

Tutorial Laboratorio 1 (Gramática)

El siguiente es un archivo que les indicará cómo correr el test suite de pruebas del primer laboratorio, así como algunos detalles a tener en cuenta.

Estructura del Test Suite

El test suite se compone de los siguientes archivos:

- CPP.cf (Contienen la gramática base/inicial y es el archivo sobre el que estarán trabajando)
- progs-test-labs1.hs (Script que corre las pruebas good y bad, y les brinda un resumen de los casos que pasan y los que no.)
- Makefile (Script que permite actualizar la definición del Test individual con el comando make)
- good/ (Carpeta que contienen los casos de prueba que deben ser parseados)
- bad/ (Carpeta que contiene los casos de prueba que deben ser rechazados por la gramática)

Pasos:

1. Descomprimir el comprimido de Aulas (en el caso de los estudiantes de docker deberán hacerlo dentro de la carpeta donde tengan definida la instancia)
2. Para generar/actualizar los archivos del Lexer, Parser y TestCPP deben correr el comando `make`
3. Para correr el Test suite hay dos opciones, todos los tests juntos o manualmente 1 a la vez.

4. Correr todos los Tests Juntos

1. Primero deberán compilar el script que ejecuta el test suite entero con el comando `ghc progs-test-lab1.hs`
2. Una vez compilado, pueden correr todos los tests con el ejecutable generado con el siguiente comando `./progs-test-lab1 CPP.cf`
3. La salida generará un mensaje corto por los casos correctos y un mensaje largo por las pruebas fallidas (tanto las good, como las bad) y al final les indicará cuantos test fueron exitosos en la categoría good y bad.

5. Correr un test individual

1. Para correr un test individual usaremos el archivo de Haskell generado por **BNFC** que ejecuta el parser *TestCPP*
2. Para correr una prueba individual deben ejecutar el comando `./TestCPP [archivo a parsear]`
 1. Por ejemplo una prueba de la carpeta good podría ser: `./TestCPP good/test1.cc`
 2. Una prueba bad podría ser: `./TestCPP bad/bad001.cc`
3. Se espera que al correr las pruebas individuales las good devuelvan un mensaje de success y el parsing del programa, y que las bad devuelvan un mensaje de Error.
4. Es **importante** correr el comando `make` luego de hacer cambios en la gramática antes de volver a correr las pruebas individuales.
6. **CONFLICTOS**: Luego de correr el comando `make` en la salida de la terminal habrá una o 2 líneas que indican cuantos conflictos hay en la gramática (deben prestar atención a la cantidad de conflictos [elaboraremos más abajo] luego de cada cambio en el archivo CPP.cf)

Conflictos


Además de los tests, hay otro criterio de aceptación de la tarea que es la cantidad de conflictos en la gramática. Los conflictos se producen cuando el *Parser* (definido por la especificación de la gramática) se ve obligado a tomar una decisión para estructurar un programa de una forma o de otra (relacionado con las ambigüedades discutidas en clase). Hay 2 tipos de conflictos: **shift-reduce** y **reduce-reduce**. Éstos conflictos son de *naturalezas* diferentes y estaremos analizando cómo se producen y cómo se pueden resolver en el teórico. Sin embargo, es necesario verificar cuando uno hace cambios en la gramática, que la cantidad de conflictos sea baja.

En las entregas, es necesario que no haya ningún conflicto del tipo **reduce-reduce** y pueden haber hasta 10 conflictos de **shift-reduce**. Normalmente al introducir una nueva regla en la gramática, si se producen muchos conflictos de alguno de los 2 tipos, es síntoma de que la regla puede estar mal o puede *chocar* con otras reglas (definidas con anterioridad).

```
$ make
bnfc CPP.cf

71 rules accepted

Use Alex 3.0 to compile LexCPP.x.
ParCPP.y Tested with Happy 1.15
no change to file .\AbsCPP.hs
no change to file .\LexCPP.x
no change to file .\ParCPP.y
no change to file .\TestCPP.hs
no change to file .\DocCPP.txt
no change to file .\SkelCPP.hs
no change to file .\PrintCPP.hs
no change to file .\ErrM.hs
happy -gca ParCPP.y
unused rules: 1
unused terminals: 1
shift/reduce conflicts: 1
reduce/reduce conflicts: 21
alex -g LexCPP.x
ghc --make TestCPP.hs -o TestCPP
[5 of 7] Compiling LexCPP          ( LexCPP.hs, LexCPP.o )
[6 of 7] Compiling ParCPP          ( ParCPP.hs, ParCPP.o )
Linking TestCPP.exe ...
```

A red handwritten mark, possibly a stylized 'J' or a checkmark, is drawn over the conflict counts in the terminal output.