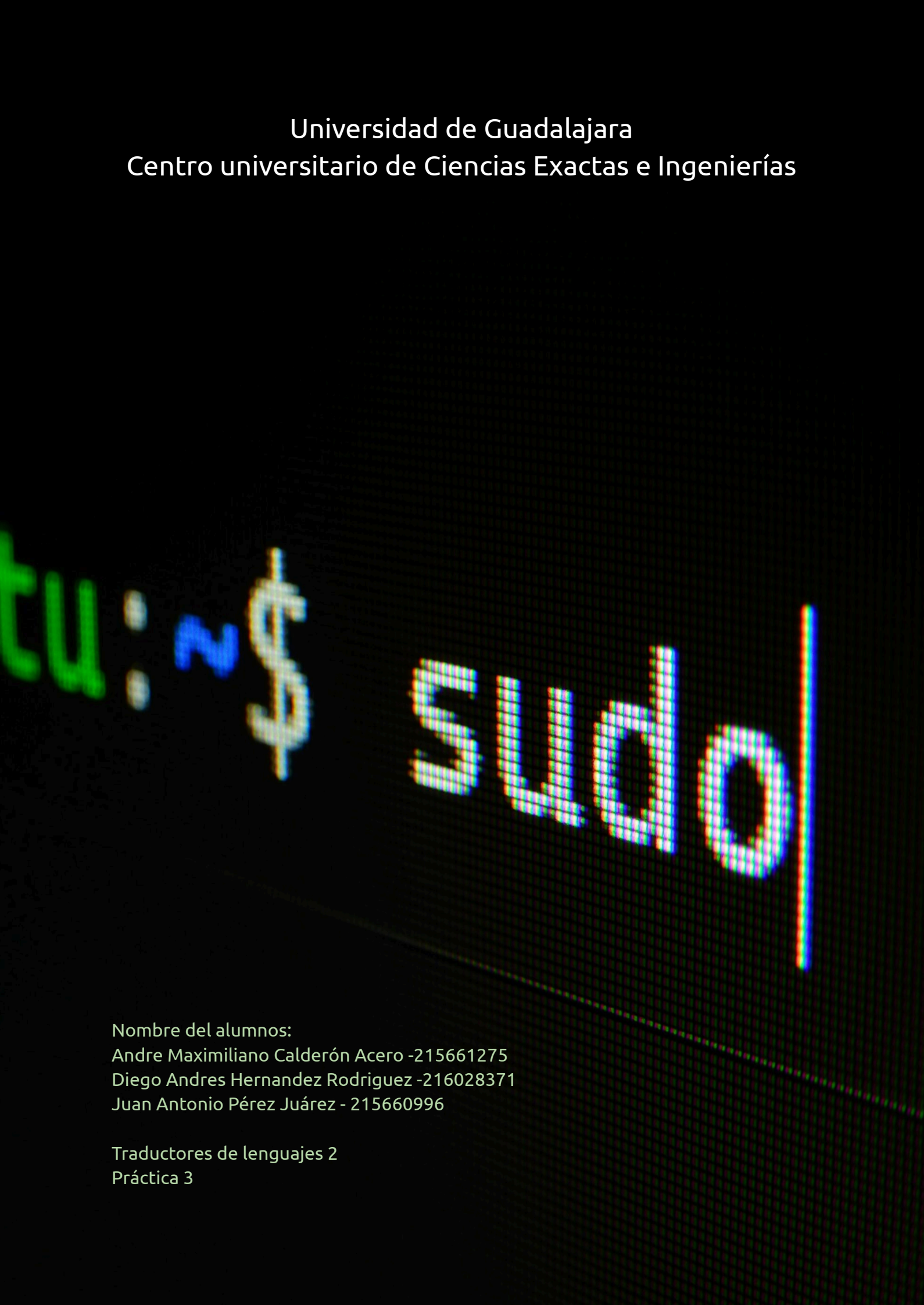
Universidad de Guadalajara
Centro universitario de Ciencias Exactas e Ingenierías



Nombre del alumnos:
Andre Maximiliano Calderón Acero -215661275
Diego Andres Hernandez Rodriguez -216028371
Juan Antonio Pérez Juárez - 215660996

Traductores de lenguajes 2
Práctica 3

Actividad

Tomar la gramática que está al inicio del capítulo 2 del libro de Teoría, diseño e implementación de compiladores de lenguaje y eliminar recursión por la izquierda y ambigüedades.

Se debe explicar qué reglas son ambiguas y cuáles tienen recursión por la izquierda y mostrar cómo se arregla cada una

Marco Teórico

En el diseño de compiladores, las gramáticas libres de contexto (GLC) deben estar estructuradas de manera que eviten problemas como la recursión por la izquierda y las ambigüedades, ya que estos pueden afectar el análisis sintáctico, especialmente en métodos descendentes como LL(1).

La **recursión por la izquierda** ocurre cuando un no terminal puede derivar en una forma que comienza consigo mismo, lo que puede llevar a bucles infinitos en analizadores predictivos. Existen dos tipos: la recursión directa, donde una producción como $\mathrm{A} \rightarrow \mathrm{A}\alpha$ $A \rightarrow A\alpha$ aparece explícitamente, y la recursión indirecta, donde un no terminal deriva en otro que eventualmente vuelve al primero. Para eliminarla, se reorganizan las producciones introduciendo nuevos no terminales. Por ejemplo, una regla como $E \rightarrow E+T \mid T$ se transforma en $\mathrm{E} \rightarrow \mathrm{TE}'$ y $\mathrm{E}' \rightarrow +\mathrm{TE}' \mid \epsilon$ eliminando la recursión directa.

Por otro lado, una gramática es **ambigua** si una misma cadena puede generarse mediante múltiples árboles de derivación, lo que dificulta la interpretación del lenguaje. Un caso clásico es la gramática de sentencias condicionales anidadas con *if-then-else*, donde el *else* puede asociarse con diferentes *if*. Para resolver esto, se redefine la gramática estableciendo una asociatividad clara, como hacer que el *else* corresponda siempre al *if* más cercano. También es común introducir jerarquías de precedencia para operadores, como darle mayor prioridad a la multiplicación sobre la suma en expresiones aritméticas.

Aplicando estos conceptos al libro *Teoría, Diseño e Implementación de Compiladores*, si en el Capítulo 2 se presenta una gramática con producciones como $E \rightarrow E+E \mid E*E \mid id$, se identifica tanto recursión por la izquierda como ambigüedad. La solución implica reescribir las reglas para eliminar la recursión y definir precedencias claras, como separar las operaciones en niveles distintos (términos y factores) y usar recursión por la derecha.

Gramática

<programa> -> begin <declaraciones> <ordenes> end
<declaraciones> -> <declaración> ; | <declaración>;<declaraciones>
<declaración> -> <tipo> <lista_variables>
<tipo> -> entero | real
<lista_variables> -> <identificador> | <identificador>, <lista_variables>
<identificador> -> <letra> | <letra> <resto_letras>
<letra> -> A|B|C|…|Z|a|b|c|…|z
<resto_letras> -> <letraN> | <letraN> <resto_letras>
<letraN> -> A|B|C|…|Z|a|b|c|…|z|0|1|…|9
<ordenes> -> <orden>; | <orden>; <ordenes>
<orden> -> <condición> | <bucle_while> | <asignar>
<condición> -> if (<comparación>) <ordenes> end | if (<comparación>) <ordenes> else <ordenes> end
<comparación> -> <operador> <condición_op> <operador>
<condición_op> -> =| <=|>=|<>|<|>
<operador> -> <identificador> | <números>
<números> -> <números_entero> | <numero_real>
<numero_entero> -> <numero> |<numero> <numero_entero>
<numero> -> 0|1|2|3|4|5|6|7|8|9
<numero_real> -> <números_entero>.<números_entero>
<bucle_while> -> while(<comparación>) <ordenes> endwhile
<asignar> -> <identificador>:=<expresiones_arit>
<expression_arit> ->
(<expresion_arit><operador_arit><expresion_arit>)
|
<identificador> | <numeros> | <expresion_arit><operador_arit><expression_arit>
<operador_arit> -> +|*|-|/

Gramática corregida

<programa> -> begin <declaraciones> <ordenes> end
<declaraciones> -> <declaración> ; <resto_declaraciones>
Recursión eliminada
<resto_declaraciones> -> <declaracion>; <resto_declaraciones>| ε
<declaración> -> <tipo> <lista_variables>
<tipo> -> entero | real
<lista_variables> -> <identificador> <resto_lista_variables>
Nueva regla
Recursión eliminada
<resto_lista_variables> -> , <lista_variables> | ε
Nueva regla
<identificador> -> <letra> <resto_letras>
<letra> -> A|B|C|…|Z|a|b|c|…|z
<resto_letras> -> <letraN> <resto_letras> | ε
<letraN> -> A|B|C|…|Z|a|b|c|…|z|0|1|…|9
<ordenes> -> <orden>;<resto_ordenes> Recursión eliminada
<resto_ordenes> -> <orden>;<resto_ordenes>| ε Nueva regla
<orden> -> <condición>
| <bucle_while>
| <asignar>
<condición> -> if (<comparación>) <ordenes> <resto_condición> Ambiguedad eliminada
<resto_condición> -> else <ordenes> end | end Nueva regla
<comparación> -> <operador> <condición_op> <identificador> | <identificador>
<condición_op> <operador> Ambiguedad eliminada
<condición_op> -> =| <=|>=|<>|<|>
<operador> -> <identificador> | <números>
<números> -> <numero_entero> | <numero_real>
<numero_entero> -> <numero> <resto_numero_entero> Recursión eliminada
<resto_numero_entero> -> <numero> <resto_numero_entero> | ε Nueva regla
<numero> -> 0|1|2|3|4|5|6|7|8|9
<numero_real> -> <numero_entero>.<numero_entero>

Conclusión:

DIEGO ANDRES HERNANDEZ RODRIGUEZ

Para poder entender cómo un lenguaje de programación interpreta la información recibida por los usuarios, es necesario tener un conocimiento teórico sobre gramáticas y las reglas de producción que se necesitan para que los diferentes sistemas digitales puedan entender qué es lo que se requiere hacer. En esta práctica se tuvo la oportunidad de poner en práctica los conocimientos vistos en teoría de la computación, aplicados a lenguajes de programación y compiladores.

Juan Antonio Pérez Juárez:

Si tuviera que definir mi relación con las gramáticas y en general con los compiladores, diría que es una relación tóxica que me está destrozando, realmente me cuesta horrores poder entender las reglas de producción sin ayuda de alguno de mis compañeros de equipo, ya no menciono las reglas sintácticas.
Pero en general siento que entiendo más que hace unos años en la clase de teoría de la computación.
Y afortunadamente hay mucha documentación.

Andre Maximiliano Calderon Acero

Comprender y poder explicar las gramáticas es algo demasiado dificil a mi parecer, aun con el material y los ejemplos vistos en clase, me queda claro que una gramática mal definida puede comprometer todo el proceso sintáctico, por eso se tienen que definir reglas y jerarquías claras que faciliten la interpretación e implementación del lenguaje.

Este tema me hizo ver la importancia que hay que tener en la teoría para poder realizar la práctica..

Bibliografía