Universidad de San Carlos de Guatemala
Centro Universitario de Occidente
Division de Ciencias de la Ingeniería
Ingenieria en Ciencias y Sistemas

Manua tecnico proyecto 1: “LL-PARSE”

David Rodolfo Martinez Miranda
20632145

Manual tecnico

Requisitos minimos

- **Tener coneccion a internet.**
- **Ram minima de 4gb.**
- **Navegador Mozilla FireFox 85.0.2 (64bit).**
- **Version de NodeJS v8.10.0**
- **npm version 3.5.2**

En este manual se poroporciona la informacion para que se puedan hacer correcciones o simplemente saber como funcionan los cimientos principales para la aplicación pueda funcionar de manera adecuada. Se comenzaran por las gramaticas que reconoce la estructura de la entrada.

Reconocedores lexicos (Forma expresion regular)

La siguiente expresion regular reconoce la palabra reservada solicitud “Wison”.

Wison

La siguiente expresion regular reconoce la palabra reservada solicitud “lambda”.

Lambda

Las siguientes expresiones recnocen a patrones de un solo carácter que pertenece token 1 a 1 con lexemas y son los siguientes

¿ | | (|) | { | } | : | ; | ? | # | * | + | ← | <=

Los siguientes patrones para lexemas son los siguientes

La siguiente expresion regular reconoce el patron para comentario de linea.

"#"[^\n]*

La siguiente expresion regular reconoce el patron para nombre de terminal.

"$_"[^\n";<|) "]*

La siguiente expresion regular reconoce el patron para expresion lexica.

\'[^\n]*\'

La siguiente expresion regular reconoce el patron para nombre de no terminal.

"%_"[^\n";\| "]*

La siguiente expresion regular reconoce el patron para comentario de bloque.

"/**"[^*"/"]*"*/"

La siguiente expresion regular reconoce el patron para digitos.

"[0-9]"

La siguiente expresion regular reconoce el patron para abecedario.

"[aA-zZ]"

Las producciones son las siguientes.

S0 : S1 EOF

S1 : WISON INTERROGACION_ABIERTA S3 INTERROGACION_CERRADA WISON
| error INTERROGACION_ABIERTA
| error DOS_PUNTOS
| error WISON
| error LLAVE_CERRADA

S3 : S4 S12

S4 : LEX LLAVE_ABIERTA DOS_PUNTOS S5 DOS_PUNTOS LLAVE_CERRADA

S5 : S5 S6
| S6
| error TERMINAL
| error NOMBRE_LEX
| error ASIGNACION_LEXEMA

S6 : TERMINAL NOMBRE_LEX ASIGNACION_LEXEMA S7 PUNTO_COMA
|error PUNTO_COMA

S7 : S7 S21
| S21
| S11

S21 : S8

S8 : EXPRESION_DIGITO S9
| EXPRESION_LEXICA S9
| EPXRESION_ALFABETO S9
| NOMBRE_LEX S9

S9 : |
CERRADURA_CLENEE

| CERRADURA_POSITIVA
| INTERROGACION_CERRADA

S11 : S11 S22

S22 : PARENTESIS_ABIERTO S8 PARENTESIS_CERRADO

S12 : SYNTAX LLAVE_ABIERTA LLAVE_ABIERTA DOS_PUNTOS S13 DOS_PUNTOS LLAVE_CERRADA LLAVE_CERRADA;

S13 : S13 S14 | S14
| error NO_TERMINAL
| error NOMBRE_SYNTAX
| error PUNTO_COMA
| error ASIGNACION_PRODUCCION

S14 : S15 | S16 | S17;

S15 : NO_TERMINAL NOMBRE_SYNTAX PUNTO_COMA

S16 : INICIO_SIMBOLO NOMBRE_SYNTAX PUNTO_COMA

S17 : NOMBRE_SYNTAX ASIGNACION_PRODUCCION S18 PUNTO_COMA

S18 : S19

S19 : S19 S20
| S20

S20 : LAMBDA
| NOMBRE_LEX
| NOMBRE_SYNTAX
| OPERADOR_O

Definicion de gramatica

La estructura basica para reconocer el lenguaje es la siguiente.

- **Comentarios**, para definir comentario se tienen dos formas
  - #Esto es un comentario de línea, tiene que iniciar con # seguido de cualquier cosa menos saltos de linea
  - **/****
  - **Esto es un**
  - **Comentario de bloque**

- ***/**
  - **Esta** definido por el inicio de /** seugido de cualquier cosa menos / o *, y termina con */

- **Inicio de estructura** para iniciar la estrucura se debe seguir el siguiente patrones

Wison ¿

?Wison

Dentro de la estrucutura pueden venir comentarios como tambien afuera de el, para hacer definiciones lexicas se utiliza la siguiente estructura.

- **Definiciones lexicas**

Lex {:

:}

Para definir una expresion regular se debe de seguir la siguiente sintaxis

Terminal $_Una_A <- 'a' ;

Terminal define que es una terminal el nombre que se le de tiene que se **$_** seguido de cualquier cosa menos espacios ni saltos de linea, para asignarle un valor se tiene lo siguiente:

- Literal, viene entre comillas ‘literal’
- Caracter, viene en comillas ‘a’
- Alfabeto [aA-zZ]
- Digito [0-9]

Estas puede ir acompanadas de las clausuras siguientes

- Clenee * (cero o mas veces)
- Cerradura positiva + (una o mas veces)

- Cerradura ? (Una o cero veces)

Las terminales que se definan en esta area seran utilizadas en el analisis sintactico que se define mas adelante.

Analisis sintactico

Para hacer el analisis sintactico se define dentro del siguiente bloque:

Syntax {{:

:}}

Declaracion de No terminales, para declarar no termianles se debe seguir la siguiente sintaxis.

No_Terminal %_Prod_A;

Con No_Terminal se declara el no terminal, y se le asigna el nombre con **%_** seguido de cualeuir cosa menos espacios o saltos de linea.

Declaracion incio simbolo, para indicar con que simbolo se inicia se usa la siguiente sintaxis

Initial_Sim %_S ;

Declaracion de una produccion, para declarar una produccion se utiliza la siguiente forma.

%_Initial_Sim <= %_Prod_A ... %_No_terminal_N o $_Terminal_N

Se indica que produccion y lo que va a derivar, ademas puede agregar mas subproducciones utilizando

Todo esto anterior para el analisis de entrada.

Clases que son la base para el analisis

Las siguientes clases son las que ayudan a hacer los cimientos para este proyecto1

- **Clase Solicitudes**

Esta clase es la encargada de recibir todos los datos que se hacen en el jison, esta los interpreta y hace los calculos necsarios para realizar las operaciones mas adelante, esta clase recibe como parametro de constructor lo siguiente.

- listadoTerminales:any[]
- listadoNoTerminales:any[]
- listadoProducciones:any[]
- listadoPosicionesNuevasProd:any[]
- simboloInicial:any,listadoErrores:any[]

Estos son instanceados y luego procesados en el mismo constructor, por otros metodos que son importados a esta clase.

- El listao de terminales hace referencia a los terminales y a su expresion.
- El listado noTerminales, son todos los no terminales que vienen, existe una clase que se llama noTerminals, esta es la encargada de agregar cada no terminal, agregar las producciones, poder calcular los primeros y siguientes, luego son seteados a las variables propias de la clase.
- Simbolo inicial es para poder saber quien es el que inicia las producciones.

Clase Manjeador Primeros

Recibe como parametros a auxiliar produccion, y nombres noTerminales

constructor(producciones: AuxiliarProduccion[], nombresNoTerminales: String[])

El listado de AuxliarProduccion, son las producciones de cada no terminal, el listado de nombresNoTerminales es para poder agregar cada primerso con los datos correspondienetes.

Esta clase es la encargada de calcular los primeros de cada no terminal, los primeros no directos como los indirectos tambien, tiene dos funciones que se encargar de esto las cuales son:

- **calcularPrimerosFase1()**

Esta es la encargada de calcular los primeros inmediatos de cada una de las producciones, y luego agrega los datos correspondientes a los pirmeros.

- **private calcularPrimerosFase2()**

Esta funcion es la encargada de calcular los primeros no directos de cada produccion y los agrega a los datos de primeros según corresponda.

La clase ManjeadorSiguientes

Se instancea con los siguientes parametros:

- nombres: String[]
- auxiliaresProd: AuxiliarProduccion[]
- primeros: Primeros[],
- simboloInicial: String

- El listado de nombres es para poder crear cada no terminal relacionado uno a uno con sus siguientes.
- Auxiliar produccion es para recorrerlo y calcular los siguientes.
- Los primeros es para calcular los siguientes en caso se utilice.
- El simbolo inicial es para agregar el token de aceptacion al simbolo inicial.

Tiene dos funciones que calculan los siguientes.

- **CalcularSiguientes()**

Calcula los siguientes inmediatos de cada funcion sin que se haga lambda su siguiente inmediato, se llama a esta funcion luego de crear la clase, en el mismo constructor.

- **CalcularSiguientesFase2()**

Agrega los primeros del siguiente en caso que se haga lambda el siguiente inmediato, asi hasta encontrar el siguiente no terminal mas proximo, llama cuando terminal la primer fase que es CalcularSiguientes().

- **Clase tabla**

Esta es la encargada de recibir todos los datos para crear las tablas correspondientes a cada situacion, eso se hace desde otra clase llamada ManejadorTabla.

- **ManejadorTabla**

Es la encargada de procesar los datos y convertirlos a arreglos tipo any para poder ejecutar estos arreglos desde las tablas de reportes, donde se muestra la infromacion desde un ejs.

Las demas clases son subclases de estas ya menciondas, solo agregan atributos correspondientes para su uso en el momento necesario.

---------- Fin Manual Tecnico ----------