Αρχές Γλωσσών Προγραμματισμού

Άσκηση 3
Προσομοίωση Μηχανών Turing

Ημερομηνία ανάρτησης : 22/01/2020
Ημερομηνία παράδοσης : έως 23/02/2020 23:59

Εισαγωγή

Μια απλή μηχανή Turing (40%)

Κατασκευή της μηχανής (20%)

Σε αυτό το τμήμα της άσκησης καλείστε να υλοποιήσετε ένα σύνολο από συναρτήσεις, που θα προσομοιώνουν την εκτέλεση μιας μηχανής Turing, καθώς και ένα σύνολο από βοηθητικές συναρτήσεις που θα σας φανούν χρήσιμες.

Συγκεκριμένα, με βάση τους τύπους που θα βρείτε στο δοσμένο αρχείο θα υλοποιήσετε τις εξής συναρτήσεις:

1. (5%) **create_tape** :: [Char] → Tape
   η οποία παίρνει ως όρισμα μια συμβολοσειρά εισόδου (π.χ. "hello") και επιστρέφει μια ταινία (τύπος Tape) έχοντας αρχικοποιήσει κατάλληλα τα πεδία left, current, και right. Συγκεκριμένα, πρέπει η λίστα left αρχικά να περιέχει μόνο το σύμβολο εκκίνησης (Start), το current να περιέχει τον πρώτο χαρακτήρα (ή Null αν η είσοδος είναι το κενό string) της συμβολοσειράς ενώ οι υπόλοιποι χαρακτήρες της εισόδου εισάγονται στην αρχή της λίστας right. Η right μετά το τέλος της συμβολοσειράς εισόδου αποτελείται από **υποχρεωτικά άπειρα Null**. Αν η λίστα right είναι πεπερασμένη, η υλοποίησή σας θα θεωρηθεί **λάθος**.
2. (6%) **update_tape** :: Tape → Symbol → Move → Tape
   η οποία παίρνει σαν είσοδο μία ταινία, ένα σύμβολο και μια κίνηση, και δημιουργεί μια ανανεωμένη ταινία, γράφοντας το σύμβολο στην τρέχουσα θέση της ταινίας, και μετακινώντας την κεφαλή κατάλληλα, όπως ορίζει η κίνηση.

Προσοχή πρέπει να δοθεί στον σχεδιασμό που έχει η ταινία: τα left και right είναι ουσιαστικά στοίβες που περιέχουν τα στοιχεία δεξιά και αριστερά του τρέχοντος στοιχείου (current). Αυτή η μοντελοποίηση επιτρέπει να κάνουμε σε σταθερό χρόνο update στην ταινία.

3. (3%) **init_machine** ::
   State → [Char] → (State → Symbol → (State, Symbol, Move)) → Machine
   η οποία παίρνει σαν όρισμα την αρχική κατάσταση της μηχανής, μια συμβολοσειρά εισόδου και μια συνάρτηση μετάβασης και επιστρέφει μια μηχανή της οποίας τα μέλη έχουν αρχικοποιηθεί με βάση τα inputs αυτά. Εδώ θα σας φανεί ιδιαίτερα χρήσιμη η create_tape.

4. (3%) **step** :: Machine → Machine
   η οποία παίρνει σαν είσοδο μια Μηχανή Turing και εκτελεί ένα βήμα του υπολογισμού της, όπως ορίζει η συνάρτηση μετάβασης της μηχανής. Εδώ θα χρειαστείτε την update_tape που ορίσατε παραπάνω.

5. (3%) **run** :: Machine → Machine
   η οποία παίρνει σαν είσοδο μια μηχανή και την "τρέχει", μέχρι να ολοκληρωθεί ο υπολογισμός της, δηλαδή να φτάσει η μηχανή σε κατάσταση Accept ή Reject. Υλοποιείται ιδιαίτερα απλά με την χρήση της step.

Συναρτήσεις μετάβασης (20%)

Μια γλώσσα είναι ένα (πιθανώς άπειρο) σύνολο από συμβολοσειρές.

Λέμε ότι μια μηχανή αποφασίζει μια γλώσσα $\mathcal{L}$ αν, και μόνο αν, (i) πάντα τερματίζει και (ii) τερματίζει σε κατάσταση Accept όταν η συμβολοσειρά εισόδου της μηχανής ανήκει στην $\mathcal{L}$ (προφανώς, σε αντίθετη περίπτωση τερματίζει σε κατάσταση Reject).

Βασιζόμενοι στην υλοποίηση των μηχανών που πραγματοποιήσατε παραπάνω, καλείστε να υλοποιήσετε ορισμένες συναρτήσεις μετάβασης οι οποίες να αποφασίζουν τις παρακάτω γλώσσες:

1. Την κενή γλώσσα $\mathcal{L} = \emptyset$, η οποία προφανώς δεν περιέχει καμία συμβολοσειρά.

2. Τη γλώσσα $\mathcal{L} = \{aaa\}$, που περιέχει μόνο την συμβολοσειρά "aaa".

3. Τη γλώσσα $\mathcal{L} = \{ab^n c \mid n \geq 0\}$, που περιέχει συμβολοσειρές που ξεκινάνε με a, ακολουθούν 0 ή περισσότερα b και τελειώνουν με c. Για παράδειγμα $\{ac, abc, abbc\} \subseteq \mathcal{L}$ δηλαδή οι συμβολοσειρές που βρίσκονται εντός των αγκυλών περιέχονται στην $\mathcal{L}$.

4. Τη γλώσσα $\mathcal{L} = \{a^{2^n} \mid n \geq 0\}$, που περιέχει συμβολοσειρές που αποτελούνται μόνο από τον χαρακτήρα a, αλλά το μήκος τους είναι αυστηρά δύναμη του δύο. Για παράδειγμα $\{a, aa, aaaa\} \subseteq \mathcal{L}$.

Οι συναρτήσεις που θα γράψετε πρέπει να έχουν **ακριβώς** τα εξής ονόματα και τύπους:

1. (5%) **empty** :: State → Symbol → (State, Symbol, Move)
2. (5%) **a3** :: State → Symbol → (State, Symbol, Move)
3. (10%) **absc** :: State → Symbol → (State, Symbol, Move)
4. (10%) **po2** :: State → Symbol → (State, Symbol, Move)

και προφανώς ο κοινός τύπος τους είναι ο ίδιος με τον τύπο του πεδίου delta στην δομή Machine.

Σημείωση: Οι γλώσσες του ερωτήματος 3 και 4 είναι απειροσύνολα, περιέχουν δηλαδή έναν άπειρο αριθμό από συμβολοσειρές. Παρ' όλα αυτά μπορούμε να κατασκευάσουμε μηχανή Turing η οποία αναγνωρίζει οποιαδήποτε συμβολοσειρά των γλωσσών αυτών, παρόλο που δεν μπορούμε να απαριθμήσουμε όλες τις συμβολοσειρές των γλωσσών σε πεπερασμένο χρόνο. Οι μηχανές αυτές αναγνωρίζουν το "pattern" της συμβολοσειράς. Κατά κάποιο τρόπο η μηχανή μας θα κάνει κάποια "επανάληψη υπό συνθήκη" ώστε να είναι αρκετά γενική για να χειριστεί οποιαδήποτε από τις συμβολοσειρές.

Συνένωση γλωσσών (Bonus 15%)

Στο ερώτημα αυτό καλείστε να υλοποιήσετε την συνάρτηση
conc :: [Char]
→ (State → Symbol → (State, Symbol, Move))
→ (State → Symbol → (State, Symbol, Move))
→ Maybe ([Char], [Char])

η οποία παίρνει ως ορίσματα μια συμβολοσειρά εισόδου w και δύο συναρτήσεις μετάβασης που ορίζουν δύο μηχανές $\mathcal{M}_1$ και $\mathcal{M}_2$ αντίστοιχα, και αποφασίζει αν η w ανήκει στην συνένωση των γλωσσών $\mathcal{L}_1$, $\mathcal{L}_2$ που αποφασίζουν οι μηχανές, δηλαδή αν υπάρχει διαχωρισμός της w τέτοιος ώστε $w = w_1w_2$ και $w_1 \in L_1$, $w_2 \in L_2$.

Η συνάρτηση αυτή θα επιστρέφει $Just(w_1, w_2)$ αν ισχύει το παραπάνω, διαφορετικά Nothing. Για περισσότερα σχετικά με τον τύπο Maybe, μπορείτε να διαβάσετε στο παράρτημα Maybe.

Απαριθμητές (20%)

Μια επέκταση των απλών μονοταινιακών μηχανών Turing είναι μια κατηγορία μηχανών οι οποίες ονομάζονται απαριθμητές. Φανταστείτε τώρα ότι έχουμε στην διάθεση μας και έναν εκτυπωτή, ο οποίος συνδέεται με την μηχανή και λειτουργεί υπό τον έλεγχο της ως εξής : Αρχικά, επεκτείνουμε το σύνολο καταστάσεων της μηχανής με μία πρόσθετη ειδική κατάσταση την οποία ονομάζουμε "Output". Οποιαδήποτε στιγμή η μηχανή βρεθεί στην κατάσταση "Output" θεωρούμε ότι στέλνει ένα σήμα στον εκτυπωτή ο οποίος τυπώνει τα περιεχόμενα της ταινίας από τον πρώτο χαρακτηρά δεξιά του συμβόλου εκκίνησης ("Start") μέχρι τον πρώτο κενό χαρακτήρα (αλλιώς ο εκτυπωτής θα έπεφτε σε άπειρο loop αφού η ταινία είναι άπειρη προς τα δεξιά).

Η βασική χρήση των απαριθμητών όπως υποδηλώνει και το όνομα τους, είναι να "απαριθμούν" τις συμβολοσειρές μιας γλώσσας $\mathcal{L}$. Προφανώς, αν η $\mathcal{L}$ είναι άπειρη γλώσσα τότε η λειτουργία του απαριθμητή δεν θα τερματίζει ποτέ αλλά αυτό δεν είναι πρόβλημα. Για παράδειγμα ο εκτυπωτής ενός απαριθμητή της γλώσσας $\mathcal{L} = \{hello, world\}$ θα τυπώσει τις συμβολοσειρές "hello" και "world" και στην συνέχεια η μηχανή θα σταματήσει να λειτουργεί (θα φτάσει σε κατάσταση "Accept"). Αντίθετα αν η γλώσσα ήταν η $\mathcal{L} = \{a^n b^n | n > 0\}$ ο εκτυπωτής θα τύπωνε διαδοχικά "ab", "aabb", "aaabbb", ... Προφανώς σε αυτή την περίπτωση η μηχανή δεν θα τερματίσει ποτέ.

Προσομοίωση του απαριθμητή (5%)

Σε αυτό το ερώτημα, χρειάζεται να ορίσετε μια εναλλακτική συνάρτηση της **run** την οποία θα ονομάσουμε **enum** :: Machine → [[Symbol]] για να προσομοιώσετε την λειτουργία του απαριθμητή. Συγκεκριμένα, θα πρέπει κάθε φορά που η μηχανή βρίσκεται στην κατάσταση "Output", να σώζετε το "χρήσιμο κομμάτι" της ταινίας σε μία λίστα, πριν προχωρήσετε την μηχανή στο επόμενο της βήμα.

Ως "χρήσιμο κομμάτι" της ταινίας ορίζουμε το περιεχόμενό της χωρίς το σύμβολο "Start" και τα trailing "Null".

Συναρτήσεις μετάβασης για απαρίθμηση (15%)

Υλοποίηστε τις παρακάτω συναρτήσεις μετάβασης που απαριθμούν τις παρακάτω γλώσσες:

1. Τη γλώσσα $\mathcal{L} = \{1^n \mid n \geq 0\}$, που περιέχει την κωδικοποίηση του $\mathbb{N}_0$ στο μοναδιαίο σύστημα αρίθμησης, σε αύξουσα σειρά.

2. Τη γλώσσα που περιέχει την κωδικοποίηση του $\mathbb{N}_0$ στο δυαδικό σύστημα αρίθμησης, σε αύξουσα σειρά

   Σημείωση: Για διευκόλυνσή σας, τα σημαντικότερα ψηφία είναι στα δεξιά, δηλαδή οι αριθμοί είναι ανεστραμμένοι συνεπώς η κωδικοποίηση έχει ως εξής:

   $$0 \to 0, \quad 1 \to 1, \quad 2 \to 01, \quad 3 \to 11, \quad 4 \to 001 \ldots$$

Οι συναρτήσεις που θα γράψετε πρέπει να έχουν **ακριβώς** τα εξής ονόματα και τύπους:

1. (5%) **unary** :: State → Symbol → (State, Symbol, Move)
2. (10%) **binary** :: State → Symbol → (State, Symbol, Move)

Μηχανή Turing με δύο ταινίες (15%)

Μια απλή επέκταση στο αρχικό μας μοντέλο, είναι η εισαγωγή μιας επιπλέον ταινίας. Στην δική μας περίπτωση, θα χρησιμοποιήσουμε τη δεύτερη ταινία σαν κάποιου είδους μνήμης, η χρησιμότητα της οποίας θα φανεί στο επόμενο ερώτημα.

Επέκταση του αρχικού ορισμού (5%)

Επεκτείνετε το data type Machine, με έναν επιπλέον constructor που θα ονομάσουμε **MemMachine**.

Αυτό που ζητείται είναι η εισαγωγή μιας επιπλέον ταινίας καθώς και η αλλαγή του τύπου της συνάρτησης μετάβασης προκειμένου να παίρνει σαν είσοδο δύο σύμβολα αντί για ένα (το τρέχον σύμβολο κάθε ταινίας) και να επιστρέφει την επόμενη κατάσταση της μηχανής, μαζί με ένα σύμβολο και μια κίνηση για κάθε ταινία.

Επέκταση των συναρτήσεων της αρχικής μηχανής (10%)

Στο ερώτημα αυτό καλείστε να πραγματοποιήσετε τις απαραίτητες, επεκτάσεις στις συναρτήσεις που ορίσαμε στο αρχικό ερώτημα (όχι αναγκαστικά όλες), προκειμένου να μπορούμε να τρέξουμε την νέα μηχανή.

Σημείωση: ***Δεν*** *χρειάζεται να επαναορίσετε συναρτήσεις. Αρκεί να προσθέσετε επιπλέον "περιπτώσεις" σε όσες από τις υπάρχουσες αυτό χρειάζεται.*

Εξαίρεση *στο παραπάνω αποτελεί η init_machine, όπου θα ορίσετε μια νέα συνάρτηση* ***init_mem_machine*** *η οποία θα λειτουργεί παρόμοια με την init_machine, αλλά θα επιστρέφει μια MemMachine με την δεύτερη ταινία αρχικοποιημένη με την κενή συμβολοσειρά (δηλαδή να περιέχει μόνο το σύμβολο Start στην αρχή της). Για να δείτε το αποτέλεσμα της init_mem_machine θα πρέπει να ορίσετε instance του Show για το νέο τύπο μηχανής με παρόμοιο τρόπο που αυτό γίνεται στον δοσμένο κώδικα για την αρχική μηχανή.*

Προαιρετικά: Για να ελέγξετε ότι η επέκταση σας είναι επιτυχής μπορείτε να δοκιμάσετε να επαναορίσετε κάποια από τις συναρτήσεις μετάβασης για τις γλώσσες προηγούμενων ερωτημάτων (με νέο όνομα προφανώς) με τέτοιο τρόπο που να αγνοείται η δεύτερη βοηθητική ταινία. Το αποτέλεσμα στην "κύρια" ταινία θα πρέπει να είναι ***ίδιο*** *με αυτό της αρχικής μηχανής.*

Αναπαράσταση συνθετότερων Μηχανών Turing με γράφο (25%)

Αν και ο αρχικός φορμαλισμός που εισαγάγαμε για την Μηχανή Turing μπορεί να υπολογίσει οποιαδήποτε υπολογίσιμη συνάρτηση, είναι φανερό ότι για μεγαλύτερες και πιο πολύπλοκες μηχανές (για παράδειγμα αντιγραφή ή ολίσθηση συμβολοσειράς) ο ορισμός μιας κατάλληλης συνάρτησης μετάβασης είναι εφικτός αλλά αρκετά σύνθετος.

Για τον λόγο αυτό, στην βιβλιογραφία χρησιμοποιείται ευρέως μια αναπαράσταση σύνθετων Μηχανών Turing με γράφους, οι κόμβοι των οποίων είναι δομικές, αρχέγονες μηχανές και οι ακμές κωδικοποιούν μεταβάσεις του υπολογισμού από κόμβο σε κόμβο, πιθανώς υπό κάποια συνθήκη. Πριν συνεχίσετε, δείτε το [Παράρτημα Β], για κατανόηση της αναπαράστασης αυτής αλλά και του πώς πρέπει να μετασχηματιστεί στο πλαίσιο της εργασίας.

Στο αρχείο που σας έχει δοθεί, υπάρχει ο ορισμός των δομών που θα χρειαστεί να χειριστείτε (τελευταίες γραμμές του αρχείου). Ορίζεται η έννοια του κόμβου καθώς και του γράφου. Οι δομές αυτές πρέπει να χρησιμοποιηθούν πιστά στην υλοποίηση των επόμενων συναρτήσεων. **Μην** ορίσετε δικές σας δομές για τους κόμβους και τον γράφο.

Βασικές συναρτήσεις μετάβασης (6%)

Στο ερώτημα αυτό καλείστε να υλοποιήσετε ορισμένες βασικές συναρτήσεις μετάβασης οι οποίες θα αποτελέσουν τα θεμέλια για την κατασκευή πιο σύνθετων μηχανών. Οι συναρτήσεις που θα υλοποιήσετε είναι οι εξής:

1. **left_**, η οποία μετακινεί την κεφαλή της κύριας ταινίας μια θέση αριστερά και αντιστοιχεί στον κόμβο **LM**.
2. **right_**, η οποία μετακινεί την κεφαλή της κύριας ταινίας μια θέση δεξιά και αντιστοιχεί στον κόμβο **RM**.
3. **push_**, η οποία κάνει "push" το τρέχον σύμβολο της κύριας ταινίας στην ταινία μνήμης και αντιστοιχεί στον κόμβο **PushM**.
4. **write_**, η οποία γράφει το τρέχον σύμβολο της ταινίας μνήμης στο κελί που βρίσκεται στην τρέχουσα θέση της κύριας ταινίας και αντιστοιχεί στον κόμβο **WriteM**.
5. **pop_**, η οποία κάνει "pop" το τρέχον σύμβολο της ταινίας μνήμης (γράφει Null) και κινεί την κεφαλή της ταινίας μνήμης μια θέση αριστερά. Αντιστοιχεί στον κόμβο **PopM**.
6. **writec_**, που ορίζει μια οικογένεια συναρτήσεων, οι οποίες απλώς γράφουν το σύμβολο c στην τρέχουσα θέση της κύριας ταινίας. Παρατηρήστε ότι μιας και το αλφάβητό μας είναι πεπερασμένο, θα μπορούσαμε να ορίσουμε μια συνάρτηση για κάθε σύμβολο. Ωστόσο, εκμεταλλευόμαστε την δυνατότητα

που μας δίνει η Haskell να κάνουμε partial application (ή αλλιώς currying) σε συναρτήσεις, έτσι ώστε για παράδειγμα η κλήση writec_ (Symbol 'a') να μου επιστρέφει μια **συνάρτηση μετάβασης** που γράφει τον χαρακτήρα 'a' στην ταινία.

Η οικογένεια συναρτήσεων αντιστοιχεί στον κόμβο **WriteCM** με όρισμα ένα **Symbol**. Για παράδειγμα από έναν κόμβο **WriteCM (Symbol 'a')** πρέπει στο επόμενο ερώτημα να κατασκευάζεται μια μηχανή με την κατάλληλη συνάρτηση μετάβασης (εδώ είναι η **writec_ (Symbol 'a')**).

Εκτέλεση του γράφου (6%)

Σκοπός αυτού του τμήματος της άσκησης είναι η κατασκευή μιας σειράς συναρτήσεων οι οποίες υλοποιούν τον μηχανισμό με τον οποίο θα μπορούμε να εκτελούμε τέτοιους γράφους. Συγκεκριμένα καλείστε να υλοποιήσετε τις παρακάτω συναρτήσεις:

1. (2%) **graph_step** :: Vertex → Tape → Tape → (Tape, Tape)

   η οποία παίρνει σαν είσοδο έναν κόμβο του γράφου (Vertex) και τις δύο ταινίες και επιστρέφει δύο νέες ταινίες, οι οποίες προκύπτουν από την εκτέλεση του κόμβου πάνω στις ταινίες εισόδου. Η μηχανή αυτή θα πρέπει να καλύπτει όλα τα base cases, δηλαδή αν το Vertex είναι LM, RM, PushM κ.ο.κ., καθώς και την αναδρομική περίπτωση όπου ο κόμβος είναι σύνθετος (CompVertex), και περιέχει έναν γράφο μέσα του, οπότε χρειάζεται ειδική μεταχείριση.

2. (2%) **interpret**
   :: [(Int, Vertex)] → [(Int, Symbol → Bool, Int)] → [Char] → (Tape, Tape)

   η οποία με είσοδο μια λίστα από ζευγάρια (Int, Vertex) όπου οι ακέραιοι κάνουν "label" τους κόμβους και μια λίστα από ακμές (Int, Symbol → Bool, Int) κατασκευάζει τον γράφο και στη συνέχεια τον "τρέχει" διασχίζοντας τις ακμές (αφού ελέγξει ότι η συνθήκη τους ικανοποιείται για τον τρέχον χαρακτήρα της κύριας ταινίας) και επιστρέφει ένα ζευγάρι (Tape, Tape), με την κατάσταση των δύο ταινιών μετά τον τερματισμό της διαδικασίας.

   Σημείωση 1 *Προσέξτε πώς θα αρχικοποιήσετε τον γράφο, καθώς οι ακμές αποθηκεύονται στο Map του γράφου με μορφή λίστας γειτνίασης αλλά στην είσοδο δίνονται ως απλή λίστα.*

   Σημείωση 2 *Θεωρήστε ότι η εκτέλεση του γράφου θα ξεκινάει πάντα από έναν κόμβο με συγκεκριμένο label (π.χ. το 0).*

3. (2%) **interpret_graph** :: CompGraph -> [Char] -> (Tape, Tape)

   η οποία με είσοδο έναν έτοιμο γράφο και μια συμβολοσειρά, εκτελεί τον γράφο αφού τοποθετήσει την συμβολοσειρά στην κύρια ταινία και επιστρέφει την τελική κύρια και βοηθητική ταινία ως έξοδο.

Ορισμός βοηθητικών συναρτήσεων (3%)

Για να γίνει ακόμα πιο εύκολη η κατασκευή των γράφων, ορίστε τις παρακάτω συναρτήσεις:

1. (1%) **create_singleton_graph** :: Vertex → CompGraph
   η οποία με είσοδο ένα Vertex, επιστρέφει έναν γράφο που περιέχει μόνο αυτόν τον κόμβο.
2. (2%) **create_chain_graph** :: [Vertex] → CompGraph
   η οποία με είσοδο μια λίστα από κόμβους, επιστρέφει έναν γράφο-αλυσίδα, η διατάξη του οποίου είναι η ίδια με την διάταξη των κόμβων στην λίστα.

Υλοποίηση μηχανών με τον παραπάνω φορμαλισμό (10%)

Προκειμένου να δοκιμάσετε την υλοποίησή σας, καθώς και να δείτε πόσο πιο εκφραστικό εργαλείο είναι το παραπάνω, καλείστε να υλοποιήσετε τις παρακάτω συναρτήσεις:

***Σημείωση 1**: Για κάποια από τα επόμενα ερωτήματα θα χρειαστεί να εισάγετε κάποια συνάρτηση "μετατροπής", που θα μετατρέπει τους χαρακτήρες '_' σε σύμβολα Null κατά το στάδιο δημιουργίας της ταινίας. Ο χαρακτήρας '_' έχει πλέον **ειδική** σημασία και δεν θα πρέπει να χρησιμοποιείται στις συμβολοσειρές εισόδου παρά μόνο όταν αυτό ζητείται από την εκφώνηση. Επίσης, το πρόγραμμα σας προφανώς δεν θα δοκιμαστεί σε εισόδους που περιέχουν το '_' εκτός από τις περιπτώσεις που αναφέρεται ρητά.*

***Σημείωση 2**: Οι γράφοι που δίνονται δεν είναι στην ίδια ακριβώς "γλώσσα" με αυτή που εμείς κωδικοποιούμε τα γραφήματα, συνεπώς πρέπει πρώτα να μετασχηματιστούν κατάλληλα. Οι διαφορές που θα πρέπει να προσέξετε περιγράφονται στο [Παράρτημα Β].*

1. (2%) **l_until** :: Symbol → Vertex
   η οποία με όρισμα ένα χαρακτήρα, επιστρέφει έναν σύνθετο κόμβο που κωδικοποιεί μια μηχανή η οποία προχωράει αριστερά μέχρι να βρει τον δοσμένο χαρακτήρα. [Figure 1]
2. (2%) **r_until** :: Symbol → Vertex
   η οποία με όρισμα ένα χαρακτήρα, επιστρέφει έναν σύνθετο κόμβο που κωδικοποιεί μια μηχανή η οποία προχωράει δεξιά μέχρι να βρει τον δοσμένο χαρακτήρα. [Figure 2]
3. (3% + 5% Bonus) **shift** :: Vertex
   μια σταθερή συνάρτηση που επιστρέφει έναν σύνθετο κόμβο που πραγματοποιεί δεξιά μετατόπιση της εισόδου κατά μία θέση. Ο γράφος της μηχανής δίνεται στο [Figure 3]. Η είσοδος θα δίνεται **απαραίτητα** ως εξής: Για μια συμβολοσειρά "w" δίνω ως είσοδο το "_w" για να λειτουργήσει σωστά η δοσμένη μηχανή και παίρνω ως έξοδο το "__w" με την κεφαλή να βρίσκεται στο δεύτερο Null όπως φαίνεται και στο αντίστοιχο παράδειγμα. [Παράρτημα Γ]

4. (3% + 5% Bonus) **copy** :: Vertex
   μια σταθερή συνάρτηση που επιστρέφει έναν σύνθετο κόμβο που αντιγράφει την είσοδο. Ο γράφος της μηχανής δίνεται στο [Figure 4]. Η είσοδος θα δίνεται **απαραίτητα** ως εξής: Για μια συμβολοσειρά "w" δίνω ως είσοδο το "$\_w$" για να λειτουργήσει σωστά η δοσμένη μηχανή και παίρνω ως έξοδο το "$\_w\_w\_$" με την κεφαλή να βρίσκεται στο τελευταίο Null όπως φαίνεται και στο αντίστοιχο παράδειγμα. [Παράρτημα Γ]

Bonus (15%)

Χρησιμοποιώντας την **create_chain_graph** και **μόνο** τις σύνθετες μηχανές 1-4 του προηγούμενου ερωτήματος ορίστε την συνάρτηση **ww** :: Vertex
η οποία επιστρέφει ένα σύνθετο κόμβο που κωδικοποιεί μια μηχανή η οποία με είσοδο μια συμβολοσειρά "$\_w$" παράγει ως έξοδο στην ταινία την "$\_\_ww$".

Μπορείτε να δείτε παραδείγματα εκτέλεσης για πολλές από τις συναρτήσεις που ζητούνται στο [Παράρτημα Γ].

Παράδοση Ασκήσεων: Η παράδοση πρέπει να γίνει μέχρι τις 23:59μμ, την 23/02/2020. Θα δημιουργήσετε ένα αρχείο το οποίο θα περιέχει τις λύσεις όλων των ασκήσεων και θα το στείλετε με email και στις τρεις παρακάτω διευθύνσεις: `cs3190001@di.uoa.gr`, `cs3190003@di.uoa.gr`, και `prondo@di.uoa.gr`. Τα ονόματα των συναρτήσεων που θα χρησιμοποιήσετε στα προγράμματά σας πρέπει να είναι **ακριβώς** τα ίδια με αυτά που καθορίζονται από την παραπάνω εκφώνηση. Καθυστερημένες ασκήσεις δεν θα βαθμολογηθούν.

Καρυστινός Νίκος
Τσατίρης Ηλίας

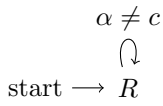

Figure 1: Γράφος για την r_until c

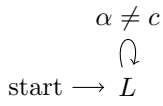

Figure 2: Γράφος για την l_until c

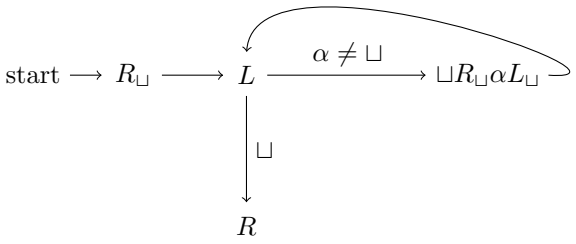

Figure 3: Γράφος για την shift

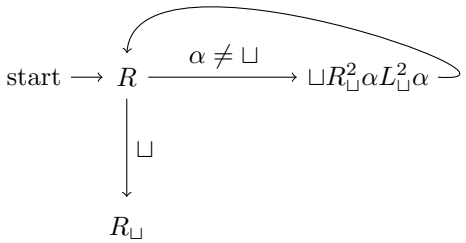

Figure 4: Γράφος για την copy

Παράρτημα Α

Maybe

Πολλές φορές οι συναρτήσεις που ορίζουμε είναι *μερικές*, δηλαδή δεν είναι ορισμένες για όλες τις τιμές του πεδίου ορισμού τους. Για παράδειγμα, η συνάρτηση της τετραγωνικής ρίζας είναι ορισμένη μόνο για μη-αρνητικούς αριθμούς. Ένας ωραίος τρόπος να "κρύψουμε" αυτή την συμπεριφορά, και να κωδικοποιήσουμε με κάποιο τρόπο ότι ο υπολογισμός μας απέτυχε, είναι με την χρήση του Maybe.

Ο ορισμός του τύπου Maybe έχει ως εξής:

```
data Maybe a = Just a | Nothing
```

Ο constructor Nothing χρησιμοποιείται για να υποδηλώσουμε ότι ο υπολογισμός απέτυχε, ενω αντίθετα ο constructor Just a, χρησιμοποιείται όταν ο υπολογισμός μας είναι επιτυχής, και μέσα του ενθυλακώνεται το αποτέλεσμα του υπολογισμού.

Για παράδειγμα, αν θέλαμε να υλοποιήσουμε μια "ασφαλή" τετραγωνική ρίζα που επιστρέφει Maybe Floating για όλους τους αριθμούς κινητής υποδιαστολής (θετικούς και μη) θα κάναμε το εξής:

```
safe_sqrt x
    | x >= 0 = Just (sqrt x)
    | otherwise = Nothing
```

Παράρτημα Β

Ας εξηγήσουμε τον συμβολισμό στην αρχική του μορφή με ένα παράδειγμα. Δείτε το γράφο στο [Figure 3].

- Το start καθορίζει τον αρχικό κόμβο του γράφου.
- Οι ακμές ορίζουν τις μεταβάσεις μεταξύ των κόμβων του γράφου, και είναι πιθανώς επαυξημένες με μια συνθήκη, η οποία υποδηλώνει πότε να "πάρουμε" μια μετάβαση.
- Το σύμβολο R δηλώνει την primitive μηχανή που κινεί την κεφαλή μία θέση δεξιά.
- Αντίστοιχα το L δηλώνει την primitive μηχανή που κινεί την κεφαλή μία θέση αριστερά.
- Η σύνθετη μηχανή $R_{\sqcup}$, της οποίας ο γράφος ορίζεται [εδώ], προχωράει δεξιά την κεφαλή μέχρι να βρεί το σύμβολο $\sqcup$ στην ταινία.
- Ομοίως η $L_{\sqcup}$, κάνει το ίδιο στην αριστερή κατεύθυνση.
- Γενικότερα η μηχανή R_c ή L_c κινεί την κεφαλή δεξιά ή αριστερά αντίστοιχα, μέχρι να βρει το σύμβολο c.

- Η συνθήκη $\alpha \neq \sqcup$ πάνω στην ακμή ελέγχει αν το τρέχον σύμβολο είναι διάφορο του $\sqcup$ και μόνο τότε γίνεται αυτή η μετάβαση. Η απουσία συνθήκης υπονοεί την σταθερά True, δηλαδή η μετάβαση γίνεται πάντα. Αξίζει να σημειωθεί ότι για ντετερμινιστικές μηχανές που μας αφορούν στην εργασία, σε κάθε βήμα το πολύ μια συνθήκη ακμής μπορεί να είναι True.
- Η μηχανή $\sqcup$ απλώς γράφει τον χαρακτήρα $\sqcup$ στην ταινία.
- Γενικότερα, όταν βλέπουμε ένα σύμβολο s του αλφαβήτου, αυτό υποδηλώνει την μηχανή που γράφει αυτό το σύμβολο στην ταινία. Το ίδιο ισχύει και για την παράμετρο α, που βλέπουμε ότι γίνεται "match" από την συνθήκη και στη συνέχεια χρησιμοποιείται.
- Ένας επιπλέον συμβολισμός που δεν εμφανίζεται στην μηχανή shift είναι ο M^n όπου M είναι κάποια primitive μηχανή και n κάποιος θετικός ακέραιος (συνήθως μεγαλύτερος της μονάδας). Το νόημα αυτού του συμβολισμού είναι ότι "επαναλαμβάνουμε" την μηχανή M, n φορές. Δηλαδή, παραδείγματος χάριν, R^2 σημαίνει δύο φορές δεξιά.
- Τέλος, η διαδοχική παράθεση μηχανών χωρίς ακμές ανάμεσά τους, χρησιμοποιείται ως συντομογραφία για την "αλυσιδωτή" συνδεσμολογία, με συνθήκη μετάβασης σε κάθε ακμή True.

Ο συμβολισμός αυτός, στον οποίο είναι δοσμένες και οι συνθετότερες μηχανές που ζητείται να υλοποιήσετε, "κλέβει" σε ένα βασικό σημείο σε σχέση με την απλή μηχανή Turing:

Στον γράφο που μόλις εξετάσαμε, ο συμβολισμός μας "θυμάται" τον χαρακτήρα α μετά την ακμή με συνθήκη $\alpha \neq \sqcup$ και στη συνέχεια τον γράφει στην ταινία. Μια μηχανή μιας ταινίας δεν μπορεί εύκολα να μιμηθεί αυτή τη συμπεριφορά, και ο τρόπος για να το κάνει προσθέτει έναν τεράστιο αριθμό καταστάσεων που εξαρτάται από το μέγεθος του αλφαβήτου. Αυτός είναι και ο λόγος που σας ζητήθηκε να ορίσετε την $MemMachine$ η οποία έχει δύο ταινίες και μπορεί να "λύσει" ευκολότερα αυτό το πρόβλημα.

Για να λύσετε το πρόβλημα της "μνήμης" χαρακτήρων σκεφτείτε πώς θα πρέπει να μετασχηματίσετε τους δοσμένους γράφους όπου χρειάζεται ώστε να καταφέρετε να "θυμάστε" χαρακτήρες και να τους χρησιμοποιείτε αργότερα με δεδομένο ότι θα έχετε για primitive μηχανές αυτές που περιγράφονται στο ερώτημα **Βασικές συναρτήσεις μετάβασης** :

1. Μια μηχανή που κινείται αριστερά στη βασική ταινία.
2. Μια μηχανή που κινείται δεξιά στη βασική ταινία.
3. Μια μηχανή που γράφει οποιονδήποτε χαρακτήρα του αλφαβήτου παραμετρικά όπως εξηγείται στο αντίστοιχο ερώτημα.
4. Μια μηχανή που γράφει το τρέχον σύμβολο της κύριας ταινίας στην ταινία μνήμης και τοποθετεί την κεφαλή της ταινίας μνήμης σε αυτό ακριβώς το σύμβολο.

5. Μια μηχανή που γράφει το τρέχον σύμβολο της ταινίας μνήμης στην κύρια ταινία.

6. Μια μηχανή που αφαιρεί το τελευταίο σύμβολο της ταινίας μνήμης και κινεί κατά μία θέση αριστερά την κεφαλή της ταινίας μνήμης.

Προφανώς, οι μηχανές 4 και 5 όταν τοποθετηθούν στο σωστό σημείο σε έναν γράφο μπορούν να μας βοηθήσουν να θυμόμαστε και να γράφουμε από μνήμης σύμβολα αντίστοιχα.

Με δεδομένα τα παραπάνω, όταν θα κάνετε interpret τον γράφο **ΑΠΑΓΟΡΕΥΕΤΑΙ** να πειράξετε τις ταινίες άμεσα και πρέπει **όλες** οι ενέργειες πάνω στις ταινίες να γίνονται μέσω μηχανών. Έτσι, δεν επιτρέπεται να χρησιμοποιήσετε την Haskell για να "θυμάστε" κάποιον χαρακτήρα.

Παράρτημα Γ

my_tape

```
> create_tape "mytape"
[->] >> 'm' << ['y','t','a','p','e',_]
```

---

update_tape

```
> let t = create_tape "mytape"

> update_tape t (Symbol 'k') R
[->,'k'] >> 'y' << ['t','a','p','e',_]
```

---

init_machine

```
> let go_right (State 0) x = (Accept, x, R)

> init_machine (State 0) "mytape" go_right
Machine {
    State: State 0
    Tape:  [->] >> 'm' << ['y','t','a','p','e',_]}
```

---

step

```
> let m = init_machine (State 0) "mytape" go_right

> step m
```

```
Machine {
    State: Accept
    Tape:  [->, 'm'] >> 'y' << ['t','a','p','e',_]}
```

---

run

```
> let as (State 0) (Symbol 'a') = (State 0, Symbol 'a', R)
      as (State 0) Null = (Accept, Null, S)
      as (State 0) x = (Reject, x, S)

> let mas = init_machine (State 0) "aaaa" as

> run mas
Machine {
    State: Accept
    Tape:  [->,'a', 'a', 'a', 'a'] >> _ <<  [_]}
```

---

conc

```
> let bs (State 0) (Symbol 'b') = (State 0, Symbol 'b', R)
      bs (State 0) Null = (Accept, Null, S)
      bs (State 0) x = (Reject, x, S)

> conc "aaaabbb" as bs
Just ("aaaa","bbb")

> conc "abaabbb" as bs
Nothing
```

---

enum

```
> take 4 $ enum (init_machine (Output) "" unary)
[[],['1'],['1','1'],['1','1','1']]

> take 4 $ enum (init_machine (State 0) "" binary)
[['0'],['1'],['0','1'],['1','1']]
```

---

init_mem_machine

```
> let m_as (State 0) (Symbol 'a') m = (State 0, Symbol 'a', R, m, S)
```

```
      m_as (State 0) Null m = (Accept, Null, R, m, S)
      m_as (State 0) x m = (Reject, x, S, m, S).

> init_mem_machine (State 0) "abc" m_as
MemMachine {
    State: State 0
    Main: [->] >> 'a' << ['b', 'c', _]
    Memo: [->] >> _ << [_]}
```

---

interpret

```
> interpret [(0, RM), (1, LM)] [(0, \x -> True, 1)] "_aa"
([->] >> _ << ['a','a',_],[->] >> _ << [_])
```

---

interpret_graph

```
> interpret_graph (create_singleton_graph shift) "_aa"
([->,_] >> _ << ['a','a',_],[->,'a'] >> 'a' << [_])

> interpret_graph (create_singleton_graph copy) "_bb"
([->,_,'b','b',_,'b','b'] >> _ << [_],[->,'b'] >> 'b' << [_])
```