ISIS-1106-1 Lenguajes y Máquinas
Taller 1
Fecha: 14 de agosto de 2015

- Esta prueba es INDIVIDUAL.
- El intercambio de información con otro estudiante está terminantemente prohibido.
- Cualquier irregularidad con respecto a estas reglas podría ser considerada fraude.

IMPORTANTE: Soy consciente de que cualquier tipo de fraude en los exámenes es considerado como una falta grave en la Universidad. Al entregar este trabajo por Sicua+ doy expreso testimonio de que este trabajo fue desarrollado de acuerdo con las normas establecidas. Del mismo modo, aseguro que no participé en ningún tipo de fraude.

1. Expresiones Aritméticas

El archivo .zip adjunto contiene el proyecto java del probador de parsers. Para este ejercicio, se va a modificar el parser (NuevoParser.jj) que se encuentra en el paquete newParser. Este parser reconoce expresiones aritméticas. La gramática que está implementada es la siguiente:

$$S \rightarrow E\ ;$$
$$E \rightarrow T\{(+ \mid -)T\}$$
$$T \rightarrow F\{(* \mid /)F\}$$
$$F \rightarrow [-](\texttt{CONSTANT} \mid (E))$$

1.1. Fracciones

Se quiere agregar fracciones. Es decir, además de números se pueden operar fracciones. Se debe agregar una definición de token para fracciones (debe llamarlo `frac` . Una fracción es una cadena de la forma

$$< \texttt{NUMERADOR} : \texttt{DENOMINADOR} >$$

donde tanto numerador como denominador son enteros (no son expresiones) con o sin signo. Además el denominador no puede ser cero. Los corchetes forman parte de la descripción textual de la fracción. Es un sólo Token!
Por ejemplo:

- `<234:208>`

- `<233234:-208>`
- `<-1:+98>`

Además de la defición del token, deben modificar la regla en javaCC que corresponde al noterminal F.

$$F \rightarrow [-](\texttt{frac} \mid \texttt{CONSTANT} \mid (E))$$

1.2. Notación científica

Se debe modificar la definición del token `CONSTANT` para que se puedan crear enteros usando la notación científica. Ejemplos de cadenas de enteros definidos en notación científica son los siguientes:

- `25e10`
- `-233E8`
- `4e10`

Tenga en cuenta que para que los números sean enteros, la parte exponencial debe ser positiva.

1.3. Números complejos

Se quieren agregar números complejos a la grámatica.
Se debe agregar una definición de token para números complejos (debe llamarlo `COMPLEX`).
Un número complejo es una cadena de la forma:

`PARTE_REAL , PARTE_IMAGINARIA`**`i`**

Para simplificar el taller, la parte real y la parte imaginaria solo aceptan enteros (no son expresiones) con o sin signo. la parte real y la parte imaginaria de los números complejos también pueden escribirse en notación científica.
Por ejemplo:

- `25E5,4i`
- `-233,6E2i`
- `0,-10i`

Además de la defición del token, deben modificar la regla en javaCC que corresponde al noterminal F.

$$F \rightarrow [-](\texttt{frac} \mid \texttt{COMPLEX} \mid \texttt{CONSTANT} \mid (E))$$

1.4. Uso de arreglos y matrices

En la sección de análisis sintáctico agregue una regla para que una expresión también pueda ser una variable o un acceso acceso a una celda de un arreglo o una matríz.
Una variable es simplemente un nombre. Un acceso a una celda de un arreglo o una matríz es un nombre seguido de uno más expresiones entre corchetes cuadrados.
Los siguientes son ejemplos:

- `myarr[3][x+3]`
- `myarr[3][3][4][7][99]`
- `myarr[3][3+7*8]`
- `myarr[yourArr[3]*5][789+AskWhy[12][9*2-XW[5]]][3+7*8][90]`

Para esta también se modifica la regla para F.

$$F \rightarrow [-](V \mid \texttt{frac} \mid \texttt{COMPLEX} \mid \texttt{CONSTANT} \mid (E))$$

Usted debe definir la regla para V: variables o acceso a arreglos. Para esto, debe agregar un terminal para los nombres. Un nombre es una cadena alfanumérica que comienza con una letra.
Note que en este caso, lo que va entre corchetes cuadrados no pordría ser una fracción ni un complejo. Necesitaría definir reglas para expresiones que no puedan ser fracciones ni complejos.
El parser resultante deberá aceptar cadenas así:

- `myarr[3][x+3] * <234:208>+ 89 * (xyz + abc)`
- `myarr[3][3][4][7][99] - 890 + ( <234:208>* <-234:208>)`
- `5E10,5i + walt[cook+simple[44/someArr[3]]] + <234:208>`

2. Redes de Petri

Suponga que queremos definir un lenguaje para describir REDES DE PETRI. A continuación se define un lenguaje y su gramática. Implemente el analizador sintáctico para este lenguaje en JavaCC e intégrelo al probador de parsers. En la wiki encuentran las instrucciones de cómo hacer esta integración.

- Comienza con la palabra reservarda PN seguida por un nombre. Luego aparece lo siguiente:
- Una lista de sitios: La palabra PLACES seguida por una secuencia de especificaciones de sitio separadas por comas. La especificación de un sitio es un nombre seguido por su capacidad entre parentesis. La capacidad es un número entero mayor o igual a cero.
- Una lista de transiciones: La palabra TRANSITIONS seguida una secuencia de especificaciones transición, separadas por punto y coma.
- La especificación de una transición es su nombre seguido por suspredecesores y sus sucesores

- Los predecesores se denotan con la palabra distinguida PRED, seguida por la una lista de sitios con sus correspondientes pesos entre paréntesis. Esta lista no puede ser vacía. Si no hay predecesores, no aparece la palabra PRED
- Los sucesores se denotan con la palabra distinguida SUCC, seguida por la una lista de sitios con sus correspondientes pesos entre paréntesis. Esta lista no puede ser vacía. Si no hay predecesores, no aparece la palabra SUCC.
- Una transición debe tener o sucesores o predecesores o ambos.
- Termina con la palabra NP

Los símbolos terminales son: `PN`, `NP`, `PLACES`, `TRANSITIONS`, `PRED`, `SUC`,‘‘`,`’’, ‘‘`;`’’, ‘‘`(`", ‘‘`)`", ‘‘`{`", "`}`", `nomb` para representar nombres y `CONSTANT` para denotar números.
Un ejemplo de una descripción se muestra a continuación:

```
PN myNet
  Places Waiting ( 100) , Ready (1), Working(1), Count(1000)
  Transitions
      Arrive
          SUCC Waiting(1);
      Start
          PRED Waiting(2) Ready(1)
          SUCC Ready(1);
      Finish
          PRED Working(1)
          SUCC Ready(1) Count(2)
NP
```