**RELAZIONE PRETTY PRINTER**

**Nguimatio Stephanie – 1042739**

**Indice**

1. Scopo del progetto
2. Installazione dei tools necessari
   1. IDE – Eclipse
   2. Texlipse
   3. Pdf4Eclipse
   4. Miktex
   5. Configurazione
   6. GitHub Desktop
3. Progetto di patenza
4. Grammatica EBNF secondo lo standard ISO\_IEC\_14977\_1996(E)
5. Il traduttore
6. Gestione dei commenti
7. Risultato della traduzione

**1 Scopo del progetto**

Lo scopo del progetto è di realizzare un traduttore che dato un file di testo **.ebnf**, genera un file latex **.tex** strutturandolo in base al contenuto dei commenti presenti nel file di origine.

Il progetto è stato svolto in diverse fasi:

* Installazione dei vari tools necessari alla sua realizzazione
* Analisi e studio delle loro funzionalità
* Realizzazione del traduttore
* Realizzazione della traduzione di un esempio di file .ebnf

**2 Installazione dei tools necessari**

Per lo svolgimento del progetto è stata scaricato l’IDE Eclipse e successivamente sono stati integrati dei plug-in quali Texlipse, Pdf4Eclipse.

Prima di tutto è stata creata una cartella: **D:\Pretty\_Printer**

**2.1 Installazione IDE Eclipse**

Scaricare eclipse-dsl-neon-1a-win32 alla pagina:

<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/neon/1a/eclipse-dsl-neon-1a-win32.zip>

estrarla in **D:\Pretty\_Printer\Eclipse**

Avviare Eclipse e scegliere un percorso conveniente per l’area di lavoro: **D:\Pretty\_Printer\Workspace**

Questa versione di Eclipse ha già il plug-in Xtext integrato. È una versione a 32 Bit, funziona sicuramente su tutti i pc ma potrebbe essere leggermente meno performante della versione a 64 Bit. Quindi per macchine a 64 Bit è possibile usare la versione corrispondente scaricabile alla pagina :

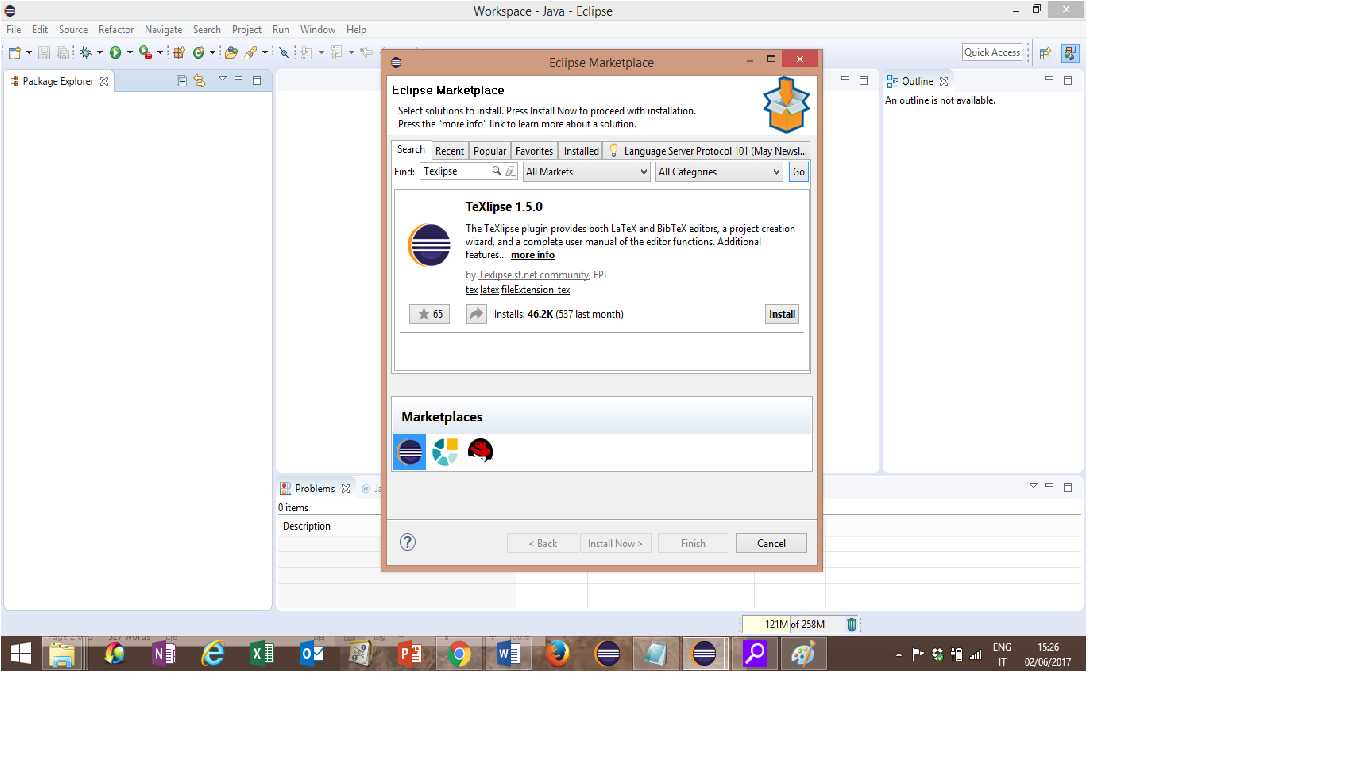
<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/neon/1a/eclipse-dsl-neon-1a-win32-x86_64.zip>

Altrimenti si può semplicemente scaricare eclipse e installare Xtext successivamente. Per installare Xtext  seguire le istruzione alla pagina : <https://eclipse.org/Xtext/download.html>

**2.2 TeXlipse**

Il plug-in Texlipse fornisce gli editori LaTeX e BibTeX, una procedura guidata di creazione di un progetto e un manuale utente completo delle funzionalità degli editori. Per installarlo :

* Selezionare Help -> Eclipse Marketplace
* Alla voce Search -> inserire TexLipse -> All Markets -> All categories -> Go

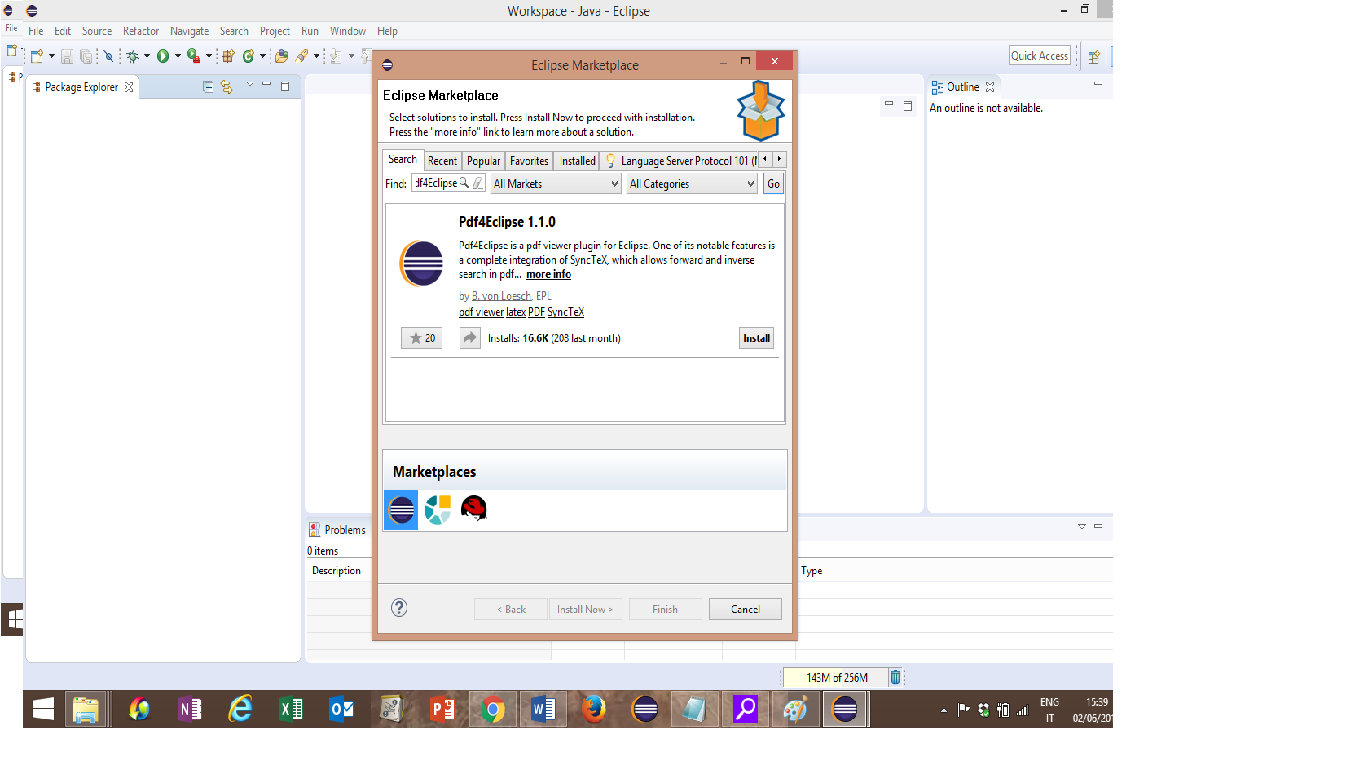


* Cliccare Install
* Finita l’installazione, Eclipse consiglia di riavviare il programma. Dopo il riavvio, Texlipse è pronto.

**2.3 Pdf4Eclipse**

Pdf4Eclipse è un plugin di visualizzazione pdf per Eclipse. Una delle sue caratteristiche notevoli è un'integrazione completa di SyncTeX, che consente la ricerca avanzata e inversa in file pdf. Quindi è perfettamente adatto per lavorare come visualizzatore per i documenti creati da TeXlipse.

Il processo di installazione è lo stesso di TeXlipse (sostituire Pdf4Eclipse alla parola TeXlipse).



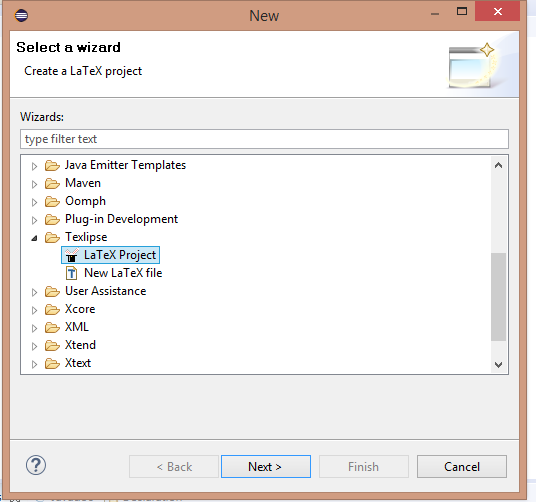
**2.4 MikTeX**

**MiKTeX** (pronuncia *mick-tech*) è una distribuzione di [TeX](https://it.wikipedia.org/wiki/TeX)/[LaTeX](https://it.wikipedia.org/wiki/LaTeX) per [Microsoft Windows](https://it.wikipedia.org/wiki/Microsoft_Windows) che è sviluppata da Christian Schenk. Contiene tutti i tools e i packages che servono a compilare i documenti LaTeX. MikTeX può aggiornarsi scaricando nuove versioni di componenti e pacchetti installati in precedenza e dispone di un processo di installazione facile. Inoltre, può chiedere agli utenti de desiderano scaricare I pacchetti che non sono ancora stati installati ma sono richiesti dal document corrente.

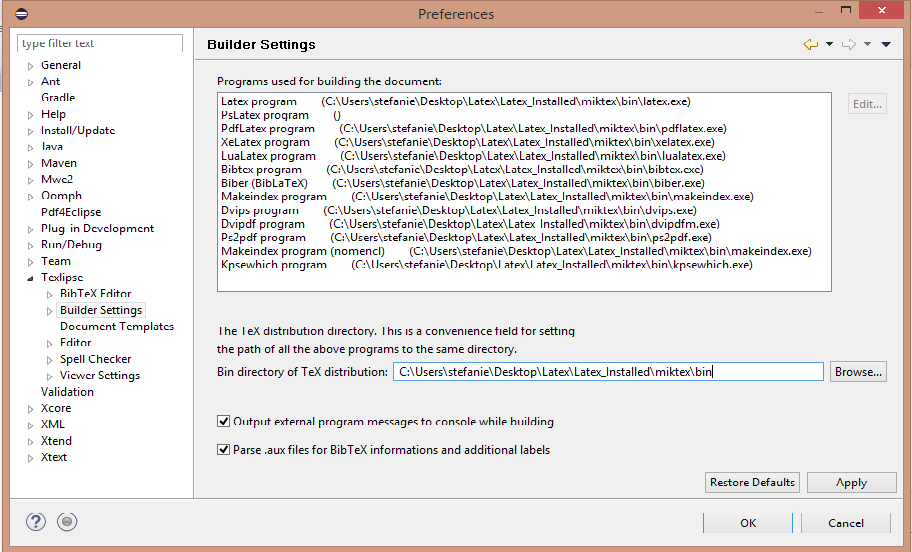
Scaricare MikTeX alla pagina : <https://miktex.org/download> . Estrarlo nella cartella del progetto : **D:\Pretty\_Printer\MikTeX.**

**2.5 Configurazione**

Dopo l’installazione di TeXlipse, è possibile aprire la prospettiva LaTeX in Eclipse.



Settare TeXlipse in modo che possa trovare i file eseguibili di LaTeX e quindi compilare i documenti LaTeX. Andare a Window –> Preferences –> TeXlipse –> Builder Settings, entrare il percorso agli eseguibili MikTeX - **D:\LaTeX\MikTeX\miktex\bin** – nella cartella Bin della distribuzione di TeX, e cliccare Apply. Questo dovrebbe riempire automaticamente tutti i programmi necessari nei programmi utilizzati per creare il documento.



**2.6 GitHub Desktop**

**GitHub** è un servizio di [hosting](https://it.wikipedia.org/wiki/Hosting) per progetti [software](https://it.wikipedia.org/wiki/Software). Il nome deriva dal fatto che GitHub è un [servizio sostitutivo del software](https://it.wikipedia.org/w/index.php?title=Servizio_sostitutivo_del_software&action=edit&redlink=1) dell'omonimo strumento di [controllo versione distribuito](https://it.wikipedia.org/wiki/Controllo_versione_distribuito), [Git](https://it.wikipedia.org/wiki/Git_(software)).

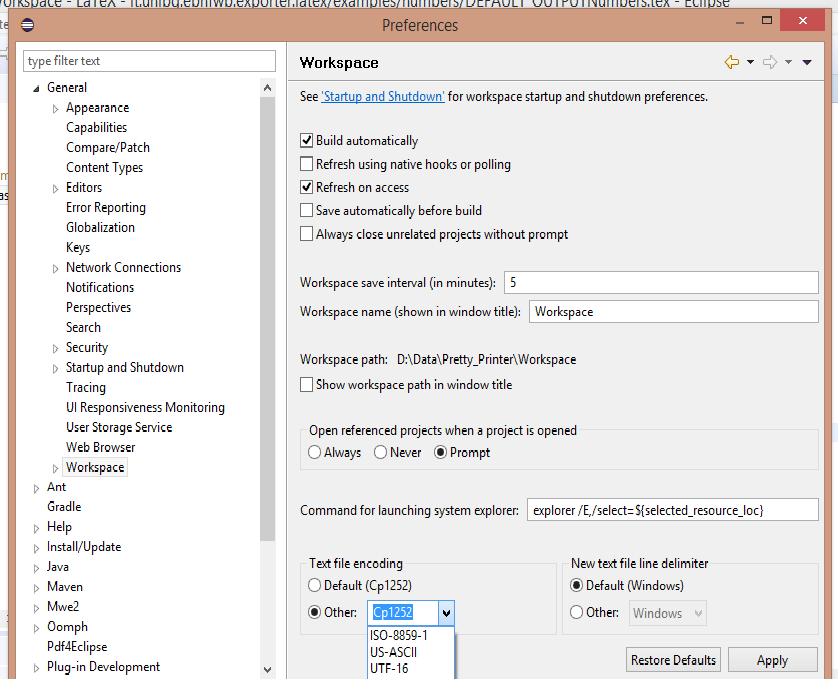
Il sito è utilizzato dagli sviluppatori open, che caricano il codice sorgente dei loro programmi e lo rendono scaricabile dagli utenti. Questi ultimi possono interagire con lo sviluppatore tramite un sistema di issue tracking, pull request e commenti che permette di migliorare il codice della repository risolvendo bug o aggiungendo funzionalità. Inoltre Github elabora dettagliate pagine che riassumono come gli sviluppatori lavorano sulle varie versioni dei repository.

GitHub Desktop per Windows è stato scaricato alla pagina <https://desktop.github.com/> .

Il progetto pretty printer fa parte di un progetto già iniziato caricato su github nel repository **UnibgInfo3A/ebnfworkbench**.

Dopo aver avuto accesso al repository, a partire da GitHub desktop, scaricarlo sulla propria macchina facendo il clone. In questo modo è possibile accedere ai progetti di interesse.

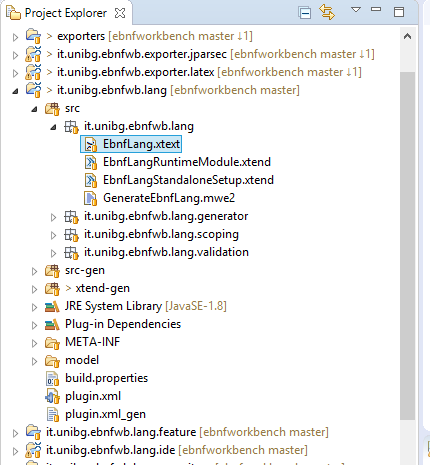
NOTA: La codifica di default del workspace va cambiata da cp1252 a utf-8 per una migliore rappresentazione dei caratteri. Per fare questo dal menu di Eclipse fare click su Window -> Preferences -> General -> Workspace nella casella Text file encoding -> Other selezionare UTF-8 da l’elenco delle cofiche e fare click su Apply poi su OK.



**3. Progetti di interesse**

I progetti utili sono **it.unibg.Ebnfwb.lang** e **it.unibg.Ebnfwb.exporter.latex** .

* **it.unibg.Ebnfwb.lang** contiene la grammatica per scrivere grammatiche ebnf.

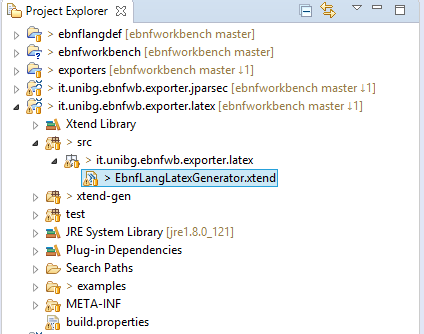


Il file **EbnfLang.xtext** contiene la grammatica ebnf ed è strutturato come segue :

1. grammar it.unibg.ebnfwb.lang.EbnfLang with org.eclipse.xtext.common.Terminals
3. generate ebnfLang "http://www.unibg.it/ebnfwb/lang/EbnfLang"
5. // Grammar definition of EBNF according to ISO/IEC 13250-6, see: http://www.iso.org/
7. // initial version from:
8. // https://xtexterience.wordpress.com/2011/05/13/an-ebnf-grammar-in-xtext/
10. //grammar org.eclipse.xtext.Ebnf hidden (WS, COMMENT)

13. **import** "http://www.eclipse.org/emf/2002/Ecore" as ecore
15. EbnfGrammar:
16. {EbnfGrammar} // Enforce object creation
17. lines+=Line\*;
19. //Line:{Line} EBNF\_COMMENT | ProductionRule;
20. Line: CommentEBNF | ProductionRule;
22. ProductionRule:{ProductionRule}
23. name=NAME '=' expr=Expression ';';
25. Expression:
26. Expression\_Alternative;
28. // The operator | defines alternatives. It has got the lowest priority and is left associative
29. Expression\_Alternative returns Expression:
30. Expression\_Concatenation ({Expression\_Alternative.elements+=current} ('|' elements+=Expression\_Concatenation)+)?;
32. // The operator , concatenates two or more symbols. It has got a higher priority than the alternative and is left associative, too
33. Expression\_Concatenation returns Expression:
34. Expression\_Exception ({Expression\_Concatenation.elements+=current} (',' elements+=Expression\_Exception)+)?;
36. // The operator - defines an exception and substracts a symbol from a set. It has got the highest priority of all operators
37. Expression\_Exception returns Expression:
38. Expression\_Repetition ({Expression\_Exception.left=current} '-' right=Expression\_Repetition)\*;
40. // The operator \* defines an exception and substracts a symbol from a set. It has got the highest priority of all operators
41. Expression\_Repetition returns Expression:
42. Expression\_Terminal |
43. ({Expression\_Repetition} repetitions=NUMBER '\*' expr=Expression\_Terminal);
45. // The parts of a rule can be
46. // - a reference pointing to another rule
47. // - a terminal symbol
48. // - a special sequence (non standardized)
49. // - a group (repetition, optional or simple group)
50. Expression\_Terminal returns Expression:
51. Expression\_Rule\_Reference |
52. Expression\_Terminal\_Symbol |
53. Expression\_Repetition\_Group |
54. Expression\_Optional\_Group |
55. Expression\_Group |
56. Expression\_Special\_Sequence;
58. // Reference another rule by its name
59. Expression\_Rule\_Reference returns Expression:
60. {Expression\_Rule\_Reference} rule=[ProductionRule|NAME];
62. // A special sequence
63. Expression\_Special\_Sequence returns Expression:
64. {Expression\_Special\_Sequence} text=SPECIAL\_SEQUENCE;
66. // A terminal symbol is a string which can be enclosed within ' or "
67. Expression\_Terminal\_Symbol returns Expression:
68. {Expression\_Terminal\_Symbol} text=TERMINAL\_SYMBOL;
70. // An expression enclosed within curly brackets means that this part can be repeated 0...n times. If the brackets are followed by an - the minimum number of repetitions is 1
71. Expression\_Repetition\_Group returns Expression:
72. {Expression\_Repetition\_Group} '{' expr=Expression '}' (atLeastOne?='-')?;
74. // An expression enclosed within brackets means that this part is optional
75. Expression\_Optional\_Group returns Expression:
76. {Expression\_Optional\_Group} '[' expr=Expression ']';
78. // Parts of an expression can be put in paranthesis to influence the parsing order
79. Expression\_Group returns Expression:
80. {Expression\_Group} '(' expr=Expression ')';
82. //Comments
83. CommentEBNF returns UserComment:
84. contentComment = EBNF\_COMMENT;
86. // Comments in EBNF are placed within (\* and \*)
87. terminal EBNF\_COMMENT returns ecore::EString:
88. '(\*' -> '\*)';
90. // A number >0
91. terminal NUMBER returns ecore::EInt:
92. '1'..'9' ('0'..'9')\*;
94. // A name of a rules (need to start and end with a non whitespace character). This differs from the original EBNF as no blank are allowed in order to simplify this grammar definition
95. terminal NAME returns ecore::EString:
96. ('a'..'z' | 'A'..'Z' | '\_')+;
98. // The contents of a terminal symbol, at least one character long
99. terminal TERMINAL\_SYMBOL returns ecore::EString:
100. ( '"'  (!'"')+ '"'  ) |
101. ( '\'' (!'\'')+ '\'' );
103. // A (non-standardized) special sequence, at least one character long
104. terminal SPECIAL\_SEQUENCE returns ecore::EString:
105. ( '?'  (!'?')+ '?'  );

* **it.unibg.Ebnfwb.exporter.latex** contiene il generatore che verrà modificato al fine di realizzare le funzionalità desiderate.



Il file **EbnfLangLatexGenerator.xtend** permette di gestire la trasformazione scritta in Xtend. Si ha a disposizione da un lato una risorsa (cioè la AST) e dall’altro l’accesso al file system per l’uscita (il file generato). Di seguito il codice iniziale del generatore :

1. package it.unibg.ebnfwb.exporter.latex
2. import org.eclipse.emf.ecore.resource.Resource
3. import org.eclipse.xtext.generator.AbstractGenerator
4. import org.eclipse.xtext.generator.IFileSystemAccess2
5. import org.eclipse.xtext.generator.IGeneratorContext
6. import org.eclipse.xtext.generator.IGenerator
7. import org.eclipse.xtext.generator.IFileSystemAccess
8. import it.unibg.ebnfwb.lang.ebnfLang.EbnfGrammar
9. import it.unibg.ebnfwb.lang.ebnfLang.ProductionRule
10. import org.w3c.dom.Comment
11. import it.unibg.ebnfwb.lang.ebnfLang.Expression
12. import java.io.File
13. import it.unibg.ebnfwb.lang.services.EbnfLangGrammarAccess.Expression\_AlternativeElements
14. import it.unibg.ebnfwb.lang.ebnfLang.Line
15. import it.unibg.ebnfwb.lang.services.EbnfLangGrammarAccess.ProductionRuleElements
16. import it.unibg.ebnfwb.lang.ebnfLang.Expression\_Alternative
17. import it.unibg.ebnfwb.lang.ebnfLang.Expression\_Concatenation
18. import it.unibg.ebnfwb.lang.ebnfLang.Expression\_Exception
19. import it.unibg.ebnfwb.lang.ebnfLang.Expression\_Repetition
20. import it.unibg.ebnfwb.lang.services.EbnfLangGrammarAccess.Expression\_Terminal\_SymbolElements
21. import it.unibg.ebnfwb.lang.ebnfLang.Expression\_Terminal\_Symbol
22. import it.unibg.ebnfwb.lang.services.\*
23. import it.unibg.ebnfwb.lang.ebnfLang.impl.Expression\_AlternativeImpl
24. import it.unibg.ebnfwb.lang.parser.antlr.EbnfLangParser
25. import it.unibg.ebnfwb.lang.parser.antlr.internal.InternalEbnfLangLexer
26. import it.unibg.ebnfwb.lang.parser.antlr.internal.InternalEbnfLangParser
27. import java.io.StringReader
28. import it.unibg.ebnfwb.lang.parser.antlr.EbnfLangAntlrTokenFileProvider
29. import org.antlr.runtime.ANTLRFileStream
30. import org.antlr.runtime.ANTLRInputStream
31. import org.antlr.runtime.TokenSource
32. import org.antlr.runtime.CommonTokenStream
34. //import it.unibg.ebnfwb.lang.EbnfLangValueConverterService
35. /\*\*
36. \* Generates latex from your model files on save.
37. \*
38. \* See https://www.eclipse.org/Xtext/documentation/303\_runtime\_concepts.html#code-generation
39. \*
40. \* http://tex.stackexchange.com/questions/24886/which-package-can-be-used-to-write-bnf-grammars
41. \*
42. \*/
43. class EbnfLangLatexGenerator implements IGenerator { //extends AbstractGenerator {
44. //  override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
45. //      fsa.generateFile('greetings.txt', 'People to greet: ' +
46. //          resource.allContents
47. //              .filter(typeof(Greeting))
48. //              .map[name]
49. //              .join(', '))
50. override doGenerate(Resource input, IFileSystemAccess fsa) {
51. for (e : input.allContents.toIterable.filter(EbnfGrammar)) {
52. fsa.generateFile(fileName.toFirstUpper + ".tex", e.compile)


56. }
57. }
59. /\*  Accedere ai commenti
60. \* e«if (\*\*\*\*\*(value) instanceof Comment) {
61. \*
62. }»\*/
63. def compile(EbnfGrammar e) '''
64. \documentclass{article}
65. \usepackage{syntax}

68. \begin{document}
70. «FOR f:e.lines»
71. «f.compile»
73. «ENDFOR»
74. \end{document}
75. '''
77. def compile(Line line) {
78. if (line instanceof ProductionRule){
79. (line as ProductionRule).compile
81. }
82. else{
83. println(line.toString())
85. }
87. }

90. def compile(ProductionRule rule) '''
91. \begin{grammar}
93. «rule.name»
94. «rule.expr.compile»

97. \end{grammar}
98. '''

101. def compile(Expression expr) '''
103. '''

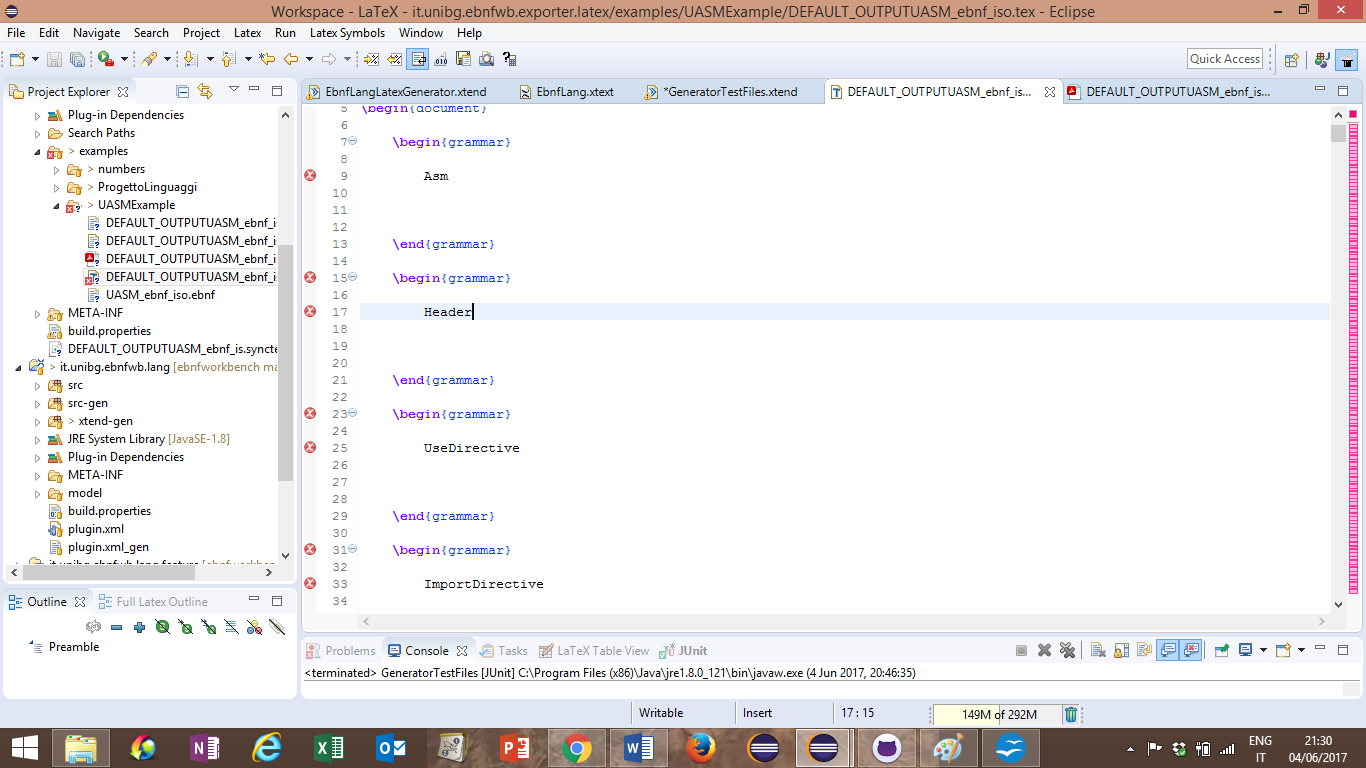
106. // the document name (for the tex file)
107. String fileName
109. def setFileName(String pn) {
110. fileName = pn;
111. }

114. }

* Per i test è stato usato Junit. I test vengono effetuati sul file di esempio **UASM\_ebnf\_iso.ebnf** nella cartella **examples/UASMExample**. Nella cartella **it.unibg.ebnfwb.exporter.latex.test.it.unibg.ebnfwb.exporter.latex** si trova il file di test **GeneratorTestFiles.xtend**. Di seguito il codice sorgente :

1. **package** it.unibg.ebnfwb.exporter.latex
3. **import** org.junit.Test
4. **import** org.junit.runner.RunWith
5. **import** org.eclipse.xtext.junit4.XtextRunner
6. **import** org.eclipse.xtext.junit4.InjectWith
7. **import** org.eclipse.xtext.generator.IGenerator
8. **import** com.google.inject.Inject
9. **import** org.eclipse.xtext.junit4.util.ParseHelper
10. **import** org.eclipse.xtext.generator.InMemoryFileSystemAccess
11. **import** **static** org.junit.Assert.\*
12. **import** org.eclipse.xtext.generator.IFileSystemAccess
13. **import** java.io.FileReader
14. **import** java.nio.CharBuffer
15. **import** java.io.FileWriter
16. **import** it.unibg.ebnfwb.lang.tests.EbnfLangInjectorProvider
17. **import** it.unibg.ebnfwb.lang.ebnfLang.EbnfGrammar
18. **import** java.io.FileInputStream
19. **import** java.nio.charset.Charset
20. **import** java.nio.channels.FileChannel
21. **import** java.nio.ByteBuffer
23. @RunWith(XtextRunner)
24. @InjectWith(EbnfLangInjectorProvider)
26. **class** GeneratorTestFiles {
28. @Inject EbnfLangLatexGenerator underTest
29. @Inject ParseHelper<EbnfGrammar> parseHelper
31. //    @Test
32. //    def void test() {
33. //      val fis = new FileInputStream('examples/numbers/numbers.ebnf');
34. //      val fc = fis.getChannel();
35. //      val bbuf = fc.map(FileChannel.MapMode.READ\_ONLY, 0, fc.size());
36. //      val cbuf = Charset.forName("8859\_1").newDecoder().decode(bbuf);
37. //      val EbnfGrammar grammar = parseHelper.parse(cbuf)
38. //        val InMemoryFileSystemAccess fsa = new InMemoryFileSystemAccess()
39. //        underTest.fileName = "numbers"
40. //        underTest.doGenerate(grammar.eResource, fsa)
41. //        fsa.textFiles.forEach[filename, fileContent|
42. //          println("writing " + filename + fileContent)
43. //          val file = new FileWriter('examples/numbers/'+filename)
44. //          file.write(fileContent.toString)
45. //          file.close
46. //        ]
47. //    }
49. @Test
50. def **void** test2() {
51. val pathfile = 'examples/UASMExample/'
52. val fileebnf = 'UASM\_ebnf\_iso'
53. val fis = **new** FileInputStream(pathfile+ fileebnf + '.ebnf');
54. val fc = fis.getChannel();
55. val bbuf = fc.map(FileChannel.MapMode.READ\_ONLY, 0, fc.size());
56. val cbuf = Charset.forName("8859\_1").newDecoder().decode(bbuf);
57. val EbnfGrammar grammar = parseHelper.parse(cbuf)
58. val InMemoryFileSystemAccess fsa = **new** InMemoryFileSystemAccess()
59. underTest.fileName = fileebnf
60. underTest.doGenerate(grammar.eResource, fsa)
61. fsa.textFiles.forEach[filename, fileContent|
62. println("writing " + filename + fileContent)
63. val file = **new** FileWriter(pathfile+filename)
64. file.write(fileContent.toString)
65. file.close
66. ]
67. }
69. }

Per eseguire il test fare click destro sul file GeneratorTestFiles.xtend -> Run As -> Junit test . Viene generato il file **DEFAULT\_OUTPUTUASM\_ebnf\_iso.tex** nella cartella **USAMExample** e il corrispondente file in pdf. A questo punto del progetto però, il file **.tex** generato contiene molti errori la cui risoluzione è anche l’obiettivo del progetto.



Gli errori sono dovuti a diversi motivi:

* Il metodo  compile (Expression expr) non è stato implementato.

1. def compile(Expression expr) '''
3. '''  .

* Nel metodo

1. def compile(EbnfGrammar e) '''
2. \documentclass{article}
3. \usepackage{syntax}

6. \begin{document}
8. «FOR f:e.lines»
9. «f.compile»
11. «ENDFOR»
12. \end{document}
13. '''

Il package syntax in latex è il package usato per scrivere grammatiche BNF. In effetti, il package syntax fornisce un certo numero di comandi e di ambienti che estende LaTeX e consente di comporre buone esposizioni di sintassi. Ad esempio fornisce l’ambiente Grammar per la composizione di grammatiche formali. All’interno di questo ambiente, le regole di produzione separate devono essere separate da linee vuote. E’ possibile utilizzare il normale comando \\ per eseguire la rottura di linea di una regola di produzione. Si noti che le regola di produzione deve iniziare con un nonterminale, nome racchiuso tra parentesi angolari (<…>), seguito da spazi vuoti, quindi di un operatore di produzione ( di solito ‘::=’) e poi di più spazi vuoti. Di seguito un esempio:

\begin{grammar}

<statement> ::= <ident> ‘=’ <expr>

| ‘for’ <ident> ‘=’ <expr> ‘to’ <expr> ‘do’ <statement>

| ‘{’ <stat-list> ‘}’

| <empty>

<stat-list> ::= <statement> ‘;’ <stat-list> | <statement>

\end{grammar}

Nel progetto in esame però si vuole traduttore grammatiche EBNF scritte secondo lo standard ISO\_IEC\_14977\_1996 in document LaTeX. Come si vedrà di seguito tali grammatiche hanno struttura diversa di quelle BNF e quindi il package syntax di LaTeX non è addatto per raggiungere l’obiettivo perseguito.

**4. Grammatica EBNF secondo lo standard ISO\_IEC\_14977**

Di seguito la struttura di una grammatica EBNF secondo lo standard ISO\_IEC\_14977.

**4.1 Struttura logica di EBNF**

* La sintassi di una regola è costituita da un meta-identificatore (il nome del simbolo non terminale definito) seguito da un simbolo di definizione seguito da un elenco di definizioni seguito da un simbolo terminatore.
* Un elenco di definizioni consiste in un elenco ordinato di una o più singole definizioni separate da un simbolo di separatore di definizione.
* Una singola definizione consiste in un elenco ordinato di uno o più termini sintattici separati l'uno dall'altro da un simbolo concatenato.
* Un termine sintattico è costituito da :

a) un fattore sintattico, o

b) un fattore sintattico seguito da un simbolo escluso seguito da un'eccezione sintattica.

* Un'eccezione sintattica consiste in un fattore sintattico soggetto alla restrizione che le sequenze di simboli rappresentate dall'eccezione sintattica possono ugualmente essere rappresentate da un fattore sintattico che non contiene meta-identificatori.
* Un fattore sintattico è composto da :

a) un numero intero seguito da un simbolo di ripetizione seguito da un sintattico-primario, oppure

b) un sintattico-primario

* Un numero intero è costituito da un elenco ordinato di una o più cifre decimali.
* Un sintattico-primario è costituito da uno dei seguenti :

a) una sequenza opzionale

b) una sequenza ripetuta

c) una sequenza raggruppata

d) un meta-identificatore

e) una stringa terminale

f) una sequenza speciale

g) una sequenza vuota

* Una sequenza opzionale è costituita da un simbolo di avvio-opzione seguito da un elenco di definizioni seguito da un simbolo di fine opzione.
* Una sequenza ripetuta consiste in un simbolo di ripetizione seguito da un elenco di definizioni seguito da un simbolo di ripetizione finale.
* Una sequenza raggruppata consiste in un simbolo di gruppo di partenza seguito da un elenco di definizioni seguito da un simbolo di gruppo finale.
* Un meta-identificatore consiste di un elenco ordinato di uno o più caratteri di identificazione soggetta, alla condizione che il primo meta-carattere di identificazione sia una lettera.
* Un meta-carattere di identificazione è un carattere o un numero decimale.
* Una stringa terminale è composta da :

a) un primo simbolo di quote seguito da una sequenza di uno o più caratteri terminale seguiti da un primo simbolo di quote, oppure

b) un secondo simbolo di quote, seguito da una sequenza di uno o più caratteri terminali secondari seguiti da un secondo simbolo di quote.

* Un primo carattere terminale è qualsiasi carattere terminale, ad eccezione di un primo simbolo di quote.
* Un secondo carattere terminale è qualsiasi carattere terminale, ad eccezione di un secondo simbolo di quote.
* Una sequenza speciale è costituita da un simbolo di sequenza speciale seguito da una sequenza (possibilmente vuota) di caratteri speciali di sequenza seguita da un simbolo di sequenza speciale.
* Un carattere di sequenza speciale è qualsiasi carattere terminale tranne un simbolo di sequenza speciale.
* Una sequenza vuota consiste nella sequenza vuota di caratteri terminali.

**4.2** **La rappresentazione di ogni carattere terminale In Extended BNF**

La rappresentazione di ciascun carattere terminale e del gap separatore

in Extended BNF usando i caratteri del 7 set di caratteri a 7 bit (ISO / IEC 646: 1991 International Reference Version) è definita di seguito:

1. **Caratteri e cifre**

Ogni lettera e cifra decimale sono rappresentati dal corrispondente carattere.

1. **Altri caratteri terminali**

La seguente tabella definisce la rappresentazione dei caratteri per ciascuno carattere terminale che non è né una lettera, né una cifra decimale

Né un altro carattere.

|  |  |
| --- | --- |
| **Metalanguag symbol** | **Normal representation** |
| Concatenate-symbol  Defining-symbol  Definition-separator-symbol  End-comment-symbol  End-group-symbol  End-option-symbol  End-repeat-symbol  Except-symbol  First-quote-symbol  Repetition-symbol  Second-quote-symbol  Special-sequence-symbol  Start-comment-symbol  Start-group-symbol  Start-option-symbol  Start-repeat-symbol  Terminator-symbol | , comma  = equals sign  | vertical line  \*) asterisk, right parenthesis  ) right parenthesis  ] right square bracket  } right curly bracket  - hyphen-minus  ‘ apostrophe  \* asterisk  " quotation mark  ? question mark  (\* left parenthesis, asterisk  ( left parenthesis  [ left square bracket  { left curly bracket  ; semicolon |

1. **Altri caratteri**

Sono qualsiasi altro carattere del set di caratteri ISO / IEC 646: 1991 che non è né:

- un carattere di controllo, nemmeno

- richiesto per rappresentare qualsiasi altro carattere terminale.

|  |  |
| --- | --- |
| space  . full stop  : colon  ! exclamation mark  + plus sign lowline  % percent sign  @ commercial at  & ampersand  # number sign | $ dollar sign  < less-than sign  > greater-than sign  / solidus  \ reverse solidus  ˆ circumflex accent  ` grave accent  ˜ tilde |

1. **Gap-separator**

Un separatore di gap è rappresentato come segue:

- uno spazio è rappresentato da un carattere spaziale,

- una tabella orizzontale è rappresentata da un Carattere di tabulazione orizzontale,

- una nuova linea è rappresentata da un (eventualmente vuoto) Sequenza di caratteri di ritorno a capo, un carattere line feed e una sequenza (eventualmente vuota) di carriage, caratteri di ritorno,

- una tabulazione verticale è rappresentata da un carattere di tabulazione verticale,

- un form feed è rappresentato da un carattere di form Feed.

Per maggiori dettagli vedere la documentazione dello standard ISO\_IEC\_14977 alla pagina :

https://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

1. **Esempio di grammatica EBNF**

Grammatica descritta nel file di esempio **UASM\_ebnf\_iso.ebnf**.

// UASM Structure.

Asm = ('asm' | 'asmmodule'), ID, Header, Body;

// Header

Header = {UseDirective}, {ImportDirective}, {ExportDirective};

// UseDirective

UseDirective = 'use', ID;

//ImportDirective

ImportDirective = 'import', ID, [ '(', ( IdDomain | IdFunction | IdRule ), {',', ( IdDomain | IdFunction | IdRule )}, ')' ];

//ExportDirective

ExportDirective = 'export', ID, ['(', ( IdDomain | IdFunction | IdRule ), {',', ( IdDomain | IdFunction | IdRule )}, ')' | '\*'];

// Body

Body = {Definition}, [('exec', IdRule)];

// Definition

Definition = TypeDefinition | FunctionDefinition | RuleDefinition;

// Transition  Rules  Deﬁnition

RuleDefinition = 'rule', ID, [ParameterDefinition], '=', Rule;

// Type of Rules

Rule = ParBlock

| SeqBlock

| SeqNext

| CaseRule

| ChooseRule

| ConditionalRule

| ExtendRule

| ForAllRule

| ImportRule

| IterateRule

| LetRule

| TurboReturnRule

| WhileRule

| UpdateRule

| SkipRule

| CallRule

| LocalRule

| PrintRule;

ParBlock = ('par', Rule, 'endpar') | ('{', Rule , '}');

SeqBlock = ('seq', Rule, 'endseq') | ('[', Rule, ']');

SeqNext = 'seq', Rule, ('next', Rule ), ['endseq'];

CaseRule = 'case', Term, 'of', ( Term, ':', Rule ), [('otherwise', Rule )], 'endcase';

ChooseRule = 'choose', VariableTerm , 'in', EnumTerm, {( VariableTerm, 'in', EnumTerm )}, [('with', Term )], 'do', Rule, [('ifnone', Rule )] , ['endchoose'];

ConditionalRule = 'if', Term, 'then', Rule, [('else', Rule )], 'endif';

SkipRule = 'skip';

UpdateRule = LocationTerm, ':=', Term;

ExtendRule = 'extend', ExtendableDomain , 'with', VariableTerm, {( VariableTerm )}, [('as', VariableTerm )], 'do', Rule, ['endextend'];

ForAllRule = 'forall', VariableTerm , 'in' , EnumTerm , {( VariableTerm , 'in', EnumTerm )}, [('with', Term )], 'do', Rule, [('ifnone', Rule )], ['endforall'];

ImportRule = 'import', VariableTerm ,'do' ,Rule, ['endimport'];

IterateRule = 'iterate', Rule, ['enditerate'];

LetRule = 'let', VariableTerm , '=', Term, {(',', VariableTerm, '=,', Term)}, 'in', Rule, ['endlet'];

TurboReturnRule = LocationTerm, ('<-' | '←'), CallRule;

WhileRule = 'while', Term, 'do', Rule, ['endwhile'];

CallRule = IdRule, [('(', Term , {(',' ,Term)}, ')' )];

LocalRule = 'local', ID, [ParameterDefinition], [(('->' | '→'), Domain)], [( 'initially', InitialFunctionDefinition )],{(',', ID, [ParameterDefinition], [(('->' | '→'), Domain)], [('initially', InitialFunctionDefinition)])} ,'in', Rule, ['endlocal'];

PrintRule = 'print', Term;

// Type Definition

TypeDefinition = DomainDefinition | EnumerateDefinition;

// Domains

Domain = StructuredDomain | BasicDomain | ExtendableDomain;

// Structured domains

StructuredDomain = 'SET', [('(Domain)')] | 'BAG', [('(Domain)')] | 'LIST', [('(Domain)')] | 'MAP', [('(Domain , Domain)')];

ExtendableDomain = 'ANY' | 'AGENT' | IdDomain;

BasicDomain = 'NUMBER' | 'INTEGER' | 'STRING' | 'CHAR' | 'BOOLEAN' | 'RULE';

// Enumerative Domain Definition

EnumerateDefinition = 'enum', ID, '=' ,'{', EnumTerm, {(',' , EnumTerm )}, '}';

// Domain definition

DomainDefinition = 'domain', ID, [('of', DomainParameterDefinition)], [(InitialDomainDefinition)];

DomainParameterDefinition = Domain | ParameterDefinition;

ParameterDefinition = '(', (ID, ('in' | '∈'), Domain | ID | Domain), {(',', (ID, ('in' | '∈'), Domain | ID | Domain))}, ')';

// Initial Domain Definition

InitialDomainDefinition = 'initially', '{', (TupleLiteral | Literal), {(',', (TupleLiteral | Literal))}, '}' ;

TupleLiteral = '(', Literal, (',', Literal), ')' ;

Literal = NumberLiteral

| BooleanLiteral

| KernelLiteral

| StringLiteral

| CharLiteral

| EnumTerm;

// Functions definition

FunctionDefinition = ControlledFunction

| StaticFunction

| DerivedFunction

| MonitoredFunction

| SharedFunction

| OutFunction;

// Controlled functions are read and written by the machine.

ControlledFunction = 'controlled' | ('function'| ('controlled', ('function'))), IdFunction, [ParameterDefinition], [('->' | ('→') , Domain)], [('initially', InitialFunctionDefinition)];

// Static functions do not change the value depending on the current state.

StaticFunction = 'static', ['function'], IdFunction, [ParameterDefinition], [('->' | ('→'), Domain)], ('always', InitialFunctionDefinition);

// Derived functions return value that depends on the current value of its inputs.

DerivedFunction = 'derived', ['function'], IdFunction, [ParameterDefinition], [('->' | ('→'), Domain)], ('=', InitialFunctionDefinition);

// Monitored functions are read by the machine and written by the environment.

MonitoredFunction = 'monitored', ['function'], IdFunction, [ParameterDefinition], [('->' | ('→'), Domain)];

// Shared functions are read and written by the machine and the environment.

SharedFunction = 'shared', ['function'], IdFunction, [ParameterDefinition], [('->' | ('→'), Domain)], ('initially', InitialFunctionDefinition);

// Out functions are read by the machine and written by the environment.

OutFunction = 'out', ['function'], IdFunction, [ParameterDefinition], [('->' | ('→'), Domain)], ('initially', InitialFunctionDefinition);

// Initial function definition

InitialFunctionDefinition = ['from'], Term;

// Terminals - Unicode   escapes

UnicodeInputCharacter = UnicodeEscape | RawInputCharacter;

UnicodeEscape = '\u', HexDigit, HexDigit, HexDigit, HexDigit ;

HexDigit = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|'a'|'b'|'c'|'d'|'e'|'f'|'A'|'B'|'C'|'D'|'E'|'F';

RawInputCharacter = ['Any Unicode Character'];

// Line   Terminator

LineTerminator = '\u000A' | '\u000D' | '\u000A\u000D';

// Whitespaces

WhiteSpace = '\u0020'

| '\u0009'

| '\u000C'

| LineTerminator;

InputCharacter = UnicodeInputCharacter, 'but not', WhiteSpace;

// Comments

Comment = LineComment | BlockComment;

LineComment = ('--' | '//'), {InputCharacter};

BlockComment = ('{-', {InputCharacter }, '-}' | '/\*', {InputCharacter}, '\*/');

// Identifiers

Identifier = Letter, {( Letter | CurrencySymbol | PunctuationCharacter | Digit | NumericLetter | CombiningMark | NonSpacingMark | '′')};

// Escape Sequences

EscapeSequence = '\b', ('backspace BS, Unicode \u0008')

| '\t', ('horizontal tab HT, Unicode \u0009')

| '\n', ('linefeed LF, Unicode \u000a')

| '\f', ('form  feed FF, Unicode \u000c')

| '\r', ('carriage return CR, Unicode \u000d')

| "\'", ('double quote "", Unicode \u0022')

| '\’', ('single  quote  ’,  Unicode  \u0027')

| '\\', ('backslash \, Unicode \u005c');

// Terms

Term = NumberTerm

| StringTerm

| BooleanTerm

| EnumTerm

| UndefLiteral

| SelfLiteral

| RuleAsTerm

| ValueTerm

| StructureTerm;

//Exp

Exp = Term | Formula | '(', Exp, ')';

// VarTerms, FunctionTerm, VariableTerm, LocationTerm

VarTerm = VariableTerm | LocationTerm;

VariableTerm = Identifier;

LocationTerm = FunctionTerm | 'result';

FunctionTerm = (Identifier, [( '(', Exp, ')' | TupleTerm )] );

// NumberTerm

NumberTerm = NumberLiteral | MathOperationTerm;

MathOperationTerm = MathOperand, ('+'| '-' | '\*' | 'div' | '/' | 'mod' | '^'), MathOperand;

MathOperand = NumberTerm | VarTermN;

// StringTerm

StringTerm = [StringLiteral | Exp, '+', Exp];

// BooleanTerm

BooleanTerm = BooleanLiteral | RelationalTerm | MemberofTerm | SubsetTerm;

// RelationalTerm

RelationalTerm = RelationOperand | RelationOperator | RelationOperand;

RelationOperand = NumberTerm | StringTerm | VarTerm;

RelationOperator = '<'

| '>'

| '<='

| '≤ (U+2264)'

| '>='

| '≥ (U+2265)';

// MemberofTerm

MemberofTerm = Exp, MembershipOperator, ( CollectionTerm | VarTermC );

MembershipOperator = 'memberof' | '∈ (U+2208)' | '∉ (U+2209)' ;

// SubsetofTerm

SubsetTerm = ( CollectionTerm | VarTermC ), 'subsetof', ( CollectionTerm | VarTermC);

SubsetOperator = 'subsetof' | '⊂ (U+2282)' | 'subseteqof' | '⊆ (U+2286)';

// EnumTerm

EnumTerm = Identifier;

//RulesAsTerm

RuleAsTerm = '@', Identifier;

// Value Terms

ValueTerm = VarTerm

| LetTerm

| ReturnTerm

| ConditionalTerm

| CaseTerm

| PickTerm

| CardinalityTerm;

// Let-Term

LetTerm = 'let', VariableTerm, '=', Exp, {(',', VariableTerm, '=', Exp )}, 'in', Exp;

// Return Term

ReturnTerm = 'return', Exp, 'in', Rule;

// Conditional Term

ConditionalTerm = Formula, '?', Exp, ':', Exp, 'if', Formula, 'then', Exp, 'else', Exp;

// Case Term

CaseTerm = 'case', Exp, 'of', ( Exp, ':', Exp, ';' ), [( 'otherwise', Exp )], 'endcase';

// Pick Term

PickTerm = 'pick', VariableTerm, 'in', ( CollectionTerm | VarTerm ), [( 'with', Formula )];

// Cardinality

CardinalityTerm = '|', CollectionTerm, '|' ;

// Structured Terms

StructureTerm = TupleTerm

| CollectionTerm;

// Tuples

TupleTerm = TupleDefTerm

| GetComponentTerm

| SetComponentTerm;

TupleDefTerm = '(', Exp, ',', Exp, {( ',', Exp )}, ')';

GetComponentTerm = 'get', ExpT, 'at', ExpN;

SetComponentTerm = 'set', ExpT, 'at', ExpN, 'to', Exp;

// Collections

CollectionTerm = Domain

| SetTerm

| BagTerm

| ListTerm

| MapTerm

| PowersetTerm

| ComprehensionTerm;

SetTerm = '{', Exp, {(',', Exp )}, '}' | '{','}';

BagTerm = '<', Exp, {(',', Exp )}, '>' | '<','>';

ListTerm = '[', Exp, {(',', Exp )}, ']' | '[',']';

MapTerm = '{', Exp, RightArrow, Exp, {(',', Exp, RightArrow, Exp )}, '}' | '{', RightArrow, '}';

RightArrow = '->' | '→ (U+2192)';

// Powerset

PowersetTerm = 'powerset(', CollectionTerm, ')';

// Comprehension   Terms

ComprehensionTerm = SetComprehensionTerm

| ListComprehensionTerm

| MapComprehensionTerm

| BagComprehensionTerm

| NumberRangeTerm;

SetComprehensionTerm = '{', Exp, '|', VariableTerm, 'in', CollectionTerm, {(',', VariableTerm, 'in', CollectionTerm )}, [('with', Formula )], '}';

ListComprehensionTerm = '[', Exp, '|', VariableTerm, 'in', CollectionTerm, {(',', VariableTerm, 'in', CollectionTerm )}, [('with', Formula )], ']';

BagComprehensionTerm = '<', Exp, '|', VariableTerm, 'in', CollectionTerm, {(',', VariableTerm, 'in', CollectionTerm )}, [('with', Formula )], '>';

MapComprehensionTerm = '{', Exp, '->', Exp, '|', VariableTerm, 'in', CollectionTerm, {(',', VariableTerm, 'in', CollectionTerm )}, [('with', Formula )], '}';

NumberRangeTerm = '[', ExpN, '..', ExpN, [('step', ExpN )], ']'

| '{', ExpN, '..', ExpN, [( 'step', ExpN )], '}';

//Collection Operations

CollectionOperationTerm = CollectionTerm, CollectionOperator, CollectionTerm;

CollectionOperator = 'union'

| '∪ (U+222A)'

| 'intersect'

| '∩ (U+2229)'

| 'difference'

| '\ (U+2216)';

// Formulas

Formula = Equation

| Negation

| LogicFormula

| QuantifierFormula

| BooleanTerm

| VarTermB;

// Equality  is  deﬁned  for  every  expression:

Equation = ( Exp, EqualityOperator, Exp );

EqualityOperator = '!=' | '=  (U+2260)' | '=';

// Inequality  x  !=  y  is  an  abbreviation  for  not  (x  =  y).

// Negation  and  logical  operators  are  only  deﬁned  for  formulas.

Negation = ('not' | '¬ (U+00AC)'), Formula;

LogicFormula = Formula, LogicalOperator, Formula;

LogicalOperator = 'and'

| '∧ (U+2227)'

| 'or'

| '∨  (U+2228)'

| 'xor'

| '  (U+22BB)'

| 'implies'

| '→ (U+2192)'

| '⇒  (U+21D2)'

| 'iff'

| '↔  (U+2194)'

| '⇔ (U+21D4)';

// Quantiﬁers  are  also  formulas.  These  deﬁnitions  have  to  be  distinguished  from

// the  corresponding  rule  deﬁnitions.  While  the  rule  deﬁnitions  perform  rules

// on  found  values,  the  terms  just  return  true  or  false.

QuantifierFormula = ForAllFormula | ExistsFormula;

ForAllFormula = 'forall', VariableTerm, 'in', CollectionTerm, {(',', VariableTerm, 'in', CollectionTerm )}, 'hold', Formula;

ExistsFormula = 'exists', ['unique'], VariableTerm, 'in', CollectionTerm,{(',', VariableTerm, 'in', CollectionTerm )}, 'with', Formula;

LowerCaseLetter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u'

| 'v' | 'w' | 'x' | 'y'| 'z';

UpperCaseLetter = 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U'

| 'V' | 'W' | 'X' | 'Y' | 'Z';

DecDigit = '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9';

ID = [[LowerCaseLetter],[UpperCaseLetter]],{[LowerCaseLetter],[UpperCaseLetter],[DecDigit]};

IdRule = ID;

IdFunction = ID;

IdDomain = ID;

BooleanLiteral = 'true' | 'false';

KernelLiteral = UndefLiteral | SelfLiteral;

StringLiteral = '"', {'^' | '"'}, '"';

CharLiteral = "'", {'^' | "'"}, "'";

UndefLiteral = 'undef';

SelfLiteral = 'self';

NumberLiteral = {[DecDigit]}, ['.',{[DecDigit]}], [('E'|'e'),( '+' | '-' ),{[DecDigit]}];

Quindi per risolvere il problema relativo al package syntax (accenato al punto 3), è stato usato il package **listings** e l’ambiente **lstlisting**.

**5. IL TRADUTTORE**

La generazione del file latex a partire dal file **.ebnf** è stata realizzata modicando il file sorgente del generatore **EbnfLangLatexGenerator.xtend** nel package **it.unibg.ebnfwb.exporter.latex**. L’operazione fondamentale è stata quella di implementare il metodo compile facendo uso del polimorfismo, in quanto la versione del metodo da eseguire viene scelta sulla base del tipo di oggetto effettivamente contenuto in una variabile *a runtime*.

* Compile(EbnfGrammar e)

In questo metodo si definisce la struttura del documento latex e per ogni riga si esegue il metodo compile.

1. def compile(EbnfGrammar e) '''
2. \documentclass{article}
3. \usepackage{color}
4. \definecolor{isabelline}{rgb}{0.96, 0.94, 0.93}
5. \pagecolor{isabelline}
6. \usepackage{listings}
7. \usepackage{textcomp}   // *per l’introduzione degli apici*
8. \usepackage{upquote}
9. \usepackage[a4paper,top=1cm,bottom=1cm,left=0.5cm,right=0.2cm]{geometry}
10. \lstset{
11. basicstyle=\ttfamily,
12. columns=flexible,
13. extendedchars= **true**
14. }

17. \begin{document}
18. «FOR f:e.lines»
19. «f.compile»
20. «ENDFOR»
21. \end{document}
22. '''

* Compile(Line line)

**def** compile(Line line) {

**if** (line **instanceof** ProductionRule)

*println*(line **as** ProductionRule).compile

**else** **if** (line **instanceof** UserComment)

*println*(line **as** UserComment).compile

**else**

*println*(line.toString)

}

* Compile(ProductionRule rule)

**def** compile(ProductionRule rule) {

**var** r = rule.expr.compile

**var** s = ""

**var** sf = ""

**var** SpecialCharMarnager mngr = **new** SpecialCharMarnager(r)

s = rule.name+ '='+ r + ';'

Traduzione in latex di alcuni caratteri speciali la cui rappresentazione unicode è stata specicata nel file **.ebnf**.

s = s.replace("'âˆª (U+222A)'","$\\cap$");

s = s.replace("'âˆ© (U+2229)'","$\\cup$");

s = s.replace("'\\ (U+2216)'","$\\setminus$");

s = s.replace("'=Â Â (U+2260)'","$\\neq$");

s = s.replace("'Â¬ (U+00AC)'","$\\lnot$");

s = s.replace("'âˆ§ (U+2227)'","$\\wedge$");

s = s.replace("'âˆ¨Â Â (U+2228)'","$\\vee$");

s = s.replace("'Â Â (U+22BB)'","$\\oplus$");

s = s.replace("'â†’ (U+2192)'","$\\rightarrow$");

s = s.replace("'â†’ (U+2192)'", "$\\rightarrow$");

s = s.replace("'â‡’Â Â (U+21D2)'","$\\Rightarrow$");

s = s.replace("'â†”Â Â (U+2194)'","$\\leftrightarrow$");

s = s.replace("'â‡” (U+21D4)'","$\\Leftrightarrow$");

s = s.replace("'â‰¤Â (U+2264)'","$\\leq$");

s = s.replace("'â‰¥Â (U+2265)'","$\\geq$");

s = s.replace("'â'", "$\\in$");

s = s.replace("'âˆ'", "$\\notin$");

s = s.replace("'âŠ‚ (U+2282)'","$\\subset$");

s = s.replace("'âŠ† (U+2286)'","$\\subseteq$");

s = s.replace("'âˆˆ (U+2208)'","$\\in$")

s = s.replace("'âˆ‰ (U+2209)'","$\\notin$")

s = s.replace("'â²'", "'′'")

//evita di spezzare una regola dopo il segno di uguaglianza

sf = s.replaceAll("[\r\n]+", "")

Introduzione di elementi per una migliore presentazione del testo:

* + L’ambiente **flushleft**; sposta il testo da destra a sinistra.
  + Opzione **mathescape= true** per la rappresentazione di simboli matematici,
  + Opzione **breaklines=true**, andare a capo.

'''

\begin{flushleft}

\begin{lstlisting}[mathescape=true, breaklines=true]

«sf»

\end{lstlisting}

\end{flushleft}

'''

}

* Compile(Expression expr)

Tutte le regole di produzione all’eccezione di Line, ProductionRule, Comment sono delle espressioni, quindi ereditano l’oggetto Expression. Ogni espressione viene implementata diversamente. Per cui il metodo compile(Expression expr) avrà la seguente struttura:

**def** compile (Expression expr) {

**if** (expr **instanceof** Expression\_Alternative)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Concatenation)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Exception)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Group)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Optional\_Group)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Repetition)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Repetition\_Group)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Rule\_Reference)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Special\_Sequence)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Terminal\_Symbol)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

**if** (expr **instanceof** Expression\_Terminal\_Symbol)

**return** '''

«**new** ToString().doSwitch(expr)»

'''

}

Inoltre per accedere al contenuto del file **.ebnf**, è stata creata la classe **ToString :**

1. **public** ToString() {
2. // TODO Auto-generated constructor stub
3. }

6. @Override
7. **public** String caseExpression\_Alternative(Expression\_Alternative object) {
8. String s="";
10. **for** (**int** i=0; i<object.getElements().size()-1; i++)
11. s+=**new** ToString().doSwitch(object.getElements().get(i)) + "|";
12. s+= **new** ToString().doSwitch(object.getElements().get(object.getElements().size()-1)).toString();
14. **return** s;
15. }

18. @Override
19. **public** String caseExpression\_Concatenation(Expression\_Concatenation object) {
20. String s="";
21. **for** (**int** i=0; i<object.getElements().size()-1; i++)
23. s+=**new** ToString().doSwitch(object.getElements().get(i)) + ",";
24. s+= **new** ToString().doSwitch(object.getElements().get(object.getElements().size()-1)).toString();
26. **return** s;
27. }
29. @Override
30. **public** String caseExpression\_Exception(Expression\_Exception object){
31. String s = "";
32. s+=**new** ToString().doSwitch(object.getLeft()) + "-" ;
33. s+= **new** ToString().doSwitch(object.getRight());
34. **return** s;
35. }
37. @Override
38. **public** String caseExpression\_Group(Expression\_Group object){
39. String s = "";
40. s = "(" + **new** ToString().doSwitch(object.getExpr())+ ")";
42. **return** s;
43. }
45. @Override
46. **public** String caseExpression\_Optional\_Group(Expression\_Optional\_Group object){
47. String s = "";
49. s = "["+ **new** ToString().doSwitch(object.getExpr())+ "]";
50. **return** s;
51. }
53. @Override
54. **public** String caseExpression\_Repetition(Expression\_Repetition object){
55. String s = "";
56. s= object.getRepetitions()+"\*"+**new** ToString().doSwitch(object.getExpr()) ;
58. **return** s;
59. }
61. @Override
62. **public** String caseExpression\_Repetition\_Group(Expression\_Repetition\_Group object){
63. String s = "";
65. s = "{" +  **new** ToString().doSwitch(object.getExpr())+ "}";
66. **return** s;
67. }
69. @Override
70. **public** String caseExpression\_Rule\_Reference(Expression\_Rule\_Reference object){
71. String s = "";
72. s= ""+object.getRule().getName()+"";
74. **return** s;
75. }
77. @Override
78. **public** String caseExpression\_Special\_Sequence(Expression\_Special\_Sequence object){
79. String s = "";
81. s = object.getText().toString();
82. **return** s;
83. }
85. @Override
86. **public** String caseExpression\_Terminal\_Symbol(Expression\_Terminal\_Symbol object){
87. String s = "";
88. s = object.getText().toString();
89. **return** s;
90. }
92. @Override
93. **public** String caseLine(Line object) {
94. String s = "";
95. s = object.toString();
96. **return** s;
97. }

100. }

**6. Gestione dei Commenti**

La grammatica EBNF descritta al punto 4 nel file **EbnfLang.xtext** non consente di gestire i commenti. In effetti SL\_COMMENT (/\*testocommento\*/) e ML\_COMMENT(testocommento) non si vedono perché sono terminali di Xtext e sono nascosti. Si è supposto che se l’utente scrive un commento usando // oppure /\* \*/ vuol dire che non lo vuole stampare in latex, se vuole invece stampare un commento in latex deve usare i simboli (\* \*).

Nella grammatica è già definito l’elemento EBNF\_COMMENT. Per far sic he vengano letti il file **EbnfLang.xtext** nel seguente modo:

* Sono state aggiunte le seguenti righe:

CommentEBNF returns UserComment:

contentComment = EBNF\_COMMENT;

* È stata modificata Line:

Line: CommentEBNF | ProductionRule;

* Ricompilato il file **EbnfLang.xtext**:

Tasto destro sul file **EbnfLang.xtext** -> Run as ->Generate Xtext Artifacts

* Poi il file **EbnfLangLatexGenerator.xtend** è stato modificato nel seguente modo:

1. def compile(Line line) {
2. **if** (line **instanceof** ProductionRule)
3. println(line as ProductionRule).compile
4. **else** **if** (line **instanceof** UserComment)
5. println(line as UserComment).compile
6. **else**
7. println(line.toString)
8. }
9. e inoltre aggiungi il metodo
10. ​​def compile (UserComment comment){
11. '''​«comment.contentComment.substring(2,comment.contentComment.length-2)»'''
12. ​ //Per non far stampare i simboli (\* \*) in latex​
13. }

Inoltre scrivendo i commenti, sono stati sostituiti tutti i simboli “speciali” di latex :

{, }, &, %, $, #, \_, ^, **⁓** .

Essi devono essere preceduti dal simbolo \ (esempio : \$ per stampare $, \\_ per stampare \_, etc…). Per fare questo è stata Creata la classe **SpecialCharManager.java** e successivamente usata all’interno del metodo def compile(UserComment comment) per sostituire i simboli speciali.

1. **public** **class** SpecialCharMarnager {
3. **private** String s ;
5. **public** SpecialCharMarnager(String s){
6. **super**();
7. **this**.s = s;
8. }
9. //Metodo che sostituisce caratteri speciali in una Stringa
10. **public** String substitute(){
12. ArrayList<String> specialCharList = **new** ArrayList<>();
13. specialCharList.add("{");
14. specialCharList.add("}");
15. specialCharList.add("&");
16. specialCharList.add("%");
17. specialCharList.add("$");
18. specialCharList.add("#");
19. specialCharList.add("^");
20. specialCharList.add("\_");
21. specialCharList.add("~");
23. **for**(**int** i = 0; i<specialCharList.size(); i++){
24. String special = specialCharList.get(i);
25. **if**(s.contains(special)){
26. s = s.replace(special,"\\"+special);
27. }
28. }
29. **return** s;
30. }
31. }

Infine è stato implementato il metodo def compile(UserComment comment) in modo da poter leggere i commenti e strutturare il file latex generato in: sezioni, sotto-sezioni, paragrafi, sotto-paragrafi. La struttura finale di tale metodo è la seguente:

1. def compile(UserComment comment){
2. //Per non far stampare i simboli (\*\*)in latex
3. var String s = comment.contentComment.substring(2,comment.contentComment.length-2)
5. var **int** regex = comment.contentComment.indexOf(" ",2)
6. var String com = comment.contentComment.substring(2,regex)
7. var SpecialCharMarnager mngr = **new** SpecialCharMarnager(s.substring(regex-1))
9. **if**(com.equalsIgnoreCase("SECTION")) {
11. **return**
12. '''
14. \section{«com»}
16. «mngr.substitute()»
18. '''
19. }**else** **if**(com.equalsIgnoreCase("SUBSECTION")) {
21. **return**
22. '''
24. \subsection{«com»}
26. «mngr.substitute()»
28. '''
29. }**else** **if**(com.equalsIgnoreCase("SUBSUBSECTION")) {
31. **return**
32. '''
34. \subsubsection{«com»}
36. «mngr.substitute()»
38. '''
39. }**else** **if**(com.equalsIgnoreCase("PARAGRAPH")) {
41. **return**
42. '''
44. \paragraph{}
46. «mngr.substitute()»
48. '''
49. }**else** **if**(com.equalsIgnoreCase("SUBPARAGRAPH")) {
51. **return**
52. '''
54. \subparagraph{}
56. «mngr.substitute()»
58. '''
59. }
61. }

**7. Risultato della traduzione**

Dopo queste modifiche il file **EbnfLangLatexGenerator.xtend** è completo. Quindi è possibile, dato un generico **file.ebnf**, generare il corrispondente **file.tex**. Per fare questo, nella cartella **test.it.unibg.ebnfwb.exporter.latex,** fare click destro sul file **EbnfLangLatexGenerator.xtend** -> Run as -> JUnit test. Nella cartella **examples/UASMExample** viene creato il file **DEFAULT\_OUTPUTUASM\_ebnf\_iso.tex** e il corrispondente pdf **DEFAULT\_OUTPUTUASM\_ebnf\_iso.pdf**.