vdash B : B'}{\Delta \vdash A\, B : A'[X \mapsto B]}$$

όπου $\mathcal{R} = \left\{ \begin{array}{lll} (\mathsf{Set}, \mathsf{Set}), & (\mathsf{Type}, \mathsf{Set}), & (\mathsf{Ext}, \mathsf{Set}), \\ (\mathsf{Set}, \mathsf{Type}), & (\mathsf{Type}, \mathsf{Type}), & (\mathsf{Ext}, \mathsf{Type}), \\ (\mathsf{Set}, \mathsf{Ext}), & (\mathsf{Type}, \mathsf{Ext}) & \end{array} \right\}$

3.3.5 Επαγωγικοί ορισμοί.

$$\frac{\begin{array}{cc} \Delta \vdash A : \mathsf{Type} & A = \Pi \vec{X}{:}\vec{B}.\, \mathsf{Set} \\ \Delta, X : A \vdash A_i : \mathsf{Set} & \mathsf{wfc}_X(A_i) \quad \textit{(για κάθε } i\textit{)} \end{array}}{\Delta \vdash \mathsf{Ind}(X{:}A)\{\underline{A}\} : A} \qquad \frac{A = \mathsf{Ind}(X{:}A')\{\underline{A}\} \quad \Delta \vdash A : A'}{\Delta \vdash \mathsf{Constr}(n, A) : A_n[X \mapsto A]}$$

$$\frac{\begin{array}{cc} B = \mathsf{Ind}(X{:}B')\{\underline{A'}\} & B' = \Pi \vec{X}{:}\vec{B'}.\, \mathsf{Set} \\ \Delta \vdash \vec{B} : (\vec{X} : \vec{B'}) \quad \Delta \vdash A : B\,\vec{B} & \Delta \vdash A' : \Pi \vec{X}{:}\vec{B'}.\, B\,\vec{X} \to \mathsf{Set} \\ \Delta \vdash A_i : \Psi_{X,B}(A'_i, A', \mathsf{Constr}(i, B)) & \textit{(για κάθε } i\textit{)} \end{array}}{\Delta \vdash \mathsf{Elim}[A'](A{:}B\,\vec{B})\{\underline{A}\} : A'\,\vec{B}\,A}$$

$$\frac{\begin{array}{cc} B = \mathsf{Ind}(X{:}B')\{\underline{A'}\} & B' = \Pi \vec{X}{:}\vec{B'}.\, \mathsf{Set} \\ \Delta \vdash \vec{B} : (\vec{X} : \vec{B'}) \quad \Delta \vdash A : B\,\vec{B} & \Delta \vdash A' : \Pi \vec{X}{:}\vec{B'}.\, B\,\vec{X} \to \mathsf{Type} \\ \Delta \vdash A_i : \Psi_{X,B}(A'_i, A', \mathsf{Constr}(i, B)) & \Delta, X : B' \vdash \mathsf{small}_X(A'_i) \quad \textit{(για κάθε } i\textit{)} \end{array}}{\Delta \vdash \mathsf{Elim}[A'](A{:}B\,\vec{B})\{\underline{A}\} : A'\,\vec{B}\,A}$$

3.4 Μετα-θεωρία της γλώσσας τύπων

Τα παρακάτω θεωρήματα μπορούν να αποδειχθούν για τη γλώσσα των τύπων. Οι αποδείξεις τους για το λογισμό των επαγωγικών κατασκευών περιέχονται στη διδακτορική διατριβή του Benjamin Werner [Wern94]. Αυτές μπορούν να προσαρμοστούν για τη γλώσσα *nflint*, όπως φαίνεται στην εργασία των Shao *et al.* [Shao02]. Αξίζει να σημειωθεί ότι οι αποδείξεις των παρακάτω πρέπει να γίνουν σε μεγάλο βαθμό από κοινού, καθώς το σύστημα τύπων και οι αναγωγές είναι πολύ στενά συνδεδεμένες, π.χ. μέσω του κανόνα της μετατροπής.

Η απροκρισιμότητα του ελέγχου τύπων και η μοναδικότητα των τύπων είναι απαραίτητες προϋποθέσεις για τον καλό ορισμό του συστήματος τύπων της *nflint*. Προφανώς οι τύποι είναι μοναδικοί ως προς $=_{\alpha\beta\eta}$.

Θεώρημα 1 (Αποκρισιμότητα του Ελέγχου Τύπων) *Υπάρχει αλγόριθμος ο οποίος, δεδομένου ενός όρου A και ενός περιβάλλοντος Δ:*

- *είτε αποφασίζει ότι δεν υπάρχει όρος B τέτοιος ώστε $\Delta \vdash A : B$,*
- *ή επιστρέφει έναν όρο B τέτοιον ώστε $\Delta \vdash A : B$.*

Θεώρημα 2 (Μοναδικότητα των Τύπων) *Αν $\Delta \vdash A : B$ και $\Delta \vdash A : B'$, τότε $B = B'$.*

Τα παρακάτω θεωρήματα αφορούν στις σχέσεις αναγωγής και τις συνδέουν με τον έλεγχο τύπων. Τα θεωρήματα συμβολής και Church-Rosser ισχύουν μόνο για καλά διατυπωμένους όρους.

Θεώρημα 3 (Συμβολή) *Έστω* $\Delta \vdash A : B$. *Αν* $A \twoheadrightarrow A_1$ *και* $A \twoheadrightarrow A_2$, *τότε υπάρχει ένας όρος* A' *τέτοιος ώστε* $A_1 \twoheadrightarrow A'$ *και* $A_2 \twoheadrightarrow A'$.

Θεώρημα 4 (Church-Rosser) *Έστω* $\Delta \vdash A_1 : B$ *και* $\Delta \vdash A_2 : B$. *Αν* $A_1 = A_2$, *τότε υπάρχει ένας όρος* A' *τέτοιος ώστε* $A_1 \twoheadrightarrow A'$ *και* $A_2 \twoheadrightarrow A'$.

Θεώρημα 5 (Ορθότητα των Αναγωγών) *Αν* $\Delta \vdash A : B$ *και* $A \twoheadrightarrow A'$, *τότε* $\Delta \vdash A' : B$.

Ένας όρος A λέγεται *κανονική μορφή* αν δεν υπάρχει όρος A' τέτοιος ώστε $A \to A'$. Ένας όρος A λέγεται *κανονικοποιήσιμος* αν υπάρχει κανονική μορφή B τέτοια ώστε $A \twoheadrightarrow B$. Ένας όρος A λέγεται *ισχυρά κανονικοποιήσιμος* αν είναι κανονικοποιήσιμος και δεν υπάρχει άπειρη ακολουθία $\to$-αναγωγών που να ξεκινάει από αυτόν τον όρο.

Το τελευταίο ενδιαφέρον θεώρημα αποδεικνύει ότι κάθε σειρά αναγωγών που ξεκινά από έναν καλά διατυπωμένο όρο τερματίζεται.

Θεώρημα 6 (Ισχυρή Κανονικοποίηση) *Αν* $\Delta \vdash A : B$, *τότε ο όρος* A *είναι ισχυρά κανονικοποιήσιμος.*

Κεφάλαιο 4

Η βιβλιοθήκη Θεωριών

Σε αυτό το κεφάλαιο, μετά από μία περιγραφή του συστήματος *Coq*, θα γίνει παρουσίαση των βιβλιοθηκών θεωριών, που μεταφέρθηκαν από το *Coq*, στην *nflint*. Στα παραδείγματα που ακολουθούν, γίνεται χρήση διαφορετικών συμβόλων στη θέση των λ και Π, όπως αυτά ορίσθηκαν στα προηγούμενα κεφάλαια. Έτσι, σε αυτό το κεφάλαιο θα χρησιμοποιηθούν τα εξής σύμβολα:

\ : το οποίο ισοδυναμεί με το λ,

: το οποίο ισοδυναμεί με το Π.

4.1 Περιγραφή του συστήματος Coq

Η γλώσσα προδιαγραφής του coq, η γλώσσα του πυρήνα του είναι η *Gallina*. Επιτρέπει την ανάπτυξη μαθηματικών θεωριών και την απόδειξη προδιαγραφών προγραμμάτων. Οι θεωρίες κατασκευάζονται από αξιώματα, υποθέσεις, παραμέτρους, λήμματα, θεωρήματα και ορισμούς σταθερών, συναρτήσεων, κατηγορημάτων και συνόλων. Το συντακτικό των λογικών αντικειμένων που περιλαμβάνονται σε θεωρίες έχει ως εξής:

```
term ::= forall binderlist , term
       |  fun binderlist => term
       |  fix fix_bodies
       |  cofix cofix_bodies
       |  let ident_with_params := term in term
       |  let fix fix_body in term
       |  let cofix cofix_body in term
       |  let ( [name , ... , name] ) [dep_ret_type] := term in term
       |  if term [dep_ret_type] then term else term
       |  term : term
       |  term -> term
       |  term arg ... arg
       |  @ qualid [term ... term]
       |  term % ident
       |  match match_item , ... , match_item [return_type] with
              [[|] equation | ... | equation] end
       |  qualid
       |  sort
       |  num
       |  _

arg  ::=  term
       |  ( ident := term )

binderlist  ::=   name ... name [: term]
       |  binder binderlet ... binderlet
```

```
binder ::=  name
       |    ( name ... name : term )

binderlet ::=  binder
       |    ( name [: term] := term )

name  ::=   ident
       |    _

qualid   ::=   ident
       |    qualid access_ident

sort  ::=   Prop  |  Set  |  Type

ident_with_params    ::=   ident [binderlet ... binderlet] [: term]

fix_bodies  ::=   fix_body
       |    fix_body with fix_body with ... with fix_body for ident
cofix_bodies   ::=   cofix_body
       |    cofix_body with cofix_body with ... with cofix_body for ident

fix_body    ::=   ident binderlet ... binderlet [annotation] [: term] := term
cofix_body  ::=   ident_with_params := term

annotation  ::=   { struct ident }

match_item  ::=   term [as name] [in term]

dep_ret_type   ::=   [as name] return_type

return_type    ::=   return term

equation    ::=   pattern , ... , pattern => term

pattern ::=    qualid pattern ... pattern
        |   pattern as ident
        |   pattern % ident
        |   qualid
        |   _
        |   num
        |   ( pattern , ... , pattern )
```

Επίσης, η γλώσσα σε επίπεδο εντολών που χρησιμοποιείται από το σύστημα αποδείξεων, η *Vernacular*, έχει το εξής συντακτικό:

```
sentence ::=   declaration
         |  definition
         |  inductive
         |  fixpoint
         |  statement [proof]

declaration ::=   declaration_keyword assums .

declaration_keyword  ::=   Axiom | Conjecture
         |  Parameter | Parameters
         |  Variable | Variables
         |  Hypothesis | Hypotheses
```

```
assums ::=  ident ... ident : term
         |  binder ... binder

definition  ::=   Definition ident_with_params := term .
         |  Let ident_with_params := term .

inductive   ::=   Inductive ind_body with ... with ind_body .
         |  CoInductive ind_body with ... with ind_body .

ind_body    ::=   ident [binderlet ... binderlet] : term :=
         [[|] ident_with_params | ... | ident_with_params]

fixpoint    ::=   Fixpoint fix_body with ... with fix_body .
         |  CoFixpoint cofix_body with ... with cofix_body .

statement   ::=   statement_keyword ident [binderlet ... binderlet] : term .

statement_keyword    ::=   Theorem | Lemma | Definition

proof    ::=   Proof . ... Qed .
         |  Proof . ... Defined .
         |  Proof . ... Admitted .
```

4.1.1 Η βιβλιοθήκη του Coq

Η βιβλιοθήκη του Coq έχει την εξής δομή:

- **Η πρωταρχική βιβλιοθήκη.** Περιέχει τις βασικές λογικές έννοιες και τους βασικούς λογικούς τύπους δεδομένων. Αποτελεί τη βάση του συστήματος και είναι διαθέσιμη από τη στιγμή που ξεκινάει να τρέχει το *Coq*.
- **Η βασική βιβλιοθήκη.** Βιβλιοθήκες γενικού σκοπού, περιέχουν διάφορες επεκτάσεις του *Coq* για σύνολα, λίστες, ταξινόμηση, αριθμητικά, κτλ. Η βιβλιοθήκη διανέμεται κανονικά με το σύστημα, ενώ τα διάφορα τμήματα της είναι απευθείας προσπελάσιμα με χρήση της κατάλληλης εντολής (*Require*).
- **Συνεισφορές χρηστών.** Προδιαγραφές και διάφορες αναπτύξεις αποδείξεων, από την κοινότητα των χρηστών του *Coq*. Δεν διανέμονται πλέον με τη βιβλιοθήκη, αλλά είναι διαθέσιμα μέσω *FTP*.

4.1.2 Σύστημα τύπων του Coq

Η θεμελιώδης τυπική γλώσσα του *Coq* είναι ο Λογισμός των Κατασκευών με Επαγωγικούς Ορισμούς *(Calculus of Constructions with Inductive Definitions)*. Αν και μέχρι την έκδοση 7 του *Coq* χρησιμοποιηθήκε ο Λογισμός των Συνεπαγωγικών Κατασκευών *(Calculus of CoInductive Constructions)*, προτιμήθηκε από την έκδοση 8 ένας πιο αδύναμος λογισμός όπου το είδος *Set* ικανοποιεί κατηγορηματικούς κανόνες, ο *Κατηγορηματικός Λογισμός Συν-Επαγωγικών Κατασκευών, (Predicate Calculus of (Co)Inductive Constructions - pCic)*.

Στον pCic όλα τα αντικείμενα έχουν τύπο. Υπάρχουν τύποι για συναρτήσεις (ή προγράμματα), υπάρχουν ατομικοί τύποι (ειδικοί τύποι δεδομένων), τύποι για αποδείξεις, αλλά και τύποι για τους ίδιους τους τύπους. Κάθε αντικείμενο που θα χρησιμοποιηθεί, πρέπει να έχει έναν τύπο. Για παράδειγμα, η έκφραση **"για κάθε x, P"** δεν επιτρέπεται από τη θεωρία τύπων, αλλά αντίθετα πρέπει να γραφεί ως **"για κάθε x που ανήκει στο T, P"**, όπου το ότι το *"x ανήκει στο P"* γράφεται ως *"x:T"*, δηλαδή "το x έχει τύπο T".

Τέλος, στον pCic υπάρχει ένας εσωτερικός μηχανισμός αναγωγής, που επιτρέπει να αποφασίσουμε αν δύο προγράμματα είναι ισοδύναμα ή αλλιώς μετατρέψιμα.

4.1.3 Οι όροι

Αν και στις περισσότερες θεωρίες τύπων γίνεται συντακτικά διαφοροποίηση μεταξύ τύπων και όρων, στον pCic οι όροι και οι τύποι ορίζονται με την ίδια συντακτική δομή. Αυτό συμβαίνει γιατί η ίδια η θεωρία τύπων αναγκάζει τους όρους και τους τύπους να οριστούν με έναν αμοιβαία αναδρομικό τρόπο και επίσης επειδή παρόμοιες δομές μπορούν να εφαρμοστούν και σε όρους και σε τύπους και συνεπώς μπορούν να μοιραστούν την ίδια συντακτική δομή.

Ας δούμε ένα παράδειγμα. Έστω ο κατασκευαστής $\rightarrow$ και έστω ότι *nat* είναι ο τύπος των φυσικών αριθμών. Τότε το $\rightarrow$ χρησιμοποιείται και για να ορίσει το: $nat \rightarrow nat$, που είναι ο τύπος των συναρτήσεων από *nat* σε *nat*, αλλά και να ορίσει το $nat \rightarrow Prop$, το οποίο είναι ο τύπος των μοναδιαίων κατηγορημάτων πάνω στους φυσικούς αριθμούς. Έστω τώρα η αφαίρεση η οποία κατασκευάζει συναρτήσεις. Συμφέρει να κατασκευάσουμε "συνηθισμένες" συναρτήσεις όπως

$$fun \;\; x : nat \qquad \Rightarrow \qquad (mult \;\; x \;\; x)$$

(θεωρώντας ότι το mult έχει ήδη ορισθεί) αλλά μπορούμε επίσης να κατασκευάσουμε κατηγορήματα πάνω σε φυσικούς αριθμούς. Για παράδειγμα, η

$$fun \;\; x : nat \qquad \Rightarrow \qquad (x = x)$$

περιγράφει ένα κατηγόρημα P, μη τυπικά γραμμένο στα μαθηματικά ως P(x) ισοδύναμο με x=x. Εάν το P έχει τύπο $nat \rightarrow Prop$, το (P x) είναι μία πρόταση, ενώ επιπλέον για κάθε x:nat, το (P x) θα εκφράζει τον τύπο των συναρτήσεων που αντιστοιχούν σε κάθε φυσικό αριθμό n ένα αντικείμενο με τύπο (P x) και συνεπώς εκφράζει αποδείξεις της μορφής *"για κάθε x.P(x)"*.

Είδη

Αφού οι τύποι αναγνωρίζονται ως όροι της γλώσσας, πρέπει να ανήκουν σε κάποιον άλλο τύπο. Ο τύπος ενός τύπου της γλώσσας είναι πάντα μία σταθερά και καλείται *είδος*. Υπάρχουν δύο βασικά είδη στη γλώσσα του pCic, το **Set** και το **Prop**.

Το είδος *Prop* είναι ο τύπος των λογικών προτάσεων. Εάν M είναι μία λογική πρόταση τότε αυτή δηλώνει μία κλάση, την κλάση των όρων που αντιπροσωπεύουν αποδείξεις για το M. Ένα αντικείμενο m που ανήκει στην M αποδεικνύει ότι πράγματι η M είναι αληθής. Ένα αντικείμενο λοιπόν με τύπο Prop ονομάζεται πρόταση.

Το είδος *Set* είναι ο τύπος των προδιαγραφών, δηλαδή προγραμμάτα και συνήθη σύνολα, όπως φυσικοί αριθμοί, boolean αριθμοί, λίστες, κτλ.

Τα παραπάνω *είδη* μπορούμε να τα χειριστούμε σαν κανονικούς όρους. Ακολούθως, και τα είδη πρέπει να έχουν τύπο. Επειδή αν απλά υποθέταμε ότι τα Set έχουν τύπο Set θα οδηγούσε σε ασυνεπή θεωρία, έχουμε άπειρα *είδη* στη γλώσσα του pCic. Αυτά τα είδη είναι, εκτός του Set και του Prop, μία ιεραρχία από περιβάλλοντα-κόσμους **Type(i)** για κάθε ακέραιο i. Ονομάζουμε S το σύνολο των *ειδών* το οποίο ορίζεται ως εξής:

$$S \text{ ισοδύναμο με } \{Prop, Set, Type(i) | i \text{ στο } \aleph\}$$

ενώ τα είδη έχουν τις εξής ιδιότητες:

$$Prop : Type(0), Set : Type(0) \text{ και } Type(i) : Type(i+1).$$

Παρ' όλο που φαίνεται περίπλοκο, ο χρήστης δεν χρησιμοποιεί ποτέ τους δείκτες i, όταν θέλει να αναφερθεί στο περιβάλλον-κόσμο Type(i), αλλά απλά γράφει Type. Το ίδιο το σύστημα αναλαμβάνει να παράξει για κάθε στιγμιότυπο του Type ένα νέο δείκτη για το συγκεκριμένο περιβάλλον και ελέγχει αν οι περιορισμοί μεταξύ των αυτών των δεικτών μπορούν να λυθούν. Δηλαδή, από την πλευρά του χρήστη έχουμε *Type : Type*.

Σταθερές

Εκτός από τα *είδη*, η γλώσσα επίσης περιέχει σταθερές που δηλώνουν αντικείμενο στο περιβάλλον. Αυτές οι σταθερές μπορεί να δηλώνουν προηγούμενως ορισμένα αντικείμενα, αλλά επίσης και αντικείμενα που σχετίζονται με επαγωγικούς ορισμούς (είτε του ίδιου του τύπου ή ενός από τους κατασκευαστές ή καταστροφείς του).

Όροι

Οι όροι κατασκευάζονται από μεταβλητές, γενικά ονόματα, κατασκευαστές, αφαιρέση, εφαρμογή, τοπικές δεσμεύσεις ορισμών (εκφράσεις "let-in") και παραγωγή. Από συντακτικής άποψης, οι τύποι δεν μπορούν να διαχωριστούν από τους όρους, εκτός του ότι οι τελευταίοι μπορούν να ξεκινούν με αφαίρεση, όπως και ότι αν ένας όρος είναι είδος ή παραγωγή, τότε πρέπει να είναι τύπος.

Η κατασκευή της γλώσσας του Λογισμού των Επαγωγικών Κατασκευών μπορεί ανα συνοψιστεί στους εξής κανόνες:

1. τα είδη *Set, Prop, Type* είναι όροι.
2. τα ονόματα των γενικών μεταβλητών του περιβάλλοντος είναι όροι.
3. οι μεταβλητές είναι όροι.
4. εάν το x είναι μία μεταβλητή και T, U είναι όροι, τότε το: *για κάθε x: T, U*, είναι όρος. Εάν το x βρίσκεται στο U τότε το *για κάθε x: T, U*, σημαίνει "για κάθε x με τύπο T, U". Αφού το U βασίζεται στο x, λέμε ότι για κάθε x:T, το U είναι εξαρτώμενη παραγωγή. Εάν το x δεν βρίσκεται στο U, τότε το *για κάθε x: T, U*, σημαίνει "εάν T τότε U". Τότε, ένα μη εξαρτώμενη παραγωγή.: $T \rightarrow U$.
5. Εάν το x είναι μία μεταβλητή και τα T, U είναι όροι, τότε το $\lambda x : T$, U είναι όρος. Αυτός είναι ένας συμβολισμός για την λαμβδα-αφαίρεση που παρουσιάστηκε προηγούμενως.
6. Εάν T και U είναι όροι τότε το (T U) είναι όρος και σημαίνει "το T εφαρμοσμένο στο U".
7. Εάν το x είναι μεταβλητή, και T, U είναι όροι τότε το: $let\ x := T$ στο U είναι όρος και δηλώνει τον όρο U και την μεταβλητή x να είναι τοπικά δεσμευμένη στο T. Το παραπάνω ισοδυναμεί με το γνωστό *"let-in"* των συναρτησιακών γλωσσών.

Συντομεύσεις

Η εφαρμογή προσεταιρίζει από τα αριστερά, δηλαδή το (t t1 t2 t3 ... tn) να ισοδυναμεί με το (... (... (t t1) t2) ... tn). Οι παραγωγές και τα βέλη προσεταιρίζουν από τα δεξιά, έτσι ώστε το *"για κάθε x:A, $B \rightarrow C \rightarrow D$"* ισοδυναμεί με το *"για κάθε x:A, $(B \rightarrow (C \rightarrow D))$"*. Επίσης ισοδύναμες, είναι και οι εκφράσεις

$$\text{για κάθε } x\ \ y\ :\ A, B\ \ \text{ή}\ \ \lambda\, x\ \ y\ :\ A, B$$

και

$$\text{για κάθε } x : A, \text{για κάθε } y : A, B\ \text{ή}\ \lambda\ x : A,\ \lambda y : A,\ B$$

Ελεύθερες Μεταβλητές Ως γνωστόν για τις ελεύθερες μεταβλητές, στην έκφραση $\lambda x : T, U$ και *"για κάθε x:T, U"* οι εμφανίσεις του x στο U είναι δεσμευμένες. Παρουσιάζονται με τους δείκτες de Bruijn στο εσωτερικό ενός όρου.

Αντικατάσταση Επίσης ως γνωστόν για την αντικατάσταση ενός όρου t στις ελεύθερες εμφανίσεις της μεταβλητής x σε ένα όρο u έχουν αποτέλεσμα τον όρο *u(x/t)*.

4.1.4 Όροι με τύπο

Ως αντικείμενα της θεωρίας τύπων, οι όροι υπόκεινται στους κανόνες των τύπων. Ένας καλά ορισμένος, ως προς τον τύπο, όρος εξαρτάται από ένα περιβάλλον το οποίο αποτελείται από ένα γενικό περιβάλλον και ένα τοπικό. Ας αναλύσουμε λίγο αυτά τα περιβάλλοντα.

Τοπικό ή μικρό πλαίσιο Το τοπικό (ή μικρό) πλαίσιο είναι μία διατεταγμένη λίστα από δηλώσεις μεταβλητών. Μία δήλωση μίας μεταβλητής x είναι είτε μία υπόθεση, που γράφεται ως x:T (όπου T είναι τύπος), είτε ένας ορισμός, που γράφεται ως x:=t:T. Για να ορίσουμε λοιπόν το πλαίσιο, χρησιμοποιούμε άγκιστρα. Για παράδειγμα, το [x:T; y:u; z:V] είναι ένα πλαίσιο, απλά δίνουμε προσοχή στους ορισμούς των μεταβλητών μέσα στο πλαίσιο αφού πρέπει να είναι σαφείς και διακριτοί. Εάν κάποιο Γ ορίζει κάποιο x, γράφουμε *"x στο Γ"*. Γράφοντας (x:T) στο Γ, εννοούμε ότι είτε x:T είναι μία υπόθεση στο Γ είτε ότι υπάρχει κάποιο t τέτοιο ώστε x:=t:T είναι ένας ορισμός στο Γ. Εάν το Γ ορίζει και κάποιο x:=t:T, τότε γράφουμε επίσης και ότι (x:=t:T) στο Γ. Είναι προφανές λοιπόν ότι τα πλαίσια πρέπει να είναι καλώς ορισμένα.

Επίσης, ο εγκλεισμός δύο πλαισίων Γ και Δ (Γ ⊂ Δ) ορίζεται ως η ιδιότητα: *για κάθε μεταβλητή x, τύπο T και όρο t, εάν (x:T) είναι στο Γ, τότε το (x:T) είναι στο Δ και εάν (x:=t:T) στο Γ τότε (x:=t:T) στο Δ.*

Τέλος, μία μεταβλητή x καλείται ελεύθερη στο Γ εάν το πλαίσιο Γ περιέχει μία δήλωση y:T τέτοια ώστε το x είναι ελεύθερο στο T.

Περιβάλλον Επειδή χειριζόμαστε γενικές δηλώσεις (σταθερές και γενικές υποθέσεις), πρέπει επίσης να θεωρήσουμε ένα γενικό περιβάλλον E. Ένα περιβάλλον είναι μία διατεταγμένη λίστα από δηλώσεις γενικών ονομάτων. Οι δηλώσεις είναι είτε υποθέσεις είτε "βασικοί" ορισμοί, που είναι συντομεύσεις από σωστά ορισμένους όρους, αλλά επίσης και ορισμοί επαγωγικών αντικειμένων.

Μία υπόθεση θα παρουσιαστεί στο περιβάλλον σαν *Assum (Γ)(c:T)*, που σημαίνει ότι το c υποθέτεται να έχει τύπο T καλά ορισμένο σε κάποιο πλαίσιο Γ. Ένας κοινός ορισμός θα παρουσιάζεται στο περιβάλλον σας *Def(Γ)(c:=t:T)*, που σημαίνει ότι το c είναι μία σταθερά που ισχύει μόνο σε κάποιο πλαίσιο Γ, του οποίου η τιμή τότε είναι t και ο τύπος T.

Στο σύστημα του Coq λοιπόν, ο ορισμός των φυσικών αριθμών έχει ως εξής:

```
Inductive nat : Set :=
  | O : nat
  | S : nat -> nat.
```

ενώ ο ορισμός της πρόσθεσης κατά Peano είναι ο εξής:

```
Fixpoint plus (n m:nat) {struct n} : nat :=
  match n with
  | O => m
  | S p => S (plus p m)
  end.
```

Ένα αναλυτικό λήμμα για την πρόσθεση κατά Peano, είναι το λήμμα της μεταβατικότητας της πρόσθεσης:

```
Lemma plus_comm : forall n m, n + m = m + n.
Proof.
intros n m; elim n; simpl in |- *; auto with arith.
intros y H; elim (plus_n_Sm m y); auto with arith.
Qed.
```

το οποίο στο επίπεδο της αναπαράστασης με όρους του pCic που εφαρμόζει το Coq γράφεται ως εξής:

```
plus_comm =
   fun n m : nat =>
     nat_ind (fun n0 : nat => n0 + m = m + n0)
             (plus_n_O m)
             (fun (y : nat) (H : y + m = m + y) =>
               eq_ind (S (m + y))
                      (fun n0 : nat => S (y + m) = n0)
                      (f_equal S H) (m + S y) (plus_n_Sm m y))
             n
 : forall n m : nat, n + m = m + n
```

4.2 Περιγραφή της υλοποίησης

Για την μετατροπή των βιβλιοθηκών των θεωριών από το σύστημα Coq στη γλώσσα τύπων nflint αναπτύχθηκε η εφαρμογή *coq2nflint*, στη γλώσσα προγραμματισμού OCaml. Πιο αναλυτικά, ο κώδικας αυτής της εφαρμογής αποτελείται από δύο αρχεία:

- **coq2nflint:** αναλύει λεκτικά τα αρχεία του Coq και επεξεργάζεται τις λεκτικές μονάδες, π.χ. αν είναι *Ορισμός, Επαγωγικός Τύπος, Θεώρημα ή Λήμμα* και αντιστοίχως καλεί συναρτήσεις από το αρχείο *transform*. Αποτελείται από 250 γραμμές κώδικα.

- **transform:** περιέχει συναρτήσεις που καλούνται από το *coq2nflint*, αναλόγως με τη μονάδα από το αρχείο Coq που είναι υπο επεργασία. Οι συναρτήσεις του μετατρέπουν τους όρους του Coq στους αντίστοιχους όρους της nflint, μετά από κατάλληλη επεργασία. Αποτελείται από 734 γραμμές κώδικα.

Η συνολική εφαρμογή δεν μετατρέπει απολύτως αυτόματα τις θεωρίες του Coq στην γλώσσα τύπων και η συνολική μεταφορά των βιβλιοθηκών έγινε με ημιαυτόματο τρόπο για τους εξής λόγους:

- Το σύστημα Coq είναι πιο εκφραστικό από τη γλώσσα τύπων nflint. Για παράδειγμα υποστηρίζει αμοιβαία επαγωγικούς τύπους (mutual inductive) και αμοιβαία αναδρομικό ορισμό συναρτήσεων. Επομένως, υπάρχουν στοιχεία στο Coq τα οποία είτε δεν μπορούν να εκφραστούν στην nflint, είτε η έκφρασή τους είναι τελείως διαφορετική, οπότε απαιτείται και τελείως διαφορετική προσέγγιση στην nflint. Παράδειγμα τέτοιας περίπτωσης ήταν ο ορισμός της πρόσθεσης στη δυαδική αριθμητική φυσικών.

- Οι κανόνες τύπων του Coq διαφέρουν λίγο από αυτούς της γλώσσας τύπων nflint. Ένα παράδειγμα είναι ότι στο Coq αν ισχύει $\Gamma \vdash A : \mathsf{Set}$ τότε ισχύει και $\Gamma \vdash A : \mathsf{Type}$, κάτι που δε συμβαίνει στην nflint. Η ιδιότητα αυτή έχει χρησιμοποιηθεί αρκετά στη βιβλιοθήκη θεωριών του Coq, π.χ. για τον ορισμό της ισότητας Leibniz, επομένως η μεταφορά αυτών των όρων στη γλώσσα nflint χρειάζεται αναπόφευκτα διόρθωση.

- Λόγω των διαφορών των δύο συστημάτων και από τη στιγμή που είναι αναπόφευκτο κάποιες αλλαγές να πρέπει να γίνουν με το χέρι, επιλέχθηκε να γίνει πιο απλή η εφαρμογή μετατροπής.

Με βάση την παραπάνω ανάλυση, τη δομή και τα αντίστοιχα θεωρήματα του Coq, θα προχωρήσουμε στην καταγραφή θεωριών και επιλεγμένων αποδείξεων στη γλώσσα τύπων nflint.

4.3 Βασικοί τύποι

Η βιβλιοθήκη των βασικών τύπων της γλώσσας τύπων είναι η παρακάτω:

- Το `unit` είναι ένας βασικός τύπος με μοναδιαίο στοιχείο τον κατασκευαστή `tt`.

```
unit
  : Set
  := Ind(unit:Set){unit}
unit_ind
  : #P:unit -> Set. P tt -> #u:unit. P u
unit_rec
  : #P:unit -> Set. P tt -> #u:unit. P u
unit_rect
  : #P:unit -> Type. P tt -> #u:unit. P u
tt
  : True
  := Constr(0,unit)
```

- Το `bool` είναι ο βασικός τύπος των boolean τιμών `true` και `false`.

```
bool
  :  Set
  := Ind(bool:Set){bool,bool}
true
  :  bool
  := Constr(0,bool)
false
  :  bool
  := Constr(1,bool)
bool_rect
  :  #P:bool -> Type. P true -> P false -> #b:bool. P b
  := \P:bool -> Type. \f:P true. \f0:P false. \b:bool.
                Elim[\b0:bool. P b0] (b:bool) {f ; f0}
bool_rec
  :   #P:bool -> Type. P true -> P false -> #b:bool. P b
  := \P:bool -> Set. \f:P true. \f0:P false. \b:bool.
                Elim[\b0:bool. P b0] (b:bool) {f ; f0}
bool_ind
  :   #P:bool -> Type. P true -> P false -> #b:bool. P b
  := \P:bool -> Set. \f:P true. \f0:P false. \b:bool.
                Elim[\b0:bool. P b0] (b:bool) {f ; f0}
```

- Ο `nat` είναι ο βασικός τύπος των φυσικών αριθμών και κατασκευάζεται από το μηδέν και το Succesor S. Το μηδέν συμβολίζεται με το γράμμα "O" και όχι με τον αριθμό "0".

```
nat
  :  Set
  := Ind(nat:Set){nat; nat -> nat}

nat_rec : nat_rect
  :  #P:nat -> Type. P O -> (#n:nat. P n -> P (S n)) -> #n:nat. P n
  := \P: nat->Type. \f: P O. \f0: #n:nat.P n -> P(S n). \n:nat.
      Elim[\n0:nat. P n0](n:nat) { f ; \n:nat. \n_z:P n. f0 n n_z}
nat_rec
  :  #P:nat -> Set. P O -> (#n:nat. P n -> P (S n)) -> #n:nat. P n
  := \P: nat->Set. \f: P O. \f0: #n:nat.P n -> P(S n). \n:nat.
```

```
     Elim[\n0:nat. P n0](n:nat) { f ; \n:nat. \n_z:P n. f0 n n_z}
nat_rec
  :  #P:nat -> Set. P 0 -> (#n:nat. P n -> P (S n)) -> #n:nat. P n
  := \P: nat->Set. \f: P 0. \f0: #n:nat.P n -> P(S n). \n:nat.
     Elim[\n0:nat. P n0](n:nat) { f ; \n:nat. \n_z:P n. f0 n n_z}
0
  : nat
  := Constr(0, nat)
S : nat -> nat
  := Constr(1, nat)
```

- Το `Empty_set` το οποίο δεν είναι κατοικημένο.

```
Empty_set
  : Set
Empty_set_ind
  : #P:Empty_set -> Set. #e:Empty_set. P e
Empty_set_rec
  : #P:Empty_set -> Set. #e:Empty_set. P e
Empty_set_rect
  : #P:Empty_set -> Type. #e:Empty_set. P e
```

- Το `identity A` είναι η οικογένεια των βασικών τύπων στο A στην οποία το μόνο μη κενό μέλος είναι ο singleton βασικός τύπος `identity A a a`, του οποίου το μόνο στοιχείο συμβολίζεται με `refl_identity A a`

```
identity
  : #A:Type. A -> A -> Set
  := \A:Type. \a:A. Ind(identity:A -> Set){identity a}
identity_ind
  : #A:Type. #a:A. #P:#a0:A.
      identity A a a0 -> Set. P a (refl_identity A a) ->
                  #y:A. #i:identity A a y. P y i
identity_rec
  : #A:Type. #a:A. #P:#a0:A.
      identity A a a0 -> Set. P a (refl_identity A a) ->
                  #y:A. #i:identity A a y. P y i
identity_rect
  : #A:Type. #a:A. #P:#a0:A.
      identity A a a0 -> Type. P a (refl_identity A a) ->
                  #y:A. #i:identity A a y. P y i
refl_identity
  : #A:Type. #a:A. identity A a a
```

- Το `option A` ειναι η επέκταση του A μαζί με ένα κενό στοιχείο, το `None`.

```
option
  : Set -> Set
option_rect
  : #A:Set. #P:option A -> Type. (#a:A. P (Some A a)) ->
      P (None A) -> #o:option A. P o
None
  : #A:Set. option A
Some
  : #A:Set. A -> option A
```

- Το `sum A B`, ισοδύναμο με το `A + B`, είναι το διακριτό άθροισμα των A και B.

```
sum
  : Set -> Set -> Set
  :=  \A:Set. \B:Set.
          Ind(sum:Set){A -> sum , B -> sum }
sum_ind
  : #A:Set. #B:Set. #P:or A B -> Set.
          (#a:A. P (inl A B a)) -> (#b:B. P (inr A B b)) ->
                 #s:or A B. P s
sum_rec
  : #A:Set. #B:Set. #P:or A B -> Set.
      (#a:A. P (inl A B a)) -> (#b:B. P (inr A B b)) ->
                 #s:or A B. P s
sum_rect
  : #A:Set. #B:Set. #P:or A B -> Type.
      (#a:A. P (inl A B a)) -> (#b:B. P (inr A B b)) ->
                 #s:or A B. P s
inl
  : #A:Set. #B:Set. A -> sum A B
  := \A:Set. \B:Set. Constr(0, sum A B)
inr
  : #A:Set. #B:Set. B -> sum A B
  := \A:Set. \B:Set. Constr(1, sum A B)
```

- Το `prod A B`, το οποίο γράφεται ως $A*B$, είναι το γινόμενο των A και B. Το ζεύγος `pair A B a b` των a και b είναι ο αντίστοιχος κατασκευαστής και εν συντομία θα γράφεται (a, b)

```
prod
  :  Set -> Set -> Set
  := \A:Set. \B:Set.
               Ind(prod:Set){A -> B -> prod}
pair
  :  #A:Set. #B:Set. A -> B -> prod A B
  := \A:Set. \B:Set.
               Constr(0, prod A B)
prod_ind
  : #A:Set. #B:Set. #P:prod A B -> Set. (#a:A. #b:B. P
          (pair A B a b)) -> #p:prod A B. P p
prod_rec
  : #A:Set. #B:Set. #P:prod A B -> Set. (#a:A. #b:B. P
      (pair A B a b)) -> #p:prod A B. P p
prod_rect
  : #A:Set. #B:Set. #P:prod A B -> Type. (#a:A. #b:B. P
      (pair A B a b)) -> #p:prod A B. P p
fst
  : #A:Set. #B:Set. prod A B -> A
  := \A:Set. \B:Set. \p: prod A B.
     Elim[\p:prod A B. A]
              (p:prod A B)
                       {\x:A. \y:B. x}
snd
  : #A:Set. #B:Set. prod A B -> B
  := \A:Set. \B:Set. \p: prod A B.
     Elim[\p:prod A B. B]
              (p:prod A B)
                       {\x:A. \y:B. y}
```

- Θεωρίες για σύγκριση βασικών τύπων.

```
comparison
  : Set
  := Ind(comparison:Set)
           {comparison,comparison,comparison}
comparison_ind
  : #P:comparison -> Set. P Eq -> P Lt -> P Gt -> #c:comparison. P c
comparison_rec
  : #P:comparison -> Set. P Eq -> P Lt -> P Gt -> #c:comparison. P c
comparison_rect
  : #P:comparison -> Type. P Eq -> P Lt -> P Gt -> #c:comparison. P c
Eq
  : comparison
  := Constr(0,comparison)
Lt
  : comparison
  := Constr(1,comparison)
Gt
  : comparison
  := Constr(2,comparison)
```

4.4 Προτασιακή και κατηγορηματική λογική

Βασικές θεωρίες της κατηγορηματικής και της προτασιακής λογικής:

- Το `True` είναι προφανώς η πρόταση που είναι πάντα αληθής.

```
True
  :  Set
  := Ind(True:Set){True}
True_ind
  : #P:Set. P -> True -> P
True_rec
  : #P:Set. P -> True -> P
True_rect
  : #P:Type. P -> True -> P
I
  : True
  := Constr(0, True)
truePrf
  : True
  := Constr(0, True)
```

- Αντίστοιχα, το `False` είναι η πρόταση που είναι πάντα ψευδής.

```
False
  :  Set
  := Ind(False:Set){}
False_ind
  : #P:Set. False -> P
False_rec
  : #P:Set. False -> P
False_rect
  : #P:Type. False -> P
```

- Το `not A` είναι προφανώς η άρνηση του A.

```
not
  :  Set -> Set
  := \p:Set. p -> False
not_eq : #k:Set. k -> k -> Set
```

- Το `and A B` είναι η ένωση των A και B. Το `conj p q` που ακολουθεί είναι η επόδειξη του `and A B`, αν το p είναι απόδειξη του A και το q είναι η απόδειξη του B. Επίσης, `proj1` και `proj2` είναι η πρώτη και δεύτερη προβολή της ένωσης αντίστοιχα.

```
and
  :  Set -> Set -> Set
  := \p1:Set. \p2:Set. Ind(and:Set){p1 -> p2 -> and}
and_ind
  : #A:Set. #B:Set. #P:Set.
      (A -> B -> P) -> and A B -> P

and_rec
  : #A:Set. #B:Set. #P:Set.
      (A -> B -> P) -> and A B -> P
and_rect
  : #A:Set. #B:Set. #P:Type.
      (A -> B -> P) -> and A B -> P
conj
  : #A:Set. #B:Set. A -> B -> and A B

proj1
  : #x:Set. #y:Set. and x y -> Set
proj2
  : #x:Set. #y:Set. and x y -> Set
```

- Το `or A B` είναι η τομή των A και B.

```
or
  :  Set -> Set -> Set
  := \p1:Set. \p2:Set. Ind(X:Set){p1 -> X, p2 -> X}
or_ind
  : #A:Set. #B:Set. #P:Set. (A -> P) -> (B -> P) -> or A B -> P
or_introl
  : #A:Set. #B:Set. A -> or A B
or_intror
  : #A:Set. #B:Set. B -> or A B
```

- Το `iff A B` εκφράζει την ισοδυναμία των A και B.

```
iff
  : Set -> Set -> Set
iff_refl
  : Set -> Set -> Set
iff_sym
  : #x:Set. #y:Set. iff x y -> Set
iff_trans
  : #x:Set. #y:Set. #z:Set. iff x y -> iff y z -> Set
```

- Το `IF_then_else P Q R` δηλώνει είτε ότι το `P and Q` είναι αληθές, είτε ότι το ∼`P and Q` είναι αληθές.

```
IF_then_else : Set -> Set -> Set -> Set
```

- Το `ex A P`, εκφράζει την ύπαρξη ενός x με τύπο A, όπου το A έχει τύπο Set, το οποίο ικανοποιεί το κατηγόρημα P. Το `ex2 A P Q` αντίστοιχα εκφράζει την ύπαρξη ενός x με τύπο A το οποίο ικανοποιεί και τα δύο κατηγορήματα P και Q.

```
ex
  : #A:Type. (A -> Set) -> Set
ex2
  : #A:Type. (A -> Set) -> (A -> Set) -> Set
ex2_ind
  : #A:Type. #P:A -> Set. #Q:A -> Set. #P0:Set.
      (#x:A. P x -> Q x -> P0) -> ex2 A P Q -> P0
ex_ind
  : #A:Type. #P:A -> Set. #P0:Set.
      (#x:A. P x -> P0) -> ex A P -> P0
ex_intro
  : #A:Type. #P:A -> Set. #x:A. P x -> ex A P
ex_intro2
  : #A:Type. #P:A -> Set. #Q:A -> Set.
      #x:A. P x -> Q x -> ex2 A P Q
all
  : #A:Type. (A -> Set) -> Set
```

- Οι καθολικοί ποσοδείκτες (ειδικά οι πρώτης τάξης) συνήθως γράφονται ως εξής: `forall x:A, P x`. Επίσης, οι υπαρξιακοί ποσοδείκτες ορίζονται ως: `all P`

```
inst
  : #A:Type. #P:A -> Set. #x:A. (#x:A. P x) -> P x
gen
  : #A:Type. #P:A -> Set. #B:Set. (#y:A. B -> P y) -> B -> #x:A. P x
```

- Το `eq A x y` εκφράζει την ισότητα, κατά Leibniz, των x και y. Τα x και y πρέπει να έχουν τον ίδιο τύπο. Ο ορισμός είναι επαγωγικός και δηλώνει την ανακλαστική ιδιότητα της ισότητας. Οι άλλες ιδιότητες (συμμετρία, μεταβατικότητα) της ισότητας κατά Leibniz αποδεικνύονται στη συνέχεια.

```
eq
  :  #k:Set. k -> k -> Set
  := \k:Set. \a:k. Ind(X:k -> Set){X a}
eq_ind
  : #A:Set. #x:A. #P:A -> Set. P x -> #y:A. eq A x y -> P y
eq_rec
  : #A:Set. #x:A. #P:A -> Set. P x -> #y:A. eq A x y -> P y
eq_rect
  : #A:Set. #x:A. #P:A -> Type. P x -> #y:A. eq A x y -> P y
refl_equal
  : #k:Set. #a:k. eq k a a
  := \k:Set. \a:k. Constr(0, eq k a)

absurd
  : #A:Set. #C:Set. A -> (A -> False) -> C

sym_eq
  : #A:Set. #x:A. #y:A. eq A x y -> eq A y x

trans_eq
  : #A:Set. #x:A. #y:A. #z:A.
      eq A x y -> eq A y z -> eq A x z
```

```
f_equal
  : #A:Set. #B:Set. #f:A -> B.
      #x:A. #y:A. eq A x y -> eq A (f x) (f y)
sym_not_eq
  : #A:Set. #x:A. #y:A.
      not_eq A x y -> eq A y x -> not_eq A x y -> eq A x y

sym_eq
  : sym_equal
  := \A:Set. \x:A. \y:A. \H: eq A x y.
     Elim[\y0:A.\H:eq A x y0. eq A y0 x]
              (H : eq A x y)
                       { refl_equal A x }
trans_eq
  : trans_equal
  := \A:Set. \x:A. \y:A. \z:A. \H: eq A x y. \H0: eq A y z.
    Elim[\y0:A. \H0:eq A z y0. eq A x y0]
             (H0: eq A y z)
                  { H }
sym_not_eq
  : sym_not_equal
  := \A:Set. \x:A. \y:A. \h1: not_eq A x y. \h2: eq A y x.
     Elim[\y0:A.\h2:eq A y0 x. not_eq A y0 y -> eq A y0 y]
              (h2:eq A y x)
                       {\k: not_eq A y y. refl_equal A y}


eq_ind_r
  : #A:Set. #x:A. #P:A -> Set. P x -> #y:A.
      eq A y x -> P y
eq_rec_r
  : #A:Set. #x:A. #P:A -> Set. P x -> #y:A.
      eq A y x -> P y
eq_rect_r
  : #A:Set. #x:A. #P:A -> Type. P x -> #y:A.
      eq A y x -> P y


f_equal2
  : #A1:Set. #A2:Set. #B:Set.
    #f:A1 -> A2 -> B. #x1:A1. #y1:A1. #x2:A2. #y2:A2.
           eq A1 x1 y1 -> eq A2 x2 y2 -> eq B (f x1 x2) (f y1 y2)
f_equal3
  : #A1:Set. #A2:Set. #A3:Set. #B:Set.
    #f:A1 -> A2 -> A3 -> B. #x1:A1. #y1:A1. #x2:A2. #y2:A2. #x3:A3. #y3:A3.
           eq A1 x1 y1 -> eq A2 x2 y2 -> eq A3 x3 y3 ->
                  eq B (f x1 x2 x3) (f y1 y2 y3)
f_equal4
  : #A1:Set. #A2:Set. #A3:Set. #A4:Set. #B:Set.
    #f:A1 -> A2 -> A3 -> A4 -> B. #x1:A1. #y1:A1. #x2:A2. #y2:A2.
           #x3:A3. #y3:A3. #x4:A4. #y4:A4.
        eq A1 x1 y1 -> eq A2 x2 y2 -> eq A3 x3 y3 -> eq A4 x4 y4 ->
                    eq B (f x1 x2 x3 x4) (f y1 y2 y3 y4)
f_equal5
  : #A1:Set. #A2:Set. #A3:Set. #A4:Set. #A5:Set. #B:Set.
    #f:A1 -> A2 -> A3 -> A4 -> A5 -> B.
           #x1:A1. #y1:A1. #x2:A2. #y2:A2. #x3:A3. #y3:A3.
                  #x4:A4. #y4:A4. #x5:A5. #y5:A5.
```

```
                    eq A1 x1 y1 -> eq A2 x2 y2 -> eq A3 x3 y3 ->
                           eq A4 x4 y4 -> eq A5 x5 y5 ->
                                  eq B (f x1 x2 x3 x4 x5) (f y1 y2 y3 y4 y5)
```

Διάφορες ιδιότητες των αποκρίσιμων προτάσεων:

```
decidable
  :  Set -> Set
  := \P:Set. or P (not P)

dec_not_not
  : #P:Set. decidable P -> (not P -> False) -> P
dec_False
  : decidable False
dec_True
  : decidable True


dec_or
  : #A:Set. #B:Set. decidable A -> decidable B ->
      decidable (or A B)
dec_and
  : #A:Set. #B:Set. decidable A -> decidable B ->
      decidable (and A B)
dec_not
  : #A:Set. decidable A -> decidable (not A)

dec_imp
  : #A:Set. #B:Set. decidable A -> decidable B ->
      decidable (A -> B)
not_not
  : #P:Set. decidable P -> not (not P) -> P


not_or
  : #A:Set. #B:Set. not (or A B) -> and (not A) (not B)
not_and
  : #A:Set. #B:Set. decidable A -> not (and A B) ->
      or (not A) (not B)
not_imp
  : #A:Set. #B:Set. decidable A -> not (A -> B) -> and A (not B)
imp_simp
  : #A:Set. #B:Set. decidable A -> (A -> B) -> or (not A) B
```

Ακολουθούν θεωρίες για σύνολα που περιέχουν πληροφορίες λογικής.

- Το `sig A P` ορίζει το υποσύνολο των στοιχείων του συνόλου A τα οποία ικανοποιούν το κατηγόρημα P. Αντίστοιχα, το `sig2 A P Q` ορίζει το υποσύνολο των στοιχείων του A τα οποία ικανοποιούν και το κατηγόρημα P και το κατηγόρημα Q.

  ```
  sig
    : #A:Set. (A -> Set) -> Set
  exist
    : #A:Set. #P:A -> Set. #x:A. P x -> sig A P


  sig2
    : #A:Set. (A -> Set) -> (A -> Set) -> Set
  ```

```
sig2_ind
  : #A:Set. #P:A -> Set. #Q:A -> Set.
      #P0:sig2 A P Q -> Set. (#x:A. #p:P x. #q:Q x. P0
                  (exist2 A P Q x p q)) -> #s:sig2 A P Q. P0 s
sig2_rec
  : #A:Set. #P:A -> Set. #Q:A -> Set.
      #P0:sig2 A P Q -> Set. (#x:A. #p:P x. #q:Q x. P0
                  (exist2 A P Q x p q)) -> #s:sig2 A P Q. P0 s
sig2_rect
  : #A:Set. #P:A -> Set. #Q:A -> Set.
      #P0:sig2 A P Q -> Type. (#x:A. #p:P x. #q:Q x. P0
                  (exist2 A P Q x p q)) ->#s:sig2 A P Q. P0 s
exist2
  : #A:Set. #P:A -> Set. #Q:A -> Set.
      #x:A. P x -> Q x -> sig2 A P Q
```

- Το `sigS A P` είναι μία λεπτή παραλλαγή του υποσυνόλου που αποδείξαμε παραπάνω, όπου το P έχει τώρα τύπο Set. Ομοίως και για το `sigS2 A P Q`.

```
sigS : #A:Set. (A -> Set) -> Set
sigS_ind : #A:Set. #P:A -> Set. #P0:sigS A P -> Set.
                   (#x:A. #p:P x. P0 (existS A P x p)) -> #s:sigS A P. P0 s
sigS_rec : #A:Set. #P:A -> Set. #P0:sigS A P -> Set.
                   (#x:A. #p:P x. P0 (existS A P x p)) -> #s:sigS A P. P0 s
sigS_rect : #A:Set. #P:A -> Set. #P0:sigS A P -> Type.
                   (#x:A. #p:P x. P0 (existS A P x p)) -> #s:sigS A P. P0 s
existS
  : #A:Set. #P:A -> Set. #x:A. P x -> sigS A P


sigS2
  : #A:Set. (A -> Set) -> (A -> Set) -> Set
existS2
  : #A:Set. #P:A -> Set. #Q:A -> Set. #x:A.
      P x -> Q x -> sigS2 A P Q
```

- Οι προβολές του sig.

```
proj1_sig
  : #A:Set. #P:A -> Set. sig A P -> A
proj2_sig
  : #A:Set. #P:A -> Set. #e:sig A P. P (proj1_sig A P e)
```

- Οι προβολές του sigS. Ένα στοιχείο y ενός υποσυνόλου $(x : A(Px))$ είναι το ζεύγος ενός στοιχείου a με τύπο A και της απόδειξης h ότι το a ικανοποιεί το P. Τότε το `projS1 A` y είναι το a και το `projS2 A` y είναι η απόδειξη του (P a).

```
projS1
  : #A:Set. #P:A -> Set. sigS A P -> A
projS2
  : #A:Set. #P:A -> Set. #x:sigS A P. P (projS1 A P x)
```

- Επέκταση Boolean τιμών.

```
sumbool
  : Set -> Set -> Set
sumbool_ind
```

```
  : #A:Set. #B:Set. #P:sumbool A B -> Set.
      (#a:A. P (left A B a)) -> (#b:B. P (right A B b)) ->
                  #s:sumbool A B. Ps
sumbool_rec
  : #A:Set. #B:Set. #P:sumbool A B -> Set.
      (#a:A. P (left A B a)) -> (#b:B. P (right A B b)) ->
                  #s:sumbool A B. Ps
sumbool_rect
  : #A:Set. #B:Set. #P:sumbool A B -> Type.
      (#a:A. P (left A B a)) -> (#b:B. P (right A B b)) ->
                  #s:sumbool A B.P s
right
  : #A:Set. #B:Set. B -> sumbool A B
left
  : #A:Set. #B:Set. A -> sumbool A B


sumor
  : Set -> Set -> Set
sumor_ind
  : #A:Set. #B:Set. #P:sumor A B -> Set.
      (#a:A. P (inleft A B a)) -> (#b:B. P (inright A B b)) ->
                  #s:sumor A B. P s
sumor_rec
  : #A:Set. #B:Set. #P:sumor A B -> Set.
      (#a:A. P (inleft A B a)) -> (#b:B. P (inright A B b)) ->
                  #s:sumor A B. P s
sumor_rect
  : #A:Set. #B:Set. #P:sumor A B -> Type.
      (#a:A. P (inleft A B a)) -> (#b:B. P (inright A B b)) ->
                  #s:sumor A B. Ps
inleft
  : #A:Set. #B:Set. A -> sumor A B
inright
  : #A:Set. #B:Set. B -> sumor A B
```

- Το σύνολο λημμάτων που εκφράζουν σε άλλη μορφή το αξίωμα της επιλογής στοιχείου από σύνολο.

```
Choice
  : #S:Set. #S':Set. #R:S -> S' -> Set.
      (#x:S. sig S' (\y:S'. R x y)) -> sig (S -> S')
                  (\f:S -> S'. #z:S. R z (f z))
Choice2
  : #S:Set. #S':Set. #R':S -> S' -> Set.
      (#x:S. sigS S' (\y:S'. R' x y)) -> sigS (S -> S')
                  (\f:S -> S'. #z:S. R' z (f z))
bool_choice
  : #S:Set. #R1:S -> Set. #R2:S -> Set.
      (#x:S. sumbool (R1 x) (R2 x)) -> sig (S -> bool)
                  (\f:S -> bool. #x:S. or (and (eq bool (f x) true)
                    (R1 x)) (and (eq bool (f x) false) (R2 x)))
```

- Το αποτέλεσμα του τύπου (Exc A) είναι είτε μία κανονική τιμή με τύπο A, είτε το λάθος: "`Inductive Exc [A:Set] : Set := value : A -> Exc A | error : Exc A`".

```
Exc
  : Set -> Set
```

```
  := option
error
  : #A:Set. Exc A
  := Some
value
  : #A:Set. A -> Exc A
  := None


except
  : #P:Set. False -> P
absurd_set
  : #A:Set. #C:Set. A -> not A -> C
```

- Οι θεωρίες του sig αλλά με τύπο Type.

```
sigT
  : #A:Type. (A -> Type) -> Set
sigT_ind
  : #A:Type. #P:A -> Type. #P0:sigT A P -> Set.
      (#x:A. #p:P x. P0 (existT A P x p)) -> #s:sigT A P. P0 s
sigT_rec
  : #A:Type. #P:A -> Type. #P0:sigT A P -> Set.
      (#x:A. #p:P x. P0 (existT A P x p)) -> #s:sigT A P. P0 s
existT
  : #A:Type. #P:A -> Type. #x:A. P x -> sigT A P
```

Παρακάτω, το τελευταίο κομμάτι της βιβλιοθήκης της προτασιακής και κατηγορηματικής λογικής, καταγράφει τις θεωρίες για τον ορισμό των καλώς ορισμένων όρων. Αποδεικνύονται η σωστά διατυπωμένη αναδρομή και η σωστά διατυπωμένη επαγωγή.

- Η αρχή της σωστά διατυπωμένης επαγωγής, σε τύπο Set.

```
Acc
  : #A:Set. (A -> A -> Set) -> A -> Set
Acc_inv
  : #A:Set. #R:A -> A -> Set. #x:A. Acc A R x -> #y:A. R y x -> Acc A R y
Acc_intro
  : #A:Set. #R:A -> A -> Set. #x:A. (#y:A. R y x -> Acc A R y) -> Acc A R x
Acc_ind
  : #A:Set. #R:A -> A -> Set. #P:A -> Set. (#x:A. (#y:A. R y x ->
Acc_rect
  : #A:Set. #R:A -> A -> Set. #P:A -> Type.
     (#x:A. (#y:A. R y x -> Acc A R y) ->
       (#y:A. R y x -> P y) -> P x) -> #x:A. Acc A R x -> P x
Acc_rec
  : #A:Set. #R:A -> A -> Set. #P:A -> Set.
     (#x:A. (#y:A. R y x -> Acc A R y) ->
       (#y:A. R y x -> P y) -> P x) -> #x:A. Acc A R x -> P x
```

- Μία απλοποιημένη εκδοχή του Acc_rect και Acc_rec.

```
Acc_iter
  : #A:Set. #R:A -> A -> Set. #P:A -> Type.
      (#x:A. (#y:A. R y x -> P y) -> P x) -> #x:A. Acc A R x -> P x
```

- Μία σχέση είναι σωστά διατυπωμένη αν κάθε στοιχείο της είναι προσπελάσιμο.

```
well_founded
  : #A:Set. (A -> A -> Set) -> Set
well_founded_ind
  : #A:Set. #R:A -> A -> Set. well_founded A R -> #P:A -> Set.
     (#x:A. (#y:A. R y x -> P y) -> P x) -> #a:A. P a
well_founded_rec
  : #A:Set. #R:A -> A -> Set. well_founded A R -> #P:A -> Set.
      (#x:A. (#y:A. R y x -> P y) -> P x) -> #a:A. P a
```

- Σωστά διατυπωμένη επαγωγή, στον τύπο Set.

```
well_founded_induction_type
  : #A:Set. #R:A -> A -> Set. well_founded A R -> #P:A -> Type.
      (#x:A. (#y:A. R y x -> P y) -> P x) -> #a:A. P a

well_founded_induction
  : #A:Set. #R:A -> A -> Set. well_founded A R -> #P:A -> Set.
      (#x:A. (#y:A. R y x -> P y) -> P x) -> #a:A. P a

well_founded_ind
  : #A:Set. #R:A -> A -> Set. well_founded A R -> #P:A -> Set.
      (#x:A. (#y:A. R y x -> P y) -> P x) -> #a:A. P a
```

- Κατασκευή σταθερού σημείου.

```
Fix_F
  : #A:Set. #R:A -> A -> Set. #P:A -> Set.
      (#x:A. (#y:A. R y x -> P y) -> P x) -> #x:A. Acc A R x -> P x
Fix
  : #A:Set. #R:A -> A -> Set. well_founded A R -> #P:A -> Set.
      (#x:A. (#y:A. R y x -> P y) -> P x) -> #x:A. P x
  := \A:Set. \R:A -> A -> Set. \Rwf:well_founded A R. \P:A -> Set.
       \F:#x:A. (#y:A. R y x -> P y) -> P x. \x:A. Fix_F A R P F x (Rwf x)
```

- Απόδειξη ότι η θεωρία "well_founded_induction" ικανοποιεί την ισότητα του σταθερού σημείου.

```
Acc_inv_dep
  : #A:Set. #R:A -> A -> Set. #P:#a:A. Acc A R a -> Set.
      (#x:A. #a:#y:A. R y x -> Acc A R y. (#y:A. #r:R y x. P y (a y r)) ->
        P x (Acc A R x a)) -> #a:A. #a:Acc A R a. P a a

Fix_F_eq
  : #A:Set. #R:A -> A -> Set. #P:A -> Set. #F:#x:A.
      (#y:A. R y x -> P y) -> P x. #x:A. #r:Acc A R x.
        eq (P x) (F x (\y:A. \p:R y x. Fix_F A R P F y
                    (Acc_inv A R x r y p)))
                           (Fix_F A R P F x r)
Fix_F_inv
  : #A:Set. #R:A -> A -> Set. well_founded A R -> #P:A -> Set. #F:#x:A.
      (#y:A. R y x -> P y) -> P x. (#x:A. #f:#y:A. R y x -> P y. #g:#y:A.
        R y x -> P y. (#y:A. #p:R y x. eq (P y) (f y p) (g y p)) ->
          eq (P x) (F x f) (F x g)) ->
                            #x:A. #r:Acc A R x. #s:Acc A R x.
                              eq (P x) (Fix_F A R P F x r) (Fix_F A R P F x s)
Fix_eq
  : #A:Set. #R:A -> A -> Set. #Rwf:well_founded A R. #P:A -> Set.
```

```
      #F:#x:A. (#y:A. R y x -> P y) -> P x.
        (#x:A. #f:#y:A. R y x -> P y. #g:#y:A. R y x -> P y.
          (#y:A. #p:R y x. eq (P y) (f y p) (g y p)) ->
               eq (P x) (F x f) (F x g)) ->
          #x:A. eq (P x) (Fix A R Rwf P F x)
                  (F x (\y:A. \p:R y x. Fix A R Rwf P F y))
```

- Αναγωγή πάνω στα ζεύγη

```
Acc_iter_2
  : #A:Set. #B:Set. #R:prod A B -> prod A B -> Set. #P:A -> B -> Type.
     (#x:A. #x':B. (#y:A. #y':B. R (pair A B y y')
            (pair A B x x') -> P y y') -> P x x') ->
                  #x:A. #x':B. Acc (prod A B) R (pair A B x x') -> P x x'
well_founded_induction_type_2
  : #A:Set. #B:Set. #R:prod A B -> prod A B -> Set. #P:A -> B -> Type.
              (#x:A. #x':B. (#y:A. #y':B. R (Constr(0, prod) A B y y')
                (pair A B x x') -> P y y') -> P x x') ->
            well_founded (prod A B) R -> (#x:A. #x':B. (#y:A. #y':B. R
                  (pair  A B y y') (pair A B x x') -> P y y') -> P x x') ->
                              #a:A. #b:B. P a b
```

4.5 Αριθμητική Peano

Σε αυτό το σημείο κατασκευάζουμε τους φυσικούς αριθμούς, όπως ορίστηκαν στους βασικούς τύπους, με βάση το μηδέν *"0"* και το successor *"S"*. Έτσι, ορίζονται η πρόσθεση, ο πολλαπλασιασμός, ανισοτικές σχέσεις, κ.ά.

```
eq_S
  : #x:nat. #y:nat. eq nat x y -> eq (S x) (S y)
```

- Η συνάρτηση του προηγούμενου.

```
pred
  :  nat -> nat
  := \n:nat.
     Elim[\n:nat. nat]
              (n:nat)
                        {
                          0;
                          \u:nat. \u_z:nat. u
                        }
pred_Sn
  : #n:nat. eq nat n (pred (S n))
eq_add_S
  : #n:nat. #m:nat. eq nat (S n) (S m) -> eq nat n m
```

- Οι συνέπειες των προηγούμενων αξιωμάτων.

```
not_eq_S
  : #n:nat. #m:nat. not (eq nat n m) ->
      eq nat (S n) (S m) -> Empty_set
IsSucc
  : nat -> Set
0_S
  : #n:nat. eq nat 0 (S n) -> Empty_set
n_Sn
  : #n:nat. eq nat n (S n) -> False
```

- Η πρόσθεση κατά Peano.

```
plus
  :  nat -> nat -> nat
  := \n:nat. \m:nat.
     Elim[\n:nat. nat]
         (n:nat)
     {
       m;
       \p:nat. \p_z:nat. S p_z
     }
plus_n_O
  : #n:nat. eq nat n (plus n O)
plus_O_n
  : #n:nat. eq nat (plus n O) n
plus_Sn_m
  : #n:nat. #m:nat.
      Ind(X:nat -> Set){X (S (plus n m))}(S (plus n m))
plus_n_Sm
  : #n:nat. #m:nat.
      Ind(X:nat -> Set){X (S (plus n m))}(plus n (S m))
```

- Ο πολλαπλασιασμός κατά Peano.

```
mult
  :  nat -> nat -> nat
  := \n:nat. \m:nat.
     Elim[\n:nat. nat]
              (n:nat)
                       {
                         O;
                         \p:nat. \p_z:nat. plus m p_z
                       }
mult_n_O
  : #n:nat. Ind(X:nat -> Set){X O} (mult n O)
mult_n_Sm
  : #n:nat. #m:nat.
      Ind(X:nat -> Set){X (plus (mult n m) n)} (mult n (S m))
```

- Ορισμός της αφαίρεσης για τους φυσικούς. Ισχύει ότι $m - n = 0$ εάν $n \geq m$

```
minus
   :  nat -> nat -> nat
   := \n:nat. \m:nat.
        Elim[\n:nat. nat -> nat]
            (n:nat)
            {
              \m:nat. O;
              \k:nat. \k_z:nat -> nat. \m:nat.
                 Elim[\m:nat. nat]
                     (m:nat)
                     {
                       S k;
                       \l:nat. \l_z:nat. k_z l
                     }
            } m
```

- Ορισμός των συνηθισμένων διατάξεων, επαγωγικά.

```
le
  :  nat -> nat -> Set
  := \n:nat.
           Ind(le:nat -> Set){le n; #m:nat. le m -> le (S m)}
le_ind
  : #n:nat. #P:nat -> Set. P n ->
      (#m:nat. le n m -> P m -> P (S m)) -> #n':nat. le n n' -> P n'
le_S
  : #n:nat. #m:nat. le n m -> le n (S m)
le_n
  : #n:nat.
      Ind(le:nat -> Set){le n; #m:nat. le m -> le (S m)} n

lt
  :  nat -> nat -> Set
  := \n:nat. \m:nat. le (S n) m
ge
  :  nat -> nat -> Set
  := \n:nat. \m:nat. le m n
gt
  :  nat -> nat -> Set
  := \n:nat. \m:nat. lt m n
```

- Ταίριασμα προτύπων για τους φυσικούς αριθμούς.

```
nat_case
  : #n:nat. #P:nat -> Set. P O -> (#m:nat. P (S m)) -> P n
```

- Ορισμός της διπλής επαγωγής.

```
nat_double_ind
  : #R:nat -> nat -> Set. (#n:nat. R O n) ->
      (#n:nat. R (S n) O) -> (#n:nat. #m:nat. R n m ->
                  R (S n) (S m)) -> #n:nat. #m:nat. R n m
```

Ακολουθούν οι βασικές ιδιότητες αριθμητικών σχέσεων κατά Peano.

- Διατάξεις στους φυσικούς αριθμούς (1).
  - Ανακλαστική ιδιότητα.

```
le_refl
  : #n:nat. le n n
  := le_n
```

  - Μεταβατική ιδιότητα.

```
le_trans
  : #n:nat. #m:nat. #p:nat. le n m -> le m p ->
     Ind(le:nat -> Set){le n; #m:nat. le m -> le (S m)}p
  := \n:nat. \m:nat. \p:nat. \H:le n m. \H0:le m p.
       le_ind m (\p:nat. le n p) H
         (\m0:nat. \H0':le m m0. \IHle:le n m0. le_S n m0 IHle) p H0
```

  - Ορισμός της διάταξη, του επόμενου και του προηγούμενο.

```
le_n_S
  : #n:nat. #m:nat. le n m ->
      Ind(le:nat -> Set)
                  {le (S n); #m:nat. le m -> le (S m)}
```

```
                          (S m)
le_n_Sn
  : #n:nat.
      Ind(le:nat -> Set){le n; #m:nat. le m -> le (S m)}(S n)
le_0_n
  : #n:nat.
      Ind(le:nat -> Set){le 0; #m:nat. le m -> le (S m)} n
le_pred_n
  : #n:nat.
      Ind(le:nat -> Set){le (pred n); #m:nat. le m -> le (S m)}n
le_Sn_le
  : #n:nat. #m:nat. le (S n) m ->
      Ind(le:nat -> Set){le n; #m:nat. le m -> le (S m)}m
le_S_n
  : #n:nat. #m:nat. le (S n) (S m) ->
      Ind(le:nat -> Set)
                  {le (pred (S n)); #m:nat. le m -> le (S m)}
                       (pred (S m))
le_pred
  : #n:nat. #m:nat. le n m -> le (pred n) (pred m)
```

– Σύγκριση με το μηδέν.

```
le_Sn_0
  : #n:nat. le (S n) 0 -> Empty_set
le_n_0_eq
  : #n:nat. le n 0 -> eq nat 0 n
```

– Ιδιότητες αρνητικών αριθμών

```
le_Sn_n
  : #n:nat. le (S n) n -> False S q)) -> #n:nat. #m:nat.
      len m -> P n m
```

– Αντισυμμετρική ιδιότητα.

```
le_antisym
  : #n:nat. #m:nat. le n m -> le m n -> eq nat n m
```

– Μια διαφορετική αρχή για την απαλοιφή, ως αναφορά τη διάταξη στους φυσικούς αριθμούς.

```
le_elim_rel
  : #P:nat -> nat -> Set. (#p:nat. P nat p) ->
      (#p:nat. #q:nat. le p q -> P p q ->
                  P (nat p) (nat q)) -> #n:nat. #m:nat. le n m -> P n m
```

• Διατάξεις στους φυσικούς αριθμούς (2).

– Μη-ανακλαστική ιδιότητα.

```
lt_irrefl
  : #n:nat. not (le (S n) n)
```

– Σχέση μεταξύ του Le και του Lt.

```
lt_le_S
  : #n:nat. #m:nat. lt n m ->
      Ind(le:nat -> Set)
                   {le (S n); #m:nat. le m -> le (S m)}
                          m
lt_n_Sm_le
  : #n:nat. #m:nat. lt n (S m) ->
```

```
      Ind(le:nat -> Set)
                    {le (pred (S n)); #m:nat. le m -> le (S m)}
                           (pred (S m))
le_lt_n_Sm
  : #n:nat. #m:nat. le n m ->
      Ind(le:nat -> Set)
                    {le (S n); #m:nat. le m -> le (S m)}
                           (S m)
le_not_lt
  : #n:nat. #m:nat. le n m -> lt m n -> False
lt_not_le
  : #n:nat. #m:nat. lt n m -> le m n -> Empty_set
```

– Ασυμμετρία.

```
lt_asym
  : #n:nat. #m:nat. lt n m -> lt m n -> False
```

– Διάταξη και Επόμενος.

```
lt_n_Sn
  : #n:nat.
      Ind(le:nat -> Set){le (S n); #m:nat. le m -> le (S m)} (S n)
lt_S
  : #n:nat. #m:nat. lt n m ->
      Ind(le:nat -> Set)
                    {le (S n); #m:nat. le m -> le (S m)}
                           (S m)
lt_n_S
  : #n:nat. #m:nat. lt n m ->
      Ind(le:nat -> Set)
                    {le (S (S n)); #m:nat. le m -> le (S m)}
                           (S m)
lt_S_n
  : #n:nat. #m:nat. lt (S n) (S m) ->
      Ind(le:nat -> Set)
                    {le (S n); #m:nat. le m -> le (S m)}
                           m
lt_0_Sn
  : #n:nat.
      Ind(le:nat -> Set)
                    {le (S 0); #m:nat. le m -> le (S m)}
                           (S n)
lt_n_0
  : #n:nat. not (le (S n) 0)
```

– Επόμενος.

```
S_pred
  : #n:nat. #m:nat. lt m n ->
      Ind(X:nat -> Set){X n}(S (pred n))
lt_pred
  : #n:nat. #m:nat. lt (S n) m ->
      Ind(le:nat -> Set)
                    {le (S n); #m:nat. le m -> le (S m)}
                           (pred m)
lt_pred_n_n
  : #n:nat. lt nat n -> lt (pred n) n
```

– Ιδιότητες για τη μεταβατικότητα.

```
lt_trans
  : #n:nat. #m:nat. #p:nat. lt n m -> lt m p ->
      Ind(le:nat -> Set){le (S n); #m:nat. le m -> le (S m)} p
lt_le_trans
  : #n:nat. #m:nat. #p:nat. lt n m -> le m p ->
      Ind(le:nat -> Set){le (S n); #m:nat. le m -> le (S m)} p
le_lt_trans
  : #n:nat. #m:nat. #p:nat. le n m -> lt m p ->
      Ind(le:nat -> Set){le (S n); #m:nat. le m -> le (S m)} p

le_lt_or_eq
  : #n:nat. #m:nat. le n m ->
      Ind(X:Set){lt n m -> X; eq nat n m -> X}
lt_le_weak
  : #n:nat. #m:nat. lt n m ->
      Ind(le:nat -> Set){le n; #m:nat. le m -> le (S m)}m
```

– Διχοτόμηση.

```
le_or_lt
  : #n:nat. #m:nat.
      Ind(X:Set){le n m -> X; lt m n -> X}
nat_total_order
  : #m:nat. #n:nat. not (eq nat m n) ->
      Ind(X:Set){lt m n -> X; lt n m -> X}
```

– Σύγκριση με το μηδέν.

```
neq_0_lt
  : #n:nat. not (eq nat 0 n) -> lt 0 n
lt_0_neq
  : #n:nat. lt 0 n -> eq nat 0 n -> False
```

- Διατάξεις στους φυσικούς (3).

– Διάταξη και Επόμενος.

```
gt_Sn_0
  : #n:nat.
      Ind(le:nat -> Set)
                    {le (S 0); #m:nat. le m -> le (S m)}
                           (S n)
gt_Sn_n
  : #n:nat.
     Ind(le:nat -> Set)
                {le (S n); #m:nat. le m -> le (S m)}
                             (S n)
gt_n_S
  : #n:nat. #m:nat. gt n m ->
     Ind(le:nat -> Set)
                {le (S (S m)); #m:nat. le m -> le (S m)}
                             (S n)
gt_S_n
  : #n:nat. #m:nat. gt (S m) (S n) ->
     Ind(le:nat -> Set)
                {le (S n); #m:nat. le m -> le (S m)}
                             m
gt_S
  : #n:nat. #m:nat. gt (S n) m ->
      Ind(X:Set){lt m n -> X; eq nat m n -> X}
gt_pred
```

```
  : #n:nat. #m:nat. gt m (S n) ->
      Ind(le:nat -> Set)
                   {le (S n); #m:nat. le m -> le (S m)}
                           (pred m)
```

– Ιδιότητα της μη ανακλαστικότητας.

```
gt_irrefl
  : #n:nat. not (lt n n)
  := lt_irrefl
```

– Ιδιότητα της ασυμμετρίας.

```
gt_asym
  : #n:nat. #m:nat. lt m n -> not (lt n m)
  := \n:nat. \m:nat. lt_asym m n
```

– Διατάξεις “μεγαλύτερο” ή “μεγαλύτερο ή ίσο”.

```
le_not_gt
  : #n:nat. #m:nat. le n m -> not (lt m n)

gt_not_le
  : #n:nat. #m:nat. gt n m -> le n m -> False
le_S_gt
  : #n:nat. #m:nat. le (S n) m ->
      Ind(le:nat -> Set)
                   {le (S n); #m:nat. le m -> le (S m)}
                           m
gt_S_le
  : #n:nat. #p:nat. lt n (S p) -> le n p
gt_le_S
  : #n:nat. #m:nat. gt m n ->
      Ind(le:nat -> Set)
                   {le (S n); #m:nat. le m -> le (S m)}
                          m
le_gt_S
  : #n:nat. #m:nat. le n m ->
      Ind(le:nat -> Set)
                   {le (S n); #m:nat. le m -> le (S m)}
                          (S m)
```

– Μεταβατικότητα

```
le_gt_trans
  : #n:nat. #m:nat. #p:nat. le m n -> gt m p ->
      Ind(le:nat -> Set)
                    {le (S p); #m:nat. le m -> le (S m)}
                                n
gt_le_trans
  : #n:nat. #m:nat. #p:nat. gt n m -> le p m ->
      Ind(le:nat -> Set)
                    {le (S p); #m:nat. le m -> le (S m)}
                                n
gt_trans
  : #n:nat. #m:nat. #p:nat. gt n m -> gt m p ->
      Ind(le:nat -> Set)
                    {le (S p); #m:nat. le m -> le (S m)}
                                n
gt_trans_S
  : #n:nat. #m:nat. #p:nat. gt (S n) m -> gt m p ->
      Ind(le:nat -> Set)
```

```
                          {le (S p); #m:nat. le m -> le (S m)}
                                n
```

– Σύγκριση με το μηδέν.

```
gt_0_eq
  : #n:nat.
      Ind(X:Set){gt n 0 -> X; eq nat 0 n -> X}
```

– Απλοποίηση και Συμβατότητα.

```
plus_gt_reg_l
  : #n:nat. #m:nat. #p:nat. gt (plus p n) (plus p m) ->
      Ind(le:nat -> Set)
                    {le (S m); #m:nat. le m -> le (S m)}
                          n
plus_gt_compat_l
  : #n:nat. #m:nat. #p:nat. gt n m ->
      Ind(le:nat -> Set)
                    {le (S (plus p m)); #m:nat. le m -> le (S m)}
                          (plus p n)
```

• Εδώ ακολουθούν θεωρίες για την πρόσθεση στην αριθμητική Peano.

– Απόδειξη ότι μηδέν είναι ουδέτερο στοιχείο για την πρόσθεση.

```
plus_0_l
  : #n:nat. Ind(X:nat -> Set){X n} n
plus_0_r
  : #n:nat. Ind(X:nat -> Set){X (plus n 0)} n
```

– Μεταβατικότητα της πρόσθεσης

```
plus_comm
  : #n:nat. #m:nat.
      Ind(X:nat -> Set)
                 {X (plus n m)}
                          (plus m n)
```

– Προσεταιριστικότητα

```
plus_Snm_nSm
  : #n:nat. #m:nat.
      Ind(X:nat -> Set)
                    {X (S (plus n m))}
                                (plus n (S m))
plus_assoc
  : #n:nat. #m:nat. #p:nat.
      Ind(X:nat -> Set)
                    {X (plus n (plus m p))}
                                (plus (plus n m) p)
plus_permute
  : #n:nat. #m:nat. #p:nat.
      Ind(X:nat -> Set)
                    {X (plus n (plus m p))}
                                (plus m (plus n p))
plus_assoc_reverse
  : #n:nat. #m:nat. #p:nat.
      Ind(X:nat -> Set)
                    {X (plus (plus n m) p)}
                                (plus n (plus m p))
```

– Απλοποίηση

```
plus_reg_l
  : #m:nat. #p:nat. #n:nat. eq nat (plus n m) (plus n p) -> eq nat m p
plus_le_reg_l
  : #n:nat. #m:nat. #p:nat. le (plus p n) (plus p m) -> le n m
plus_lt_reg_l
  : #n:nat. #m:nat. #p:nat. lt (plus p n) (plus p m) -> lt n m
```

– Συμβατότητα στις διατάξεις.

```
plus_le_compat_l
  : #n:nat. #m:nat. #p:nat. le n m -> le (plus p n) (plus p m)

plus_le_compat_r
  : #n:nat. #m:nat. #p:nat. le n m ->
      Ind(le:nat -> Set)
                    {le (plus n p); #m:nat. le m -> le (S m)}
                           (plus m p)
le_plus_l
  : #n:nat. #m:nat. le n (plus n m)
le_plus_r
  : #n:nat. #m:nat.
      Ind(le:nat -> Set)
                    {le m; #m:nat. le m -> le (S m)}
                           (plus n m)
le_plus_trans
  : #n:nat. #m:nat. #p:nat. le n m ->
      Ind(le:nat -> Set)
                    {le n; #m:nat. le m -> le (S m)}
                           (plus m p)
lt_plus_trans
  : #n:nat. #m:nat. #p:nat. lt n m ->
      Ind(le:nat -> Set)
                    {le (S n); #m:nat. le m -> le (S m)}
                           (plus m p)
plus_lt_compat_l
  : #n:nat. #m:nat. #p:nat. lt n m -> lt (plus p n) (plus p m)
plus_lt_le_compat
  : #n:nat. #m:nat. #p:nat. #q:nat. le (S n) m -> le p q ->
      Ind(le:nat -> Set)
                    {le (plus (S n) p); #m:nat. le m -> le (S m)}
                           (plus m q)
plus_le_compat
  : #n:nat. #m:nat. #p:nat. #q:nat. le n m -> le p q ->
      Ind(le:nat -> Set)
                     {le (plus n p); #m:nat. le m -> le (S m)}
                                (plus m q)
plus_le_lt_compat
  : #n:nat. #m:nat. #p:nat. #q:nat. le n m -> le (S p) q ->
      Ind(le:nat -> Set)
                    {le (plus (S n) p); #m:nat. le m -> le (S m)}
                           (plus m q)
plus_lt_compat
  : #n:nat. #m:nat. #p:nat. #q:nat. lt n m -> lt p q ->
      Ind(le:nat -> Set)
                 {le (S (plus n p)); #m:nat. le m -> le (S m)}
                   (plus m q)
```

– Λήμματα αντιστροφής

```
plus_is_O
```

```
  : #m:nat. #m:nat.
      eq nat (plus m m) 0 -> and (eq nat m 0) (eq nat m 0)
plus_is_one
  :  #m:nat. #n:nat. eq nat (plus m n) (nat nat) ->
       sumbool (and (eq nat m nat) (eq nat n (nat nat)))
                   (and (eq nat m (nat nat)) (eq nat n nat))
```

– Παραγόμενες ιδιότητες

```
plus_permute_2_in_4
  : #m:nat. #n:nat. #p:nat. #q:nat.
      Ind(X:nat -> Set)
                  {X (plus (plus m n) (plus p q))}
                           (plus (plus m p) (plus n q))
```

– Αναδρομή-ουράς στην πρόσθεση: το **tail_plus** είναι ένας διαφορετικός ορισμός για το **plus** ο οποίος είναι αναδρομικός ως προς την ουρά, ενώ το **plus** δεν είναι.

```
plus_acc
  : nat -> nat -> nat
tail_plus
  : nat -> nat -> nat
plus_tail_plus
  : nat -> nat -> Set
```

- Στη συνέχεια, ακολουθούν οι θεωρίες για την αφαίρεση κατά την αριθμητική Peano.

  – Το μηδέν όταν βρίσκεται δεξιά στην αφαίρεση, είναι ουδέτερο στοιχείο.

```
minus_n_O
  : #n:nat.
      Ind(X:nat -> Set){X n}(minus n 0)
```

  – Μετάθεση με τον επόμενο

```
minus_Sn_m
  : #n:nat. #m:nat. le m n ->
      Ind(X:nat -> Set)
                    {X (S (minus n m))}
                           (minus (S n) m)
pred_of_minus
  : #x:nat.
      Ind(X:nat -> Set)
                  {X (pred x)}
                    (minus x (S 0))
```

  – Διαγωνιοποίηση

```
minus_n_n
  : #n:nat.
      Ind(X:nat -> Set){X 0}(minus n n)
```

  – Απλοποίηση

```
minus_plus_simpl_l_reverse
  : #n:nat. #m:nat. #p:nat.
      Ind(X:nat -> Set)
                    {X (minus n m)}
                           (minus (plus p n) (plus p m))
```

  – Συσχέτιση με το **plus**.

```
plus_minus
  : #n:nat. #m:nat. #p:nat. eq nat n (plus m p) -> eq nat p (minus n m)
minus_plus
  : #n:nat. #m:nat.
      Ind(X:nat -> Set) {X (minus (plus n m) n)} m


le_plus_minus
  : #n:nat. #m:nat. le n m ->
      Ind(X:nat -> Set) {X m} (plus n (minus m n))
le_plus_minus_r
  : #n:nat. #m:nat. le n m ->
      Ind(X:nat -> Set) {X (plus n (minus m n))} m

\item Συσχέτιση με τη διάταξη.
\begin{ncode}{1}
le_minus
  : #i:nat. #h:nat.
      Ind(le:nat -> Set)
                  {le (minus i h); #m:nat. le m -> le (S m)}
                    i
lt_minus
  : #n:nat. #m:nat. le m n -> lt O m -> lt (minus n m) n
lt_O_minus_lt
  : #n:nat. #m:nat. lt O (minus n m) -> lt m n
not_le_minus_O
  : #y:nat. #x:nat. not (le x y) -> eq nat (minus y x) O
```

- Βασικές θεωρίες της αριθμητικής Peano για τον πολλαπλασιασμό είναι οι:
  - Ιδιότητες του μηδέν για τον πολλαπλασιασμό.

```
mult_O_r
  : #n:nat. Ind(X:nat -> Set){X (mult n O)} O
mult_O_l
  : nat -> Ind(X:nat -> Set){X O} O
```

  - Επιμεριστική ιδιότητα

```
mult_plus_distr_r
  : #n:nat. #m:nat. #p:nat.
      Ind(X:nat -> Set)
                   {X (mult (plus n m) p)}
                          (plus (mult n p) (mult m p))
mult_plus_distr_l
  : #n:nat. #m:nat. #p:nat.
      eq nat (mult n (plus m p)) (plus (mult n m) (mult n p))
mult_minus_distr_r
  : #n:nat. #m:nat. #p:nat.
      eq nat (mult (minus n m) p) (minus (mult n p) (mult m p))
```

  - Προσεταιριστικότητα

```
mult_assoc_reverse
  : #n:nat. #m:nat. #p:nat.
      Ind(X:nat -> Set)
                    {X (mult (mult n m) p)}
                            (mult n (mult m p))
mult_assoc
  : #n:nat. #m:nat. #p:nat.
      Ind(X:nat -> Set)
```

```
                        {X (mult n (mult m p))}
                               (mult (mult n m) p)
```

– Μεταβατικότητα

```
mult_comm
  : #n:nat. #m:nat.
      Ind(X:nat -> Set){X (mult n m)} (mult m n)
```

– Απόδειξη ότι το ένα είναι το ουδέτερο στοιχείο του πολλαπλασιασμού.

```
mult_1_l
  : #n:nat. Ind(X:nat -> Set){X (plus n O)} n
mult_1_r
  : #n:nat. Ind(X:nat -> Set){X (mult n (S O))} n
```

– Συμβατότητα ως προς τη διάταξη.

```
mult_O_le
  : #n:nat. #m:nat.
      Ind(X:Set){eq nat m O -> X; le n (mult m n) -> X}


mult_le_compat_l
  : #n:nat. #m:nat. #p:nat.
      le n m -> le (mult p n) (mult p m)
mult_le_compat_r
  : #m:nat. #n:nat. #p:nat. le m n ->
      Ind(le:nat -> Set)
                    {le (mult m p); #m:nat. le m -> le (S m)}
                           (mult n p)
mult_le_compat
  : #m:nat. #n:nat. #p:nat. #q:nat. le m n -> le p q ->
      Ind(le:nat -> Set)
                    {le (mult m p); #m:nat. le m -> le (S m)}
                            (mult n q)
mult_S_lt_compat_l
  : #m:nat. #m:nat. #p:nat. lt m p -> lt (mult (S m) m) (mult (S m) p)
mult_lt_compat_r
  : #n:nat. #m:nat. #p:nat. lt n m -> lt O p -> lt (mult n p) (mult m p)
mult_S_le_reg_l
  : #m:nat. #n:nat. #p:nat. le (mult (S m) n) (mult (S m) p) ->
      Ind(le:nat -> Set)
                    {le n; #m:nat. le m -> le (S m)}
                           p
```

– Απόδειξη ότι τα $n| \to 2 * n$ και $n| \to 2n + 1$ είναι διακριτά μεταξύ τους.

```
odd_even_lem
  : #p:nat. #q:nat.
      not (eq nat (plus (mult (S (S O)) p) (S O)) (mult (S (S O)) q))
```

– Ιδιότητα της αναδρομής της ουράς για το mult. Το `tail_mult` είναι ένας διαφορετικός ορισμός για το **mult** ο οποίος είναι αναδρομικός ως προς την ουρά, ενώ το **mult** δεν είναι.

```
mult_acc
  : nat -> nat -> nat -> nat
mult_acc_aux
  :  #n:nat. #m:nat. #p:nat. eq nat (plus m (mult n p)) (mult_acc m p n)
tail_mult
  : nat -> nat -> nat
```

```
mult_tail_mult
  :  #n:nat. #m:nat. eq nat (mult n m) (tail_mult n m)
```

- Θεωρίες για σύγκριση συνόλων και για αναζήτηση ύπαρχξης όρων σε σύνολα.

```
between
  : (nat -> Set) -> nat -> nat -> Set
  := \P:nat -> Set. \k:nat.
      Ind(between:nat -> Set)
                    {between k; #l:nat. between l -> P l -> between (S l)}

bet_emp
  : #P:nat -> Set. #k:nat. between P k k
  := \P:nat -> Set. \k:nat. Constr(0, between P k)

bet_S
  : #P:nat -> Set. #k:nat. #l:nat. between P k l -> P l -> between P k (S l)
  := \P:nat -> Set. \k:nat. Constr(1, between P k)


between_ind
  : #P:nat -> Set. #k:nat. #P:nat -> Set. P k ->
      (#l:nat. between P k l -> P l -> P l -> P (S l)) -> #n:nat.
                  between P k n -> P n

bet_eq
  : #P:nat -> Set. #k:nat. #l:nat. eq nat l k -> between P k l
  := \P:nat -> Set. \k:nat. \l:nat. \H:eq nat l k.
       eq_ind nat l (\k':nat. between P k' l) (between P l) k H

between_le
  : #P:nat -> Set. #k:nat. #l:nat. between P k l -> le k l
  := \P:nat -> Set. \k:nat. \l:nat. \H:between P k l.
       between_ind P k (\l:nat. le k l)
                    (le_n k) (\l:nat. \H':between P k l. \IHbetween:le k l.
                           \H0:P l. le_S k l IHbetween) l H

between_Sk_l
  : #P:nat -> Set. #k:nat. #l:nat. between P k l -> le (S k) l ->
           between P (S k) l

between_restr
  : #P:nat -> Set. #k:nat. #l:nat. #m:nat. le k l -> le l m ->
      between P k m -> between P l m

exists_between
  : (nat -> Set) -> nat -> nat -> Set

exists_S
  : #Q:nat -> Set. #k:nat. #l:nat. exists_between Q k l ->
                exists_between Q k (S l)

exists_le
  : #Q:nat -> Set. #k:nat. #l:nat. le k l -> Q l ->
       exists_between Q k (S l)

exists_between_ind
  : #Q:nat -> Set. #k:nat. #P:nat -> Set.
      (#l:nat. exists_between Q k l -> P l -> P (S l)) ->
```

```
                 (#l:nat. le k l -> Q l ->  P (S l)) -> #n:nat.
                   exists_between Q k n -> P n

exists_le_S
  : #Q:nat -> Set. #k:nat. #l:nat. exists_between Q k l -> le (S k) l

exists_lt
  : #Q:nat -> Set. #k:nat. #l:nat. exists_between Q k l -> lt k l

exists_S_le
  : #Q:nat -> Set. #k:nat. #l:nat. exists_between Q k (S l) -> le k l

in_int
  : nat -> nat -> nat -> Set

in_int_intro
  : #p:nat. #q:nat. #r:nat. le p r -> lt r q -> in_int p q r

in_int_lt
  : #p:nat. #q:nat. #r:nat. in_int p q r -> lt p q

in_int_p_Sq
  : #p:nat. #q:nat. #r:nat. in_int p (S q) r ->
      or (in_int p q r) (eq nat r q)

in_int_S
  : #p:nat. #q:nat. #r:nat. in_int p q r -> in_int p (S q) r

in_int_Sp_q
  : #p:nat. #q:nat. #r:nat. in_int (S p) q r -> in_int p q r

between_in_int
  : #P:nat -> Set. #k:nat. #l:nat. between P k l -> #r:nat.
      in_int k l r -> P r

in_int_between
  : #P:nat -> Set. #k:nat. #l:nat. le k l -> (#r:nat.
      in_int k l r -> P r) -> between P k l

exists_in_int
  : #Q:nat -> Set. #k:nat. #l:nat. exists_between Q k l ->
      ex2 nat (\m:nat. in_int k l m) (\m:nat. Q m)

in_int_exists
  : #Q:nat -> Set. #k:nat. #l:nat. #r:nat. in_int k l r -> Q r ->
      exists_between Q k l

between_or_exists
  : #P:nat -> Set. #Q:nat -> Set. #k:nat. #l:nat. le k l ->
     (#n:nat. in_int k l n -> or (P n) (Q n)) -> or (between P k l)
              (exists_between Q k l)

between_not_exists
  : #P:nat -> Set. #Q:nat -> Set. #k:nat. #l:nat. between P k l ->
      (#n:nat. in_int k l n -> P n -> not (Q n)) ->
                  not (exists_between Q k l)

P_nth
```

```
  : (nat -> Set) -> (nat -> Set) -> nat -> nat -> nat -> Set

nth_O
  : #P:nat -> Set. #Q:nat -> Set. #init:nat. P_nth P Q init init O

nth_S
  : #P:nat -> Set. #Q:nat -> Set. #init:nat. #k:nat. #l:nat. #n:nat.
      P_nth P Q init k n -> between P (S k) l -> Q l ->
                  P_nth P Q init l (S n)

P_nth_ind
  : #P:nat -> Set. #Q:nat -> Set. #init:nat. #P:nat -> nat -> Set.
      P init O -> (#k:nat. #l:nat. #n:nat. P_nth P Q init k n ->
                  P k n -> between P (S k) l -> Q l -> P l (S n)) ->
                    #n:nat. #n:nat. P_nth P Q init n n -> P n n

nth_le
  : #P:nat -> Set. #Q:nat -> Set. #init:nat. #l:nat. #n:nat.
      P_nth P Q init l n -> le init l

eventually
  : (nat -> Set) -> nat -> Set

event_O
  : #Q:nat -> Set. eventually Q O -> Q O
```

- Θεωρίες για την υπολογισιμότητα αναζήτησης στοιχείων σε σύνολο.

```
O_or_S
  : #n:nat. sumor (sig nat (\m:nat. eq nat (S m) n)) (eq nat O n)
eq_nat_dec
  : #n:nat. #m:nat. sumbool (eq nat n m) (not (eq nat n m))
dec_eq_nat
  : #n:nat. #m:nat. decidable (eq nat n m)
```

- Βασικές θεωρίες για τη σύγκριση αριθμών, στην αριθμητική Peano, είναι οι παρακάτω:
  - Σύγκριση αριθμών κατά Peano:

```
zerop
  : #n:nat. sumbool (eq nat n O) (lt O n)
lt_eq_gt_dec
  : #n:nat. #m:nat. sumor (sumbool (lt n m) (eq nat n m)) (lt m n)
gt_eq_gt_dec
  : #n:nat. #m:nat. sumor (sumbool (gt m n) (eq nat n m)) (gt n m)
le_lt_dec
        : #n:nat. #m:nat. sumbool (le n m) (lt m n)
le_le_S_dec
  : #n:nat. #m:nat. sumbool (le n m) (lt m n)
le_ge_dec
  : #n:nat. #m:nat. sumbool (le n m) (ge n m)
le_gt_dec
  : #n:nat. #m:nat. sumbool (le n m) (lt m n)
le_lt_eq_dec
  : #n:nat. #m:nat. le n m -> sumbool (lt n m) (eq nat n m)
```

  - Αποδείξεις για την υπολογισιμότητα:

```
dec_le
  : #x:nat. #y:nat. decidable (le x y)
dec_lt
  : #x:nat. #y:nat. decidable (lt x y)
dec_gt
  : #x:nat. #y:nat. decidable (gt x y)
dec_ge
  : #x:nat. #y:nat. decidable (ge x y)
not_eq2
  :  #n:nat. #m:nat. not (eq nat n m) -> or (lt n m) (lt m n)
not_le2
  : #x:nat. #y:nat. not (le x y) -> gt x y
not_gt2
  : #x:nat. #y:nat. not (gt x y) -> le x y
not_ge2
  : #x:nat. #y:nat. not (ge x y) -> lt x y
not_lt2
  : #x:nat. #y:nat. not (lt x y) -> ge x y
```

- Θεωρίες για το παραγοντικό, στην αριθμητική Peano:

```
fact
   : nat -> nat
   := \n:nat.
      Elim[\n:nat. nat]
          (n:nat)
      {
          S 0;
          \n':nat. \n_z:nat. mult (S n') n_z
      }

fact_le
  : #n:nat. #m:nat. le n m -> le (fact n) (fact m)

fact_neq_0
  : #n:nat. eq nat (fact n) 0 -> not_eq nat 0 (fact n) -> eq nat 0 (fact n)

lt_0_fact
  : #n:nat. lt 0 (fact n)
```

Κεφάλαιο 5

Συμπεράσματα

5.1 Συνεισφορά

Η συνεισφορά και τα συμπεράσματα της παρούσας εργασίας συνοψίζονται στα εξής:

- Μελετήθηκε ο λάμβδα λογισμός, οι παραλλαγές του με τύπους και τα αντίστοιχα συστήματα λογικής. Μελετήθηκε επίσης και η σχέση που έχουν μεταξύ τους τα συστήματα λάμβδα λογισμού (ή λογικής) ως προς τη θέση τους στο λάμβδα κύβο (ή στον κύβο λογικής αντίστοιχα).

- Μελετήθηκε η γλώσσα τύπων nflint σε σύγκριση με το σύστημα αποδείξεων Coq. Έγινε προσπάθεις να αναλυθούν και να γεφυρωθούν οι διαφορές τους, ώστε το σύστημα Coq να αποτελέσει τη βάση για την ανάπτυξη βιβλιοθηκών απο θεωρίες για τη γλώσσα τύπων nflint.

- Αναπτύχθηκε ένα πρόγραμμα που μεταφράζει θεωρίες του Coq στη γλώσσα τύπων nflint. Με το πρόγραμμα αυτό μεταφράστηκαν σωστά ένα μεγάλο μέρος των θεωριών του Coq όπου ορίζονται οι βασικοί τύποι, η προτασιακή και κατηγορηματική λογική και η αριθμητική Peano.

- Το μέρος των θεωριών όπου το σύστημα αυτόματης μετάφρασης απέτυχε μεταφράστηκε με το χέρι. Επί του συνόλου των θεωριών έγιναν διορθώσεις και συμπληρώσεις.

5.2 Μελλοντική έρευνα

Στόχος της ευρύτερης ερευνητικής προσπάθειας είναι η δημιουργία ενός ολοκληρωμένου συστήματος ελέγχου της αξιοπιστίας εκτελέσιμου κώδικα. Σκοπός ενός τέτοιου συστήματος θα είναι ο έλεγχος του κώδικα και των ιδιοτήτων του, αφού πρώτα μεταφραστούν στην κατάλληλη τυπική λογική.

Με τη δημιουργία μιας βιβλιοθήκης θεωριών, γίνεται ένα βήμα για την ολοκλήρωση του στόχου της δημιουργίας ενός τέτοιου συστήματος. Η μελλοντική εργασία για την ολοκλήρωση ενός τέτοιου συστήματος περιλαμβάνει:

- Να επεκταθεί η βιβλιοθήκη με άλλες χρήσιμες θεωρίες, όπως η δυαδική αριθμητική φυσικών και ακεραίων αριθμών, η αριθμητική πραγματικών, η θεωρία συνόλων, κ.λπ.

- Να οριστεί και να υλοποιηθεί μια κατάλληλη γλώσσα υπολογισμών. Η γλώσσα αυτή θα αποτελεί την ενδιάμεση γλώσσα στην οποία θα μεταγλωττίζονται όλες οι δομικές μονάδες της αρχικής γλώσσας προγραμματισμού.

- Να οριστεί και να υλοποιηθεί κατάλληλα μια γλώσσα προγραμματισμού υψηλού επιπέδου, η οποία θα εκμεταλλεύεται τη γλώσσα τύπων και θα μεταφράζεται στη γλώσσα υπολογισμών.

- Να γίνει αναδιοργάνωση και επέκταση της βιβλιοθήκης θεωριών που αναπτύχθηκε στην παρούσα εργασία, ώστε να ανταποκρίνεται καλύτερα στις ανάγκες κατασκευής αποδείξεων μέσα σε προγράμματα.
- Σημαντικό είναι επίσης να βελτιωθεί η υλοποίηση της γλώσσας τύπων που παρουσιάστηκε σε αυτή την εργασία. Όπως φάνηκε και από τις μετρήσεις απόδοσης, σημαντικά περιθώρια βελτίωσης υπάρχουν στον αλγόριθμο που εκτελεί τους υπολογισμούς. Τέλος απαραίτητη είναι η ανάπτυξη ενός αλγορίθμου χειρισμού των σφαλμάτων, έτσι ώστε να γίνονται αντιληπτά από τον χρήστη με ευκολία.

Βιβλιογραφία

[Bare93] H. P. Barendregt, "Lambda Calculi with Types", in S. Abramsky, D.M. Gabbay and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, vol. 2, Oxford University Press, 1993.

[Bare01] H. Barendregt and H. Geuvers, "Proof-Assistants Using Dependent Type Systems", in A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, vol. II, chapter 18, pp. 1149–1238, Elsevier Science Publishers, B.V., 2001.

[CDev03] The Coq Development Team, "The Coq Proof Assistant Reference Manual", Version 7.4, URL: `http://coq.inria.fr/`, February 2003.

[Chai04] Emmanuel Chailloux, Pascal Manoury and Bruno Pagano, *Developing Applications With Objective Caml*, Éditions O'REILLY, 18 rue Séguier, 75006 Paris, FRANCE, 2004. Translated by Francisco Albacete, Mark Andrew, Martin Anlauf, Christopher Browne, David Casperson, Gang Chen, Harry Chomsky, Ruchira Datta, Seth Delackner, Patrick Doane, Andreas Eder, Manuel Fahndrich, Joshua Guttman, Theo Honohan, Xavier Leroy, Markus Mottl, Alan Schmitt, Paul Steckler, Perdita Stevens, Francois Thomasset.

[Ende01] Herbert B. Enderton, *A Mathematical Introduction to Logic*, Harcourt/Academic Press, 2001.

[Freg80] Friedrich Ludwig Gottlob Frege, *The Foundations of Arithmetic: A logico-mathematical enquiry into the concept of number*, Northwestern University Press, Evanston, Illinois, 1980. Translated by J. L. Austin.

[Geuv95] H. Geuvers, "The Calculus of Constructions and Higher Order Logic", in Ph. De Groote, editor, *The Curry-Howard Isomorphism*, vol. 8 of *Cahiers du Centre de Logique*, pp. 139–191, Université Catholique de Louvain, Academia, Louvain-la-Neuve, Belgium, 1995.

[Huet04] Gérard Huet, Gilles Kahn and Christine Paulin-Mohring, "The Coq Proof Assistant: A Tutorial", Coq Version 8, URL: `http://coq.inria.fr/doc/tutorial.html`, February 2004.

[Papa03] Nikolaos Papaspyrou, Dimitrios Vytiniotis and Vasileios Koutavas, "Logic-enhanced type systems: Programming language support for reasoning about security and other program properties", in *4th Panhellenic Logic Symposium (PLS04)*, p. foo, Thessaloniki, Greece, July 2003.

[Pier89] Benjamin Pierce, Scott Dietzen and Spiro Michaylov, "Programming in Higher-order Typed Lambda-Calculi", Technical Report CMU-CS-89-111, Carnegie Mellon Univeristy, March 1989.

[Shao02] Z. Shao, B. Saha, V. Trifonov and N. Papaspyrou, "A Type System for Certified Binaries", in *Proceedings of the 29th Annual Symposium on Principles of Programming Languages (POPL 2002)*, pp. 217–232, Portland, OR, USA, January 2002.

[Wern94] B. Werner, *Une Théorie des Constructions Inductives*, Ph.D. thesis, Université Paris VII, Paris, France, May 1994.

[Κολέ04] Γιώργος Κολέτσος, "Μαθηματική Λογική", Σημειώσεις για το μάθημα της Μαθηματικής Λογικής, Σχολή Ηλεκτρολόγων Μηχ. και Μηχ. Υπολογιστών, Εθνικό Μετσόβιο Πολυτεχνείο, 2004.

[Παπα04] Μιχάλης Α. Παπακυριάκου, "Διπλωματική Εργασία: Υλοποίηση μίας γλώσσας τύπων με εφαρμογές στην πιστοποίηση εκτελέσιμου κώδικα", Οκτώβριος 2004. Εργαστήριο Τεχνολογίας Λογισμικού.

[Ροντ03] Πάνος Ροντογιάννης και Νίκος Παπασπύρου, "Θεωρία Γλωσσών Προγραμματισμού", Σημειώσεις για το μάθημα Γλώσσες Προγραμματισμού ΙΙ, Σχολή Ηλεκτρολόγων Μηχ. και Μηχ. Υπολογιστών, Εθνικό Μετσόβιο Πολυτεχνείο, 2003.