ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ, ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Πτυχιακή εργασία

Κατασκευή Front-End για την γλώσσα R

Αλέξανδρος Α. Πλέσσιας

2025201100068

Επιβλέποντες:

ΜΑΣΣΕΛΟΣ Κ. Γ.

Καθηγητής

ΔΗΜΗΤΡΟΥΛΑΚΟΣ Γ.

Μέλος Ε.ΔΙ.Π.

Τρίπολη, Μάρτιος 2017

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Περίληψη

Στην παρούσα εργασία θα μελετήσουμε την κατασκευή του εμπρός τμήματος της στατιστικής γλώσσας R που αποτελείτε από έναν λεκτικό αναλυτή και δυο συντακτικούς αναλυτές, ο ένας λειτουργεί σαν φίλτρο της γραμματικής και ο άλλος είναι η καθαυτού γραμματική. Για κάθε είσοδο θα παράγεται ένα συντακτικό δένδρο, ένα αφηρημένο συντακτικό δένδρο, η απεικόνιση του συντακτικού δένδρου, η απεικόνιση του αφηρημένο συντακτικό δένδρο και ο πίνακα συμβόλων.

Στο εμπρός τμήμα του μεταγλωττιστή εφαρμόζονται αρκετά μοτίβα σχεδίασης, τα οποία είναι αντικειμενοστραφή έτσι μας επιτρέπουν την εύκολη συντήρηση του κώδικα αλλα και την καλύτερη οργάνωση του. Τέλος έγιναν αρκετές δοκιμές και αναλύσεις στα αποτελέσματα ώστε να είναι σωστή τόσο η γραμματική όσο και η εργασία σαν σύνολο.

Λέξεις κλειδιά: μεταγλωττιστής, εμπρός τμήμα μεταγλωττιστή, στατιστική γλώσσα R.

Abstract

In the present work we study the structure of the front end of the statistical language R, which consists of a lexical analyzer and two parsers, one acts as grammar filter and the other is the grammar itself. For each input will produce a parse tree, an abstract syntax tree (AST), the illustration of syntax tree, the representation of abstract syntax tree (AST) and a symbol table.

In front end (FE) of the compiler implemented several design patterns, which are object oriented so allow us to easily maintain the code but also the best organization. Finally they made several tests and analyzes on the results in order to have correct both the grammar and the work as a whole.

Keywords: compiler, front end (FE) of compiler, statistical language R.

1 Εισαγωγή

1.1 Περιγραφή του προβλήματος

Η διπλωματική εργασία αναφέρεται στην σχεδίαση του εμπρόσθιου τμήματος (front end) μεταγλωττιστή για την γλώσσα R. Η γλώσσα R έχει μια μεγάλη βάση χρηστών (περίπου 2 εκατομμύρια) και αυτό την κάνει πολύ ενδιαφέρουσα σε ότι αφορά της εφαρμογές στην σχεδίαση ψηφιακών συστημάτων για την επιτάχυνση της επεξεργασίας των προγραμμάτων της. Στην αγορά δεν υπάρχουν πολλά και αξιόπιστα εργαλεία ανάπτυξης για την γλώσσα R με εξαίρεση το RStudio το οποίο θα αναλύσω στην πορεία. Εστί για αρχή δημιουργώντας το εμπρόσθιο τμήμα που αποτελείτε από τον λεκτικό αναλυτή, τον συντακτικό αναλυτή, την οπτικοποίηση της ενδιάμεσης αναπαράστασης και τον πίνακα συμβόλων, ανοίγει ο δρόμος για την δημιουργία ενός IDE της R.

Η σημαντικότερη τεχνική δυσκολία ήταν η δημιουργία της γραμματικής της R μιας και δεν υπήρχε κάπου διαθέσιμη, έτσι χρειάστηκε η ανάπτυξη της από την αρχή. Ακόμα έγινε χρήση κατάλληλων εργαλείων όπως το ANTLR για αντικειμενοστραφή προσέγγιση του προβλήματος και της γλώσσας C# για υλοποίηση των μοτίβων σχεδίαση του επισκέπτη, του παρατηρητή, του επαναλήπτη και της πρόσοψης.

1.1.1 Βιβλιογραφική αναφορά σε σχετικές εργασίες

Δεν υπάρχουν αναφορές οι οποίες να προσεγγίζουν απόλυτα το κομμάτι του Front End της γλώσσας R. Η μόνη αναφορά που μπορεί να γίνει είναι στους Ross Ihaka και Robert Gentleman για το άρθρο "R: A language for data analysis and graphics" στο περιοδικό στην Journal of Computational and Graphical Statistics (σελ. 299 - 314) το 1996 που αναφέρεται στην γέννηση της γλώσσας R.

1.1.2 Μέθοδο και τρόπο επίλυσης του προβλήματος

Οι μέθοδοι που χρησιμοποίησα για την επίλυση του προβλήματος ήταν η **υπόθεση** και ο **αναλυτικός τρόπος**. Η μέθοδος της υπόθεσης χρησιμοποιήθηκε για την δημιουργία της γραμματικής της R όπου στηριζόμουν σε υποθέσεις οι οποίες βελτίωναν την συγγραφή της γραμματικής μου, ειδικά σε κομμάτια τα οποία δεν ήταν ευνόητα από τον ορισμό της

γλώσσας. Από την άλλη ο αναλυτικός τρόπος με βοήθησε στο να επεξεργαστώ την γλώσσα R και τα εργαλεία τα οποία χρειαζόμουν, την επεξεργασία των σχέσεων ανάμεσα στα μοτίβα που εφάρμοσα, την παραγωγή αλγορίθμων, την συγγραφή του κώδικα της εργασίας και τέλος τον έλεγχο των αποτελεσμάτων.

1.2 Οργάνωση κειμένου

Τα εργαλεία που χρησιμοποίησα για την ανάπτυξη του προγράμματος αλλα και κάποια αλλα βοηθητικά παρουσιάζονται στο Κεφάλαιο 2. Το Κεφάλαιο 3 αναφέρεται στην περιγραφή της γλώσσας R μαζί με τον λεκτικό και τον συντακτικό αναλυτή της που έφτιαξα. Στο Κεφάλαιο 4 αναλύεται η περιγραφή και η χρήση των μοτίβων που χρησιμοποιήθηκαν. Η κατασκευή του εμπρόσθιου μέρους (front end) του μεταγλωττιστή της γλώσσας R μαζί με την περιγραφή κάποιων σημαντικών κλάσεων αναφέρεται στο Κεφάλαιο 5. Στο Κεφάλαιο 6 περιγράφονται οι κλάσεις που χρησιμοποιήθηκαν για την δημιουργία του συντακτικού δένδρου, αφηρημένου συντακτικού δένδρου και του πίνακα συμβόλων μαζί με την μέθοδο απεικόνισης τους. Στο Κεφάλαιο 7 περιγράφεται το πλαίσιο στο οποίο έγιναν οι δοκιμές της γραμματικής μαζί με κάποια ενδεικτικά αποτελέσματα αλλά και το που υπάρχουν όλα τα αποτελέσματα. Τέλος στο Κεφάλαιο 8 υπάρχουν κάποια συμπεράσματα αλλά και κάποιες μελλοντικές κατευθύνσεις της εργασίας.

2 Εργαλεία/Γλώσσες που χρησιμοποιήθηκαν

Για την εργασία χρησιμοποιήθηκαν διάφορα εργαλεία, από την δημιουργία της γραμματικής της R μέχρι την γραφική της αναπαράσταση. Πιο αναλυτικά χρησιμοποιήθηκαν τα εξής:

- Γλώσσας προγραμματισμού C#
- Στατιστική γλώσσα προγραμματισμού R
- Γεννήτρια συντακτικών αναλυτών ANTLR
- Visual Studio 2015 εργαλείο ανάπτυξης για την C#
- RStudio εργαλείο ανάπτυξης για την R
- Εργαλείο παραγωγής γράφων Graphviz

Στις παρακάτω υποενότητες θα αναλυθούν διεξοδικά, αναφορικά με τις δυνατότητες τους και την χρήση τους.

2.1 Γλώσσα προγραμματισμού C#

Η C# (C Sharp, ελληνική προφ. Σι Σάρπ) είναι μια γλώσσα προγραμματισμού σχεδιασμένη για την δημιουργία διαφόρων εφαρμογών που τρέχουν στο .NET Framework (Liberty & MacDonald, 2008). Η C# είναι απλή, ισχυρή, κάνει ασφαλή χρήση των τύπων της και είναι αντικειμενοστραφής. Οι πολλές καινοτομίες της C# βοηθούν στην ταχεία ανάπτυξη εφαρμογών διατηρώντας πάντα την εκφραστικότητα και την κομψότητα των γλωσσών τύπου C.

Η σύνταξη της C# είναι πολύ εκφραστική, αλλα επίσης και πολύ απλή και εύκολη στην εκμάθηση της. Στο συντακτικό της χρησιμοποιεί τις κυρτές αγκύλες το οποίο είναι άμεσα αναγνωρίσιμο από οποιονδήποτε είναι εξοικειωμένος με γλώσσες όπως οι C, C++ ή Java. Οι προγραμματιστές που ξέρουν κάποια από τις γλώσσες που προαναφέρθηκαν τυπικά μπορούν να δουλέψουν παραγωγικά με την C# μέσα σε ένα μικρό χρονικό διάστημα. Το συντακτικό της C# απλοποιεί πολλές πολυπλοκότητες της C++ και παρέχει δυνατά χαρακτηριστικά όπως τύπους που επιτρέπουν την τιμή null (nullable value types), enumerations, delegates, λάμδα

εκφράσεις (lambda expressions) και άμεση διαχείριση της μνήμης, τα οποία δεν μπορείς να τα βρεις στην Java. Η C# υποστηρίζει γενικευμένες μεθόδους και τύπους, οι οποίοι παρέχουν αυξημένη ασφάλεια στους τύπους και στην απόδοση, και iterators, οι οποίοι επιτρέπουν στις συλλογές μας να έχουν προσαρμοσμένη συμπεριφορά επανάληψης η οποία είναι εύκολο να χρησιμοποιηθεί από τον προγραμματιστή. Ένα πολύ δυνατό ακόμα χαρακτηριστικό είναι και οι εκφράσεις LINQ (Language INtegrated Query). Η έκδοση που χρησιμοποιήσαμε είναι η **C# 6**.

Σαν αντικειμενοστραφής γλώσσα, η C# υποστηρίζει έννοιες όπως ενθυλάκωση, κληρονομικότητα και πολυμορφισμός. Όλες οι μεταβλητές και οι μέθοδοι, συμπεριλαμβανομένης και την μεθόδου Main, δηλαδή το σημείο εισόδου μιας εφαρμογής, είναι ενθυλακωμένα σε έναν εσωτερικό ορισμό κλάσεων. Μια κλάση μπορεί να κληρονομήσει απ' ευθείας από την γονική κλάση αλλα μπορεί να εφαρμόσει και οποιαδήποτε interfaces.

Εκτός από τις βασικές αρχές της αντικειμενοστρέφειας, η C# καθιστά εύκολο να αναπτυχθεί λογισμικό μεσω διαφόρων καινοτομιών της όπως:

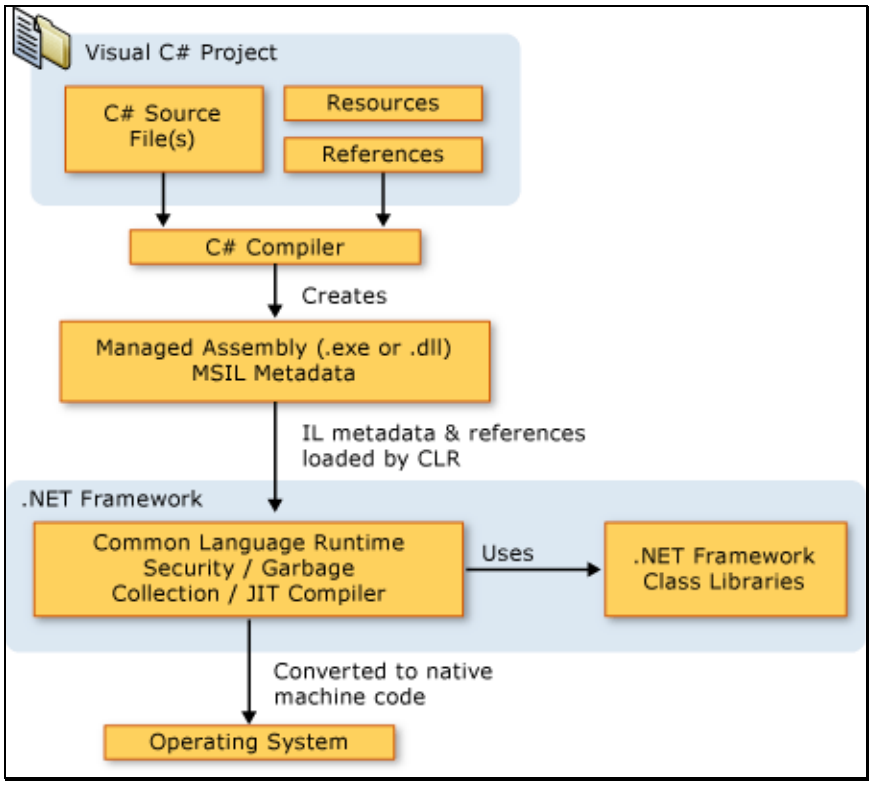
- Οι ενθυλακωμένες υπογραφές μεθόδων καλούνται **delegates**, οι οποίες επιτρέπουν την ειδοποίηση μεσω γεγονότων για την ασφαλή χρήση των τύπων.
- **Properties**, οι οποίες χρησιμοποιούνται ως accessors (getters και setters) για πρόσβαση σε ιδιωτικές μεταβλητές.
- **Attributes,** οι οποίες παρέχουν μεταδεδομένα σχετικά με τους τύπους σε χρόνο εκτέλεσης.
- **Ενσωματωμένη XML** τεκμηρίωση σχολίων.
- **LINQ (Language INtegrated Query)**, η οποία παρέχει την δυνατότητα ενσωμάτωσης ερωτήσεων σε μια ποικιλία από πηγές δεδομένων.

2.1.1 Πλατφόρμα αρχιτεκτονικής .NET Framework

Τα προγράμματα της C# τρέχουν στο .NET Framework, ένα αναπόσπαστο στοιχείο των Windows που περιλαμβάνει ένα εικονικό σύστημα εκτέλεσης το οποίο καλείτε **CLR** και ένα ενιαίο σύνολο από κλάσεις βιβλιοθηκών. Το CLR είναι εμπορική εφαρμογή της Microsoft του CLI, ένα διεθνές πρότυπο που είναι η βάση για την εκτέλεση και την ανάπτυξη στην

οποία οι γλώσσες και οι βιβλιοθήκες συνεργάζονται απόλυτα. Οι εκδόσεις που χρησιμοποιήσαμε είναι οι **.NET Framework 4.5 και 4.6.1**.

Ο πηγαίος κώδικας της C# μεταγλωττίζεται σε μια **ενδιάμεση γλώσσα (IL – Intermediate Language)** που είναι σύμφωνη με τις προδιαγραφές της CLI. Ο κώδικας της IL και οι πόροι, όπως τα bitmaps και οι συμβολοσειρές, αποθηκεύονται στο δίσκο σε ένα εκτελέσιμο αρχείο το οποίο καλείτε **assembly**, τυπικά είναι εάν αρχείο με προέκταση .exe ή .dll. Ένα assembly περιέχει ένα **manifest** το οποίο παρέχει πληροφορίες σχετικά με τον τύπο της assembly, την έκδοση και τις απατήσεις ασφαλείας.



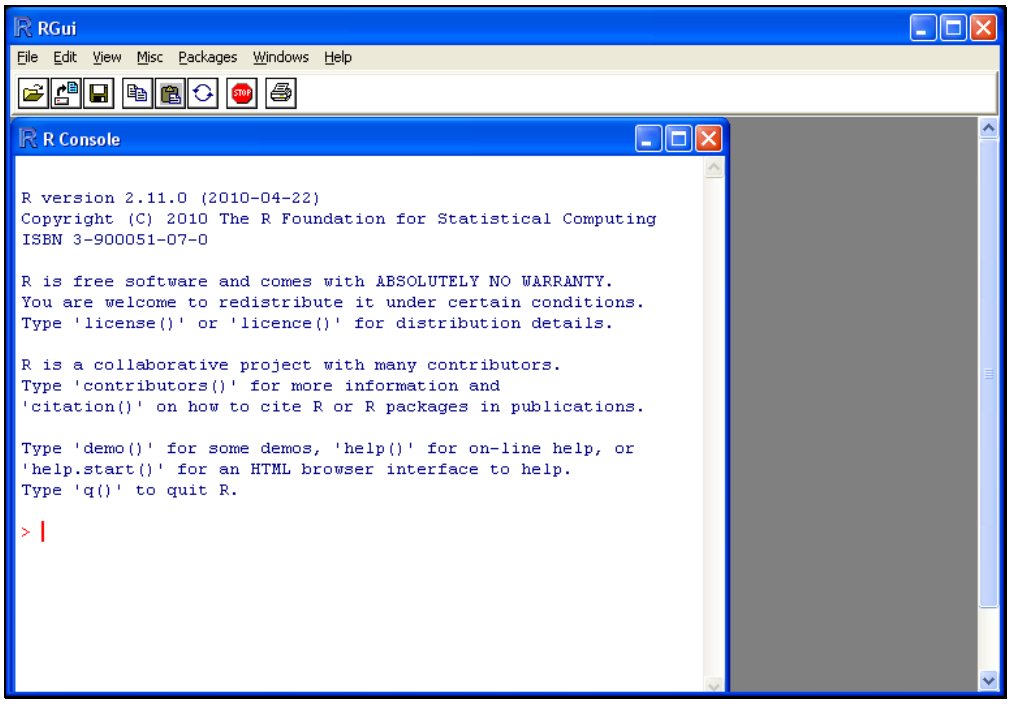
Εικόνα 2-1: Απεικόνιση των σχέσεων του μεταγλώττισης της C#, του .NET Framework, των κλάσεων των βιβλιοθηκών και των assemblies

Όταν εκτελείται ένα πρόγραμμα σε C#, το assembly αρχείο φορτώνεται στην CLR, το οποίο μπορεί να λάβει διάφορες ενέργειες με βάση τις πληροφορίες του manifest. Στη συνέχεια, αν οι απαιτήσεις ασφάλειας τηρούνται, η CLR εκτελεί τον **JIT** μεταγλωττιστή ώστε να μετατρέπει τον IL κώδικα σε εντολές μηχανής. Η CLR επίσης παρέχει και άλλες υπηρεσίες σχετικά με την αυτόματη συλλογή σκουπιδιών, την διαχείριση εξαιρέσεων πόρων. Ο κώδικας που εκτελείτε από την CRL μερικές φορές αναφέρεται ως "διαχειρίσιμος κώδικας"

σε αντίθεση με τον "μη διαχειριζόμενο κώδικα", ο οποίος είναι οι εντολές μηχανής του στοχευμένου συστήματος. Στην παραπάνω εικόνα φαίνεται η λειτουργία του .NET Framework.

2.2 Γλώσσα προγραμματισμού στατιστικής R

Η R προσφέρει ένα ολοκληρωμένο σύνολο υπηρεσιών λογισμικού για ανάλυση δεδομένων, υπολογισμών και γραφημάτων. Το λογισμικό R γράφτηκε αρχικά από τους Ross Ihaka και Robert Gentleman (Ihaka & Gentleman, 1996) στα μέσα της δεκαετίας του 90 (σε αυτούς οφείλει το όνομά της). Από το 1997 αναπτύσσεται από το R Development Core Team. Η R εφαρμόζει μια διάλεκτο της γλώσσας S, αλλά η R μπορεί να κάνει τα ίδια πράγματα με την S και με πολύ λιγότερο κώδικα. Είναι **γλώσσα ανοικτού κώδικα (open-source language)** αποτελεί μέρος του έργου GNU και επομένως, δεν απαιτείται η αγορά άδειας για τη χρήση της δηλαδή είναι δωρεάν, άρα όλοι μπορούν να έχουν πρόσβαση στον κώδικά της και να κάνουν διορθώσεις. Επιπλέον, επιτρέπει στον χρήστη να αλληλεπιδρά και με άλλες γλώσσες (C/C++, Java, Python), με αρχεία δεδομένων (Excel, Access) και με άλλα στατιστικά πακέτα (SAS, Stata, SPSS, Minitab).



Εικόνα 2-2: Απεικόνιση της κονσόλας της R

Η R δεν συνιστάται για ανάλυση μεγάλων δεδομένων. Το βασικό **μειονέκτημα** της R είναι ότι **καταναλώνει πολύ μνήμη**. Είναι 'αργή' γλώσσα και άρα ως προς τον χρόνο εκτέλεσης των εντολών δεν είναι τόσο αποδοτική.

Το βασικότερο **πλεονέκτημα** είναι ότι η R έχει πάνω από **5000 πακέτα** και χρησιμοποιείται σε πολλούς επιστημονικούς τομείς. Κατ' αρχάς χρησιμοποιείται σε **εταιρίες** όπως οι Microsoft, TIBCO, Google, Oracle, HP, LinkedIn και Facebook που κάνουν ανάλυση δεδομένων. Επιπλέον **χρησιμοποιείται και σε άλλους τομείς**, όπως στα οικονομικά, στην αστρονομία, στην χημεία, στην φαρμακευτική, στην ιατρική, στο μάρκετινγκ και αλλα.

Την τελευταία δεκαετία, η γλώσσα προγραμματισμού R έχει αποκτήσει μία εκρηκτική δημοσιότητα ενώ **υπολογίζεται ότι πάνω από τρία εκατομμύρια αναλυτές την χρησιμοποιούν** σαν εναλλακτική επιλογή ως κύριο στατιστικό εργαλείο για τις αναλύσεις τους. Παράλληλα, αποτελεί μία από τις καλύτερες επιλογές στην ακαδημαϊκή κοινότητα καθώς επιλύει απαιτητικά ερευνητικά ερωτήματα καλύπτοντας ένα ευρύ φάσμα από επιστήμες της βιολογίας και της ιατρικής μέχρι και τις οικονομικές επιστήμες. Το αποτέλεσμα είναι η ραγδαία αύξηση τόσο των αναλυτών που τη χρησιμοποιούν, όσο και των εφαρμογών που αναπτύσσονται σε αυτήν από μεγάλες εταιρίες που διαχειρίζονται δεδομένα, όπως οι εταιρείες που προαναφέραμε. Η έκδοση που χρησιμοποιήσαμε είναι η **R 3.3.2 (Sincere Pumpkin Patch)**.

2.3 Γεννήτρια συντακτικών αναλυτών ANTLR

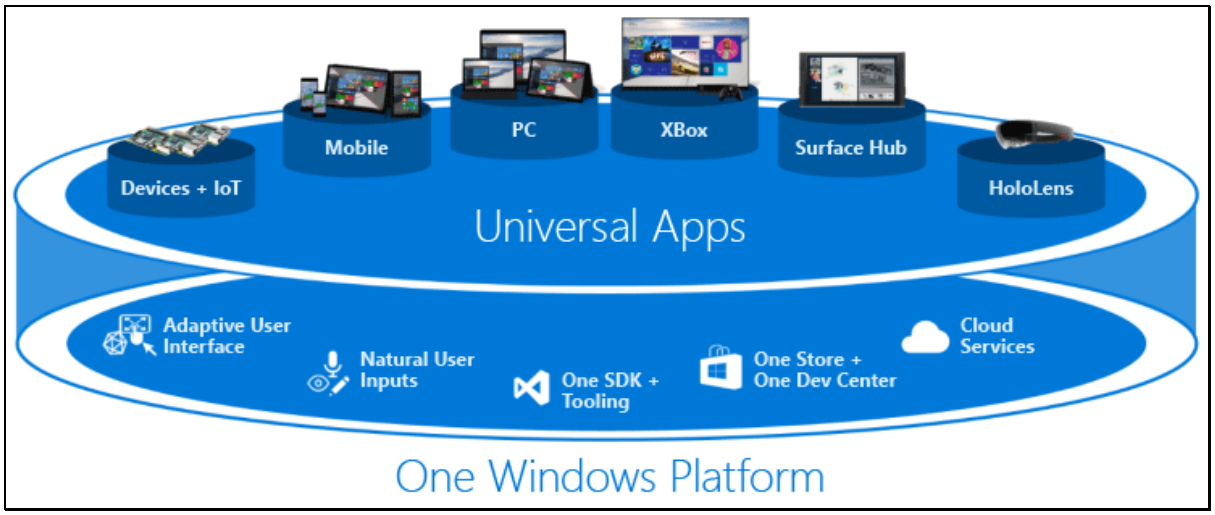
Το **ANTLR** (Parr, 2013) είναι μια δυνατή **γεννήτρια συντακτικών αναλυτών** (parser generator) για ανάγνωση, επεξεργασία, εκτέλεση ή μετάφραση δομημένου κειμένου ή δυαδικών αρχείων η οποία υπάρχει από το 1988. Χρησιμοποιείται ευρέως για την κατασκευή γλωσσών, εργαλείων και frameworks. Το ANTLR χρησιμοποιείτε ευρέως επειδή είναι εύκολο στην κατανόηση, δυνατό, ευέλικτο, γεννά έξοδο κατανοητή ως προς τον άνθρωπο, είναι **ελεύθερο λογισμικό υπό την άδεια BSD** και έχει ενεργή κοινότητα. Από μια γραμματική, το ANTLR παράγει έναν συντακτικό αναλυτή ο οποίος μπορεί να κατασκευάσει ένα συντακτικό δέντρο το οποίο στη συνέχεια μπορούμε να διασχίσουμε για να ανακτήσουμε όποια πληροφορία θελήσουμε. Η έκδοση που χρησιμοποιήσαμε είναι η **Antlr 4 4.5.3-rc1**.

Το ANTLR έχει έναν **κυρίαρχο ρόλο στην αγορά** των γεννητριών συντακτικών αναλυτών (parser generator) και έχει περίπου πέντε χιλιάδες (5000) προγραμματιστές που κατεβάζουν

το λογισμικό κάθε μήνα, το οποίο είναι μεγάλο ειδικά για ένα εξειδικευμένο προγραμματιστικό εργαλείο. Κάθε **μεγάλη εταιρία** όπως οι Twitter, Google, Oracle, IBM και Yahoo, έχουν μεγάλες εφαρμογές που βασίζονται στο ANTLR. Χαρακτηριστικά το Twitter χρησιμοποιεί το ANTLR για το parsing σχεδόν δύο (2) δισεκατομμύρια ερωτήσεων (αναζητήσεων) που γίνονται καθημερινά.

2.4 Visual Studio εργαλείο ανάπτυξης για την C#

Το Visual Studio είναι ένα IDE της Microsoft από το 1997. Χρησιμοποιείτε για την **ανάπτυξη προγραμμάτων για Windows**, καθώς και για ιστοσελίδες, εφαρμογές διαδικτύου, υπηρεσίες διαδικτύου και εφαρμογές για κινητά. Η νεότερη έκδοση είναι το Visual Studio 2015. Υποστηρίζει διαφορετικές γλώσσες προγραμματισμού και επιτρέπει την σύνταξη κώδικα και την αποσφαλμάτωση (σε διαφορετικούς βαθμούς) σχεδόν για κάθε γλώσσα προγραμματισμού από αυτές εφόσον υπάρχει. Η έκδοση που χρησιμοποιήσαμε είναι η **Microsoft Visual Studio Community 2015 (14.0.25422.01 Update 3)**.



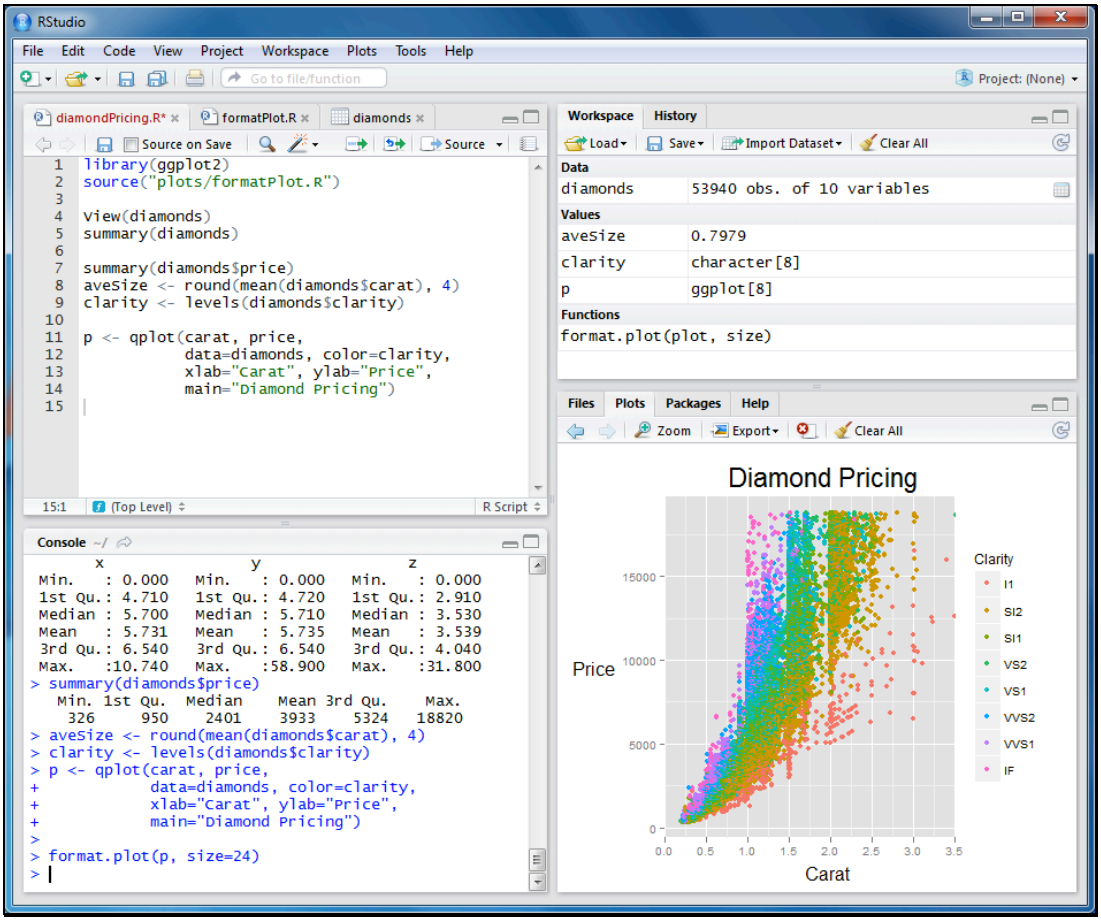
Εικόνα 2-3: Οι πλατφόρμες ανάπτυξης εφαρμογών του Visual Studio 2015

Οι ενσωματωμένες γλώσσες περιλαμβάνουν την C, C++ και C++/CLI (μεσω του Visual C++), VB.NET (μέσω Visual Basic . NET) C# (μέσω Visual C#) και F# (από το Visual Studio 2010). Υποστηρίζει και άλλες γλώσσες όπως Python, Ruby και Node.js. Ακόμα υποστηρίζει XML/XSLT, HTML/XHTML, JavaScript και CSS. Η Java και η J# υποστηρίζονταν στο παρελθόν.

2.5 RStudio εργαλείο ανάπτυξης για την R

Το RStudio είναι ένα **ανοιχτού κώδικα IDE** για την R, μια προγραμματιστική γλώσσα για στατιστικούς υπολογισμούς και γραφικά. Το RStudio ιδρύθηκε από τον J.J. Allaire το 2011 και είναι γραμμένο σε C++. Το RStudio είναι διαθέσιμο σε δυο (2) εκδόσεις την RStudio Desktop, όπου το πρόγραμμα τρέχει τοπικά στον ηλεκτρονικό υπολογιστή μας και την RStudio Server, όπου έχουμε πρόσβαση στο RStudio χρησιμοποιώντας έναν web browser ενώ τρέχει (το RStudio Server) σε έναν απομακρυσμένο Linux server. Η Desktop έκδοση είναι διαθέσιμη για Windows, macOs και Linux. Η έκδοση που χρησιμοποιήσαμε είναι η **RStudio 1.0.136**.

Το RStudio περιλαμβάνει μια κονσόλα, έναν επεξεργαστή κειμένου με επισήμανση του συντακτικού της γλώσσας και που υποστηρίζει άμεση εκτέλεση κώδικα, καθώς και εργαλεία για σχεδίαση, αποσφαλμάτωση, έναν χώρο εργασίας και το ιστορικό. Τα χαρακτηριστικά που προαναφέρθηκαν φαίνονται στην παρακάτω εικόνα.



Εικόνα 2-4: Το γραφικό περιβάλλον (GUI) του RStudio

Το RStudio έχει και δυο (2) εμπορικές εκδόσεις για οργανισμούς που δεν μπορούν να χρησιμοποιήσουν AGPL λογισμικό, η RStudio Desktop κοστίζει περίπου 950 ευρώ το χρόνο και η RStudio Server κοστίζει περίπου 9.450 ευρώ το χρόνο. Οι εταιρείες Microsoft, TIBCO, Google, Oracle, HP και άλλες χρηματοδοτούν το RStudio.

2.6 Εργαλείο παραγωγής γράφων Graphviz

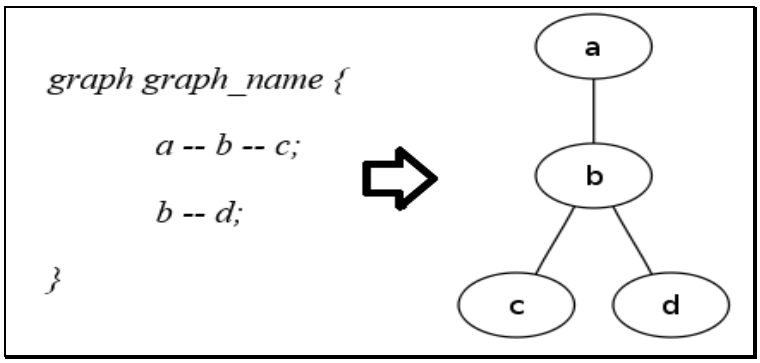
Το Graphviz (Graph Visualization Software) είναι ένα **εργαλείο ανοιχτού κώδικα** το οποίο ξεκίνησε από τα εργαστήρια AT&T το 2000 για τον σχεδιασμό γράφων στην γλώσσα DOT (Emden, et al., 2006) και είναι διαθέσιμο για Windowds, macOs και Linux. Παρέχει επίσης βιβλιοθήκες για να επιτρέπει σε εφαρμογές λογισμικού να χρησιμοποιούν αυτό το εργαλείο. Η έκδοση που χρησιμοποιήσαμε είναι η **Graphviz 2.38**.

2.6.1 Γλώσσα περιγραφής γράφων DOT

Η DOT είναι μια γλώσσα περιγραφής γράφων. Τα αρχεία γραφημάτων DOT τυπικά έχουν την **επέκταση gv ή dot**. Επιτρέπει κατευθυνόμενους και μη κατευθυνόμενους γράφους.

2.6.1.1 Μη κατευθυνόμενος γράφος

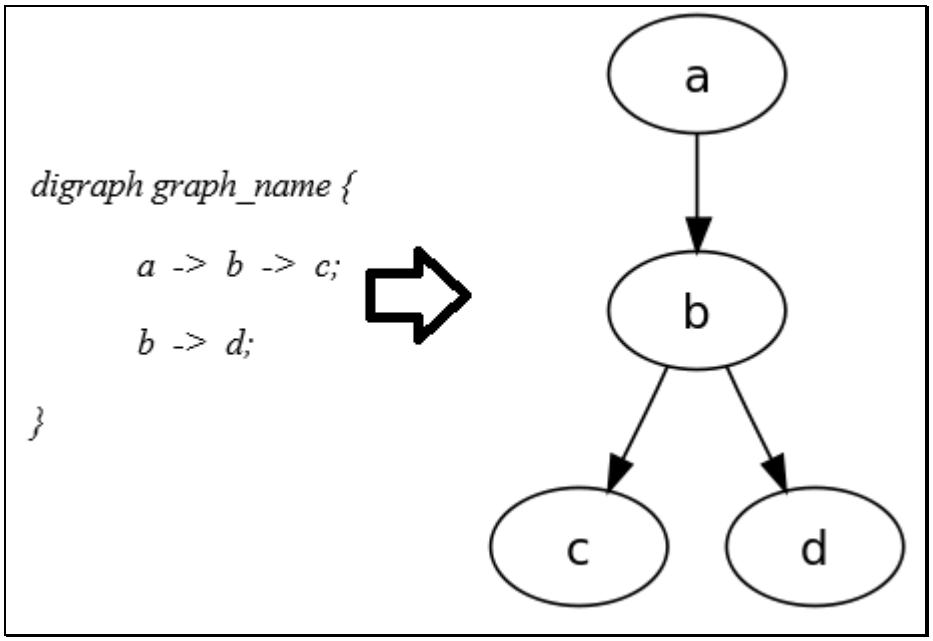
Στην απλούστερη μορφή της, η DOT μπορεί να χρησιμοποιηθεί για την περιγραφή ενός μη κατευθυνόμενου γράφου. Ένας μη κατευθυνόμενος γράφος δείχνει απλές σχέσεις μεταξύ των αντικειμένων. Η λέξη κλειδί "**graph**" χρησιμοποιείτε για να ξεκινήσει ένα νέο γράφημα και οι κόμβοι περιγράφονται μέσα στις **κυρτές αγκύλες**. Η **διπλή παύλα ('--')** χρησιμοποιείται για να δείξει τις σχέσεις μεταξύ των κόμβων. Ακολουθεί εικόνα με παράδειγμα κώδικα μαζί με τον γράφο που παράγεται.



Εικόνα 2-5: Μη κατευθυνόμενος γράφος, με χρήση DOT και ο γράφος που παράγεται

2.6.1.2 Κατευθυνόμενος γράφος

Πανόμοια με τους μη κατευθυνόμενου γράφους, η DOT μπορεί να περιγράψει έναν κατευθυνόμενο γράφο, όπως και διαγράμματα ροής και δέντρα με εξαρτήσεις. Το συντακτικό είναι ίδιο με αυτό του μη κατευθυνόμενου γράφου, εκτός από το ότι χρησιμοποιούμαι την λέξη κλειδί **"digraph"** για να ξεκινήσει ο γράφος, και με το **βέλος ('->')** δείχνουμε τις σχέσεις ανάμεσα στους κόμβους. Ακολουθεί εικόνα με παράδειγμα κώδικα μαζί με τον γράφο που παράγεται.



Εικόνα 2-6: Κατευθυνόμενος γράφος, με χρήση DOT και ο γράφος που παράγεται

2.6.1.3 Ιδιότητες

Διάφορες ιδιότητες (attributes) μπορούν να εφαρμοστούν σε ένα γράφημα, κομβο ή στις ακμές ενός DOT αρχείου. Αυτές οι ιδιότητες μπορούν να ελέγχουν ζητήματα όπως το **χρώμα**, το **σχήμα** και το **στυλ των γραμμών**. Οι ιδιότητες αυτές εισάγονται μέσα σε **αγκύλες ('[]')** και εάν έχουμε περισσότερες από μια τότε τις χωρίζουμε με κόμμα ή με περισσότερα ζεύγοι από αγκύλες.

3 Η γραμματική της R

3.1 Περιγραφή της γραμματικής της R

Στις παρακάτω υποενότητες θα αναλύσουμε την γραμματική της R όπως αυτή αναφέρεται στο εγχειρίδιο της (Team R Core, 2016) .

3.1.1 Σχόλια (Comments)

Τα σχόλια στην R αγνοούνται από τον parser. Κάθε κείμενο το οποίο ξεκινάει από τον χαρακτήρα # μέχρι το τέλος της γραμμής θεωρείτε σχόλιο, εκτός εάν ο χαρακτήρας # είναι μέσα σε ένα string το οποίο είναι μέσα σε απλά εισαγωγικά . Για παράδειγμα,

x <- 1 # Αυτό είναι σχόλιο…

y <- " # ... αλλα αυτό δεν είναι."

3.1.2 Σύμβολα (Tokens)

Τα σύμβολα είναι τα στοιχειώδη δομικά στοιχεία μιας γλώσσας προγραμματισμού. Αυτά (τα σύμβολα) αναγνωρίζονται κατά την διάρκεια της *λεκτικής ανάλυσης* κατά την οποία (θεωρητικά, τουλάχιστον) πραγματοποιείτε πριν από την *συντακτική ανάλυση* που πραγματοποιείτε από τον ίδιο τον parser.

3.1.2.1 Σταθερές (Constants)

Υπάρχουν πέντε (5) τύποι από σταθερές: ακέραιες, λογικές, αριθμητικές, μιγαδικές και αλφαριθμητικές.

Επιπλέον, υπάρχουν και τέσσερις (4) ειδικές σταθερές: NULL, NA, Inf και NaN. Το NULL χρησιμοποιείτε για να μας δείξει το κενό αντικείμενο. Το NA χρησιμοποιείτε για απούσες τιμές δεδομένων. Το Inf σημαίνει το άπειρο και το NaN δηλαδή μια απροσδιόριστη τιμή που χρησιμεύει στους υπολογισμούς κινητής υποδιαστολής, την συστηματική χρήση του την εισήγαγε η IEEE, για παράδειγμα NaN είναι το αποτέλεσμα των ενεργειών 1/0 και 0/0 αντίστοιχα.

Οι *λογικές σταθερές* είναι είτε το TRUE είτε το FALSE.

Οι **αριθμητικές σταθερές** ακολουθούν ένα όμοιο συντακτικό με αυτό της γλώσσας C. Αποτελούνται από ένα ακέραιο μέρος που αποτελείτε από μηδέν ή περισσότερα ψηφία, ακολουθούμενο από '.' και ένα δεκαδικό μέρος από μηδέν ή περισσότερα ψηφία προαιρετικά ακολουθούμενο από ένα εκθετικό μέρος που αποτελείται από το 'E' ή το 'e', ένα προαιρετικό πρόσημο και ένα αλφαριθμητικό με ένα ή περισσότερα ψηφία. Είτε το κλασματικό ή το δεκαδικό μέρος μπορεί να είναι άδειο, αλλά όχι και τα δύο ταυτόχρονα.

Έγκυρες αριθμητικές σταθερές: 1 10 0.1 .2 1e-7 1.2e+7

Οι αριθμητικές σταθερές μπορούν επίσης να είναι και **δεκαεξαδικές**, ξεκινώντας με '0X' ή '0x' ακολουθούμενο από μηδέν ή περισσότερα ψηφία, 'a-f' ή 'A-F'. Οι *σταθερές δεκαεξαδικής κινούμενης υποδιαστολής* υποστηρίζουν την C99 σύνταξης, π.χ. '0x1.1p1'.

Υπάρχει μια ξεχωριστή κλάση για τις **ακέραιες σταθερές**. Αυτές δημιουργούνται από την χρήση του προσδιορισμού L στο τέλος του αριθμού. Για παράδειγμα, το 123L είναι μια ακέραια τιμή και όχι μια αριθμητική τιμή. Το επίθημα L μπορεί να χρησιμοποιηθεί για να περαστεί κάθε μη-μιγαδικός αριθμός με σκοπό την δημιουργία ενός ακέραιου. Έτσι ώστε να μπορεί να χρησιμοποιηθεί με αριθμούς δοσμένους από δεκαεξαδικοί ή επιστημονική σημειογραφία. Ωστόσο, εάν η τιμή δεν είναι ένας έγκαιρος ακέραιος, μια προειδοποίηση θα εμφανιστεί και η αριθμητική τιμή θα δημιουργηθεί. Τα ακόλουθα παραδείγματα δείχνουν μια έγκυρη *ακέραια σταθερά*, τιμές οι οποίες θα παράξουν μια προειδοποίηση και θα δώσουν μια *αριθμητική σταθερά* και *συντακτικά λάθη*.

Έγκυρες ακέραιες σταθερές: 1L, 0x10L, 1000000L, 1e6L

Έγκυρες αριθμητικές σταθερές: 1.1L, 1e-3L, 0x1.1p-2

Συντακτικά λάθη: 12iL 0x1.1

Μια προειδοποίηση θα εμφανιστεί για τις δεκαδικές τιμές που περιέχουν μια περιττή υποδιαστολή, π.χ. 1.L. Είναι λάθος να έχεις υποδιαστολή σε μια δεκαεξαδική σταθερά χωρίς τον δυαδικό εκθέτη.

Σημειώστε επίσης ότι το προηγούμενο πρόσημο (+ ή -) θεωρείτε σαν μοναδιαίος τελεστής, όχι ως μέρος της σταθεράς.

Οι **μιγαδικές σταθερές** έχουν την μορφή ενός δεκαδικού αριθμού ακολουθούμενο από το 'i'. Παρατηρείστε ότι μόνο ο καθαρά φανταστικός αριθμός είναι πραγματική σταθερά, άλλοι

μιγαδικοί αριθμοί αναλύονται από μοναδιαίους ή δυαδικούς τελεστές στο αριθμητικό και φανταστικό μέρος τους.

Έγκυρες μιγαδικές σταθερές: 2i 4.1i 1e-2i

Οι **αλφαριθμητικές σταθερές** οριοθετούνται από ένα ζευγάρι μονών (‘’) ή διπλών (‘"’) εισαγωγικών και μπορούν να περιέχουν όλους τους άλλους εκτυπώσιμους χαρακτήρες. Τα εισαγωγικά και άλλοι ειδικοί χαρακτήρες μέσα στα αλφαριθμητικά καθορίζονται χρησιμοποιώντας **ακολουθίες διαφυγής**:

\' *μονά εισαγωγικά*

\" *διπλά εισαγωγικά*

\n *νέα γραμμή*

\r *επιστροφή φορέα (carriage return)*

\t *χαρακτήρας tab*

\b *οπισθοδιάστημα (backspace)*

\a *κουδούνισμα*

\f *form feed*

\v *κατακόρυφο tab*

\\ *ανάποδη κάθετος (σλας)*

\nnn *Τα byte nnn – είναι η ακολουθία από ένα, δυο ή τρία οκταδικά ψηφία με ένα εύρος 0 ... 7 είναι αποδεκτό.*

\xnn *Τα byte nn – είναι η ακολουθία από ένα ή δυο δεκαεξαδικά ψηφία με εύρος 0 ... 9, A ... F, και a ... f.*

\unnnn \u{nnnn} *(όπου οι τοπικές ρυθμίσεις υποστηρίζουν multibyte, αλλιώς υπάρχει λάθος). Κωδικοποίηση χαρακτήρα unicode για ένα δεδομένο δεκαεξαδικό κώδικα – ακολουθίες μέχρι τέσσερα (4) δεκαεξαδικά ψηφία. Ο χαρακτήρας πρέπει να είναι έγκυρος κατά τις τρέχουσες τοπικές ρυθμίσεις.*

\Unnnnnnnn \U{nnnnnnnn} *(όπου οι τοπικές ρυθμίσεις υποστηρίζουν multibyte και όχι για τα Windows, αλλιώς υπάρχει λάθος). Κωδικοποίηση χαρακτήρα unicode για ένα δεδομένο δεκαεξαδικό κώδικα – ακολουθίες μέχρι οκτώ (8) δεκαεξαδικά ψηφία*

Ένα μονό εισαγωγικό μπορεί επίσης να ενσωματωθεί άμεσα σε ένα αλφαριθμητικό οριοθετούμενο από διπλά εισαγωγικά και το αντίστροφο.

Από την έκδοση 2.8.0 της R, το 'nul' (\0) δεν επιτρέπεται σε ένα αλφαριθμητικό, έτσι χρησιμοποιώντας το \0 σε μια αλφαριθμητική σταθερά τερματίζει την σταθερά (συνήθως με μια ειδοποίηση) και οι επιπλέον χαρακτήρες μέχρι το κλείσιμο των εισαγωγικών σαρώνονται αλλά αγνοούνται.

3.1.2.2 Αναγνωριστικά (Identifiers)

Τα αναγνωριστικά αποτελούνται από μια ακολουθία γραμμάτων, ψηφίων, την τελεία (‘.’) και την κάτω παύλα (‘_’). Δεν πρέπει να ξεκινάνε με ψηφίο, κάτω παύλα (‘_’) ή με την τελεία (‘.’) ακολουθούμενη από ψηφίο.

Ο ορισμός ενός γράμματος εξαρτάται από τις τρέχουσες τοπικές ρυθμίσεις: το ακριβές σύνολο των χαρακτήρων που επιτρέπονται δίνεται από την έκφραση (*isalnum(c)* || *c == '. '* || *c == '_'*) της γλώσσας C και θα περιλαμβάνει τονισμένα γράμματα πολλών Δυτικών Ευρωπαϊκών γλωσσών.

Παρατηρήστε ότι τα αναγνωριστικά που αρχίζουν με τελεία (‘.’) δεν είναι στην εξ' ορισμού λίστα της *ls* συνάρτηση και ότι το '…' και '..1', '..2', και τα λοιπά είναι ειδικά.

Παρατηρήστε επίσης ότι τα *αντικείμενα* μπορούν να έχουν ονόματα που δεν υπάρχουν στα αναγνωριστικά. Αυτά είναι προσβάσιμα σε γενικές γραμμές μέσω των *get* και *assign*, αν και μπορεί επίσης να αντιπροσωπεύονται από αλφαριθμητικά σε ορισμένες περιορισμένες περιπτώσεις, όταν δεν υπάρχει καμία ασάφεια (π.χ. "x" ← 1). Τα *get* και *assign* δεν περιορίζονται στα ονόματα που έχουν τα αναγνωριστικά, αυτά δεν αναγνωρίζουν τελεστές δεικτών ή λειτουργίες αντικατάστασης. Τα ακόλουθα ζεύγη δεν είναι ισοδύναμα:

x$a ← 1	*assign ("x$a", 1)*
x[[1]]	*get ("x[[1]]")*
names (x) ← nm	*assign ("names(x)", nm)*

3.1.2.3 Δεσμευμένες λέξεις (Reserved words)

Τα αναγνωριστικά που ακολουθούν έχουν ένα ιδιαίτερο νόημα και δεν μπορούν να χρησιμοποιηθούν σαν ονόματα αντικειμένων.

if else repeat while function for in next break

TRUE FALSE NULL Inf NaN

*NA NA_integer_ NA_real_ NA_complex_ NA_character_*

... ..1 ..2 και τα λοιπά

3.1.2.4 Ειδικοί τελεστές (Special operators)

Η R επιτρέπει στους χρήστες της να ορίζουν έμφυτους (infix) τελεστές. Αυτά έχουν την μορφή ενός αλφαριθμητικού από χαρακτήρες οριοθετούμενους από τον χαρακτήρα '%'. Το αλφαριθμητικό μπορεί να περιέχει οποιοδήποτε εκτυπώσιμο χαρακτήρα εκτός από το '%'.Οι ακολουθίες διαφυγής για τα αλφαριθμητικά δεν ισχύουν εδώ.

Σημειώστε ότι οι παρακάτω τελεστές είναι προκαθορισμένοι

%% %% %/% %in% %o% %x%*

3.1.2.5 Διαχωριστικά (Separators)

Αν και δεν υπάρχουν αυστηρά ορισμένα σύμβολα, τα τμήματα των **whitespace** χαρακτήρων (κενά, tabs και form feeds, στα Windows και στο UTF-8 υπάρχουν άλλα Unicode whitespace χαρακτήρες) χρησιμεύουν για την οριοθέτηση των συμβόλων σε περίπτωση ασάφειας, (π.χ. συγκρίνετε το x←5 και το x ← 5).

Οι **νέες γραμμές (newlines)** έχουν μια λειτουργία η οποία είναι ένας συνδυασμός του διαχωρισμού των συμβόλων και του τερματισμού μιας έκφρασης (expression). Αν μια έκφραση (expression) μπορεί να τερματιστεί στο τέλος μιας γραμμής ο parser θα αναλάβει να το κάνει αυτό, αλλιώς την νέα γραμμή (newline) την αντιμετωπίζει σαν *whitespace*. Τα ερωτηματικά (‘;’) μπορούν να χρησιμοποιηθούν για τον διαχωρισμό των στοιχειωδών εκφράσεων (expressions) μέσα στην ίδια γραμμή.

Ειδικοί κανόνες ισχύουν για την δεσμευμένη λέξη **else**: μέσα σε μια compound έκφραση, μια νέα γραμμή (newline) πριν το **else** είναι απορριπτέα, ενώ στο ακραίο επίπεδο, η νέα γραμμή (newline) τερματίζει την **κατασκευή του if** και το else που ακολουθεί προκαλεί συντακτικό λάθος. Αυτή η κάπως ανώμαλη συμπεριφορά παρουσιάζεται επειδή η R πρεπει να μπορεί να χρησιμοποιηθεί σε μια διαλογική λειτουργία (interactive mode) και στη συνέχεια θα πρεπει

να αποφασιστεί αν η έκφραση εισόδου είναι πλήρες, ημιτελής ή άκυρη μόλις ο χρήστης πατήσει RET.

Το **κόμμα (‘,’)** χρησιμοποιείτε για να χωρίσει τα ορίσματα μιας συνάρτησης και τους πολλαπλούς δείκτες.

3.1.2.6 Τελεστές συμβόλων (Operator token)

Η R χρησιμοποιεί τους παρακάτω τελεστές συμβόλων:

+ - * / %% ^	αριθμητική
> >= < <= == !=	σχεσιακή
! & |	λογική
~	μοντέλο της formulae (Chambers & Hastie, 1992)
→ ←	ανάθεση
$	δεικτοδότηση λίστας
:	ακολουθία

3.1.2.7 Ομαδοποίηση (Grouping)

Οι **απλές παρενθέσεις ‘(’ και ‘)’** χρησιμοποιούνται ρητά για την ομαδοποίηση μέσα σε εκφράσεις (expressions) και για να οριοθετήσουν τη λίστα παραμέτρων για μια συνάρτηση και για την κλήση μιας συνάρτησης.

Τα **άγκιστρα ‘{’ και ‘}’** οριοθετούν τμήματα (blocks) από εκφράσεις (expressions) για τους ορισμούς των συναρτήσεων, τις εκφράσεις (expressions) συνθηκών και τις επαναληπτικές δομές.

3.1.2.8 Διευθυνσιοδότηση συμβόλων (Indexing tokens)

Η δεικτοδότηση των πινάκων και των διανυσμάτων γίνεται χρησιμοποιώντας **μονές ή διπλές αγκύλες ‘[]’ και ‘[[]]’**. Επίσης, δεικτοδότηση με ετικέτα μπορεί να γίνει χρησιμοποιώντας τον τελεστή **‘$’**.

3.1.3 Εκφράσεις (Expressions)

Ένα πρόγραμμα της R αποτελείται από μια ακολουθία από εκφράσεις της R. Μια έκφραση μπορεί να είναι μια απλή έκφραση αποτελούμενη από μια σταθερά ή ένα αναγνωριστικό, ή μπορεί να είναι μια compound έκφραση κατασκευασμένη από αλλα μέρη (τα οποία μπορεί να είναι και τα ίδια εκφράσεις).

3.1.3.1 Κλήση συνάρτησης (Function calls)

Μια κλήση συνάρτησης παίρνει τη μορφή της αναφοράς που ακολουθείτε από μια λίστα ορισμάτων χωρισμένα με κόμματα μεταξύ τους μέσα σε ένα σύνολο από παρενθέσεις.

αναφορά_συνάρτησης (όρισμα_1, όρισμα _2, ... , όρισμα_n)

Η **αναφορά συνάρτησης** μπορεί να είναι είτε

- ένα **αναγνωριστικό** (το όνομα της συνάρτησης)
- ένα **αλφαριθμητικό** (το όνομα της συνάρτησης, αρκεί η συνάρτηση να έχει ένα όνομα το οποίο δεν θα είναι ένα έγκυρο αναγνωριστικό)
- μια **έκφραση** (η οποία θα πρεπει να έχει αξιολογηθεί (evaluate) σε ένα αντικείμενο συνάρτησης)

Κάθε **όρισμα** μπορεί να έχει μια **ετικέτα** (tag=expr), ή απλά μια απλή έκφραση. Μπορεί επίσης να είναι και κενή ή μπορεί να είναι και κάποιο ειδικό σύμβολο '…', '..2' και τα λοιπά. Μια **ετικέτα** μπορεί να είναι ένα αναγνωριστικό ή ένα αλφαριθμητικό.

Παραδείγματα:

f (x)

g (tag = value,, 5)

"odd name"("strange tag" = 5, y)

(function (x) x^2)(5)

3.1.3.2 Infix και prefix τελεστές (Infix and prefix operators)

Η σειρά προτεραιότητας (υψηλότερη πρώτα) των τελεστών είναι:

::

$ @

^

- + μοναδιαίο (unary)

:

%xyz%

* /

+ - δυαδικό (binary)

> >= < <= == !=

!

& &&

| ||

~ μοναδιαίο και δυαδικό

→ ->>

= ως ανάθεση

← <<-

Ο εκθετικός τελεστής '^' και η αριστερή ανάθεση μαζί με την πρόσθεση, την αφαίρεση και τους τελεστές '← - = <<-' **ομαδοποιούνται από τα δεξιά στα αριστερά**, όλοι οι άλλοι τελεστές **ομαδοποιούνται από τα αριστερά στα δεξιά**. Έτσι, το 2 ^ 2 ^ 3 είναι 2 ^ 8 και όχι 4 ^ 3.

Παρατηρήστε ότι οι τελεστές %% και %/% για ακαριαίο υπόλοιπο και διαίρεση έχουν υψηλότερη προτεραιότητα από τον πολλαπλασιασμό και την διαίρεση.

Αν και δεν είναι αυστηρά ένας **τελεστής**, πρεπει να σημειωθεί ότι το **'='** χρησιμοποιείτε για προθήκη ετικετών στα ορίσματα όταν καλούμε μια συνάρτηση και για την ανάθεση προκαθορισμένων τιμών μέσα στον ορισμό μιας συνάρτησης.

Το **σύμβολο $** είναι κατά κάποιον τρόπο ένα είδος τελεστή, αλλα δεν επιτρέπει αυθαιρεσία από την δεξιά του μεριά και εξετάζει τον υπό κατασκευή δείκτη. Είχε την υψηλότερη προτεραιότητα από οποιαδήποτε άλλο τελεστή.

Στον parser υπάρχει μια ισοδυναμία ανάμεσα στους **μοναδιαίους και στους δυαδικούς τελεστές** με την κλήση μιας συνάρτησης, όπου ο τελεστής αντίστοιχη στο όνομα της συνάρτησης και οι τελεστέοι στα ορίσματα της συνάρτησης.

Οι **παρενθέσεις** καταγράφονται ως ισοδύναμοι με έναν μοναδιαίο τελεστή, με το όνομα '('.

Παρατηρήστε ότι τα σύμβολα ανάθεσης είναι και αυτά τελεστές ακριβώς όπως είναι οι αριθμητικοί, οι σχεσιακοί και οι λογικοί. Κάθε έκφραση επιτρέπεται στην στοχευμένη πλευρά μιας ανάθεσης, όσον αφορά τον parser (2 + 2 ← 5 είναι μια έγκυρη έκφραση όσον αφορά τον parser, ο αξιολογητής (evaluator) όμως θα την απορρίψει). Παρόμοια σχόλια ισχύουν για το τον τελεστή μοντέλο φόρμουλας ('~').

3.1.3.3 Κατασκευές δεικτών (Index constructions)

Η R έχει τρεις (3) δομές δεικτοδότησης, δύο (2) εκ των οποίων είναι συντακτικά παρόμοιες αν και με κάπως διαφορετική σημασιολογία:

αντικείμενο [όρισμα_1, ... , όρισμα_n]

αντικείμενο [[όρισμα_1, ... , όρισμα_n]]

Το **αντικείμενο** μπορεί τυπικά να είναι οποιαδήποτε έγκυρη έκφραση, αλλα εννοείτε ότι πρεπει να δείχνει ή να αξιόλογη (evaluate) σε ένα υποσύνολο. Τα **ορίσματα** γενικά μπορεί να είναι αριθμητικά ή δείκτες χαρακτήρων, αλλα είναι πιθανά και αλλα είδη ορισμάτων (π.χ. λογικά).

Σε εσωτερικό επίπεδο, αυτές οι κατασκευές δεικτών αποθηκεύονται ως κλήσεις συναρτήσεων με ονόματα '[' και ' [[' αντίστοιχα.

Η τρίτη δομή δείκτη είναι:

αντικείμενο $ ετικέτα

Εδώ, το **αντικείμενο** συμπεριφέρεται όπως αναφέραμε πιο πάνω, ενώ η **ετικέτα** είναι ένα αναγνωριστικό ή ένα αλφαριθμητικό. Σε εσωτερικό επίπεδο, αποθηκεύεται ως κλήση συνάρτησης με το όνομα '$'.

3.1.3.4 Compound εκφράσεις (Compound expressions)

Μια compound έκφραση είναι της μορφής:

{ έκφραση_1 ; έκφραση_2 ; …… ; έκφραση_n }

Τα ερωτηματικά μπορούν να αντικατασταθούν από νέες γραμμές (newlines). Σε εσωτερικό επίπεδο, αποθηκεύεται ως κλήση της συνάρτησης με την '{' σαν όνομα συνάρτησης και τις εκφράσεις σαν ορίσματα.

3.1.3.5 Στοιχεία ελέγχου ροής (Flow control elements)

Η R περιέχει τις ακόλουθες δομές ελέγχου σαν ειδικές συντακτικές δομές:

if (συνθήκη) έκφραση

if (συνθήκη) έκφραση_1 else έκφραση_2

while (συνθήκη) έκφραση

repeat έκφραση

for (μεταβλητή in λίστα) έκφραση

Οι **εκφράσεις** σε αυτές τις δομές θα είναι τυπικά εκφράσεις τύπου compound.

Εντός των βρόγχων επανάληψης (while, repeat, for), μπορεί κανείς να χρησιμοποιήσει το **break** (τερματίζει τον βρόγχο) και το **next** (μεταπηδά στην επόμενη επανάληψη) .

Σε εσωτερικό επίπεδο, οι κατασκευές (των δομών ελέγχου) αποθηκεύονται σαν τις κλήσεις των συναρτήσεων:

"if" (συνθήκη, έκφραση)

"if" (συνθήκη, έκφραση_1, έκφραση_2)

"while" (συνθήκη, έκφραση)

"repeat" (έκφραση)

"for" (μεταβλητή, λίστα, έκφραση)

"break" ( )

"next" ( )

3.1.3.6 Ορισμός συνάρτησης (Function definitions)

Ο ορισμός μιας συνάρτησης έχει την μορφή:

function (λίστα_παραμέτρων) σώμα_συνάρτησης

Το **σώμα της συνάρτησης** είναι μια έκφραση, συχνά είναι μια έκφραση τύπου compound. Η **λίστα παραμέτρων** είναι μια λίστα χωρισμένη με κόμματα οπού κάθε αντικείμενο της μπορεί να είναι ένα αναγνωριστικό, ή της μορφής '*αναγνωριστικό = default*', ή ένα ειδικό σύμβολο '…'.Το *default* μπορεί αν είναι οποιαδήποτε έγκυρη έκφραση.

Παρατηρήστε ότι τα ορίσματα μιας συνάρτησης σε αντίθεση με τις ετικέτες κτλ. δεν μπορούν να έχουν "περίεργα όνομα" δοσμένα ως αλφαριθμητικά.

Σε εσωτερικό επίπεδο, ο ορισμός της συνάρτησης αποθηκεύεται ως κλήση συνάρτησης με όνομα συνάρτησης το **function** και 2 ορίσματα, την λίστα_παραμέτρων και σώμα_συνάρτησης. Η λίστα_παραμέτρων αποθηκεύεται σε διπλέτες ετικέτα – τιμή οπού η ετικέτα είναι το όνομα της κάθε παραμέτρου και η τιμή είναι η εξ' ορισμού έκφραση της.

3.2 Ο λεκτικός αναλυτής της R

Η λεκτική ανάλυση (lexical analysis) είναι η διαδικασία μετατροπής μιας ακολουθίας χαρακτήρων σε μια σειρά από σύμβολα. Ακολουθούν οι κανόνες των Εικόνων 3-1 ως 3-6 που βασίζονται στην έκδοση 3.3.2 της R που περιγράφονται στην προηγούμενη ενότητα (την 3.1).

```
HEX :  HEXADEMICAL | HEX_FLOATING_CONSTANTS ;

fragment
HEXADEMICAL:   '0' ('x'|'X') HEXDIGIT+ [Ll]? ;

// Hexadecimal floating point constants are supported using C99 syntax, e.g. '0x1.1p1'.
// The p separates the base number from the exponent.
// I add this for PR#15753 and PR#15976.
fragment
HEX_FLOATING_CONSTANTS :   '0' ('x'|'X') HEXDIGIT+ ('p'|'P') '-'? HEXDIGIT+ [Ll]? ;
```

Εικόνα 3-1: Ο κανόνας για τους δεκαεξαδικούς (HEX) αριθμούς

```
FLOAT:  DIGIT+ '.' DIGIT* EXP? [Ll]?
     |  DIGIT+ EXP? [Ll]?
     |  '.' DIGIT+ EXP? [Ll]?
     ;

fragment
DIGIT:  '0'..'9' ;

fragment
EXP :   ('E' | 'e') ('+' | '-')? INT ;
```

Εικόνα 3-2: Ο κανόνας για τις αριθμητικές (FLOAT) σταθερές

```
COMPLEX
     :   INT 'i'
     |   FLOAT 'i'
     ;
```

Εικόνα 3-3: Ο κανόνας για τις μιγαδικές (COMPLEX) σταθερές

```
STRING
     :   '"' ( ESC | ~[\\"] )*? '"'
     |   '\'' ( ESC | ~[\\'] )*? '\''
     |   '`' ( ESC | ~[\\`] )*? '`'
     |   '`\\``' // ## PR#15621 backticks could not be escaped
     ;

fragment
ESC :   '\\' [abtnfrvx"'\\]  // I add x for "fa\xE7ile".
     |   UNICODE_ESCAPE
     |   HEX_ESCAPE
     |   OCTAL_ESCAPE
     ;

fragment
UNICODE_ESCAPE
     :   '\\' 'u' HEXDIGIT HEXDIGIT HEXDIGIT HEXDIGIT
     |   '\\' 'u' '{' HEXDIGIT HEXDIGIT HEXDIGIT HEXDIGIT '}'
     ;

fragment
OCTAL_ESCAPE
     :   '\\' [0-3] [0-7] [0-7]
     |   '\\' [0-7] [0-7]
     |   '\\' [0-7]
     ;

fragment
HEX_ESCAPE
     :   '\\' HEXDIGIT HEXDIGIT?
     ;
```

Εικόνα 3-4: Ο κανόνας για τις αλφαριθμητικές (STRING) σταθερές

```
// https://cran.r-project.org/doc/manuals/R-lang.html#Identifiers
ID  :   '.' // i add this for "reg-tests-1c -> function(.)"
    |   '.' (LETTER|'_'|'.') (LETTER|DIGIT|'_'|'.')*    // gel also special ... and ..DIGIT
    |   LETTER (LETTER|DIGIT|'_'|'.')*
    ;

fragment LETTER  : [a-zA-Z] ;
```

Εικόνα 3-5: Ο κανόνας για τα αναγνωριστικά (ID)

```
USER_OP :   '%' .*? '%' ;   // Used for special operators (as matrix product, outer product, kronecker product,  mattching etc).

COMMENT :   '#' ~[\r\n]* -> channel(HIDDEN); // I use this non-gready method : http://stackoverflow.com/questions/23976617/parsing-single-line-comments

// Match both UNIX and Windows newlines
NL      :   '\r'? '\n' ;

WS      :   [ \t\u000C]+ -> skip ; // The (-> skip) completely drops the token.
```

Εικόνα 3-6: Ο κανόνας για τις μιγαδικές (COMPLEX) σταθερές

3.3 Ο συντακτικός αναλυτής της R

Ακολουθεί ο κώδικας του συντακτικού αναλυτή (parser analysis) της R ο οποίος είναι φτιαγμένος με βάση την περιγραφή της ενότητας 3.1. Στην Εικόνα 3-7 φαίνεται ο κανόνας ενός προγράμματος της R το οποίο αποτελείτε από μια έκφραση (expr) ακολουθούμενη από ερωτηματικό ή νέα γραμμή (NL), ή μόνο απο μια νέα γραμμή (NL), καμία ή περισσότερες φορές ακολουθούμενη από το τέλος τους αρχείου (EOF).

```
prog:   (   expressions+=expr (';'|NL) |   NL )* EOF
    ;
```

Εικόνα 3-7: Κανόνας προγράμματος της R

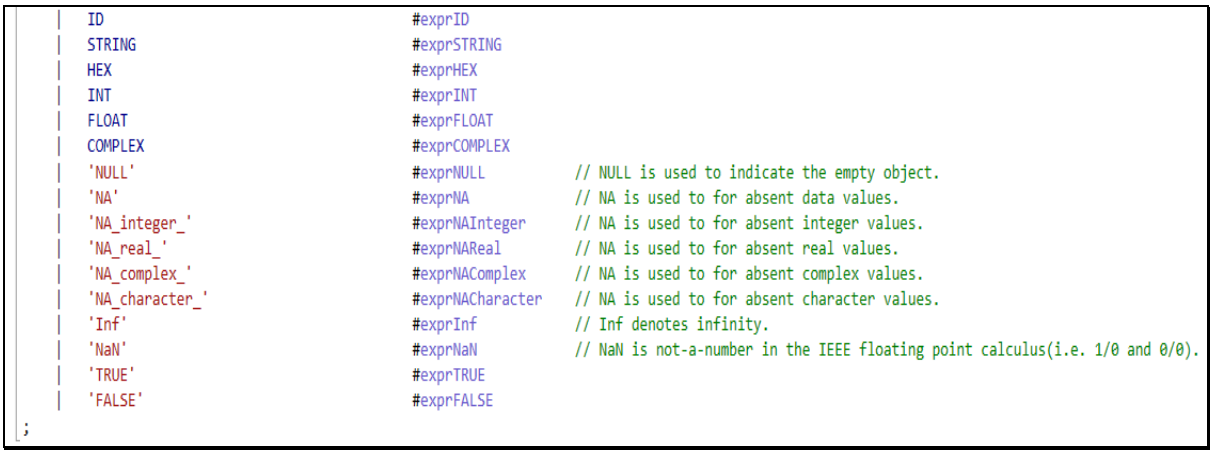
Στις Εικόνες 3-8 ως 3-10 φαίνεται ο κανόνας της έκφρασης (expr) που έχει να κάνει με τις εκφράσεις που περιγράφονται στην Ενότητα 3.1.3.

```
expr:   arraybase=expr '[[' arrayoffset=sublist ']' ']'          #exprIndexingByVectors      // '[[' follows R's yacc grammar
    |   arraybase=expr '['  arrayoffset=sublist ']'              #exprIndexingBasic
    |   left=expr op=('::'|':::') right=expr                     #exprSingleDoubleColonsOperators
    |   left=expr op=('$'|'@') right=expr                        #exprDollarAtOperators      // Indexing tagged lists may be done using  $.
    |   <assoc=right> left=expr '^' right=expr                   #exprExponentiationBinary
    |   op=('-'|'+') expr                                        #exprMinusOrPlusUnary
    |   left=expr ':' right=expr                                 #exprColonOperator          //It creates the series of numbers in sequence for a vector.
    |   left=expr USER_OP right=expr                             #exprWrappedWithPercent     // anything wrapped in %: '%' .* '%'
    |   left=expr op=('*'|'/') right=expr                        #exprMultiplicationOrDivisionBinary
    |   left=expr op=('+'|'-') right=expr                        #exprMinusOrPlusBinary
    |   left=expr op=('>'|'>='|'<'|'<='|'=='|'!=') right=expr    #exprComparisons
    |   '!' expr                                                 #exprNotUnary
    |   left=expr op=('&'|'&&') right=expr                       #exprAndBinary          // Vectorized Or Not Vectorized.
    |   left=expr op=('|'|'||') right=expr                       #exprOrBinary           // Vectorized Or Not Vectorized.
    |   '~' expr                                                 #exprTildeUnary         //Used For Model Formulae.
    |   left=expr '~' right=expr                                 #exprTildeBinary        //Used For Model Formulae.
    |   left=expr op=('<-'|'<<-'|'='|'->'|'->>'|':=') right=expr #exprAssignmentOpetators // Using leftward or rightward operators.
```

Εικόνα 3-8: Κανόνας για τις εκφράσεις (1/3)

```
    |   'function' '(' funargs=formlist? ')' functionbody=expr     #exprDefineFunction     // define function. function ( arglist ) body
    |   funid=expr '(' funargs=sublist ')'                        #exprCallFunction       // call function
    |   '{' exprlist '}'                                          #exprCompound           // compound statement
    |   'if' '(' ifcond=expr ')' thenbody=expr                    #exprIfStatement
    |   'if' '(' ifcond=expr ')' thenbody=expr 'else' elsebody=expr #exprIfElseStatement
    |   'for' '(' ID 'in' for_list=expr ')' for_body=expr         #exprFor
    |   'while' '(' whilecond=expr ')' whilebody=expr             #exprWhile          // Repeat until statement1 evaluates to FALSE.
    |   'repeat' expr                                             #exprRepeat         // Repeat statement until find break.
    |   '?' expr                                                  #exprHelp           // Produces the function documentation.
    |   left=expr '?' right=expr                                  #exprHelpForMethods // Looks for the overall methods documentation.
                                                                                      //I add this exist in gram.y have lowest priority.
    |   'next'                                                    #exprNextStatement
    |   'break'                                                   #exprBreakStatement
    |   '(' expr ')'                                              #exprParenthesis
```

Εικόνα 3-9: Κανόνας για τις εκφράσεις (2/3)



```
    |   ID                        #exprID
    |   STRING                    #exprSTRING
    |   HEX                       #exprHEX
    |   INT                       #exprINT
    |   FLOAT                     #exprFLOAT
    |   COMPLEX                   #exprCOMPLEX
    |   'NULL'                    #exprNULL           // NULL is used to indicate the empty object.
    |   'NA'                      #exprNA             // NA is used to for absent data values.
    |   'NA_integer_'             #exprNAInteger      // NA is used to for absent integer values.
    |   'NA_real_'                #exprNAReal         // NA is used to for absent real values.
    |   'NA_complex_'             #exprNAComplex      // NA is used to for absent complex values.
    |   'NA_character_'           #exprNACharacter    // NA is used to for absent character values.
    |   'Inf'                     #exprInf            // Inf denotes infinity.
    |   'NaN'                     #exprNaN            // NaN is not-a-number in the IEEE floating point calculus(i.e. 1/0 and 0/0).
    |   'TRUE'                    #exprTRUE
    |   'FALSE'                   #exprFALSE
;
```

Εικόνα 3-10: Κανόνας για τις εκφράσεις (3/3)

Στην Εικόνα 3-11 έχουμε τον κανόνα της λίστας από εκφράσεις (exprlist) που τον βρίσκουμε μέσα σε compound εκφράσεις. Ο κανόνας της λίστας από εκφράσεις (exprlist) αποτελείτε από μια έκφραση (expr), ακολουθούμενη από ερωτηματικό ή νέα γραμμή (NL) ακολουθούμενη από έκφραση (expr) εάν υπάρχει και όλο αυτό καμία ή περισσότερες φορές, είτε από τίποτα.

```
exprlist
    :   expr ((';'|NL) expr?)*
    |
    ;
```

Εικόνα 3-11: Κανόνας λίστας από εκφράσεις

Στην Εικόνα 3-12 έχουμε τους κανόνες που βρίσκουμε μέσα στο όρισμα μιας συνάρτησης. Ο πρώτος αναφέρεται σε μια λίστα από ορίσματα (formlist) και αναφέρεται σε ένα όρισμα (form), ακολουθούμενο από κόμμα και όρισμα (form), καμία ή περισσότερες φορές. Ο δεύτερος κανόνας αναφέρεται στο όρισμα (form) το οποίο είτε είναι ένα αναγνωριστικό (ID) είτε ένα αναγνωριστικό (ID) ακολουθούμενο από ίσον και μια έκφραση (expr).

```
formlist : form (',' form)* ;

form:   ID
    |   ID '=' expr
    ;
```

Εικόνα 3-12: Οι κανόνες formlist και form

Στην Εικόνα 3-13 έχουμε τους κανόνες που βρίσκουμε είτε μέσα σε μια λίστα από τιμές ενός πίνακα είτε μέσα στην κλήση μιας συνάρτησης. Ο πρώτος αναφέρεται σε μια λίστα από παραμέτρους (sublist) και αναφέρεται σε μια παράμετρο (sub), ακολουθούμενο από κόμμα και παράμετρο (sub), καμία ή περισσότερες φορές. Ο δεύτερος κανόνας αναφέρεται στην παράμετρο (sub) η οποία μπορεί να είναι μια έκφραση (expr) ή ένα αναγνωριστικό (ID) ακολουθούμενο από ίσον ή ένα αλφαριθμητικό (STRING) ακολουθούμενο από ίσον ή το NULL ακολουθούμενο από ίσον ή τίποτα.

```
sublist : sub (',' sub)*
    ;

sub :   expr
    |   ID '='
    |   STRING '='
    |   'NULL' '='
    |
    ;
```

Εικόνα 3-13: Οι κανόνες sublist και sub

4 Μηχανισμοί διαχείρισης της αναπαράστασης

Σε γενικές γραμμές ο σχεδιασμός είναι ανεξάρτητος από την γλώσσα προγραμματισμού και μπορεί να προσαρμοστεί εύκολα στις εκάτοστε ανάγκες λόγω της εκτεταμένης χρήση του πολυμορφισμού (polymorphism) και της κληρονομικότητας (inheritance) καθώς και διαφόρων μοτίβων σχεδίασης.

4.1 Μοτίβα που χρησιμοποιήθηκαν

Οπότε για να μπορέσει το Front End της R να είναι προσαρμόσιμο, ευέλικτο αλλά και επεκτάσιμο χρησιμοποιώ κάποια από τα παρακάτω μοτίβα:

- Composite
- Visitor
- Observer
- Listener
- Iterator
- Façade

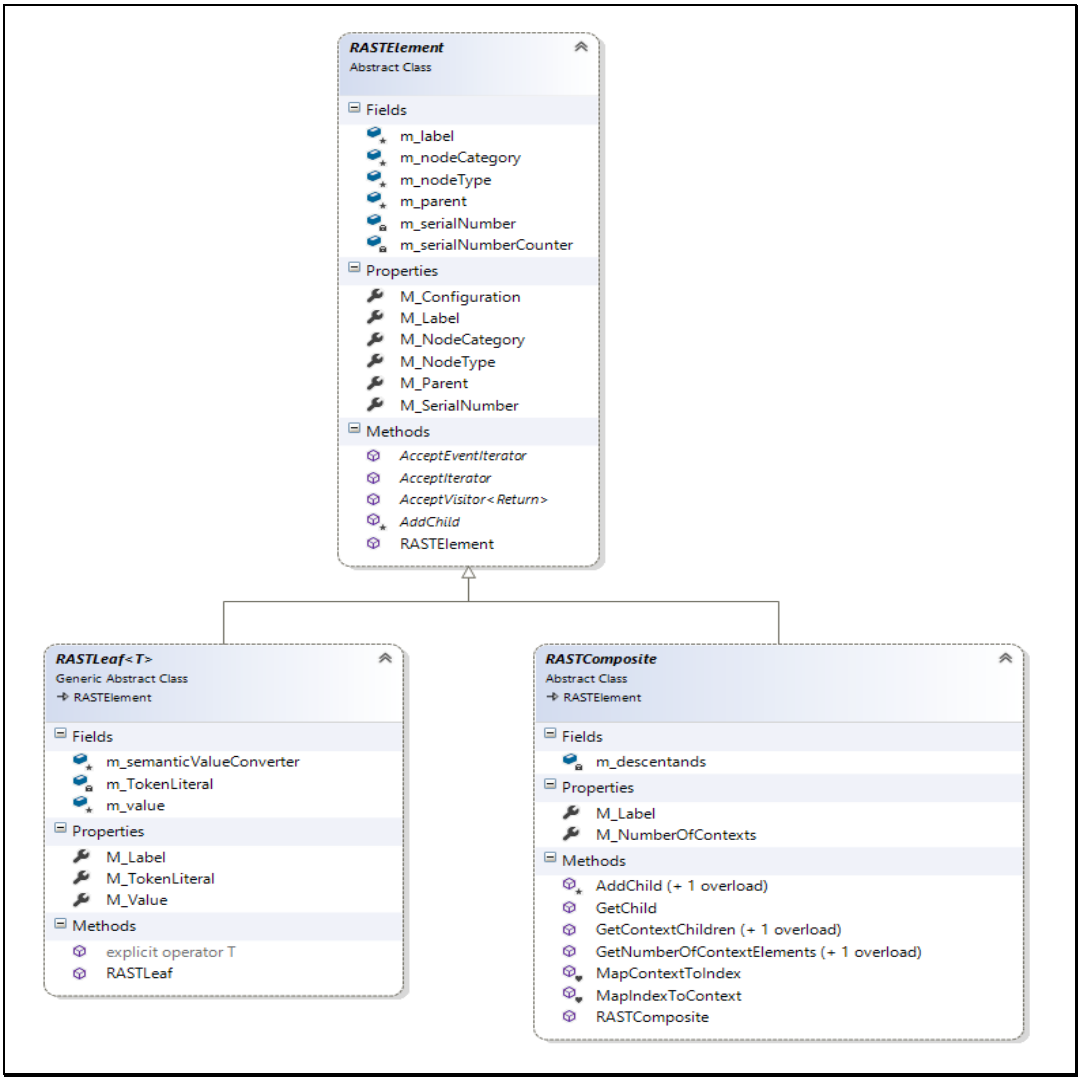
Στις παρακάτω υποενότητες θα αναλυθούν διεξοδικά, αναφορικά με τις δυνατότητες τους και την χρήση τους.

4.1.1 Μοτίβο σύνθεσης (Composite pattern)

Το μοτίβο του συνθέτη (compound pattern) (Gamma, et al., 1994) περιγράφει μια ομάδα από αντικείμενα τα οποία πρεπει να αντιμετωπίζονται με τον ίδιο τρόπο ως ένα ενιαίο αντικείμενο. Ο σκοπός του είναι να **συνθέτη** αντικείμενα σε ιεραρχική δενδροειδή μορφή. Υλοποιώντας αυτό το μοτίβο μπορούμε να διαχειριστούμε απλά αντικείμενα αλλα και σύνθετα ομοιόμορφα.

Κάθε φορά που θέλουμε να πραγματοποιήσουμε κάποιο πέρασμα στους κόμβους του AST πρεπει να δημιουργήσουμε κάποιες βασικές κλάσεις οι οποίες χρησιμοποιούν το μοτίβο που περιγράφουμε για τη διαχείριση της πληροφορίας (της αναπαράστασης). Ποιο συγκεκριμένα

για την δεντρική αναπαράσταση έχουν κατασκευαστεί τρεις (3) κλάσεις οι οποίες προσδιορίζουν τη δομή του δένδρου και είναι υπεύθυνες για την αποθήκευση της πληροφορίας και φαίνονται στην Εικόνα 4-1.



Εικόνα 4-1: Βασικές κλάσεις του AST που εφαρμόζουν το μοτίβο συνθέτης (composite)

RASTElement class: αναπαριστά έναν κομβο του AST δέντρου και έχει κάποια βοηθητικά πεδία και μεθόδους. Εικόνα 4-2 ant

RASTLeaf class: αναπαριστά τα φύλλα του AST που συνήθως είναι τερματικά σύμβολα της γραμματικής.

RASTComposite class: αναπαριστά έναν σύνθετο κομβο του AST ο οποίος μπορεί να αποτελείτε και από άλλους κόμβους.

4.1.2 Μοτίβο επισκέπτη (Visitor pattern)

Το μοτίβο επισκέπτη (visitor pattern) (Gamma, et al., 1994) είναι ένας τρόπος να **διαχωρίζουμε τον αλγόριθμο από την δομή πάνω στην οποία επιδρά**. Η διάσχιση του δένδρου γίνεται καλώντας την αντίστοιχη μέθοδο του κόμβου που θέλουμε να προσπελάσουμε. Έτσι μια εικονική μέθοδος (virtual method) μπορεί να παρακαμφθεί (override) και να καλείτε από τον κώδικα του κάθε κόμβου ώστε πάντα να είμαστε σίγουροι ότι θα καλείτε η μέθοδος που αντιστοιχεί στον τύπο της τρέχουσας κλάσης. Τα μοτίβο αυτό το χρησιμοποιείτε για την προσπέλαση του AST αλλα και από το εργαλείο ANTLR.

Κάθε φορά που θέλουμε να πραγματοποιήσουμε κάποιο πέρασμα στους κόμβους του AST πρεπει να δημιουργήσουμε κάποιες βασικές κλάσεις οι οποίες χρησιμοποιούν το μοτίβο που περιγράφουμε. Ποιο συγκεκριμένα έχουμε τρεις (3) βασικές κλάσεις επισκέπτη που φαίνονται στην Εικόνα 4-2.

IASTAbstractConcreteVisitor interface: περιέχει τα prototypes όλων των επισκεπτών (visitors) μεθόδων για κάθε κομβο του AST.

RASTAbstractConcreteVisitor class: περιέχει τις εικονικές υλοποιήσεις των μεθόδων οι οποίες μπορούν να παρακαμφθούν για να χρησιμοποιηθούν με διαφορετικό τρόπο.

RASTAbstractVisitor class: αναφέρεται στην AST composite δομή με μια υψηλού επιπέδου κλάση τύπου T και μεθόδους Visit.

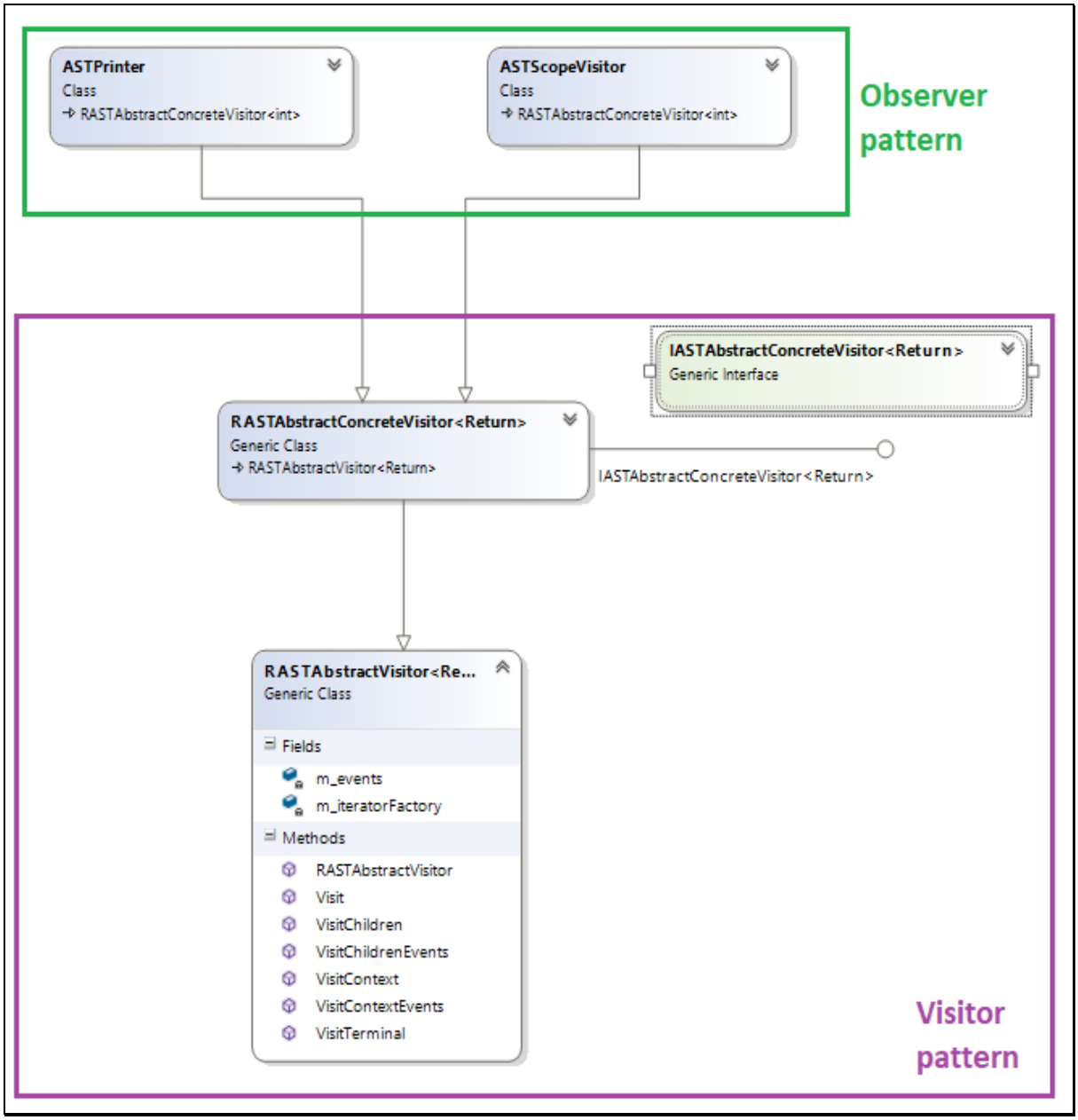
4.1.3 Μοτίβο παρατηρητή (Observer pattern)

Σε συνδυασμό με το μοτίβο του επισκέπτη, χρησιμοποιείτε το μοτίβο του παρατηρητή (observer pattern) (Gamma, et al., 1994). Ένα αντικείμενο διατηρεί μια λίστα από παρατηρητές και τους ενημερώνει αυτόματα με οποιαδήποτε αλλαγή κατάστασης συνήθως καλώντας μια από τις μεθόδους τους. Χρησιμοποιείτε κυρίως για την **κατασκευή συστημάτων διαχείρισης γεγονότων**.

Κάθε φορά που εισερχόμαστε ή εξερχόμαστε σε έναν κόμβο, ενεργοποιείτε ένα γεγονός. Έτσι, μπορούμε να χρησιμοποιήσουμε τα γεγονότα έτσι ώστε να πραγματοποιήσουμε οποιαδήποτε ενέργεια σε οποιαδήποτε σημείο του AST δένδρου και πιο συγκεκριμένα έχουμε δυο (2) κλάσεις παρατηρητή που φαίνονται στην Εικόνα 4-2.

ASTPrinter class: υλοποιεί τις εικονικές μεθόδους για κάθε κόμβο του AST μεσω override με σκοπό την δημιουργία ενός γράφου.

ASTScopeVisitor class: υλοποιεί τις εικονικές μεθόδους για κάθε κόμβο του AST μεσω override με σκοπό την δημιουργία του πίνακα συμβόλων.



Εικόνα 4-2: Κλάσεις περάσματος του AST που εφαρμόζουν το μοτίβο επισκέπτη (visitor) και παρατηρητή (observer)

4.1.4 Μοτίβο επαναλήπτη (Iterator pattern)

Σε συνδυασμό με το μοτίβο του παρατηρητή, χρησιμοποιούμε και το μοτίβο του επαναλήπτη (iterator pattern) (Gamma, et al., 1994). Ένας επαναλήπτης χρησιμοποιείτε για να διασχίσουμε το περιεχόμενο ενός στοιχείου.

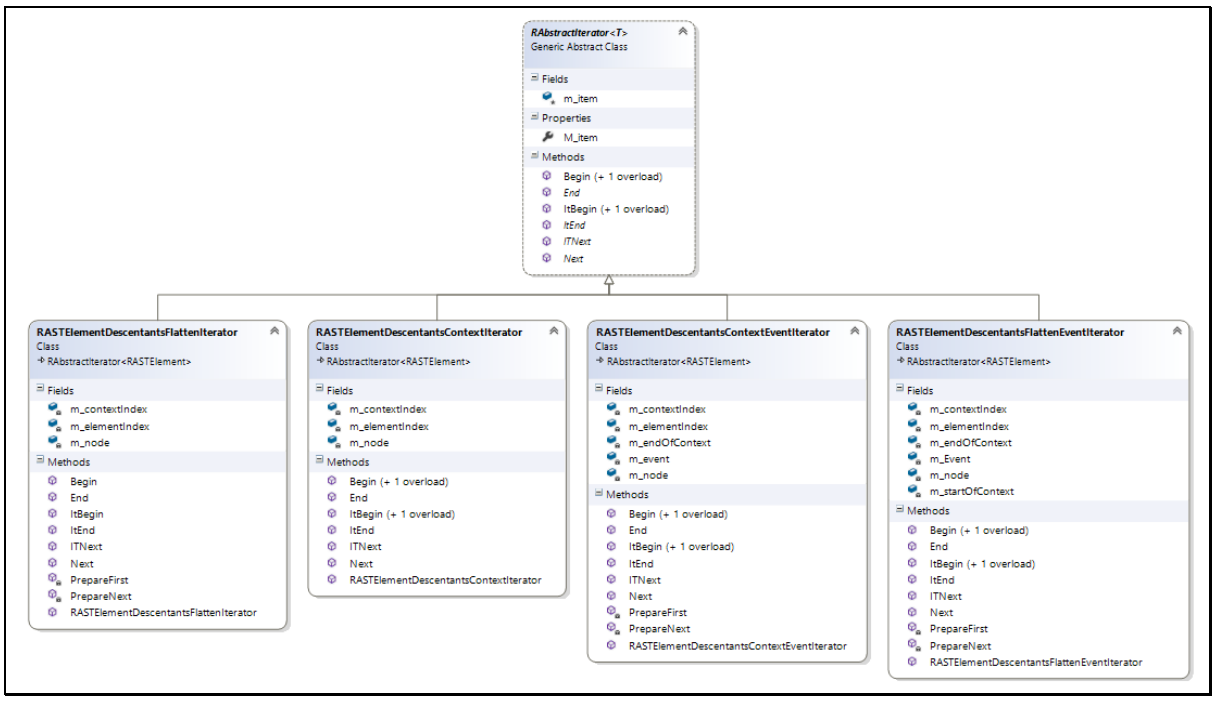
Υπάρχουν αρκετές κλάσεις ώστε ο κώδικας μας να είναι επεκτάσιμος. Πιο συγκεκριμένα έχουμε πέντε (5) κλάσεις παρατηρητή που φαίνονται στην Εικόνα 4-3.

RAbstractIterator interface: παρέχει μια γενική διεπαφή του μοτίβου του επαναλήπτη.

RASTElementDescentantsContextIterator class: αναφέρεται σε έναν επαναλήπτη ενός απογόνου, ενός συγκεκριμένου σύνθετου αντικειμένου που βρίσκετε σε μια συγκεκριμένη θέση.

RASTElementDescentantsContextEventIterator class: αναφέρεται στα γεγονότα που σχετίζονται με την διάσχιση της δομής του AST. Αυτή η κλάση εξειδικεύεται ανάλογα με τον τύπο του κόμβου.

RASTElementDescentantsFlattenEventIterator class: αναφέρεται στα γεγονότα που σχετίζονται με την διάσχιση της δομής του AST. Αυτή η κλάση εξειδικεύεται ανάλογα με τον τύπο του κόμβου.



Εικόνα 4-3: Κλάσεις διάσχισης περιεχομένου που εφαρμόζουν το μοτίβο επαναλήπτη (iterator)

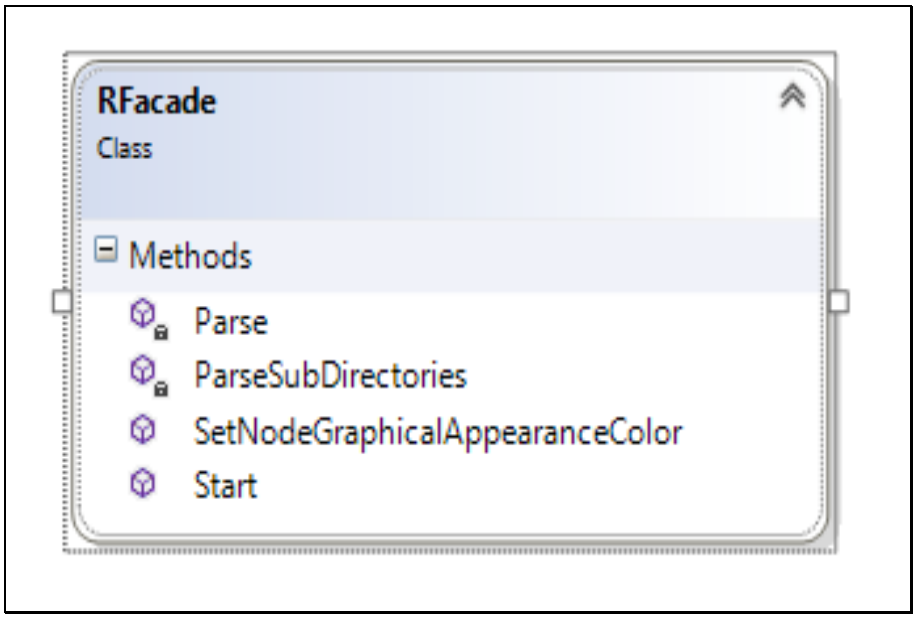
4.1.5 Μοτίβο πρόσοψης (Façade pattern)

Το μοτίβο πρόσοψης (façade pattern) (Gamma, et al., 1994) χρησιμοποιείτε όταν είναι επιθυμητή η παροχή μιας ενιαίας διεπαφής σε ένα σύνολο από διεπαφές ενός υποσυστήματος. Το συγκεκριμένο μοτίβο σχεδίασης ορίζει μια υψηλού επιπέδου διεπαφή

που κάνει το υποσύστημα ευκολότερο στην χρήση. Έτσι πετυχαίνουμε την χρήση πολλών κλάσεων που επικοινωνούν μεταξύ τους, ελάττωση της πολυπλοκότητας και να έχουμε μια απλή διεπαφή.

Εδώ χρησιμοποιούμαι μόνο μια (1) κλάση η οποία καλύπτει όλα τα υποσυστήματα μας και απεικονίζεται στην Εικόνα 4-4.

RFacade class: αυτή η κλάση χειρίζεται την εκτέλεση όλου του προγράμματος. Μεσω των μεθόδων της παίρνει την είσοδο ή τις εισόδους του χρήστη όπου και γίνεται το φιλτράρισμα, η λεκτική ανάλυση και η συντακτική ανάλυση και έπειτα ακολουθούν τα περάσμα και η δημιουργία των συντακτικών δένδρων καθώς και η ρύθμιση του εικαστικού των κόμβων.



Εικόνα 4-4: Η κύρια κλάση του συστήματος η οποία εφαρμόζει το μοτίβο πρόσοψης (façade)

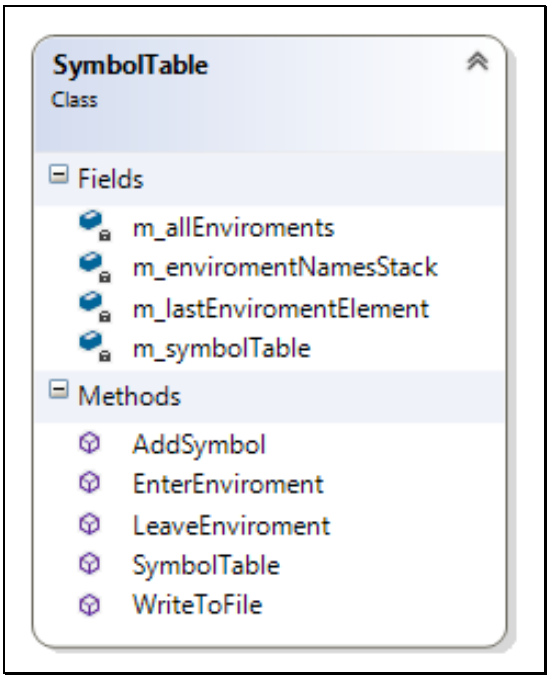
4.2 Πίνακας συμβόλων (Symbol Table)

Για την περαιτέρω ανάπτυξη του μεταγλωττιστή είναι απαραίτητο να γνωρίζουμε τα σύμβολα του προγράμματος έτσι ώστε να τα διαχειριστούμε αναλόγως. Για να αξιολογηθεί ένα σύμβολο χρησιμοποιούμε κανόνες οι οποίοι μας επιτρέπουν να συνδέσουμε μια τιμή με ένα σύμβολο. Κάθε γλώσσα προγραμματισμού έχει τέτοιους κανόνες. Στην R οι κανόνες αυτοί είναι αρκετά απλοί και λέγονται **lexical scope** (Peng, 2016). Αυτό σημαίνει ότι οι δεσμεύσεις μεταβλητών ισχύουν κατά την δημιουργία τους. Τα σύμβολα αυτά μπορεί να είναι bound ή unbound. Οριοθετημένα σύμβολα είναι αυτά που υπάρχουν είτε στα ορίσματα της συνάρτησης ή μέσα στο σώμα της. Οποιοδήποτε άλλο σύμβολο υπάρχει στο σώμα της

συνάρτησης είναι είτε local είτε unbound. Τοπικές μεταβλητές είναι αυτές που ορίζονται μέσα σε μια συνάρτηση. Κατά την διάρκεια της αξιολόγησης εάν βρεθεί ένα μη οριοθετημένο σύμβολο τότε η R προσπαθεί να εντοπίσει μια τιμή για αυτό. Η R ψάχνει πρώτα το environment της συνάρτησης και προχωράει προς τα επάνω μέχρι να φτάσει στο global περιβάλλον. Στην γλώσσα S υπάρχουν διαφορετικοί κανόνες scoping.

Με το **λεκτικό scoping** κοιτάμε εκεί που ορίζεται η συνάρτηση ενώ με το **δυναμικό scoping** κοιτάμε εκεί που καλείτε η συνάρτηση. Όταν μια συνάρτηση ορίζεται στο global environment και στην συνέχεια καλείτε από το global περιβάλλον, τότε το περιβάλλον ορισμού και το περιβάλλον κλήσης είναι τα ίδια. Αυτό μπορεί μερικές φορές να μας δώσει την αίσθηση του δυναμικού scoping. Εδώ χρησιμοποιούμαι μόνο μια (1) κλάση η οποία καλύπτει την δημιουργία και την αποθήκευση στην Εικόνα 4-5.

SymbolTable class: αυτή η κλάση διαχειρίζεται τα σύμβολα του προγράμματος της εσόδου είτε είναι τοπικά, unbound ή ορίσματα συνάρτησης και τα αποθηκεύει στον πίνακα συμβόλων.



Εικόνα 4-5: Η κλάση δημιουργίας του πίνακα συμβόλων (symbol table)

5 Κατασκευή του Front End της R

Για την κατασκευή του προγράμματος χρησιμοποίησα το Visual Studio 2015 και χρειάστηκε να δημιουργηθεί ένα solution με δυο (2) projects τα:

- RGrammar
- RFrontEnd

Στις παρακάτω υποενότητες θα αναλυθεί η χρήση τους καθώς και ο λόγος της δημιουργίας τους.

5.1 RGrammar project

Το project RGrammar διαχειρίζεται μόνο οτιδήποτε έχει να κάνει με την γραμματική και με το εργαλείο ANTL δηλαδή την γεννήτρια συντακτικών αναλυτών, έχει σαν .NET Framework το 4.5 για λόγους συμβατότητας με το πακέτο του ANTLR, στο οποίο έγινε αναφορά στην υποενότητα 2.3. Έπειτα, έγινε εγκατάσταση στο project του πακέτου του ANTLR μεσω του NuGet Package Manager (Tools → NuGet Package Manager → Package Manager Console) εισάγοντας την παρακάτω εντολη.

Install-Package Antlr4 -Version 4.5.3-rc1 –Pre

Έχοντας πλέον εγκατεστημένο το ANTLR, προχώρησα στην δημιουργία δυο (2) αντικείμενων, των R.g4 και RFilter.g4. Το RFilter.g4 είναι το φίλτρο της γραμματικής και το R.g4 είναι η γραμματική της R.

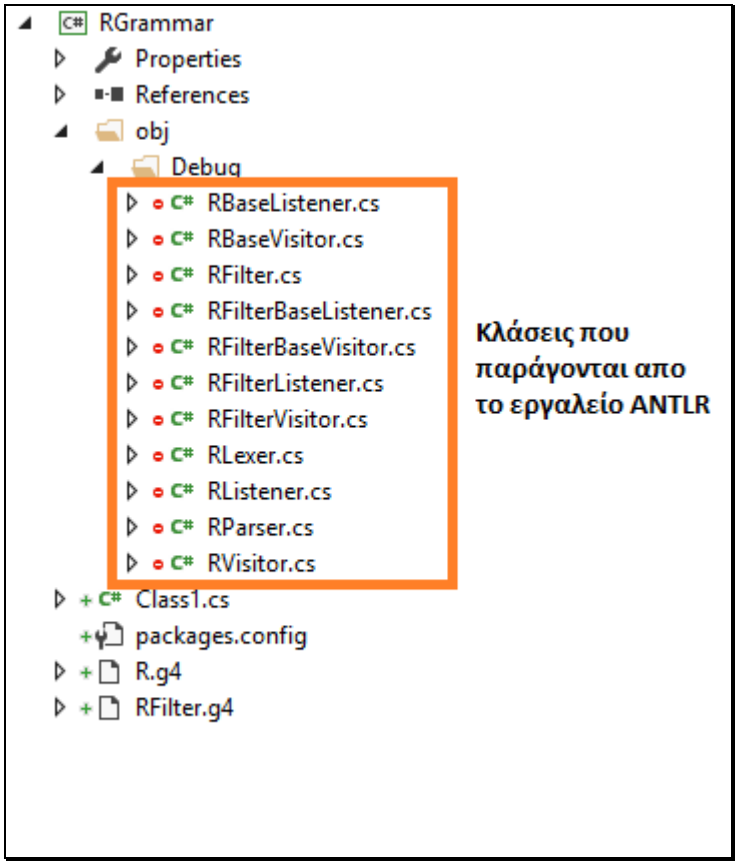
Το αντικείμενο **RFilter.g4** είναι ουσιαστικά ένας parser που φιλτράρει τα προγράμματα της R πριν αυτά κάνουν parse με την γραμματική της R, χρησιμοποιεί το λεξιλόγιο της R φαίνεται στα options και απεικονίζεται στην Εικόνα 5-1. Ο λόγος που γίνεται αυτό είναι οι νέες γραμμές (NL), τις οποίες η γραμματική σε αφήνει να τις εφαρμόσεις οπουδήποτε μέσα στον κώδικα ο χρήστης επιθυμεί και φαίνεται στον κανόνα elem**.** Έτσι, φτιάχνοντας ένα φίλτρο το οποίο πετάει τις αλλαγές γραμμών (NL) με τον μηχανισμό HiddenChannel που φαίνεται στον κανόνα eat καταφέρνουμε να μην χρειάζεται να τις εισάγουμε στην γραμματική μας με αποτέλεσμα να έχουμε μια μικρή και κατανοητή γραμματική. Μας

βοηθάει ακόμα στην περίπτωση του else κοιτώντας δυο (2) θέσεις πιο πριν από το τρέχον σύμβολο (_input.Lt(-2)) για να δει εάν υπάρχει νέα γραμμή και να την πετάξει, πάντα στην θέση -1 είναι το σύμβολο else και στην θέση -2 το σύμβολο που θέλουμε να εξετάσουμε, αξίζει να αναφέρουμε ότι μιλάμε για σύμβολα και όχι χαρακτήρες. Αυτός ο μηχανισμός φαίνεται στις σειρές 53 – 72 του κανόνα elem.

```
34  parser grammar RFilter;
35
36  options { tokenVocab=R; }
37
38  stream : (elem|NL|';')* EOF ;
39
40  eat :   (NL {((IWritableToken)$NL).Channel = TokenConstants.HiddenChannel;} )+ ;
41
42  elem:   op eat?
43      |   atom
44      |   '{' eat? (elem|NL|';')* '}'
45      |   '(' (elem|eat)* ')'
46      |   '[' (elem|eat)* ']'
47      |   '[[' (elem|eat)* ']' ']'
48      |   'function' eat? '(' (elem|eat)* ')' eat?
49      |   'for' eat? '(' (elem|eat)* ')' eat?
50      |   'while' eat? '(' (elem|eat)* ')' eat?
51      |   'if' eat? '(' (elem|eat)* ')' eat?
52      |   'else' eat? {
53
54          /* 10.4.5 Flow control elements (from R-lang manual)
55          R contains the following control structures as special syntactic constructs
56          if ( cond ) expr
57          if ( cond ) expr1 else expr2
58          -> The expressions in these constructs will typically be compound expressions.
59          */
60
61          /*
62          And the "input" attribute from the abstract class Parser (which your
63          generated parser extends from) now has an underscore ("_input") in front of it.
64          Lt(-2) : Predicates minus 2 positions/tokens from current.
65          */
66
67          IWritableToken tok = (IWritableToken) _input.Lt(-2);
68          if ( ((IToken)tok).Type.Equals(NL)) // (if (_input.Lt(-2) == NL) )
69              tok.Channel= TokenConstants.HiddenChannel;
70              // I set the token's Channel from Default to HiddenChannel.
71              // Anything on different channel than DEFAULT_CHANNEL is not parsed by parser.
72          }
73      ;
74
75  atom:   'next' | 'break' | ID | STRING | HEX | INT | FLOAT | COMPLEX | 'NULL'
76      |   'NA' | 'NA_integer_' | 'NA_real_' | 'NA_complex_' |'NA_character_'| 'Inf'
77      | 'NaN' | 'TRUE' | 'FALSE'
78      ;
79
80
81  op  :   '+'|'-'|'*'|'/'|'^'|'<'|'<='|'>='|'>'|'=='|'!='|'&'|'&&'|USER_OP|
82          'repeat'|'in'|'?'|'!'|'='|':'|'~'|'$'|'@'|'<-'|'<<-'|'->'|'->>'|'='|'::'|':::'|
83          ','|'||'| '|'
84      ;
```

Εικόνα 5-1: Ο κώδικας του φίλτρου της R που εφαρμόζεται πριν την γραμματική

Το αντικείμενο **R.g4** περιέχει τον λεκτικό και τον συντακτικό αναλυτή της R οι οποίοι αναλυθήκαν εκτενώς στις υποενότητες 3.2 και 3.3 αξίζει να σημειωθεί ότι το λεξιλόγιο (R.tokens) χρησιμοποιείτε και απο το RFilter.g4. Μετά την επιτυχή μεταγλώττιση του RGrammar project παράγονται μεσω του ANTLR οι βοηθητικές κλάσεις της Εικόνας 5-2 που θα χρησιμοποιήσουμε στο project RFrontEnd.



Εικόνα 5-2: Οι κλάσεις που γεννιόνται από το ANTLR στο project RGrammar

5.2 RFrontEnd project

Το project RFrontEnd χρησιμοποιεί το .NET Framework 4.6.1 ώστε να έχουμε την πιο ενημερωμένη έκδοση της C#. Όλες οι λειτουργίες γίνονται μεσω της κλάσης RFacade η οποία είναι υπεύθυνη για το διάβασμα του αρχείου (μέθοδος Parse) ή των αρχείων (μέθοδος ParseSubDirectories) τύπου R που θα δώσει ο χρήστης σαν είσοδο στο πρόγραμμα.

Η σημαντικότερη μέθοδος του project αλλά και όλου του προγράμματος είναι η **Parse** ο κώδικας της οποίας φαίνεται στις Εικόνες 5-3 και 5-4.

Αρχικά, ανοίγει έναν StreamReader με είσοδο ένα αρχείο τύπου R το οποίο το διαβάζει με συγκεκριμένη κωδικοποίηση ανά byte, μετά σειρά έχει η κλάση του AntlrInputStream που απορρόφα το περιεχόμενο του StreamReader και του συμπεριφέρεται σαν έναν πίνακα από χαρακτήρες. Στην συνέχεια, καλείτε ο RLexer, ο λεκτικός αναλυτής της γραμματικής και αυτός με την σειρά του παίρνει την είσοδο του από τον πίνακα του AntlrInputStream και εφαρμόζει ταιριάσματα (matches) ανάμεσα στην είσοδο του και στους κανόνες του. Έπειτα καλείτε η CommonTokenStream που αποθηκεύει ολόκληρή την συλλογή από όλα τα σύμβολα του RLexer δηλαδή μόνο αυτά που ανήκουν στο προκαθορισμένο κανάλι (default channel) και όχι όσα ανήκουν στο κρυφό κανάλι (Hidden Channel).

```
23          static int Parse(string loc)
24          {
25
26              // Reads characters from a byte stream in a particular encoding.
27              StreamReader reader = new StreamReader(loc);
28
29              // Vacuum all input from a Reader/InputStream and then treat it like a char[] buffer.
30              // Can also pass in a String or char[] to use.
31              AntlrInputStream input = new AntlrInputStream(reader);
32
33              /* A lexer is recognizer that draws input symbols from a character stream. lexer grammars result in a subclass of this object.
34                 A Lexer object uses simplified match() and error recovery mechanisms in the interest of speed.
35               */
36              RLexer lexer = new RLexer(input);
37
38              /* This class extends BufferedTokenStream with functionality to filter token streams to tokens on a particular channel(tokens
39               * where Token.getChannel() returns a particular value). This token stream provides access to all tokens by index or when calling
40               * methods like BufferedTokenStream.getText(). The channel filtering is only used for code accessing tokens via the lookahead
41               * methods BufferedTokenStream.LA(int), LT(int), and LB(int). By default, tokens are placed on the default channel(Token.DEFAULT_CHANNEL),
42               * but may be reassigned by using the->channel(HIDDEN) lexer command, or by using an embedded action to call Lexer.setChannel(int).
43               * Note: lexer rules which use the->skip lexer command or call Lexer.skip() do not produce tokens at all, so input text matched by such a
44               * rule will not be available as part of the token stream, regardless of channel.
45               *
46               *  A collection of all tokens fetched from the token source. The list is considered a complete view of the input once fetchedEOF is set to true.
47               */
48              CommonTokenStream tokens = new CommonTokenStream(lexer);
```

Εικόνα 5-3: Ο κώδικας της μεθόδου Parse (1/2)

Μετά τον λεκτικό αναλυτή, καλείτε ο συντακτικός αναλυτής του φίλτρου ο RFilter, ο οποίος παίρνει την συλλογή από τα σύμβολα (του λεκτικού) και εφαρμόζει τους κανόνες του δηλαδή την απαλοιφή τον νέων γραμμών ανάμεσα στις εκφράσεις. Επίσης καλείτε ο συντακτικός αναλυτής της R ο οποίος παίρνει την συλλογή από τα σύμβολα που έχουν φιλτραριστεί (αφού έχει γίνει reset στον δείκτη τους για να μπορούν να γίνουν reuse) και εφαρμόζονται οι κανόνες της γραμματικής του με αποτέλεσμα να δημιουργηθεί το Συντακτικό Δένδρο (tree).

Τέλος, χρησιμοποιούμε το συντακτικά δένδρο ως είσοδο στις υπόλοιπες κλάσεις για να φτιάξουμε τα αρχεία τύπου DOT και με την βοήθεια του εργαλείου Graphviz να δημιουργήσουμε εικόνες τύπου gif με το κανονικό και το αφηρημένο συντακτικό δένδρο, εδώ χρησιμοποιούμε και πολλά από τα μοτίβα που περιγράφηκαν στην ενότητα 4 καθώς και

για την δημιουργία του πίνακα συμβόλων (symbol table), τα οποία θα αναλυθούν εκτενώς στην επόμενη ενότητα. Η μέθοδος επιστέφει τον αριθμό τον σφαλμάτων που υπάρχουν εάν δεν υπάρχουν επιστέφει 0 και επαναλαμβάνεται τόσες φορές όσες είναι και τα αρχεία.

```
49
50              // Print tokens BEFORE filtering.
51              //tokens.Fill(); // Get all tokens from lexer until EOF
52              //Console.WriteLine("BEFORE");
53              //foreach (IToken tok in tokens.GetTokens())
54              //{
55              //    Console.WriteLine(tok);
56              //    //Console.WriteLine(tok.Text);
57              //}
58
59              RFilter filter = new RFilter(tokens);   // Parse with filter.
60              filter.stream();                        // Call start rule: stream .
61              tokens.Reset();                         // Reset all the token (actually use seek(0) for reuse)
62
63              //Print tokens AFTER filtering.
64              //Console.WriteLine("AFTER");
65              //foreach (IToken tok in tokens.GetTokens())
66              //{
67              //    Console.WriteLine(tok);
68              //    Console.WriteLine(tok.Text);
69              //}
70
71              RParser parser = new RParser(tokens);   // Parse with RParser.
72              IParseTree tree = parser.prog();      // Call start rule: prog .
73
74              PTPrinter PTvisitor = new PTPrinter(loc);
75              PTvisitor.Visit(tree);
76
77              ASTGenerator ast = new ASTGenerator();
78              ast.Visit(tree);
79
80              ASTPrinter ASTVisitor = new ASTPrinter(loc);
81              ASTVisitor.Visit(ast.M_Root);
82
83              ASTScopeVisitor ASTScopeVisitor = new ASTScopeVisitor();
84              ASTScopeVisitor.Visit(ast.M_Root);
85
86              return (parser.NumberOfSyntaxErrors + filter.NumberOfSyntaxErrors);
87          }
```

Εικόνα 5-4: Ο κώδικας της μεθόδου Parse (2/2)

Επίσης υπάρχουν και όλες οι υλοποιήσεις των μοτίβων που αναφέρθηκαν στην ενότητα 4 ποιο συγκεκριμένα των ASTComposite, ASTEvents, ASTFacrories, ASTIterator, ASTVisitor στους ομότιτλους φακέλους του project.

6 Διάσχιση της υψηλού επιπέδου αναπαράστασης

Σε αυτήν την ενότητα θα ασχοληθούμε με το Συντακτικό Δένδρο (ΣΔ), το Αφηρημένου Συντακτικού Δένδρου (ΑΣΤ) (Neamtiu, et al., 2015) και τα περάσματα που θα κάνουμε σε αυτά με την χρήση του μοτίβου επισκέπτη (visitor). Πιο συγκεκριμένα θα ασχοληθούμε με τα εξής θέματα:

- Δημιουργία ΑΣΔ από το ΣΔ
- Οπτικοποίηση του ΣΔ και του ΑΣΔ
- Αναγνώριση συμβόλων του προγράμματος

Στις παρακάτω υποενότητες θα εξηγηθεί ο κώδικάς που εφαρμόζεται για να επιτευχθούν τα παραπάνω θέματα.

6.1 Δημιουργία ΑΣΔ από το ΣΔ

Για την δημιουργία του ΑΣΔ θα χρειαστεί να κάνουμε ένα πέρασμα στο ΣΔ το οποίο προκύπτει απο το εργαλείο ANTLR όταν έχουμε επιτυχή συντακτική ανάλυση του εκάστοτε αρχείου εισόδου. Για το πέρασμα αυτό έχει δημιουργηθεί η κλάση **ASTGenerator** η οποία αποτελείτε από μεθόδους που υλοποιούν το μοτίβο visitor για κάθε κομβο του ΣΔ. Μέσα από το πέρασμα από αυτές τις μεθόδους θα δημιουργηθεί το ΑΣΔ, τώρα θα δούμε το περιεχόμενο που έχουν αυτές οι μέθοδοι.

Για τους μη τερματικούς κόμβους κάνουμε τις εξής ενέργειες. Εάν δεν είμαστε στον κόμβο της ρίζας, αποθηκεύουμε τον γονιό του κόμβου. Φτιάχνουμε ένα νέο στοιχείο με τύπο όμοιο του κόμβου, ανανεώνουμε την στοίβα των γονιών κάνοντας ώθηση (push) το στοιχειού που μόλις φτιάξαμε. Στον γονικό κομβο προσθέτουμε ένα νέο παιδί δίνοντας το στοιχείο που μόλις φτιάξαμε μαζί με τον τύπο του. Επισκεπτόμαστε όλα τα παιδία του στοιχείου δίνοντας σαν όρισμα το περιεχόμενο (context) του και τον τύπο του και έπειτα καλείτε η αντίστοιχη εικονική μέθοδος επίσκεψης του ANTLR. Αφού έχει επιστέψει από τα τις επισκέψεις των παιδιών βγάζω (pop) την κορυφή από την στοίβα με τους γονείς. Στην Εικόνα 6 -1 φαίνεται ενδεικτικά ο κώδικας του κόμβου της έκφρασης for.

```
public override int VisitExprFor( RParser.ExprForContext context)
{
    RASTComposite parent = m_parents.Peek();

    // PREORDER ACTIONS
    // Create new element
    RASTComposite newElement = new RFor(parent);
    // Update parents stack
    m_parents.Push(newElement);

    // Add new element to the parent's descentants
    parent.AddChild(newElement, m_currentContext.Peek());

    // VISIT CHILDREN
    VisitElementInContext(context.ID(), ContextType.CT_EXPR_FOR_NAME);
    VisitElementInContext(context.for_list, ContextType.CT_EXPR_FOR_VECTOR);
    VisitElementInContext(context.for_body, ContextType.CT_EXPR_FOR_BODY);

    // POSTORDER ACTIONS

    // Update parents stack
    m_parents.Pop();

    return 0;
}
```

Εικόνα 6-1: Κώδικας της μεθόδου VisitExprFor της κλάσης ASTGenerator

Για τους τερματικούς κόμβους δεν χρειάζεται κάποια ειδική μεταχείριση και για αυτό και δεν αναφέρω κάτι. Μετά το πέρασμα του ΣΔ με την βοήθεια των visitor μεθόδων που ανέλυσα προηγουμένως όλη η πληροφορία σχετικά με τη δομή του ΑΣΤ έχει αποθηκευτεί στη δομή που έχουμε κατασκευάσει και είναι διαθέσιμη για περαιτέρω επεξεργασία.

6.2 Οπτικοποίηση του ΣΔ και του ΑΣΔ

Ο ευκολότερος τρόπος για να γίνει πιο κατανοητή η πληροφορία που υπάρχει στο ΣΔ και στο ΑΣΔ χρειάζεται να τα οπτικοποιήσουμε. Η Οπτικοποίηση αυτών θα γίνει εφαρμόζοντας δυο (2) περάσματα. Σε κάθε πέρασμα θα παίρνουμε την απαραίτητη πληροφορία και θα την γράφουμε σε ένα αρχείο τύπου DOT το οποίο όταν ολοκληρώνεται θα το τρέχει το εργαλείο Graphviz.

Για την οπτικοποίηση του ΣΔ θα γίνει χρήση της κλάσης **PTPrinter** η οποία αποτελείτε από μεθόδους που υλοποιούν το μοτίβο visitor για κάθε κομβο του ΣΔ. Για όλους τους κόμβους εκτελούνται οι εξής ενέργειες. Δημιουργούμε μια ετικέτα τύπου αλφαριθμητικού η οποία

περιέχει το πρόθεμα του κανόνα μια κάτω παύλα έναν αύξοντα αριθμό και έπειτα την γράφουμε στο αρχείο με όνομα ίδιο με αυτό του αρχείου και κατάληξης dot. Κάνουμε ώθηση (push) σε μια στοίβα επισκεπτόμαστε το περιεχόμενο του (κόμβου) και έπειτα βγάζουμε (pop) την ετικέτα του τρέχοντα κανόνα. Στην Εικόνα 6-2 φαίνεται ενδεικτικά ο κώδικας του επισκέπτη (visitor) του κόμβου της έκφρασης for για το τύπωμα του ΣΔ.

```
public override int VisitExprFor( RParser.ExprForContext context)
{
    string label = "For_" + ms_ASTElementCounter.ToString();
    m_outputStream.WriteLine("\"{0}\"->\"{1}\";", m_PTPath.Peek(), label);
    ms_ASTElementCounter++;
    m_PTPath.Push(label);

    base.VisitExprFor(context);

    // POSTORDER ACTIONS
    m_PTPath.Pop();

    return 0;
}
```

Εικόνα 6-2: Κώδικας της μεθόδου VisitExprFor της κλάσης PTPrinter

Αφού έχουμε επισκεφτεί όλους τους κόμβους με αποτέλεσμα το αρχείο γλώσσας DOT να έχει ολοκληρωθεί και όσο είμαστε στην μέθοδο VisitProg (Prog είναι ο κανόνας εκκίνησης της γραμματικής R) τρέχουμε το Graphviz ξεκινώντας μια νέα διεργασία, βλέπε Εικόνα 6-3.

```
Process process = new Process();
// Configure the process using the StartInfo properties.
process.StartInfo.FileName = @".\Graphviz\bin\dot.exe";
process.StartInfo.Arguments = "-Tgif " + m_outputFile + " -o" +
    Path.GetFileNameWithoutExtension(m_outputFile) + ".gif";
process.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
process.Start();
process.WaitForExit();// Waits here for the process to exit.
```

Εικόνα 6-3: Κομμάτι κώδικα της μεθόδου VisitProg της κλάσης PTPrinter, η οποία καλεί το Graphviz

Για την οπτικοποίηση του ΣΔ θα γίνει χρήση της κλάσης **ASTPrinterVisitor** η οποία αποτελείτε από μεθόδους που υλοποιούν το μοτίβο visitor για κάθε κομβο του ΑΣΔ. Για όλους τους κόμβους εκτελούνται οι εξής ενέργειες. Δημιουργούμε δυο (2) μεταβλητές τύπου αλφαριθμητικού οπού η πρώτη έχει το όνομα της ομάδας (clusterName) και η δεύτερη η οποία περιέχει όνομα του περιεχομένου (contextName). Έπειτα για κάθε μέθοδο επισκεπτόμαστε το περιεχόμενο ή τα περιεχόμενα του κανόνα και φτιάχνουμε έναν υπο-

γράφο (subgraph) με ειδικό στυλ για τους τερματικούς κόμβους πράσινο χρώμα και τους μη τερματικούς κίτρινο χρώμα. Ενδεικτικά ο κανόνας for έχει τα περιεχόμενα CT_EXPR_FOR_NAME, CT_EXPR_FOR_VECTOR και CT_EXPR_FOR_BODY και στην Εικόνα 6-4 φαίνεται ενδεικτικά ο κώδικας του επισκέπτη (visitor) του κόμβου της έκφρασης for για το τύπωμα του ΑΣΔ.

```
public override int VisitFor(RASTElement currentNode)
{

    RASTComposite current = currentNode as RASTComposite;
    string clusterName;
    string contextName;
    m_outputStream.WriteLine("\"{0}\"->\"{1}\"", currentNode.M_Parent.M_Label, currentNode.M_Label);

    // Visit for name(of vector) context
    if (current.GetNumberOfContextElements(ContextType.CT_EXPR_FOR_NAME) > 0) { // if not a leaf
        clusterName = "cluster" + ms_clusterCounter++;
        contextName = ContextType.CT_EXPR_FOR_NAME.ToString();
        m_outputStream.WriteLine(
            "subgraph {0} {{\n node [style=filled,color=white];\n style=filled;\n color=lightgrey;\n label = \"{1}\";\n",
            clusterName, contextName);
        foreach (RASTElement element in current.GetContextChildren(ContextType.CT_EXPR_FOR_NAME))
        {
            m_outputStream.WriteLine("\"{0}\"", element.M_Label);
            if (RConfigurationSettings.m_nodeTypeConfiguration[element.M_NodeType].M_Color != Color.C_DEFAULT)
            {
                m_outputStream.WriteLine(" [fillcolor = " + RConfigurationSettings.m_nodeTypeConfiguration[element.M_NodeType].
                    M_ColorName + "]");
            }
        }
        m_outputStream.WriteLine("}");
    }

    // Visit for vector context
    if (current.GetNumberOfContextElements(ContextType.CT_EXPR_FOR_VECTOR) > 0){ // if not a leaf
        clusterName = "cluster" + ms_clusterCounter++;
        contextName = ContextType.CT_EXPR_FOR_VECTOR.ToString();
        m_outputStream.WriteLine(
            "subgraph {0} {{\n node [style=filled,color=white];\n style=filled;\n color=lightgrey;\n label = \"{1}\";\n",
            clusterName, contextName);
        foreach (RASTElement element in current.GetContextChildren(ContextType.CT_EXPR_FOR_VECTOR))
        {
            m_outputStream.WriteLine("\"{0}\"", element.M_Label);
            if (RConfigurationSettings.m_nodeTypeConfiguration[element.M_NodeType].M_Color != Color.C_DEFAULT)
            {
                m_outputStream.WriteLine(" [fillcolor = " + RConfigurationSettings.m_nodeTypeConfiguration[element.M_NodeType].
                    M_ColorName + "]");
            }
        }
        m_outputStream.WriteLine("}");
    }

    // Visit for body context
    if (current.GetNumberOfContextElements(ContextType.CT_EXPR_FOR_BODY) > 0) { // if not a leaf
        clusterName = "cluster" + ms_clusterCounter++;
        contextName = ContextType.CT_EXPR_FOR_BODY.ToString();
        m_outputStream.WriteLine(
            "subgraph {0} {{\n node [style=filled,color=white];\n style=filled;\n color=lightgrey;\n label = \"{1}\";\n",
            clusterName, contextName);
        foreach (RASTElement element in current.GetContextChildren(ContextType.CT_EXPR_FOR_BODY))
        {
            m_outputStream.WriteLine("\"{0}\"", element.M_Label);
            if (RConfigurationSettings.m_nodeTypeConfiguration[element.M_NodeType].M_Color != Color.C_DEFAULT)
            {
                m_outputStream.WriteLine(" [fillcolor = " + RConfigurationSettings.m_nodeTypeConfiguration[element.M_NodeType].
                    M_ColorName + "]");
            }
        }
        m_outputStream.WriteLine("}");
    }

    base.VisitFor(currentNode);

    return 0;
}
```

Εικόνα 6-4: Κώδικας της μεθόδου VisitFor της κλάσης ASTPrinterVisitor

Και εδώ ισχύει το ίδιο με την περίπτωση του ΣΔ, αφού έχουμε επισκεφτεί όλους τους κόμβους με αποτέλεσμα το αρχείο γλώσσας DOT να έχει ολοκληρωθεί και όσο είμαστε στην μέθοδο τρέχουμε το Graphviz ξεκινώντας μια νέα διεργασία, βλέπε Εικόνα 6-3.

6.3 Αναγνώριση συμβόλων του προγράμματος

Ένα ακόμα πέρασμα χρειάζεται και για την αναγνώριση των συμβόλων, τα οποία αποθηκεύονται στον πίνακα συμβόλων. Αυτή η πληροφορία είναι ιδιαίτερα χρήσιμη για το πίσω μέρος (back end) του μεταγλωττιστή.

Το πέρασμα εφαρμόζεται στο ΑΣΤ με την βοήθεια της κλάσης **ASTScopeVisitor**. Πιο συγκεκριμένα υπάρχει το global scope το οποίο περιέχει όλες τις μεταβλητές και τις κλήσεις συναρτήσεων του προγράμματος και όχι των βιβλιοθηκών της R. Κάθε συνάρτηση έχει το δικό της scope και για την αναγνώριση των συμβόλων χρησιμοποιούμε τις μεθόδους VisitProg (για να ξέρουμε ποτέ είμαστε στο global scope και πότε βγαίνουμε), VisitDefineFunction (για να ξέρουμε πότε είμαστε στο scope της συνάρτησης και τις παραμέτρους της, VisitAssignmentOpetators (για να ξέρουμε πότε ακριβώς έχουμε αναθέσεις) και την VisitIDENTIFIER (για να ξέρουμε ποιες μεταβλητές χρησιμοποιεί μια συνάρτηση ή το κύριο πρόγραμμα μας). Στην Εικόνα 6-5 φαίνεται ενδεικτικά ο κώδικας του VisitAssignmentOpetators.

```
public override int VisitAssignmentOpetators(RASTElement currentNode)
{
    RASTComposite current = currentNode as RASTComposite;

    // Visit assignment opetator left context
    if ( current.GetNumberOfContextElements(ContextType.CT_EXPR_ASSIGNMENT_OPETATORS_LEFT) > 0)
    {
        foreach (RASTElement element in current.GetContextChildren(ContextType.CT_EXPR_ASSIGNMENT_OPETATORS_LEFT))
        {
            scopeSystem.AddSymbol(element.M_Label,"Local Variable");
        }
    }

    base.VisitAssignmentOpetators(currentNode);

    return 0;

}
```

Εικόνα 6-5: Κώδικας της μεθόδου VisitAssignmentOpetators της κλάσης ASTScopeVisitor

Στον πίνακα συμβόλων υπάρχει το όνομα του συμβόλου σε ποιο scope το είδαμε και ένας χαρακτηρισμός που μας ενημερώνει εάν το σύμβολο είναι τοπικό ή μη οριοθετούμενο, σε αυτήν την περίπτωση μας λέει σε ποιο scope υπάρχει. Αφού έχουμε επισκεφτεί όλους τους κόμβους τότε αποθηκεύουμε τον πίνακα συμβόλων, στιγμιότυπα του οποίου θα δείτε στην επόμενη ενότητα.

7 Δοκιμές γραμματικής και αποτελέσματα

7.1 Δοκιμές γραμματικής

Αρχικά βρήκα από το διαδίκτυο 163 αρχεία της γλώσσας R τα οποία τα έλεγξα με το IDE RStudio για να είμαι σίγουρος ότι είναι όντως σωστά συντακτικά. Τα αρχεία αυτά βρίσκονται στον φάκελο '**Tests Repo**'. Ύστερα μετέφερα τα αρχεία του φακέλου 'Tests Repo' στον φάκελο '**Testbench**', είναι ο φάκελος από τον οποίων παίρνει την είσοδο του το πρόγραμμα μου. Έτρεξα το πρόγραμμα με την δική μου γραμματική της R και δεν έβγαλε κανένα συντακτικό λάθος (ο αριθμός των συντακτικών λαθών εμφανίζεται στην κονσόλα). Αξίζει να σημειωθεί ότι 42 αρχεία που υπάρχουν στον υποφάκελο **'Rscripts 1 (42 files)'** του φακέλου 'Tests Repo' παρέχονται από τον οργανισμό της R για τέτοιου είδους χρήση δηλαδή τον έλεγχο της γραμματικής.

7.2 Αποτελέσματα προγράμματος

Από την επιτυχή εκτέλεση του προγράμματος προκύπτουν κάποια αρχεία εξόδου στον φάκελο με την διεύθυνση **'RFrontEndSolution\RFrontEnd\bin\Debug'**. Για κάθε αρχείο εισόδου δημιουργείται μια πεντάδα από αρχεία η οποία περιλαμβάνει:

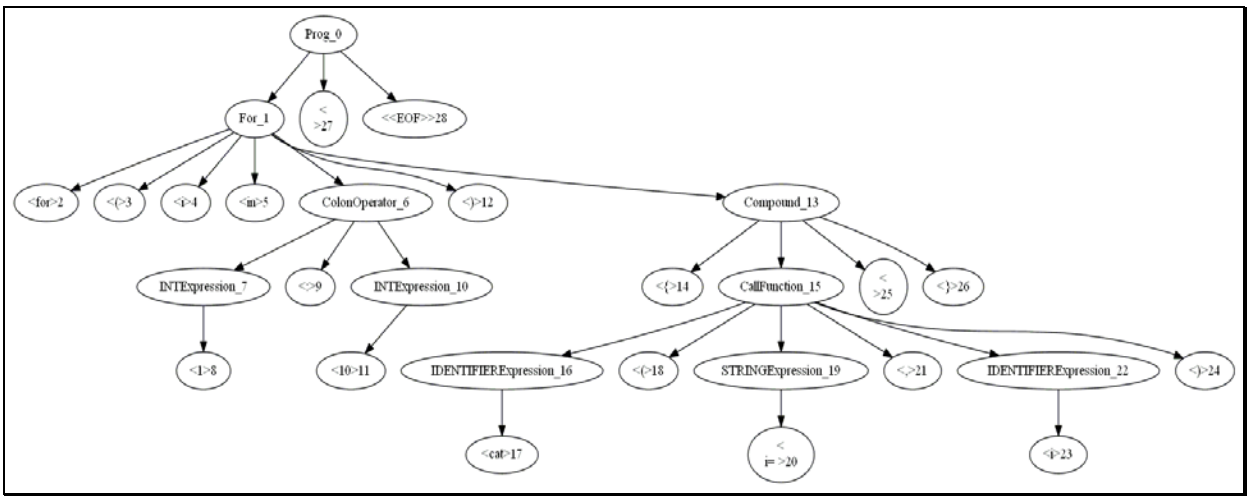
- ένα αρχείο τύπου dot, με το όνομα του αρχείου της εισόδου
- ένα αρχείο τύπου gif που περιέχει την οπτικοποίηση του ΣΔ, με το όνομα του αρχείου της εισόδου.
- ένα αρχείο τύπου dot, με το όνομα του αρχείου της εισόδου και το επίθημα AST
- ένα αρχείο τύπου gif που περιέχει την οπτικοποίηση του ΑΣΔ, με το όνομα του αρχείου της εισόδου και το επίθημα AST
- ένα αρχείο τύπου txt που περιέχει τον πίνακα συμβόλων, με το όνομα του αρχείου της εισόδου και το επίθημα ST

Στον φάκελο **'Results with .gifs'** υπάρχουν όλα τα αποτελέσματα και των 163 αρχείων σε ομότιτλους υποφακέλους. Η δημιουργία των αρχείων τύπου gif μεσω του προγράμματος

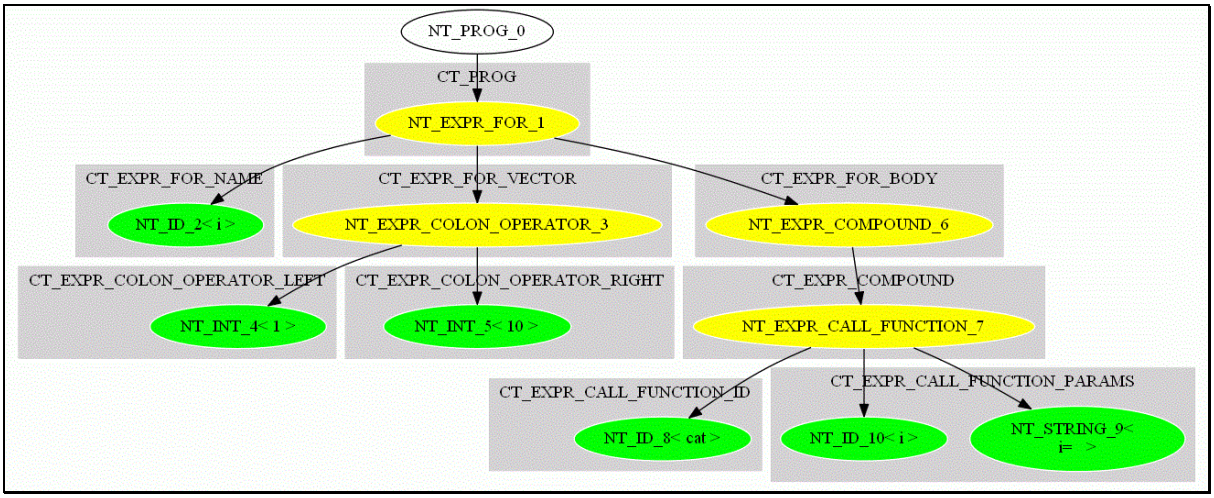
Graphviz δεν ήταν εφικτή για τρία (3) αρχεία επειδή ήθελαν αρκετή υπολογιστική ισχύ λόγω της πολυπλοκότητας τους. Ακολουθεί παράδειγμα ενός προγράμματος με μια απλή for και τις δυο (2) απεικονίσεις των ΣΔ και ΑΣΔ στις Εικόνες 7-1. 7-2 και 7-3.

```
1  for (i in 1:10) {
2    cat("\ni= ",i)
3  }
```

Εικόνα 7-1: Κώδικας προγράμματος με μια απλή for



Εικόνα 7-2: Απεικόνιση του ΣΔ (Συντακτικού Δένδρου) μιας for



Εικόνα 7-3: Απεικόνιση του ΑΣΔ (Αφηρημένου Συντακτικού Δένδρου) μιας for

Ακολουθεί ενδεικτικό παράδειγμα ενός προγράμματος μαζί με τον πίνακα συμβόλων που παράγεται στην Εικόνα 7-4 το, οποίο περιέχει αρκετές ιδιαίτερες περιπτώσεις.

```
d =2;

k <- function (){
  y <- 5

  f <- function() {

    g <- function(x) {
      d = d-1;
      cat("d = ",d)

      if ( d < 3){ k() }
      else{ f() }

      result = x+y;
      cat("\nresult = ",result)
      return (g)
    }

    return(g)
  }

  return (f)
}

m = 45;

xx <- k()  # xx points to f
xxx <- xx() #xxx points to g
xxxx <-xxx(3)
```

Scope	Variable	Typification
Global	d	Local Variable
k	y	Local Variable
g	x	Formal Parameter
g	d	Local Variable
g	cat	Free (System)
g	k	Funtion Call
g	f	Funtion Call
g	result	Local Variable
g	y	Free (Exists in k)
Global	m	Local Variable
Global	xx	Local Variable
Global	k	Funtion Call
Global	xxx	Local Variable
Global	xxxx	Local Variable

Symbol Table

Εικόνα 7-4: Κώδικας μαζί με τον πίνακα συμβόλων που παράγεται

8 Συμπεράσματα - μελλοντικές κατευθύνσεις

8.1 Συμπεράσματα

Η συγκεκριμένη εργασία περιλαμβάνει το εμπρός κομμάτι (front end) του μεταγλωττιστή της γλώσσας R δηλαδή τον λεκτικό αναλυτή, τον συντακτικό αναλυτή, την δημιουργία της ενδιάμεσης αναπαράστασης και τον πίνακα συμβόλων. Το μεγαλύτερο ποιοτικά μέρος της εργασίας το αφιέρωσα στην συγγραφή της γραμματικής της R, δηλαδή να εκμαιεύσω τους κανόνες της από τον ορισμό της γλώσσας και φυσικά να τους ελέγξω και στην υλοποίηση των μοτίβων σχεδίασης.

Το ότι χρησιμοποιήθηκε το εργαλείο ANTLR και όχι τα καθιερωμένα εργαλεία Flex & Bison (Levine, 2009) μου έδωσε την δυνατότητα να χρησιμοποιήσω μια αντικειμενοστραφή γλώσσα την C# και όχι την γλώσσα υψηλού επιπέδου C (Kernigham & Ritchie, 1988). Έτσι, εκμεταλλευόμενος όλα τα χαρακτηριστικά μιας αντικειμενοστραφής γλώσσας δηλαδή τον πολυμορφισμό και την κληρονομικότητα κατάφερα να φτιάξω έναν κώδικα **εύκολα συντηρήσιμο και αναγνώσιμο κώδικα**.

Χωρίς την βοήθεια των μοτίβων σχεδίασης δεν θα ήταν εφικτή η κατασκευή ή για να είμαι ποιο συγκεκριμένος θα ήταν πολύ δύσκολη η δημιουργία του εμπρόσθιου τμήματος (front end) του μεταγλωττιστή λόγω του μεγάλου μεγέθους του κώδικα που χρειαζόταν. Ενώ τώρα με έναν **μικρό σε μέγεθος κώδικα (περίπου 10.000 γραμμές)** έχει υλοποιηθεί όλη η εργασία. Τα μοτίβα ακόμα βοήθησαν στην **σωστή σχεδίαση και οργάνωση** χωρίζοντας όλες τις κλάσεις με βάση το μοτίβο που εφαρμόστηκε ώστε να δημιουργηθούν **ξεχωριστά κομμάτια ανάπτυξης**.

8.2 Μελλοντικές κατευθύνσεις

Από την στιγμή που το εμπρόσθιο μέρος (front end) του μεταγλωττιστή είναι έτοιμο το αμέσως επόμενο βήμα είναι η **δημιουργία του πίσω μέρους (back end)** του μεταγλωττιστή το οποίο περιλαμβάνει την βελτιστοποίηση και την παραγωγή κώδικα μηχανής. Έτσι θα έχει δημιουργηθεί ένας ολοκληρωμένος μεταγλωττιστής της R. Η υλοποίηση **αναπτυξιακού περιβάλλοντος μεταγλώττισης** με κειμενογράφο θα ήταν εφικτή εάν υπάρχει ένας

ολοκληρωμένος μεταγλωττιστές, αξίζει να αναφερθεί ότι μόνο το RStudio παρέχει ένα ολοκληρωμένο IDE.

Μια πιο προχωρημένη ιδέα θα ήταν η υλοποίηση ενός αυτοματοποιημένου συστήματος δηλαδή μια **γεννήτρια παραγωγής κώδικα** (Fraser, et al., 1992) η οποία θα δημιούργει το περιεχόμενο των κλάσεων των μοτίβων που αναλύθηκαν από την εκάστοτε γραμματικής.

Βιβλιογραφία

Chambers, J. M. & Hastie, T. J., 1991. Chapter 2. Στο: *Statistical Models in S.* California: CRC Press, Inc., pp. 18-42.

Emden, G., Koutsofios, E. & North, S. C., 2006. Drawing graphs with dot. *dot User's Manual*, 26 January, pp. 2-17.

Fraser, C., Hanson, D. & Proebsting, T., 1992. Engineering a simple, efficient code-generator generator. *ACM Letters on Programming Languages and Systems (LOPLAS),* I(3), pp. 213-226.

Gamma, E. και συν., 1995. Composite. Στο: *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston: Pearson Education India, pp. 183-195.

Gamma, E. και συν., 1995. Facade. Στο: *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston: Pearson Education India, pp. 208-217.

Gamma, E. και συν., 1995. Iterator. Στο: *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston: Pearson Education India, pp. 289-304.

Gamma, E. και συν., 1995. Observer. Στο: *Design Patterns: Elements of Reusable Object-Oriented Software.* Bosotn: Pearson Education India, pp. 326-337.

Gamma, E. και συν., 1995. Visitor. Στο: *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston: Pearson Education India, pp. 366-381.

Ihaka, R. & Gentleman, R., 1996. A language for data analysis and graphics. *Journal of Computational and Graphical Statistics,* 5(3), pp. 299-314.

Kernigham, B., Ritchie, D. & Ejeklint, P., 1988. *The C Programming Language.* 2nd επιμ. New Jersey: prentice-Hall Englewood Cliffs.

Levine, J., 2009. *Flex & Bison.* 1st επιμ. California: O'Reilly Media, Inc..

Liberty, J. & MacDonald, B., 2006. C# and .NET Programming. Στο: *Learning C# 2005.* California: O'Reilly Media, Inc., pp. 1-19.

Neamtiu, I., Foster, J. & Hicks, M., 2005. Understanding source code evolution using abstract syntax tree matching. *ACM SIGSOFT Software Engineering Notes,* 30(4), pp. 1-5.

Parr, T., 2013. *The Definitive ANTLR 4 Reference*. 2nd επιμ. North Carolina: Pragmatic Bookshelf.

Peng, R., 2015. Scoping Rules of R. Στο: *R Programming for Data Science*. s.l.:Lulu.com, pp. 79-87.

Team R Core, 2016. Parser. Στο: *R Language Definition*. 3.3.2 επιμ. Vienna: R Core Team, pp. 45-51.

Παραρτήματα

Κατάλογος διαγραμμάτων

Απόδοση όρων

bound	οριοθετημένα
environment	περιβάλλον, οπού μπορεί να ανήκει μια μεταβλητή
enumerations	απαριθμήσεις
global	μεταβλητή που είναι διάθεση σε όλο το πρόγραμμα
iterator	επαναλήπτης
lexer	κομμάτι του μεταγλωττιστή που σπάει το κείμενο του προγράμματος σε σύμβολα
lexical scope	λεκτική οριοθέτηση
local	τοπική μεταβλητή
null	μεταβλητή χωρίς τιμή
parser	κομμάτι του μεταγλωττιστή που μετατρέπει τα σύμβολα σε parse tree
server	εξυπηρετητής ή διακομιστής
unbound	μη οριοθετημένα
web browser	φυλλομετρητή ιστοσελίδων

Συντομογραφίες

AGPL	Affero General Public License
ANTLR	ANother Tool for Language Recognition
AST	Abstract Syntax Tree
BE	Back End
BSD	Berkeley Software Distribution
CLI	Common Language Infrastructure
CLR	Common Language Runtime
EOF	End Of File
FE	Front End
GNU	GNU's Not Unix!
IDE	Integrated Development Environment
JIT	Just In Time
NA	Not Available
NaN	Not a Number