Αρχές Γλωσσών Προγραμματισμού

Ημερομηνία ανάρτησης: 21/01/2022
Ημερομηνία παράδοσης: 27/02/2022

Άσκηση 3
Κανονικές εκφράσεις

Εισαγωγή

Για την άσκηση αυτή καλείστε να φτιάξετε έναν διερμηνέα για κανονικές εκφράσεις. Η άσκηση χωρίζεται σε τρία κύρια μέρη:

1) Διάβασμα κανονικής έκφρασης και δημιουργία μη ντετερμινιστικού αυτόματου αναπαράστασης (NFA)
2) Μετατροπή μη ντετερμινιστικού αυτόματου σε ντετερμινιστικό (DFA)
3) Ταίριασμα συμβολοσειρών με βάση δοσμένη κανονική έκφραση

Δεν υπάρχουν προαπαιτούμενες γνώσεις για όλα τα παραπάνω, καθώς δίνεται εξήγηση στη συνέχεια για τη θεωρία και τους αλγορίθμους που θα χρειαστεί να υλοποιήσετε.

Κανονικές εκφράσεις

Οι κανονικές εκφράσεις είναι απλά ένας τρόπος να περιγράψεις ένα σύνολο από συμβολοσειρές. Το σύνολο αυτό αλλιώς ονομάζεται και γλώσσα της κανονικής έκφρασης. Η πιό μικρή κανονική έκφραση αποτελείται από ένα και μόνο γράμμα και αναπαριστά το μονοσύνολο συμβολοσειρών που αποτελούνται από αυτό το γράμμα. Με τη χρήση ορισμένων πράξεων, ωστόσο, μπορούμε να συνθέσουμε και να παράξουμε πιο πολύπλοκες κανονικές εκφράσεις. Μπορούμε να κατασκευάσουμε κανονικές εκφράσεις ως εξής:

1) Ένα απλό γράμμα, είναι μία κανονική έκφραση. Παραδείγματος χάριν η κ.ε. "a" συμβολίζει το σύνολο {"a"}
2) Κενό σύμβολο "ϵ" . Αυτό το σύμβολο μπορεί να περιγράψει την κενή συμβολοσειρά {""}
3) Εάν A και B δύο κανονικές εκφράσεις, τότε και η παράθεση τους (concatenation) AB είναι επίσης κανονική έκφραση. Πχ. "ab" συμβολίζει το σύνολο {"ab"}
4) Εάν A και B δύο κανονικές εκφράσεις, τότε και η ένωση τους (union) είναι κανονική έκφραση, και αναπαρίσταται ως “A|B”. Το σύνολο της ένωσης δύο κανονικών εκφράσεων είναι και η ένωση των επιμέρους συνόλων τους. Πχ. "$a|bc$"συμβολίζει το σύνολο {"a"} $\cup$ {"bc"} $=$ {"a", "bc"}
5) Εάν A κανονική έκφραση, τότε και το αστέρι Kleene (Kleene star) A* είναι κανονική έκφραση. Το Kleene star συμβολίζει μηδέν ή περισσότερες φορές το σύνολο της κανονικής έκφρασης στην οποία εφαρμόζεται. Πχ. "(abc) * " συμβολίζει το σύνολο {"", "abc", "$abcabc$", "$abcabcabc$", ...}.

Οι παρενθέσεις χρησιμοποιούνται, όπως και στις μαθηματικές εκφράσεις, όταν θέλουμε να εκφράσουμε ξεκάθαρα προτεραιότητες. Διαφορετικά, το Kleene star έχει την μεγαλύτερη προτεραιότητα και ακολουθείται από την παράθεση, και τελευταία είναι η ένωση.

Μερικά παραδείγματα:

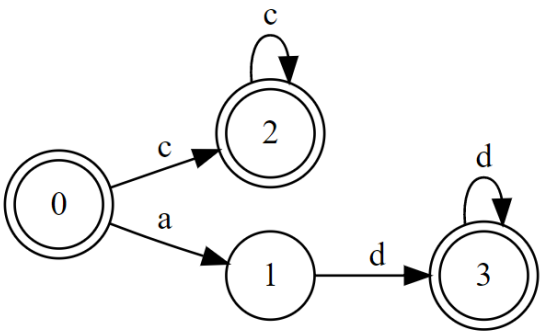
- $a|(bc)\ *$ συμβολίζει το σύνολο {"a","", "bc", "$bcbc$", "$bcbcbc$", …} Προσοχή, εφόσον το δεύτερο μέρος της ένωσης είναι Kleene star, το σύνολο περιέχει και την κενή συμβολοσειρά.
- "$abcd\ *$ " συμβολίζει το σύνολο {"abc", "$abcd$", "$abcdd$", "$abcddd$", …}. Το Kleene star εδώ εφαρμόζεται μόνο στο d.
- "$(a\ *\ |(bc))\ *$ " συμβολίζει το σύνολο {"", "a", "bc", "aa", "abc", "$bcbc$", "bca", "$aabc$", "$abcabc$", …}
- "$c(ab|\epsilon)$" εδώ, χρησιμοποιώντας το κενό σύμβολο, μπορούμε να εκφράσουμε ότι θέλουμε το δεύτερο μέρος να εμφανίζεται μηδέν ή μία φορά. Συνεπώς έχουμε το σύνολο {"c", "cab"}

Πεπερασμένα αυτόματα (Finite state machines)

Κάθε κανονική έκφραση μπορεί να περιγραφεί από (τουλάχιστον) ένα πεπερασμένο αυτόματο. Τα πεπερασμένα αυτόματα είναι απλές “υπολογιστικές μηχανές”. Τα π.α. αποτελούνται από ένα σύνολο από καταστάσεις, και από κάθε κατάσταση μπορούν να μεταβούν σε άλλες καταστάσεις διαβάζοντας σύμβολα εισόδου. Για το λόγο αυτό, μπορούν να αναπαρασταθούν σαν γράφοι με τους κόμβους να αποτελούν τις καταστάσεις και τις ακμές να αποτελούν τις μεταβάσεις μεταξύ των καταστάσεων. Τα χαρακτηριστικά ενός πεπερασμένου αυτόματου είναι:

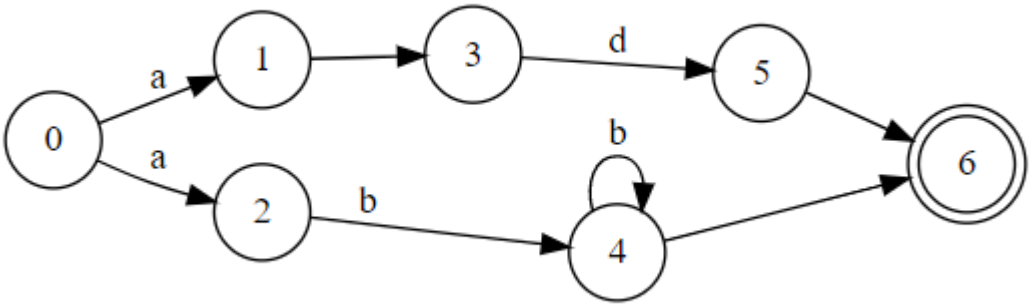
1) Ένα πεπερασμένο σύνολο καταστάσεων K
2) Ένα αλφάβητο εισόδου Σ
3) Μια συνάρτηση μετάβασης δ: $K \times \Sigma \rightarrow K$
4) Μία κατάσταση $s \in K$ που ονομάζεται αρχική κατάσταση
5) Ένα υποσύνολο του K, $F \subseteq K$, που ονομάζεται σύνολο τελικών καταστάσεων

Η συνάρτηση μετάβασης περιγράφει πως λειτουργεί το αυτόματο. Είναι ουσιαστικά μία συνάρτηση με παραμέτρους μία κατάσταση και ένα σύμβολο εισόδου και η τιμή της είναι η επόμενη κατάσταση του αυτόματου. Η αρχική κατάσταση είναι η κατάσταση στην οποία βρίσκεται το αυτόματο πριν κάνει οποιαδήποτε μετάβαση. Οι τελικές καταστάσεις είναι οι καταστάσεις στις οποίες αν καταλήξει το αυτόματο λέμε πως η είσοδος γίνεται αποδεκτή.



Το παραπάνω αυτόματο, για παράδειγμα, θα μπορούσε να περιγράφει την κανονική έκφραση ¨$c\ *\ |add\ *$ ". Η αρχική κατάσταση είναι ο κόμβος 0. Διαβάζοντας την είσοδο “c”, το αυτόματο θα μεταβεί στην κατάσταση 2, η οποία είναι και τελική. Από αυτή την κατάσταση,

διαβάζοντας συνεχόμενα “c” καταλήγει πάλι στην ίδια κατάσταση. Αντίστοιχα και για το “a” και το “d”. Η αρχική κατάσταση, σε αυτό το αυτόματο, είναι και τελική, και ως αποτέλεσμα δέχεται και την κενή συμβολοσειρά. Συνεπώς μπορούμε να δούμε πως δέχεται το σύνολο $\{"", "c", "ad", "cc", "add", "ccc", "addd", \ldots\}$. Αυτό το αυτόματο ονομάζεται ντετερμινιστικό αυτόματο (Deterministic Finite Automaton - DFA) γιατί από κάθε κατάσταση υπάρχει μόνο μία μετάβαση για κάθε σύμβολο εισόδου. Σε αντίθεση, υπάρχουν και τα μη ντετερμινιστικά αυτόματα (Non-deterministic Finite Automaton - NFA).



Ένα παράδειγμα NFA είναι το παραπάνω. Όπως φαίνεται, η αρχική κατάσταση 0 έχει δύο μεταβάσεις για το σύμβολο “a”. Αυτό σημαίνει πως αν δοθεί “a” σαν είσοδος, το αυτόματο μπορεί να μεταβεί ή στην κατάσταση 1 ή στην κατάσταση 2. Εναλλακτικά μπορεί να φανταστεί κανείς ότι το αυτόματο βρίσκεται ταυτόχρονα και στις δύο αυτές καταστάσεις (αυτή η διαφορετική θεώρηση θα μας βοηθήσει να κατανοήσουμε καλύτερα τη μετατροπή ενός NFA σε DFA). Επιπλέον, τα NFA έχουν τη δυνατότητα για κενές μεταβάσεις, όπως φαίνεται στις καταστάσεις 1, 4 και 5. Οι κενές μεταβάσεις μπορούν να αλλάξουν την κατάσταση του αυτόματου χωρίς είσοδο, και θεωρητικά αυτό σημαίνει πως το αυτόματο “βρίσκεται” και στις δύο καταστάσεις. Στο αυτόματο του παραδείγματος, εάν η τωρινή κατάσταση είναι η 5, μπορεί να θεωρηθεί πως και η 6 είναι τωρινή. Τέλος, εάν εξετάσετε το αυτόματο του παραδείγματος, θα δείτε πως αναπαριστά την κανονική έκφραση $"ad|abb\ *\ "$.

Τύποι δεδομένων

Για να υλοποιήσετε τα ζητούμενα της άσκησης προτείνεται να βασιστείτε στους ακόλουθους τύπους.

```
type StateId = Int
type TransChar = Char
type Inputs = [TransChar]
type Transition = (StateId, StateId, TransChar)
type States = [StateId]
type Transitions = [Transition]
type FirstState = StateId
type LastStates = States
type Fsa = (States, Inputs, Transitions, FirstState, LastStates)
```

Για την διευκόλυνση σας, έχει γραφτεί parser για κανονικές εκφράσεις που σας δίνεται. Χρειάζεται απλά να γράψετε `import RegParser` στο αρχείο κώδικα σας, αφού πρώτα εγκαταστήσετε το parsec με την ακόλουθη εντολή

```
cabal install parsec
```

Μπορείτε να κάνετε parse ένα string καλώντας την συνάρτηση

```
parseRegexpr :: [Char] -> RegExpr
```

Αυτή η συνάρτηση θα σας επιστρέψει την κανονική έκφραση με τον εξής τύπο:

```
data RegExpr = Letter(Char)
             | AnyLetter
             | EmptyChar
             | Kleene(RegExpr)
             | Concat(RegExpr, RegExpr)
             | Union(RegExpr, RegExpr)
```

Όπως και οι κανονικές εκφράσεις, ο τύπος είναι αναδρομικός. Για παράδειγμα, αν κάνετε parse το "$(acd|b)\ *$ ", θα πάρετε το αντικείμενο
Kleene (Union (Concat (Letter ‘a’, Concat (Letter ‘c’, Letter ’d’))), Letter ’b’)).
Το σύμβολο κάτω παύλα “_” χρησιμοποιείται για την αναπαράσταση του κενού συμβόλου, και το σύμβολο τελεία “.” μεταφράζεται στον τύπο AnyLetter, ο οποίος εξηγείται στο ερώτημα 4.

Ζητούμενες συναρτήσεις

1. Δημιουργία αυτόματου από κανονική έκφραση (40%)

```
makeNfa :: [Char] -> Fsa
```

Η συνάρτηση αυτή θα δέχεται σαν όρισμα την κανονική έκφραση ως συμβολοσειρά, και θα κατασκευάζει έναν γράφο για το μη ντετερμινιστικό αυτόματο ο οποίος θα ακολουθεί τον τύπο Fsa όπως περιγράφεται. Για να υλοποιήσετε αυτό το μέρος, μπορείτε να χρησιμοποιήσετε τον αλγόριθμο Thompson που δίνεται στη συνέχεια, αλλά δεν είναι υποχρεωτικό να τον ακολουθήσετε αυστηρά.

2. Μετατροπή NFA σε DFA (40%)

```
nfaToDfa :: Fsa -> Fsa
```

Από το ερώτημα 1 θα έχετε κατασκευάσει ένα μη ντετερμινιστικό αυτόματο. Πιθανώς αυτό το αυτόματο να έχει πολλαπλές μεταβάσεις με το ίδιο σύμβολο ή κενές μεταβάσεις. Θα πρέπει να το μετατρέψετε σε ισοδύναμο ντετερμινιστικό αυτόματο. Για να το υλοποιήσετε αυτό, μπορείτε να ακολουθήσετε τον αλγόριθμο που επίσης σας δίνεται.

3. Ταίριασμα συμβολοσειράς με κανονική έκφραση (20%)

```
regexFullMatch :: ([Char], [Char]) -> Bool
```

Σε αυτή τη συνάρτηση χρησιμοποιώντας τα αυτόματα που δημιουργήσατε στα άλλα ερωτήματα θα πρέπει να κάνετε ταίριασμα συμβολοσειρών με μία δοσμένη κανονική

έκφραση (το πρώτο όρισμα). Δηλαδή, η συνάρτηση θα επιστρέφει True εάν η συμβολοσειρά γίνεται δεκτή από το αυτόματο και δεν υπάρχουν άλλοι χαρακτήρες στην είσοδο.

Bonus ερωτήματα (25%)

4. Ταίριασμα wildcard (10%)

Για αυτό το ερώτημα δεν χρειάζεται να υλοποιήσετε κάποια άλλη συνάρτηση, απλά να κάνετε τις προηγούμενες υλοποιήσεις να δουλεύουν και για “wildcard”. Το wildcard αναπαρίσταται σε μία κανονική έκφραση με το σύμβολο τελεία ‘.’, και στο αντικείμενο που επιστρέφει ο parser είναι ο τύπος AnyLetter. Η σημασία του είναι ότι θα ταίριαζε με οποιοδήποτε γράμμα. Για παράδειγμα η κανονική έκφραση "$a\,.\,b$" έχει ως σύνολο το $\{"aab", "abb", "acb", "adb", "aeb", \ldots\}$. Με βάση τους αλγόριθμους που σας δίνονται, μπορείτε να σκεφτείτε πως αυτοί επεκτείνονται για αυτή την περίπτωση.

5. Μερικό ταίριασμα συμβολοσειράς (10%)

```
regexPartMatch :: ([Char], [Char]) -> [[Char]]
```

Για αυτή την συνάρτηση, παρόμοια με την συνάρτηση του ερωτήματος 3, θα πρέπει να ταιριάξετε μια κανονική έκφραση με μια συμβολοσειρά. Η διαφορά είναι όμως πως αντί να επιστρέφετε True ή False, θα επιστρέφετε όλα τα matches που βρίσκετε μέσα σε μία λίστα.

6. Readme (5%)

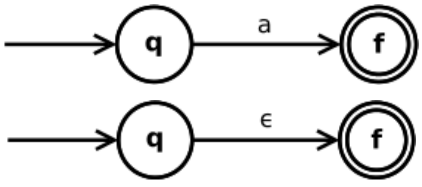
Γράψτε περιληπτικά τι έχετε υλοποιήσει.

Παράρτημα Α: Χρήσιμοι αλγόριθμοι

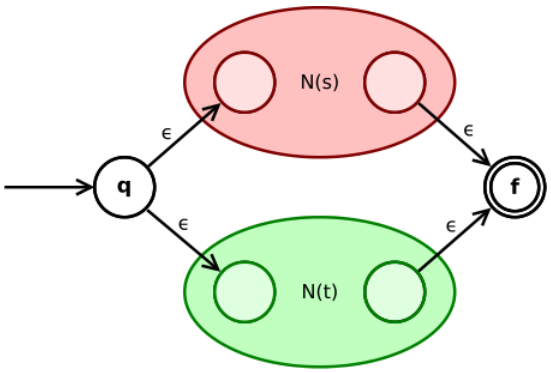
Αλγόριθμος Thompson για κατασκευή αυτόματου από κανονική έκφραση

Ο αλγόριθμος λειτουργεί αναδρομικά, κατασκευάζοντας μικρά NFAs για τις υποεκφράσεις μιας κανονικής έκφρασης και τις συνδέει με κενές μεταβάσεις. Η αναδρομή φτάνει μέχρι την πιό βασική κανονική έκφραση η οποία είναι ένα σύμβολο ή το κενό σύμβολο. Τα NFAs συνδέονται ως εξής:

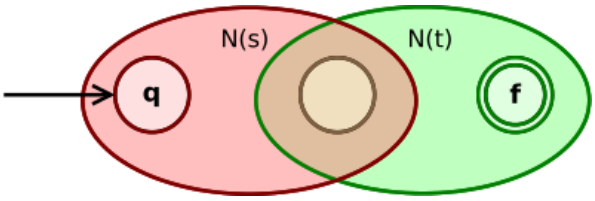
Ο απλός χαρακτήρας και το κενό σύμβολο απλά παράγουν μια ακμή από την αρχική κατάσταση στην τελική.



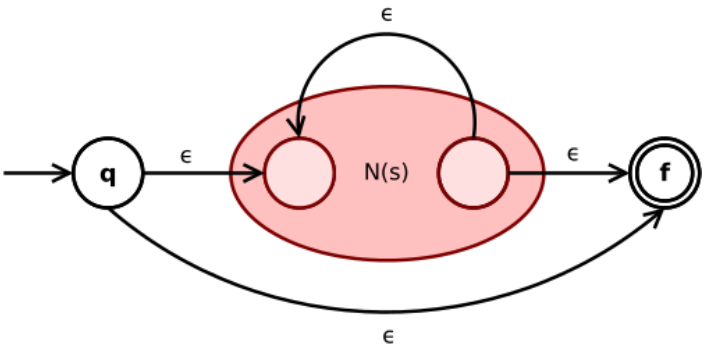
Η ένωση δύο κανονικών εκφράσεων s και t, s|t, μετατρέπεται στα δύο μικρότερα NFAs του s και του t αντίστοιχα. Η αρχική κατάσταση του συνολικού NFA ενώνεται με τις αρχικές καταστάσεις των μικρότερων NFAs, και οι τελικές καταστάσεις τους ενώνονται με την τελική κατάσταση του ολικού NFA.



Για την μετάθεση δύο εκφράσεων s και t, αρκεί να ταυτήσουμε την τελική κατάσταση του s με την αρχική κατάσταση του t.



Τέλος, για το Kleene star μιας έκφρασης s, η αρχική και τελική κατάσταση συνδέονται με κενές μεταβάσεις με την αρχική και τελική κατάσταση του NFA του s, αντίστοιχα. Ακόμα, η τελική κατάσταση του NFA του s συνδέεται με κενή μετάβαση με την αρχική του κατάσταση, ενώ η αρχική κατάσταση του εξωτερικού NFA συνδέεται με την τελική.



Για περισσότερες λεπτομέρειες μπορείτε να δείτε **εδώ.**

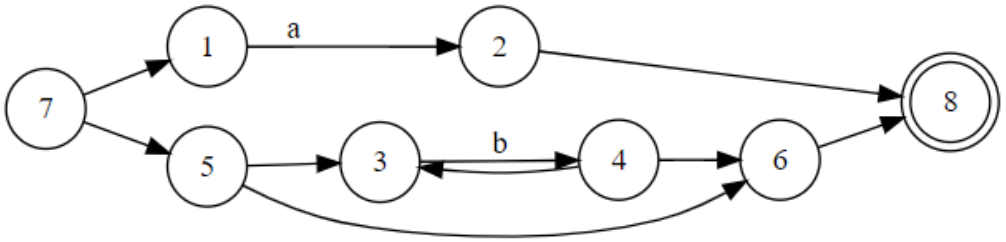
Αλγόριθμος για μετατροπή NFA σε DFA

Ο αλγόριθμος αποτελείται από τα ακόλουθα βήματα:

1) Ξεκινάμε εξετάζοντας την αρχική κατάσταση του NFA.
2) Για την κατάσταση που εξετάζουμε, βρίσκουμε το σύνολο των καταστάσεων οι οποίες συνδέονται με κενή μετάβαση. Το σύνολο αυτών των καταστάσεων, μαζί με την κατάσταση που εξετάζουμε, θα είναι μια καινούργια κατάσταση του DFA.
3) Για κάθε γράμμα του αλφαβήτου του αυτόματου, βρίσκουμε τις καταστάσεις που μπορεί να μεταβεί το αυτόματο από την καινούργια σύνολο-κατάσταση. Για να το υπολογίσουμε αυτό πρέπει να εξετάσουμε τις μεταβάσεις όλων των καταστάσεων του συνόλου. Κάθε σύνολο καταστάσεων που μπορούμε να φτάσουμε θα θεωρηθεί ως κατάσταση του DFA.
4) Για κάθε καινούργια κατάσταση που βρίσκουμε, και δεν έχουμε ξαναεξετάσει επαναλαμβάνουμε τα βήματα από το βήμα 2.

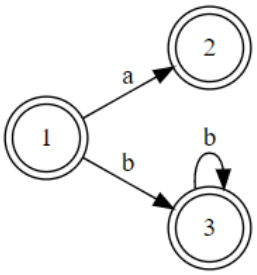
5) Οι καταστάσεις του DFA που “περιέχουν” κάποια τελική κατάσταση του NFA είναι τελικές καταστάσεις του DFA
6) Η αρχική σύνολο-κατάσταση που περιέχει την αρχική κατάσταση του NFA είναι η αρχική κατάσταση του DFA

Για να το δούμε με ένα παράδειγμα, θα πάρουμε το αυτόματο που δημιουργείται από τον αλγόριθμο Thompson για την έκφραση "$a|b\ *$ ". Η αρχική κατάσταση είναι η 7.



Καταστάσεις	ε	a	b
7	{1,7,5,3,6,8}		
{1,7,5,3,6,8}		{2,8}	{3,4,6,8}
{2,8}	{2,8}		
{2,8}		-	-
{3,4,6,8}	{3,4,6,8}		
{3,4,6,8}		-	{3,4,6,8}

Τελικά καταλήγουμε με τις καινούργιες καταστάσεις-σύνολα {1,7,5,3,6,8}, {2,8}, {3,4,6,8}, τις οποίες αν μετονομάσουμε σε 1, 2 και 3 αντίστοιχα, προκύπτει το ισοδύναμο ντετερμινιστικό αυτόματο.



Παράρτημα Β: Παραδείγματα εκτέλεσης

Δίνονται μερικά παραδείγματα εκτέλεσης, όμως προφανώς, τα δικά σας αποτελέσματα μπορεί να είναι σε διαφορετική σειρά ή οι κόμβοι να έχουν διαφορετική αρίθμηση. Το σημαντικό είναι να λειτουργούν σωστά και να υπάρχει σωστή αναπαράσταση.

```
makeNfa "a|b"
([1,2,3,4,5,6],"ab",[(1,2,'a'),(3,4,'b'),(5,1,'_'),(5,3,'_'),(2,6,'_'),(
4,6,'_')],5,[6])


makeNfa "(a|b)*"
([1,2,3,4,5,6,7,8],"ab",[(1,2,'a'),(3,4,'b'),(5,1,'_'),(5,3,'_'),(2,6,'_
'),(4,6,'_'),(7,5,'_'),(6,8,'_'),(7,8,'_'),(6,5,'_')],7,[8])
makeNfa "(aa|bb)*"
([1,2,3,4,5,6,7,8,9,10,11,12],"ab",[(1,2,'a'),(3,4,'a'),(2,3,'_'),(5,6,'
b'),(7,8,'b'),(6,7,'_'),(9,1,'_'),(9,5,'_'),(4,10,'_'),(8,10,'_'),(11,9,
'_'),(10,12,'_'),(11,12,'_'),(10,9,'_')],11,[12])
makeNfa "(a)|(b|a*)"
([1,2,3,4,5,6,7,8,9,10,11,12],"ab",[(1,2,'a'),(3,4,'b'),(5,6,'a'),(7,5,'
_'),(6,8,'_'),(7,8,'_'),(6,5,'_'),(9,3,'_'),(9,7,'_'),(4,10,'_'),(8,10,'
_'),(11,1,'_'),(11,9,'_'),(2,12,'_'),(10,12,'_')],11,[12])


nfaToDfa(makeNfa "a|b")
([1,2,3],"ab",[(1,2,'a'),(1,3,'b')],1,[3,2])
nfaToDfa(makeNfa "(a|b)*")
([1,2,3],"ab",[(1,1,'a'),(2,1,'a'),(1,2,'b'),(2,2,'b'),(3,1,'a'),(3,2,'b
')],3,[3,2,1])
nfaToDfa(makeNfa "(a)|(b|a*)")
([1,2,3,4],"ab",[(1,2,'a'),(1,3,'b'),(2,4,'a'),(4,4,'a')],1,[4,3,2,1])


regexFullMatch("ab|c","ab")
True
regexFullMatch("ab|.","abd")
False
regexFullMatch("(ab|.)*","abd")
True

regexPartMatch("(ab)*", "abababde")
["", "ab","abab","ababab"]
regexPartMatch("(ac|dd)*", "acacddeff")
["","ac","acac","acacdd"]
```

Απορίες

Για τυχόν απορίες σχετικά με την εργασία μπορείτε να επικοινωνήσετε μαζί μου μέσω email στο m.gkioka@di.uoa.gr.

Μαρκέλλα Γκιόκα