Instrucciones para debuggear el Parser

Esto documento contiene instrucciones sobre cómo obtener información sobre el Parser generado por BNFC y cómo analizar el proceso entero de Parsing al correr una prueba (tanto del parsing exitoso, como de los fallidos).

Al correr el comando de BNFC, se genera un archivo `ParCPP.y` que contiene la especificación de Happy (herramienta para generar Parsers) de la carmática CPP.

El archivo contiene la versión minimalisata del Parser y hay 2 operaciones que podemos hacer para aprovechar dicho archivo.

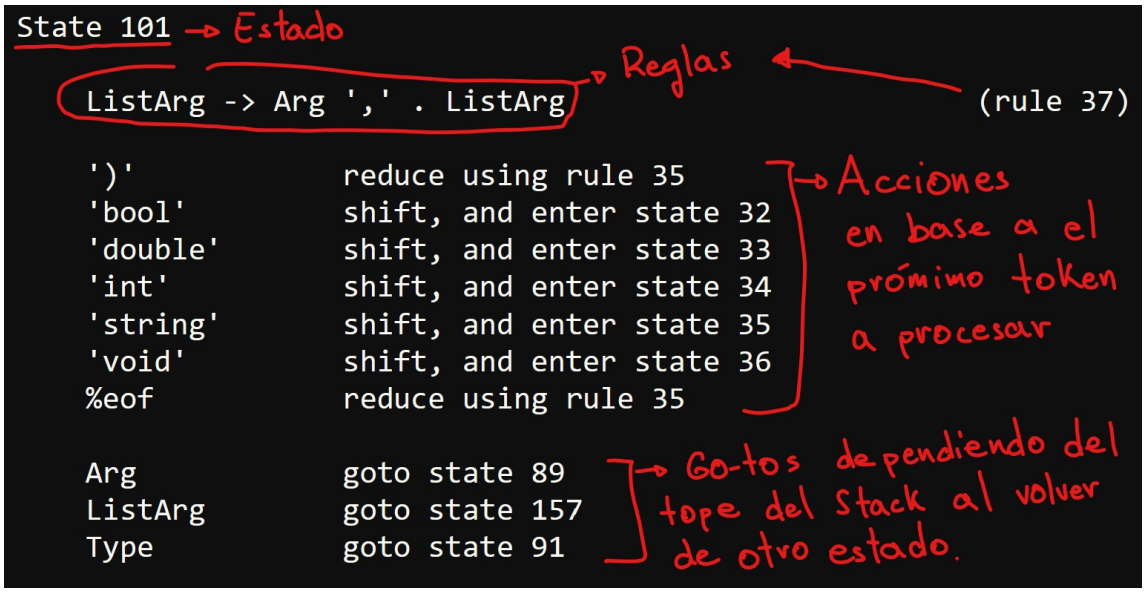
Obtener toda la información del Parser

Para obtener un archivo que contenga toda la información del comportamiento del Parser podemos correr el comando:

```
happy -i ParCPP.y
```

Este comando generará un archivo `ParCPP.info` que contiene la tabla entera del **LALR(k)** parser generado por BNFC y más información (cómo la lista de no terminales, terminales, cantidad de reglas, etc).

El siguiente es un ejemplo de un estado de dicho archivo con sus respectivas partes:



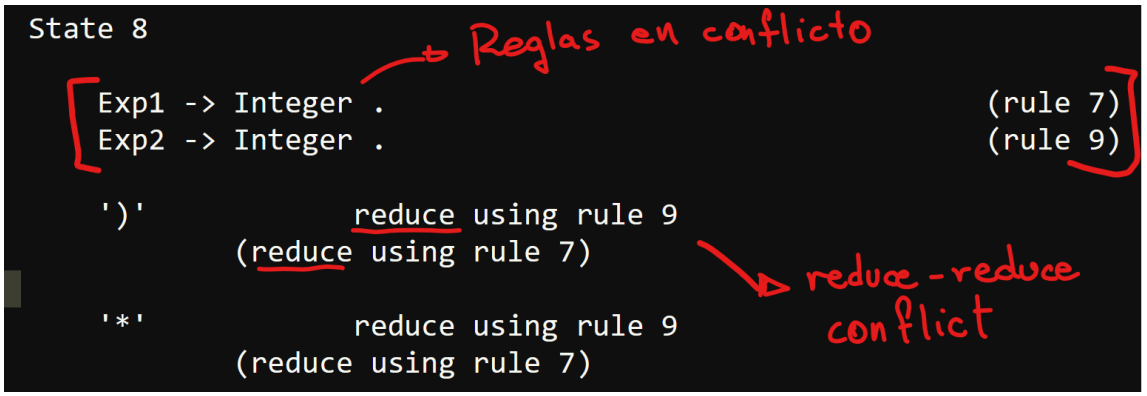
Buscar Conflictos dentro del archivo de información

Los conflictos de las reglas de la gramática (*shift-reduce* y *reduce-reduce*) se encuentran en la tabla de estados del archivo `ParCPP.info`. Para encontrarlos, se puede buscar la secuencia `(reduce` dentro del archivo.

En la sección de **Acciones** de los estados en los que encuentren `(reduce` habrán conflictos.

Si la línea por encima del `(reduce` tiene una acción de *shift*, el tipo de conflicto es de *shift-reduce* ya que el parser tiene que elegir entre realizar la acción de shift o la de reduce (cómo discutimos en el teórico, por convención se eleige la de shift).

En cambio, si la acción encima del conflicto es del tipo *reduce*, el tipo de conflicto será de *reduce-reduce*.



En el ejemplo de arriba se puede ver un conflicto de *reduce-reduce* entre la regla 7 y la 9 debido a que mabas producen el mismo resultado en dicho estdo.

Debuggear el Parser para una entrada especifica

Y que tenemos la lista de los estados en el archivo `ParCPP.info`, técnicamente se puede hacer el trace desde el estado inicial hasta cualquier expresión que deseemos paresear. Sin embargo incluso para lenguajes de tamaños pequeños dicha tabla se vuelve enorme y es difícil para una persona seguir rigurosamente los pasos hasta alcanzas una expresión en concreto.

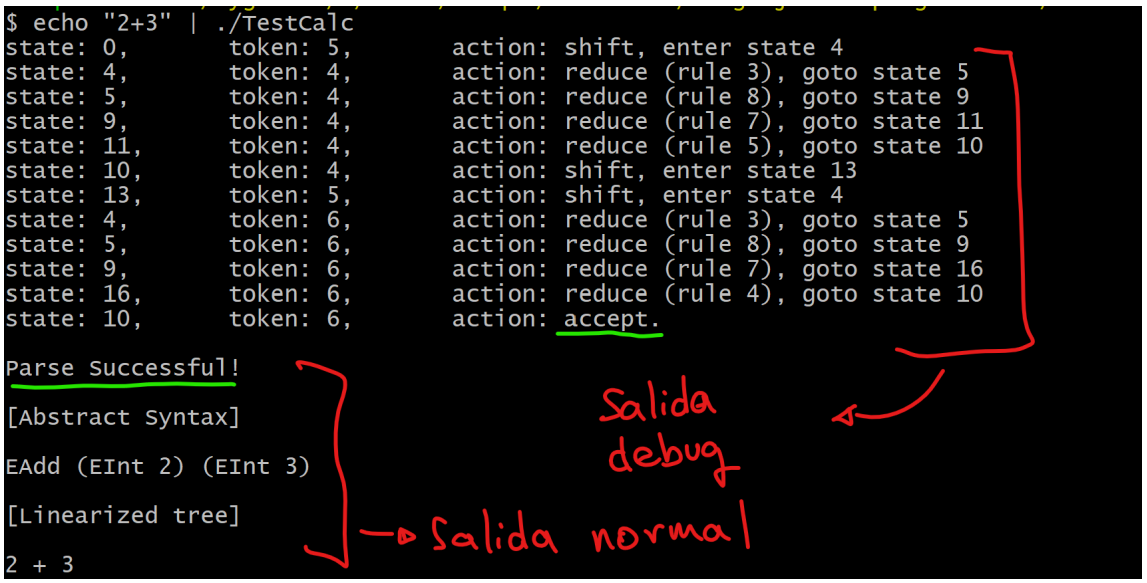
Por ello le podemos pedir a Happy que nos agregue información de *debug* en nuestro Parser para poder así tener una salida del trace de la ejecución del Parser.

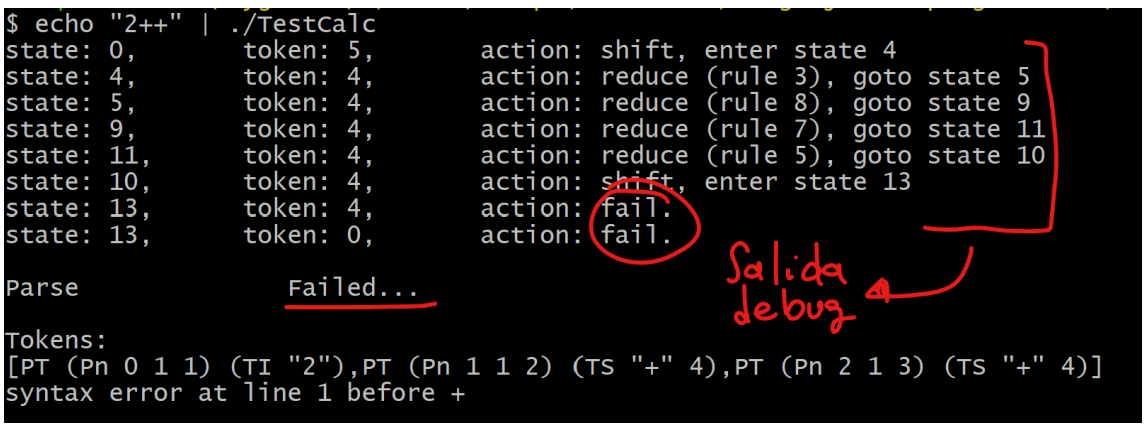
Para ello podemos correr los siguientes comandos:

1. `happy -da PraCPP.y` va a agregar la salida de debug a nuestro Parser
2. `ghc ./TestCPP.hs` una vez tengamos un parser distinto es importante actualizar el archivo ejecutable que usamos para probar nuestro parser y lexer con la nueva versión del parser.
3. `./TestCPP archivo.cc` una ver actualizado el archivo de Test con el nuevo parser podemos pasarle un archivo para que lo procese y tendremos no

sólo la salida tradicional, sino también la salida del debug en la consola.

Los siguiente son ejemplos de salidas exitosas y fallidas:





Siguiendo el *trace* del debug

Una vez tenemos el archivo `ParCPP.info` y la salida del debug tenemos toda la información sobre los pasos que ha hecho el Parser. Pero para poder comprender cómo actuaron las distintas reglas de la gramática, podemos combinar ambos para seguir el trace de debug.

La salida del debug nos indica qué estado se está visitando en cada instante y qué acción se tomó en cada momento. Esto lo podemos *seguir* en el archivo de `ParCPP.info` para poder observar qué reglas son las que están enumeradas en la salida del debug.

Nota: La única información omitida en la salida del debug es que una vez que un estado termina de efectuarse (realiza la acción correspondiente al siguiente token a procesar), si no se mueve a un nuevo estado, vuelve al estado anterior. Una vez en el estado anterior, la única sección de acciones que se pueden realizar es la de los **go-to**.

Es decir: Si pasamos del estado 0 al 9 y en el 9 ya realizamos una acción de reduce (que cambia lo que hay en el tope del stack), en el trace se vuelve al estado 0 y se evaluan los *go-to* dependiendo de lo que haya quedado en el stack para saber el siguiente estado a visitar.