

# Indice

<b>1. Prefazione</b>	<b>2</b>
1.1 Version History	2
<b>2. Descrizione del prodotto software</b>	<b>3</b>
<b>3. Funzionalità oggetto di test</b>	<b>4</b>
3.1 Elenco delle funzionalità oggetto di test	4
<b>4. Esiti dei test</b>	<b>6</b>
4.1 Esiti e copertura dei test di unità	6
4.1.1 Esiti dei test per la classe Stato	6
4.1.2 Esiti dei test per la classe RegolaDiProduzione	7
4.1.3 Esiti dei test per la classe NonTerminale	7
4.1.4 Esiti dei test per la classe Solver	8
4.1.5 Esiti dei test per la classe Terminale	8
4.2 Esiti e copertura dei test di sistema	9

# 1. Prefazione

## 1.1 Version History

- Versione 0.0 - 16/04/2019 : Definizione piano di test
- Versione 1.0 - 04/05/2019 : Esiti dei test della versione 1
- Versione 2.0 - TBD : Esiti dei test della versione 2

## 2. Descrizione del prodotto software

Il software, sviluppato in linguaggio Java in ambiente di sviluppo Eclipse, è suddiviso nei 5 packages descritti di seguito:

- **app** - questo package contiene il file *Main.java*, oltre ad altri file di supporto, utilizzati per la creazione dell'interfaccia grafica;
- **graph** - questo package contiene tutti i file necessari alla generazione del grafo;
- **lr1package** - questo package contiene due degli output di ANTLR, strumento utilizzato per generare il compilatore, ossia *PrototipoLR1Lexer.java* e *PrototipoLR1Parser.java*; il terzo e ultimo file di output (*PrototipoLR1.tokens* è invece situato all'esterno di questo package;
- **myPackage** - questo package contiene il file *Environment.java*, all'interno del quale è definita la classe Environment, necessaria per il corretto funzionamento dei file output di ANTLR ed ereditata da un progetto di esempio;
- **solver** - questo package contiene tutti i file che contengono definizioni di classi custom utilizzate per il processo di riconoscimento delle grammatiche e per rappresentare gli elementi costituenti delle grammatiche.

La versione oggetto di test è la 2.0, pubblicata sul branch *master* della repository di Github del progetto (<https://github.com/d-presciani/progettoLFC>) in data **TBD**.

### 3. Funzionalità oggetto di test

Le funzionalità sottoposte a test di unità saranno quelle definite nelle classi contenute all'interno del package *solver*, essendo queste le classi scritte manualmente e quelle che consentono l'effettivo funzionamento del programma; si trascurano di effettuare test di unità sui restanti package in quanto generati automaticamente da ANTLR e difficilmente sottoponibili a test di tale granularità.

È previsto anche un test di sistema, fornendo al sistema diversi input, per verificare la corretta integrazione tra i vari componenti e, al contempo, il corretto funzionamento del programma nel suo complesso.

Oltre ai suddetti test, è prevista anche l'esecuzione di un'analisi statica del codice tramite i due plugin di Eclipse *SpotBugs* e *PMD*.

#### 3.1 Elenco delle funzionalità oggetto di test

Di seguito vengono riportate le classi, con i relativi metodi, per cui saranno effettuati i test di unità.

- Stato
  - private void aggiungiCompletamento(RegolaDiProduzione nuovaRdp, RegolaDiProduzione regolaPadre)
  - public void aggiungiCore(RegolaDiProduzione rdp)
  - public Stato()
  - public void espandiStato(LinkedList<Stato> listaStati, LinkedList<Transizione> listaTransizioni)
  - public String toString()
- RegolaDiProduzione
  - public RegolaDiProduzione()
  - public RegolaDiProduzione(NonTerminale parSX, List<Carattere> parDX)
  - public RegolaDiProduzione(RegolaDiProduzione reg)
  - public void addSeguito(String s)
  - public void avanzaPuntino()
  - public void calcolaAnnullabilita()
  - public boolean equals(Object o)
  - public String toString()
- NonTerminale
  - public NonTerminale (String lettera)
  - public void addRegola (RegolaDiProduzione reg)
  - public void calcolaAnnullabile()
  - public List<String> calcolaInizi(LinkedList<RegolaDiProduzione> prevReg)

- public void controlloProduzioni() throws ErroreSemantico
- public String getLettera()
- public List<RegolaDiProduzione> getRegole()
- public boolean isAnnullabile()
- public boolean isTerminale()
- public void setAnnullabile()
- public void stampaRegole()
- public String toString()
- Solver
  - public Risultati solve(LinkedList<NonTerminale> listaNT,  
LinkedList<RegolaDiProduzione> listaReg)
- Terminale
  - public Terminale(String car)
  - public List<String> calcolaInizi(LinkedList<RegolaDiProduzione> prevReg)
  - public String getLettera()
  - public List<RegolaDiProduzione> getRegole()
  - public boolean isAnnullabile()
  - public boolean isTerminale()
  - public String toString()

## 4. Esiti dei test

### 4.1 Esiti e copertura dei test di unità

Di seguito sono riportati gli esiti dei test e la copertura degli stessi per ognuna delle classi precedentemente evidenziate come oggetto di test di unità; nel complesso tali test hanno consentito una copertura del package *solver* pari al 95,6%.

▼  solver	95,6 %	2.314	107	2.421
>  Carattere.java	100,0 %	3	0	3
>  ErroreSemantico.java	100,0 %	4	0	4
>  NonTerminale.java	100,0 %	177	0	177
>  RegolaDiProduzione.java	95,6 %	238	11	249
>  Risultati.java	62,5 %	15	9	24
>  Solver.java	100,0 %	162	0	162
>  Stato.java	96,1 %	1.674	68	1.742
>  Terminale.java	100,0 %	29	0	29
>  Transizione.java	38,7 %	12	19	31

Figura 1: Copertura dei test per il package solver

#### 4.1.1 Esiti dei test per la classe Stato

Per la classe Stato sono stati scritti, all'interno della classe StatoTest, 22 test, superati correttamente dal programma, che garantiscono una copertura della classe Stato pari al 96,1%.

```

▼  StatoTest [Runner: JUnit 5] (0,001 s)
   aggiuntaConCompPres() (0,000 s)
   aggiuntaCompletamenti() (0,000 s)
   testToString() (0,000 s)
   espansioneStatoCore() (0,000 s)
   aggiuntaCoreNoInizi() (0,000 s)
   aggiuntaRegolaCoreIndice() (0,000 s)
   controlloLR1Spostamento() (0,000 s)
   espansioneStatoCompletamentiDuplicati() (0,000 s)
   espansioneStatoCoreDuplicati() (0,000 s)
   espansioneStatoCompletamento() (0,000 s)
   espansioneDaComp() (0,000 s)
   generazione() (0,000 s)
   aggiuntaRegolaCoreNT() (0,000 s)
   controlloLR1RiduzioneCore() (0,000 s)
   aggiuntaCoreAnnullabileSeguitiPadre() (0,000 s)
   ricalcoloInizi() (0,000 s)
   testToStringNoComplementi() (0,000 s)
   aggiuntaCoreNullaSeguitiPadre() (0,000 s)
   controlloLR1RiduzioneCompletamenti() (0,000 s)
   aggiuntaRegolaCore() (0,000 s)
   aggiuntaRegolaCoreNulla() (0,000 s)
   aggiuntaCoreCarattereSingolo() (0,000 s)

```

Figura 2: Esiti dei test della classe Stato

### 4.1.2 Esiti dei test per la classe RegolaDiProduzione

Per la classe RegolaDiProduzione sono stati scritti, all'interno della classe RegolaDiProduzioneTest, 6 test, superati correttamente dal programma, che garantiscono una copertura della classe RegolaDiProduzione pari al 95,6%.

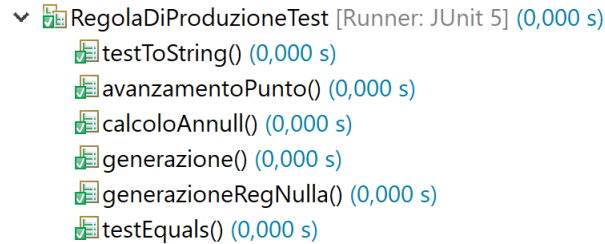


Figura 3: Esiti dei test della classe RegolaDiProduzione

### 4.1.3 Esiti dei test per la classe NonTerminale

Per la classe NonTerminale sono stati scritti, all'interno della classe NonTerminaleTest, 13 test, superati correttamente dal programma, che garantiscono una copertura della classe NonTerminale pari al 100,0%.

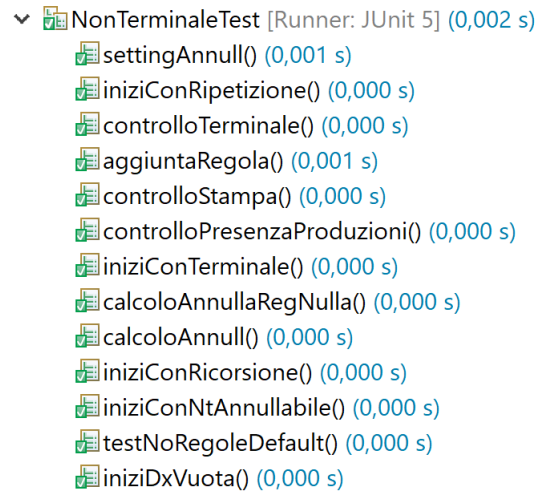


Figura 4: Esiti dei test della classe NonTerminale

### 4.1.4 Esiti dei test per la classe Solver

Per la classe Solver sono stati scritti, all'interno della classe SolverTest, 2 test, superati correttamente dal programma, che garantiscono una copertura della classe Solver pari al 100,0%.

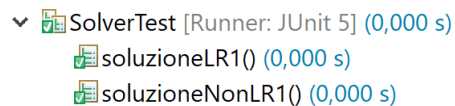


Figura 5: Esiti dei test della classe Solver

### 4.1.5 Esiti dei test per la classe Terminale

Per la classe Terminale sono stati scritti, all'interno della classe TerminaleTest, 7 test, superati correttamente dal programma, che garantiscono una copertura della classe Terminale pari al 100,0%.

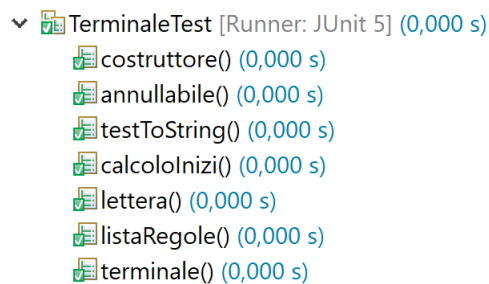


Figura 6: Esiti dei test della classe Terminale



## 4.2 Esiti e copertura dei test di sistema

I test di sistema sono stati effettuati eseguendo il metodo *main* della classe *Riconoscitore* più volte, passandogli come input i file contenuti all'interno della cartella "lfc/resources" presente nella repository del progetto (<https://github.com/d-presciani/progettoLFC>); come oracoli per valutare la correttezza dei test sono stati utilizzati i temi svolti in classe durante il corso di Linguaggi formali e compilatori.

Durante i test di sistema è stata rilevata anche la copertura del codice nelle varie esecuzioni; nella seguente immagine è possibile vedere la copertura complessiva (ottenuta combinando la copertura delle singole esecuzioni).






















Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ app	 0,0 %	0	588	588
> Main.java	 0,0 %	0	588	588
▼ graph	 0,0 %	0	374	374
> JGraphXDrawer.java	 0,0 %	0	353	353
> RelationshipEdge.java	 0,0 %	0	12	12
> RisImmagine.java	 0,0 %	0	9	9
▼ Ir1package	 0,0 %	0	1.426	1.426
> PrototipoLR1Lexer.java	 0,0 %	0	701	701
> PrototipoLR1Parser.java	 0,0 %	0	725	725
▼ myPackage	 0,0 %	0	72	72
> Environment.java	 0,0 %	0	72	72
▼ solver	 95,6 %	2.314	107	2.421
> Carattere.java	 100,0 %	3	0	3
> ErroreSemantico.java	 100,0 %	4	0	4
> NonTerminale.java	 100,0 %	177	0	177
> RegolaDiProduzione.java	 95,6 %	238	11	249
> Risultati.java	 62,5 %	15	9	24
> Solver.java	 100,0 %	162	0	162
> Stato.java	 96,1 %	1.674	68	1.742
> Terminale.java	 100,0 %	29	0	29
> Transizione.java	 38,7 %	12	19	31

Figura 7: Copertura del codice relativa ai test di sistema