Propuesta TP Final

1 Introducción

1.1 Lenguajes formales y gramáticas

Un lenguaje formal es un conjunto de cadenas finitas sobre un alfabeto finito dado. Para definir cuáles son las cadenas aceptadas por un determinado lenguaje y cómo construirlas, pueden utilizarse gramáticas formales.
Una gramática formal es una tupla de 4 elementos (NT, T, s, P), donde:

- NT es un conjunto de símbolos no terminales.
- T es un conjunto de símbolos terminales.
- $s \in NT$ es el no terminal inicial.
- P es un conjunto de reglas de producción.

En 1956, Noam Chomsky definió una clasificación jerárquica para gramáticas formales que lleva a clasificar también los lenguajes que estas definen. De allí, existen 4 clases de lenguajes, cada una contenida en la anterior:

- Lenguajes sin restricciones o de tipo 0.
- Lenguajes sensibles al contexto o de tipo 1.
- Lenguajes libres de contexto o de tipo 2.
- Lenguajes regulares o de tipo 3.

La idea es que el TP se enfoque en los lenguajes y, más específicamente, las gramáticas regulares o de tipo 3.

1.2 Lenguajes y gramáticas regulares

Los lenguajes regulares son los más sencillos que se consideran y pueden ser procesados por un autómata finito. En sus gramáticas, las reglas de producción tienen únicamente un no terminal en el lado izquierdo y un solo terminal, posiblemente acompañado por un no terminal, en el lado derecho. También se permite en el lado derecho la cadena vacía (λ). Algunos ejemplos de estas reglas podrían ser:

- $A \to aB$
- $A \to c$
- $B \to Cb$
- $B \to \lambda$
- $\sigma \to aA$

Este conjunto de gramáticas se divide a su vez en dos subconjuntos: *izquierdas*, aquellas en las que en el lado derecho de las producciones el no terminal va a la izquierda; y *derechas*, en las cuales el no terminal va a la derecha. Si bien lo más común es encontrar gramáticas derechas, puesto que son más intuitivas y simples de entender, también se utilizan mucho las izquierdas.

1.3 Propiedades de cierre

Otra característica importante de los lenguajes regulares es que son cerrados bajo ciertas operaciones. Esto significa que si uno o varios lenguajes son regulares, entonces determinados lenguajes relacionados con estos también lo son. Esta particularidad hace que sea práctico y sencillo manipularlos para construir nuevos lenguajes a partir de lenguajes existentes.
Algunas de estas operaciones son:

- Unión: equivalente a la unión de conjuntos. Si L_1 y L_2 son regulares, entonces $L_1 \cup L_2$ también es regular.
- Intersección: equivalente a la intersección entre conjuntos. Si L_1 y L_2 son regulares, entonces $L_1 \cap L_2$ también es regular.
- Complemento: equivalente al complemento de un conjunto, donde el universo es el conjunto de todas las cadenas finitas que se pueden formar con los símbolos del alfabeto. Si L es regular, entonces $\overline{L}$ también es regular.

- Reversa: la reversa de un lenguaje L incluye las reversas de todas las cadenas pertenecientes a él, donde la reversa de una cadena $a_0a_1a_2...a_n$ es la cadena $a_na_{n-1}...a_1a_0$. Si L es regular, entonces L^R también es regular.
- Concatenación: dados dos lenguajes L_1 y L_2, su concatenación incluye todas las cadenas c_1c_2, donde $c_1 \in L_1$ y $c_2 \in L_2$. Si L_1 y L_2 son regulares, entonces L_1L_2 también es regular.

1.4 Equivalencia

Se dice que dos gramáticas son equivalentes si aceptan el mismo lenguaje. Pero esto no es algo fácil de determinar a simple vista, ya que dos gramáticas equivalentes pueden verse muy distintas a simple vista. Esto se debe a que la misma gramática puede expresarse con distinta cantidad de no terminales y reglas, e incluso las mismas reglas pueden escribirse de diferentes formas.
Resulta interesante entonces tener un método automático que pueda decidir su equivalencia. Esta es la principal inspiración de este proyecto. Para gramáticas de tipo 0, 1 y 2 esto no es decidible. Es por esto que se hace foco en las de tipo 3.

2 Objetivo del TP

El objetivo del trabajo es facilitar la manipulación y revisión de gramáticas regulares, permitiendo operar con ellas y también hacer ciertas consultas sobre las mismas.

Elementos necesarios:

- una representación interna para las gramáticas.
- un *parser* que permita cargar gramáticas desde archivos (estos archivos pueden tener una extensión especial, por ejemplo `.grm`). A continuación se muestra un ejemplo con una posible sintaxis para las mismas (no necesariamente debe usarse):

Derecha	*Izquierda*
`& -> "a" A | "b" B | \;`	`& -> A "b" | B "a";`
`A -> "b" B | "a" &;`	`A -> B "b" | A "a";`
`B -> "b";`	`B -> & "a" | \;`

- un *parser* que reconozca términos con las diferentes operaciones que permitimos entre gramáticas.
- un *pretty printer* para gramáticas y otro para términos.
- un módulo que defina las operaciones entre gramáticas.

Operaciones posibles:

- Consultas de pertenencia: determina si una cadena es aceptada por una gramática.
- Consultas de equivalencia: determina si dos gramáticas son equivalentes, es decir, si ambas aceptan el mismo lenguaje.
- Operaciones binarias: unión, intersección, resta y concatenación.
- Operaciones unarias: complemento y reversa.
- Conversión de lado: toma una gramática derecha y devuelve una izquierda equivalente o viceversa.

Esta herramienta resultaría útil a la hora de construir gramáticas y validar la corrección de las mismas. Si, por ejemplo, se quiere construir un lenguaje que consta de dos partes, se puede construir ambas por separado y luego dejar que el intérprete genere la unión. A su vez, si se prefiere realizar la unión por cuenta propia de la manera que resulte más intuitiva, luego se puede consultar la equivalencia de la gramática obtenida con la generada por el programa para verificar que sea correcta. Además, utilizar las consultas de pertenencia puede ser muy útil a la hora de controlar que la gramática que se creó acepte realmente las cadenas que se esperan.