Compilers 2

MiniLatexToLatex

Καραπιπεράκης Εμμανουήλ
A.M:2022201800075
dit18075@go.uop.gr

Διδάσκων: Δημητρουλάκος Γρηγόριος

Περιεχόμενα:

Εισαγωγή

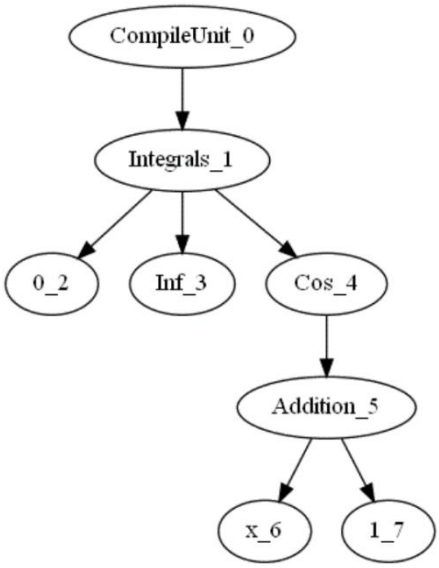
Αρχικά χρειάστηκε να πραγματοποιηθεί μια έρευνα σχετικά με το Latex πριν ξεκινήσει η κατασκευή του Μεταγλωττιστή **MiniLatexToLatex**.
Τι δυνατότητες έχει; Ποιο είναι το format ενός αρχείου .tex; Από τι εντολές μπορεί να αποτελείται ένα τέτοιο αρχείο;
Μελετώντας το documentation του latex αλλά και διάφορα συναφή άρθρα, κατέληξα πως για να μπορέσει ένα αρχείο .tex να γίνει compile πρέπει να έχει ένα **\documentclass** στο οποίο μπορώ να ορίσω και το μέγεθος της γραμματοσειράς του αρχείου που θα παραχθεί. Ακολούθως απαιτούνται οι κατάλληλες βιβλιοθήκες που περιέχουν τις μαθηματικές σχέσεις που θα χρησιμοποιήσω. Επιπλέον μπορώ να βάλω και άλλα χρήσιμα στοιχεία όπως ημερομηνία, author, τίτλο. Τέλος θα χρειαστώ 2 tags **\begin{document}** και **\end{document}**, όπου ανάμεσα σε αυτά τα στοιχεία θα μπουν οι εντολές που θα δίνει ο χρήστης στο αρχείο εισόδου.

Γραμματική

Για τη γραμματική μου επέλεξα ενδεικτικά ορισμένους κανόνες, όπως τριγωνομετρικές συναρτήσεις(\sin,\cos,\arctan), τετραγωνική ρίζα(\sqrt), \floor και \ceil,άθροισμα(\sum),όρια(\lim), ολοκληρώματα(\int), μερικά ελληνικά γράμματα τα οποία αποτελούν τερματικά στοιχεία της γραμματικής. Επίσης συμπεριέλαβα και πράξεις ανάμεσα στους κανόνες που προανέφερα, όπως πρόσθεση, πολλαπλασιασμός, ύψωση σε κάποια δύναμη κ.α
Έτσι ως είσοδο ο χρήστης θα δίνει διάφορες εντολές σε latex και θα δημιουργώ 2 ενδιάμεσες αναπαραστάσεις(συντακτικό και αφηρημένο συντακτικό δέντρο) και στη συνέχεια το τελικό .tex αρχείο.

Συντακτικό δέντρο

Αφού κατασκεύασα τη γραμματική, επόμενο βήμα ήταν η δημιουργία του συντακτικού δέντρου, έτσι ώστε να έχω μία ενδιάμεση αναπαράσταση για να είμαι σίγουρος πως δεν υπάρχουν προβλήματα στη γραμματική.
Έτσι δημιούργησα το αρχείο **STPrinterVisitor.cs** το οποίο κληρονομεί το GrammarBaseVisitor που παράγει το ANTLR και κατασκεύασα έναν κανόνα visit για κάθε Label που έχω ορίσει στο Grammar.g4. Σε κάθε διάσχιση, ενημερώνω κατάλληλα ένα αρχείο .dot το οποίο στη συνέχεια θα το εκτελέσω και θα δημιουργηθεί το .gif αρχείο, στο οποίο εικονίζεται το δέντρο που δημιουργήθηκε.
Για παράδειγμα εάν το input του χρήστη είναι `\int_{0}^{\infty} \cos(x+1)` τότε θα παραχθεί το παρακάτω δέντρο:
Όπως φαίνεται CompileUnit είναι το αρχικό σύμβολο της γραμματικής μας, το οποίο αποτελείται από μια ή περισσότερες εκφράσεις(expr+). Αποκάτω βρίσκεται ένα ολοκλήρωμα, όπου το 1ο παιδί είναι ο κάτω όρος, δηλαδή 0.Το 2ο παιδί είναι ο πάνω όρος δηλαδή το άπειρο και το 3ο παιδί συμβολίζει το σώμα του ολοκληρώματος. Στο συγκεκριμένο παράδειγμα πρόκειται για ένα συνημίτονο αθροίσματος μιας μεταβλητής(x) και ενός αριθμού(1). Αν εισαγάγω την ίδια εντολή σε ένα .tex αρχείο θα λάβω το παρακάτω αποτέλεσμα:



$$\int_0^{\infty} \cos(x+1)$$

Αφηρημένο Συντακτικό δέντρο

Ακολούθως κατασκεύασα το ΑΣΔ. Αρχικά δημιούργησα το αρχείο **ASTElement.cs**, το οποίο περιέχει στοιχεία απαραίτητα για τη δημιουργία και τη διάσχιση του δέντρου, όπως τον τύπο που μπορεί να έχει ένας κόμβος, μεθόδους AddChild και GetChild για την εισαγωγή ή ανάκτηση κάποιου παιδιού. Επίσης υπάρχουν και μεταβλητές που είναι κοινές για όλους τους κόμβους όπως m_nodetype και m_parent
Στο αρχείο **ASTBaseVisitor.cs** βρίσκονται οι μέθοδοι visit. Όπως φαίνεται είναι εικονικές(virtual) δηλαδή σε κάποιο άλλο αρχείο για παράδειγμα που κληρονομεί το ASTBaseVisitor μπορώ να τις κάνω override αλλάζοντας το περιεχόμενο τους. Επίσης ο τύπος αυτών των μεθόδων είναι Generic(T), δηλαδή μπορεί να γίνει int για παράδειγμα αν γίνει override σε κάποιο άλλο αρχείο. Τελευταίο σημείο αναφοράς είναι πως οι κόμβοι που είναι τερματικά στοιχεία της γραμματικής μας, δηλαδή φύλλα(leafs) επιστρέφουν default(T) αντί να πραγματοποιήσουν VisitChildren(node)

```
178         public virtual T VisitAssign(LAssign node)
179         {
180             return VisitChildren(node);
181         }
182
183         public virtual T VisitNumber(LNUMBER node)
184         {
185             return default(T);
186         }
187
188         public virtual T VisitVariable(LVARIABLE node)
189         {
190             return default(T);
191         }
192
```

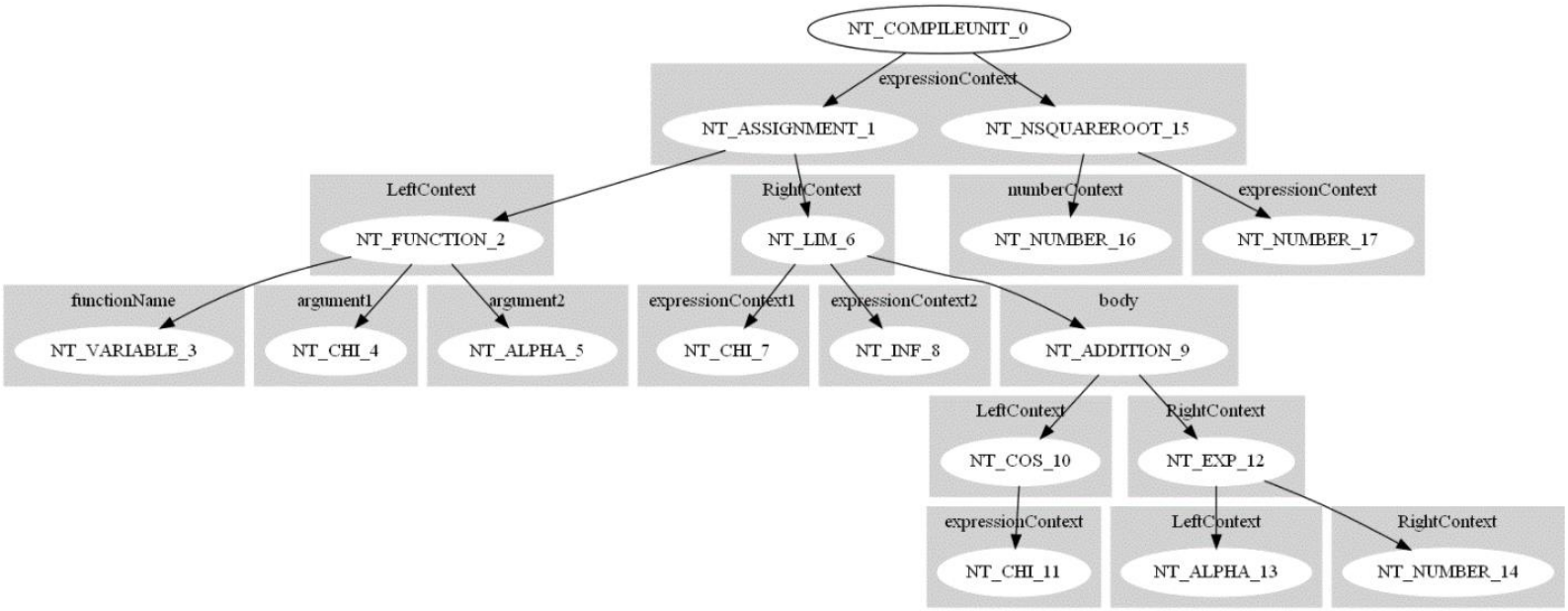
Στο αρχείο **LatexType.cs** βρίσκονται οι υποκλάσεις του ΑΣΔ, όπου υλοποιείται και η μέθοδος accept() για την οποία θα γίνει αναφορά στη συνέχεια, στα μοτίβα σχεδίασης που χρησιμοποιήθηκαν.
Στο **ASTGenerator.cs** δημιουργείται το δέντρο. Αξίζει να σημειωθεί πως για τους κανόνες της γραμματικής που μπορούσαν να υπάρχουν μια ή περισσότερες φορές έπρεπε να πραγματοποιήσουμε ένα for each, όπου πραγματοποιούσαμε Visit σε όλα τα παιδιά. Το for each αυτό πραγματοποιείται χάρη στο iterator pattern που έχει κατασκευαστεί στο ASTElement.cs
Τέλος στο αρχείο **ASTPrinterVisitor.cs** κάνουμε override τις μεθόδους Visit που προαναφέραμε και ενημερώνουμε κατάλληλα ένα .dot αρχείο το οποίο στη συνέχεια θα δημιουργήσει το .gif που περιέχει το δέντρο μας

Για input:

```
f(\chi,\alpha) = \lim_{\chi\to\infty} \cos(\chi) + \alpha^2

\sqrt[2]{5}
```

Το δέντρο που θα δημιουργηθεί θα είναι:



Όπως φαίνεται έχουμε 2 αρχικούς κανόνες, ένα assignment όπου σε μία συνάρτηση με όνομα μια μεταβλητή και ορίσματα χ(\chi) και α(\alpha) θα θέσουμε το όριο $\chi \to \infty$ με body το άθροισμα του cos(χ) και του α^2.
2ος κανόνας είναι απλά μια νιοστή τετραγωνική ρίζα ενός αριθμού.

Μοτίβα σχεδίασης

Συνολικά χρησιμοποιήσαμε 4 μοτίβα σχεδίασης.
Το 1ο είναι το **Composite Pattern** για το οποίο απαιτείται η δημιουργία μιας αφηρημένης γονικής κλάσης(`public abstract class ASTElement`) η οποία περιέχει έναν τύπο κόμβου, έναν σειριακό αριθμό, μια αναφορά στον parent και μια αναφορά στα παιδία καθώς και ένα αλφαριθμητικό που να αντιπροσωπεύει το όνομα του κόμβου.

```
public abstract class ASTElement : IEnumerable<ASTElement>
{

    private List<ASTElement>[] m_children = null;  //References to the children of the node
    private ASTElement m_parent;   //Reference to the parent of the node
    private int m_serial;  //Node's id
    private string m_name;  //Node's name
    private static int m_serialCounter = 0;   //Node's id (for the whole tree)
    private NodeType m_type;
```

Το 2ο μοτίβο που χρησιμοποιήσαμε είναι το **Visitor Pattern**(μοτίβο επισκέπτη), το οποίο περιλαμβάνει 4 βήματα υλοποίησης.

1. Δημιουργία κύριας αφηρημένης γονικής κλάσης του Visitor(`public abstract class ASTBaseVisitor<T>`) και σχεδίαση μιας μεθόδου για κάθε τύπο συντακτικού κόμβου με όνομα Visit.(ASTBaseVisitor.cs)
2. Για κάθε ξεχωριστό πέρασμα, δημιουργήθηκε μια υποκλάση της κύριας αφηρημένης κλάσης Visitor με τις ίδιες μεθόδους.(ASTPrinterVisitor.cs)
3. Εισαγωγή της αφηρημένης συνάρτησης Accept(). (ASTElement.cs)
4. Υλοποίηση των συναρτήσεων Accept στις υποκλάσεις του ΑΣΔ(LatexType.cs)

Όπου Accept(): είναι η μέθοδος που διαθέτουν όλα τα αντικείμενα του ΑΣΔ και η οποία καλεί την αντίστοιχη με τον τύπο του κόμβου Visit<NodeType>() μέθοδο.

Το 3ο μοτίβο που σχεδιάστηκε είναι το **Iterator pattern**(μοτίβο επαναλήπτη). Πρόκειται για τις λεπτομέρειες που κρύβονται από τον χρήστη και του παρέχουν τη foreach statement δομή χωρίς να χρειάζεται ο χρήστης να γνωρίζει τα ενδότερα της δομής. Δηλαδή δημιουργείται μόνο μια φορά από τον κατασκευαστή.
Έτσι στο αρχείο ASTElement.cs βρίσκονται οι μέθοδοι που υλοποιούν το συγκεκριμένο μοτίβο(MoveNext(), Reset())

4ο και τελευταίο μοτίβο που χρησιμοποιήθηκε στη γέννηση του αρχείου είναι ο **Construction Builder** ο οποίος είναι ένας συλλέκτης, δηλαδή μαζεύει πληροφορία και όταν συλλέξει όλη την πληροφορία που απαιτείται, τότε και μόνο τότε θα γεννήσει το αντικείμενο εξόδου.

Γέννηση αρχείου

Όπως αναφέρθηκε στην εισαγωγή, υπάρχουν ορισμένα δεδομένα που πρέπει να μπουν οπωσδήποτε για να μπορεί να γίνει η μεταγλώττιση του αρχείου.

```
public override CodeContainer VisitCompileUnit(LCompileUnit node)
{
    m_translatedFile = new CCFile(true);

    m_translatedFile.AddCode("\\documentclass[12pt]{article}\n", CCFile.mc_PREPROCESSOR);
    m_translatedFile.AddCode("\\usepackage{geometry}\n", CCFile.mc_PREPROCESSOR);
    m_translatedFile.AddCode("\\usepackage{graphicx}\n", CCFile.mc_PREPROCESSOR);
    m_translatedFile.AddCode("\\usepackage{amsmath}\n", CCFile.mc_PREPROCESSOR);
    m_translatedFile.AddCode("\\usepackage{esint}\n", CCFile.mc_PREPROCESSOR);
    m_translatedFile.AddCode("\\title{Compilers 2}\n", CCFile.mc_PREPROCESSOR);
```

Στη συνέχεια θα μπει το \begin{document} και \end{document} όπου ανάμεσα σε αυτά τα 2 tags θα μπει ο κώδικας.

```
public class CMainFunctionContainer : CFunctionContainer
{
    public CMainFunctionContainer(CComboContainer parent) : base(parent)
    {

    }

    public override CodeContainer AssemblyCodeContainer()
    {
        CodeContainer rep = new CodeContainer(CodeContainerType.CT_CODEREPOSITORY, this);
        rep.AddCode("\\begin{document}");
        rep.AddNewLine();
        rep.AddCode(AssemblyContext(mc_BODY));
        rep.AddCode("\\end{document}");
        return rep;
    }
}
```

Ο οποίος βέβαια πιο πριν έχει μερική ακόμα πληροφορία όπως το όνομα του author, το τρέχον date και τον τίτλο. Τέλος με χρήση σχολίων(%) στο .tex φαίνεται που αρχίζει ο κώδικας και που τελειώνει:

```
public override CodeContainer AssemblyCodeContainer()
{
    CodeContainer rep = new CodeContainer(CodeContainerType.CT_CODEREPOSITORY, MParent);
    rep.AddNewLine();
    rep.AddCode("\\author{Karapiperakis Emmanouil}");
    rep.AddNewLine();
    rep.AddCode("\\date{" + DateTime.Now.ToString("MM/dd/yyyy") + "}");
    rep.AddNewLine();
    rep.AddCode("\\maketitle");
    rep.AddNewLine();
    rep.AddCode("%*** CODE STARTS HERE ***");
    rep.AddNewLine();
    rep.AddCode(AssemblyContext(mc_COMPOUNDSTATEMENT_BODY));
    rep.AddNewLine();
    rep.AddCode("%*** CODE ENDS HERE ***");
    return rep;
}
```

Τέλος κάνοντας override τους visitors, διασχίζουμε το δέντρο και αποθηκεύουμε την απαραίτητη πληροφορία που μας ενδιαφέρει. Να σημειωθεί πως στην αρχή και στο τέλος της κάθε γραμμής προσθέτουμε τα σύμβολα “\[“ και “\]” αντίστοιχα προκειμένου να στοιχηθεί το αποτέλεσμά μας με κατάλληλο τρόπο στο αρχείο που θα δημιουργηθεί από την μεταγλώττιση του .tex.
Όμως η γέννηση του αρχείου δουλεύει μόνο για μερικούς κανόνες της γραμματικής και όχι για όλους.
πχ για το παρακάτω input το .tex που θα γεννηθεί φαίνεται δίπλα

input.txt

```
\sqrt{\sqrt{\sqrt{2}}}

\cos(\sqrt{x})

\cos(y)

\sqrt{\cos(\sqrt{2})}

\sqrt{5}

\sqrt[2]{5}

\sqrt[2]{\cos(\sqrt{3})}

\lfloor \arccos(\cos(x)) \rfloor

\lceil \sin(x) \rceil

\sqrt[5]{\arctan(\arcsin(\arccos(y)))}
```

input.tex

```
\documentclass[12pt]{article}
\usepackage{geometry}
\usepackage{graphicx}
\usepackage{amsmath}
\usepackage{esint}
\title{Compilers 2}
\begin{document}
\author{Karapiperakis Emmanouil}
\date{03/02/2022}
\maketitle
%*** CODE STARTS HERE ***
\[ \sqrt{\sqrt{\sqrt{2}}} \]
\[ \cos(\sqrt{x}) \]
\[ \cos(y) \]
\[ \sqrt{\cos(\sqrt{2})} \]
\[ \sqrt{5} \]
\[ \sqrt[2]{5} \]
\[ \sqrt[2]{\cos(\sqrt{3})} \]
\[ \lfloor \arccos(\cos(x)) \rfloor \]
\[ \lceil \sin(x) \rceil \]
\[ \sqrt[5]{\arctan(\arcsin(\arccos(y)))} \]
%*** CODE ENDS HERE ***
\end{document}
```

Το οποίο αν το κάνουμε compile θα λάβουμε το ακόλουθο αποτέλεσμα:

Compilers 2

Karapiperakis Emmanouil

03/02/2022

$$\sqrt{\sqrt{\sqrt{2}}}$$
$$\cos(\sqrt{x})$$
$$\cos(y)$$
$$\sqrt{\cos(\sqrt{2})}$$
$$\sqrt{5}$$
$$\sqrt[2]{5}$$
$$\sqrt[2]{\cos(\sqrt{3})}$$
$$\lfloor \arccos(\cos(x)) \rfloor$$
$$\lceil \sin(x) \rceil$$
$$\sqrt[5]{\arctan(\arcsin(\arccos(y)))}$$