2014

Tarea Programada 2

IMPLEMENTACIÓN DE INTÉRPRETE DE LENGUAJE DE PROLOG

INSTITUTO TECNOLÓGICO DE COSTA RICA

KENNETH CÉSPEDES GARBANZO

CAROLINA ROBLES URIBE

MARIANA MORA ARAYA

YAXIRI AZOFEIFA GARCIA

TABLA DE CONTENIDO

DESCRIPCIÓN DEL PROGRAMA

Un lenguaje de programación es un lenguaje formal diseñado para expresar procesos que son llevados a cabo por un computador. Pueden ser usados para crear programas con un fin específico. Existen diferentes paradigmas que clasifican estos lenguajes, en este caso se deberá implementar uno perteneciente al paradigma lógico llamado Prolog. En el desarrollo del programa se deberá simular el comportamiento de áreas específicas de Prolog. Por lo que se deberá implementar una base de conocimientos, que deberá ser administrada por el usuario de manera que pueda agregar la cantidad ilimitada de hechos o reglas que este desee. De la misma manera, un motor de inferencia que se encargara de revisar las reglas de inferencia solicitadas y contestar al usuario según así sea la respuesta requerida. Modo consulta es otra característica que deberá estar implementada en el programa, en donde el usuario interactuara con el sistema por medio de consultas. En el caso de que una regla tenga antecedentes, deberán validar los antecedentes antes de poder responder. Se deberá implementar backtracking, para el proceso de resolución de metas. Tanto en modo consulta como a la hora de ingresar hechos a la base se deberá de proceder con el análisis léxico y sintáctico correspondientes a la estructura permitida por Prolog. El programa será implementado en el lenguaje C++.

DISEÑO DEL PROGRAMA

MÉTODOS ESTUDIADOS

Análisis Léxico

La fase de rastreo (scanner), tiene las funciones de leer el programa fuente como un archivo de caracteres y dividirlo en tokens. Los tokens son las palabras reservadas de un lenguaje, secuencia de caracteres que representa una unidad de información en el programa fuente. En cada caso un token representa un cierto patrón de caracteres que el analizador léxico reconoce, o ajusta desde el inicio de los caracteres de entrada. De tal manera es necesario generar un mecanismo computacional que nos permita identificar el patrón de transición entre los caracteres de entrada, generando tokens, que posteriormente serán clasificados. Este mecanismo es posible crearlo a partir de un tipo específico de máquina de estados llamado autómata finito

El analizador léxico opera bajo petición del analizador sintáctico devolviendo un componente léxico conforme el analizador sintáctico lo va necesitando para avanzar en la gramática. Los componentes léxicos son los símbolos terminales de la gramática. Suele implementarse como una subrutina del analizador sintáctico. Cuando recibe la orden "obtén el siguiente componente léxico", el analizador léxico lee los caracteres de entrada hasta identificar el siguiente componente léxico.

Análisis sintáctico

El análisis sintáctico es un análisis a nivel de sentencias, y es mucho más complejo que el análisis léxico. Su función es tomar el programa fuente en forma de tokens, que recibe del analizador léxico, y determinar la estructura de las sentencias del programa. Este proceso es similar a determinar la estructura de una frase en castellano, determinando quien es el sujeto, predicado, el verbo y los complementos. El análisis sintáctico agrupa a los tokens en clases sintácticas (denominadas no terminales en la definición de la gramática), tales como expresiones, procedimientos, etc. El analizador sintáctico o parser obtiene un árbol sintáctico (otra estructura equivalente) en la cual las hojas son los tokens, y cualquier nodo que no sea una hoja, representa un tipo de clase sintáctica (operaciones). Los árboles sintácticos se construyen con un conjunto de reglas conocidas como gramática, y que definen con total precisión el lenguaje fuente. Al proceso de reconocer la estructura del lenguaje fuente se conoce con el nombre de análisis sintáctico (parsing). Hay distintas clases de analizadores o reconocedores sintácticos, pero en general se clasifican en 2 grandes grupos: A.S. Ascendentes y A.S. Descendentes. La principal tarea del analizador sintáctico no es comprobar que la sintaxis del programa fuente sea correcta, sino construir una representación interna de ese programa y en el caso en que sea un programa incorrecto, dar un mensaje de error. Para ello, el analizador sintáctico (A.S.) comprueba que el orden en que el analizador léxico le va entregando los tokens es válido. Si esto es así significará que la sucesión de símbolos que representan dichos tokens puede ser generada por la gramática correspondiente al lenguaje del código fuente.

Bactracking

Backtracking (o búsqueda atrás) es una técnica de programación para hacer búsqueda sistemática a través de todas las congelaciones posibles dentro de un espacio de búsqueda. Para lograr esto los algoritmos de tipo Backtracking construyen posibles soluciones de manera sistemática. En general, dado una solución candidata S:

- Verifican si S es solución. Si lo es, hacen algo con ella (depende del problema).
- Construyen todas las posibles extensiones de S, e invocan recursivamente al algoritmo con todas ellas

A veces los algoritmos de tipo backtracking se usan para encontrar una solución, pero otras veces interesa que las revisen todas.

DECISIONES DEL DISEÑO

A continuación de especifican decisiones tomadas para el diseño usado en el programa:

- Se desarrolló en el lenguaje de Programación C++ porque consideramos que la estructura del manejo de este lenguaje se podría aplicar con los conocimientos obtenidos por el lenguaje C, además de su aporpiada implementación del software.
- Primeramente se utilizo la funcion para incluir a la Base de Conocimientos con la lectura de <define> para iniciar la inclusión y </define> para terminar. Y luego de esto inicia el método consulta a la Base de Conocimientos.
- Se diseñó el programa de manera tal que pudiera correrse desde consola solamente.
- La implementación del manejo de validaciones léxicas y sintácticas fueron implementadas de manera manual sin uso de librerías mediante el manejo de tokens y strings.

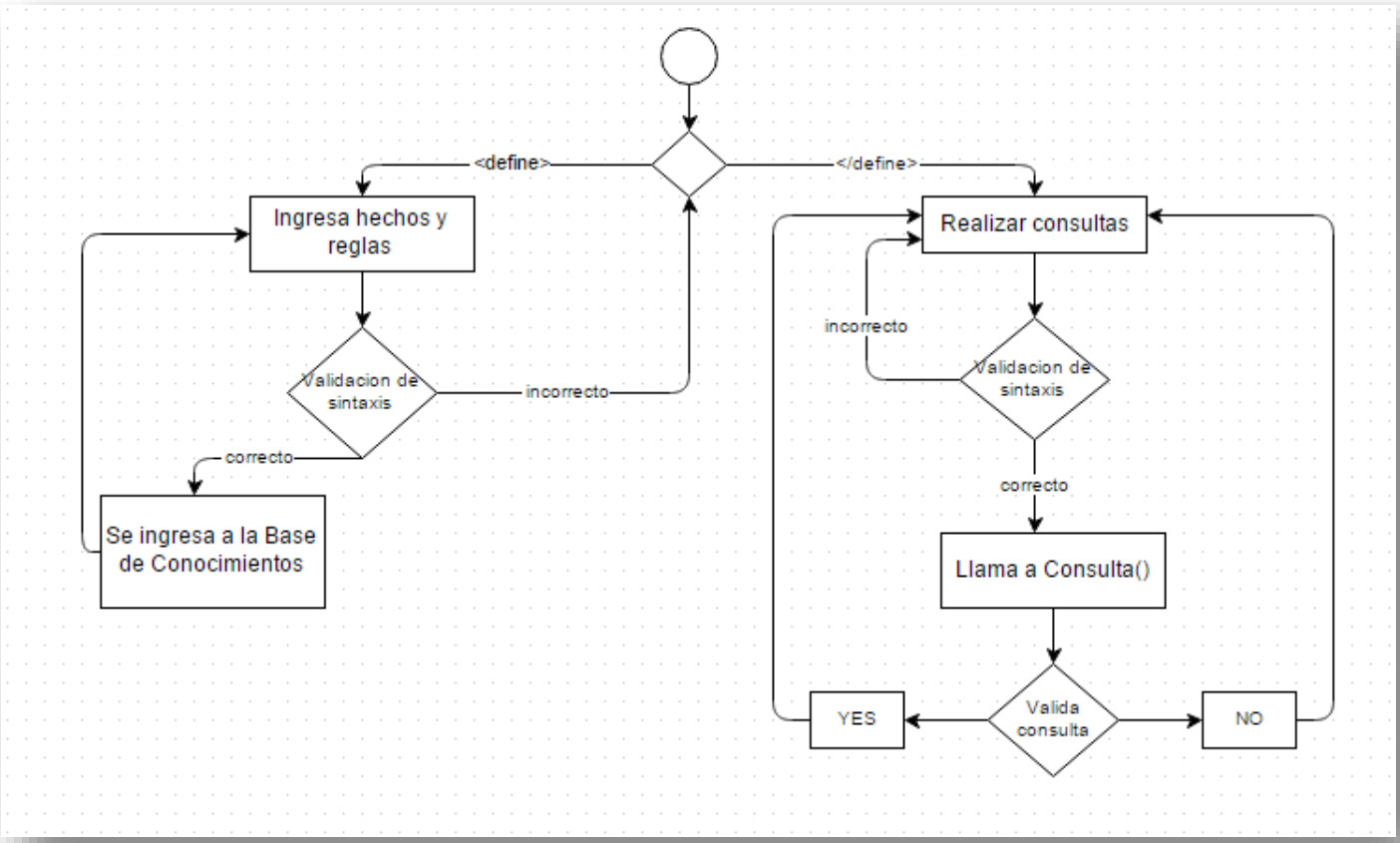
ALGORITMOS USADOS

Analizador

Nombre de la Función	***Funcionamiento***	***Valor de retorno***
Programa()	Es el que inicia todo el procedimiento de análisis. El cual agarra el primer token ingresado por la regla o hecho y llama a **lp()**.	No existe valor de retorno.
Lp()	Para iniciar llama a **prop()**, cuando prop() no funciona más llama al siguiente token y analiza si la persona pone </define> para terminar el análisis o llama a **lp1()**.	El valor de retorno será al archivo **BaseConocimiento.txt** con el análisis aceptado.
Prop()	Llama a **izq()**, cuando este no se cumple más identifica lo que se encuentra luego de encabezado. Si este es **":-"** agarra el siguiente token y llama a **asig()**, y luego de terminar este, verifica si el siguiente token es un punto para terminar con la regla correctamente. De lo contrario, que sea diferente de **":-"** ósea refiriéndose a un hecho para terminar con **"."**, nada más verifica éste y termina.	El valor de retorno será al archivo **BaseConocimiento.txt** con el análisis aceptado.

Nombre de la Función	*Funcionamiento*	*Valor de retorno*
Asig()	Llama a **der()** al inicializar, luego identifica que el token que se encuentre luego de der() sea diferente de punto o coma llama a **asig()**.	El valor de retorno será al archivo **BaseConocimiento.txt** con el análisis aceptado.
Der()	Llama a las funciones **expa()** y luego a **izq()**.	No existe valor de retorno.
Izq()	Al inicializar llama a **id()**, al terminar esta función verifica el funcionamiento de los paréntesis utilizando la función **lt()** para la información que se encuentra dentro de ellos. Y si no hubiera parámetros termina allí la función.	El valor de retorno será al archivo **BaseConocimiento.txt** con el análisis aceptado.
Lt()	Llama a **termino()** y luego a **lt1()**.	No existe valor de retorno.
Expa()	Inicia llamando a **id()**, luego verifica si existe un **"is"**, si existe llama a **arit().**	El valor de retorno será al archivo **BaseConocimiento.txt** con el análisis aceptado.
Arit()	Llama a **termino()** y a**rit1()**.	No existe valor de retorno.
Arit1()	Verifica si lo que se encuentra un **"-"**, si es llama a **termino()** y luego a **arit1()**.	No existe valor de retorno.
Termino()	Verifica si es diferente de token es **"_"**, si es llama **id()** y a **num()**.	El valor de retorno será al archivo **BaseConocimiento.txt** con el análisis aceptado.
Id()	Verifica si el texto inicia con minúscula, esta regla se da especialmente para los encabezados de las reglas o los hechos.	El valor de retorno será al archivo **BaseConocimiento.txt** con el análisis aceptado.
Num()	Verifica si el texto inicia con minúscula, mayúscula o número, esta regla se da especialmente para los parámetros de las reglas.	El valor de retorno será al archivo **BaseConocimiento.txt** con el análisis aceptado.
Main()	Realiza el procedimiento principal, inicia la clase **analiza()** que contiene todas las clases anteriores y cierra el archivo **BaseConocimiento.txt** que contiene todas las reglas y hechos.	No existe valor de retorno.

DIAGRAMA LÓGICO



LENGUAJE DE PROGRAMACIÓN ESCOGIDO Y RAZONES DE ESTA ESCOGENCIA

El lenguaje de programación escogido fue C++. Las principales razones por esta escogencia fueron el aprendizaje de un nuevo lenguaje que tenía una gran relación con el lenguaje C, utilizado para la programación anterior. Además de la implementación de clases que facilitaría el manejo de la información. Y de acuerdo a las evaluaciones para esta tarea programada, el porcentaje extra ganado por su uso.

LIBRERÍAS USADAS

STRING.H

Es un archivo de la biblioteca estandar del lenguaje de programación C que contiene la definición de macros, constantes, funciones y tipos de utilidad para trabajar con cadenas de caracteres y algunas operaciones para la manipulacion de memoria. Las funciones para estas cadenas de caracteres solo se trabajan con conjuntos de caracteres ASCII.

STDLIB.H

Es un archivo de cabecera de la biblioteca estándar de propósito general del lenguaje de programación C, el cual es compatible con C++. Contiene los prototipos de funciones para la gestión de memoria dinámica, control de procesos y otras.

IOSTREAM

Es un componente de la biblioteca estándar del lenguaje de programación C++ el cual utiliza operaciones de entrada y salida de datos. El flujo de entrada y salida de datos en C++ no se encuentra definida dentro de la sintaxis básica y se provee por medio de librerías de funciones especializadas como iostream. Este define los siguientes objetos:

- **cin:** Flujo de entrada (que entra).
- **cout:** Flujo de salida (que sale).
- **cerr:** Flujo de error no almacenado.
- **clog:** Flujo de error almacenado.

Todos los objetos derivados de iostream hacen parte del espacio de nombres std. Por lo tanto se debe incluir dentro del código "using namespace std".

STDIO.H

Stdio.h significa "standard input.output header" (cabecera estándar E/S), es la biblioteca estándar del lenguaje de programación C, el archivo de cabecera que contiene las definiciones de macros, constantes, funciones y definiciones de tipos usados por varias operaciones estándar de entrada y salida de datos.

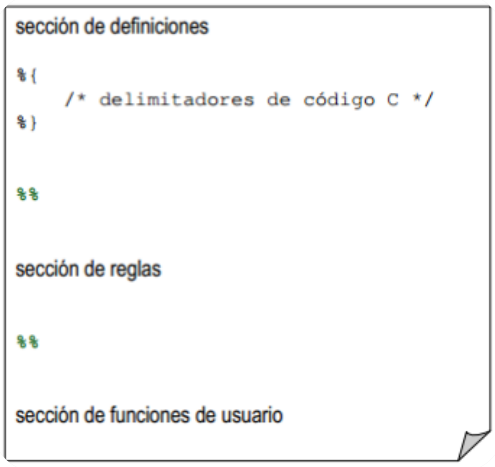
FSTREAM

Se utiliza especialmente para el manejo de archivos, mostrando las diferentes maneras para abrir un archivo.

LEX.YY.C

La estructura del fichero de especificación Lex/Flex se compone de tres secciones separadas por líneas que contienen el separador %%.

Código C encerrado entre líneas con los caracteres %{ y %} que se copia literalmente en el fichero de salida lex.yy.c antes de la definición de la función yylex(). Habitualmente esta sección contiene declaraciones de variables y funciones que se utilizan posteriormente en la sección de reglas así como directivas #include.



ANÁLISIS DE RESULTADOS

OBJETIVOS ALCANZADOS

- Se logró implementar las validaciones de reglas léxicas que revisa token por token antes de pasar al análisis sintáctico.
- Se logró la implementación de análisis sintáctico que se encarga de validar la estructura de lo ingresado por el usuario
- Se logró los almacenamientos de datos en la base de conocimientos.

OBJETIVOS NO ALCANZADOS

- No se logró ningún tipo de consultas.
- No se logró la implementación del backtracking.

MANUAL DE USUARIO

Se encuentra adjunto con el nombre Manual de Usuario.pdf.

CONCLUSIÓN GRUPAL

El proyecto programado 2, requirió de mucha investigación y análisis de comportamientos tanto del lenguaje Prolog como el manejo de datos de C++. Además se necesitó de empeño y tiempo para alcanzar objetivos.

Se implementaron muchos de los temas vistos en clase y se aumentó el conocimiento de la gran mayoría de estos. Se notó claramente con la investigación, los diferentes comportamientos de prolog por lo que era necesario para su implementación. Cabe mencionar que uno de las implementaciones más difíciles del proyecto fue el motor de inferencia y el manejo del backtracking.