Universidad de La Laguna

Grado en Ingeniería Informática

Computabilidad y Algoritmia

PRÁCTICA 7

Diseño y simplificación de gramáticas en JFLAP

Factor de ponderación: 8

Semana del 27 al 31 de octubre de 2025

1. Objetivos

El objetivo de la práctica es consolidar los conocimientos adquiridos sobre gramáticas regulares y gramáticas independientes del contexto [1]. Para diseñar, simplificar y validar las gramáticas, utilizaremos la herramienta JFLAP [6]. Tal y como ya hemos visto, JFLAP es un paquete de gran interés en el ámbito de la enseñanza de lenguajes formales pues constituye una herramienta interactiva para visualización y simulación de autómatas finitos, gramáticas y máquinas de Turing, entre otros. Puesto que en esta práctica utilizaremos JFLAP para trabajar con gramáticas, será necesario descargar e instalar la herramienta siguiendo estos sencillos pasos [7].

Para esta práctica será necesario realizar los ejercicios propuestos en este enunciado y llevarlos resueltos a la sesión práctica en el aula de ordenadores. Las propuestas de solución de los ejercicios se entregarán a modo de informe. Durante la sesión presencial se les podrá proponer la resolución de nuevos ejercicios.

Tal y como venimos insistiendo a lo largo de las prácticas de la asignatura, si el alumnado tiene dudas respecto a cualquiera de estos aspectos, debiera acudir al foro de discusiones de la asignatura para plantearlas allí. Se espera que, a través de ese foro, el alumnado intercambie experiencias y conocimientos, ayudándose mutuamente a resolver dichas dudas. También el profesorado de la asignatura intervendrá en las discusiones que pudieran suscitarse, si fuera necesario.

2. Gramáticas independientes del contexto

Desde la antigüedad, los lingüistas han descrito las gramáticas de los idiomas en términos de su estructura de bloques, y han descrito cómo las oraciones se construyen recursivamente a partir de frases más pequeñas y, finalmente, de palabras o elementos de palabras individuales. Una propiedad esencial de estas estructuras de bloques es que las unidades lógicas nunca se solapan. Por ejemplo, la frase en inglés:

John, whose blue car was in the garage, walked to the grocery store

usando corchetes como meta-símbolos puede agruparse como:

[John [, [whose [blue car]] [was [in [the garage]]],]] [walked [to [the [grocery store]]]]

Una gramática independiente del contexto (*Context Free Grammar, CFG*) [2] proporciona un mecanismo simple y matemáticamente preciso para describir los métodos por los cuales las cadenas de algún lenguaje formal se construyen a partir de bloques más pequeños, capturando la "estructura de bloques" de las frases de manera natural. Su simplicidad hace que este formalismo sea adecuado para un estudio matemático riguroso. En cualquier caso, las características de la sintaxis de los lenguajes naturales no se pueden modelar mediante gramáticas independientes del contexto.

Este formalismo y también su clasificación como un tipo especial de gramática formal fue desarrollado a mediados de los años 50 del siglo pasado por Noam Chomsky [3]. Basándose en esta notación de gramáticas basada en reglas, Chomsky agrupó los lenguajes formales en una serie de cuatro subconjuntos anidados conocidos como la *Clasificación de Chomsky* [4]. Esta clasificación fue y sigue siendo fundamental para la teoría de lenguajes formales, especialmente para la teoría de los lenguajes de programación y el desarrollo de compiladores.

La estructura de bloques fue introducida en los lenguajes de programación por el proyecto Algol (1957-1960), el cual, como consecuencia, también incluía una gramática independiente del contexto para describir la sintaxis del lenguaje. Esto se convirtió en una característica estándar de los lenguajes de programación, aunque no solo de éstos. En [5], a modo de ejemplo, se utiliza una gramática independiente del contexto para especificar la sintaxis de JavaScript.

Las gramáticas independientes del contexto son lo suficientemente simples como para permitir la construcción de algoritmos de análisis eficientes que, para una cadena dada, determinan si - y cómo - se puede generar esa cadena a partir de la gramática. El *algoritmo de Cocke-Younger-Kasami* es un ejemplo de ello, mientras que los ampliamente usados analizadores LR y LL son algoritmos más simples que tratan sólo con subconjuntos más restrictivos de gramáticas independientes del contexto.

Formalmente una *gramática independiente del contexto* G (por simplicidad nos referiremos a "una gramática") viene definida por una tupla $G \equiv (\Sigma,\ V,\ S,\ P)$ cada uno de cuyos componentes se explican a continuación:

- Σ: Conjunto de símbolos terminales (o alfabeto de la gramática), $\Sigma \cap V = \emptyset$
- V: Conjunto de símbolos no terminales ($V \neq \emptyset$)
- S: Símbolo de arranque (***S****tart*) o axioma de la gramática ($S \in V$)
- P: Conjunto de reglas de producción:

  $P = \{A \to \alpha \mid A \in V,\ \alpha \in (V \cup \Sigma)^*\}$

  $P \subset V \times (V \cup \Sigma)^*,\ (P \neq \emptyset)$

Habitualmente, para especificar una gramática, se especifican todas sus reglas de producción (P), y se sigue un convenio de notación que permite determinar todos los elementos. El símbolo de arranque es aquel cuyas producciones aparecen en primer lugar en la lista de producciones. Por ejemplo, las siguientes producciones:

$S \to U \mid W$

$U \to TaU \mid TaT$

$W \to TbW \mid TbT$

$T \to aTbT \mid bTaT \mid \epsilon$

definen una gramática independiente del contexto para el lenguaje de cadenas de letras a y b en las que hay un número diferente de unas que de otras. Como se puede apreciar, a partir de las producciones anteriores podremos deducir la especificación completa de la gramática:

- $\Sigma = \{a,\ b\}$
- $V = \{S,\ U,\ W,\ T\}$
- El símbolo de arranque es S
- Las reglas de producción son las de la relación anterior

3. Gramáticas regulares

Por su parte, las gramáticas regulares pueden generar lenguajes regulares, que son los lenguajes más simples en la jerarquía de Chomsky y que pueden ser reconocidos por autómatas finitos. Es importante resaltar que todos los lenguajes generados por gramáticas regulares son también generados por gramáticas independientes del contexto. Esto significa que las gramáticas regulares son un subconjunto de las gramáticas independientes del contexto. Sin embargo, no todos los lenguajes generados por gramáticas independientes del contexto pueden ser generados por gramáticas regulares.

Una *gramática regular* G es una 4-tupla $G = (\Sigma, N, S, P)$, donde:

- Σ es un alfabeto
- N es un conjunto de símbolos no terminales
- S es un no terminal llamado *símbolo inicial (start) o de arranque*
- P es un conjunto de reglas de sustitución denominadas *producciones*, tales que (caben dos alternativas):
  - O bien todas las producciones son lineales por la derecha: $A \to uB|v$ ($A, B \in N$ y $u, v \in \Sigma^*$)
  - O todas las producciones son lineales por la izquierda: $A \to Bu|v$ ($A, B \in N$ y $u, v \in \Sigma^*$)

Las gramáticas regulares tienen una estructura simple que las hace especialmente útiles para describir patrones regulares, como los que se encuentran en expresiones regulares.

4. Simplificación de gramáticas

Existe un proceso que permite la simplificación de cualquier gramática independiente del contexto. Dicho procedimiento se basa en los pasos o algoritmos siguientes (téngase en cuenta que una aplicación preliminar del último algoritmo (3) puede mejorar la eficiencia del proceso):

1. Eliminación producciones vacías
2. Eliminación de producciones unitarias
3. Eliminación de símbolos y producciones inútiles

4.1. Eliminación de producciones vacías

Una producción gramatical es vacía si es de la forma:

$$A \to \varepsilon$$

El objetivo será modificar la gramática para eliminar este tipo de producciones, sin cambiar el lenguaje generado. Hay que tener en cuenta que si $\varepsilon \in L(G)$ no se podrán eliminar todas las producciones vacías. En este caso se mantendrá como única producción vacía:

$$S \to \varepsilon$$

Como primer paso de este proceso de eliminación de producciones vacías, es necesario calcular el conjunto de variables anulables, siguiendo el proceso que se muestra a continuación:

Algorithm 1 Cálculo del conjunto de variables anulables

$H = \emptyset$;
for all producción de la forma $A \to \varepsilon$ **do**
    $H = H \cup \{A\}$;
end for
while H cambie **do**
    for all producción $B \to A_1A_2...A_n$ donde $A_i \in H, \forall i = 1, 2, ...n$ **do**
        $H = H \cup \{B\}$;
    end for
end while

Una vez calculado el conjunto de variables anulables procederemos a ejecutar el algoritmo de eliminación de producciones vacías:

Algorithm 2 Eliminación de producciones vacías

Eliminar todas las producciones vacías ($A \to \varepsilon$);
for all producción de la forma $A \to X_1X_2...X_n, X_i \in (\Sigma \cup V)$ **do**
    Eliminar la producción $A \to X_1X_2...X_n$;
    Añadir todas las producciones $A \to Y_1Y_2...Y_n$, donde:
        $Y_i = X_i$ si X_i no es anulable;
        $(Y_i = X_i) \vee (Y_i = \varepsilon)$ si X_i es anulable;
        $Y_i \neq \epsilon \ \forall i$ (no se introducen producciones vacías);
end for

4.2. Eliminación de producciones unitarias

Una producción es unitaria si tiene la forma:

$$A \to B \quad (A, B \in V)$$

Este tipo de producciones hacen la gramática innecesariamente compleja pues simplemente renombran un símbolo no terminal. Este tipo de producciones se pueden eliminar siguiendo los pasos que se indican a continuación:

Algorithm 3 Eliminación de producciones unitarias

```
H = ∅;  //H ⊆ (V × V) de modo que (A, B) ∈ H ⇔ A ⇒* B
for all producción de la forma A → B do
    H = H ∪ {(A, B)};
end for
while H cambie do
    for all par de parejas de la forma (A, B)(B, C) do
        if (A, C) ∉ H then
            H = H ∪ {(A, C)};
        end if
    end for
end while
Eliminar todas las producciones unitarias;
for all pareja (A, B) ∈ H do
    for all producción B → α do
        Añadir a la gramática una producción A → α;
    end for
end for
```

4.3. Eliminación de símbolos y producciones inútiles

Un símbolo $X \in (\Sigma \cup V)$ se dice que es útil $\Leftrightarrow \exists$ una derivación de la forma:

$$S \Rightarrow^+ \alpha X \beta \Rightarrow^* w \in \Sigma^*$$

Para que un símbolo X sea útil ha de intervenir en la derivación de una cadena $u \in \Sigma^*$:

- X ha de ser "alcanzable" desde S: $S \Rightarrow^+ \alpha X \beta$
- Desde X ha de poder derivarse una cadena de Σ^*: $X \Rightarrow^* u \in \Sigma^*$

Además, se dice que una producción es útil si lo son todos sus símbolos. Teniendo esto en cuenta, la eliminación de símbolos y producciones inútiles consta de dos etapas:

1. Eliminar los no terminales desde los que no se puede derivar una cadena de Σ^* y las producciones en las que aparezcan dichos símbolos.
2. Eliminar aquellos no terminales que no se puedan derivar partiendo de S y las producciones en las que aparecen.

Algorithm 4 Eliminación de símbolos y producciones inútiles: Etapa 1

$V' = \emptyset$;
for all producción de la forma $A \to w$ **do**
  $V' = V' \cup \{A\}$;
end for
while V' **do** cambie
  for all producción de la forma $B \to \alpha$ **do**
    if todos los no-terminales de α pertenecen a V' **then**
      $V' = V' \cup \{B\}$;
    end if
  end for
end while
Eliminar todos los no terminales que estén en V y no en V'
Eliminar todas las producciones donde aparezca algún símbolo de los eliminados

Algorithm 5 Eliminación de símbolos y producciones inútiles: Etapa 2

$J = \{S\}$;
$V' = \{S\}$;
$\Sigma' = \emptyset$;
while $J \neq \emptyset$ **do**
  Extraer un no terminal A de J: $J = J - \{A\}$;
  for all producción de la forma $A \to \alpha$ **do**
    Poner todos los terminales de α en Σ';
    for all no-terminal B en α **do**
      if $B \notin V'$ **then**
        $J = J \cup \{B\}$;
        $V' = V' \cup \{B\}$;
      end if
    end for
  end for
end while
Eliminar todos los no terminales de V que no estén en V' y todos los terminales que no estén en Σ^*
Eliminar todas las producciones donde aparezca un símbolo o variable de los eliminados

5. Diseño y simplificación de gramáticas en JFLAP

Para trabajar con gramáticas en JFLAP debemos elegir la opción `Grammar` desde el menú principal que encontramos al abrir la herramienta:

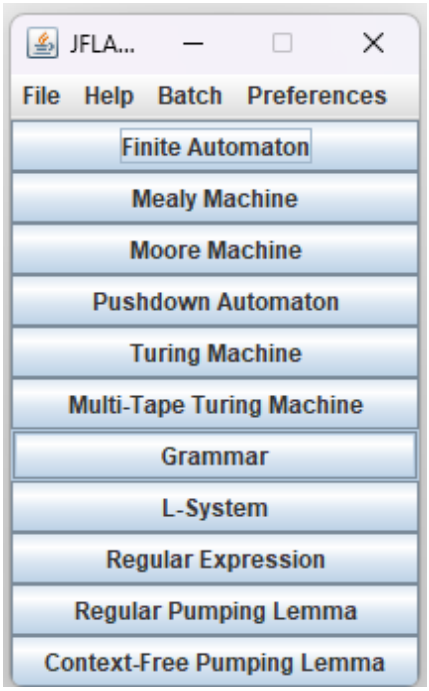

Figura 1: Gramáticas en JFLAP

Tras elegir dicha opción, se nos abrirá una ventana de edición en la que podremos ir especificando una a una las producciones de la gramática con la que queramos trabajar. A modo de ejemplo, en la Figura 2 se muestra una gramática que genera el lenguaje siguiente:

$$L = \{(ab)^n c^{2m-1} \mid n \geq 0, m \geq 1\}$$

JFLAP ofrece la opción de ir simplificando paso a paso la gramática diseñada. Para ello, bastará con elegir la opción `Transform Grammar` del menú `Convert`. En el primer paso de la transformación se aplicará el proceso de eliminación de producciones vacías (`Lambda Removal`). JFLAP ofrece la opción de aplicar el algoritmo paso a paso (opción `Do Step`), o bien, de forma completa (opción `Do All`). Ejecutando el proceso completo para la gramática de ejemplo, se obtiene un resultado como el que se muestra en la Figura 3. Nótese cómo la producción $A \to \epsilon$ ha desaparecido de la gramática y, en su lugar, han aparecido producciones como las siguientes:

$$S \to C$$

$$A \to ab$$

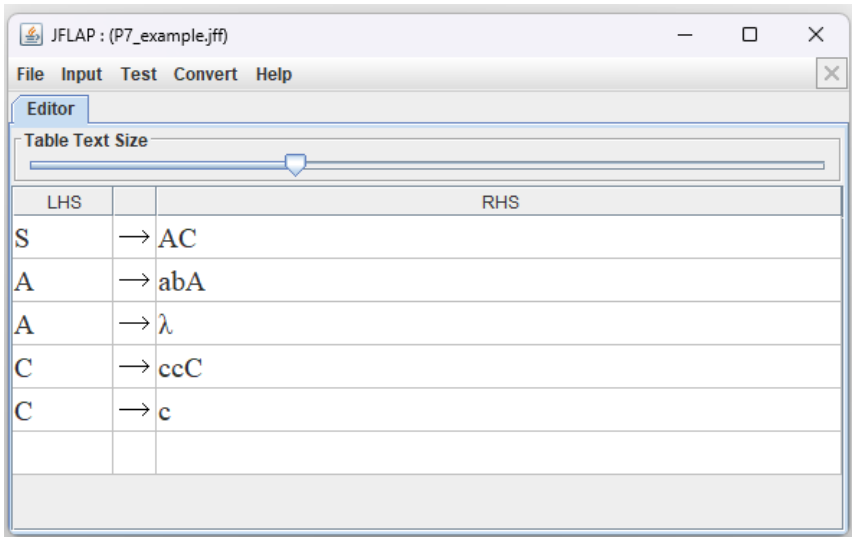

Figura 2: Diseño de una gramática en JFLAP

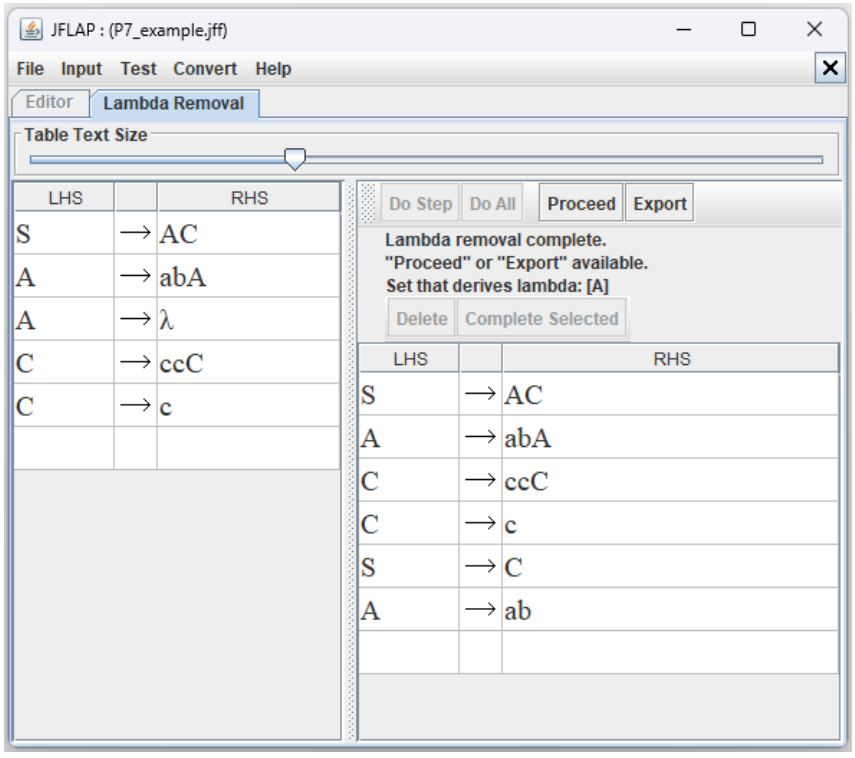

Figura 3: Eliminación de producciones vacías en JFLAP

Tras eliminar las producciones vacías, elegiremos la opción `Proceed` para continuar con el proceso de simplificación de la gramática. El siguiente paso consistirá en la eliminación de producciones unitarias (`Unit Removal`). Si elegimos, por ejemplo, la opción `Do All`, obtendremos un resultado como el siguiente:

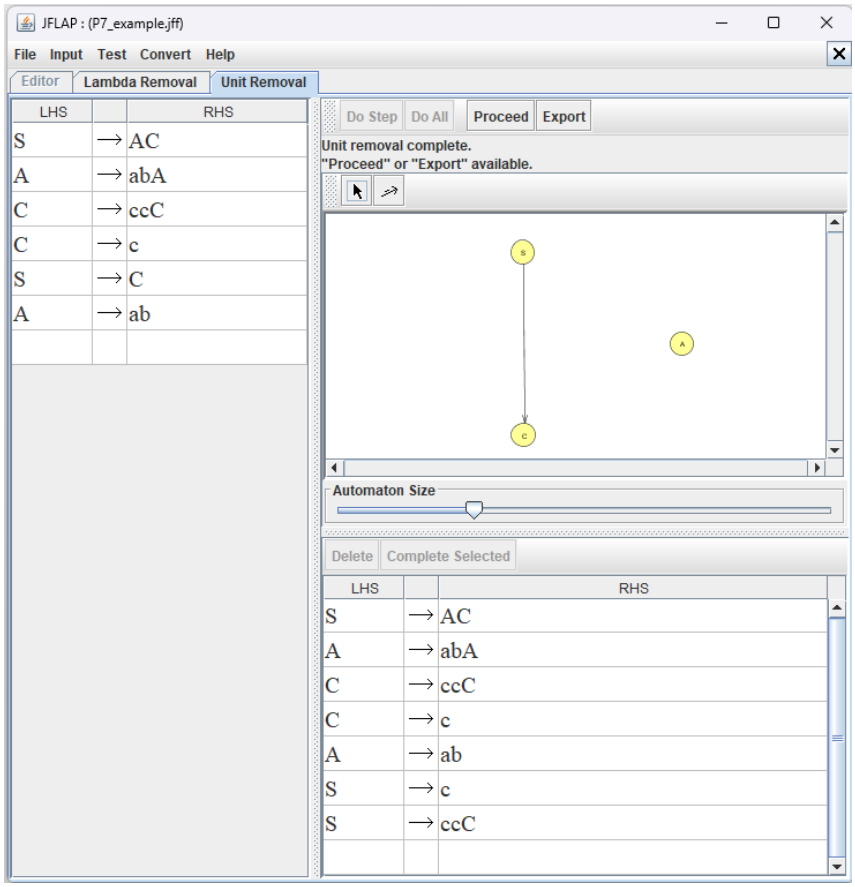

Figura 4: Eliminación de producciones unitarias en JFLAP

En este caso, podemos comprobar cómo la gramática se ha transformado para evitar la producción unitaria $S \to C$, la cual ha sido sustituida por las producciones:

$$S \to c$$

$$S \to ccC$$

Puesto que la gramática resultante ya está simplificada, JFLAP se saltaría el algoritmo de eliminación de símbolos y producciones inútiles (`Useless Removal`) y pasaría directamente a la conversión a Forma Normal de Chomsky. Sin embargo, en esta práctica no trabajaremos con la conversión a Forma Normal de Chomsky y, por lo tanto, exportaremos la gramática resultante tras el proceso de simplificación. Para exportar la gramática en cualquier parte del proceso, podemos elegir la opción `Export` que aparece en la zona superior derecha de la ventana auxiliar que se abre durante los pasos de la simplificación.

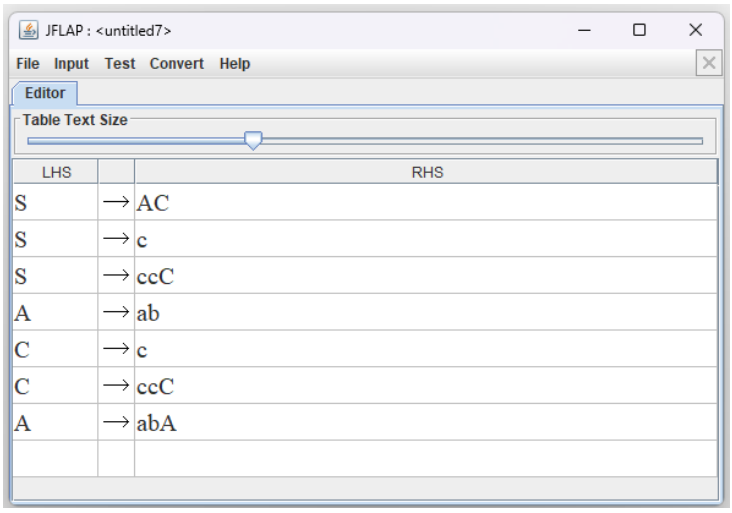

Figura 5: Exportación de la gramática simplificada

Finalmente, téngase en cuenta que JFLAP ofrece opciones para validar qué cadenas se generan o no a partir de la gramática diseñada. En este sentido, una de las opciones más sencillas de entender visualmente es la de `Brute Force Parse` que se encuentra en el menú `Input`. Esta opción genera una representación gráfica del árbol de análisis sintáctico para la cadena especificada. A modo de ejemplo, véase la Figura 6 donde pueda apreciarse cómo la cadena $ababccc \in L(G)$.

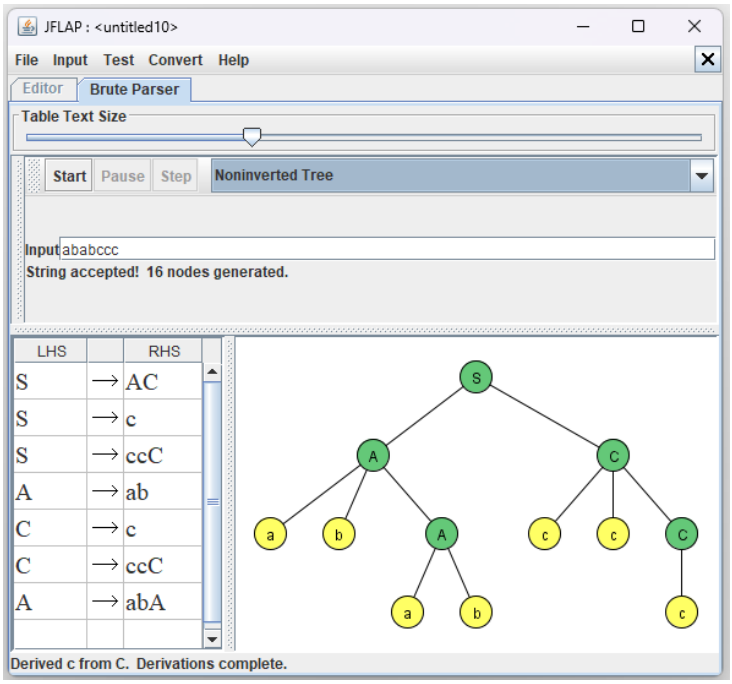

Figura 6: Análisis de cadenas en JFLAP

6. Ejercicios

Con el objetivo de profundizar en el diseño y la simplificación de gramáticas se proponen los siguientes ejercicios. La idea es que se realice, para cada ejercicio, un diseño de gramática independiente del contexto y se aplique paso a paso, el proceso completo de simplificación de la misma. Una vez realizado dicho proceso de simplificación, se utilizará JFLAP para validar el correcto diseño de la gramática así como para validar la correcta simplificación de la misma.

1. Diseñar una gramática independiente del contexto que genere el lenguaje
$$L = \{a^n b^n \mid n \geq 0\}$$

2. Diseñar una gramática independiente del contexto que genere el lenguaje
$$L = \{a^n b^m \mid n, m \geq 0, n \neq m\}$$

3. Diseñar una gramática independiente del contexto para el lenguaje
$$L = \{ww^I \mid w \in \{a, b\}^*\}$$

4. Diseñar una gramática independiente del contexto que genere el lenguaje
$$L = \{a^n b^m c^n \mid n \geq 0, m \; impar\}$$

5. Diseñar una gramática independiente del contexto que genere el lenguaje
$$L = \{a^n b^m c^m d^n \mid n, m \geq 1\}$$

6. Diseñar una gramática independiente del contexto que genere el lenguaje
$$L = \{a^n b^m \mid n > m \geq 0\}$$

7. Diseñar una gramática independiente del contexto que genere el lenguaje
$$L = \{a^i b^j c^{i+j} \mid i, j \geq 1, i + j \; par\}$$

8. Diseñar una gramática independiente del contexto que genere el lenguaje de las expresiones booleanas con los operadores AND, OR, y NOT, usando paréntesis para agrupar. Ejemplos: “(true AND false)”, “NOT (true OR false)”, “true AND (false OR true)”.

9. Diseñar una gramática independiente del contexto que genere expresiones aritméticas simples con suma y multiplicación, utilizando los símbolos +, ∗, (,) y los números 0, 1, etc. Ejemplos: “1+2”, “(1+2)*3”, “4*(5+6)”.

10. Diseñar una gramática independiente del contexto que genere el lenguaje de listas anidadas usando corchetes, como en los lenguajes de programación. Ejemplos: [], [[]], [[[]]], [[1,2],[3,4]].

7. Criterios de evaluación

Se señalan a continuación los aspectos más relevantes (la lista no es exhaustiva) que se tendrán en cuenta a la hora de evaluar esta práctica:

- Se valorará que el alumnado haya realizado, con anterioridad a la sesión de prácticas, y de forma efectiva, todas las tareas propuestas en este guion.
- Se valorarán las soluciones ofrecidas a los ejercicios planteados en este guion, tanto en relación al diseño de gramáticas como a la simplificación de las mismas.
- También se valorará la adecuada utilización de la herramienta JFLAP.
- Finalmente, se valorará el formato y claridad del informe de prácticas presentado, así como la adecuación y cumplimiento de las instrucciones aquí proporcionadas. En este sentido, el informe deberá contener (para cada uno de los ejercicios planteados) al menos lo siguiente:

  1. El enunciado del ejercicio.
  2. Una breve explicación de la gramática diseñada, describiendo los criterios seguidos durante el diseño, así como las reglas y elementos que la componen.
  3. Una imagen con la gramática diseñada en JFLAP.
  4. Al menos tres ejemplos de cadenas que pueden ser generadas a partir de la gramática (incluir cada árbol de análisis sintáctico).
  5. Una descripción detallada del proceso, paso a paso, de simplificación de la gramática junto con una verificación de que el resultado realizado de forma autónoma coincide con el resultado obtenido con cada algoritmo en JFLAP.

Si el alumnado tiene dudas respecto a cualquiera de estos aspectos, debiera acudir al foro de discusiones de la asignatura para plantearlas allí. Se espera que, a través de ese foro, el alumnado intercambie experiencias y conocimientos, ayudándose mutuamente a resolver dichas dudas. También el profesorado de la asignatura intervendrá en las discusiones que pudieran suscitarse, si fuera necesario.

Referencias

[1] Transparencias del Tema 3 de la asignatura: Lenguajes y Gramáticas Independientes del Contexto, https://campusvirtual.ull.es/2526/ingenieriaytecnologia/mod/resource/view.php?id=11876

[2] Context Free Grammar, https://en.wikipedia.org/wiki/Context-free_grammar

[3] Noam Chomsky https://en.wikipedia.org/wiki/Noam_Chomsky

[4] Chomsky hierarchy https://en.wikipedia.org/wiki/Chomsky_hierarchy

[5] JavaScript 1.4 Grammar https://www-archive.mozilla.org/js/language/grammar14.html

[6] JFLAP: Java Formal Language and Automata Package, https://www.jflap.org/

[7] Instalación de JFLAP, https://campusvirtual.ull.es/2526/ingenieriaytecnologia/mod/page/view.php?id=11871

[8] Susan H. Rodger and Thomas W. Finley: *An Interactive Formal Languages and Automata Package*. Jones & Bartlett Publishers, Sudbury, MA, 2006. ISBN 0763738344. https://www.jflap.org/jflapbook/jflapbook2006.pdf