Κατασκευή Front-End για την γλώσσα R

ΠΑΡΟΥΣΙΑΣΗ

ΑΛΕΞΑΝΔΡΟΣ ΠΛΕΣΣΙΑΣ

Περιγραφή

Η εργασία αναφέρεται στην σχεδίαση του εμπρόσθιου τμήματος (Front End) μεταγλωττιστή για την γλώσσα R. Η γλώσσα R έχει μια μεγάλη βάση χρηστών και αυτό την κάνει πολύ ενδιαφέρουσα σε ότι αφορά της εφαρμογές στην σχεδίαση ψηφιακών συστημάτων για την επιτάχυνση της επεξεργασίας των προγραμμάτων της.

Front-End

Τι είναι ένας μεταγλωττιστής? (1/2)

Ένας μεταγλωττιστής είναι ένα πρόγραμμα που διαβάζει ένα πρόγραμμα γραμμένο σε μία γλώσσα και το μετατρέπει σε ένα πρόγραμμα γραμμένο σε άλλη γλώσσα.

Ο πηγαίος κώδικας είναι τυπικά σε μία γλώσσα υψηλού επιπέδου (π. χ. Pascal, C, C ++, Java, Perl, C #, κ.λπ.). Ο εκτελέσιμος κώδικας μπορεί να είναι μία σειρά εντολών μηχανής που μπορούν να εκτελεστούν από την CPU άμεσα ή μπορεί να είναι μια ενδιάμεση αναπαράσταση που ερμηνεύεται από μια εικονική μηχανή (π. χ. Java byte code).

Εν ολίγοις, ο μεταγλωττιστής μετατρέπει ένα πρόγραμμα από μια μορφή αναγνώσιμη από τον άνθρωπο (human-readable) σε μια μορφή αναγνώσιμη από την μηχανή (machine-readable).

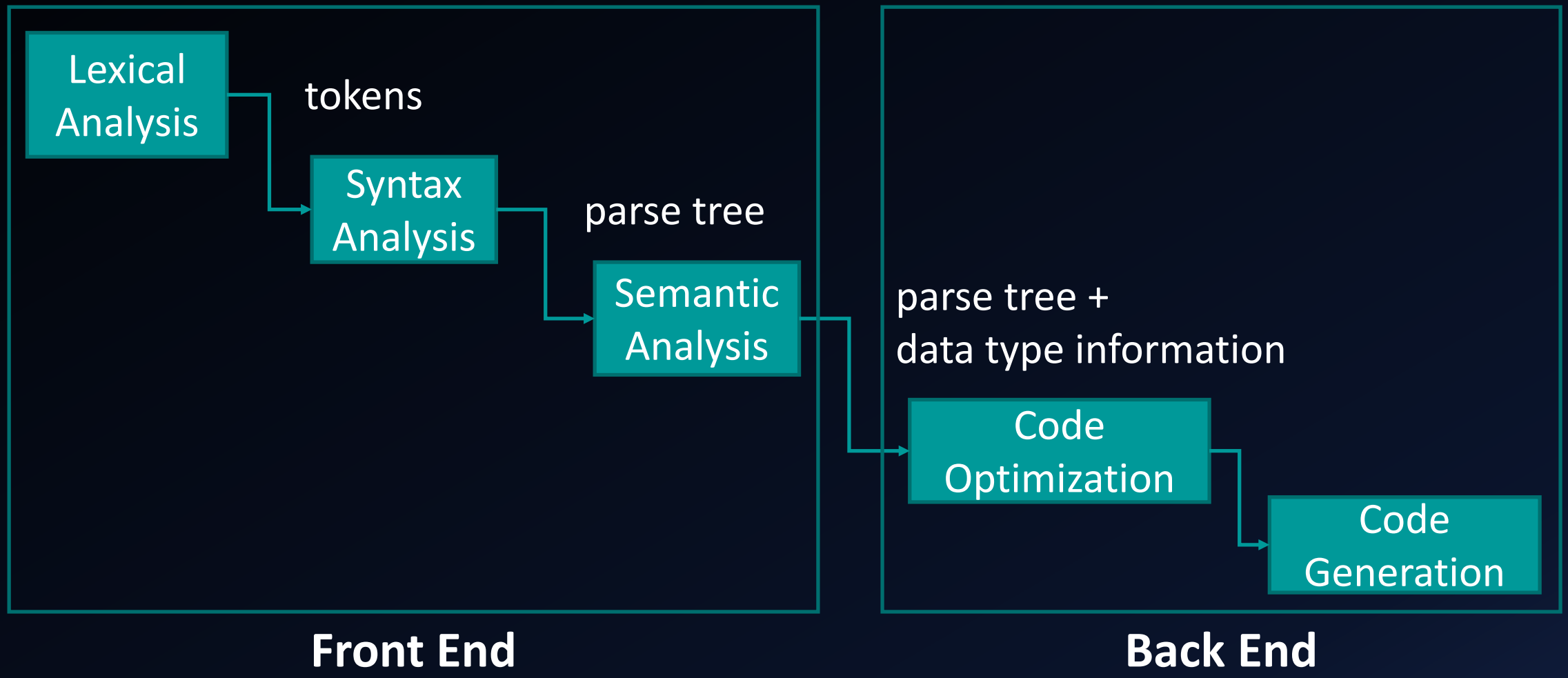
Τι είναι ένας μεταγλωττιστής? (2/2)

Τώρα, το πώς λειτουργεί ένας μεταγλωττιστής? Είναι πραγματικά περίπλοκος. Θα προσπαθήσω όμως να περιγράψω εν συντομία τα κύρια στάδια της μεταγλώττισης (αλλά αυτό θα είναι μια πολύ βιαστική ανασκόπηση).

- ***Lexing** - να σπάσει το κείμενο του προγράμματος σε “tokens”. Τα tokens είναι οι “λέξεις” της γλώσσας προγραμματισμού, όπως αναγνωριστικά (keywords, variable names, function names, κτλ.) ή των τελεστών (=, *, & κτλ.).
- ***Parsing** - μετατρέπουν την ακολουθία των tokens σε ένα parse tree, το οποίο είναι μια δομή δεδομένων που αντιπροσωπεύει διάφορες γλωσσικές δομές όπως: type declarations, variable declarations, function definitions, loops, expressions, κτλ.
- **Βελτιστοποίηση** - αξιολογεί σταθερές εκφράσεις, βελτιστοποιεί αχρησιμοποίητες μεταβλητές ή απρόσιτο (unreachable) κώδικα, ξετυλίγει βρόχους (unroll loops) ανν είναι δυνατόν, κτλ.
- **Μετάφραση** του συντακτικού δένδρου σε εντολές μηχανής (ή JVM byte code).

Και πάλι, τονίζω ότι αυτή είναι μια πολύ σύντομη περιγραφή. Οι σύγχρονη μεταγλωττιστές είναι πολύ έξυπνοι και κατά συνέπεια πολύ περίπλοκη.

Τι είναι το Front End?



Στατιστική γλώσσα προγραμματισμού R

Περιγραφή της στατιστική γλώσσα προγραμματισμού 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 στα μέσα της δεκαετίας του 90 (σε αυτούς οφείλει το όνομά της). Από το 1997 αναπτύσσεται από το R Development Core Team.

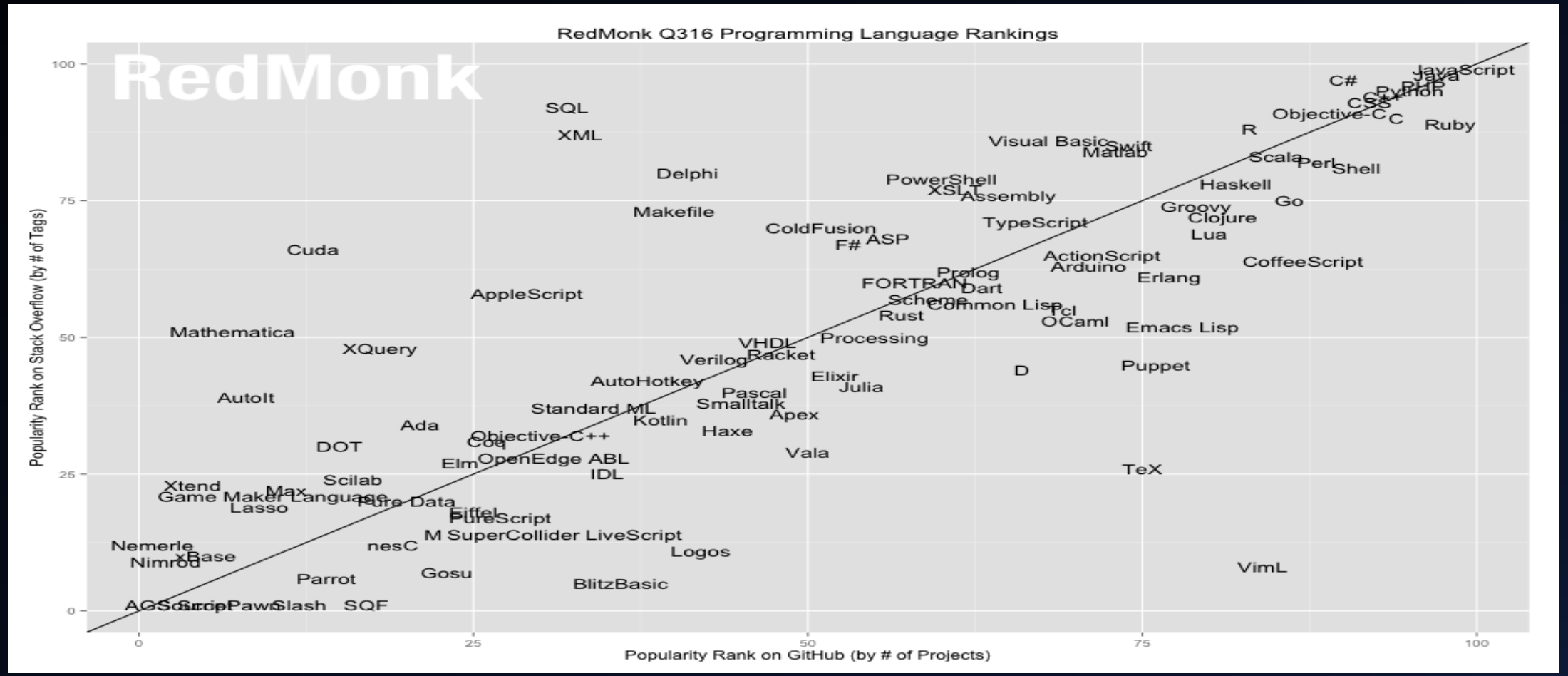
Είναι γλώσσα ανοικτού κώδικα (open-source language) αποτελεί μέρος του έργου GNU και επομένως, δεν απαιτείται η αγορά άδειας για τη χρήση της δηλαδή είναι δωρεάν, άρα όλοι μπορούν να έχουν πρόσβαση στον κώδικά της και να κάνουν διορθώσεις. Επιπλέον, επιτρέπει στον χρήστη να αλληλεπιδρά και με άλλες γλώσσες (C/C++, Java, Python), με αρχεία δεδομένων (Excel, Access) και με άλλα στατιστικά πακέτα (SAS, Stata, SPSS, Minitab).

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.

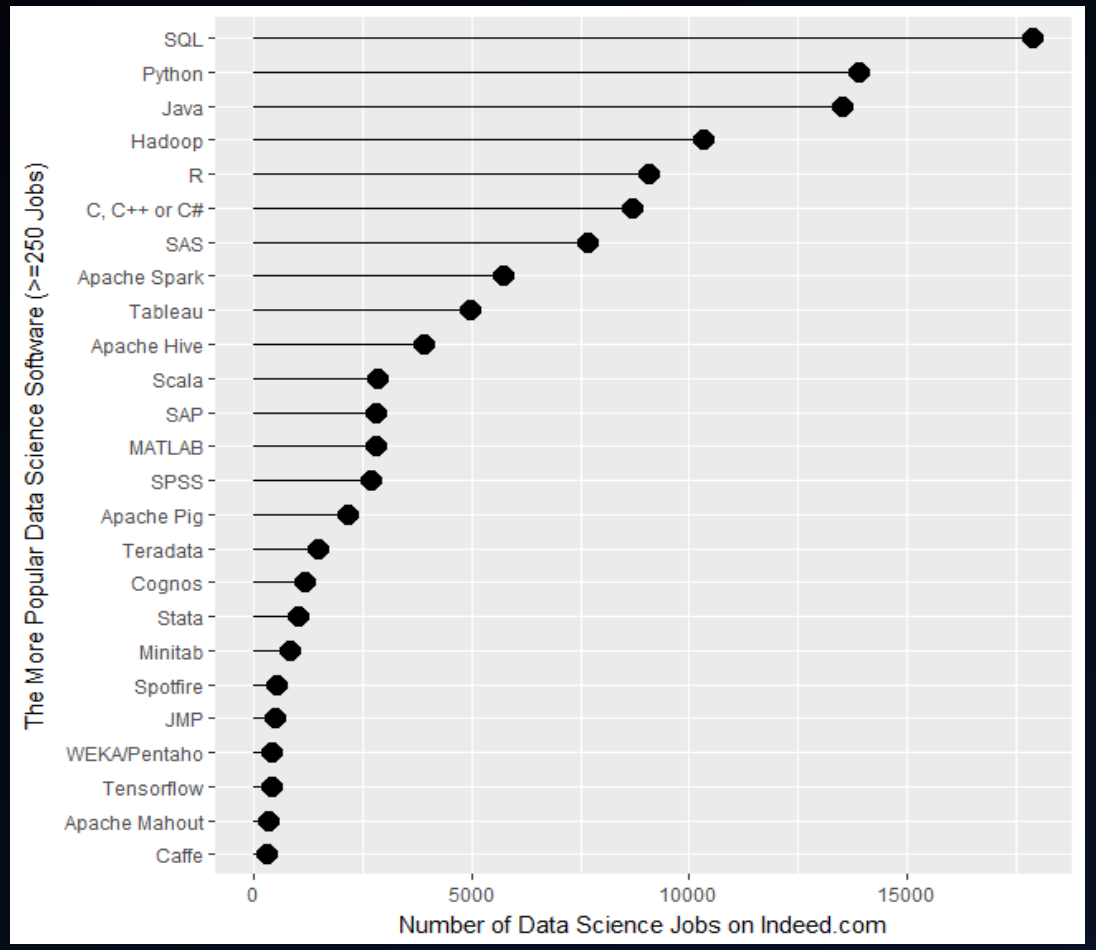
Πλεονεκτήματα και μειονεκτήματα της γλώσσας R

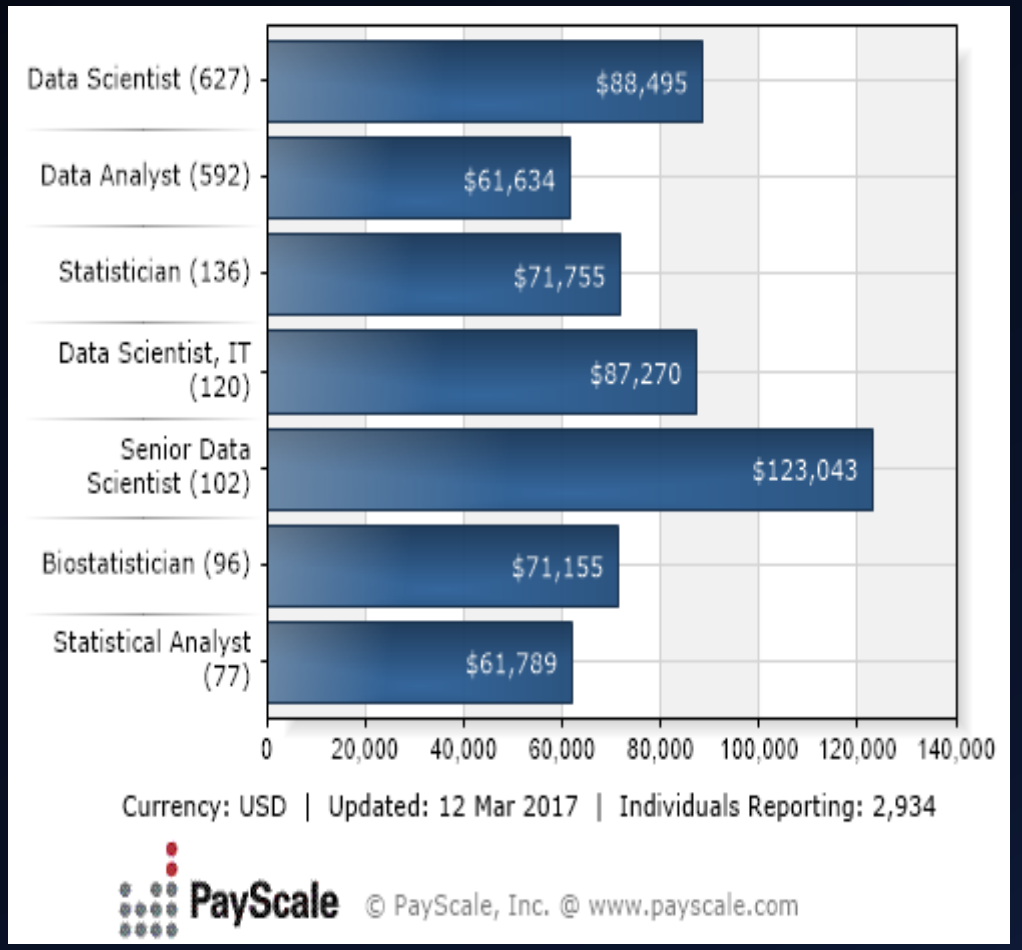
- ✔ Σήμερα έχει πάνω από 10.254 πακέτα (είχε 4.284 το 2013 και ~6.000 το 2015) τα οποία αναφέρονται σε πολλούς επιστημονικούς τομείς.
- ✔ Χρησιμοποιείται και σε άλλους τομείς όπως στα οικονομικά, στην αστρονομία, στην χημεία, στην φαρμακευτική, στην ιατρική, στο μάρκετινγκ και αλλα.
- ✔ Χρησιμοποιείται σε εταιρίες όπως οι Microsoft, TIBCO, Google, Oracle, HP, LinkedIn και Facebook για να κάνουν ανάλυση δεδομένων
- ✘ Καταναλώνει πολύ μνήμη.
- ✘ Είναι 'αργή' γλώσσα ως προς τον χρόνο εκτέλεσης των εντολών.

Η δημοφιλία της γλώσσας R (Ιούλιος 2016, 12η θέση)



Η δυναμική της γλώσσας R στην αγορά εργασίας σήμερα





RFrontEnd Solution

RGRAMMAR PROJECT

RFRONTEND PROJECT

Φίλτρο της R

Πρόκειται ουσιαστικά για έναν parser που φιλτράρει τα προγράμματα της R πριν αυτά κάνουν parse με την γραμματική της R, χρησιμοποίει το λεξιλόγιο της γραμματικής, της R. Ο λόγος που γίνεται αυτό είναι οι νέες γραμμές (NL), τις οποίες η γραμματική σε αφήνει να τις εφαρμόσεις οπουδήποτε επιθυμεί ο χρήστης μέσα στον κώδικα.

Φτιάχνοντας ένα φίλτρο το οποίο πετάει τις αλλαγές γραμμών (NL) με τον μηχανισμό HiddenChannel* καταφέρνουμε να μην χρειάζεται να τις εισάγουμε στην γραμματική μας με αποτέλεσμα να έχουμε μια μικρή και κατανοητή γραμματική

*Είναι πεδίο της κλάσης CommonTokenStream του εργαλείου ANTLR (Another Tool For Language Recognition).

Λεκτικός και συντακτικός αναλυτής της R

Η σημαντικότερη δυσκολία ήταν η δημιουργία της γραμματικής της R μιας και δεν υπήρχε κάπου διαθέσιμη*, έτσι χρειάστηκε η ανάπτυξη της από την αρχή. Για να γίνει αυτό στηρικτικά αποκλειστικά στον ορισμό της γλώσσας και πιο ειδικά στην έκδοση 3.3.2 (Οκτώβριος 2016).

Για να βεβαιωθώ ότι είναι σωστή η γραμματική 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. Αφού έλεγξα ότι είναι σωστά 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 από αυτά που εξέτασα παρέχονται από τον οργανισμό της R για τέτοιου είδους χρήση δηλαδή τον έλεγχο της γραμματικής.

*Υπήρχε αλλα ήταν είτε outdated είτε λάθος.

RFrontEnd Solution

RGRAMMAR PROJECT

RFRONTEND PROJECT

Μοτίβα που χρησιμοποιήθηκαν (1/3)

- Composite: περιγράφει **μια ομάδα από αντικείμενα τα οποία πρεπει να αντιμετωπίζονται με τον ίδιο τρόπο** ως ένα ενιαίο αντικείμενο. 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).
- Visitor: 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 **virtual method** μπορεί να γίνει **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 χρησιμοποιείτε για την προσπέλαση του AST αλλα και από το εργαλείο ANTLR.

*Abstract Syntax Tree = Αφηρημένο Συντακτικό Δένδρο

Μοτίβα που χρησιμοποιήθηκαν (2/3)

- Observer: χρησιμοποιείτε κυρίως για την **κατασκευή συστημάτων διαχείρισης γεγονότων** κάθε φορά που εισερχόμαστε ή εξερχόμαστε σε έναν κόμβο, ενεργοποιείτε ένα γεγονός. Ένα αντικείμενο διατηρεί μια λίστα από παρατηρητές και τους ενημερώνει αυτόματα με οποιαδήποτε αλλαγή κατάστασης συνήθως καλώντας μια από τις μεθόδους τους. Έτσι, μπορούμε να χρησιμοποιήσουμε τα γεγονότα αυτά για να πραγματοποιήσουμε οποιαδήποτε ενέργεια (δημιουργία των αρχείων τύπου DOT* και του πίνακα συμβόλων) σε οποιαδήποτε σημείο του AST.
- Iterator: χρησιμοποιείτε για να διασχίσουμε το περιεχόμενο ενός στοιχείου. Υπάρχουν αρκετές κλάσεις ώστε ο κώδικας μας να είναι επεκτάσιμος. Σε συνδυασμό με το μοτίβο του παρατηρητή, χρησιμοποιούμε και το μοτίβο του επαναλήπτη.

* Γλώσσα περιγραφής γράφων.

Μοτίβα που χρησιμοποιήθηκαν (3/3)

- Façade: 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. Μια τέτοια κλάση χειρίζεται την εκτέλεση όλου του προγράμματος.

Περιγραφή μεθόδου parse (1/2)

```
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);
```

Όλες οι λειτουργίες γίνονται μεσω της κλάσης RFacade η οποία είναι υπεύθυνη για το διάβασμα του αρχείου ή των αρχείων τύπου R που θα δώσει ο χρήστης σαν είσοδο στο πρόγραμμα. Η σημαντικότερη μέθοδος του project αλλά και όλου του προγράμματος είναι η **Parse**.

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 **λεκτικός αναλυτής της γραμματικής** και αυτός με την σειρά του παίρνει την είσοδο του από τον πίνακα του AntlrInputStream και εφαρμόζει ταιριάσματα 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).

Περιγραφή μεθόδου parse (2/2)

Μετά τον λεκτικό αναλυτή, καλείτε ο **συντακτικός αναλυτής του φίλτρου**, 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 και εφαρμόζονται οι κανόνες της γραμματικής του με αποτέλεσμα να δημιουργηθεί το Συντακτικό Δένδρο από το οποίο προκύπτει το Αφηρημένο Συντακτικό Δένδρο.

Τέλος, απο το Συντακτικό Δένδρο και το Αφηρημένο Συντακτικό Δένδρο **φτιάχνονται τα αρχεία τύπου DOT** και με την βοήθεια του εργαλείου Graphviz* μπορούμε να δημιουργήσουμε εικόνες τύπου gif με το κανονικό και το αφηρημένο συντακτικό δένδρο αλλά και για την δημιουργία του **πίνακα συμβόλων**.

Η μέθοδος επιστέφει τον αριθμό τον σφαλμάτων που υπάρχουν εάν δεν υπάρχουν επιστέφει 0 και επαναλαμβάνεται τόσες φορές όσες είναι και τα αρχεία.

* Εργαλείο παραγωγής γράφων.

```
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      }
```

Μετατροπή Συντακτικού Δένδρου (ΣΔ) σε Αφηρημένο Συντακτικό Δένδρο (ΑΣΔ) (1/2)

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 επισκέπτη 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 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 context του και τον τύπο του και έπειτα καλείτε η αντίστοιχη εικονική μέθοδος επίσκεψης του ANTLR. Αφού έχει επιστέψει από τις επισκέψεις των παιδιών γίνεται pop από την στοίβα με τους γονείς.

Για τους τερματικούς κόμβους δεν χρειάζεται κάποια ειδική μεταχείριση.

Μετατροπή Συντακτικού Δένδρου (ΣΔ) σε Αφηρημένο Συντακτικό Δένδρο (ΑΣΔ) (2/2)

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.

Αριστερά φαίνεται ενδεικτικός κώδικας της μεθόδου VisitExprFor της κλάσης ASTGenerator.

```
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;
}
```

Lexical scoping

Lexical scoping (1/2)

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. Αυτό σημαίνει ότι οι δεσμεύσεις μεταβλητών ισχύουν κατά την δημιουργία τους.

Τα σύμβολα αυτά μπορεί να είναι **bound** ή **unbound**. Bound σύμβολα είναι αυτά που υπάρχουν είτε στα ορίσματα της συνάρτησης ή μέσα στο σώμα της. Οποιοδήποτε άλλο σύμβολο υπάρχει στο σώμα της συνάρτησης είναι unbound.

Τοπικές μεταβλητές είναι αυτές που ορίζονται μέσα σε μια συνάρτηση.

Κατά την διάρκεια της αξιολόγησης εάν βρεθεί ένα **unbound** 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.

*Υπάρχει και ένα ακόμα επίπεδο πιο πάνω το empty περιβάλλον.

Lexical scoping (2/2)

Με το lexical 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.

Για το γέμισμα του πίνακα συμβόλων χρειάζεται ένα πέρασμα στο ΑΣΤ με την βοήθεια της κλάσης **ASTScopeVisitor**. Η κλάση **SymbolTable** είναι αυτή που διαχειρίζεται τα σύμβολα του προγράμματος της εισόδου δηλαδή τα αποθηκεύει μαζί με έναν χαρακτηρισμό για κάθε σύμβολο όπως τοπικό, παράμετρος, unbound (και που υπάρχει) και εάν έχουμε κλήση συναρτήσεων.

Αποτελέσματα

Αποτελέσματα

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 που δεν διέθετα.

Στις επόμενες διαφάνειες ακολουθεί παράδειγμα ενός προγράμματος με μια απλή for και αυτό γίνεται επειδή μεγαλύτερα προγράμματα δεν θα χωράνε, χαρακτηριστικά ένα πρόγραμμα με ~100 σειρές μπορεί να παράγει μια εικόνα ανάλυσης ~ 32.000 x 500 pixels.

Αποτελέσματα

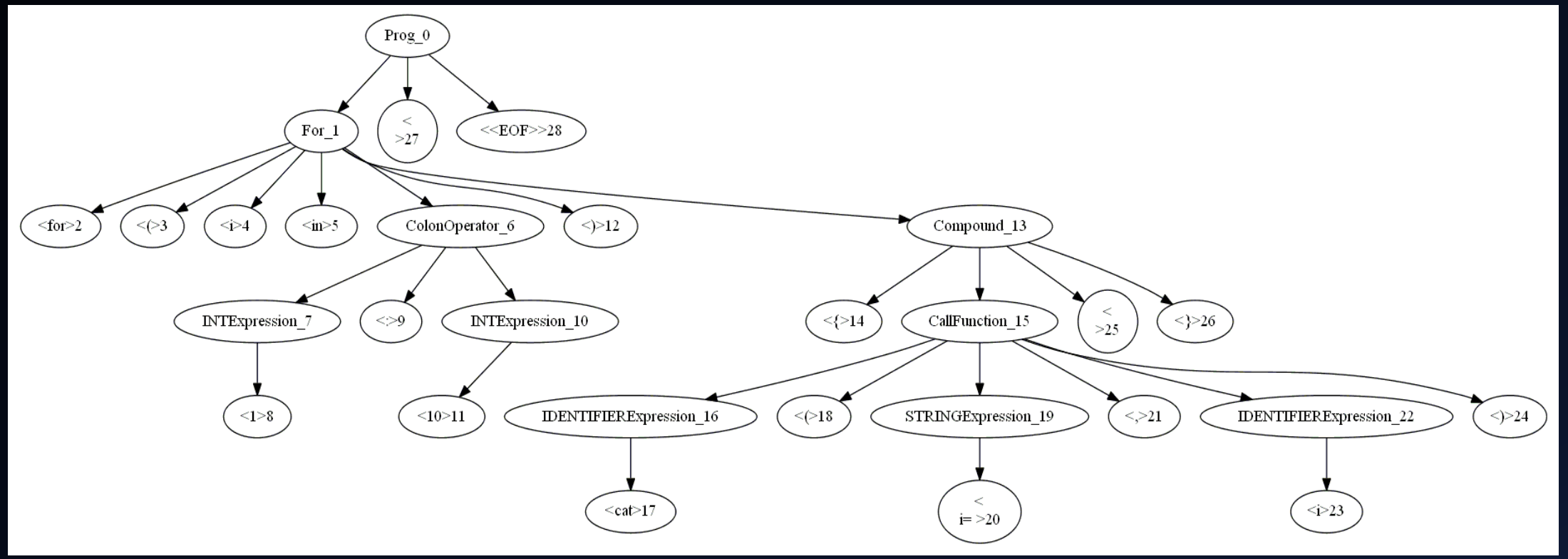
ΑΠΕΙΚΟΝΙΣΗ ΕΝΟΣ ΣΔ

ΑΠΕΙΚΟΝΙΣΗ ΕΝΟΣ ΑΣΔ

ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ

Απεικόνιση ενός ΣΔ

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

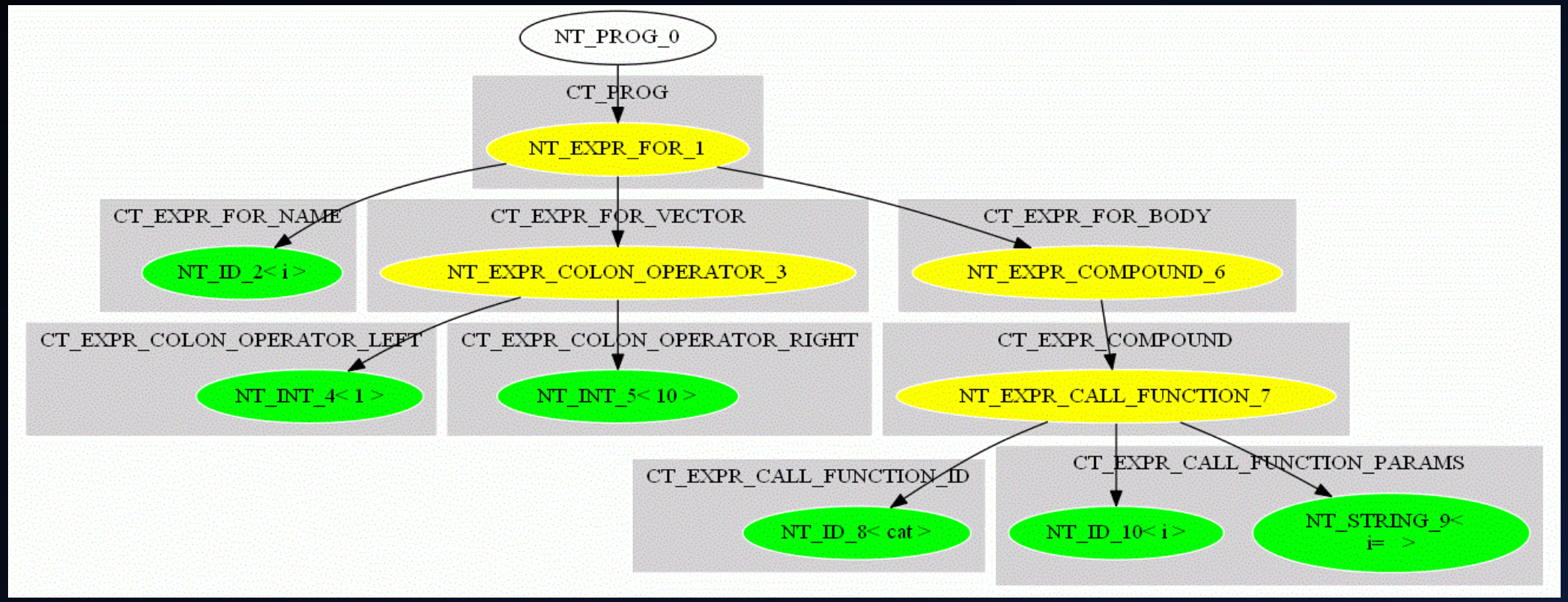


Αποτελέσματα

ΑΠΕΙΚΟΝΙΣΗ ΕΝΟΣ ΣΔ
ΑΠΕΙΚΟΝΙΣΗ ΕΝΟΣ ΑΣΔ
ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ

Απεικόνιση ενός ΑΣΔ

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



Αποτελέσματα

ΑΠΕΙΚΟΝΙΣΗ ΕΝΟΣ ΣΔ
ΑΠΕΙΚΟΝΙΣΗ ΕΝΟΣ ΑΣΔ
ΠΙΝΑΚΑΣ ΣΥΜΒΟΛΩΝ

Πίνακας συμβόλων

```r
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;
```

Κώδικας R

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

Συμπεράσματα

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 μου έδωσε την δυνατότητα να χρησιμοποιήσω μια αντικειμενοστραφή γλώσσα την C# και όχι την γλώσσα υψηλού επιπέδου C. 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. 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.

Μελλοντικές κατευθύνσεις

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. Έτσι θα έχει δημιουργηθεί ένας ολοκληρωμένος μεταγλωττιστής της R. Αξίζει να αναφερθεί ότι μόνο το RStudio παρέχει ένα ολοκληρωμένο IDE και πως το Visual Studio 15 (που βγήκε την προηγούμενη εβδομάδα) υποστηρίζει την γλώσσα R.

Μια πιο προχωρημένη ιδέα θα ήταν η υλοποίηση μιας γεννήτριας παραγωγής κώδικα, η οποία θα δημιούργει το περιεχόμενο των κλάσεων των μοτίβων που αναφέρθηκαν για την εκάστοτε γραμματικής αυτόματα.

Ερωτήσεις, απορίες, παρατηρήσεις …