λ-Λογισμός κι εφαρμογές του στις γλώσσες προγραμματισμού

Ευθύμης Πετρόπουλος-Τράκας
Ορέστης Ροδίτης-Κουτσαντώνης

Μελέτη του λ-λογισμού και της επίδρασής σου στο σχεδιασμό και ανάπτυξη των γλωσσών προγραμματισμού και τα συστήματα τύπων τους.

λ-Λογισμός

Ο λ-λογισμός είναι ένα τυπικό σύστημα της Μαθηματικής Λογικής για την έκφραση της υπολογισιμότητας με βάση την αφαίρεση και σύνθεση συναρτήσεων, με τη χρήση δέσμευσης μεταβλητών κι αντικατάστασης τους. Η δημιουργία του αποδίδεται στον Alonso Church το 1930 και τον Stephen Kleene, ο οποίος απέδειξε την λογική του ασυνέπεια το 1935. Το 1936, ο Church, απομονώνοντας το κομμάτι που αφορά την υπολογισιμότητα δημιούργησε αυτό που είναι τώρα γνωστό ως λ-λογισμός χωρίς τύπους και αργότερα, το 1940, τον λ-λογισμό με τύπους. Με τη χρήση του λ-λογισμού ο Church έδωσε αρνητική απάντηση στο πρόβλημα απόφασης του David Hilbert και όρισε τις υπολογίσιμες συναρτήσεις. Πλέον αποτελεί ένα καθολικό μοντέλο υπολογισιμότητας που μπορεί να χρησιμοποιηθεί για να προσομοιώσει οποιαδήποτε μηχανή Turing.

Δομή των εκφράσεων στο λ-λογισμό

- μεταβλητές x_1 , x_2 ... x_n
- τα σύμβολα αφαίρεσης λ και .
- παρενθέσεις $( )$

Το σύνολο των εκφράσεων, Λ, μπορεί να οριστεί αναδρομικά:

- Αν x είναι μεταβλητή, τότε $x \in \Lambda$
- Αν x είναι μεταβλητή και $M \in \Lambda$, τότε $(\lambda x . M) \in \Lambda$
- Αν $M, N \in \Lambda$, τότε $(M N) \in \Lambda$

λ-Λογισμός

Όπως αναφέραμε παραπάνω, ο λ-λογισμός είναι ισοδύναμος με ένα Turing πλήρες υπολογιστικό μοντέλο και αποτελεί τη μικρότερη δυνατή καθολική γλώσσα προγραμματισμού. Παρόλα αυτά, ο λ-λογισμός είναι μια τυποποίηση της υπολογισιμότητας που δίνει έμφαση στη χρήση κανόνων μετασχηματισμού, και δεν ενδιαφέρεται για τη μηχανή που τους υλοποιεί. Σχετίζεται περισσότερο με το λογισμικό και έχει αποτελέσει τη βάση του δηλωτικού συναρτησιακού προγραμματισμού.

Στον λ-λογισμό κάθε έκφραση αποτελεί μια ανώνυμη συνάρτηση, ενός μόνο ορίσματος και επιστρέφει μια μοναδική τιμή. Δεν υπάρχουν συναρτήσεις πολλών ορισμάτων, αλλά μερική εφαρμογή μιας συνάρτησης σε μια άλλη (Curry-ing) και σύνθεσή τους.

Αντικατάσταση:

Η αντικατάσταση (substitution), η οποία συμβολίζεται με $E[V := E']$, αντικαθιστά μία μεταβλητή V από την έκφραση E′ σε κάθε σημείο που η V είναι ελεύθερη στην E. Η αντικατάσταση σε όρους του λογισμού λάμδα ορίζεται αναδρομικά στη δομή των όρων, ως εξής:

- $x[x := N] \equiv N$
- $y[x := N] \equiv y$, αν $x \neq y$
- $(M1\ M2)[x := N] \equiv (M1[x := N])(M2[x := N])$
- $(\lambda y.M)[x := N] \equiv \lambda y\ (M[x := N])$, αν $x \neq y$ και $y \notin FV(N)$

λ-Λογισμός

α-μετατροπή:

Η α-μετατροπή επιτρέπει την αλλαγή ονόματος σε δεσμευμένες μεταβλητές. Π.χ. η λx . x γίνεται λy . y . Συχνά, σε πολλές χρήσεις του λ-λογισμού, όροι που διαφέρουν μόνο κατά μία α-μετατροπή θεωρούνται ισοδύναμοι.

Κατά την α-μετατροπή μίας αφαίρεσης, μετονομάζονται μόνο εκείνες οι εμφανίσεις μεταβλητών που δεσμεύονται από την ίδια αφαίρεση. Π.χ. μία α-μετατροπή της έκφρασης λx . λx . x είναι η λy . λx . x, αλλά όχι η λy . λx . y.

Η α-μετατροπή δεν είναι δυνατή αν θα προκαλούσε τη δέσμευση μίας μεταβλητής από διαφορετική αφαίρεση. Π.χ. αν αντικαταστήσουμε το x με y στην έκφραση λx . λy . x , παίρνουμε λy . λy . y .

β-αναγωγή:

Η β-αναγωγή εκφράζει την έννοια της εφαρμογής συνάρτησης. Η αναγωγή βήτα της έκφρασης ((λV . E)E') είναι απλά E[V := E'].

η-μετατροπή:

Η η-μετατροπή εκφράζει την έννοια της εκτατικότητας (extensionality), που σε αυτό το πλαίσιο είναι ότι δύο συναρτήσεις είναι ίδιες αν και μόνο αν δίνουν το ίδιο αποτέλεσμα για όλα τα ορίσματα. Η μετατροπή ήτα μετατρέπει την έκφραση λx . fx σε f και αντίστροφα, όταν η μεταβλητή x δεν εμφανίζεται ελεύθερη μέσα στην f.

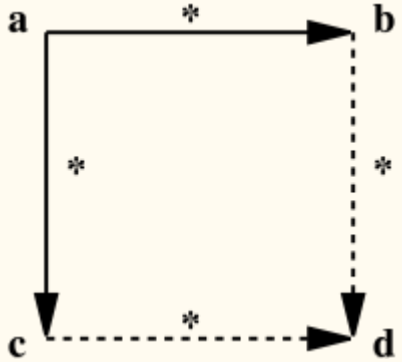
λ-Λογισμός

β-κανονική μορφή:

Στον λ-λογισμό, η β-κανονική μορφή είναι η μορφή μια έκφρασης στην οποία δεν μπορούν να εφαρμοστούν επιπλέον β-αναγωγές. Ένας όρος βρίσκεται σε βη-κανονική μορφή, αν δεν μπορούν να εφαρμοστούν ούτε β-αναγωγή ούτε η-μετατροπή σε αυτόν. Στο λ-λογισμό ένας β-redex, είναι ένας όρος στη μορφή (λx . A) M.

Θεώρημα Church-Rosser:

Σύμφωνα με το θεώρημα Church-Rosser, αν η επαναλαμβανόμενη εφαρμογή των βημάτων αναγωγής κάποια στιγμή τελειώσει, τότε θα έχουμε παράγει μια β-κανονική μορφή. Η σειρά με την οποία θα εφαρμοστούν οι κανόνες αυτοί στην έκφραση, δεν παίζει ρόλο στην κανονική μορφή στην οποία θα μετασχηματιστεί τελικά, αν υπάρχει. Επομένως, σύμφωνα με αυτό το θεώρημα, κάθε έκφραση έχει το πολύ μία μοναδική κανονική μορφή.



λ-Λογισμός

Ισομορφισμός Curry-Howard

Ο ισομορφισμός Curry-Howard εκφράζει μια αντιστοιχία μεταξύ των αποδείξεων στον προτασιακό λογισμό και των όρων του λ-λογισμού με τύπους και πιο συγκεκριμένα μια αντιστοιχία στον τρόπο που μια έκφραση του προτασιακού λογισμού αποδεικνύεται από μια θεωρία, και στον τρόπο που μια έκφραση του λ-λογισμού προκύπτει ως β-ισοδύναμη από μια άλλη έκφραση του λ-λογισμού. Ουσιαστικά, μια απόδειξη μιας φόρμουλας φ αντιστοιχεί σε έναν όρο τύπου φ του λ-λογισμού με τύπους, και η τελική φόρμουλα φ αντιστοιχεί στον τελικό τύπο φ της έκφρασης.

Church-Turing Υπόθεση:

Στην επιστήμη των υπολογιστών, η υπόθεση Church-Turing, αποτελεί μια υπόθεση για τη φύση των υπολογίσιμων συναρτήσεων. Αναφέρει ότι μια συνάρτηση στο πεδίο των φυσικών αριθμών είναι υπολογίσιμη από έναν άνθρωπο ακολουθώντας έναν αλγόριθμο, αγνοώντας περιορισμούς πόρων, αν και μόνο αν είναι υπολογίσιμη από μια μηχανή Turing. Ο Church και ο Turing, απέδειξαν ότι μια συνάρτηση είναι λ-υπολογίσιμη αν και μόνο αν είναι υπολογίσιμη από μια μηχανή Turing και είναι γενικά αναδρομική.

λ-Λογισμός

Συνδυαστές Σταθερού σημείου

Στην επιστήμη των υπολογιστών, ένας συνδυαστής σταθερού σημείου (fixed-point combinator), αποτελεί μια συνάρτηση fix τέτοια ώστε για κάθε συνάρτηση f γυρίζει ένα σταθερό σημείο x αυτής της συνάρτησης. Σταθερό σημείο συνάρτησης ονομάζουμε μια τιμή η οποία όταν εφαρμοσθεί σαν είσοδος στη συνάρτηση, γυρίζει τον εαυτό της σαν αποτέλεσμα. Με άλλα λόγια, ο συνδυαστής fix, όταν εφαρμοσθεί σε μια συνάρτηση f, επιστρέφει το ίδιο αποτέλεσμα όπως η εφαρμογή της f στο αποτέλεσμα της εφαρμογής της fix στην f.

$$x = fx$$

Αναδρομή

Η σχέση που ικανοποιεί συνεπώς είναι η:

$$\text{fix } f = f\ (\text{fix } x),\ \forall\ f$$

Ο πιο γνωστός τέτοιος συνδυαστής, που μας ενδιαφέρει άμεσα καθώς ορίζει την αναδρομή στο λ-λογισμό είναι ο Y-Combinator του Curry ο οποίος ορίζεται ως εξής:

$$Y = \lambda f\ .\ (\lambda x\ .\ f(x\ x))\ (\lambda x\ .\ f(x\ x))$$

Θεωρία τύπων

Εμπλουτισμός του λ-λογισμού με τύπους

Κίνητρο: Η επιβολή περιορισμών στα προγράμματα, σε αντιστοιχία με περιπτώσεις του πραγματικού κόσμου

Διάφορες εκδοχές του λ-λογισμού με τύπους

Πιο κύριες:

- Σύστημα Church
- Σύστημα Curry

Θεωρία τύπων: Σύστημα Curry

- Προσθήκη τύπων σε υπάρχοντες όρους του λ-λογισμού
- Βασικές έννοιες: Σύνολο βασικών τύπων U
- σ,τ τύποι : σ → τ τύπος (για συναρτήσεις!)
- Σύνολο Γ από ζευγάρια $(x_i:t_i)$ - μεταβλητή x έχει τύπο t
- Τύπος ολόκληρου όρου συμπεραίνεται μέσω επαγωγικών κανόνων - οι τύποι των μεταβλητών αρκούν για την επαγωγή
- Αν ο όρος επιδέχεται τύπο!

Θεωρία τύπων: Σύστημα Church

- Ο όρος του λ-λογισμού με τύπους a la Church δημιουργείται ταυτόχρονα με την τυποποίησή του
- Σε κάθε στάδιο κατασκευής του ο όρος φέρει τύπο, και μόνο σωστοί τύποι μπορούν να συνθέσουν νέο τύπο!
- Η τυποποίηση είναι κατά τα άλλα παρόμοια με του Curry
- Αφού ο όρος σίγουρα έχει τύπο, μπορεί με ασφάλεια να απο-τυποποιηθεί ώστε να προκύψει όρος του καθαρού λ-λογισμού - αναδρομικά
- Ελαφρά τροποποίηση ισοδυναμιών όρων και θεωρημάτων, παίρνοντας υπόψιν τους τύπους, ώστε να ισχύουν
- Απονομή τύπων ισοδύναμη με διαμέριση μεταβλητών στον καθαρό λ-λογισμό!

Θεωρία τύπων: Άλλες επεκτάσεις

- Ποικίλες άλλες επεκτάσεις
- Απόδοση επιπλέον δυνατοτήτων μέσω αυτών (ορισμός Boolean τύπων, Nat, ορισμός αριθμητικών τύπων κ.α.)

Δηλωτικός Προγραμματισμός

Στην επιστήμη των υπολογιστών ο δηλωτικός προγραμματισμός (declarative programming) αποτελεί ένα προγραμματιστικό υπόδειγμα, στο οποίο ο υπολογισμός του ζητούμενου υπολογίζεται περιγράφοντας τις επιθυμητές ιδιότητες του. Σε αντίθεση με τον προστακτικό προγραμματισμό, μπορούμε να πούμε ότι περιγράφουμε το “τι” επιθυμούμε να υπολογίσουμε, κι όχι το “πως” θέλουμε να υπολογιστεί κάτι. Γνωστές γλώσσες προγραμματισμού δηλωτικού υποδείγματος, είναι η SQL, η Haskell, η LISP, η SML, η Erlang, η Prolog, οι κανονικές εκφράσεις (regular expressions), η CSS και πολλές ακόμη.

- Constraint
  - Constraint Logic
    - Concurrent constraint logic
- Dataflow
  - Flow-based
  - Cell-oriented (spreadsheets)
  - Reactive
- Functional
  - Functional Logic
  - Purely Functional
- Logic
  - Abductive Logic
  - Answer Set
  - Concurrent Logic
  - Functional Logic
  - Inductive Logic

Δηλωτικός Προγραμματισμός

Αρκετές γλώσσες προγραμματισμού σήμερα, υποστηρίζουν περισσότερα του ενός υποδείγματα. Τέτοιες γλώσσες είναι οι SQL, Prolog, C++, Python και άλλες.
Συγκεκριμένα για τη C++ με τη χρήση προτύπων μεταβλητού αριθμού παραμέτρων (variadic templates) μπορεί κανείς να επιτύχει δηλωτικό προγραμματισμό.

Επίσης με τη χρήση δηλωτικού προγραμματισμού και προτύπων της C++, είναι δυνατόν να υλοποιήσουμε μια Turing πλήρη μετα-γλώσσα προγραμματισμού, η οποία υπολογίζει όσα της βάζουμε κατά τη διαδικασία μεταγλώττισης. Η Turing πληρότητα των προτύπων της C++, προκύπτει σαν ατύχημα από τη σύγκρουση κι συγχώνευση δύο χαρακτηριστικών: από την προτυποποίηση και την ειδίκευση προτύπων. Αυτά τα δύο χαρακτηριστικά επιτρέπουν στα πρότυπα της C++ να δρουν σαν μια μη τυποποιημένη γλώσσα επανεγγραφής.

Δηλωτικός Προγραμματισμός

Μια ενδιαφέρουσα περίπτωση εφαρμογής δηλωτικού προγραμματισμού σε γλώσσες με προστακτικό χαρακτήρα είναι η χρήση της γλώσσας Python. Η Python είναι μια πλήρης γλώσσα κατά βάση προστακτικού προγραμματισμού, αντικειμενοστραφής, με ισχυρά χαρακτηριστικά και δυναμικό έλεγχο τύπων.

Είναι εφοδιασμένη με ποικίλες βιβλιοθήκες, οι οποίες είναι γραμμένες σε γλώσσες χαμηλού επιπέδου και οι οποίες έχουν την ικανότητα να επικοινωνούν με προγράμματα γραμμένα σε καθαρή Python, με χαρακτηριστικό παράδειγμα τις βιβλιοθήκες που χρησιμοποιούνται τα τελευταία χρόνια για Data Analysis και Machine Learning. Ο κώδικας γραμμένος σε Python αποφεύγει να πραγματοποιεί υπολογισμούς και αρκεί να λειτουργεί σαν “κόλλα” ανάμεσα στα πιο γρήγορα κομμάτια της εφαρμογής, δηλώνοντας τους υπολογισμούς που επιθυμεί οι συνδεδεμένες βιβλιοθήκες να κάνουν.

Συναρτησιακός προγραμματισμός

- Υπόδειγμα προγραμματισμού
- Υπολογισμός ισούται με αποτίμηση μαθηματικών συναρτήσεων
- Αποφυγή μεταβλητής κατάστασης, παρενεργειών
- Υποκατηγορία δηλωτικού προγραμματισμού
- Διαφάνεια αναφοράς
- Σημαντική διευκόλυνση κατανόησης και ελέγχου ορθότητας προγράμματος
- Θεωρητική θεμελίωση μέσω λ-λογισμού (συναρτήσεις ως δεδομένα, currying...), με τύπους και χωρίς!

Συναρτησιακός προγραμματισμός:Optimizations

- Μια συνάρτηση το αποτέλεσμα της οποίας δε χρησιμοποιείται μπορεί με ασφάλεια να αγνοηθεί
- Σταθερό αποτέλεσμα συνάρτησης για ορίσματα ίδιας τιμής
  (αν δεν έχουν παρενέργειες...)

- Δύο εκφράσεις χωρίς εξαρτήσεις δεδομένων εκτελούνται με οποιαδήποτε σειρά ή/και παράλληλα
- Κάθε στρατηγική αποτίμησης μπορεί να χρησιμοποιηθεί χωρίς να επηρεαστεί η ορθότητα του προγράμματος
  (πάλι, αν δεν υπάρχουν παρενέργειες)

Συναρτησιακός προγραμματισμός: Αναδρομή

- Ελλείψει προστακτικού μοντέλου, οι επαναλήψεις υλοποιούνται με αναδρομή
  - ισοδύναμες σύμφωνα με λ-λογισμό!
- Άπειρα επίπεδα αναδρομής → Turing completeness ΑΛΛΑ μαζί με τα αρνητικά - περιλαμβάνει και προγράμματα που δεν τερματίζουν!
- Αντιμετώπιση σε κάποιες περιπτώσεις με περιορισμό επιπέδων αναδρομής (απολύτως συναρτησιακός προγραμματισμός)

Συναρτησιακός προγραμματισμός:Αποτίμηση

- Αυστηρή αποτίμηση: Οι εκφράσεις αποτιμώνται εξ’ ολοκλήρου όταν συναντώνται στην εκτέλεση
- Οκνηρή αποτίμηση: Οι εκφράσεις αποτιμώνται όταν ζητηθεί το αποτέλεσμά τους.

Θεωρία Κατηγοριών

Η Θεωρία Κατηγοριών είναι το πεδίο εκείνο των μαθηματικών που εξετάζει τις γενικές ιδιότητες και χαρακτηριστικά των διαφόρων μαθηματικών δομών μέσα από την μελέτη σχέσεων μεταξύ αντικειμένων αυτών των δομών.
Η θεωρία κατηγοριών χρησιμοποιείται για να τυποποιήσει τα μαθηματικά και τις έννοιές της ως συλλογή των αντικειμένων και των βελών.

Ως κατηγορία ορίζουμε λοιπόν τη μαθηματική οντότητα η οποία έχει κάποιες επιθυμητές ιδιότητες και δομή:

i. Μια συλλογή αντικειμένων A,B,...
ii. Μια συλογή μορφισμών / βελών f,g,...
iii. $f: A \to B$, $\Pi^{\circ}(f)$, $\Pi\, T\, (f)$
iv. $\circ$: σύνθεσης: $f: A \to B$, $g: B \to C$, $g \circ f: A \to C$
    - προσεταιριστική ιδιότητα

      $f: \alpha \to \beta$
      $g: \beta \to \gamma$ $\quad h \circ (g \circ f) = (h \circ g) \circ f$
      $h: \gamma \to \delta$
    - ταυτότητα: Για κάθε αντικείμενο X, υπάρχει ένας μοναδικός μορφισμός:
      $\forall\ A \in C, id_A: A \to A$

Θεωρία Κατηγοριών

Μορφισμοί (Βέλη)

Ένας μορφισμός φ: α → β είναι :

- μονομορφισμός (ή αμφί)
- επιμορφισμός (ή επί)
- ομομορφισμός
- ενδομορφισμός
- αυτομορφισμός
- retraction
- section

Κάθε retraction είναι ένας επιμορφισμός, και κάθε section είναι μονομορφισμός. Επιπλέον, οι ακόλουθες τρεις δηλώσεις είναι ισοδύναμες:

- f είναι μονομορφισμος και retraction
- f είναι επιμορφισμος και section
- f είναι ισομορφισμος

Συναρτητές

Οι συναρτητές αποτελούν κατηγορία μορφισμών μεταξύ κατηγοριών κι έχουν τις εξής ιδιότητες:

- Διατηρούν τα ταυτοτικά βέλη (identity arrows)
- Διατηρούν τη σύνθεση των σύνθετων βελών-μορφισμών (composible arrows)

$$A \to^{f} B \mapsto^{\text{covariant}} FA \to^{F(f)} FB$$

$$A \to^{f} B \mapsto^{\text{contravariant}} FB \to^{F(f)} FA$$

Τα Functors δομούν-συντηρούν τους χάρτες μεταξύ των κατηγοριών. Μπορούν να θεωρηθούν ως μορφισμοί στην κατηγορία όλων των κατηγοριών.

Θεωρία Κατηγοριών

Μονοειδές (Monoid)

Στη θεωρία κατηγοριών ένα μονοειδές σε μια κατηγορία μονοειδών, είναι ένα αντικείμενο M με δύο μορφισμούς:

- $\mu : M \otimes M \to M$, που καλείται γινόμενο
- $\eta : I \to M$, που καλείται μονάδα (unit)

και ικανοποιούνται οι παρακάτω σχέσεις:

- $(a \bullet b) \bullet c = a \bullet (b \bullet c)$
- $e \bullet a = a \bullet e = a, \ \forall \ (a,b,c) \in S$

Μονάδα (Monad)

Στον συναρτησιακό προγραμματισμό ηλεκτρονικών υπολογιστών, μια μονάδα (monad) είναι μια δομή η οποία αναπαριστά τους υπολογισμούς οι οποίοι ορίζονται σε μια σειρά από βήματα. Ο τύπος της δομής μονάδας ορίζει αυτή την σειρά εφαρμογής υπολογισμών ή μια κλήση ενθυλακωμένων συναρτήσεων του ίδιου τύπου. Αυτή η δομή επιτρέπει στον προγραμματιστή να δημιουργεί επεξεργασίες οι οποίες επεξεργάζονται δεδομένα με βήματα. Σε κάθε ενέργεια υπάρχουν περαιτέρω κανόνες επεξεργασίας οι οποίες παρέχονται από την μονάδα.

Η Γλώσσα Haskell

Η Haskell είναι μια πρότυπη αμιγώς συναρτησιακή γλώσσα προγραμματισμού, που σημαίνει ότι οι συναρτήσεις της δεν έχουν παρενέργειες, γενικής χρήσης με ισχυρούς τύπους. Πήρε το όνομά της από τον Haskell Curry. Στη Haskell, "μια συνάρτηση είναι μέλος πρώτης τάξης". Ως συναρτησιακή γλώσσα, χρησιμοποιεί σαν κύρια δομή ελέγχου τη συνάρτηση.

- οκνηρή αποτίμηση
- ταίριασμα προτύπων (pattern matching)
- συμπερίληψη λιστών (list comprehensions)
- κλάσεις τύπων (typeclasses)
- πολυμορφισμό τύπων.

Η Haskell έχει ένα ισχυρό σύστημα τύπων βασισμένο στην εξαγωγή τύπων.
Υπάρχει ένας ξεχωριστός τύπος που απεικονίζει τις παρενέργειες, ορθογώνιος προς τον τύπο των συναρτήσεων. Ο τύπος που απεικονίζει τις παρενέργειες είναι η Μονάδα (Monad) και αποτελεί ένα γενικό πλαίσιο που μπορεί να μοντελοποιήσει διαφορετικούς τρόπους υπολογισμού, όπως η διαχείριση λαθών, ο μη-ντετερμινισμός, η λεξιλογική ανάλυση και η μνήμη συναλλαγών σε λογισμικό (software transactional memory). Οι Μονάδες ορίζονται σαν απλοί τύποι δεδομένων αλλά η Haskell παρέχει κάποιες συντακτικές διευκολύνσεις (syntactic sugar) για τη χρήση τους.

Η Γλώσσα Haskell

Το συντακτικό της Haskell είναι αρκετά επηρεασμένο από γλώσσες όπως η Clean, η FP, η Miranda και η Standard ML. Θυμίζει αρκετά το συντακτικό στα μαθηματικά και γι' αυτό είναι εύκολα κατανοητή από ανθρώπους με ισχυρό μαθηματικό υπόβαθρο ή ερευνητές στο χώρο του λ-λογισμού και της θεωρίας τύπων. Αποτελεί ένα πολύ εκφραστικό συντακτικό, με αποτέλεσμα με λίγο κώδικα να μπορεί να εκφράσει κανείς λύση σε πολύπλοκα και μεγάλα προβλήματα. Σε συνδυασμό με τη σύνθεση συναρτήσεων και την ικανότητα αφαίρεσης, λόγω της αντιμετώπισης των συναρτήσεων ως μέλη πρώτης τάξης, είναι πολύ εύκολο κανείς να γράψει ιδιωματικό κώδικα που να είναι πολύ περιεκτικός και υψηλού επιπέδου, δηλαδή ουσιαστικά κάποιος όταν τον διαβάζει να βλέπει απευθείας τον τρόπο λύσης.

Υλοποίηση του παραγοντικού με διάφορους τρόπους:

```
-- με αναδρομή
factorial 0 = 1
factorial n = n * factorial (n - 1)

-- με αναδρομή και χρήση guards
factorial n
  | n < 2 = 1
  | otherwise = n * factorial (n - 1)

-- με λίστες
factorial n = product [1..n]

-- με αναδρομή αλλά χωρίς pattern matching
factorial n = if n > 0 then n * factorial (n-1) else 1
```

Η Γλώσσα Haskell

Λόγω της οκνηρής αποτίμησης της Haskell, μπορεί κανείς να δημιουργήσει δομές που δεν άλλες γλώσσες φαντάζουν περίεργες ή και μη υλοποιήσιμες. Τέτοια παραδείγματα αποτελούν οι άπειρες λίστες, που προκύπτουν από το συνδυασμό δύο χαρακτηριστικών:

- Της οκνηρής αποτίμησης
- Της φύσης του δηλωτικού προγραμματισμού

```
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)

-- Δήλωση τύπου
fib :: Int -> Integer

-- Με χρήση αυτοαναφερόμενων δεδομένων
fib n = fibs !! n
      where fibs = 0 : scanl (+) 1 fibs

-- Το ίδιο, ρητά γραμμένο
fib n = fibs !! n
      where fibs = 0 : 1 : next fibs
            next (a : t@(b:_)) = (a+b) : next t

-- Παρόμοια ιδέα, όμως με χρήση της συνάρτησης zipWith
fib n = fibs !! n
      where fibs = 0 : 1 : zipWith (+) fibs (tail fibs)

-- Με χρήση συνάρτησης γεννήτριας
fib n = fibs (0,1) !! n
      where fibs (a,b) = a : fibs (b,a+b)
```

Η Γλώσσα Haskell

Ιδιαιτερότητες

- Για κλήση συνάρτησης, πάντα ίδιο αποτέλεσμα για ίδια ορίσματα
- Εύκολο να επιχειρηματολογήσουμε για ορθότητα
- Εύκολη παραλληλοποίηση προβλημάτων
- Nested Data Parallelism
- Δύσκολο να επιχειρηματολογήσουμε για απόδοση του προγράμματος
- “Ξένη” προσέγγιση όσον αφορά τις παρενέργειες, για προγραμματιστές που έχουν μάθει σε κάποια άλλη γλώσσα

Παρ’ όλες όμως όλες τις ιδιαιτερότητες της και της κριτικής που έχει δεχτεί όλα αυτά τα χρόνια, η Haskell παραμένει μια γλώσσα που κατά την άποψη πολλών δείχνει τη σωστή κατεύθυνση στο πως πρέπει να είναι και να σχεδιάζονται οι γλώσσες προγραμματισμού. Επίσης αποτελεί μια πολύ εξελιγμένη γλώσσα με πολύ καθαρές δομές και διαχωρισμό εννοιών, πράγμα που την κάνει ιδανική τόσο για τη διδασκαλία στην επιστήμη των υπολογιστών, όσο και σε εφαρμογές που χρειάζονται και επωφελούνται από τη δυνατότητα παραλληλοποίησης.

Η Γλώσσα Lisp

- Οικογένεια γλωσσών προγραμματισμού, που επηρεάστηκε σημαντικά από τη θεωρία και τη σύνταξη του λ-λογισμού
- Πολλαπλά προγραμματιστικά υποδείγματα (συναρτησιακό, προστακτικό, μεταπρογραμματισμό...)
- Πρώτη γλώσσα που εισήγαγε Garbage Collection, δεύτερη πιο παλιά γλώσσα υψηλού επιπέδου μετά τη Fortran

- Lisp = LISt Processor
- Όλες οι εκφράσεις της γλώσσας είναι s-expressions, και συντάσσονται ως λίστες
- S-expression: εκφράσεις που αντιστοιχούν σε δυαδικά δέντρα, έχοντας *άτομα* ως φύλλα.
- Διαδοχική σύνθεση φύλλων στον κάθε κόμβο αποφέρει αποτέλεσμα
- Λίστα στην Lisp ορίζεται με (cons a b) == (a.b): a αριστερό παιδί κόμβου, b δεξί παιδί κόμβου → λίστες στην Lisp αντιστοιχούν ευθέως σε s-expressions

Lisp: Συντακτικό

- Η Lisp δεν εκτελεί κώδικα, αλλά “αποτιμά εκφράσεις” - τις s-expressions που αναφέρθηκαν
- Οι συναρτήσεις έχουν prefix notation: (+ 2 3 4) προσθέτει τους αριθμούς 2,3,4.
- Το πρώτο σύμβολο ενός “γυμνού” s-expression αντιμετωπίζεται σαν συνάρτηση, εκτός αν ζητήσουμε διαφορετικά - ‘(+ 2 3 4) αντιμετωπίζεται ως λίστα με τα αντίστοιχα στοιχεία και δεν αποτιμάται περεταίρω
- Στοιχεία μιας λίστας είναι atoms ή άλλες λίστες
- Τα atoms είναι mutable, παραβιάζοντας τις αρχές του καθαρού συναρτησιακού προγραμματισμού.
- Atoms: Αριθμός ή σύμβολο (όνομα)
- Ορισμός ανώνυμων λ συναρτήσεων
- Κάθε valid s-expression αποτιμάται
- → δυνατότητα παραγωγής κώδικα από το πρόγραμμα για τον εαυτό του, με εκφράσεις που έχουν παραχθεί χρησιμοποιώντας οποιαδήποτε
  δυνατότητα της γλώσσας!

Lisp: Συντακτικό

- (lambda (arg) (+ arg 1)
- (defun foo (a b c d) (+ a b c d))
- (setf x 5)
- (defmacro setTo10(num) (setf num 10) (print num))
- (cond ((> a 20) (format t "~% a is greater than 20")) (t (format t "~% value of a is ~d " a)))
- (if (> a 20) (format t "~% a is less than 20")) (format t "~% value of a is ~d " a)
- (defun factorial (n)
  (if (= n 0) 1
  (* n (factorial (- n 1)))))

Επίλογος - Συμπεράσματα

- Ο λ-λογισμός επηρέασε σημαντικά ολόκληρη την επιστήμη των υπολογιστών και τη θεωρία γλωσσών προγραμματισμού και τύπων
- Όσο η απομάκρυνση από το κατώτερο επίπεδο προγραμματισμού είναι δυνατή λόγω ισχυρότερων υπολογιστικών συστημάτων και εξελιγμένων γλωσσών προγραμματισμού, η χρήση των αρχών του λ-λογισμού θα είναι όλο και πιο σημαντική
- Η κατανόηση των γλωσσών που στηρίχθηκαν στον λ-λογισμό είναι απαραίτητη για την σωστή κατανόηση των εννοιών της υπολογισιμότητας και της Turing πληρότητας
- Η βαθιά γνώση των αρχών σχεδιασμού και υλοποίησης των γλωσσών είναι αναπόσπαστο μέρος της εκπαίδευσης κάθε σύγχρονου προγραμματιστή
- Διευκολύνεται η ανάπτυξη προγραμμάτων και η ικανότητα να αποφανθούμε για τις δυνατότητες επίλυσης προβλημάτων