Tema 1: Lenguajes Formales

Autómatas y Lenguajes Formales

Dpto. de Ingeniería de la Información y las Comunicaciones

UNIVERSIDAD DE MURCIA

Lenguajes Naturales

Algunas características

- Se usan para para la comunicación entre humanos.
- Gran variedad de componentes lingüísticos, estructura compleja, mucha expresividad.
- **Reglas gramaticales poco rígidas y difíciles de especificar con precisión.**
- **Ambigüedad y significado dependendiente del contexto.**

Lenguajes Naturales

Algunas características

- Se usan para para la comunicación entre humanos.
- Gran variedad de componentes lingüísticos, estructura compleja, mucha expresividad.
- **Reglas gramaticales poco rígidas y difíciles de especificar con precisión.**
- **Ambigüedad y significado dependendiente del contexto.**

Procesamiento del lenguaje natural

Campo de investigación de la **Inteligencia Artificial** para:

- Facilitar comunicación humano-ordenador (*interfaces inteligentes*).
- Traducción automática de un idioma a otro.
- Reconocimiento de la escritura y del habla.

Es complejo. Resultados limitados.

Lenguajes Formales

- Un **lenguaje formal** es un conjunto de cadenas que cumplen una propiedad que puede expresarse de manera precisa (*formal*).
- Con el uso de lenguajes formales se consigue **más eficiencia y más precisión** en tareas de procesamiento.

Lenguajes Formales

- Un **lenguaje formal** es un conjunto de cadenas que cumplen una propiedad que puede expresarse de manera precisa (*formal*).
- Con el uso de lenguajes formales se consigue **más eficiencia y más precisión** en tareas de procesamiento.

Ejemplos de lenguajes formales

- **El lenguaje de programación C.**
  Sus cadenas son programas que respetan la sintaxis de C.

Lenguajes Formales

- Un **lenguaje formal** es un conjunto de cadenas que cumplen una propiedad que puede expresarse de manera precisa (*formal*).
- Con el uso de lenguajes formales se consigue **más eficiencia y más precisión** en tareas de procesamiento.

Ejemplos de lenguajes formales

- El lenguaje de programación C.
  Sus cadenas son programas que respetan la sintaxis de C.
- **El lenguaje de la lógica proposicional**.
  Sus cadenas son fórmulas proposicionales bien formadas.
  Ej.: La cadena "$v_1 \vee (\neg v_2 \wedge v_3)$" es una fórmula del lenguaje y "$\vee(v_1 \wedge v_2)$" no lo es.

Lenguajes Formales

- Un **lenguaje formal** es un conjunto de cadenas que cumplen una propiedad que puede expresarse de manera precisa (*formal*).
- Con el uso de lenguajes formales se consigue **más eficiencia y más precisión** en tareas de procesamiento.

Ejemplos de lenguajes formales

- El lenguaje de programación C.
  Sus cadenas son programas que respetan la sintaxis de C.
- El lenguaje de la lógica proposicional.
  Sus cadenas son fórmulas proposicionales bien formadas.
  Ej.: La cadena "$v_1 \vee (\neg v_2 \wedge v_3)$" es una fórmula del lenguaje y "$\vee(v_1 \wedge v_2)$" no lo es.
- **El lenguaje de direcciones de correo electrónico.**
  Contiene cadenas que respetan la sintaxis de las direcciones de correo. **Ej.:** "unalumno@um.es" es una cadena del lenguaje, pero "unalumno@es" no lo es.

Lenguajes Formales

- Un **lenguaje formal** es un conjunto de cadenas que cumplen una propiedad que puede expresarse de manera precisa (*formal*).
- Con el uso de lenguajes formales se consigue **más eficiencia y más precisión** en tareas de procesamiento.

Ejemplos de lenguajes formales

- El lenguaje de programación C.
- El lenguaje de la lógica proposicional.
- El lenguaje de direcciones de correo electrónico.

Formalismos de especificación y procesamiento

- Expresiones regulares.
- Gramáticas.
- Autómatas.

Ejemplo de descripción informal de sintaxis

1. **Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.**
2. En un bloque de sentencias debe haber al menos una sentencia.
3. Una sentencia puede ser una sentencia de asignación o una sentencia if o una sentencia while.
4. Una sentencia de asignación consiste en ...
5. ...

Ejemplo de gramática para el lenguaje anterior

$\langle \text{programa} \rangle \rightarrow \textit{begin}\langle \text{bloque-sentencias} \rangle \textit{end}$
$\langle \text{bloque-sentencias} \rangle \rightarrow \langle \text{sentencia} \rangle \mid \langle \text{sentencia} \rangle\langle \text{bloque-sentencias} \rangle$
$\langle \text{sentencia} \rangle \rightarrow \langle \text{sent-if} \rangle \mid \langle \text{sent-while} \rangle \mid \langle \text{sent-asig} \rangle$
$\langle \text{sent-asig} \rangle \rightarrow \ldots$
...

Ejemplo de descripción informal de sintaxis

1. Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.
2. **En un bloque de sentencias debe haber al menos una sentencia.**
3. Una sentencia puede ser una sentencia de asignación o una sentencia if o una sentencia while.
4. Una sentencia de asignación consiste en ...
5. ...

Ejemplo de gramática para el lenguaje anterior

$\langle\text{programa}\rangle \rightarrow \textit{begin}\langle\text{bloque-sentencias}\rangle\textit{end}$
$\langle\text{bloque-sentencias}\rangle \rightarrow \langle\text{sentencia}\rangle \mid \langle\text{sentencia}\rangle\langle\text{bloque-sentencias}\rangle$
$\langle\text{sentencia}\rangle \rightarrow \langle\text{sent-if}\rangle \mid \langle\text{sent-while}\rangle \mid \langle\text{sent-asig}\rangle$
$\langle\text{sent-asig}\rangle \rightarrow \dots$
$\dots$

Ejemplo de descripción informal de sintaxis

1. Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.
2. En un bloque de sentencias debe haber al menos una sentencia.
3. **Una sentencia puede ser una sentencia de asignación o una sentencia if o una sentencia while.**
4. Una sentencia de asignación consiste en ...
5. ...

Ejemplo de gramática para el lenguaje anterior

$\langle\text{programa}\rangle \rightarrow \textit{begin}\langle\text{bloque-sentencias}\rangle\textit{end}$

$\langle\text{bloque-sentencias}\rangle \rightarrow \langle\text{sentencia}\rangle \mid \langle\text{sentencia}\rangle\langle\text{bloque-sentencias}\rangle$

$\langle\text{sentencia}\rangle \rightarrow \langle\text{sent-if}\rangle \mid \langle\text{sent-while}\rangle \mid \langle\text{sent-asig}\rangle$

$\langle\text{sent-asig}\rangle \rightarrow \ldots$

...

Ejemplo de descripción informal de sintaxis

1. Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.
2. En un bloque de sentencias debe haber al menos una sentencia.
3. Una sentencia puede ser una sentencia de asignación o una sentencia if o una sentencia while.
4. **Una sentencia de asignación consiste en . . .**
5. . . .

Ejemplo de gramática para el lenguaje anterior

$\langle$programa$\rangle \rightarrow$ *begin*$\langle$bloque-sentencias$\rangle$*end*

$\langle$bloque-sentencias$\rangle \rightarrow \langle$sentencia$\rangle \mid \langle$sentencia$\rangle\langle$bloque-sentencias$\rangle$

$\langle$sentencia$\rangle \rightarrow \langle$sent-if$\rangle \mid \langle$sent-while$\rangle \mid \langle$sent-asig$\rangle$

$\langle$sent-asig$\rangle \rightarrow \ldots$

. . .

Ejemplo de descripción informal de sintaxis

1. Un programa comienza por la palabra clave *begin*, va seguido de un bloque de sentencias y termina con la palabra clave *end*.
2. En un bloque de sentencias debe haber al menos una sentencia.
3. Una sentencia puede ser una sentencia de asignación o una sentencia if o una sentencia while.
4. Una sentencia de asignación consiste en ...
5. ...

Ejemplo de gramática para el lenguaje anterior

$\langle$programa$\rangle \rightarrow$ *begin*$\langle$bloque-sentencias$\rangle$*end*
$\langle$bloque-sentencias$\rangle \rightarrow \langle$sentencia$\rangle \mid \langle$sentencia$\rangle\langle$bloque-sentencias$\rangle$
$\langle$sentencia$\rangle \rightarrow \langle$sent-if$\rangle \mid \langle$sent-while$\rangle \mid \langle$sent-asig$\rangle$
$\langle$sent-asig$\rangle \rightarrow \ldots$
...

Aplicaciones

Desde su nacimiento en los años 40 del siglo XX, la **Teoría de Autómatas y Lenguajes Formales** ha encontrado aplicación en campos muy diversos, no sólo en el ámbito de los lenguajes de programación.

Algunos de ellos son:

- **Búsqueda, extracción, validación y análisis de cadenas de patrón regular.**
- **Desarrollo de compiladores, intérpretes y conversores de formato.**
- Procesamiento del lenguaje natural.
- Modelado y verificación de protocolos de comunicación.
- Modelado de sistemas de eventos discretos.
- Diseño de sistemas digitales secuenciales y automatismos industriales.

Alfabetos y cadenas

- Un **alfabeto** (o *vocabulario*) V es un conjunto finito y no vacío de elementos llamados **símbolos**.
  Ej.: $V = \{a, b, c\}$ es un alfabeto de tres símbolos.
  Alfabetos de *interpretación más común*:

  $V_{dig} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ alfabeto de dígitos decimales.
  $V_{lat} = \{a, b, \ldots, z, A, \ldots, Z\}$ alfabeto de letras latinas.
  $V_{bin} = \{0, 1\}$ alfabeto de dígitos binarios.
  V_{ASCII} alfabeto formado por todos los caracteres ASCII.

Alfabetos y cadenas

- Un **alfabeto** (o *vocabulario*) V es un conjunto finito y no vacío de elementos llamados **símbolos**.
  Ej.: $V = \{a, b, c\}$ es un alfabeto de tres símbolos.
- Una **cadena** (o *palabra*) w es una secuencia finita de símbolos de cierto alfabeto V.
  Ej.: Si el alfabeto es $V = \{a, b, c\}$ tenemos que $aa, abb, bcacc$ son cadenas formadas por símbolos de V.

Alfabetos y cadenas

- Un **alfabeto** (o *vocabulario*) V es un conjunto finito y no vacío de elementos llamados **símbolos**.
  Ej.: $V = \{a, b, c\}$ es un alfabeto de tres símbolos.
- Una **cadena** (o *palabra*) w es una secuencia finita de símbolos de cierto alfabeto V.
  Ej.: Si el alfabeto es $V = \{a, b, c\}$ tenemos que $aa, abb, bcacc$ son cadenas formadas por símbolos de V.
- La **longitud** de una cadena w se denota $|w|$ y es el número de símbolos que aparecen en w.
  Ej.: Si $V = \{a, b, c\}$ entonces $|bcacc| = 5$.

Alfabetos y cadenas

- Un **alfabeto** (o *vocabulario*) V es un conjunto finito y no vacío de elementos llamados **símbolos**.
  Ej.: $V = \{a, b, c\}$ es un alfabeto de tres símbolos.
- Una **cadena** (o *palabra*) w es una secuencia finita de símbolos de cierto alfabeto V.
  Ej.: Si el alfabeto es $V = \{a, b, c\}$ tenemos que $aa, abb, bcacc$ son cadenas formadas por símbolos de V.
- La **longitud** de una cadena w se denota $|w|$ y es el número de símbolos que aparecen en w.
  Ej.: Si $V = \{a, b, c\}$ entonces $|bcacc| = 5$.
- **La cadena vacía** se denota como λ y se tiene que $|\lambda| = 0$.

El sentido de los símbolos

- Los símbolos no siempre representan caracteres de una cadena de texto que se desea procesar.
- Los símbolos son entidades **abstractas** que se usan para representar (*simbolizar*) **componentes indivisibles de un lenguaje**, que se combinan para formar *secuencias o cadenas* muy diversas.

El sentido de los símbolos

- Los símbolos no siempre representan caracteres de una cadena de texto que se desea procesar.
- Los símbolos son entidades **abstractas** que se usan para representar (*simbolizar*) **componentes indivisibles de un lenguaje**, que se combinan para formar *secuencias o cadenas* muy diversas.

Ejemplo de alfabeto para las bases de ADN

- Los símbolos del alfabeto $\textsc{Vadn} = \{A, C, G, T\}$ sirven para representar las bases que se conectan en una molécula de ADN, a saber: **A**denina, **C**itosina, **G**uanina, **T**imina.
- Las cadenas representan **secuencias de ADN**, como **GATTACA**.
- En *Bioinformática* es de vital interés **analizar secuencias de ADN**. La Teoría de Autómatas y Lenguajes Formales ayuda en parte.

El sentido de los símbolos

- Los símbolos no siempre representan caracteres de una cadena de texto que se desea procesar.
- Los símbolos son entidades **abstractas** que se usan para representar (*simbolizar*) **componentes indivisibles de un lenguaje**, que se combinan para formar *secuencias o cadenas* muy diversas.

Ejemplo de alfabeto para las bases de ADN

- Los símbolos del alfabeto $\textsc{Vadn} = \{A, C, G, T\}$ sirven para representar las bases que se conectan en una molécula de ADN, a saber: **A**denina, **C**itosina, **G**uanina, **T**imina.
- Las cadenas representan **secuencias de ADN**, como **GATTACA**.
- En *Bioinformática* es de vital interés **analizar secuencias de ADN**. La Teoría de Autómatas y Lenguajes Formales ayuda en parte.

Ejemplo de alfabeto de eventos

Podemos usar **nombres significativos** para **símbolos que representan eventos** que se producen en una ventana gráfica:
Ej.: $\textsc{Vevboton} = \{aceptar, cancelar, cerrar\}$.

Lenguaje universal

- El **lenguaje universal** con alfabeto V, que se denota V^*, es el conjunto que contiene las cadenas de cualquier longitud que se forman con los símbolos del alfabeto V.
  Siempre contiene la cadena vacía: $\lambda \in V^*$
  - **Ej.:** Dado el alfabeto $V_{dig} = \{0, 1, 2, \ldots, 9\}$, en el lenguaje universal V_{dig}^* están todas las cadenas que representan números naturales en notación decimal y además la cadena vacía.
  - Tenemos que $\lambda, 357, 25, \underline{09802} \in V_{dig}^*$.
  - Es incorrecto escribir $245 \in V_{dig}$.
    Es correcto escribir $2 \in V_{dig}^*$ y también $2 \in V_{dig}$.

Lenguaje universal

- El **lenguaje universal** con alfabeto V, que se denota V^*, es el conjunto que contiene las cadenas de cualquier longitud que se forman con los símbolos del alfabeto V.
  Siempre contiene la cadena vacía: $\lambda \in V^*$
  - **Ej.:** Dado el alfabeto $V_{dig} = \{0, 1, 2, \ldots, 9\}$, en el lenguaje universal V_{dig}^* están todas las cadenas que representan números naturales en notación decimal y además la cadena vacía.
  - Tenemos que $\lambda, 357, 25, \underline{09802} \in V_{dig}^*$.
  - Es incorrecto escribir $245 \in V_{dig}$.
    Es correcto escribir $2 \in V_{dig}^*$ y también $2 \in V_{dig}$.
- V^+ es el conjunto de las cadenas con alfabeto V de longitud mayor o igual que 1.
  Se cumple que: $V^* = V^+ \cup \{\lambda\}$.

Concatenación de cadenas

La **operación fundamental** con cadenas de cierto lenguaje universal V^* es la **concatenación**:

$$\circ : V^* \times V^* \longrightarrow V^*$$

Ej.: dado un alfabeto $V = \{a, b\}$ y las cadenas $x = abb$, $y = aaaa$, entonces $\mathbf{x} \circ \mathbf{y} = \mathbf{xy} = \mathbf{abbaaaa}$

Concatenación de cadenas

La **operación fundamental** con cadenas de cierto lenguaje universal V^* es la **concatenación**:

$$\circ : V^* \times V^* \longrightarrow V^*$$

Ej.: dado un alfabeto $V = \{a, b\}$ y las cadenas $x = abb$, $y = aaaa$, entonces $\mathbf{x \circ y = xy = abbaaaa}$

Propiedades de la concatenación

- **Propiedad asociativa:** $\forall x, y, z \in V^* : \ x \circ (y \circ z) = (x \circ y) \circ z$
- **Elemento identidad λ:** $\forall x \in V^* : \ \lambda \circ x = x \circ \lambda = x$

Concatenación de cadenas

La **operación fundamental** con cadenas de cierto lenguaje universal V^* es la **concatenación**:

$$\circ : V^* \times V^* \longrightarrow V^*$$

Ej.: dado un alfabeto $V = \{a, b\}$ y las cadenas $x = abb$, $y = aaaa$, entonces $\mathbf{x \circ y = xy = abbaaaa}$

Propiedades de la concatenación

- **Propiedad asociativa:** $\forall x, y, z \in V^* : x \circ (y \circ z) = (x \circ y) \circ z$
- **Elemento identidad λ:** $\forall x \in V^* : \lambda \circ x = x \circ \lambda = x$

Subcadena, prefijo y sufijo

- z es **subcadena** de w si $\exists\, x, y : w = xzy$
  Ej.: $ba, abab, \lambda$ son subcadenas de $a\underline{ba}b$

Concatenación de cadenas

La **operación fundamental** con cadenas de cierto lenguaje universal V^* es la **concatenación**:

$$\circ : V^* \times V^* \longrightarrow V^*$$

<u>**Ej.:**</u> dado un alfabeto $V = \{a, b\}$ y las cadenas $x = abb$, $y = aaaa$, entonces $\mathbf{x} \circ \mathbf{y} = \mathbf{xy} = \mathbf{abbaaaa}$

Propiedades de la concatenación

- **Propiedad asociativa:** $\forall x, y, z \in V^* : \; x \circ (y \circ z) = (x \circ y) \circ z$
- **Elemento identidad λ:** $\forall x \in V^* : \; \lambda \circ x = x \circ \lambda = x$

Subcadena, prefijo y sufijo

- z es **subcadena** de w si $\exists\, x, y : w = xzy$
  <u>**Ej.:**</u> $ba, abab, \lambda$ son subcadenas de $\underline{a\underline{ba}b}$
- z es **prefijo** de w si $\exists\, x : \; w = zx$ y es **sufijo** si $\exists\, x : \; w = xz$
  <u>**Ej.:**</u> ab es prefijo de $\underline{ab}ab$ y es es sufijo de $a\underline{bab}$

Concatenación de cadenas

La **operación fundamental** con cadenas de cierto lenguaje universal V^* es la **concatenación**:

$$\circ : V^* \times V^* \longrightarrow V^*$$

Ej.: dado un alfabeto $V = \{a, b\}$ y las cadenas $x = abb$, $y = aaaa$, entonces $\mathbf{x \circ y = xy = abbaaaa}$

Propiedades de la concatenación

- **Propiedad asociativa:** $\forall x, y, z \in V^* : \; x \circ (y \circ z) = (x \circ y) \circ z$
- **Elemento identidad λ:** $\forall x \in V^* : \; \lambda \circ x = x \circ \lambda = x$

Subcadena, prefijo y sufijo

- z es **subcadena** de w si $\exists x, y : w = xzy$
- z es **prefijo** de w si $\exists x : \; w = zx$ y es **sufijo** si $\exists x : \; w = xz$
- λ es subcadena, prefijo y sufijo **de cualquier cadena.**
- **Toda cadena** es subcadena, prefijo y sufijo **de sí misma.**

Potencia de cadenas

w^n denota la **potencia n-ésima** de la cadena w y se define **formalmente de forma recursiva** como:

1. REGLA BASE: para $n = 0 \quad \xrightarrow{def} \quad w^0 = \lambda$
2. REGLA RECURSIVA: si $n > 0$ entonces $w^n = w \circ w^{n-1}$

Potencia de cadenas

w^n denota la **potencia n-ésima** de la cadena w y se define **formalmente de forma recursiva** como:

1. REGLA BASE: para $n = 0 \xrightarrow{def} w^0 = \lambda$
2. REGLA RECURSIVA: si $n > 0$ entonces $w^n = w \circ w^{n-1}$

Ejemplo de cálculo de potencia para $w = aba$

$w^0 = (aba)^0 = \lambda$ **regla base**

Potencia de cadenas

w^n denota la **potencia n-ésima** de la cadena w y se define **formalmente de forma recursiva** como:

1. REGLA BASE: para $n = 0 \quad \xrightarrow{def} \quad w^0 = \lambda$
2. REGLA RECURSIVA: si $n > 0$ entonces $w^n = w \circ w^{n-1}$

Ejemplo de cálculo de potencia para $w = aba$

$w^0 = (aba)^0 = \lambda$ **regla base**
$w^1 = w \circ w^0 =$

Potencia de cadenas

w^n denota la **potencia n-ésima** de la cadena w y se define **formalmente de forma recursiva** como:

1. REGLA BASE: para $n = 0 \xrightarrow{def}$ $w^0 = \lambda$
2. REGLA RECURSIVA: si $n > 0$ entonces $w^n = w \circ w^{n-1}$

Ejemplo de cálculo de potencia para $w = aba$

$w^0 = (aba)^0 = \lambda$ **regla base**
$w^1 = w \circ w^0 = aba \circ \lambda = aba$

Potencia de cadenas

$\boxed{w^n}$ denota la **potencia n-ésima** de la cadena w y se define **formalmente de forma recursiva** como:

1. REGLA BASE: para $n = 0 \quad \xrightarrow{def} \quad \boxed{w^0 = \lambda}$
2. REGLA RECURSIVA: si $n > 0$ entonces $\boxed{w^n = w \circ w^{n-1}}$

Ejemplo de cálculo de potencia para $w = aba$

$$w^0 = (aba)^0 = \lambda$$
$$w^1 = w \circ w^0 = aba \circ \lambda = aba$$
$$w^2 = w \circ w^1 =$$

Potencia de cadenas

$\boxed{w^n}$ denota la **potencia n-ésima** de la cadena w y se define **formalmente de forma recursiva** como:

1. REGLA BASE: para $n = 0 \quad \xrightarrow{def} \quad \boxed{w^0 = \lambda}$
2. REGLA RECURSIVA: si $n > 0$ entonces $\boxed{w^n = w \circ w^{n-1}}$

Ejemplo de cálculo de potencia para $w = aba$

$$
\begin{aligned}
w^0 &= (aba)^0 = \lambda \\
w^1 &= w \circ w^0 = aba \circ \lambda = aba \\
w^2 &= w \circ w^1 = aba \circ aba = abaaba
\end{aligned}
$$

Potencia de cadenas

w^n denota la **potencia n-ésima** de la cadena w y se define **formalmente de forma recursiva** como:

1. REGLA BASE: para $n = 0 \quad \xrightarrow{def} \quad w^0 = \lambda$
2. REGLA RECURSIVA: si $n > 0$ entonces $w^n = w \circ w^{n-1}$

Ejemplo de cálculo de potencia para $w = aba$

$$
\begin{aligned}
w^0 &= (aba)^0 = \lambda \\
w^1 &= w \circ w^0 = aba \circ \lambda = aba \\
w^2 &= w \circ w^1 = aba \circ aba = abaaba \\
w^3 &= w \circ w^2 =
\end{aligned}
$$

Potencia de cadenas

w^n denota la **potencia n-ésima** de la cadena w y se define **formalmente de forma recursiva** como:

1. REGLA BASE: para $n = 0 \quad \xrightarrow{def} \quad w^0 = \lambda$
2. REGLA RECURSIVA: si $n > 0$ entonces $w^n = w \circ w^{n-1}$

Ejemplo de cálculo de potencia para $w = aba$

$w^0 = (aba)^0 = \lambda$
$w^2 = w \circ w^1 = aba \circ aba = abaaba$
$w^3 = w \circ w^2 = aba \circ abaaba = abaabaaba$

Potencia de cadenas

$\boxed{w^n}$ denota la **potencia n-ésima** de la cadena w y se define **formalmente de forma recursiva** como:

1. REGLA BASE: para $n = 0 \xrightarrow{def} \boxed{w^0 = \lambda}$
2. REGLA RECURSIVA: si $n > 0$ entonces $\boxed{w^n = w \circ w^{n-1}}$

Ejemplo de cálculo de potencia para $w = aba$

$$w^0 = (aba)^0 = \lambda$$
$$w^2 = w \circ w^1 = aba \circ aba = abaaba$$
$$w^3 = w \circ w^2 = aba \circ abaaba = abaabaaba$$

La potencia tiene mayor precedencia que la concatenación

Ej.: Para la **cadena** $w = 234$ se tiene:

- $w^2 = (234)^2 = 234 \circ 234 = 234234$
  Sin paréntesis es: $234^2 = 23 \circ (4)^2 = 23 \circ 44 = 2344$.
- $(234)^0 = \lambda$; si fuera potencia numérica sería $(234)^0 = 1$.

Reflexión o inversión de cadenas

w^R denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. REGLA BASE, para $w = \lambda \xrightarrow{def}$ $w^R = \lambda^R = \lambda$
2. R. RECURSIVA, si $w = a \circ z, a \in V, z \in V^*$ entonces $w^R = (a \circ z)^R = z^R \circ a$

La reflexión tiene mayor precedencia que la concatenación.

Reflexión o inversión de cadenas

w^R denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. Regla base, para $w = \lambda \xrightarrow{def} w^R = \lambda^R = \lambda$
2. R. recursiva, si $w = a \circ z, a \in V, z \in V^*$ entonces $w^R = (a \circ z)^R = z^R \circ a$

La reflexión tiene mayor precedencia que la concatenación.

Ejemplo de cálculo descendente recursivo de $(abb)^R$

Comprobamos que $(aab)^R = baa$ según definición recursiva:

Reflexión o inversión de cadenas

w^R denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. Regla base, para $w = \lambda \xrightarrow{def} w^R = \lambda^R = \lambda$
2. R. recursiva, si $w = a \circ z, a \in V, z \in V^*$ entonces $w^R = (a \circ z)^R = z^R \circ a$

La reflexión tiene mayor precedencia que la concatenación.

Ejemplo de cálculo descendente recursivo de $(abb)^R$

Comprobamos que $(aab)^R = baa$ según definición recursiva:

$$(aab)^R =$$

Reflexión o inversión de cadenas

w^R denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. REGLA BASE, para $w = \lambda \xrightarrow{def} w^R = \lambda^R = \lambda$
2. R. RECURSIVA, si $w = a \circ z, a \in V, z \in V^*$ entonces $w^R = (a \circ z)^R = z^R \circ a$

La reflexión tiene mayor precedencia que la concatenación.

Ejemplo de cálculo descendente recursivo de $(abb)^R$

Comprobamos que $(aab)^R = baa$ según definición recursiva:

$$(aab)^R = (a \circ ab)^R \overbrace{=}^{def}$$

Reflexión o inversión de cadenas

w^R denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. Regla base, para $w = \lambda \xrightarrow{def} w^R = \lambda^R = \lambda$
2. R. recursiva, si $w = a \circ z, a \in V, z \in V^*$ entonces $w^R = (a \circ z)^R = z^R \circ a$

La reflexión tiene mayor precedencia que la concatenación.

Ejemplo de cálculo descendente recursivo de $(abb)^R$

Comprobamos que $(aab)^R = baa$ según definición recursiva:

$$(aab)^R = (a \circ ab)^R \overbrace{=}^{def} (ab)^R \circ a = (a \circ b)^R \circ a =$$

Reflexión o inversión de cadenas

$\boxed{w^R}$ denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. REGLA BASE, para $w = \lambda \xrightarrow{def} \boxed{w^R = \lambda^R = \lambda}$
2. R. RECURSIVA, si $w = a \circ z, a \in V, z \in V^*$ entonces $\boxed{w^R = (a \circ z)^R = z^R \circ a}$

La reflexión tiene mayor precedencia que la concatenación.

Ejemplo de cálculo descendente recursivo de $(abb)^R$

Comprobamos que $(aab)^R = baa$ según definición recursiva:

$$(aab)^R = (a \circ ab)^R \overbrace{=}^{def} (ab)^R \circ a = (a \circ b)^R \circ a = (b^R) \circ a \circ a =$$

Reflexión o inversión de cadenas

w^R denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. REGLA BASE, para $w = \lambda \xrightarrow{def} w^R = \lambda^R = \lambda$
2. R. RECURSIVA, si $w = a \circ z, a \in V, z \in V^*$ entonces $w^R = (a \circ z)^R = z^R \circ a$

La reflexión tiene mayor precedencia que la concatenación.

Ejemplo de cálculo descendente recursivo de $(abb)^R$

Comprobamos que $(aab)^R = baa$ según definición recursiva:

$$(aab)^R = (a \circ ab)^R \overbrace{=}^{def} (ab)^R \circ a = (a \circ b)^R \circ a = (b^R) \circ a \circ a = (\lambda^R) \circ b \circ a \circ a =$$

Reflexión o inversión de cadenas

w^R denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. REGLA BASE, para $w = \lambda \xrightarrow{def} \boxed{w^R = \lambda^R = \lambda}$
2. R. RECURSIVA, si $w = a \circ z, a \in V, z \in V^*$ entonces $\boxed{w^R = (a \circ z)^R = z^R \circ a}$

La reflexión tiene mayor precedencia que la concatenación.

Ejemplo de cálculo descendente recursivo de $(abb)^R$

Comprobamos que $(aab)^R = baa$ según definición recursiva:

$$(aab)^R = (a \circ ab)^R \overbrace{=}^{def} (ab)^R \circ a = (a \circ b)^R \circ a = (b^R) \circ a \circ a = (\lambda^R) \circ b \circ a \circ a = \lambda \circ b \circ a \circ a$$

Reflexión o inversión de cadenas

w^R denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. REGLA BASE, para $w = \lambda \xrightarrow{def} w^R = \lambda^R = \lambda$
2. R. RECURSIVA, si $w = a \circ z, a \in V, z \in V^*$ entonces $w^R = (a \circ z)^R = z^R \circ a$

La reflexión tiene mayor precedencia que la concatenación.

Ejemplo de cálculo descendente recursivo de $(abb)^R$

Comprobamos que $(aab)^R = baa$ según definición recursiva:

$$(aab)^R = (a \circ ab)^R \overbrace{=}^{def} (ab)^R \circ a = (a \circ b)^R \circ a = (b^R) \circ a \circ a =$$
$$(\lambda^R) \circ b \circ a \circ a = \lambda \circ b \circ a \circ a \overbrace{=}^{resolver}$$

Reflexión o inversión de cadenas

w^R denota la **cadena refleja** de w y se define **formalmente de forma recursiva** como:

1. REGLA BASE, para $w = \lambda \xrightarrow{def} \boxed{w^R = \lambda^R = \lambda}$
2. R. RECURSIVA, si $w = a \circ z, a \in V, z \in V^*$ entonces $\boxed{w^R = (a \circ z)^R = z^R \circ a}$

La reflexión tiene mayor precedencia que la concatenación.

Ejemplo de cálculo descendente recursivo de $(abb)^R$

Comprobamos que $(aab)^R = baa$ según definición recursiva:

$$(aab)^R = (a \circ ab)^R \overbrace{=}^{def} (ab)^R \circ a = (a \circ b)^R \circ a = (b^R) \circ a \circ a =$$

$$(\lambda^R) \circ b \circ a \circ a = \lambda \circ b \circ a \circ a \overbrace{=}^{resolver} baa$$

Lenguajes. Descripción matemática

- Un **lenguaje** L con alfabeto V es un *subconjunto* del **lenguaje universal** V^*: $L \subseteq V^*$

Lenguajes. Descripción matemática

- Un **lenguaje** L con alfabeto V es un *subconjunto* del **lenguaje universal** V^*: $L \subseteq V^*$
- El **lenguaje vacío** $\varnothing$ y el propio **alfabeto** V pueden considerarse lenguajes, por ser subconjuntos de V^*.

Lenguajes. Descripción matemática

- Un **lenguaje** L con alfabeto V es un *subconjunto* del **lenguaje universal** V^*: $\boxed{L \subseteq V^*}$
- El **lenguaje vacío** $\varnothing$ y el propio **alfabeto** V pueden considerarse lenguajes, por ser subconjuntos de V^*.
- Un **lenguaje formal** es un lenguaje cuyas cadenas **cumplen cierta propiedad que puede describirse de manera precisa**.

Lenguajes. Descripción matemática

- Un **lenguaje** L con alfabeto V es un *subconjunto* del **lenguaje universal** V^*: $L \subseteq V^*$
- El **lenguaje vacío** $\varnothing$ y el propio **alfabeto** V pueden considerarse lenguajes, por ser subconjuntos de V^*.
- Un **lenguaje formal** es un lenguaje cuyas cadenas **cumplen cierta propiedad que puede describirse de manera precisa**.

Lenguajes. Descripción matemática

- Un **lenguaje** L con alfabeto V es un *subconjunto* del **lenguaje universal** V^*: $L \subseteq V^*$
- El **lenguaje vacío** $\varnothing$ y el propio **alfabeto** V pueden considerarse lenguajes, por ser subconjuntos de V^*.
- Un **lenguaje formal** es un lenguaje cuyas cadenas **cumplen cierta propiedad que puede describirse de manera precisa**.

Descripción formal de lenguajes como conjuntos

- **Definición por extensión**. Sólo sirve para lenguajes finitos.

  Cardinalidad de L, se denota $|L|$: número de cadenas en L.

  Ej.: lenguaje BINLON2: " está formado por todas las cadenas de dígitos binarios de longitud 2" (**descripción informal**).

Lenguajes. Descripción matemática

- Un **lenguaje** L con alfabeto V es un *subconjunto* del **lenguaje universal** V^*: $L \subseteq V^*$
- El **lenguaje vacío** $\varnothing$ y el propio **alfabeto** V pueden considerarse lenguajes, por ser subconjuntos de V^*.
- Un **lenguaje formal** es un lenguaje cuyas cadenas **cumplen cierta propiedad que puede describirse de manera precisa**.

Descripción formal de lenguajes como conjuntos

- **Definición por extensión**. Sólo sirve para lenguajes finitos.

  Cardinalidad de L, se denota $|L|$: número de cadenas en L.

  Ej.: lenguaje $\textsc{BinLon2}$: " está formado por todas las cadenas de dígitos binarios de longitud 2" (**descripción informal**).

  Definición por extensión: $\textsc{BinLon2} = \{00, 01, 10, 11\}$; $|\textsc{BinLon2}| = 4$

Lenguajes. Descripción matemática

- Un **lenguaje** L con alfabeto V es un *subconjunto* del **lenguaje universal** V^*: $L \subseteq V^*$
- El **lenguaje vacío** $\varnothing$ y el propio **alfabeto** V pueden considerarse lenguajes, por ser subconjuntos de V^*.
- Un **lenguaje formal** es un lenguaje cuyas cadenas **cumplen cierta propiedad que puede describirse de manera precisa**.

Descripción formal de lenguajes como conjuntos

- **Definición por extensión**. Sólo sirve para lenguajes finitos.

  Cardinalidad de L, se denota $|L|$: número de cadenas en L.

  Ej.: lenguaje $\textsc{BinLon2}$: " está formado por todas las cadenas de dígitos binarios de longitud 2" (**descripción informal**).

  Definición por extensión: $\textsc{BinLon2} = \{00, 01, 10, 11\}$; $|\textsc{BinLon2}| = 4$
- **Definición por comprensión**: $L = \{w \in V^* \mid P(w)\}$

Ejemplo: de descripción informal a descripción por comprensión

BinIg consiste en (descripción informal):

> *"cadenas de ceros y unos que tienen igual número de ceros que de unos"*

Ejemplo: de descripción informal a descripción por comprensión

BinIg consiste en (descripción informal):

> *"cadenas de ceros y unos que tienen igual número de ceros que de unos"*

Descripción matemática por comprensión:

$$\textsc{BinIg} = \left\{ w \in \{0,1\}^* \mid \boxed{\text{ceros(w)=unos(w)}} \right\}$$

$\textsc{BinIg} \subseteq \{0,1\}^*$

Ejemplo: de descripción informal a descripción por comprensión

BinIg consiste en (descripción informal):

> *"cadenas de ceros y unos que tienen igual número de ceros que de unos"*

Descripción matemática por comprensión:

$$\textsc{BinIg} = \left\{ w \in \{0,1\}^* \mid \text{ceros(w)=unos(w)} \right\}$$

$\textsc{BinIg} \subseteq \{0,1\}^*$

Ejemplo: simplificando las descripciones por comprensión

$$L = \{w \in \{a,b\}^* \mid w = a^n b \wedge n \geq 0\}$$

Ejemplo: de descripción informal a descripción por comprensión

BinIg consiste en (descripción informal):

> *"cadenas de ceros y unos que tienen igual número de ceros que de unos"*

Descripción matemática por comprensión:

$$\textsc{BinIg} = \left\{ w \in \{0,1\}^* \mid \boxed{\text{ceros(w)=unos(w)}} \right\}$$

$\textsc{BinIg} \subseteq \{0,1\}^*$

Ejemplo: simplificando las descripciones por comprensión

$$L = \{w \in \{a,b\}^* \mid w = a^n b \wedge n \geq 0\}$$

Se **simplifica** como: $L = \{a^n b \mid n \geq 0\}$

Ejemplo: de descripción informal a descripción por comprensión

BinIg consiste en (descripción informal):

> *"cadenas de ceros y unos que tienen igual número de ceros que de unos"*

Descripción matemática por comprensión:

$$\textsc{BinIg} = \left\{ w \in \{0,1\}^* \mid \text{ceros(w)=unos(w)} \right\}$$

$\textsc{BinIg} \subseteq \{0,1\}^*$

Ejemplo: simplificando las descripciones por comprensión

$$L = \{w \in \{a,b\}^* \mid w = a^n b \land n \geq 0\}$$

Se **simplifica** como: $L = \{a^n b \mid n \geq 0\}$
Generar cadenas: $n = 0 \rightsquigarrow a^0 b = b$, $n = 1 \rightsquigarrow ab$, $n = 2 \rightsquigarrow aab$, etc., luego $b, ab, aab \in L$

Operaciones con lenguajes como conjuntos

- **Unión:** $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$

Operaciones con lenguajes como conjuntos

- **Unión:** $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$
- **Intersección:** $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$

Operaciones con lenguajes como conjuntos

- **Unión:** $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$
- **Intersección:** $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$

  Ej.: Sea $\textsc{BinImpar} = \{z1 \mid z \in \{0,1\}^*\}$. Tenemos que:

  $\textsc{BinIg} \cap \textsc{BinImpar} = \{w \mid w \in \textsc{BinIg} \wedge w \in \textsc{BinImpar}\}$

  Descripción por comprensión (explícita):

  $\textsc{BinIg} \cap \textsc{BinImpar} = \{w \in \{0,1\}^* \mid ceros(w) = unos(w), w = z1\}$

Operaciones con lenguajes como conjuntos

- **Unión:** $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$
- **Intersección:** $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$

  Ej.: Sea $\textsc{BinImpar} = \{z1 \mid z \in \{0,1\}^*\}$. Tenemos que:

  $\textsc{BinIg} \cap \textsc{BinImpar} = \{w \mid w \in \textsc{BinIg} \wedge w \in \textsc{BinImpar}\}$

  Descripción por comprensión (explícita):
  $\textsc{BinIg} \cap \textsc{BinImpar} = \{w \in \{0,1\}^* \mid ceros(w) = unos(w), w = z1\}$
- **Diferencia:** $L_1 - L_2 = \{w \mid w \in L_1 \wedge w \notin L_2\}$

Operaciones con lenguajes como conjuntos

- **Unión:** $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$
- **Intersección:** $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$

  Ej.: Sea $\textsc{BinImpar} = \{z1 \mid z \in \{0,1\}^*\}$. Tenemos que:

  $\textsc{BinIg} \cap \textsc{BinImpar} = \{w \mid w \in \textsc{BinIg} \wedge w \in \textsc{BinImpar}\}$

  Descripción por comprensión (explícita):

  $\textsc{BinIg} \cap \textsc{BinImpar} = \{w \in \{0,1\}^* \mid ceros(w) = unos(w), w = z1\}$
- **Diferencia:** $L_1 - L_2 = \{w \mid w \in L_1 \wedge w \notin L_2\}$

  $\textsc{BinIg} - \textsc{BinImpar} =$

  $\{w \in \{0,1\}^* \mid (ceros(w) = unos(w) \wedge w = z0) \vee w = \lambda\}$

Operaciones con lenguajes como conjuntos

- **Unión:** $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$
- **Intersección:** $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$

  Ej.: Sea $\textsc{BinImpar} = \{z1 \mid z \in \{0,1\}^*\}$. Tenemos que:

  $\textsc{BinIg} \cap \textsc{BinImpar} = \{w \mid w \in \textsc{BinIg} \wedge w \in \textsc{BinImpar}\}$

  Descripción por comprensión (explícita):

  $\textsc{BinIg} \cap \textsc{BinImpar} = \{w \in \{0,1\}^* \mid ceros(w) = unos(w), w = z1\}$
- **Diferencia:** $L_1 - L_2 = \{w \mid w \in L_1 \wedge w \notin L_2\}$

  $\textsc{BinIg} - \textsc{BinImpar} =$

  $\{w \in \{0,1\}^* \mid (ceros(w) = unos(w) \wedge w = z0) \vee w = \lambda\}$
- **Complementario (respecto de V^*):** $\overline{L} = V^* - L$

$$\overline{\textsc{BinIg}} = \{0,1\}^* - \textsc{BinIg} = \{w \in \{0,1\}^* \mid ceros(w) \neq unos(w)\}$$

Propiedades de las operaciones de lenguajes como conjuntos

- **Asociativa:** $L_1 \cap (L_2 \cap L_3) = (L_1 \cap L_2) \cap L_3$ y $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- **Conmutativa:** $L_1 \cap L_2 = L_2 \cap L_1$ y $L_1 \cup L_2 = L_2 \cup L_1$

Propiedades de las operaciones de lenguajes como conjuntos

- **Asociativa:** $L_1 \cap (L_2 \cap L_3) = (L_1 \cap L_2) \cap L_3$ y $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- **Conmutativa:** $L_1 \cap L_2 = L_2 \cap L_1$ y $L_1 \cup L_2 = L_2 \cup L_1$
- **Distributiva:**
  1. Intersección respecto de unión: $L_1 \cap (L_2 \cup L_3) = (L_1 \cap L_2) \cup (L_1 \cap L_3)$
  2. Unión respecto de intersección: $L_1 \cup (L_2 \cap L_3) = (L_1 \cup L_2) \cap (L_1 \cup L_3)$

Propiedades de las operaciones de lenguajes como conjuntos

- **Asociativa:** $L_1 \cap (L_2 \cap L_3) = (L_1 \cap L_2) \cap L_3$ y $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- **Conmutativa:** $L_1 \cap L_2 = L_2 \cap L_1$ y $L_1 \cup L_2 = L_2 \cup L_1$
- **Distributiva:**
  1. Intersección respecto de unión: $L_1 \cap (L_2 \cup L_3) = (L_1 \cap L_2) \cup (L_1 \cap L_3)$
  2. Unión respecto de intersección: $L_1 \cup (L_2 \cap L_3) = (L_1 \cup L_2) \cap (L_1 \cup L_3)$
- **Leyes de De Morgan:** $\overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$ y $\overline{L_1 \cup L_2} = \overline{L_1} \cap \overline{L_2}$
- **Propiedades del complementario:**

$$\overline{L_1} \cap L_1 = \varnothing, \quad \overline{L_1} \cup L_1 = V^* \quad y \quad \overline{\overline{L_1}} = L_1$$

Propiedades de las operaciones de lenguajes como conjuntos

- **Asociativa:** $L_1 \cap (L_2 \cap L_3) = (L_1 \cap L_2) \cap L_3$ y $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- **Conmutativa:** $L_1 \cap L_2 = L_2 \cap L_1$ y $L_1 \cup L_2 = L_2 \cup L_1$
- **Distributiva:**
  1. Intersección respecto de unión: $L_1 \cap (L_2 \cup L_3) = (L_1 \cap L_2) \cup (L_1 \cap L_3)$
  2. Unión respecto de intersección: $L_1 \cup (L_2 \cap L_3) = (L_1 \cup L_2) \cap (L_1 \cup L_3)$
- **Leyes de De Morgan:** $\overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$ y $\overline{L_1 \cup L_2} = \overline{L_1} \cap \overline{L_2}$
- **Propiedades del complementario:**

$$\overline{L_1} \cap L_1 = \varnothing, \quad \overline{L_1} \cup L_1 = V^* \quad y \quad \overline{\overline{L_1}} = L_1$$

- **Propiedades del lenguaje vacío:** $L_1 \cap \varnothing = \varnothing$ y $L_1 \cup \varnothing = L_1$
- **Propiedades del lenguaje universal:**

$$L_1 \cap V^* = L_1 \quad y \quad L_1 \cup V^* = V^*$$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} =$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} = \{ab \circ cc, ab \circ a,$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} = \{ab \circ cc, ab \circ a, c \circ cc, c \circ a,$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} = \{ab \circ cc, ab \circ a, c \circ cc, c \circ a, \lambda \circ cc, \lambda \circ a\} =$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $\boxed{L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} = \{ab \circ cc, ab \circ a, c \circ cc, c \circ a, \lambda \circ cc, \lambda \circ a\} = \{abcc, aba, ccc, ca, cc, a\}$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} = \{ab \circ cc, ab \circ a, c \circ cc, c \circ a, \lambda \circ cc, \lambda \circ a\} = \{abcc, aba, ccc, ca, cc, a\}$

- **Propiedades**
  - **Elemento nulo**

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} = \{ab \circ cc, ab \circ a, c \circ cc, c \circ a, \lambda \circ cc, \lambda \circ a\} = \{abcc, aba, ccc, ca, cc, a\}$

- **Propiedades**
  - **Elemento nulo $\varnothing$:** $L \circ \varnothing = \varnothing$
  - **Elemento identidad**

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} = \{ab \circ cc, ab \circ a, c \circ cc, c \circ a, \lambda \circ cc, \lambda \circ a\} = \{abcc, aba, ccc, ca, cc, a\}$

- **Propiedades**
  - **Elemento nulo $\varnothing$:** $L \circ \varnothing = \varnothing$
  - **Elemento identidad $\{\lambda\}$:** $L \circ \{\lambda\} = \{\lambda\} \circ L = L$
  - **Propiedad asociativa:**

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $\boxed{L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} = \{ab \circ cc, ab \circ a, c \circ cc, c \circ a, \lambda \circ cc, \lambda \circ a\} = \{abcc, aba, ccc, ca, cc, a\}$

- **Propiedades**
  - **Elemento nulo $\varnothing$:** $L \circ \varnothing = \varnothing$
  - **Elemento identidad $\{\lambda\}$:** $L \circ \{\lambda\} = \{\lambda\} \circ L = L$
  - **Propiedad asociativa:** $(L_1 \circ L_2) \circ L_3 = L_1 \circ (L_2 \circ L_3)$
  - **Distributiva de la concatenación de lenguajes con respecto a...**

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$

  Ej.: Calcular la concatenación:
  $\{ab, c, \lambda\} \circ \{cc, a\} = \{ab \circ cc, ab \circ a, c \circ cc, c \circ a, \lambda \circ cc, \lambda \circ a\} = \{abcc, aba, ccc, ca, cc, a\}$

- **Propiedades**
  - **Elemento nulo $\varnothing$:** $L \circ \varnothing = \varnothing$
  - **Elemento identidad $\{\lambda\}$:** $L \circ \{\lambda\} = \{\lambda\} \circ L = L$
  - **Propiedad asociativa:** $(L_1 \circ L_2) \circ L_3 = L_1 \circ (L_2 \circ L_3)$
  - **Distributiva de la concatenación de lenguajes con respecto a... la unión:**
    1. $L_1 \circ (L_2 \cup L_3) = (L_1 \circ L_2) \cup (L_1 \circ L_3)$
    2. $(L_2 \cup L_3) \circ L_1 = (L_2 \circ L_1) \cup (L_3 \circ L_1)$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. Regla base: para $n = 0 \xrightarrow{def} L^0 = \{\lambda\}$
  2. Regla recursiva: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. Regla base: para $n = 0 \quad \xrightarrow{def} \quad L^0 = \{\lambda\}$
  2. Regla recursiva: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$A^0 = \{\lambda\}$ (regla base)

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. Regla base: para $n = 0 \xrightarrow{def} L^0 = \{\lambda\}$
  2. Regla recursiva: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$A^0 = \{\lambda\}$ (regla base)
$A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} =$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. REGLA BASE: para $n = 0 \xrightarrow{def} L^0 = \{\lambda\}$
  2. REGLA RECURSIVA: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$$A^0 = \{\lambda\} \text{ (regla base)}$$
$$A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}$$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $\boxed{L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}}$
- **Potencia:** se define L^n de forma recursiva como
  1. REGLA BASE: para $n = 0 \quad \xrightarrow{def} \quad \boxed{L^0 = \{\lambda\}}$
  2. REGLA RECURSIVA: si $n > 0$ entonces $\boxed{L^n = L \circ L^{n-1}}$

Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$$A^0 = \{\lambda\} \;\; \text{(regla base)}$$
$$A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}$$
$$A^2 = A \circ A^1 = \{ab, c\} \circ \{ab, c\} =$$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. Regla base: para $n = 0 \xrightarrow{def} L^0 = \{\lambda\}$
  2. Regla recursiva: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$$A^0 = \{\lambda\} \text{ (regla base)}$$
$$A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}$$
$$A^2 = A \circ A^1 = \{ab, c\} \circ \{ab, c\} = \{abab, abc,$$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $\boxed{L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}}$
- **Potencia:** se define L^n de forma recursiva como
  1. REGLA BASE: para $n = 0$ $\xrightarrow{def}$ $\boxed{L^0 = \{\lambda\}}$
  2. REGLA RECURSIVA: si $n > 0$ entonces $\boxed{L^n = L \circ L^{n-1}}$

Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$$A^0 = \{\lambda\} \text{ (regla base)}$$
$$A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}$$
$$A^2 = A \circ A^1 = \{ab, c\} \circ \{ab, c\} = \{abab, abc, cab, cc\}$$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. REGLA BASE: para $n = 0 \quad \xrightarrow{def} \quad L^0 = \{\lambda\}$
  2. REGLA RECURSIVA: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$$A^0 = \{\lambda\} \text{ (regla base)}$$
$$A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}$$
$$A^2 = A \circ A^1 = \{ab, c\} \circ \{ab, c\} = \{abab, abc, cab, cc\}$$
$$A^3 = A \circ A^2 = \{ab, c\} \circ \{abab, abc, cab, cc\} =$$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. Regla base: para $n = 0 \xrightarrow{def} L^0 = \{\lambda\}$
  2. Regla recursiva: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$A^0 = \{\lambda\}$ (regla base)
$A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}$
$A^2 = A \circ A^1 = \{ab, c\} \circ \{ab, c\} = \{abab, abc, cab, cc\}$
$A^3 = A \circ A^2 = \{ab, c\} \circ \{abab, abc, cab, cc\} =$
$\{ababab, ababc, abcab, abcc,$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. Regla base: para $n = 0 \quad \xrightarrow{def} \quad L^0 = \{\lambda\}$
  2. Regla recursiva: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$$A^0 = \{\lambda\} \text{ (regla base)}$$
$$A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}$$
$$A^2 = A \circ A^1 = \{ab, c\} \circ \{ab, c\} = \{abab, abc, cab, cc\}$$
$$A^3 = A \circ A^2 = \{ab, c\} \circ \{abab, abc, cab, cc\} =$$
$$\{ababab, ababc, abcab, abcc, cabab, cabc, ccab, ccc\}$$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $\boxed{L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}}$
- **Potencia:** se define L^n de forma recursiva como
  1. REGLA BASE: para $n = 0 \quad \xrightarrow{def} \quad \boxed{L^0 = \{\lambda\}}$
  2. REGLA RECURSIVA: si $n > 0$ entonces $\boxed{L^n = L \circ L^{n-1}}$

  Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$$
\begin{aligned}
&A^0 = \{\lambda\} \ \text{(regla base)}\\
&A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}\\
&A^2 = A \circ A^1 = \{ab, c\} \circ \{ab, c\} = \{abab, abc, cab, cc\}\\
&A^3 = A \circ A^2 = \{ab, c\} \circ \{abab, abc, cab, cc\} =\\
&\{ababab, ababc, abcab, abcc, cabab, cabc, ccab, ccc\}
\end{aligned}
$$

- **Lenguaje reflejo:** $\boxed{L^R = \{w^R \mid w \in L\}}$

  Ej.: $\{ab, c\}^R =$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. REGLA BASE: para $n = 0 \xrightarrow{def} L^0 = \{\lambda\}$
  2. REGLA RECURSIVA: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

  Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$$A^0 = \{\lambda\} \text{ (regla base)}$$
$$A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}$$
$$A^2 = A \circ A^1 = \{ab, c\} \circ \{ab, c\} = \{abab, abc, cab, cc\}$$
$$A^3 = A \circ A^2 = \{ab, c\} \circ \{abab, abc, cab, cc\} = \{ababab, ababc, abcab, abcc, cabab, cabc, ccab, ccc\}$$

- **Lenguaje reflejo:** $L^R = \{w^R \mid w \in L\}$

  Ej.: $\{ab, c\}^R = \{(ab)^R, c^R\} =$

Operaciones que extienden las operaciones con cadenas

- **Concatenación:** $L_1 \circ L_2 = \{x \circ y \mid x \in L_1 \wedge y \in L_2\}$
- **Potencia:** se define L^n de forma recursiva como
  1. REGLA BASE: para $n = 0$ $\xrightarrow{def}$ $L^0 = \{\lambda\}$
  2. REGLA RECURSIVA: si $n > 0$ entonces $L^n = L \circ L^{n-1}$

  Ej.: Calculamos sucesivas potencias del lenguaje $A = \{ab, c\}$:

$$
\begin{aligned}
&A^0 = \{\lambda\} \text{ (regla base)}\\
&A^1 = A \circ A^0 = \{ab, c\} \circ \{\lambda\} = \{ab \circ \lambda, c \circ \lambda\} = \{ab, c\}\\
&A^2 = A \circ A^1 = \{ab, c\} \circ \{ab, c\} = \{abab, abc, cab, cc\}\\
&A^3 = A \circ A^2 = \{ab, c\} \circ \{abab, abc, cab, cc\} =\\
&\{ababab, ababc, abcab, abcc, cabab, cabc, ccab, ccc\}
\end{aligned}
$$

- **Lenguaje reflejo:** $L^R = \{w^R \mid w \in L\}$

  Ej.: $\{ab, c\}^R = \{(ab)^R, c^R\} = \{ba, c\}$

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L:
  $L^* = \bigcup_{n=0}^{\infty} L^n$ $\equiv \{w_1 w_2 \ldots w_k \mid k \geq 0,\ \text{donde cada } w_i \in L\}$
- **Interpretación:** L^* contiene todas las cadenas que se obtienen al concatenar cero o más cadenas de L.
- **Nota:** $\lambda \in L^*$ sea cual sea el lenguaje L.

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L:
  $L^* = \bigcup_{n=0}^{\infty} L^n \equiv \{w_1 w_2 \dots w_k \mid k \geq 0, \text{ donde cada } w_i \in L\}$
- **Interpretación:** L^* contiene todas las cadenas que se obtienen al concatenar cero o más cadenas de L.
- **Nota:** $\lambda \in L^*$ sea cual sea el lenguaje L.

  Ej.: Sea $C = \{00, 11\}$. Obtenemos la descripción de C^*:

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L:
  $L^* = \bigcup_{n=0}^{\infty} L^n \equiv \{w_1 w_2 \dots w_k \mid k \geq 0, \text{ donde cada } w_i \in L\}$
- **Interpretación:** L^* contiene todas las cadenas que se obtienen al concatenar cero o más cadenas de L.
- **Nota:** $\lambda \in L^*$ sea cual sea el lenguaje L.

  Ej.: Sea $C = \{00, 11\}$. Obtenemos la descripción de C^*:

$$C^* = \{w_1 w_2 \dots w_k \mid k \geq 0, \text{ donde cada } w_i \in \{00, 11\}\}$$

Ejemplos de cadenas de C^*: $\lambda, 00, 11, 0000, 0011, 1100, 1111, \dots$

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L:
  $L^* = \bigcup_{n=0}^{\infty} L^n \equiv \{w_1 w_2 \dots w_k \mid k \geq 0,\ \text{donde cada } w_i \in L\}$
- **Interpretación:** L^* contiene todas las cadenas que se obtienen al concatenar cero o más cadenas de L.
- **Nota:** $\lambda \in L^*$ sea cual sea el lenguaje L.

  Ej.: Sea $C = \{00, 11\}$. Obtenemos la descripción de C^*:

  $$C^* = \{w_1 w_2 \dots w_k \mid k \geq 0,\ \text{donde cada } w_i \in \{00, 11\}\}$$

  Ejemplos de cadenas de C^*: $\lambda, 00, 11, 0000, 0011, 1100, 1111, \dots$
- El **lenguaje universal** V^* puede definirse como:

  $$\boxed{V^* = \bigcup_{n=0}^{\infty} V^n}$$ el alfabeto V es un **generador** de V^*

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L:
  $L^* = \bigcup_{n=0}^{\infty} L^n \equiv \{w_1 w_2 \dots w_k \mid k \geq 0,\ \text{donde cada } w_i \in L\}$
- La **clausura positiva** (o **cierre positivo**) de L contiene todas las cadenas que se obtienen al concatenar una o más cadenas de L:
  $L^+ = \bigcup_{n=1}^{\infty} L^n \equiv \{w_1 w_2 \dots w_k \mid k > 0,\ \text{donde cada } w_i \in L\}$

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L:
  $L^* = \bigcup_{n=0}^{\infty} L^n \equiv \{w_1 w_2 \ldots w_k \mid k \geq 0, \text{ donde cada } w_i \in L\}$
- La **clausura positiva** (o **cierre positivo**) de L contiene todas las cadenas que se obtienen al concatenar una o más cadenas de L:
  $L^+ = \bigcup_{n=1}^{\infty} L^n \equiv \{w_1 w_2 \ldots w_k \mid k > 0, \text{ donde cada } w_i \in L\}$

Cálculo de clausura y clausura positiva

Dados los lenguajes $A = \{a\}$ y $B = \{b\}$, vamos a describir por comprensión $(A \circ B)^*$ y $(A \circ B)^+$.

$\Rightarrow$ Primero obtenemos $A \circ B = \{ab\}$.

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L: $L^* = \bigcup_{n=0}^{\infty} L^n \equiv \{w_1 w_2 \dots w_k \mid k \geq 0, \text{ donde cada } w_i \in L\}$
- La **clausura positiva** (o **cierre positivo**) de L contiene todas las cadenas que se obtienen al concatenar una o más cadenas de L: $\boxed{L^+ = \bigcup_{n=1}^{\infty} L^n} \equiv \{w_1 w_2 \dots w_k \mid k > 0, \text{ donde cada } w_i \in L\}$

Cálculo de clausura y clausura positiva

Dados los lenguajes $A = \{a\}$ y $B = \{b\}$, vamos a describir por comprensión $(A \circ B)^*$ y $(A \circ B)^+$.

$\rightrightarrows$ Primero obtenemos $A \circ B = \{ab\}$.

$\rightrightarrows$ **Clausura:** $(A \circ B)^* = \{ab\}^* =$

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L:
  $L^* = \bigcup_{n=0}^{\infty} L^n \equiv \{w_1 w_2 \ldots w_k \mid k \geq 0, \text{ donde cada } w_i \in L\}$
- La **clausura positiva** (o **cierre positivo**) de L contiene todas las cadenas que se obtienen al concatenar una o más cadenas de L:
  $L^+ = \bigcup_{n=1}^{\infty} L^n \equiv \{w_1 w_2 \ldots w_k \mid k > 0, \text{ donde cada } w_i \in L\}$

Cálculo de clausura y clausura positiva

Dados los lenguajes $A = \{a\}$ y $B = \{b\}$, vamos a describir por comprensión $(A \circ B)^*$ y $(A \circ B)^+$.

$\rightrightarrows$ Primero obtenemos $A \circ B = \{ab\}$.

$\rightrightarrows$ **Clausura:** $(A \circ B)^* = \{ab\}^* = \{(ab)^n \mid n \geq 0\}$.

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L:
  $L^* = \bigcup_{n=0}^{\infty} L^n \equiv \{w_1 w_2 \ldots w_k \mid k \geq 0, \text{ donde cada } w_i \in L\}$
- La **clausura positiva** (o **cierre positivo**) de L contiene todas las cadenas que se obtienen al concatenar una o más cadenas de L:
  $\boxed{L^+ = \bigcup_{n=1}^{\infty} L^n} \equiv \{w_1 w_2 \ldots w_k \mid k > 0, \text{ donde cada } w_i \in L\}$

Cálculo de clausura y clausura positiva

Dados los lenguajes $A = \{a\}$ y $B = \{b\}$, vamos a describir por comprensión $(A \circ B)^*$ y $(A \circ B)^+$.

$\rightrightarrows$ Primero obtenemos $A \circ B = \{ab\}$.

$\rightrightarrows$ **Clausura:** $(A \circ B)^* = \{ab\}^* = \{(ab)^n \mid n \geq 0\}$.

$\rightrightarrows$ **Clausura positiva:** $(A \circ B)^+ = \{ab\}^+ =$

Clausura y clausura positiva de Kleene

- Se define la **clausura de Kleene** (o **cierre**) de un lenguaje L: $L^* = \bigcup_{n=0}^{\infty} L^n \equiv \{w_1 w_2 \dots w_k \mid k \geq 0, \text{ donde cada } w_i \in L\}$
- La **clausura positiva** (o **cierre positivo**) de L contiene todas las cadenas que se obtienen al concatenar una o más cadenas de L: $\boxed{L^+ = \bigcup_{n=1}^{\infty} L^n} \equiv \{w_1 w_2 \dots w_k \mid k > 0, \text{ donde cada } w_i \in L\}$

Cálculo de clausura y clausura positiva

Dados los lenguajes $A = \{a\}$ y $B = \{b\}$, vamos a describir por comprensión $(A \circ B)^*$ y $(A \circ B)^+$.

$\Rightarrow$ Primero obtenemos $A \circ B = \{ab\}$.

$\Rightarrow$ **Clausura:** $(A \circ B)^* = \{ab\}^* = \{(ab)^n \mid n \geq 0\}$.

$\Rightarrow$ **Clausura positiva:** $(A \circ B)^+ = \{ab\}^+ = \{(ab)^n \mid n > 0\}$.

Preguntas de evaluación en apuntes

- Contiene problemas de **razonamiento**, **aplicación o cálculo** y **preguntas tipo test**.
- Son una colección de preguntas de examen, aunque no incluye todo tipo de preguntas posibles de examen.

- **Problemas resueltos:** actividad para auto-evaluación.

- **Problemas propuestos:** actividad para resolver en parte en clase (teoría o prácticas).

- **Preguntas tipo test:** actividad para resolver en parte en clase (teoría o prácticas).