# Teoría de la Computación

Práctica No. 3: Prueba de membresía para Lenguajes Libres de Contexto

**Objetivo de la práctica**

Implementar el algoritmo de *Cocke*, *Younger* y *Kasami* (**CYK**) para hacer la prueba de membresía de cadenas, para *Lenguajes Libres de Contexto* (**LLC**).

**Fecha límite de entrega**

Miércoles 18 de Mayo de 2011.

**Desarrollo**

El alumno deberá desarrollar una aplicación que use el algoritmo **CYK** para hacer la *prueba de membresía* de cadenas de caracteres al lenguaje *quest*, el cual se describe mediante la gramática de la figura 1. Es decir, deberá determinar si una cadena $w \in \Sigma^*$ está en el lenguaje $\mathcal{L}$(*quest*).

Para ello, su programa deberá leer una cadena y responder afirmativamente si el algoritmo tiene éxito ó negativamente en el caso contrario. El algoritmo **CYK** se muestra más adelante.

$$\begin{aligned} E &\rightarrow PqEeE \mid P \\ P &\rightarrow PoC \mid C \\ C &\rightarrow CaT \mid T \\ T &\rightarrow nP \mid lEr \mid x \end{aligned}$$

**Figura 1**. La gramática *quest*

**El algoritmo CYK**

Asuma una gramática $G = (N, \Sigma, S, P)$ sin producciones $\lambda$ que está en la *Forma Normal de Chomsky* (**FNC**). Si $w \in \Sigma^*$ es una cadena de longitud $n$, entonces $w_{ij}$ es la *subcadena* de $w$ que comienza en la posición $i$ de $w$ ($i$ es un valor entre $1$ y $n$) y que tiene longitud $j$ ($j$ es algún valor apropiado entre $1$ y $n - i$). El algoritmo **CYK** (figura 2) se basa en el cálculo dinámico de conjuntos de *símbolos no terminales* $N_{ij}$, tales que $A \in N_{ij}$ si y solo si $A \Rightarrow^* w_{ij}$.

1. Para todo $i = 1, \ldots, n$ hacer $N_{i1}$ igual al conjunto $\{A \mid A \Rightarrow w_{i1}\}$
2. Para todo $j = 2, \ldots, n$ hacer lo siguiente:
3. Para todo $i = 1, \ldots, n - j + 1$ hacer lo siguiente:
4. Inicializar el conjunto $N_{ij}$ como vacío
5. Para todo $k = 1, \ldots j - 1$ hacer lo siguiente:
6. Para todo $B \in N_{ik}$ y todo $C \in N_{i+k, j-k}$ hacer lo siguiente:
7. Agregar $A$ al conjunto $N_{ij}$, para cada producción de la forma $A \rightarrow BC$
8. Generar *verdadero* si $S \in N_{1n}$ o *falso* en el caso contrario

**Figura 2**. El algoritmo **CYK**

**Lineamientos generales para la realización de su solución**

1. **Reescritura de la gramática**. Un requisito para la aplicación del algoritmo **CYK** es que la gramática objetivo esté en la **FNC**. La gramática *quest* no se presenta de esta manera. Luego, el primer paso para resolver esta práctica consistirá en reescribir la gramática *quest* en la **FNC**. Deberá mostrar este procedimiento así como el resultado final.

2. **Diseño de la implementación**. Estudie las siguientes recomendaciones. Cualquier solución alternativa deberá ser señalada y justificada.

   - Se sugiere usar números enteros para representar los símbolos de su gramática. Por ejemplo, los símbolos generadores podrán codificarse como números enteros mayores de 200. Los símbolos terminales (letras minúsculas del alfabeto) podrán representarse con el valor ASCII que les corresponde (valores numéricos comprendidos entre 97 y 122 —es decir entre `'a'` y `'z'`—).
   - Puede usar la siguiente estructura para la representación de las reglas sintácticas. La ausencia de algún número puede representarse mediante el valor 0:

```
typedef struct {
  int generador;
  int primero;
  int segundo;
} regla;
```

*Representación gramatical*. Para la representación de la gramática (conjunto de reglas sintácticas) podrá utilizar una representación vectorial. Por ejemplo, puede utilizar la clase `vector` [1] de la STL (*Standard Template Library*) de C++. La declaración de su gramática podría tener la siguiente forma:

```
typedef std::vector<regla> gramatica;
```

*Representación de la tabla dinámica*. Asuma la siguiente tabla dinámica que se genera para la prueba de alguna cadena de longitud 3:

| | | $j$ = 1 | 2 | 3 |
|---|---|---|---|---|
| $i$ = 1 | | $N_{11}$ | $N_{12}$ | $N_{13}$ |
| 2 | | $N_{21}$ | $N_{22}$ | |
| 3 | | $N_{31}$ | | |

Cada uno de los conjuntos de la tabla puede representarse mediante la clase `set` [2] de la STL de C++. Por ejemplo, puede definir el siguiente tipo de dato:

```
typedef std::set<int> conj;
```

A su vez, cada una de las filas de la tabla puede representarse convenientemente mediante un *mapeo*, usando la clase `map` [3] de la STL de C++. Por ejemplo, el tipo de la fila puede definirse de la siguiente manera:

```
typedef std::map<int,conj> fila;
fila mifila;
```

De esta manera será posible relacionar algún valor de la variable $j$ con el conjunto (columna) correspondiente, por ejemplo mediante `mifila[2]`. Similarmente, puede utilizarse un mapeo para generar la lista de filas. Por ejemplo, de la siguiente manera:

```
typedef std::map<int,fila> tabla;
tabla N;
```

Así, será posible relacionar algún valor de la variable $j$ con su fila correspondiente, por ejemplo como `N[2]`. Luego, ya que esta operación hace una referencia a un objeto tipo `fila`, podrán realizarse operaciones como `N[2][2]` para referirse al conjunto $N_{22}$.

*Operaciones básicas*. Deberá implementar una función de nombre `addSymbols`. Esta función toma como argumentos las referencias de tres conjuntos: $N_{ij}$ (`dest`), $N_{ik}$ (`src1`) y $N_{i+k,j-k}$ (`src2`) y realiza una combinatoria de los elementos de $N_{ik}$ con los elementos de $N_{i+k,j-k}$ (línea 6 del algoritmo).

```
void addSymbols(conj &dest, conj &src1, conj &src2);
```

Para cada una de estas combinaciones $BC$ ($B \in N_{ik}$ y $C \in N_{i+k,j-k}$) habrá que implementar una búsqueda de producciones de la forma $A \rightarrow BC$ a lo largo de la gramática, y para cada una de ellas agregar el símbolo $A$ al conjunto $N_{ij}$ (línea 7). Esta operación será realizada mediante la función `buildSet`:

```
void buildSet(conj &s, int b, int c);
```

Esta función deberá recibir como argumentos una referencia al conjunto `dest` de la función `addSymbols` (el argumento `s`) así como el criterio para la búsqueda de producciones (los argumentos `b` y `c` para los símbolos $B$ y $C$ respectivamente, que fueron generados en `addSymbols`). Adicional a la anterior, deberá implementar otra versión de la función `buildSet`:

```
void buildSet(conj &s, int t);
```

Esta forma de `buildset` deberá localizar en la gramática todas aquellas producciones que solo generen al terminal `t`, para agregar el símbolo generador correspondiente al conjunto $N_{i1}$ (argumento `s`). Esta función implementa la operación de inicialización del algoritmo (línea 1). Para el mantenimiento de la tabla dinámica, deberá implementar algunas otras funciones. Las principales se mencionan a continuación:

- Una función de nombre `isMember`, que determine si algún símbolo es miembro o no de un conjunto. Esta función recibe como argumentos una referencia del conjunto objetivo (`s`) y un símbolo gramatical (`a`). Genera como resultado **`true`** si el símbolo `a` es miembro del conjunto objetivo ó **`false`** en el caso contrario.

  ```
  bool isMember(conj &s, int a);
  ```

- Una función de nombre `analize`, que implementa propiamente el algoritmo **CYK**. Recibe como argumentos una referencia a una gramática (`g`) y la cadena (`w`, objeto de la clase `string` [4]) que va a ser analizada. Esta función genera como resultado **`true`** si la prueba de membrecía tiene éxito, ó **`false`** en el caso contrario.

  ```
  bool analize(gramatica &g, string &w);
  ```

2. **Lenguaje de implementación**. Se exhorta al uso del lenguaje ANSI C/C++ como lenguaje de desarrollo. Atienda los siguientes puntos:
   a) El uso de interfaces gráficas y/o entornos de desarrollo será opcional, y su uso se hará responsabilidad del alumno. Cualquier inconveniente originado por el uso de ellos que impida la revisión del proyecto será causa de la anulación de la misma.

b) No podrá hacer uso de herramientas auxiliares ó librerías especiales que suplanten las técnicas requeridas para el desarrollo de su implementación. Cualquier solución alterna deberá ser señalada y justificada, y no deberá alterar en ningún caso los requisitos a evaluar. El no apego a este punto podrá será causa de la anulación de su proyecto.

3. **Asesoría para su elaboración**. El alumno podrá asesorarse en todo momento con el profesor durante el desarrollo de su proyecto, empleando para ello preferentemente los horarios destinados a este fin.

## Requerimientos para la evaluación de la práctica

Los siguientes requerimientos serán solicitados para la evaluación de esta práctica:

1. **Verificación de la aplicación**. Se hará una corrida de prueba, durante la cual se proporcionará una cadena de entrada arbitraria a su programa. Se verificará que la respuesta dada por el programa sea correcta. Aunque no es requerido, se considerará una apreciación favorable el que su programa muestra en pantalla el contenido final de la tabla dinámica.

2. **Comprensión del problema**. El alumno deberá tener una clara comprensión técnica y teórica del problema, la cual podrá ejercitar durante el desarrollo de esta práctica. Este punto será evaluado mediante una breve indagatoria que el profesor hará durante la verificación de la aplicación así como la revisión del código fuente.

3. **Documentación del proyecto**. Deberá elaborarse un reporte de práctica de acuerdo a lo que se expone en el *Manual para la elaboración de los Reportes de Práctica*. Para esta práctica deberá informar:
   a) Análisis, depuración y puesta de la **FNC** de la gramática *quest*.
   b) Cuales estructuras de datos empleadas y como las utilizó.
   c) Algoritmos implementados (búsqueda de símbolos, actualización de conjuntos, prueba de membresía).

   La entrega del reporte impreso será un requisito necesario para dar inicio a la revisión de su proyecto.

## Ejemplo

Una cadena de la forma `xqxex` debería provocar un resultado positivo en su programa, mientras que una cadena como `rxl` debería provocar un resultado negativo.

## Referencias


[1] http://www.cplusplus.com/reference/stl/vector
[2] http://www.cplusplus.com/reference/stl/set
[3] http://www.cplusplus.com/reference/stl/map
[4] http://www.cplusplus.com/reference/string/string