UNIVERSIDAD MARIANO GALVEZ DE GUATEMALA

CENTRO UNIVERSITARIO CAMPUS COBÁN

Estudiante

Mauro Jom Lem 0902-23-4740

CARRERA:

Ingeniería en Sistemas de Información y Ciencias de la Computación

CURSO:

Autómatas

CATEDRÁTICO:

HECTOR ARMANDO MACZ BAC

AUTÓMATAS DE PILA PDA

Ejercicio 2

$$L = \{x^{2n} y^{m} x^{3m} y^{n} z^{+} \mid n \geq 1, m \geq 1\}$$

Introducción

El presente informe describe el desarrollo de un autómata de pila (PDA) creado para reconocer el lenguaje formal

$$L = \{x^{2n}y^{m}x^{3m}y^{n}z^{+} \mid n \geq 1,\ m \geq 1\}.$$

El proyecto forma parte del curso Autómatas y Lenguajes Formales de la Universidad Mariano Gálvez de Guatemala y tiene como finalidad aplicar los principios teóricos de los autómatas en una simulación práctica.

El sistema fue implementado con HTML, CSS y JavaScript, permitiendo ejecutar el simulador directamente desde un navegador. La aplicación muestra de forma visual cómo el autómata procesa las cadenas, resalta los estados activos y representa el contenido de la pila en cada paso.

En este documento se presentan la descripción formal del autómata, su gramática equivalente, la tabla de transiciones, ejemplos de evaluación de cadenas y una explicación general de su funcionamiento. Con ello, se busca demostrar la relación entre los conceptos teóricos y su aplicación práctica en la simulación de máquinas abstractas.

Lenguaje del Autómata

Lenguaje definido:

$$L = \{ x^{2n}\, y^{m}\, x^{3m}\, y^{n}\, z^{+} \mid n \geq 1,\, m \geq 1 \}$$

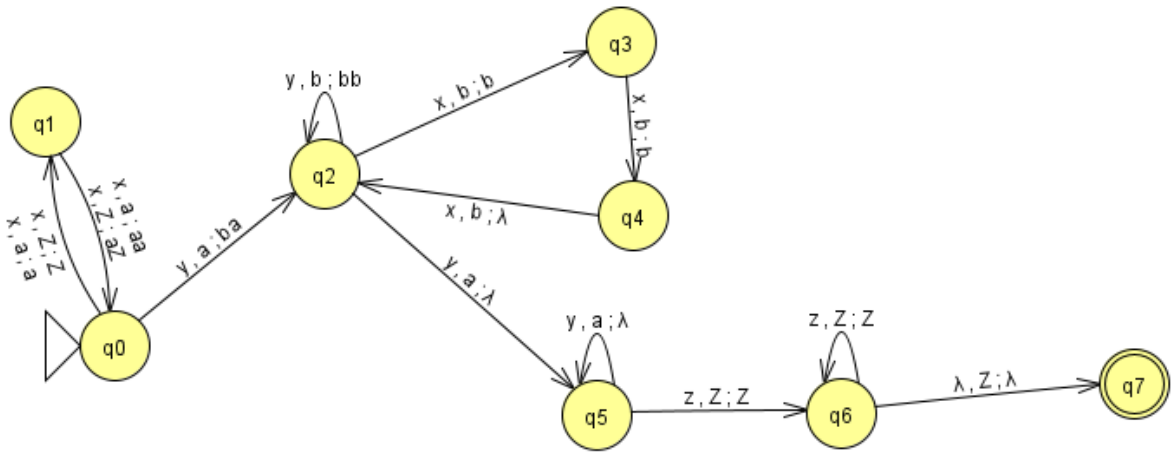
Descripción general:

El lenguaje genera cadenas que inician con una cantidad par de x, seguidas por una cantidad m de y, luego tres veces esa cantidad de x, posteriormente una cantidad de y igual a n = x_1 / 2, y termina con al menos una z.

El autómata verifica estas proporciones utilizando una pila para apilar y desapilar símbolos que representan los conteos de cada bloque.

Diseño inicial del autómata

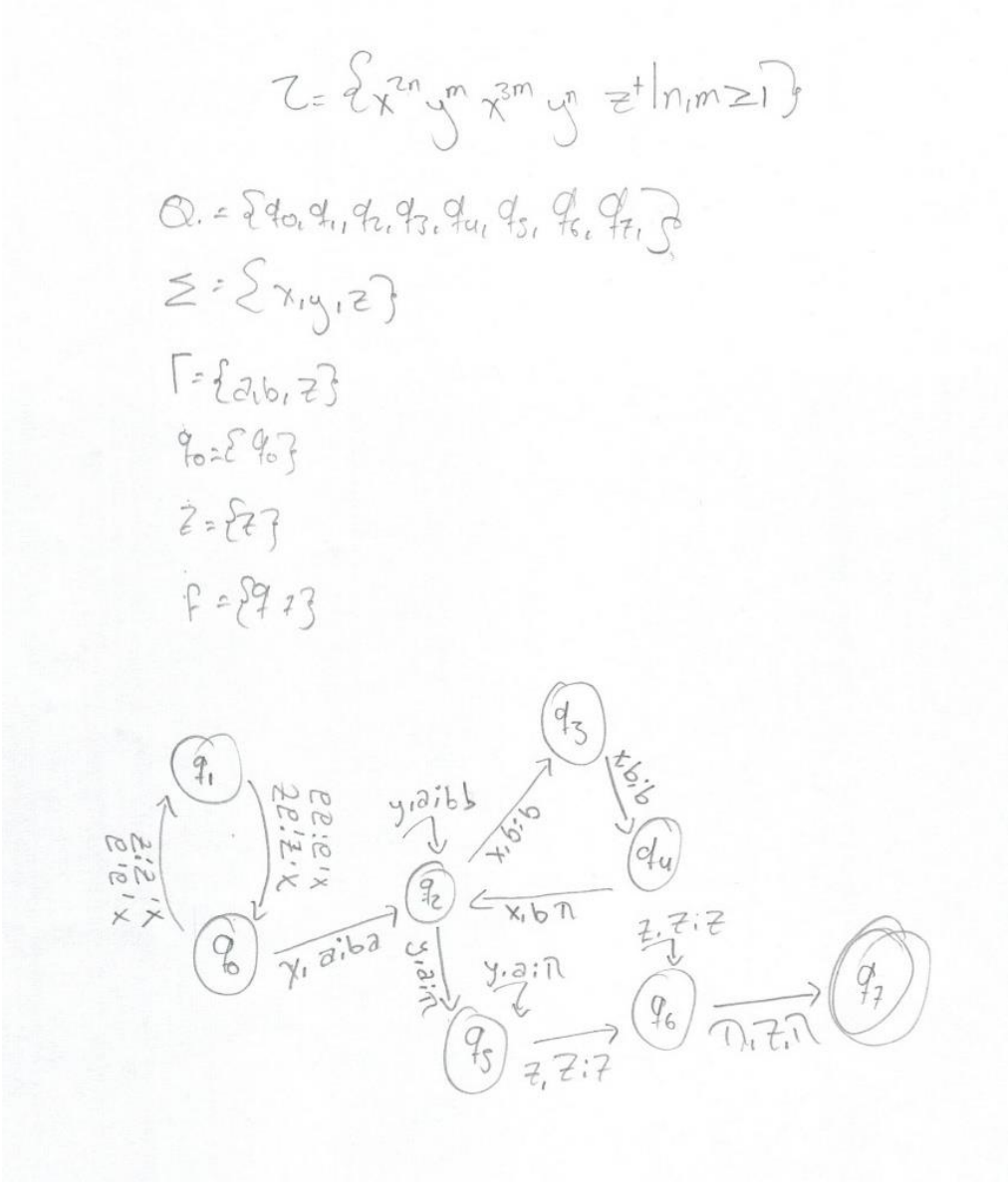
El diseño inicial del autómata de pila (PDA) se realizó utilizando la herramienta académica JFLAP, la cual permitió construir y comprobar visualmente la estructura del autómata, sus estados y transiciones de pila.
En esta etapa se representó la séptupla formal, la función de transición δ, y se verificó el comportamiento del autómata con diferentes cadenas de prueba para asegurar que reconociera correctamente el lenguaje definido.



Desarrollo manual del autómata

Como parte del proceso de análisis y construcción del lenguaje asignado, se elaboró el autómata de pila (PDA) de forma manual, representando cada uno de sus componentes: el grafo, la tabla de transiciones, la séptupla formal, la gramática y las derivaciones correspondientes.
Este trabajo manual permitió comprender la relación entre los símbolos del lenguaje, las reglas de producción y las acciones de la pila.

Se adjuntan las imágenes escaneadas de las hojas elaboradas a mano, que muestran cada etapa del diseño del autómata y sus respectivas comprobaciones. Dichas evidencias reflejan el desarrollo teórico previo a la implementación digital realizada en el simulador web.



$S=$ $(q_0, xz) = (q_1, Z)$
$(q_1, xz) = (q_0, az)$
$(q_0, x, a) = (q_1, a)$
$(q_1, x, a) = (q_0, aa)$

$(q_1, y, a) = (q_2, ba)$
$(q_2, y, b) = (q_2, bb)$
$(q_3, x, b) = (q_3, b)$
$(q_3, x, b) = (q_4, b)$
$(q_4, x, b) = (q_2, \Pi)$
$(q_2, y, a) = (q_5, \Pi)$
$(q_5, y, a) = (q_5, \Pi)$
$(q_5, z, z) = (q_6, z)$
$(q_6, z, z) = (q_6, z)$
$(q_6, \Pi, z) = q_7, \Pi)$

Tabla de Transición

Estado	Entrada	Desapilar	A
→ q_0	x	$z \to zz$	q_1
q_1	x	$z \to az$	q_0
q_0	y	$z \to ba$	q_2
q_2	y	$z \to bb$	q_2
q_2	x	$z \to b$	q_3
q_3	x	$z \to b$	q_4
q_4	x	$z \to b\Pi$	q_2
q_2	y	$z \to a\Pi$	q_5
q_5	y	$z \to a\Pi$	q_5
q_5	z	$z \to zz$	q_6
q_6	z	$z \to zz$	q_6
q_6	Π	$z \to \Pi$	q_7*

Gramatica

$G = (V, \Sigma, \Pi, S)$

- $V = \{S, A, B, C\}$
- $\Sigma = \{x, y, z\}$

$\Pi =$

$S \rightarrow xxAyC$
$A \rightarrow xxAy \mid B$
$B \rightarrow yBxxx \mid yxxx$
$C \rightarrow zC \mid z$

Derivación a la izquierda.

Cadena: xxyxxxyz

S
$\rightarrow xxAyC$
$\rightarrow xxByC$
$\rightarrow xxyxxxyC$
$\rightarrow xxyxxxyz$

$\Rightarrow xxyxxxyz$

Cadena: xxxyyxxxxxyyzz

S
→ xxAyC
xxxxAyyC
xxxxByyC
xxxxyBxxxyyC
xxxxyyxxxxxyyC
xxxxyyxxxxxxyyzC
xxxxyyxxxxxyyzz

ID Descripción instantánea.

Cadena: xxyxxxyz

(q0, xxyxxxyz, Sz) ⊢ (q1, xxyxxxyz, xxAyCz) ⊢
(q1, xyxxxyz, xAyCz) ⊢ (q1, yxxxyz, AyCz) ⊢
(q1, yxxxyz, ByCz) ⊢ (q1, yxxxyz, yxxxyCz) ⊢
(q1, xxxyz, xxxyCz) ⊢ (q1, xxyz, xxyCz) ⊢
(q1, xyz, xyCz) ⊢ (q1, yz, yCz) ⊢
(q1, z, Cz) ⊢ (q1, z, z) ⊢ (q1, ε, ε)

Cadena: xxxx yy xxxxxxyy zz

$(q_0, xxxx\ yy\ xxxxxx\ yy\ zz, SZ) \vdash$

$(q_1, xxxx\ yy\ xxxxxx\ yy\ zz, xxAy(z) \vdash$

$(q_1, xxxyy\ xxxxxxyy\ zz, xAy(z) \vdash$

$(q_1, xxyy\ xxxxx\ yy\ zz, xAy(z) \vdash$

$(q_1, xxyy\ xxxxx\ yy\ zz, xxAyy(z) \vdash$

$(q_1, xyy\ xxxxxx\ yy\ zz, xAyy(z) \vdash$

$(q_1, yy\ xxxxxx\ yy\ zz, Ayy(z) \vdash$

$(q_1, yy\ xxxxxx\ yy\ zz, Byy(z) \vdash$

$(q_1, yy\ xxxxxx\ yy\ zz, yBxxyy(z) \vdash$

$(q_1, yxx\ xxxx\ yy\ zz, Bxxyy(z) \vdash$

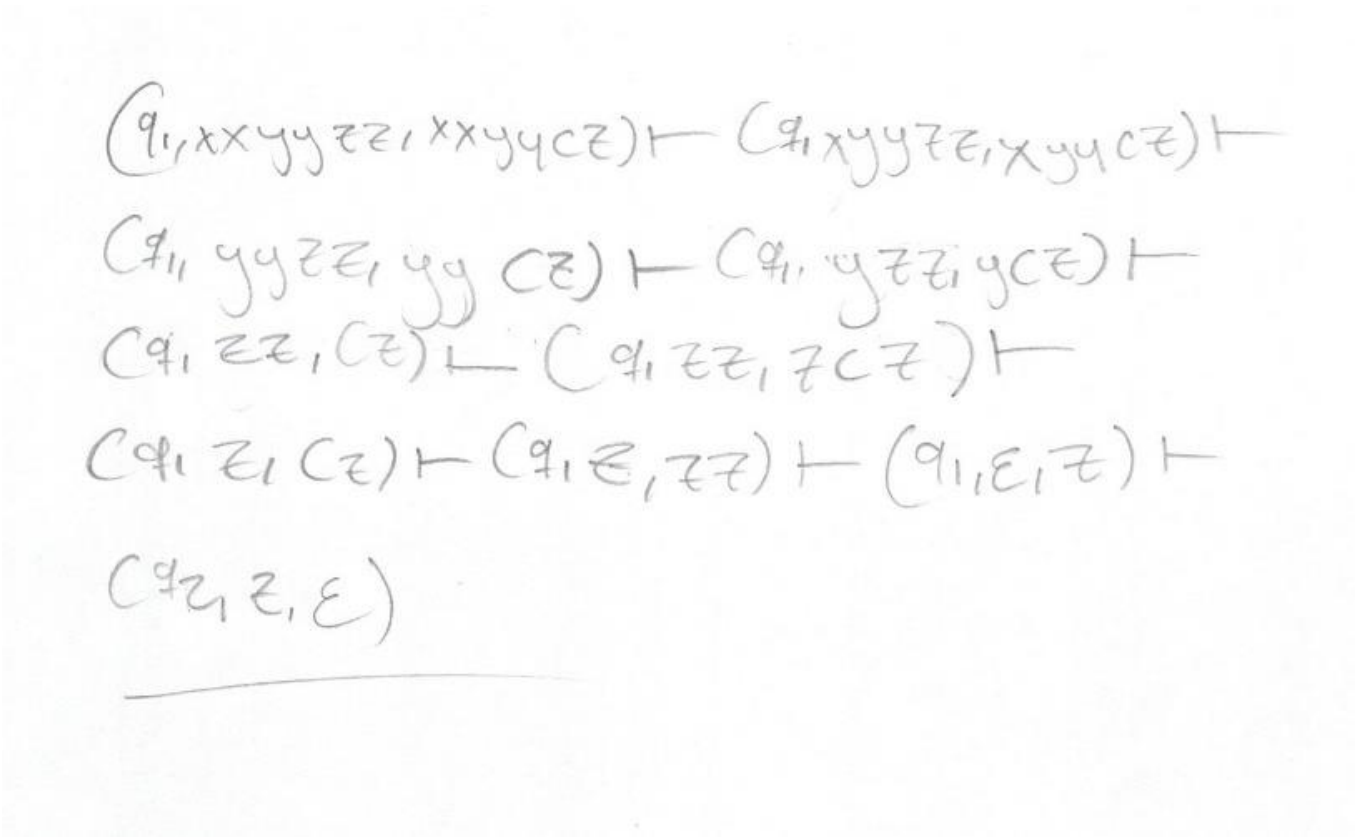
$(q_1, yxxxxx\ yy\ zz, yxxxxx\ yy(z) \vdash$

$(q_1, xxxxx\ yy\ zz, xxxxx\ yy(z) \vdash$

$(q_1, xxxxyy\ zz, xxxx\ yy(z) \vdash$

$(q_1, xxxx\ yy\ zz, xxx\ yy(z) \vdash$

$(q_1, xxx\ yy\ zz, xxx\ yy(z) \vdash$



Simulación en entorno web

Una vez definido y comprobado el autómata de pila en su versión teórica, se procedió a la implementación digital del modelo mediante un entorno web interactivo. La interfaz fue desarrollada con HTML, CSS y JavaScript, lo que permite ejecutar la simulación directamente desde el navegador sin requerir instalación adicional.

En la siguiente figura se presenta la interfaz principal del simulador, donde se observan claramente las distintas secciones funcionales del sistema:

En la parte izquierda se muestra el grafo del autómata, con sus estados (q_0–q_7) y transiciones etiquetadas con los valores de entrada, el símbolo de pila leído y la acción de apilado o desapilado.

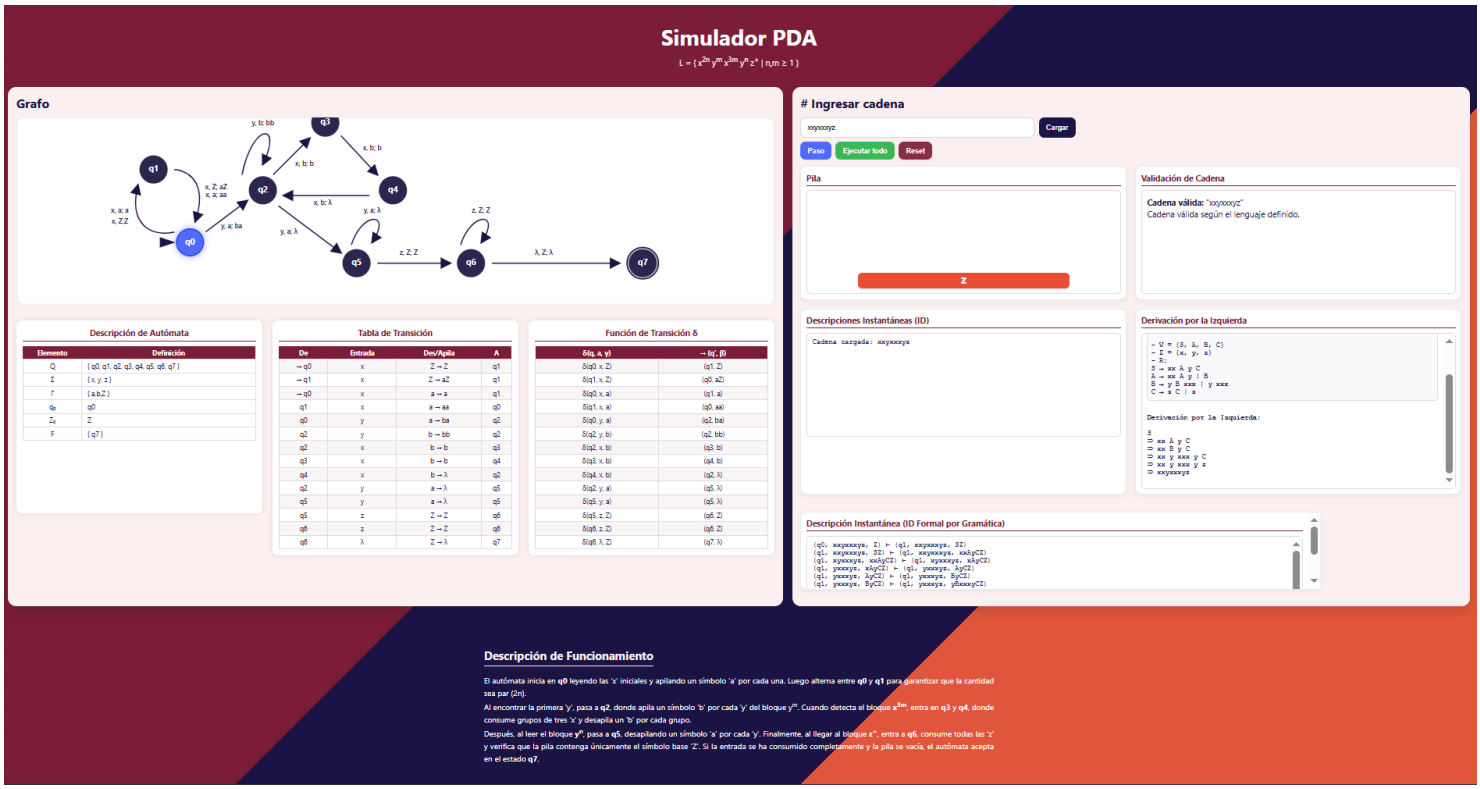
Debajo del grafo se incluyen tres cuadros descriptivos que muestran la séptupla formal del autómata, la tabla de transición y la función de transición δ, facilitando la interpretación del modelo matemático implementado.

En la parte derecha se encuentra el panel de control, donde el usuario puede ingresar una cadena, cargarla, y luego ejecutar la simulación paso a paso o completa mediante los botones “Paso” y “Ejecutar todo”.

La sección inferior del panel derecho visualiza el contenido dinámico de la pila, las descripciones instantáneas (ID) y la derivación por la izquierda, que permiten seguir el proceso de análisis en tiempo real.

En la parte superior derecha se muestra además un cuadro de validación de cadena, que indica si la entrada es aceptada o rechazada de acuerdo con las reglas definidas en el lenguaje formal.

La interfaz se diseñó con colores diferenciados, resaltando en azul el estado activo del autómata, mientras que la pila cambia conforme se realizan operaciones de apilado o desapilado.



Código fuente del proyecto

Como parte del desarrollo del simulador web, se elaboró e implementó el código fuente correspondiente al autómata de pila (PDA).
El sistema fue programado utilizando los lenguajes HTML, CSS y JavaScript, organizados en tres archivos principales:

index.html: estructura del simulador, interfaz de usuario y grafo del autómata.

style.css: estilos visuales, distribución de los paneles y diseño del entorno gráfico.

pda.js: lógica funcional del autómata, definición de transiciones, control de pila y ejecución paso a paso.

El código completo se adjunta al presente informe como evidencia del trabajo técnico realizado.
Dicho código permite reproducir íntegramente el funcionamiento del simulador y verificar la correcta implementación del autómata descrito.

JS

```
console.clear();
console.log("Simulador PDA cargado correctamente");

// PDA para el lenguaje: L = { x^(2n) y^m x^(3m) y^n z^+ | n, m ≥ 1 }

const svgStates = ["q0", "q1", "q2", "q3", "q4", "q5", "q6", "q7"];
let machine = null;
let runTimer = null;

//  Elementos del DOM
const $log = document.getElementById("log");
const $stack = document.getElementById("stack");
const $cad = document.getElementById("cadena");
const $alerta = document.getElementById("alerta");
const $derivacionLista = document.getElementById("derivacion-lista");
const $idFormal = document.getElementById("id-formal");

// Botones
document.getElementById("btn-load").onclick = loadInput;
document.getElementById("btn-step").onclick = stepOnce;
document.getElementById("btn-run").onclick = runAll;
document.getElementById("btn-reset").onclick = resetAll;
```

```
// Estado inicial
highlight("q0");
renderStack(["Z"]);
appendLog("Simulador listo. Escribe una cadena y presiona 'Cargar'.");

// ---------- MÁQUINA ----------
function initialMachine(input) {
  const entrada = input.trim().split("");
  return { entrada, i: 0, pila: ["Z"], estado: "q0", halted: false, accepted: false };
}

// ---------- VALIDACIÓN ----------
function validateLanguage(str) {
  const match = str.match(/^(x+)(y+)(x+)(y+)(z+)$/);
  if (!match) {
    return {
      valid: false,
      reason: "No sigue el patrón x^n y^m x^k y^l z^+ (todas deben estar presentes y en orden)."
    };
  }

  const [, x1, y1, x2, y2, z] = match;
  const n = x1.length / 2;
  const m = y1.length;

  // Validaciones estructurales
  if (x1.length < 2)
    return { valid: false, reason: "Debe haber al menos dos 'x' iniciales (x^2n)." };

  if (x1.length % 2 !== 0)
    return { valid: false, reason: "Las primeras 'x' deben ser pares (2n)." };

  if (y1.length < 1)
    return { valid: false, reason: "Debe haber al menos una 'y' (y^m)." };

  if (x2.length % 3 !== 0)
    return { valid: false, reason: "Las 'x' intermedias deben ser múltiplos de tres (x^3m)." };

  if (x2.length !== 3 * m)
    return {
      valid: false,
      reason: `El bloque x^{3m} tiene longitud ${x2.length}, pero debería ser ${3 * m} (3 × m).`
    };

  if (y2.length !== n)
    return {
      valid: false,
      reason: `Las 'y' finales (${y2.length}) deben ser n = x_1/2 = ${n}.`
    };

  if (z.length < 1)
    return { valid: false, reason: "Debe haber al menos una 'z' (z^+)." };
```

```
  return {
    valid: true,
    reason: "Cadena válida según el lenguaje definido."
  };
}

// ---------- CARGA ----------
function loadInput() {
  const s = ($cad.value || "").trim();
  if (!s) return alert("Ingresa una cadena, por favor.");

  const check = validateLanguage(s);
  if (!check.valid) {
    $alerta.innerHTML = `<b>Cadena inválida:</b> "${s}"<br>${check.reason}`;
    $derivacionLista.innerHTML = "<em>Cadena no válida. Derivación no generada.</em>";
    $idFormal.textContent = "Cadena no válida. No se genera ID formal.";
  } else {
    $alerta.innerHTML = `<b>Cadena válida:</b> "${s}"<br>${check.reason}`;
    const pasos = derivarCadena(s);
    $derivacionLista.textContent = pasos.join("\n⇒ ");
    const formal = generarIDFormalSimple(s);
    $idFormal.textContent = formal.join("\n");
  }

  machine = initialMachine(s);
  svgStates.forEach(id => document.getElementById(id)?.classList.remove("active"));
  highlight(machine.estado);
  renderStack(machine.pila);
  $log.textContent = "";
  appendLog(`Cadena cargada: ${s}`);
}

// ---------- CONTROLES ----------
function resetAll() {
  stopRun();
  machine = null;
  $cad.value = "";
  $log.textContent = "";
  renderStack(["Z"]);
  highlight("q0");
  appendLog("Simulador reiniciado.");
  $alerta.innerHTML = "<em>Esperando cadena...</em>";
  $derivacionLista.innerHTML = "<em>Esperando cadena...</em>";
  $idFormal.textContent = "Esperando entrada formal...";
}

function ensureLoaded() {
  if (!machine) {
    alert("Primero presiona 'Cargar' con una cadena.");
    return false;
  }
  if (machine.halted) {
    alert("La máquina ya finalizó. Presiona 'Reset' para reiniciar.");
```

```
      return false;
    }
    return true;
}

function runAll() {
    if (!ensureLoaded()) return;
    stopRun();
    runTimer = setInterval(() => {
      if (!stepCore()) stopRun();
    }, 600);
}

function stopRun() {
    if (runTimer) {
      clearInterval(runTimer);
      runTimer = null;
    }
}

function stepOnce() {
    stopRun();
    if (!ensureLoaded()) return;
    stepCore();
}

// ---------------------- PDA CORE ----------------------
function stepCore() {
    const M = machine;
    const { entrada, i, pila, estado } = M;
    const simbolo = entrada[i] ?? null;
    const cima = pila[pila.length - 1] ?? null;

    if (estado === "q0") {
      if (simbolo === "x" && (cima === "Z" || cima === "a")) return goTo("q1", 1);
      if (simbolo === "y" && cima === "a") return goTo("q2", 1, () => pila.push("b"));
      return reject("Error en q0: se esperaba x o y con cima válida.");
    }

    if (estado === "q1") {
      if (simbolo === "x" && (cima === "Z" || cima === "a"))
        return goTo("q0", 1, () => pila.push("a"));
      return reject("Error en q1: se esperaba x.");
    }

    if (estado === "q2") {
      if (simbolo === "y" && cima === "b") return stayHere(1, () => pila.push("b"));
      if (simbolo === "x" && cima === "b") return goTo("q3", 1);
      if (simbolo === "y" && cima === "a") return goTo("q5", 1, () => pila.pop());
      return reject("Error en q2.");
    }

    if (estado === "q3") {
```

```
    if (simbolo === "x" && cima === "b") return goTo("q4", 1);
    return reject("Error en q3.");
  }

  if (estado === "q4") {
    if (simbolo === "x" && cima === "b") return goTo("q2", 1, () => pila.pop());
    return reject("Error en q4.");
  }

  if (estado === "q5") {
    if (simbolo === "y" && cima === "a") return stayHere(1, () => pila.pop());
    if (simbolo === "z" && cima === "Z") return goTo("q6", 1);
    if (simbolo === null) return reject("Falta z final.");
    return reject("Error en q5.");
  }

  if (estado === "q6") {
    if (simbolo === "z" && cima === "Z") return stayHere(1);
    if (simbolo === null && cima === "Z") return goTo("q7", 0, () => pila.pop());
    return reject("Error en q6.");
  }

  if (estado === "q7") {
    M.halted = true;
    M.accepted = true;
    appendLog("Cadena aceptada.");
    return false;
  }

  return reject("Estado desconocido.");
}

// ---------- FUNCIONES DE TRANSICIÓN ----------

function goTo(nuevo, consumed = 1, action = null) {
  const prev = machine.estado;
  const pilaAntes = [...machine.pila];
  if (action) action();
  const pilaDespues = [...machine.pila];

  recordID(prev, nuevo, consumed, pilaAntes, pilaDespues);
  machine.estado = nuevo;
  machine.i += consumed;

   animateTransition(prev, nuevo);

  moveTo(nuevo);
  renderStack(machine.pila);
  return true;
}

// Transición con bucle (mismo estado)
function stayHere(consumed = 1, action = null) {
```

```
  const estadoActual = machine.estado;
  const pilaAntes = [...machine.pila];
  if (typeof action === "function") action();
  const pilaDespues = [...machine.pila];

  recordID(estadoActual, estadoActual, consumed, pilaAntes, pilaDespues);
  machine.i += consumed;

  //activa la animación CSS de bucle
  animateTransition(estadoActual, estadoActual);

  renderStack(machine.pila);
  moveTo(estadoActual);
  return true;
}

// Registro de ID formal
function recordID(prev, next, consumed, pilaAntes, pilaDespues) {
  const entradaAntes = machine.entrada.slice(machine.i);
  const entradaDespues = machine.entrada.slice(machine.i + consumed);
  const fmt = (x) => (Array.isArray(x) ? x.join("") : x) || "ε";
  const restA = fmt(entradaAntes);
  const restB = fmt(entradaDespues);
  const pilaA = fmt(pilaAntes);
  const pilaB = fmt(pilaDespues);
  $log.textContent += `(${prev}, ${restA}, ${pilaA}) ⊢ (${next}, ${restB}, ${pilaB})\n`;
}

// Rechazo
function reject(msg) {
  machine.halted = true;
  machine.accepted = false;
  appendLog(" " + msg);
  return false;
}

// ---------- CONTROL DE CLASES PARA ANIMACIÓN ----------

// Esta función solo aplica o quita clases CSS
function animateTransition(prev, next) {
  const prevNode = document.getElementById(prev);
  const nextNode = document.getElementById(next);
  if (!nextNode) return;

  // Si es bucle (mismo estado)
  if (prev === next) {
    nextNode.classList.add("looping");
    setTimeout(() => nextNode.classList.remove("looping"), 600);
    return;
  }

  // Si es transición a otro estado
  nextNode.classList.add("transitioning");
```

```
  setTimeout(() => nextNode.classList.remove("transitioning"), 600);
}

// ---------- UTILIDADES ----------
function appendLog(l) { $log.textContent += l + "\n"; }
function renderStack(p) {
  $stack.innerHTML = "";
  p.forEach((s, i) => {
    const div = document.createElement("div");
    div.className = "cell" + (i === p.length - 1 ? " top" : "");
    div.textContent = s;
    $stack.appendChild(div);
  });
}
function moveTo(id) {
  svgStates.forEach((s) => {
    const el = document.getElementById(s);
    if (el) el.classList.remove("active");
  });
  highlight(id);
}
function highlight(id) {
  const el = document.getElementById(id);
  if (el) el.classList.add("active");
}

// ---------- GRAMÁTICA ----------
const grammarRules = {
  S: ["xx A y C"],
  A: ["xx A y", "B"],
  B: ["y B xxx", "y xxx"],
  C: ["z C", "z"],
};

function derivarCadena(cadena) {
  const pasos = [];
  let actual = "S";
  pasos.push(actual);

  const bloques = cadena.match(/^(x+)(y+)(x+)(y+)(z+)$/);
  if (!bloques)
    return ["La cadena no cumple el patrón x^(2n) y^m x^(3m) y^n z^+."];

  const [_, X1, Y1, X2, Y2, Z] = bloques;
  const n = X1.length / 2;
  const m = Y1.length;
  const x3m = X2.length;
  const y2 = Y2.length;
  const zCount = Z.length;

  if (!Number.isInteger(n) || n < 1)
    return ["Error: la cantidad de x inicial no es múltiplo de 2."];
  if (x3m !== 3 * m)
```

```
    return [` Error: x^(3m) debería ser ${3 * m}, pero es ${x3m}.`];
  if (y2 !== n)
    return [` Error: y finales (${y2}) deben ser iguales a n=${n}.`];

  actual = actual.replace("S", grammarRules.S[0]);
  pasos.push(actual);

  for (let i = 1; i < n; i++) {
    actual = actual.replace("A", grammarRules.A[0]);
    pasos.push(actual);
  }

  actual = actual.replace("A", grammarRules.A[1]);
  pasos.push(actual);

  for (let i = 0; i < m - 1; i++) {
    actual = actual.replace("B", grammarRules.B[0]);
    pasos.push(actual);
  }

  actual = actual.replace("B", grammarRules.B[1]);
  pasos.push(actual);

  for (let i = 0; i < zCount - 1; i++) {
    actual = actual.replace("C", grammarRules.C[0]);
    pasos.push(actual);
  }

  actual = actual.replace("C", grammarRules.C[1]);
  pasos.push(actual);

  pasos.push(cadena);
  return pasos;
}

function generarIDFormalSimple(cadena) {
  const pasos = [];
  const fmt = (x) => (x && x.length ? x : "ε");

  let entrada = cadena.split("");
  let pila = ["Z"];
  let estado = "q0";

  // --- función auxiliar para registrar cada paso ---
  function emitir(prev, next, pilaAntes, pilaDespues, entradaRestante) {
    const mostrarPila = (p) => fmt([...p].reverse().join(""));
    pasos.push(` (${prev}, ${fmt(entradaRestante.join(""))}, ${mostrarPila(pilaAntes)}) ⊢ (${next},
${fmt(entradaRestante.join(""))}, ${mostrarPila(pilaDespues)})`);
  }

  // --- Paso inicial ---
  const pilaAntes = [...pila];
  pila.push("S");
```

```
emitir("q0", "q1", pilaAntes, pila, entrada);
estado = "q1";

// --- Ciclo principal ---
while (pila.length > 0) {
  const cima = pila[pila.length - 1];

  // --- S ---
  if (cima === "S") {
    const pilaAntes2 = [...pila];
    pila.pop();
    // S → xx A y C
    pila.push("C");
    pila.push("y");
    pila.push("A");
    pila.push("x");
    pila.push("x");
    emitir(estado, estado, pilaAntes2, [...pila], entrada);
    continue;
  }

  // --- A ---
  if (cima === "A") {
    const pilaAntes2 = [...pila];
    pila.pop();

    if (entrada[0] === "x") {
      // A → xx A y
      pila.push("y");
      pila.push("A");
      pila.push("x");
      pila.push("x");
    } else {
      // A → B
      pila.push("B");
    }

    emitir(estado, estado, pilaAntes2, [...pila], entrada);
    continue;
  }
  // --- B ---
  if (cima === "B") {
    const pilaAntes2 = [...pila];
    pila.pop();

    if (entrada[0] === "y") {
      // B → y B xxx
      pila.push("x");
      pila.push("x");
      pila.push("x");
      pila.push("B");
      pila.push("y");
    } else {
```

```
            // No más 'y', eliminar B sin expandir

          }

          emitir(estado, estado, pilaAntes2, [...pila], entrada);
          continue;
        }
        // --- C ---
        if (cima === "C") {
          const pilaAntes2 = [...pila];
          pila.pop();

          // C → z C | z
          if (entrada.filter(ch => ch === "z").length > 1) {
            pila.push("C");
            pila.push("z");
          } else {
            pila.push("z");
          }

          emitir(estado, estado, pilaAntes2, [...pila], entrada);
          continue;
        }

        // --- Consumo de terminales ---
        if (["x", "y", "z"].includes(cima)) {
          if (entrada[0] === cima) {
            const pilaAntes2 = [...pila];
            pila.pop();
            entrada.shift();
            emitir(estado, estado, pilaAntes2, [...pila], entrada);
            continue;
          } else {
            pasos.push(` Error: esperaba '${cima}', pero se encontró '${entrada[0] || "ε"}'.`);
            break;
          }
        }

        // --- Finalización ---
        if (cima === "Z" && entrada.length === 0) {
          const pilaAntes2 = [...pila];
          pila.pop();
          emitir(estado, "q2", pilaAntes2, [], entrada);
          pasos.push("(q2, ε, ε) Cadena aceptada");
          break;
        }

        break;
      }

      return pasos;
    }
```

HTML

```
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8" />
  <title>Simulador PDA – Ejercicio 2</title>
  <link rel="stylesheet" href="style.css" />
</head>

<body>
  <!-- ENCABEZADO -->
  <header class="hero">
    <h1>Simulador PDA</h1>
    <p>L = { x<sup>2n</sup> y<sup>m</sup> x<sup>3m</sup> y<sup>n</sup> z<sup>+</sup> | n,m ≥
1 }</p>
  </header>

  <main class="layout">
    <!-- PANEL IZQUIERDO: Grafo y tablas -->
    <section class="panel graph">
      <h2>Grafo</h2>
      <div class="graph-container">

        <!-- SVG del Grafo -->
        <div class="graph-svg">
          <svg id="pda-svg" viewBox="0 0 1250 360" aria-label="Grafo PDA">
            <defs>
              <marker id="arrow" viewBox="0 0 12 12" refX="10" refY="6" markerWidth="10"
markerHeight="10"
                orient="auto">
                <path d="M 0 0 L 12 6 L 0 12 z" fill="#1A1446" />
              </marker>
            </defs>

            <!-- q1 a q0 -->
            <path d="M200,100 C210,100 270,111 240,200" class="edge" marker-end="url(#arrow)" />
            <text x="280" y="150" class="label">x, a; aa</text>
            <text x="280" y="135" class="label">x, Z; aZ</text>

            <!-- q0 a q1 -->
            <path d="M200,220 C160,230 110,210 130,130" class="edge" marker-end="url(#arrow)" />
            <text x="95" y="200" class="label">x, Z;Z</text>
            <text x="95" y="180" class="label">x, a; a</text>
            <!-- q0 a q2 -->
            <path d="M260,220 L340,160" class="edge" marker-end="url(#arrow)" />
            <text x="310" y="210" class="label">y, a; ba</text>
```

```
<!-- BUCLE q2 -->
<path d="M330,110 C350,9 410,8 370,90" class="edge" marker-end="url(#arrow)" />
<text x="370" y="11" class="label">y, b; bb</text>

<!-- q2 → q3 -->
<path d="M520,35 L590,110" class="edge" marker-end="url(#arrow)" />
<text x="450" y="90" class="label">x, b; b</text>

<!-- q3 → q4 -->
<path d="M390,110 L460,40" class="edge" marker-end="url(#arrow)" />
<text x="580" y="60" class="label">x, b; b</text>

<!-- q4 → q2 -->
<path d="M575,150 L410,150" class="edge" marker-end="url(#arrow)" />
<text x="485" y="165" class="label">x, b; λ</text>

<!-- q2 → q5 -->
<path d="M400,170 L520,260" class="edge" marker-end="url(#arrow)" />
<text x="420" y="220" class="label">y, a; λ</text>
<!-- BUCLE Q5 -->
<path d="M540,245 C560,180 620,180 580,240" class="edge" marker-end="url(#arrow)" />
<text x="580" y="180" class="label">y, a; λ</text>

<!-- q5 → q6 -->
<path d="M590,280 L730,280" class="edge" marker-end="url(#arrow)" />
<text x="650" y="260" class="label">z, Z; Z</text>

<!-- BUCLE q6 -->
<path d="M750,245 C770,180 830,180 790,240" class="edge" marker-end="url(#arrow)" />
<text x="789" y="180" class="label">z, Z; Z</text>

<!-- q6 → q7 -->
<path d="M810,280 L1055,280" class="edge" marker-end="url(#arrow)" />
<text x="910" y="260" class="label">λ, Z; λ</text>

<!-- ESTADOS -->
<g id="q0" class="node" transform="translate(230,240)">
  <circle r="26" /><text class="nodelabel">q0</text>
  <path d="M -58 -10 L -28 0 L -58 10 Z" class="start" />
</g>

<g id="q1" class="node" transform="translate(160,100)">
  <circle r="26" /><text class="nodelabel">q1</text>
</g>

<g id="q2" class="node" transform="translate(370,140)">
  <circle r="26" /><text class="nodelabel">q2</text>
</g>

<g id="q3" class="node" transform="translate(490,10)">
  <circle r="26" /><text class="nodelabel">q3</text>
</g>
```

```
      <g id="q4" class="node" transform="translate(620,140)">
        <circle r="26" /><text class="nodelabel">q4</text>
      </g>

      <g id="q5" class="node" transform="translate(550,280)">
        <circle r="26" /><text class="nodelabel">q5</text>
      </g>

      <g id="q6" class="node" transform="translate(770,280)">
        <circle r="26" /><text class="nodelabel">q6</text>
      </g>

      <g id="q7" class="node accept" transform="translate(1100,280)">
        <circle r="26" />
        <circle r="30" class="double" /><text class="nodelabel">q7</text>
      </g>
    </svg>
  </div>

  <!-- CUADROS INFORMATIVOS -->
  <div class="graph-details">
    <!-- Descripción -->
    <div class="info-box">
      <h3>Descripción de Autómata</h3>
      <table class="elements-table">
        <tr>
          <th>Elemento</th>
          <th>Definición</th>
        </tr>
        <tr>
          <td>Q</td>
          <td>{ q0, q1, q2, q3, q4, q5, q6, q7 }</td>
        </tr>
        <tr>
          <td>Σ</td>
          <td>{ x, y, z }</td>
        </tr>
        <tr>
          <td>Γ</td>
          <td>{ a,b,Z }</td>
        </tr>
        <tr>
          <td>q0</td>
          <td>q0</td>
        </tr>
        <tr>
          <td>Z0</td>
          <td>Z</td>
        </tr>
        <tr>
          <td>F</td>
          <td>{ q7 }</td>
        </tr>
```

```
  </table>
</div>

<div class="info-box">
  <h3>Tabla de Transición</h3>
  <table class="transition-table small">
    <tr>
      <th>De</th>
      <th>Entrada</th>
      <th>Des/Apila</th>
      <th>A</th>
    </tr>
    <tr>
      <td>→ q0</td>
      <td>x</td>
      <td>Z → Z</td>
      <td>q1</td>
    </tr>
    <tr>
      <td>→ q1</td>
      <td>x</td>
      <td>Z → aZ</td>
      <td>q1</td>
    </tr>
    <tr>
      <td>→ q0</td>
      <td>x</td>
      <td>a → a</td>
      <td>q1</td>
    </tr>
    <tr>
      <td>q1</td>
      <td>x</td>
      <td>a → aa</td>
      <td>q0</td>
    </tr>
    <tr>
      <td>q0</td>
      <td>y</td>
      <td>a → ba</td>
      <td>q2</td>
    </tr>
    <tr>
      <td>q2</td>
      <td>y</td>
      <td>b → bb</td>
      <td>q2</td>
    </tr>
    <tr>
      <td>q2</td>
      <td>x</td>
      <td>b → b</td>
      <td>q3</td>
```

```
      </tr>
      <tr>
        <td>q3</td>
        <td>x</td>
        <td>b → b</td>
        <td>q4</td>
      </tr>
      <tr>
        <td>q4</td>
        <td>x</td>
        <td>b → λ</td>
        <td>q2</td>
      </tr>

      <tr>
        <td>q2</td>
        <td>y</td>
        <td>a → λ</td>
        <td>q5</td>
      </tr>
      <tr>
        <td>q5</td>
        <td>y</td>
        <td>a → λ</td>
        <td>q5</td>
      </tr>
      <tr>
        <td>q5</td>
        <td>z</td>
        <td>Z → Z</td>
        <td>q6</td>
      </tr>
      <tr>
        <td>q6</td>
        <td>z</td>
        <td>Z → Z</td>
        <td>q6</td>
      </tr>
      <tr>
        <td>q6</td>
        <td>λ</td>
        <td>Z → λ</td>
        <td>q7</td>
      </tr>
    </table>
</div>

<!-- Función δ -->
<div class="info-box">
    <h3>Función de Transición δ</h3>
    <table class="transition-table formal">
      <tr>
        <th>δ(q, a, γ)</th>
```

```
  <th>→ (q', β)</th>
</tr>
<tr>
  <td>δ(q0, x, Z)</td>
  <td>(q1, Z)</td>
</tr>
<tr>
  <td>δ(q1, x, Z)</td>
  <td>(q0, aZ)</td>
</tr>
<tr>
  <td>δ(q0, x, a)</td>
  <td>(q1, a)</td>
</tr>
<tr>
  <td>δ(q1, x, a)</td>
  <td>(q0, aa)</td>
</tr>
<tr>
  <td>δ(q0, y, a)</td>
  <td>(q2, ba)</td>
</tr>
<tr>
  <td>δ(q2, y, b)</td>
  <td>(q2, bb)</td>
</tr>
<tr>
  <td>δ(q2, x, b)</td>
  <td>(q3, b)</td>
</tr>
<tr>
  <td>δ(q3, x, b)</td>
  <td>(q4, b)</td>
</tr>
<tr>
  <td>δ(q4, x, b)</td>
  <td>(q2, λ)</td>
</tr>
<tr>
  <td>δ(q2, y, a)</td>
  <td>(q5, λ)</td>
</tr>
<tr>
  <td>δ(q5, y, a)</td>
  <td>(q5, λ)</td>
</tr>
<tr>
  <td>δ(q5, z, Z)</td>
  <td>(q6, Z)</td>
</tr>
<tr>
  <td>δ(q6, z, Z)</td>
  <td>(q6, Z)</td>
```

```
            </tr>
            <tr>
              <td>δ(q6, λ, Z)</td>
              <td>(q7, λ)</td>
            </tr>
          </table>
        </div>

    </div>
  </div>
</section>

<!-- PANEL DERECHO -->
<section class="panel controls">
  <h2># Ingresar cadena</h2>
  <div class="row">
    <input id="cadena" placeholder="ej. xxyxxxyz" />
    <button id="btn-load">Cargar</button>
  </div>
  <div class="row">
    <button id="btn-step">Paso</button>
    <button id="btn-run">Ejecutar todo</button>
    <button id="btn-reset" class="ghost">Reset</button>
  </div>

  <!-- CUADROS DE PILA, VALIDACIÓN, ID Y DERIVACIÓN -->
  <div class="grid-cuadros">
    <div class="stack-box">
      <h3>Pila</h3>
      <div id="stack" class="stack"></div>
    </div>
    <div class="alert-box">
      <h3>Validación de Cadena</h3>
      <div id="alerta" class="alert-content"><em>Esperando cadena...</em></div>
    </div>
    <div class="id-box">
      <h3>Descripciones Instantáneas (ID)</h3>
      <pre id="log" class="log"></pre>
    </div>
    <div class="derivacion-box">
      <h3>Derivación por la Izquierda</h3>
      <div id="derivacion" class="derivacion-content">
        <h4>Gramática G:</h4>
        <pre class="gramatica">
    G = (V, Σ, R, S)

    - V = {S, A, B, C}
    - Σ = {x, y, z}
    - R:
    S → xx A y C
    A → xx A y | B
    B → y B xxx | y xxx
    C → z C | z
```

```
            </pre>
            <h4>Derivación por la Izquierda:</h4>
            <div id="derivacion-lista">
              <em>Esperando cadena...</em>
            </div>
          </div>

        </div>

        <div class="id-formal-box">
          <h3>Descripción Instantánea (ID Formal por Gramática)</h3>
          <pre id="id-formal" class="log"></pre>
        </div>
      </div>
    </section>
  </main>

  <!-- DESCRIPCIÓN FINAL -->
  <section class="descripcion-final">
    <h2>Descripción de Funcionamiento</h2>
    <p>El autómata inicia en <strong>q0</strong> leyendo las ‘x’ iniciales y apilando un símbolo ‘a’ por
cada una.
      Luego alterna entre <strong>q0</strong> y <strong>q1</strong> para garantizar que la cantidad
sea par (2n).
    </p>
    <p>Al encontrar la primera ‘y’, pasa a <strong>q2</strong>, donde apila un símbolo ‘b’ por cada ‘y’
del bloque
      y<sup>m</sup>.
      Cuando detecta el bloque <strong>x<sup>3m</sup></strong>, entra en <strong>q3</strong> y
      <strong>q4</strong>, donde consume grupos de tres ‘x’ y desapila un ‘b’ por cada grupo.
    </p>
    <p>Después, al leer el bloque <strong>y<sup>n</sup></strong>, pasa a <strong>q5</strong>,
desapilando un símbolo
      ‘a’ por cada ‘y’.
      Finalmente, al llegar al bloque <strong>z<sup>+</sup></strong>, entra a <strong>q6</strong>,
consume todas
      las ‘z’ y verifica que la pila contenga únicamente el símbolo base ‘Z’.
      Si la entrada se ha consumido completamente y la pila se vacía, el autómata acepta en el estado
      <strong>q7</strong>.
    </p>
  </section>
  <script src="pda.js"></script>
</body>

</html>
```

CSS

```
:root {
  --blue: #1A1446;
  --wine: #801638;
  --orange: #E94F37;
  --cream: #FCEFEF;
  --white: #fff;
  --muted: #9aa0b3;
}

/* ------- RESETEO GENERAL ------ */
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

body {
  font-family: ui-sans-serif, system-ui, Arial, Helvetica, sans-serif;
  color: var(--blue);
  background: linear-gradient(135deg, var(--wine) 0 45%, var(--blue) 45% 70%, var(--orange) 70%
100%);
  background-attachment: fixed;
}

/* --------ENCABEZADO -------- */
.hero {
  text-align: center;
  padding: 36px 20px 18px;
  color: var(--white);
}

.hero h1 {
  font-size: 42px;
  margin-bottom: 6px;
}

.hero p {
  opacity: 0.95;
}

/* -------- ESTRUCTURA PRINCIPAL -------- */
.layout {
  display: grid;
  grid-template-columns: 1.6fr 1.4fr;
  gap: 18px;
  padding: 18px;
}

.panel {
```

```
  background: var(--cream);
  border-radius: 16px;
  padding: 16px;
  box-shadow: 0 10px 25px rgba(0, 0, 0, .12);
}

.panel h2,
.panel h3 {
  margin-bottom: 12px;
}

/* -------- GRAFO -------- */
.graph {
  min-height: 420px;
}

#pda-svg {
  width: 100%;
  height: 360px;
  background: #fff;
  border-radius: 12px;
  box-shadow: inset 0 0 0 2px #efeef4;
}

/* Flechas y etiquetas */
.edge {
  stroke: #1A1446;
  stroke-width: 2.2;
  fill: none;
}

.label {
  font-size: 13px;
  fill: #1A1446;
  text-anchor: middle;
  dominant-baseline: middle;
  font-weight: 600;
}

/* -------- NODOS -------- */
.node circle {
  fill: #2b264f;
  stroke: #2b264f;
  stroke-width: 2;
  transition: all .25s ease;
}

.node .double {
  fill: transparent;
  stroke: #2b264f;
  stroke-width: 3;
}
```

```
.node .nodelabel {
  fill: #fff;
  font-weight: 700;
  text-anchor: middle;
  dominant-baseline: middle;
}

.node .start {
  fill: var(--blue);
}

.node.active circle {
  fill: #3b6cff;
  stroke: #2046d4;
  filter: drop-shadow(0 0 6px rgba(59, 108, 255, .6));
}

.node.accept.active circle {
  fill: #1db954;
  stroke: #11883d;
}

/* -------- BOTONES E INPUT -------- */
.controls .row {
  display: flex;
  gap: 8px;
  margin: 8px 0;
}

input {
  flex: 1;
  padding: 10px 12px;
  border-radius: 10px;
  border: 2px solid #d8d8e1;
  outline: none;
  max-width: 450px;
}

button {
  padding: 10px 14px;
  border: none;
  border-radius: 10px;
  color: #fff;
  font-weight: 700;
  background: var(--blue);
  cursor: pointer;
  transition: transform .06s ease, background .2s ease;
}

button:hover {
  transform: translateY(-1px);
}
```

```
button:active {
  transform: translateY(0);
}

button.ghost {
  background: #8c2a41;
}

#btn-step {
  background: #3b6cff;
}

#btn-run {
  background: #1db954;
}

/* -------- PILA -------- */
.stack {
  height: 260px;
  border: 2px solid var(--blue);
  background: #fff;
  border-radius: 12px;
  display: flex;
  flex-direction: column-reverse;
  align-items: center;
  gap: 6px;
  padding: 10px;
  overflow: auto;
}

.stack .cell {
  width: 70%;
  min-height: 30px;
  display: flex;
  align-items: center;
  justify-content: center;
  background: var(--wine);
  color: #fff;
  font-weight: 800;
  border-radius: 8px;
  box-shadow: 0 2px 0 rgba(0, 0, 0, .1);
}

.stack .cell.top {
  background: #f0442b;
}

/* -------- CUADROS BAJO EL GRAFO -------- */
.graph-container {
  display: flex;
  flex-direction: column;
  gap: 25px;
}
```

```
.graph-details {
  display: flex;
  justify-content: space-between;
  align-items: flex-start;
  gap: 20px;
}

.info-box {
  flex: 1;
  background: #fff;
  border-radius: 10px;
  padding: 12px;
  box-shadow: 0 0 8px rgba(0, 0, 0, .1);
  text-align: center;
  min-height: 370px;
}

.info-box h3 {
  color: #801638;
  margin-bottom: 10px;
  border-bottom: 2px solid #801638;
  padding-bottom: 4px;
  font-size: 16px;
}

.elements-table,
.transition-table {
  width: 100%;
  border-collapse: collapse;
  font-size: 13px;
}

.elements-table th,
.transition-table th {
  background: #801638;
  color: #fff;
  padding: 4px;
}

.elements-table td,
.transition-table td {
  border: 1px solid #ddd;
  padding: 4px;
  text-align: center;
}

.elements-table td:nth-child(2) {
  text-align: left;
  padding-left: 10px;
}

.transition-table.small tr:nth-child(even),
```

```
.transition-table.formal tr:nth-child(even) {
  background: #f9f4f6;
}

/* -------- CUADROS DE LA DERECHA -------- */
.grid-cuadros {
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-template-rows: auto auto;
  gap: 16px;
  margin-top: 12px;
}

.grid-cuadros>div {
  background: var(--white);
  border: 2px solid #eceaf4;
  border-radius: 12px;
  padding: 10px;
  box-shadow: 0 3px 8px rgba(0, 0, 0, 0.05);
}

.grid-cuadros h3 {
  color: var(--wine);
  font-size: 16px;
  margin-bottom: 8px;
  border-bottom: 2px solid var(--wine);
  padding-bottom: 3px;
}

/* Contenidos internos */
.stack,
.alert-content,
.log,
.derivacion-content {
  height: 200px;
  overflow-y: auto;
  background: #fff;
  border-radius: 8px;
  border: 1px solid #ddd;
  padding: 10px;
  width: 100%;
}

/* -------- LOG (ID) -------- */
.log {
  font-family: 'Courier New', monospace;
  color: var(--blue);
  font-size: 13px;
  white-space: pre-line;
  background: #fff;
  border-radius: 8px;
  border: 1px solid #ddd;
  padding: 10px;
```

```
}

.log span.active-line {
  display: block;
  background: #e9eefb;
  border-left: 3px solid var(--blue);
  padding-left: 6px;
  transition: all .3s ease;
}

/* -------- DERIVACIÓN -------- */
.derivacion-content {
  font-family: 'Courier New', monospace;
  font-size: 13px;
  white-space: pre-line;
  color: var(--blue);
  height: 300px;
  overflow-y: auto;
  background: #fff;
  border-radius: 8px;
  border: 1px solid #ddd;
  padding: 10px;
  box-sizing: border-box;
}

.derivacion-content strong {
  color: #801638;
}

.derivacion-content em {
  color: #a30000;
  font-style: normal;
  font-weight: 600;
}

/* -------- DESCRIPCIÓN FINAL -------- */
.descripcion-final {
  margin: 40px auto 60px;
  padding: 20px 60px;
  max-width: 1100px;
  color: #fff;
  font-size: 15px;
  line-height: 1.6;
  text-align: justify;
}

.descripcion-final h2 {
  color: var(--cream);
  text-align: center;
  font-size: 22px;
  border-bottom: 2px solid var(--cream);
  display: inline-block;
  margin-bottom: 15px;
```

```
}

/* -------- RESPONSIVE -------- */
@media (max-width: 900px) {
  .layout {
    grid-template-columns: 1fr;
  }

  .graph-details {
    flex-direction: column;
  }

  .grid-cuadros {
    grid-template-columns: 1fr;
  }
}

.gramatica {
  font-family: 'Courier New', monospace;
  font-size: 13px;
  color: var(--blue);
  background: #faf9fc;
  border: 1px solid #ddd;
  border-radius: 6px;
  padding: 8px;
  margin-bottom: 10px;
  white-space: pre-line;

  text-align: left;
  margin-left: 0;
  padding-left: 6px;
  display: block;
}

.id-formal-box {
  width: 200%;
  /* Ajusta el ancho horizontal aquí */
  max-width: 1000px;
  /* Opcional: límite máximo */
  margin: 1rem auto;
  /* Centrado horizontal */
  padding: 1rem;
  background-color: #f9f9f9;
  border: 1px solid #ccc;
  border-radius: 6px;
  box-sizing: border-box;
  max-height: 150px;
  /* Limita la altura */
  overflow-y: auto;
}

@keyframes stateTransition {
  0% {
```

```
    transform: scale(1);
    fill: #1A1446;
    filter: drop-shadow(0 0 0 transparent);
  }

  50% {
    transform: scale(1.25);
    fill: #00C3FF;
    filter: drop-shadow(0 0 10px #00C3FF);
  }

  100% {
    transform: scale(1);
    fill: #1A1446;
    filter: drop-shadow(0 0 0 transparent);
  }
}

@keyframes stateLoop {
  0% {
    transform: scale(1);
    fill: #1A1446;
  }

  30% {
    transform: scale(1.2);
    fill: #3AF9FF;
    filter: drop-shadow(0 0 8px #00C3FF);
  }

  60% {
    transform: scale(1);
    fill: #1A1446;
    filter: none;
  }

  100% {
    transform: scale(1);
    fill: #1A1446;
  }
}

.node.transitioning circle {
  animation: stateTransition 0.6s ease-out;
}

.node.looping circle {
  animation: stateLoop 0.6s ease-out;
```