

# Challenge a tema: Linguaggi

## A. P. Linguaggi e Modelli Computazionali M

Autore Pietro Bertozzi

Docente Enrico Denti

Anno Accademico 2025

Sabato, 03.05.2025

# Contesto e Premesse

# CyberCup?

## DEFINIZIONE

**CyberCup.IT** è una competizione italiana di cybersecurity che coinvolge università e istituzioni accademiche in tutta Italia.

## ATTENZIONE

L'**Università di Bologna** Università di Bologna è rappresentata dal **CTF team di ULISSE**, sotto la guida del **professor Marco Prandini**. Io ne faccio parte.

# Challenge?

## DEFINIZIONE

Le challenge di sicurezza informatica sono esercizi pratici che simulano scenari reali di attacco o difesa, sviluppando competenze tecniche e problem solving.






## ATTENZIONE

Lo stesso approccio, basato sull'apprendimento attivo, può essere applicato anche agli altri rami dell'informatica.

Nel nostro caso ai **linguaggi di programmazione**.

# Challenge Efficace

Quale è la ricetta per una ottima challenge?

-  Obiettivo formativo chiaro (**analisi dei requisiti**).
-  Problema interessante ma accessibile.
-  Curva di apprendimento stimolante.
-  Gamification.
-  Spunti di approfondimento.

# **Analisi dei Requisiti**

## **Obbiettivi Formativi**

# Challenge Proposte

## OBBIETTIVO

«**Ambiguous Collisions**» invita a riflettere sul significato del linguaggio, introduce il formalismo delle grammatiche e stressa il concetto di **ambiguità**.

## OBBIETTIVO

«**Language Generator**» stimola la comprensione del funzionamento degli interpreti, guidando passo dopo passo nella **creazione** di **piccole grammatiche** in grado di generare frasi con semplici significati semantici.

# **Analisi dei Requisiti**

## **Standard Architetture e Compatibilità**



# Docker e i Container

Nelle challenge Jeopardy, **Docker è fondamentale**: ogni esercizio viene eseguito in un **container isolato**, che contiene tutto l'ambiente necessario per la sfida.

Questo approccio garantisce portabilità, sicurezza e coerenza, facilitando la distribuzione e il deployment sia per chi organizza che per chi partecipa.

La necessità di essere containerizzabile influenza direttamente la progettazione della challenge, modellando la struttura del codice, dei servizi e delle interazioni.

La compatibilità con l'ambiente Docker deve essere considerata già nella fase di analisi dei requisiti: **una challenge non facilmente containerizzabile diventare impraticabile in un contesto Jeopardy.**

# Ambiguous Collisions

## Analisi del Problema

# Ambiguità e Indecidibilità

Capire **se una grammatica è ambigua** — cioè se può generare almeno una frase con due derivazioni diverse — è un problema **indecidibile**: non esiste un algoritmo generale che possa rispondere sempre correttamente.

L'ambiguità implica dover controllare tutte le frasi del linguaggio, ma una grammatica genera, di norma, un insieme infinito di stringhe.

Non esiste una lunghezza oltre la quale si possa smettere di cercare: **l'ambiguità potrebbe nascondersi ovunque**.

Tuttavia, **se una stringa sospetta è nota**, si possono generare tutte le sue derivazioni e verificare direttamente se è ambigua.

# Ambiguous Collisions

## Grammatiche Utilizzate

# Tre Piccoli Esercizi

Sono presentate tre grammatiche semplici ma volutamente ambigue, che illustrano in modo diretto alcune problematiche classiche nella definizione sintattica dei linguaggi:

1. **Operatore Associativo a Destra e a Sinistra**
2. **Grammatica Definita a Strati**
3. **Dangling Else**

## **OBIETTIVO**

Il partecipante dovrà, per ogni grammatica, trovare una frase ambigua.

# Operatore Associativo a Destra e a Sinistra

## CODICE

```
grammar Level1;
```

```
s : a ;
```

```
a : c | a b a ;
```

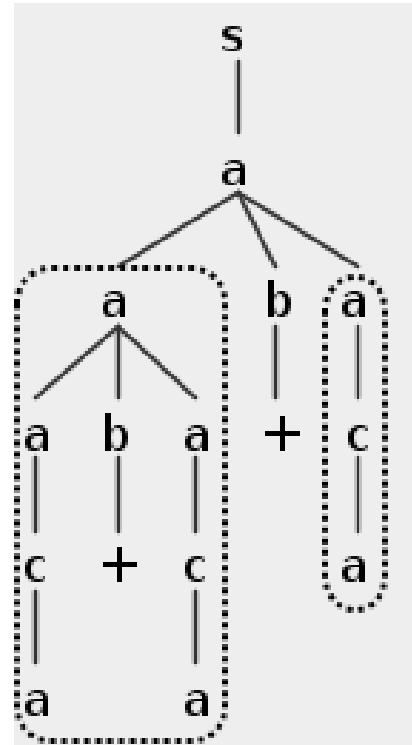
```
b : PLUS ;
```

```
c : A ;
```

```
PLUS : '+' ;
```

```
A : 'a' ;
```

```
WS : [ \t\r\n]+ -> skip ;
```



# Grammatica Definita a Strati

## CODICE

```
grammar Level2;
```

```
s : a ;  
a : c | b c ;  
b : NOT ;  
c : e | c d e ;  
d : AND | OR ;  
e : f | b f ;  
f : h | f g h ;  
g : LT | GT | EQ ;  
h : j | h i j ;  
i : STAR | SLASH ;  
j : l | j k l ;  
k : PLUS | MINUS ;  
l : B ;
```

## CODICE

```
NOT    : '!' ;  
AND    : '&' ;  
OR     : '|' ;  
LT     : '<' ;  
GT     : '>' ;  
EQ     : '=' ;  
STAR   : '*' ;  
SLASH  : '/' ;  
PLUS   : '+' ;  
MINUS  : '-' ;  
B      : 'b' ;  
WS     : [ \t\r\n]+ -> skip ;
```

# Dangling Else

## CODICE

```
grammar Level3;
```

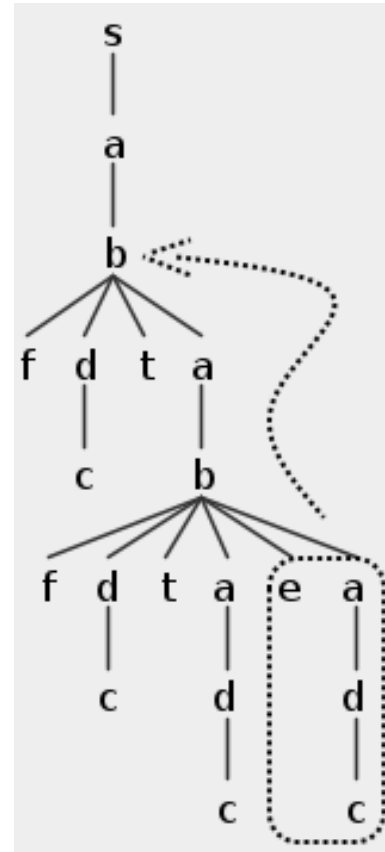
```
s : a ;
```

```
a : b | c | d ;
```

```
b : F d T a | F d T a E a ;
```

```
c : W d D a ;
```

Poi la grammatica continua,  
correggendo quella precedente, usando  
d come generico blocco di codice.





# Ambiguous Collisions

## Architettura del Sistema

# Microservizio Testuale






La challenge è eseguita all'interno di un **container Docker isolato**, progettato per esporre un'**interfaccia testuale** attraverso un **socket TCP** sulla porta 1337.

Questo è ottenuto tramite un docker-compose service che monta lo script Python come entrypoint e reindirizza la porta del container verso l'esterno (1337:1337).

Questa connessione apre un canale raw di testo, che replica un'interfaccia terminale remota, dove **l'utente può interagire con il programma Python come se fosse in una sessione locale**.





La logica della challenge vive interamente nello script Python, che legge da stdin e scrive su stdout, rendendolo perfetto per essere containerizzato come un **microservizio testuale**.

# Perché Usare il CYK Parser

-  Funziona con grammatiche in forma normale di Chomsky (CNF) -> posso definire staticamente le versioni CNF per i tre linguaggi richiesti.
-  Si basa su programmazione dinamica -> è una tecnica che mi è familiare ho capito subito l'algoritmo.
-  Permette di ricostruire tutti gli alberi sintattici -> ottimo per avere la situazione totalmente sotto controllo.
-  Complessità  $O(n^3)$  -> gestibile con frasi brevi (sotto i 10 caratteri).
-  Alternative esistono (Earley, GLR) -> più complesse da implementare senza che ci sia davvero un vantaggio in questo caso.

# Perché Non Usare il CYK Parser

In questo caso, la soluzione che ho adottato è la più semplice in assoluto. So già quali erano le ambiguità, quindi ho semplicemente implementato dei controlli diretti, come un semplice «if risposta corretta».

-  Fornisce una valutazione immediata.
-  Riduce il carico computazionale -> cruciale durante una competizione con molti partecipanti per evitare di compromettere le performance.
-  Ideale per scenari dove l'affidabilità operativa è fondamentale.
-  Purtroppo... Meno soddisfazione dal punto di vista teorico.

# Ambiguous Collisions

## Test e Collaudo

# Ma la Challenge Funziona?

1. **Conferma con ANTLR e GRUN:** Presa visione degli alberi di derivazione durante la stesura delle grammatiche stesse, utilizzando l'opzione -gui.
2. **CYK Parser:** Implementato per confermare le ambiguità.
3. **Test BlackBox:** Alcuni degli altri membri del team hanno testato le mie challenge, mentre io testavo le loro.
4. **Test WhiteBox con writeup:** Questo step non ha davvero funzionato... sono stati individuati molti più errori nei writeup che nelle challenge stesse...
5. **Collaudo CTF:** Collaudo finale durante la competizione che si è tenuta nel fine settimana del 5 aprile 2025. Passato a pieni voti.

# Codice CYK Parser

## CODICE

```
for (int span = 2; span <= n; span++) {
    for (int start = 0; start <= n - span; start++) {
        int end = start + span;
        table[start][end] = new HashSet<>();
        for (int split = start + 1; split < end; split++) {
            for (String left : table[start][split]) {
                for (String right : table[split][end]) {
                    Set<String> combined =
grammar.closure(grammar.getBinaryProducers(left, right));
                    if (!combined.isEmpty()) {
                        System.out.println("table[" + start + "][" + end + "] ← " +
combined + " (da " + left + "+" + right + ")");
                        table[start][end].addAll(combined);
                    }
                }
            }
        }
    }
}
```

# Test CYK Parser

## CODICE

```
pietrobertozzi@LAPTOP-SUQ13SMQ:/mnt/c/Users/HP/OneDrive/Desktop/Magistrale/attivita-progettuale-linguaggi-modelli-computazionali-m/ambiguous-collisions/cyk$ java CYKParser
```

## CODICE

```
--- Derivazioni CYK ---
table[0][1] ← [A, C, S] (per 'a')
table[1][2] ← [B] (per '+')
table[2][3] ← [A, C, S] (per 'a')
table[3][4] ← [B] (per '+')
table[4][5] ← [A, C, S] (per 'a')
table[1][3] ← [X] (da B+A)
table[3][5] ← [X] (da B+A)
table[0][3] ← [A, S] (da A+X1)
table[2][5] ← [A, S] (da A+X1)
table[1][5] ← [X] (da B+A)
table[0][5] ← [A, S] (da A+X1)
table[0][5] ← [A, S] (da A+X1)
```

```
Accepted: true
Number of derivations for 'S': 2
```

## CODICE

```
--- Tabella CYK ---
table[0][1] = [A, C, S]
table[0][2] = []
table[0][3] = [A, S]
table[0][4] = []
table[0][5] = [A, S]
table[1][2] = [B]
table[1][3] = [X1]
table[1][4] = []
table[1][5] = [X1]
table[2][3] = [A, C, S]
table[2][4] = []
table[2][5] = [A, S]
table[3][4] = [B]
table[3][5] = [X]
table[4][5] = [A, C, S]
```



# Language Generator

## Analisi del Problema

# E' un Argomento Difficile

C'è un problema...

I partecipanti difficilmente sono esperti di linguaggi e grammatiche.

1. Cosa è un riconoscitore ll/lr?
2. Cosa cambia davvero tra riconoscitore, generatore, interprete e compilatore?

Serve un nuovo strumento nuovo che:

1. incuriosisca un ignorante (colui che non sa) a studiare come funzionano davvero i linguaggi e approfondire l'argomento,
2. permetta una sfida alla portata di tutti, e non solo agli ingegneri informatici...

Quali aspetti semplificare e quali lasciare intatti?

# Generatore Semplice e Interprete più Semplice

Seguono le semplificazioni molto pesanti che allontanano gli strumenti della challenge dalle versioni serie discusse a lezione:

1. Solo una delle frasi del linguaggio ha effettivamente significato, le altre non fanno nulla.
2. Il significato non dipende dalla derivazione, ma dipende solo ed esclusivamente dal risultato finale.
3. L'eventuale semantica associata ad ogni frase corrisponde a quella che la frase avrebbe in lingua inglese, il linguaggio più noto di tutti.
4. Il generatore espande i simboli non terminali utilizzando Depth-First-Search oppure Breadth-First-Search.

# I Mattoncini Rimangono

Sopravvivono le idee di linguaggio, grammatica, scopo, produzione, generazione, interpretazione, rappresentazione interna attraverso una struttura ad albero...

Chi prima della sfida non ne aveva idea adesso può cominciare a studiare in autonomia per capire come funziona davvero e i mattoncini li ha...

...Chiaramente c'è una bella differenza tra i mattoncini e le case.

## OBBIETTIVO

Per risolvere la Challenge è necessario comporre diverse grammatiche mettendo assieme produzioni già pronte, una per ogni livello.

# Language Generator

## Grammatiche Utilizzate

# Hint

## CODICE

```
grammar Hint;
```

```
r  : H I N T ;
```

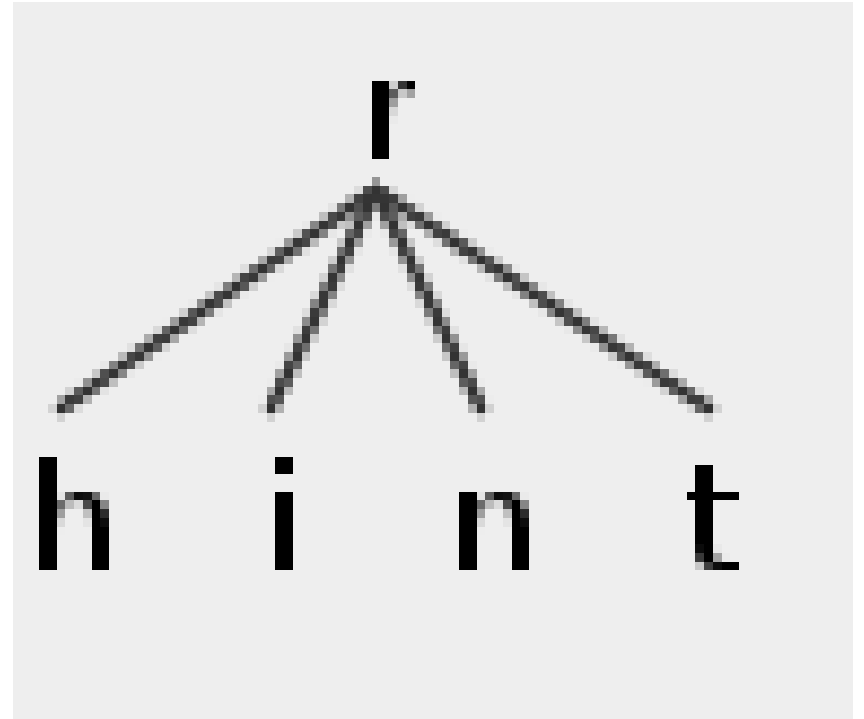
```
H  : 'h' ;
```

```
I  : 'i' ;
```

```
N  : 'n' ;
```

```
T  : 't' ;
```

```
WS : [ \t\r\n]+ -> skip ;
```



# PleaseHint

## CODICE

```
grammar PleaseHint;
```

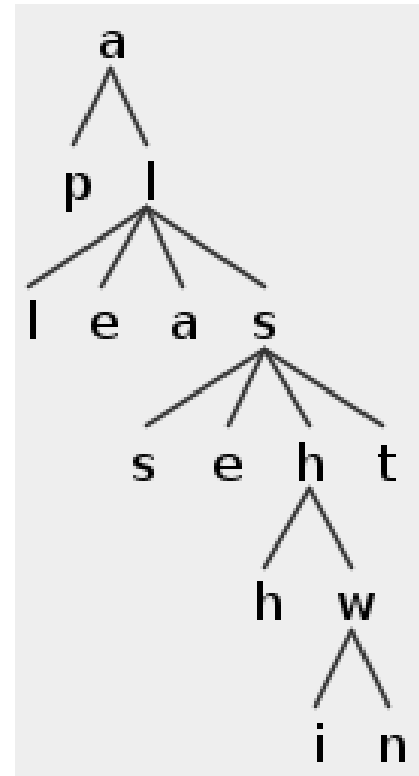
```
a  : P l | A G ;
```

```
h  : I N | H w ;
```

```
l  : L E A s ;
```

```
s  : S H | S E h T ;
```

```
w  : I N;
```



# ClearLog

## CODICE

```
grammar ClearLog;
```

```
b : C l k | S h ;
```

```
g : G ;
```

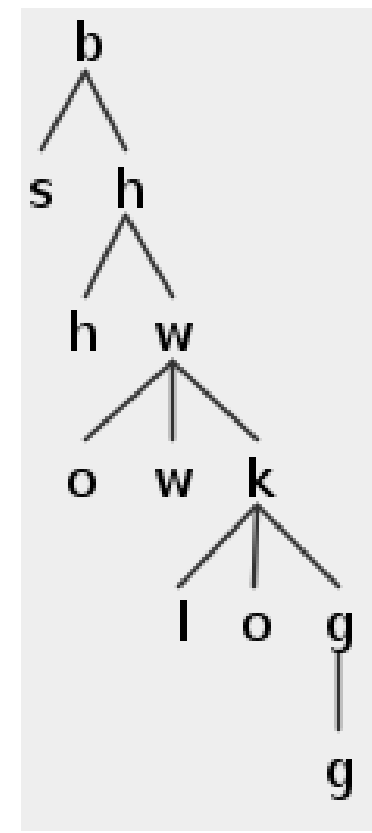
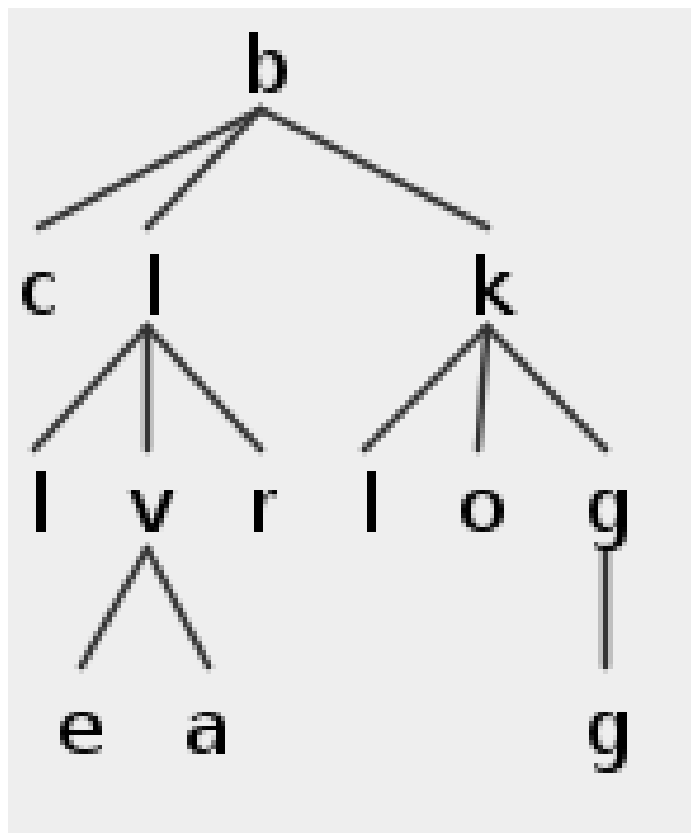
```
h : I N | H w ;
```

```
k : L O g ;
```

```
l : L v R ;
```

```
v : E A | H O | E R ;
```

```
w : I N | O W k ;
```





# ShowLog

## CODICE

```
grammar ShowLog;
```

```
g  : G ;
```

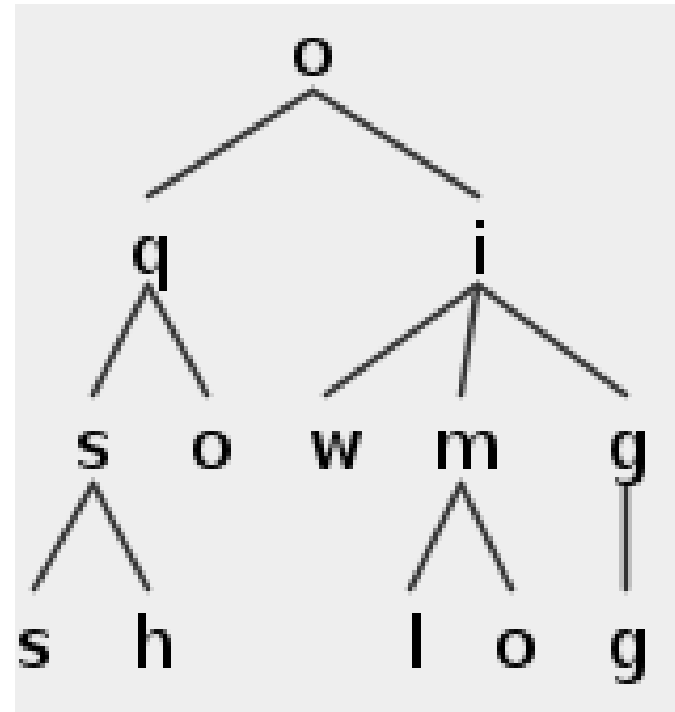
```
i  : W m g | L ;
```

```
m  : L O ;
```

```
o  : H A N | q i ;
```

```
q  : s O | G ;
```

```
s  : S H ;
```



# ChangeTraversal

## CODICE

```
grammar ChangeTraversal;
```

```
g : G p | g T R t ;
```

```
n : C o g ;
```

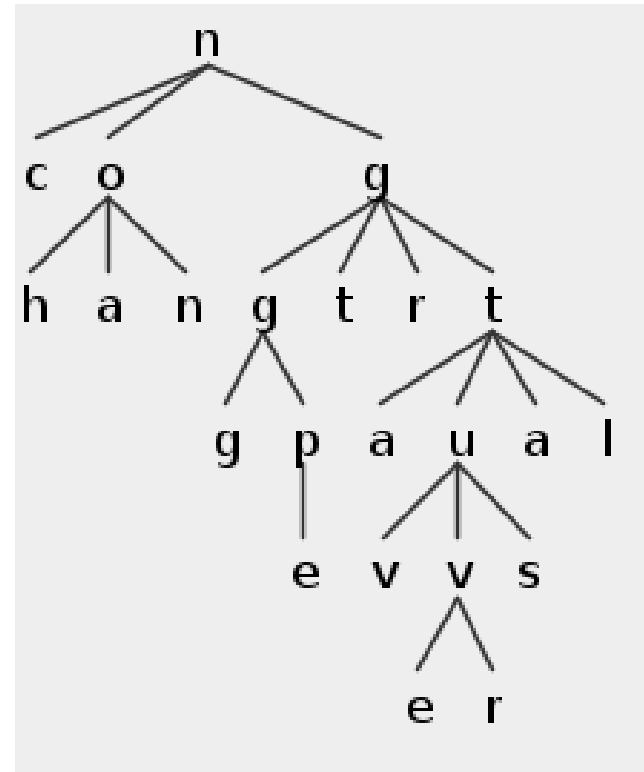
```
o : H A N ;
```

```
p : A | E | O | ;
```

```
t : A u A L ;
```

```
u : V v S ;
```

```
v : E A | H O | E R ;
```



# GetFlag

## CODICE

```
grammar GetFlag;
```

```
a  : A G ;
```

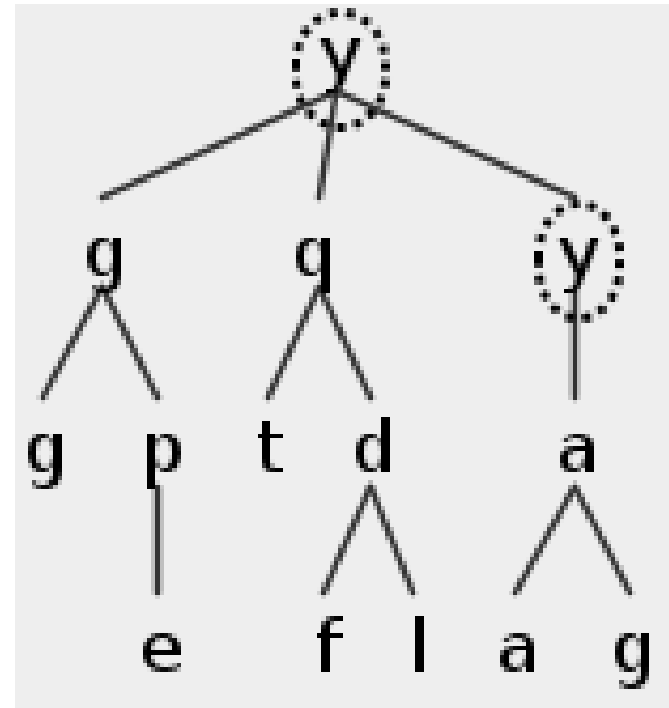
```
d  : F L ;
```

```
g  : G p ;
```

```
p  : A | E | O | ;
```

```
q  : T d | G ;
```

```
y  : g q y | a ;
```



# Language Generator

## Architettura del Sistema

# Microservizio Web PHP

La challenge è contenuta in un **contenitore Docker PHP** basato sull'immagine `php:8.2-apache`, che espone un'**interfaccia web** sulla porta 5000.

Tutti i file statici e dinamici (HTML, CSS, JavaScript, PHP) vengono copiati nella root di Apache, formando un'app **monolitica e autosufficiente**.

Logica del gioco, tracciamento del progresso e generazione delle risposte (hint, log, flag) sono gestiti in PHP via `$_SESSION`, **senza database o file persistenti**.

In questo modo è semplice supportare **diversi utenti in contemporanea**.

Il servizio è definito in un `docker-compose` che espone il container su `localhost:5000`, rendendolo accessibile come una **classica web app interattiva**.

# Language Generator

## Test e Collaudo

# Ma la Challenge Funziona?

1. **Conferma con ANTLR e GRUN:** Presa visione degli alberi di derivazione durante la stesura delle grammatiche stesse, utilizzando l'opzione -gui.
2. **Test BlackBox:** Alcuni degli altri membri del team hanno testato le mie challenge, mentre io testavo le loro.
3. **Test WhiteBox con writeup:** Questo step non ha davvero funzionato... sono stati individuati molti più errori nei writeup che nelle challenge stesse...
4. **Collaudo CTF:** Collaudo finale durante la competizione che si è tenuta nel fine settimana del 5 aprile 2025. Passato a pieni voti.

# Sviluppi Futuri



# Prossimamente...

## 1. Ambiguous Collisions 2

- grammatiche casuali
- innumerevoli livelli
- necessità di scriptare soluzione ed interazione

## 2. JSFuck ...

- sanitizzazione superficiale dell'input
- necessità di aggirare la blacklist usando type coercion

## 3. Prototype Pollution ...

- manipolazione abusiva degli oggetti globali in ambienti dinamici