INSTITUTO TECNOLOGICO DE COSTA RICA

DOCUMENTACION DE COMPILADOR SICTACTICO DE XHTML

IC 5701 COMPILADORES

ESCUELA DE COMPUTACION

INTEGRANTES

Andrés Ramírez Fuentes 201013880

Larissa Rivas Carvajal 201022184

FECHA DE ENTREGA

3 de junio del 2013

Contenido

Descripción del problema

El problema consiste en realizar un analizador sintáctico para la gramática de un xhtml reducido (que contenga algunos tags en específico), que pueda construir un árbol sintáctico a partir del parser que fue desarrollado en Flex, como además detectar errores que sean obtenidos en la fase tanto de análisis léxico como en el sintáctico. El compilador debe de ser capaz de construir un árbol que al finalizar la etapa del parser pueda imprimir el árbol de forma completa mostrando así de manera gráfica como este ha sido construido a través de reducciones de punteros.

Para la realización de este problema se deberá utilizar un generador de parsers, que en nuestro caso es bison que obtiene tokens del scanner que es realizado por flex.

Se deberá de construir una gramática correcta, que no presente ningún tipo de conflicto ya sea, reduce-reduce o shit-reduce o cualquier otro tipo que no permita el análisis sintáctico del mismo.

Además, a pesar de que los generadores sintácticos son capaces de resolver algunos tipos de conflictos, la construcción de la gramática tiene que ser correcta, sin *warnings* o errores presentes.

Dividir el análisis sintáctico en un archivo aparte, del resto de código, así como también código de C para la construcción del árbol, lo cual ayude a una visibilidad y claridad a la lectura del mismo.

Utilizar recursos suministrados por los generadores tales como variables que permitan la comunicación entre los analizadores, sin utilizar librerías que realicen funciones del análisis sintáctico ni ningún tema relacionado.

Los errores deben de ser impresos en el stderr y la salida del parser en el stdout. Se usará la redirección suministrada desde la consola, y no lectura de archivos, por lo que se requiere la construcción de un make que realice la unión de todos los ficheros necesarios para un correcto funcionamiento.

Diseño del programa

Tokens implementados en parser.y

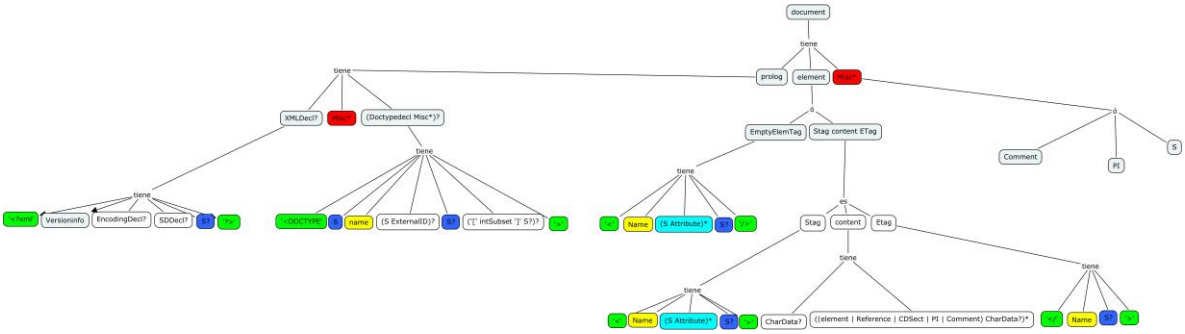
Terminales:

- T_ENTER T_SPACE = reconoce las lineas de enter y espacio.
- T_COMMENT_IN T_COMMENT_END = tokens de inicio y cierre de comentario.
- T_CDATA_I T_CDATA_F CONT_DATA = tokens de inicio y cierre de cdata.
- T_XMLDEC T_F_XMLDEC T_EQ T_ENCOD T_ENDID T_VNUMBER T_VS = tokens pertenecientes a la declaración del xml.
- T_DOCTYPE T_SYSTEM T_PULIC T_NAME T_ENDDOCTYPE T_PUBIDLIT T_SYSTLIT = okens pertenecientes a la declaración del doctype.
- T_ETAG T_TAG T_FTAG T_ESTAG T_CONTENT T_ATRIBUTO T_FIN_ETAG T_FIN_STAG T_CONTENIDO = tokens que son elementos de tags, tales como el <, >, </ entre otros.

No – terminales:

- Prolog Element Misc OPCIONESMisc = partes del documento principal.
- T_S T_SS = espacios.
- Doctype ExternalID = relacionado con elementos del doctype.
- XMLdec XMLdec_E VersionINfo EncoDecl = relacionado con elementos del xml.
- CONTENT OPCIONES SCTAG ESTAG EETAG T_ATRIB T_COMMENT_E ETAG = elementos del tag y relacionados.

Análisis del árbol sintáctico:



Para el diseño del programa buscamos las reglas que se describen en la documentación de la w3c que se encuentra en http://www.w3.org/TR/2000/REC-xml-20001006, la cual describe todas las reglas sintácticas para la construcción de un documento en formato xhtml.

Para el diseño de nuestro analizador sintáctico, se utilizó un subconjunto de reglas las cuales permiten validar el conjunto suministrado por parte del profesor como requisitos del programa.

Primeramente se buscan todas las reglas que se encuentran en la w3c que posean alguna de los tags que se deben de validar para este programa. Esto permitiría verificar que lo que se pide pueda ser construido en un documento de xhtml.

Como segundo paso, se construye una gramática de manera que todos los tokens que son recolectados por el scanner puedan pertenecer al scanner, y aquellos elementos de las reglas que no son recolectadas por el scanner, son eliminadas evitando así, construir una gramática que posee elementos inútiles, o que reciba elementos que no se encuentran en sus reglas sintácticas.

Para la implementación de las reglas sintácticas, todos los tokens son devueltos en forma de identificador los cuales están adecuadamente construidos con la sub-gramática elaborada en el punto anterior.

Cada uno de los elementos de la gramática sea este, terminal o no terminal, son convertidos a estructuras de C, los cuales representan los nodos del árbol, y que conforme son llamados, se realizan operaciones para que haga manejo de punteros y se cree la estructura del árbol.

Los nodos de cada árbol están constituidos de forma tal, que posean un puntero a un hijo, y un puntero a un hermano, además de guardar la información del tipo de nodo que se está creando como su contenido.

El tipo de nodo, se guarda para poder mostrarlo en la impresión del árbol, y el contenido, es todo aquello que haya leído el scanner, ya sea el nombre de un tag, o un símbolo, esto con la finalidad de poder hacer un análisis semántico en el tercer proyecto.

Las estructuras de los árboles se han creado en un fichero denominado clases.h que se implemente de forma separa y es incluida con un #include en archivo parser.y por motivos de facilidad en el manejo de mantenimiento y encapsulamiento. Además de las estructuras, en este archivo tenemos los métodos que son utilizados por el parser.y para creación de los nodos y para la impresión de árbol.

La construcción del árbol permite que el nodo solo posea un único hijo, como un único hermano. El hijo referencia al hijo del nodo en el que se está en ese elemento, que se accede de forma nodo->hijo. El asunto, es que este no es el único hijo que posee el nodo padre, ya que de forma indirecta, los hermanos del nodo hijo son también hijos del nodo padre aunque el nodo padre no tenga una forma de accederlos de manera directa.

Esta implementación permite el crear hermanos dinámicos, lo cual evita un uso de listas que puede llegar a ser ineficiente por la cantidad de métodos que requieren para ser manejados. De esta otra forma, no hace falta más que un par de ciclos recursivos que sirvan para recorrer toda la estructura del árbol con un coste mayor de rendimiento. Además, gracias a este método de construcción, la impresión se puede llevar a cabo de tal forma que se pueda mostrar los padres primero y seguidamente los hijos.

Para la impresión del árbol, se utilizan funciones recursivas, que van recorriendo el árbol en profundidad buscando primeramente a los hijos, y al llegar al final, recorre buscando a sus hermanos si este lo posee. Esta forma recursiva permite imprimir de forma ordenada a como fue construido el árbol una representación gráfica como está compuesto el documento con precisión y eficiencia.

Librerías externas

Para la realización de este programa, solo se utiliza una librería externa para el manejo de cadenas. <string.h> la cual es una librería estándar del lenguaje de programación C que contiene funciones utilizadas en el programa.

Entre las funciones utilizadas, tenemos el strcpy el cual recibe como parámetro 2 punteros de cadena, y copia en la cadena del primer parámetro la cadena que se encuentre en el segundo puntero.

La otra función utilizada es el strcmp que retorna un numero entero mayor, igual o menor a cero dependiendo del resultado de la comparación. Obtendremos un 0 en caso de que las dos cadenas sean del mismo tamaño, en caso contrario se obtendrá otro número diferente a 0 el cual indicará que las cadenas no son iguales.

Esta librería presenta problema de seguridad con ciertas funciones como por ejemplo el desbordamiento de buffer o como es conocido el buffer overflow, pero en este programa no se implementan strings largos los cuales podrían provocar este tipo de problema.

Análisis de resultados

El scanner revisa el código xhtml y le envía los tokens al parser, en el parser se verifica que los tokens que van entrando concuerden con la gramática descrita, dentro de la gramática se crean los nodos para formar el árbol n-ario de la misma.

Los objetivos alcanzados en la realización de este proyecto son; la correcta obtención de los tokens en el scanner desde el archivo del parser de la primera tarea programada del curso, además del correcto funcionamiento de la gramática, sin errores de tipo reduce/reduce o shift/reduce a lo largo del código xhtml. Así como la correcta estructura del árbol n-ario del análisis sintáctico, correspondiente a cada estructura de los documentos de prueba. También realiza la impresión de dicho árbol resultante y la impresión de los errores en caso de que ocurra alguno, ya sea en el analizador léxico como en el sintáctico con el número de fila y de columna donde ocurrió el error.

El objetivo que no se pudo alcanzar es la correcta impresión del contenido del total de los nodos, ya que obtención de la información se está realizando de manera errónea por motivo de que las pseudo-variables envían información inapropiada a lo que debería de enviar, es decir, en vez de enviar únicamente la información del token correspondiente envían además la información de sus hermanos.

Se intentó acceder de distinta forma a la información pero no se logró obtener los datos requeridos.

Conclusión personal

La construcción de una gramática requiere mucho tiempo de análisis de forma tal que no presenten conflictos a la hora de su ejecución. La integración de bison con el analizador léxico flex permite la creación de compiladores eficientes que no requieren de mucho tiempo y evita errores a la hora de implementación.

Este segundo proyecto, nos permitió entender el funcionamiento de un compilador en su etapa de análisis sintáctico así como la construcción a través de reducciones y shift (cambios) del árbol sintáctico el cual permite evaluar si las cadenas procesadas son correctas al lenguaje creado.

Además, nos permitió ver los posibles conflictos que se emprenden a la hora de realizar un compilador así como las decisiones en materia de diseño que se deben de tomar. Mucho del proceso de la creación de una etapa de análisis sintáctico, consiste en modelar reglas que eviten los conflictos pero que a su vez sean lo suficientemente eficientes como para ser ejecutados en un entorno y ambiente específico.

Es necesario entender comprender que la parte primordial del análisis sintáctico es la creación de un árbol que se construye de sus ramas hacia su raíz, es decir dependiendo de lo que va leyendo así va construyendo el árbol, que es una de las técnicas más poderosas para la implementación de compiladores.

Bibliografía

Bison 2.7 (s. f.). *Manual de Bison*, 1. Recuperado de http://www.gnu.org/software/bison/manual/bison.html

Donnelly, C. & stallman, R. (1999, 12 de Febrero). *El Generador de Analizadores Sint ? acticos compatible con YACC.* Estados unidos: Free Software Foundation.

Escuela de informatica uvigo (2008, 23 de Enero). *Generador de analizadores sintacticos BISON*colombia: uvigo.