# Diagramas de Transición y Arquitectura del Analizador Léxico

## 1. ¿Qué es un diagrama de transición?

Un **diagrama de transición** es una representación gráfica del comportamiento del **analizador léxico**. Su función es mostrar cómo este analizador reconoce los diferentes **tokens** de un lenguaje (como identificadores, números, operadores, etc.).

Podemos pensar en un diagrama de transición como un **diagrama de flujo** que indica qué hacer dependiendo del **carácter de entrada** que se está leyendo.
Generalmente, se construye **un diagrama por cada tipo de token** posible.

### Componentes principales

- **Círculos:** representan los **estados** (puntos de decisión del analizador).
- **Flechas:** representan los **movimientos o transiciones** entre estados, según el carácter leído.
- **Círculo doble:** indica un **estado de aceptación o final**, es decir, que se ha reconocido un lexema completo (por ejemplo, un operador o palabra reservada). A la derecha de este círculo suele aparecer la **acción** a realizar (como devolver el token al analizador sintáctico).
- **Asterisco (*) en las flechas:** indica que se ha leído un carácter extra y será necesario "regresarlo" al flujo de entrada porque no pertenece al token actual.
- **Flecha inicial:** indica el **estado de inicio** (imaginario) desde donde comienza el análisis.

### Ejemplo: diagrama de transición para operadores relacionales (relop)

El proceso sería el siguiente:

1. Se lee el primer carácter:
    - Si es <, se lee el siguiente:
        - Si es =, se devuelve `(relop, LE)` → "menor o igual que".
        - Si es >, se devuelve `(relop, NE)` → "distinto de".
        - Si es otro carácter, se devuelve `(relop, LT)` → "menor que" y se regresa el carácter leído.
    - Si el primer carácter es =, se devuelve `(relop, EQ)` → "igual que".

## 2. Reconocimiento de palabras reservadas e identificadores

En la mayoría de los lenguajes, **las palabras reservadas** (como `if`, `while`, `then`, etc.) **no pueden usarse como identificadores**.
Esto genera dos preguntas importantes:

1. ¿Cómo distinguir entre un **identificador** y una **palabra reservada** si ambos coinciden con el mismo patrón?
2. ¿Qué significan las funciones `obten_complex()` e `instal_id()`?

### Manejo de palabras reservadas

Antes de ejecutar el analizador léxico:

- **Se insertan las palabras clave en la tabla de identificadores**.
- Cada entrada se marca indicando que es una palabra reservada.
  De esta forma, cuando el analizador reconoce un lexema:
- Llama a **`instal_id()`**, que verifica si el lexema ya existe en la tabla.
  - Si **no existe**, lo agrega como **identificador**.
  - Si **ya está registrado como palabra clave**, se marca como tal.
  - En ambos casos, devuelve un puntero a la entrada correspondiente.
- Luego, **`obten_complex()`** examina el lexema y determina si el token es un **identificador (`id`)** o una **palabra clave reservada**.

En algunos lenguajes antiguos (como PL/I), **no existían palabras reservadas**, por lo que esta distinción debía hacerse durante el análisis sintáctico.

# 3. Reconocimiento de espacios en blanco

El diagrama de los espacios en blanco es el más sencillo de todos, pues su expresión regular también lo es.

- El símbolo **`delim`** representa cualquiera de los caracteres de espacio en blanco: espacio, tabulación o salto de línea.
- El **asterisco final** en el diagrama indica que deben leerse todos los espacios consecutivos hasta encontrar un carácter que **no sea blanco**.
- En el **estado de aceptación**, **no se realiza ninguna acción**; el analizador simplemente **reinicia** la búsqueda desde el principio, ya que debe seguir buscando el siguiente token.

# 4. Reconocimiento de números

El diagrama para reconocer **números** es más complejo, pero sigue el mismo principio: se deriva directamente de su **expresión regular**.

Cuando se llega a un **estado de aceptación**, el número reconocido se almacena en una **tabla de números**, similar a la tabla de identificadores.

Esta tabla:

- Guarda el valor numérico encontrado.
- Puede incluir información adicional, como si el número es **entero** o **real**.

En algunos lenguajes, también podrían aceptarse **números complejos**, lo que haría el diagrama un poco más extenso.

# 5. Arquitectura de un analizador léxico basado en diagramas de transición

El analizador léxico puede implementarse como una **serie de fragmentos de código**, uno por cada diagrama de transición.

Cada fragmento:

- Contiene **casos o estados numerados** (por ejemplo, `estado 0`, `estado 1`, etc.).
- En cada caso se lee un carácter y, según su valor, se pasa al siguiente estado.
- Cuando se alcanza un **estado de aceptación**, se ejecuta la **acción correspondiente** (como devolver el token al analizador sintáctico).

## Funciones importantes

- **`regresa()`**: devuelve un carácter leído de más al flujo de entrada.
- **`fallo()`**: se ejecuta si ningún camino del diagrama coincide con la entrada actual.
  - No se trata de un error real; solo significa que **ese token no coincide**.
  - El puntero de entrada se restaura al punto de inicio y se intenta con el **siguiente diagrama**.
  - Si ningún diagrama reconoce la entrada, entonces sí hay un **error léxico real** y debe reportarse.

El **orden de los diagramas** es importante: si una entrada coincide con más de un token, **se elige el primero que se prueba.**

```
int estado = 0, inicio = 0;
  int valor_lexico;
  int fallo(){
   delantero=inicio_lexema;
   switch(inicio){
     case 0: inicio = 9; break;
     case 9: inicio = 12; break;
```

```
   case 12: inicio = 25; break;
  case 25: inicio = 28; break;
  case 28: recupera(); break;
  default: /* error */
 }
}

complex sigte_complex(){
 while(1){
    switch (estado) {
      case 0:
             c = sigtecar();
             if(c == blanco || c == tab || c == newline){
                estado = 0;
                inicio_lexema++;
             }
             else if (c == '<') estado = 1;
                  else if (c == '=') estado = 5;
                       else if (c == '>') estado = 6;
                            else estado = fallo();
      break;
  /* aquí debe ir la implementción de los casos 1 al 8 */
     case 9:
             c = sigtecar();
             if (isletter(c)) estado = 10;
             else estado = fallo(); break;
      case 10:
             c = sigtecar();
             if (isletter(c)) estado = 10;
```

```
                else if (isdigit(c)) estado = 10;
                     else estado = 11; break;
        case 11:
                regresa(1);
                valor_lexico = instala_id();
                return(obten_complex());
   /* aquí debe ir la implementción de los casos 12 al 24 */
        case 25:
                c = sigtecar();
                if (isdigit(c)) estado = 26;
                else estado = fallo(); break;
        case 26:
                c = sigtecar();
                if (isdigit(c)) estado = 26;
                else estado = 27; break;
        case 27:
                regresa(1);
                valor_lexico = instala_num();
                return( NUM );
     }
  }
}
```

# 6. Métodos alternativos para combinar diagramas de transición

Existen dos formas adicionales de manejar múltiples diagramas:

### a) Prueba en paralelo

- Cada carácter leído se envía **a todos los diagramas simultáneamente**, siempre que aún no hayan fallado.
- Se debe tener cuidado si un diagrama acepta la entrada antes que otro, ya que podría haber otro que acepte una **cadena más larga** (prefijo mayor).

## b) Combinación en un solo diagrama

- Todos los diagramas pueden **fusionarse en uno solo**.
- Esto es sencillo cuando cada diagrama empieza con **caracteres distintos**, porque solo se necesita un estado inicial con múltiples flechas de salida.
- Es más complicado cuando **varios tokens pueden iniciar con el mismo carácter**, ya que las transiciones se superponen y se deben manejar con cuidado.

## En Resumen

Los **diagramas de transición** son una herramienta fundamental para diseñar e implementar **analizadores léxicos**.
Permiten visualizar de forma clara cómo se reconocen los diferentes tokens de un lenguaje y sirven como base directa para escribir el código del analizador.
A través de ellos, el proceso de lectura, decisión y devolución de tokens se vuelve **estructurado, predecible y fácil de mantener.**