Objetivos generales

- Aplicar conocimientos de analisis lexico y sintactico.
- Aplicar traducción dirigida por la sintaxis.
- Explorar herramientas modernas para el desarrollo web.

Objetivos específicos

- Combinar la funcionalidad de la herramienta Jison haciendo uso del entorno de ejecución NodeJs en aplicaciones reales.
- Utilizar atributos sintetizados para manejo de información durante una traducción.
- Utilizar una tabla de símbolos primitiva.

Descripción de la actividad

El analizador sintáctico LL es un analizador sintáctico descendente, que se forma a partir de un conjunto de gramáticas libres de contexto. En este analizador las entradas son analizadas de izquierda a derecha, y sus construcciones de derivación se hacen también de la misma forma. La premisa más importante del análisis sintáctico descendente (ASD) es que intenta encontrar entre las producciones de la gramática la derivación por la izquierda del símbolo inicial para una cadena de entrada, apoyándose en ocasiones de herramientas como las tablas de símbolos, pilas, análisis predictivo, recursividad entre otras que hacen cada vez más poderoso a un analizador de este tipo.

Para trabajar el análisis sintáctico descendente se debe realizar primeramente algunas operaciones para que la gramática sea LL las cuales son:

- ELIMINAR AMBIGÜEDAD: Para eliminar la ambigüedad se debe reescribir la gramática.
- ELIMINAR RECURSIVIDAD POR LA IZQUIERDA: Una gramática es recursiva por la izquierda si tiene un nodo Terminal a tal que existe una derivación A->Aα para alguna cadena . Es decir, por simple observación podemos identificar.
- Factorizar: Se trata de reescribir las producciones de la gramática con igual comienzo para retrasar la decisión hasta haber visto lo suficiente de la entrada como para elegir la opción correcta.

En este punto del curso el estudiante de Organizaciòn de lenguajes y compiladores 1 ya debe conocer las características más importantes de cómo trabajar con un analizador sintáctico descendente. Así que como estudiante de dicho curso se le pide poner en práctica los conocimientos adquiridos en la clase para poder desarrollar una herramienta que permita la costrucciòn de analizadores sintácticos descendentes básicos, y que a partir de ciertas entradas se pueda hacer uso de estos analizadores para evaluar si la gramática definida en el analizador sintáctico acepta la cadena sin ningún problema y puede formar un único árbol de derivación, o si por el contrario la gramática es ambigua y no pasa las restricciones para ser un analizador LL y que permite más de un árbol de derivación para ciertas entradas.

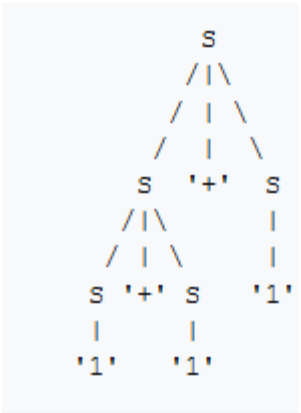
Interfaz gráfica

Toda la interfaz gráfica de la aplicaciòn debe ser desarrollada usando el framework Angular en cualquier versión (opcionalmente) o desarrollarla a partir de HTML y JavaScript con solo NodeJs, lo que se le pide en la primera parte es simplemente un área de edición de texto, donde se pueda escribir texto plano, o cargar un archivo con instrucciones que servirán para generar un analizador sintáctico, y de la misma forma en esta parte se debe contar con un mecanismo como un botón que permita enviar y procesar la gramática para que esta esté disponible y pueda ser utilizada para analizar cadenas entrantes.

La segunda parte de la aplicación consiste en poner a disposición del usuario todos los analizadores generados, y que se pueda seleccionar uno para hacer uso de él desplegando de nuevo un área de entrada de texto y carga de archivos para poder procesar las entradas y que al momento que el usuario decida evaluar una entrada se le permita visualizar si esta entrada es aceptada por el analizador sintáctico utilizado, y que del mismo modo se pueda confirmar a través de la generación de su árbol único de derivaciòn que la gramática no es ambigua, en caso contrario de que la entrada genere más de un árbol de derivación, estos se deben desplegar de igual modo logrando generar la mayor parte de escenarios posibles de árboles de derivación, y notificarle al usuario que su gramática es ambigua.

Puntos a tomar en cuenta en la interfaz de usuario:

- Los árboles de derivaciòn deben ser visibles al usuario de forma intuitiva, es decir se busca que al menos se trate de construir una representación básica del árbol con sus nodos y caminos posibles representado las entradas que se desean procesar. Por ejemplo para la entrad
- a 1+1+1 haciendo uso de la gramática $S \rightarrow S + S, S \rightarrow 1$, un árbol de derivación se vería así:



- De igual forma se debe contar con un área donde se notifiquen posibles errores léxicos, sintácticos y semánticos al momento de construir los analizadores sintácticos con el lenguaje que se especifica más adelante, y al momento de probar los analizadores generados tambien reportar posibles errores de las entradas a procesar.

Generaciòn de los analizadores sintácticos

Para esta parte fuera de la interfaz gráfica se debe desarrollar con la herramienta Jison, un analizador que permita leer una estructura donde es posible declarar tanto el analizador léxico como el sintáctico.

La estructura consta de la típica definición de una gramática libre de contexto con la 4-tupla N, T, P, S, Donde N representa los símbolos no terminales, T los terminales, P el conjunto de producciones y S el símbolo inicial, englobados en dos partes de un lenguaje llamado Wison de la siguiente manera.

```
#Esto es un comentario de línea

#Estructura Wison

Wison ¿
Lex {:
     /**
      Esto es un
      Comentario de bloque
    */

    # Declaración de terminales de la forma:
    # Terminal $_NOMBRE <- EXPRESIÓN ;

    Terminal $_Una_A     <- ‘a’ ;    # cualquier carácter alfanumérico por separado
    Terminal $_Mas       <- ‘+’ ;    # cualquier carácter especial por separado
    Terminal $_Punto     <- ‘.’ ;    # cualquier carácter especial por separado
    Terminal $_P_Ab      <- ‘(’ ;    # cualquier carácter especial por separado
    Terminal $_P_Ce      <- ‘)’ ;    # cualquier carácter especial por separado
    Terminal $_FIN       <- ‘FIN’;   # cualquier palabra reservada
    Terminal $_Letra     <- [aA-zZ]; # alfabeto completo en mayusculas y minusculas
    Terminal $_NUMERO    <- [0-9] ;  # Digitos del 0 al 9
    Terminal $_NUMEROS   <- [0-9]* ; # Estrella de Kleene para hacer 0 o n veces
    Terminal $_NUMEROS_2 <- [0-9]+ ; # Cerradura positiva para hacer 1 on veces
    Terminal $_NUMEROS_3 <- [0-9]? ; # reconoce la cláusula ? para hacer 0 o 1 vez
    Terminal $_Decimal   <- ([0-9]*)($_Punto)($_NUMEROS_2); # terminal combinado

:}
```

```
Syntax {{:
   # Declaración de no terminales de la forma
   # No_Terminal %_Nombre ;

   No_Terminal %_Prod_A;
   No_Terminal %_Prod_B;
   No_Terminal %_Prod_C;
   No_Terminal %_S;


   # Simbolo inicial de la forma
   # Initial_Sim %_Nombre ;

   Initial_Sim %_S ;

   #Todo símbolo no terminal debe ser declarado antes de usarse en las producciones
   # Las producciones son de la siguiente forma
   # %_Initial_Sim  <= %_Prod_A ... %_No_terminal_N o $_Terminal_N ... ;

   %_S <= %_Prod_A $_FIN ;
   %_Prod_A <= $_P_Ab %_Prod_B $_P_Ce ;
   %_Prod_B <= %_Prod_B %Prod_C | %_Prod_C ;
   %_Prod_C <= $_Una_A $_Mas $_Una_A ;
:}}

?Wison

# Fin de estructura Wison
```

La estructura Wison como se mencionó anteriormente está dividida en dos partes, la primera parte es la del bloque Lex de la parte léxica del analizador donde se debe tomar en cuenta lo siguiente:

- Se deben manejar las 3 cláusulas básicas para expresiones regulares (*, +, ?).
- Las declaraciones de caracteres y palabras reservadas deben ir entre comillas simples.
- Las estructuras [aA-zZ] y [0-9], son especiales y son las únicas a manejar en este lenguaje para definir secuencias de caracteres alfanumericos.
- Para la concatenación de expresiones se utilizan los paréntesis ‘()’.
- Los nombres de los no terminales deben iniciar con los símbolos %_

La Segunda parte es la Sintax, donde van todas las reglas sintácticas del analizador, donde se deben tomar en cuenta los siguientes puntos:

- Todos los no terminales deben ser declarados antes de usarse.
- Los nombres de los terminales deben iniciar con los símbolos $_
- Es obligatorio tener especificado el símbolo inicial para las producciones.

Lo que va despues del **carácter # en la misma línea** indica un **comentario de línea simple**, y lo que va dentro del **bloque multilínea con los caracteres** **/** ***/** indican **comentarios de bloque**. Esto debe tomarse en cuenta en toda la estructura Wison, al igual que toda palabra reservada de Wison como Lex, Syntax, Terminal etc, son case_sensitive, y los **separadores a tomar en cuenta de forma automática** para toda gramática definida en Wison son: saltos, tabulaciones, espacios en blanco y todo carácter especial que pueda venir tanto a la hora de definir la gramática como en las entradas a la hora de probar un analizador generado con Wison.

Importante

- Lenguajes de programación y Frameworks válidos: Angular, Nodejs, JavaScript, Typescript, HTML, CSS, Bootstrap.
- Usar la herramienta Jison para cualquier tipo de análisis léxico y sintáctico.
- Práctica obligatoria para tener derecho al siguiente proyecto.
- Las copias obtendrán nota de cero y se notificará a coordinación.
- Si se va a utilizar código de internet, entender la funcionalidad para que se tome como válido.

Entrega

La fecha de entrega es el día jueves 15 de abril a las 14:00. Los componentes a entregar en repositorio de github son:

- Código fuente (Proyecto completo con todas sus dependencias o módulos internos y externos)
- Manual técnico de la aplicación con un apartado del procedimiento para construir las gramáticas utilizadas y la explicación de cada una de ellas.
- Manual de usuario (Intuitivo).

Calificación

Pendiente de definir.