TPE Parsers

Parser de Gramáticas LL(1)

Autómatas, Teoría de Lenguajes y Compiladores

Integrantes: Augusto Nizzo Mc Intosh
Gonzalo M. Rey

Indice

Introducción

Para este trabajo, se realizó un programa que dada una gramática LL(1), genere un parser que permita determinar si una cadena pertenece o no al lenguaje generado por dicha gramática.

Consideraciones realizadas

Se tomó como consideración que la entrada estaba normalizada cómo el enunciado del Trabajo Práctico lo indica. Esto nos facilitó considerablemente la codificación y prueba de las gramáticas, dado que de otra forma, hubiéramos tenido que parsear la gramática entrante chequeando si la misma era regular, LL(1), etc.

Descripción del desarrollo del TP

El trabajo está desarrollado en Java, utilizando para la inclusión de dependecias y compilación Maven 2.

Se implementaron clases que representan los elementos de un parser LL(1), cómo una clase Grammar, una clase Parser, una clase PredictionTable, etc.

El parser está basado en el pseudocódigo provisto por la cátedra y haciendo uso de las excepciones de Java para representar una entrada inválida en la tabla.

Las clases más importantes implementadas son:

- Symbol: Esta clase representa un símbolo de la gramática, esta clase tiene métodos como "isTerminal", "isEndOfLine", además es la clase encargada de generar símbolos especiales.
- Production: Representa a una producción, sus métodos son los que permiten obtener una parte izquierda y una derecha.
- Parser: Es la case encargada de recibir una tabla de predicciones y una cadena e informar si la cadena pertenece al lenguaje generado por la gramática que generó la tabla.
- PredictionTable: Es la clase encargada de informar cual/es son los símbolos, que, deacuerdo al tope de la pila y el puntero en la cadena, hay que pushear al stack.
- Grammar: Es la clase que representa a una gramática, es la cual, a partir de sus producciones retorna una tabla de prediccion para ser utilizada en un parser LL(1)

La clase WiredTable, fue utilizada para probar el desarrollo del parser sin tener la generación de la tabla de predicciones. Esta valida las cadenas que se forman a partir de las producciones: S->(S)|a

Dificultades encontradas en el desarrollo del TP

La dificultad más importante en el desarrollo, fue traducir
Otra de las grandes dificultades con las que nos topamos fue durante el desarrollo de la tabla de predicciones, con la cual pretendíamos procesar gramáticas que superaban las de una LL(1), complicandonos innecesariamente; una vez que nos dimos cuenta que restricciones tenía un parser de dicha gramática, pudimos avanzar rápidamente con ella.el algoritmo coloquial de la generación de la tabla predictiva a partir de las producciones a un

lenguaje de programación como Java, así como también, la implementación de manera correcta de las entidades que forman a una gramática y a un parser.

Descripción de las gramáticas de ejemplo

Los tests agregados representan gramáticas con distinta complejidad que nos permitieran probar distinta variedad de cosas.

El primer test se corresponde con la anidación de parentesis permitiendo expresar cadenas de la forma "", "()", "(())", "((()))", etc...

El test2 expresa algo similar, con cadenas de la forma "", "ab", "aabb", etc...

El test3 acepta cadenas de la forma "", "ab", "acdb" y "acqdb".

Futuras y posibles extensiones

Entre las posibles y futuras extensiones, una de las mas intuitivas que notamos tiene que ver con la validación de la gramática recibida por archivo de configuración; esta misma podría ser validada por un parser como el desarrollado para el primer TP de la materia, haciendolo así más versátil y robusto.

Otra posible extensión seria hacer el parser un LL(k), que permitiera expresar ambigüedades y cadenas mas complejas.

Otra extensión interesante, coherente con la extensión anteriormente menciona es la detección si una gramática es Libre de Contexto y además, si es LL(1).

Uso del programa

Compilación: El el directorio raiz donde se encuentra el código se debe ejecutar el script "compilar"

Uso:

El uso es de la siguiente forma:

$> java -jar parserll.jar <producciones> <cadena>

Donde "producciones" es el path al archivo de entrada que contiene las producciones a utilizar. "cadena" es la cadena que se quiere analizar.

Conclusiones

El desarrollo de el trabajo práctico, fue sencillo, pero tiene una riquza teórica muy valiosa, ya que muestra, con que sencilles, se puede generar y probar un lenguaje de este tipo, que puede ser usado para hacer, por ejemplo scripting o validaciones de algun mensaje que tenga que tener algúna validacíón de la sintaxis.