PROCESADORES DE LENGUAJES

TEMA II.- ANÁLISIS LÉXICO

Prof. Dr. Nicolás Luis Fernández García

Departamento de Informática y Análisis Numérico
Escuela Politécnica Superior de Córdoba
Universidad de Córdoba

Programa

- Tema I.- Introducción
- Tema II .- Análisis Lexicográfico
- Tema III.- Fundamentos Teóricos del Análisis Sintáctico
- Tema IV.- Análisis Sintáctico Descendente
- Tema V.- Análisis Sintáctico Ascendente

Programa

1. Introducción
2. Especificación de componentes léxicos
3. Reconocimiento de componentes léxicos
4. Implementación de los analizadores léxicos
5. Detección y recuperación de errores

Programa

Programa

Programa

Programa

Contenido del tema

Contenido de la sección

Introducción

El análisis léxico en el proceso de traducción

Análisis Léxico

- Primera fase del proceso de traducción
- Lee el código fuente "carácter a carácter"
- Genera los componentes léxicos
- Procedimiento auxiliar del Análisis Sintáctico
- Crea la Tabla de Símbolos
- El Gestor de errores procesa los errores léxicos detectados

Introducción

El análisis léxico en el proceso de traducción

Análisis Léxico

- Primera fase del proceso de traducción
- Lee el código fuente "carácter a carácter"
- Genera los componentes léxicos
- Procedimiento auxiliar del Análisis Sintáctico
- Crea la Tabla de Símbolos
- El Gestor de errores procesa los errores léxicos detectados

Introducción

El análisis léxico en el proceso de traducción

Análisis Léxico

- Primera fase del proceso de traducción
- Lee el código fuente "carácter a carácter"
- Genera los componentes léxicos
- Procedimiento auxiliar del Análisis Sintáctico
- Crea la Tabla de Símbolos
- El Gestor de errores procesa los errores léxicos detectados

Introducción

El análisis léxico en el proceso de traducción

Análisis Léxico

- Primera fase del proceso de traducción
- Lee el código fuente "carácter a carácter"
- Genera los componentes léxicos
- Procedimiento auxiliar del Análisis Sintáctico
- Crea la Tabla de Símbolos
- El Gestor de errores procesa los errores léxicos detectados

Introducción

El análisis léxico en el proceso de traducción

Análisis Léxico

- Primera fase del proceso de traducción
- Lee el código fuente "carácter a carácter"
- Genera los componentes léxicos
- Procedimiento auxiliar del Análisis Sintáctico
- Crea la Tabla de Símbolos
- El Gestor de errores procesa los errores léxicos detectados

Introducción

El análisis léxico en el proceso de traducción

Análisis Léxico

- Primera fase del proceso de traducción
- Lee el código fuente “carácter a carácter”
- Genera los componentes léxicos
- Procedimiento auxiliar del Análisis Sintáctico
- Crea la Tabla de Símbolos
- El Gestor de errores procesa los errores léxicos detectados

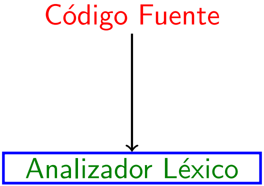
Introducción

El análisis léxico en el proceso de traducción

Código Fuente

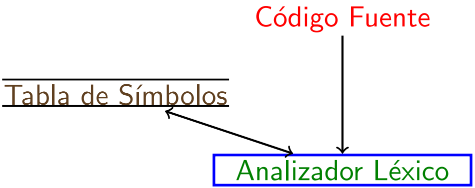
Introducción

El análisis léxico en el proceso de traducción



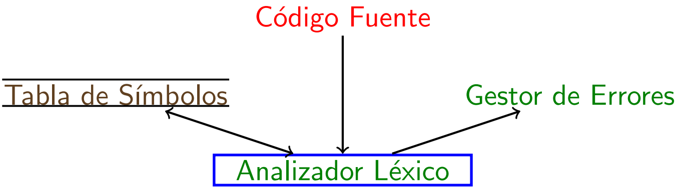
Introducción

El análisis léxico en el proceso de traducción



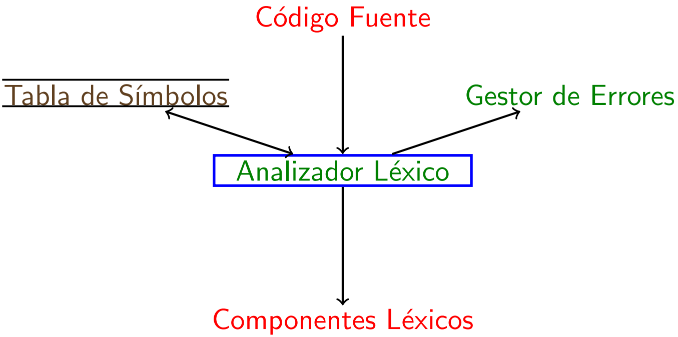
Introducción

El análisis léxico en el proceso de traducción



Introducción

El análisis léxico en el proceso de traducción



Contenido de la sección

Introducción

Componentes Léxicos

Definición (Componente léxico)

Elemento más simple con significado propio de un lenguaje de programación.

Ejemplo

- *Identificadores: variables, palabras claves, ...*
- *Números*
- *Cadenas de caracteres*
- *Operadores: aritméticos, relacionales, lógicos, ...*
- *Signos de puntuación*
- *Etc.*

Introducción

Componentes Léxicos

Definición (Componente léxico)

Elemento más simple con significado propio de un lenguaje de programación.

Ejemplo

- *Identificadores*: variables, palabras claves, ...
- *Números*
- *Cadenas de caracteres*
- *Operadores: aritméticos, relacionales, lógicos, ...*
- *Signos de puntuación*
- *Etc.*

Introducción

Componentes Léxicos

Definición (Componente léxico)

Elemento más simple con significado propio de un lenguaje de programación.

Ejemplo

- *Identificadores: variables, palabras claves, ...*
- *Números*
- *Cadenas de caracteres*
- *Operadores: aritméticos, relacionales, lógicos, ...*
- *Signos de puntuación*
- *Etc.*

Introducción

Componentes Léxicos

Definición (Componente léxico)

Elemento más simple con significado propio de un lenguaje de programación.

Ejemplo

- *Identificadores: variables, palabras claves, ...*
- *Números*
- *Cadenas de caracteres*
- *Operadores: aritméticos, relacionales, lógicos, ...*
- *Signos de puntuación*
- *Etc.*

Introducción

Componentes Léxicos

Definición (Componente léxico)

Elemento más simple con significado propio de un lenguaje de programación.

Ejemplo

- *Identificadores: variables, palabras claves, ...*
- *Números*
- *Cadenas de caracteres*
- *Operadores: aritméticos, relacionales, lógicos, ...*
- *Signos de puntuación*
- *Etc.*

Introducción

Componentes Léxicos

Definición (Componente léxico)

Elemento más simple con significado propio de un lenguaje de programación.

Ejemplo

- *Identificadores: variables, palabras claves, ...*
- *Números*
- *Cadenas de caracteres*
- *Operadores: aritméticos, relacionales, lógicos, ...*
- *Signos de puntuación*
- *Etc.*

Introducción

Componentes Léxicos

Definición (Componente léxico)

Elemento más simple con significado propio de un lenguaje de programación.

Ejemplo

- *Identificadores: variables, palabras claves, ...*
- *Números*
- *Cadenas de caracteres*
- *Operadores: aritméticos, relacionales, lógicos, ...*
- *Signos de puntuación*
- *Etc.*

Introducción

Componentes Léxicos

Los componentes léxicos

- También se denominan "tokens"
- Son los símbolos terminales de las gramáticas de contexto libre que generan los lenguajes de programación.
- Son en realidad códigos numéricos

Introducción

Componentes Léxicos

Los componentes léxicos

- También se denominan "tokens"
- Son los símbolos terminales de las gramáticas de contexto libre que generan los lenguajes de programación.
- Son en realidad códigos numéricos

Introducción

Componentes Léxicos

Los componentes léxicos

- También se denominan "tokens"
- Son los símbolos terminales de las gramáticas de contexto libre que generan los lenguajes de programación.
- Son en realidad códigos numéricos

Contenido de la sección

Introducción

Tabla de Símbolos

Tabla de Símbolos

- Se crea durante el Análisis Léxico
- Puede almacenar:
  - Números
  - Cadenas
  - ...
  - Pero, sobre todo, identificadores

Introducción

Tabla de Símbolos

Tabla de Símbolos

- Cuando el analizador léxico reconoce un identificador:
  - Se inserta el identificador en la Tabla de Símbolos
  - Devuelve el componente léxico de identificador y un puntero o índice a su posición en la Tabla de Símbolos
- La información del identificador depende de su naturaleza:
  - Variable: tipo de dato, valor, ...
  - Función: número y tipo de argumentos, ...
  - Etc.

Introducción

Tabla de Símbolos

Tabla de Símbolos

- Cuando el analizador léxico reconoce un identificador:
  - Se inserta el identificador en la Tabla de Símbolos
  - Devuelve el componente léxico de identificador y un puntero o índice a su posición en la Tabla de Símbolos
- La información del identificador depende de su naturaleza:
  - Variable: tipo de dato, valor, ...
  - Función: número y tipo de argumentos, ...
  - Etc.

Introducción

Tabla de Símbolos

Ejemplo (*dividendo = divisor * cociente + resto*)

Nombre	**Tipo**	**Valor**	...
cociente			
dividendo			
divisor			
resto			

Introducción

Tabla de Símbolos

Nota

La información de los identificadores se completa durante todas las fases del proceso de traducción

Contenido de la sección

Introducción

Palabras claves

Palabras claves o reservadas

- Son identificadores con un significado especial
- Algunos lenguajes de programación también permiten usar las palabras claves como variables
  - + Las palabras claves no serían palabras reservadas
  - + Dificulta la legibilidad de los programas

Ejemplo (PL/I)

$$IF\ (IF > 1)\ THEN\ THEN = 0;$$

Introducción

Palabras claves

Palabras claves o reservadas

- Son identificadores con un significado especial
- Algunos lenguajes de programación también permiten usar las palabras claves como variables
  - + Las palabras claves no serían palabras reservadas
  - + Dificulta la legibilidad de los programas

Ejemplo (PL/I)

$$IF\ (IF > 1)\ THEN\ THEN = 0;$$

Introducción

Palabras claves

Palabras claves o reservadas

- Son identificadores con un significado especial
- Algunos lenguajes de programación también permiten usar las palabras claves como variables
  - + Las palabras claves no serían palabras reservadas
  - + Dificulta la legibilidad de los programas

Ejemplo (PL/I)

$$IF\ (IF > 1)\ THEN\ THEN = 0;$$

Introducción

Palabras claves

Palabras claves

- Tipos de reconocimiento de las palabras claves:
  1.- **Implícito**: preinstalación en la Tabla de Símbolos
  2.- **Explícito**: reconocimiento específico de cada palabra clave.

Introducción

Palabras claves

Palabras claves

- Tipos de reconocimiento de las palabras claves:
  1.- **Implícito**: preinstalación en la Tabla de Símbolos
  2.- **Explícito**: reconocimiento específico de cada palabra clave.

Introducción

Palabras claves

Palabras claves

- Tipos de reconocimiento de las palabras claves:
  1.- **Implícito**: preinstalación en la Tabla de Símbolos
  2.- **Explícito**: reconocimiento específico de cada palabra clave.

Introducción

Palabras claves

Palabras claves

1.- Preinstalación en la Tabla de Símbolos:

- Se almacena el nombre y el componente léxico
- Al reconocer un identificador, se consulta la Tabla de Símbolos para comprobar si es una palabra clave o no.
- Ventajas
  - La programación del analizador léxico es más sencilla
- Inconvenientes
  - El reconocimiento es más lento
  - Se necesita más memoria para la Tabla de Símbolos

Introducción

Palabras claves

Palabras claves

1.- Preinstalación en la Tabla de Símbolos:

+ Se almacena el nombre y el componente léxico
+ Al reconocer un identificador, se consulta la Tabla de Símbolos para comprobar si es una palabra clave o no.
+ Ventajas
  - La programación del analizador léxico es más sencilla
+ Inconvenientes
  - El reconocimiento es más lento
  - Se necesita más memoria para la Tabla de Símbolos

Introducción

Palabras claves

Palabras claves

1.- Preinstalación en la Tabla de Símbolos:

- Se almacena el nombre y el componente léxico

Ejemplo

Nombre	**Componente Léxico**	...	...
if	*IF*		
while	*WHILE*		
...			

- Al reconocer un identificador, se consulta la Tabla de Símbolos para comprobar si es una palabra clave o no.
- Ventajas
  - La programación del analizador léxico es más sencilla

Introducción

Palabras claves

Palabras claves

1.- Preinstalación en la Tabla de Símbolos:

- Se almacena el nombre y el componente léxico
- Al reconocer un identificador, se consulta la Tabla de Símbolos para comprobar si es una palabra clave o no.
- Ventajas
  - La programación del analizador léxico es más sencilla
- Inconvenientes
  - El reconocimiento es más lento
  - Se necesita más memoria para la Tabla de Símbolos

Introducción

Palabras claves

Palabras claves

1.- Preinstalación en la Tabla de Símbolos:

+ Se almacena el nombre y el componente léxico
+ Al reconocer un identificador, se consulta la Tabla de Símbolos para comprobar si es una palabra clave o no.
+ Ventajas
  - La programación del analizador léxico es más sencilla
+ Inconvenientes
  - El reconocimiento es más lento
  - Se necesita más memoria para la Tabla de Símbolos

Introducción

Palabras claves

Palabras claves

1.- Preinstalación en la Tabla de Símbolos:

- Se almacena el nombre y el componente léxico
- Al reconocer un identificador, se consulta la Tabla de Símbolos para comprobar si es una palabra clave o no.
- Ventajas
  - La programación del analizador léxico es más sencilla
- Inconvenientes
  - El reconocimiento es más lento
  - Se necesita más memoria para la Tabla de Símbolos

Introducción

Palabras claves

Palabras claves

2.- Reconocimiento específico de cada palabra clave.

+ Cada palabra clave es reconocida de forma independiente de los demás identificadores y palabras claves
+ Ventajas
  - El reconocimiento es más rápido
  - No se necesita aumentar la memoria de la Tabla de Símbolos: las palabras claves no se almacenan
+ Inconvenientes
  - La programación del analizador léxico es más compleja

Introducción

Palabras claves

Palabras claves

2.- Reconocimiento específico de cada palabra clave.

+ Cada palabra clave es reconocida de forma independiente de los demás identificadores y palabras claves
+ Ventajas
  - El reconocimiento es más rápido
  - No se necesita aumentar la memoria de la Tabla de Símbolos: las palabras claves no se almacenan
+ Inconvenientes
  - La programación del analizador léxico es más compleja

Introducción

Palabras claves

Palabras claves

2.- Reconocimiento específico de cada palabra clave.

+ Cada palabra clave es reconocida de forma independiente de los demás identificadores y palabras claves
+ Ventajas
  - El reconocimiento es más rápido
  - No se necesita aumentar la memoria de la Tabla de Símbolos: las palabras claves no se almacenan
+ Inconvenientes
  - La programación del analizador léxico es más compleja

Contenido de la sección

1. Introducción
   - El análisis léxico en el proceso de traducción
   - Componentes Léxicos
   - Tabla de Símbolos
   - Palabras claves
   - Ejemplo
   - Autonomía del analizador léxico
2. Especificación de componentes léxicos
3. Reconocimiento de componentes léxicos
4. Implementación de los analizadores léxicos

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

*dividendo = divisor * cociente + resto ;*

- dividendo
  - Es reconocido como IDENTIFICADOR
  - Se devuelve el componente léxico y el puntero a la Tabla de Símbolos

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

*dividendo = divisor * cociente + resto ;*

- Se eliminan
  - Espacios en blanco
  - Tabuladores
  - Saltos de línea

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

*dividendo = divisor * cociente + resto ;*

- Signo =
  - + Se devuelve el token de ASIGNACIÓN

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

$$dividendo = divisor * cociente + resto ;$$

Nota

- *El Análisis Sintáctico sólo necesita saber que se ha reconocido el componente léxico ASIGNACIÓN*
- *No le importa si el símbolo es $=$ o $:=$ o cualquier otro*
- *No interesa el texto concreto, sino la categoría a la que pertenece*

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

*dividendo = divisor * cociente + resto ;*

- divisor
  - Es reconocido como IDENTIFICADOR
  - Se devuelve el componente léxico y el puntero a la Tabla de Símbolos

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

$$dividendo = divisor * cociente + resto ;$$

- Signo *
  - Se devuelve el token de PRODUCTO

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

*dividendo = divisor * cociente + resto ;*

- cociente
  - + Es reconocido como IDENTIFICADOR
  - + Se devuelve el componente léxico y el puntero a la Tabla de Símbolos

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

*dividendo = divisor * cociente + resto ;*

- Signo +
  - + Se devuelve el token de SUMA

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

*dividendo = divisor * cociente + resto ;*

- resto
  - Es reconocido como IDENTIFICADOR
  - Se devuelve el componente léxico y el puntero a la Tabla de Símbolos

Introducción

Ejemplo

Ejemplo

Sentencia del lenguaje C

*dividendo = divisor * cociente + resto ;*

- Signo ;
  + Se devuelve el token de FIN DE SENTENCIA

Contenido de la sección

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
  - + La separación de tareas facilita el mantenimiento y mejora del traductor
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
  - Los componentes léxicos pueden ser denotados por Expresiones Regulares
  - Los Autómatas Finitos Deterministas (AFD) reconocen las palabras denotadas por las expresiones regulares
  - Los Analizadores Léxicos están basados en los Autómatas Finitos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
  - Los componentes léxicos pueden ser denotados por Expresiones Regulares
  - Los Autómatas Finitos Deterministas (AFD) reconocen las palabras denotadas por las expresiones regulares
  - Los Analizadores Léxicos están basados en los Autómatas Finitos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
  - Los componentes léxicos pueden ser denotados por Expresiones Regulares
  - Los Autómatas Finitos Deterministas (AFD) reconocen las palabras denotadas por las expresiones regulares
  - Los Analizadores Léxicos están basados en los Autómatas Finitos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
  - El Análisis Léxico es la única fase que tiene contacto con el código fuente
  - Puede procesar el texto: eliminar espacios en blanco, comentarios
  - Almacena la posición de los saltos de línea para informar sobre la localización de los errores detectados
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
  - + El Análisis Léxico es la única fase que tiene contacto con el código fuente
  - + Puede procesar el texto: eliminar espacios en blanco, comentarios
  - + Almacena la posición de los saltos de línea para informar sobre la localización de los errores detectados
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
  + El Análisis Léxico es la única fase que tiene contacto con el código fuente
  + Puede procesar el texto: eliminar espacios en blanco, comentarios
  + Almacena la posición de los saltos de línea para informar sobre la localización de los errores detectados
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
  - + Las operaciones de lectura / escritura son computacionalmente muy costosas
  - + Se puede mejorar la eficiencia si se codifican con sentencias de bajo nivel: ensamblador, ...
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
  - Las operaciones de lectura / escritura son computacionalmente muy costosas
  - Se puede mejorar la eficiencia si se codifican con sentencias de bajo nivel: ensamblador, ...
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad
  - La codificación de los caracteres pueden variar de un entorno de ejecución a otro: ASCII, EBCDIC, ...
  - El cambio de codificación sólo requerirá modificar el Analisis Léxico, no siendo necesario modificar el resto de fases.

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad
  - + La codificación de los caracteres pueden variar de un entorno de ejecución a otro: ASCII, EBCDIC, ...
  - + El cambio de codificación sólo requerirá modificar el Analisis Léxico, no siendo necesario modificar el resto de fases.

Introducción

Autonomía del analizador léxico

Razones para separar el análisis léxico del análisis sintáctico

- Modularidad
- Menor complejidad de los componentes léxicos
- Pre-procesamiento del código fuente
- Mejora en la eficiencia del analizador léxico
- Portabilidad

Contenido del tema

Contenido de la sección

1. Introducción
2. Especificación de componentes léxicos
   - Descripción de los componentes léxicos
   - Palabras y lenguajes formales
   - Expresiones Regulares
3. Reconocimiento de componentes léxicos
4. Implementación de los analizadores léxicos
5. Detección y recuperación de errores

Especificación de componentes léxicos

Descripción de los componentes léxicos

Componentes léxicos

- Elementos básicos de los lenguajes de programación
- Se describen mediante
  - Una descripción informal
  - Una descripción formal mediante expresiones regulares
  - Ejemplos o paradigmas

Especificación de componentes léxicos

Descripción de los componentes léxicos

Componentes léxicos

- Elementos básicos de los lenguajes de programación
- Se describen mediante
  - Una descripción informal
  - Una descripción formal mediante expresiones regulares
  - Ejemplos o paradigmas

Especificación de componentes léxicos

Descripción de los componentes léxicos

Componentes léxicos

- Elementos básicos de los lenguajes de programación
- Se describen mediante
  - Una descripción informal
  - Una descripción formal mediante expresiones regulares
  - Ejemplos o paradigmas

Especificación de componentes léxicos

Descripción de los componentes léxicos

Componentes léxicos

- Elementos básicos de los lenguajes de programación
- Se describen mediante
  - Una descripción informal
  - Una descripción formal mediante expresiones regulares
  - Ejemplos o paradigmas

Especificación de componentes léxicos

Descripción de los componentes léxicos

Componentes léxicos

- Elementos básicos de los lenguajes de programación
- Se describen mediante
  - Una descripción informal
  - Una descripción formal mediante expresiones regulares
  - Ejemplos o paradigmas

Especificación de componentes léxicos

Descripción de los componentes léxicos

Ejemplo (Componentes léxicos en el lenguaje C 1 / 7)

- *IDENTIFICADOR*
- **Descripción informal:** *cadenas de caracteres compuestas por letras, cifras y el símbolo de subrayado, pero que no comienza por una cifra.*
- **Descripción formal:**

  $(letra + subrayado)(letra + cifra + subrayado)^*$
- **Ejemplos o paradigmas:**

  dividendo, divisor, cociente, resto, suma_total, x_1, ...

Especificación de componentes léxicos

Descripción de los componentes léxicos

Ejemplo (Componentes léxicos en el lenguaje C 2 / 7)

- *NÚMERO*
- **Descripción informal:** *números enteros, reales, ...*
- **Descripción formal:**

$$cifra\ cifra^{*}\ (\epsilon + .cifra^{*}(\epsilon + (E + e)(\epsilon + "+" + "-")cifra\ cifra^{*}))$$

- **Ejemplos o paradigmas:** *9, 19.7, 97.7e2, ...*

Especificación de componentes léxicos

Descripción de los componentes léxicos

Ejemplo (Componentes léxicos en el lenguaje C 3 / 7)

- *IF*
- **Descripción informal:** *palabra clave de la sentencia condicional if*
- **Descripción formal:** *if*
- **Ejemplo o paradigma:** *if*

Especificación de componentes léxicos

Descripción de los componentes léxicos

Ejemplo (Componentes léxicos en el lenguaje C 4 / 7)

- *FOR*
- **Descripción informal:** *palabra clave de la sentencia de repetición for*
- **Descripción formal:** *for*
- **Ejemplo o paradigma:** *for*

Especificación de componentes léxicos

Descripción de los componentes léxicos

Ejemplo (Componentes léxicos en el lenguaje C 5 / 7)

- *ASIGNACIÓN*
- **Descripción informal:** *signo igual para la sentencia de asignación*
- **Descripción formal:** =
- **Ejemplo o paradigma:** =

Especificación de componentes léxicos

Descripción de los componentes léxicos

Ejemplo (Componentes léxicos en el lenguaje C 6 / 7)

- *MAYOR_IGUAL_QUE*
- **Descripción informal:** *operador relacional mayor o igual que*
- **Descripción formal:** $>=$
- **Ejemplo o paradigma:** $>=$

Especificación de componentes léxicos

Descripción de los componentes léxicos

Ejemplo (Componentes léxicos en el lenguaje C 7 / 7)

- *FIN_SENTENCIA*
- **Descripción informal:** *signo de punto y coma para indicar el fin de una sentencia*
- **Descripción formal:** ;
- **Ejemplo o paradigma:** ;

Especificación de componentes léxicos

Descripción de los componentes léxicos

Nota

Sólo interesa saber el significado (Componente Léxico) que se asocia a uno o más signos, no cómo son dichos signos.

Ejemplo (Lenguaje Fortran)

La expresión regular para el componente léxico MAYOR_IGUAL_QUE es: *.(G+g)(E+e).*

- *.GE.*
- *.gE.*
- *.Ge.*
- *.ge.*

Contenido de la sección

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Alfabeto o vocabulario)

- *Conjunto finito y no vacío de símbolos que permiten formar la palabras pertenecientes a un lenguaje*
- *Se suele denotar por* Σ *o* V
- $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$

Especificación de componentes léxicos

Palabras y lenguajes formales

Ejemplo

- $\Sigma_1 = \{0, 1\}$ *(Alfabeto binario)*
- $\Sigma_2 = \{0, 1, 2, \cdots, 9\}$
- $\Sigma_3 = \{a, b, c, \cdots, z\}$
- $\Sigma_4 = \{if, else\}$
- $\Sigma_5 = \{ab, ca, bbc\}$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Palabra o cadena)

- *Secuencia finita de símbolos pertenecientes a un alfabeto*

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Palabra o cadena)

- *Secuencia finita de símbolos pertenecientes a un alfabeto*

Ejemplo (Palabras definidas sobre)

- $\Sigma_1 = \{0, 1\}$: *0, 1, 00, 01, 10, 11, ..., 010101, ...*
- $\Sigma_3 = \{a, b, c, \cdots, z\}$: *aab, valor, punto,...*
- $\Sigma_5 = \{ab, ca, bbc\}$: *ab, bbc, abab, abbbc,...*

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Longitud de una palabra x)

- *Número de símbolos de un alfabeto que componen dicha palabra.*
- *Se denota por* $|x|$
- *Si* $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$ *y* $x = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_k}$ *entonces* $|x| = k$

Especificación de componentes léxicos

Palabras y lenguajes formales

Nota

La longitud de una palabra depende del alfabeto sobre el que esté definida.

Especificación de componentes léxicos

Palabras y lenguajes formales

Nota

La longitud de una palabra depende del alfabeto sobre el que esté definida.

Ejemplo (Longitud de la palabra $x = abab$ definida sobre)

- $\Sigma_3 = \{a, b, c, \cdots, z\}$: $|x| = 4$
- $\Sigma_5 = \{ab, ca, bbc\}$: $|x| = 2$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Lenguaje universal definido sobre Σ)

- *Conjunto de palabras compuestas por cero o más símbolos de Σ.*
- *Se representa por Σ^*.*

Especificación de componentes léxicos

Palabras y lenguajes formales

Ejemplo (Palabras definidas sobre un alfabeto Σ)

- *Si* $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$ *entonces* Σ^* *se puede generar a partir de las palabras de:*
  - $|x| = 0$: *"palabra vacía" que se denota por* ϵ *o* λ.
  - $|x| = 1$: $x = \sigma_1$, $x = \sigma_2$, $\cdots$, $x = \sigma_n$
  - $|x| = 2$: $x = \sigma_1\sigma_1$, $x = \sigma_1\sigma_2$, $\cdots$
  - *Etc.*

Especificación de componentes léxicos

Palabras y lenguajes formales

Ejemplo (Palabras definidas sobre $\Sigma = \{a, b, c\}$)

- $|x| = 0$: $x = \epsilon$.
- $|x| = 1$: $x = a$, $x = b$, $x = c$
- $|x| = 2$: $x = aa$, $x = ab$, $x = ac$, $x = ba$, $\cdots$.
- *Etc.*
- *En resumen,* $\Sigma^* = \{\epsilon, a, b, c, aa, ab, ac, ba, \cdots\}$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Lenguajes formales)

- *L es un lenguaje formal definido sobre Σ si $L \subseteq \Sigma^*$*

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Lenguajes formales)

- *L es un lenguaje formal definido sobre Σ si $L \subseteq \Sigma^*$*

Ejemplo (Lenguajes formales definidos sobre Σ)

- $L_\emptyset = \emptyset$.
- Σ
- Σ^*
- $L_\sigma = \{\sigma\}$ *donde* $\sigma \in \Sigma$
- $L_\epsilon = \{\epsilon\}$
- $\Sigma^+ = \{x | x \in \Sigma^* \wedge |x| \geq 1\}$

Especificación de componentes léxicos

Palabras y lenguajes formales

Nota

- $L_\epsilon = \{\epsilon\} \neq L_\emptyset = \emptyset$
- $\epsilon \in \Sigma^+ \Longleftrightarrow \epsilon \in \Sigma$

Especificación de componentes léxicos

Palabras y lenguajes formales

Ejemplo (Lenguajes formales sobre $\Sigma = \{a, b, c\}$)

- $L_\emptyset = \emptyset$
- $L_\epsilon = \{\epsilon\}$
- $\Sigma = \{a, b, c\}$
- $\Sigma^* = \{\epsilon, a, b, c, aa, ab, ac, \cdots\}$
- $L_a = \{a\}$
- $L = \{a, ab, abb, abbb, \cdots\}$

Nota

L puede ser denotado por la expresión regular ab^*

Especificación de componentes léxicos

Palabras y lenguajes formales

Operaciones con palabras

- Concatenación
- Potencia

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Concatenación de palabras)

- *Sea* $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$
- $x = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_p}$, $y = \sigma_{j_1}\sigma_{j_2}\cdots\sigma_{j_q} \in \Sigma^*$
- *La concatenación de* x *con* y *se denota por* $x \cdot y$ *o simplemente* xy
- $xy = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_p}\sigma_{j_1}\sigma_{j_2}\cdots\sigma_{j_q}$

Ejemplo (Concatenación de palabras sobre $\Sigma = \{a, b, c\}$)

Si $x = ab$, $y = bcc$ *entonces* $xy = abbcc$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Concatenación de palabras)

- *Sea* $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$
- $x = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_p},\ y = \sigma_{j_1}\sigma_{j_2}\cdots\sigma_{j_q} \in \Sigma^*$
- *La concatenación de* x *con* y *se denota por* $x \cdot y$ *o simplemente* xy
- $xy = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_p}\sigma_{j_1}\sigma_{j_2}\cdots\sigma_{j_q}$

Ejemplo (Concatenación de palabras sobre $\Sigma = \{a, b, c\}$)

$$Si\ x = ab\ ,\ y = bcc\ \ entonces\ xy = abbcc$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la concatenación de palabras

- **Operación cerrada sobre Σ^*:** si $x, y \in \Sigma^*$ entonces $xy \in \Sigma^*$
- $|xy| = |x| + |y|$
- **Asociativa:** $x(yz) = (xy)z = xyz$
- **Existencia de elemento neutro:** ϵ.

$$x\epsilon = \epsilon x = x$$

- **No conmutativa:** $xy \neq yx$

Ejemplo (No conmutativa)

- *Sea* $\Sigma = \{a, b, c\}$
- *Si* $x = ab$, $y = bcc$ *entonces* $xy = abbcc \neq bccab = yx$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Potencia de una palabra)

- *Sea* $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$
- $x = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_p} \in \Sigma^*$
- *La potencia "i-ésima" de x se denota por* x^i
- $x^i = \underbrace{xx\cdots x}_{i\ veces}$

Especificación de componentes léxicos

Palabras y lenguajes formales

Ejemplo (Potencia una palabra definida sobre $\Sigma = \{a, b, c\}$)

Si $x = abb$ *entonces*

- $x^0 = \epsilon$
- $x^1 = x = abb$
- $x^2 = xx = abbabb$
- $x^3 = xxx = abbabbabb$
- *Etc.*

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Potencia de una palabra (versión recursiva))

- $x^0 = \epsilon$
- $x^i = x x^{i-1}$

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la potencia de una de palabra

- **Operación cerrada sobre Σ^*:**

  $$\text{si } x \in \Sigma^* \text{ entonces } \forall i \in \mathbb{N}, x^i \in \Sigma^*$$

- $|x^i| = i|x|$

Especificación de componentes léxicos

Palabras y lenguajes formales

Operaciones con lenguajes formales

- Unión
- Concatenación
- Potencia
- Clausura o cierre de Kleene
- Clausura positiva
- Intersección
- Diferencia
- Complementación

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Unión de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$$

Ejemplo

Si $L_1 = \{ab, bc, bcc\}$ *y* $L_2 = \{a, bc, abb, ac\}$

entonces $L_1 \cup L_2 = \{ab, bc, bcc, a, abb, ac\}$

Nota

No hay palabras o cadenas repetidas

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Unión de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$$

Ejemplo

Si $L_1 = \{ab, bc, bcc\}$ *y* $L_2 = \{a, bc, abb, ac\}$

entonces $L_1 \cup L_2 = \{ab, bc, bcc, a, abb, ac\}$

Nota

No hay palabras o cadenas repetidas

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Unión de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 \cup L_2 = \{x | x \in L_1 \vee x \in L_2\}$$

Ejemplo

Si $L_1 = \{ab, bc, bcc\}$ *y* $L_2 = \{a, bc, abb, ac\}$

entonces $L_1 \cup L_2 = \{ab, bc, bcc, a, abb, ac\}$

Nota

No hay palabras o cadenas repetidas

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la unión de lenguajes

- **Operación cerrada sobre Σ^*:**

$$\text{si } L_1, L_2 \subseteq \Sigma^* \text{ entonces } L_1 \cup L_2 \subseteq \Sigma^*$$

- **Asociativa:** $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3 = L_1 \cup L_2 \cup L_3$
- **Conmutativa:** $L_1 \cup L_2 = L_2 \cup L_1$
- **Existencia de elemento neutro:** $\emptyset$

$$L \cup \emptyset = \emptyset \cup L = L$$

- **Idempotente:** $L \cup L = L$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Concatenación de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 L_2 = \{x | x = yz \wedge y \in L_1 \wedge z \in L_2\}$$

Ejemplo

Si $L_1 = \{ab, a, bb\}$ *y* $L_2 = \{bc, c, aa\}$

entonces $L_1 L_2 = \{abbc, abc, abaa, ac, aaa, bbbc, bbc, bbaa\}$

Nota

No hay palabras o cadenas repetidas

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Concatenación de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 L_2 = \{x | x = yz \wedge y \in L_1 \wedge z \in L_2\}$$

Ejemplo

Si $L_1 = \{ab, a, bb\}$ *y* $L_2 = \{bc, c, aa\}$

entonces $L_1 L_2 = \{abbc, abc, abaa, ac, aaa, bbbc, bbc, bbaa\}$

Nota

No hay palabras o cadenas repetidas

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Concatenación de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 L_2 = \{x | x = yz \wedge y \in L_1 \wedge z \in L_2\}$$

Ejemplo

Si $L_1 = \{ab, a, bb\}$ *y* $L_2 = \{bc, c, aa\}$

entonces $L_1 L_2 = \{abbc, abc, abaa, ac, aaa, bbbc, bbc, bbaa\}$

Nota

No hay palabras o cadenas repetidas

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la concatenación de lenguajes

- **Operación cerrada sobre Σ^*:**

$$\text{si } L_1, L_2 \subseteq \Sigma^* \text{ entonces } L_1 L_2 \subseteq \Sigma^*$$

- **Asociativa:** $L_1 (L_2 L_3) = (L_1 L_2) L_3 = L_1 L_2 L_3$
- **Existencia de elemento neutro:** $L_\epsilon = \{\epsilon\}$

$$L\{\epsilon\} = \{\epsilon\} L = L$$

- **No conmutativa:** $L_1 L_2 \neq L_2 L_1$

Nota

La concatenación lenguajes formales **no** *es conmutativa porque la concatenación de palabras tampoco lo es*

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la concatenación de lenguajes

- **Operación cerrada sobre Σ^*:**

$$\text{si } L_1, L_2 \subseteq \Sigma^* \text{ entonces } L_1 L_2 \subseteq \Sigma^*$$

- **Asociativa:** $L_1\,(L_2\,L_3) = (L_1\,L_2)\,L_3 = L_1\,L_2\,L_3$
- **Existencia de elemento neutro:** $L_\epsilon = \{\epsilon\}$

$$L\,\{\epsilon\} = \{\epsilon\}\,L = L$$

- **No conmutativa:** $L_1 L_2 \neq L_2 L_1$

Nota

La concatenación lenguajes formales **no** *es conmutativa porque la concatenación de palabras tampoco lo es*

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Potencia de un lenguaje formal)

Si $L \subseteq \Sigma^*$ *e* $i \in \mathbb{N}$ *entonces*

$$L^i = \{x | x = x_{j_1} \cdots x_{j_i} \wedge x_{j_1}, \cdots, x_{j_i} \in L\}$$

Ejemplo

$$L^0 = \{x^0 | x \in L\} = \{\epsilon\}$$

$$L^1 = L$$

$$L^2 = LL$$

$$\ldots$$

$$L^i = LL^{i-1}$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Potencia de un lenguaje formal)

Si $L \subseteq \Sigma^*$ *e* $i \in \mathbb{N}$ *entonces*

$$L^i = \{x | x = x_{j_1} \cdots x_{j_i} \wedge x_{j_1}, \cdots, x_{j_i} \in L\}$$

Ejemplo

$$L^0 = \{x^0 | x \in L\} = \{\epsilon\}$$

$$L^1 = L$$

$$L^2 = LL$$

$$\ldots$$

$$L^i = LL^{i-1}$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la potencia de un lenguaje formal

- **Operación cerrada sobre Σ^*:**

$$\text{si } L \subseteq \Sigma^* \text{ entonces } \forall i \in \mathbb{N} \quad L^i \subseteq \Sigma^*$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Clausura de Kleene de un lenguaje formal)

Si $L \subseteq \Sigma^*$ *entonces*

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \cdots$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Ejemplo

Si $L = \{a\}$ *entonces*

$$\begin{aligned} L^* &= \bigcup_{i=0}^{\infty} L^i \\ &= L^0 \cup L^1 \cup L^2 \cup \cdots \\ &= \{\epsilon\} \cup \{a\} \cup \{aa\} \cdots \\ &= \{\epsilon, a, aa, \cdots\} \end{aligned}$$

$L^* = \{a\}^*$ *puede ser denotado por la expresión regular* a^*

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la clausura de un lenguaje formal

- Operación cerrada sobre Σ^*:

  $$\text{si } L \subseteq \Sigma^* \text{ entonces } L^* \subseteq \Sigma^*$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Clausura positiva de un lenguaje formal)

Si $L \subseteq \Sigma^*$ *entonces*

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L^1 \cup L^2 \cup \cdots$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la clausura positiva de un lenguaje formal

- **Operación cerrada sobre Σ^*:**

$$\text{si } L \subseteq \Sigma^* \text{ entonces } L^+ \subseteq \Sigma^*$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Ejemplos (Operaciones con lenguajes formales)

Si $L_1 = \{a\}$ *y* $L_2 = \{b\}$ *entonces*

$$
\begin{aligned}
L_3 &= L_1 L_2 = \{ab\} \\
L_4 &= L_2 L_1 = \{ba\} \\
L_5 &= L_3 \cup L_4 = \{ab, ba\} \\
L_6 &= L_5^* = \bigcup_{i=0}^{\infty} L_5^i = L_5^0 \cup L_5^1 \cup L_5^2 \cup \cdots \\
&= \{\epsilon\} \cup \{ab, ba\} \cup \{abab, abba, baab, baba\} \cdots \\
&= \{\epsilon, ab, ba, abab, abba, baab, baba, \cdots\}
\end{aligned}
$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Intersección de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$$

Ejemplo

Si $L_1 = \{ab, bc, bcc\}$ *y* $L_2 = \{a, bc, abb, ac\}$

entonces $L_1 \cap L_2 = \{bc\}$

Nota

No hay palabras o cadenas repetidas

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Intersección de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$$

Ejemplo

Si $L_1 = \{ab, bc, bcc\}$ *y* $L_2 = \{a, bc, abb, ac\}$

entonces $L_1 \cap L_2 = \{bc\}$

Nota

No hay palabras o cadenas repetidas

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Intersección de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 \cap L_2 = \{x | x \in L_1 \wedge x \in L_2\}$$

Ejemplo

Si $L_1 = \{ab, bc, bcc\}$ *y* $L_2 = \{a, bc, abb, ac\}$

entonces $L_1 \cap L_2 = \{bc\}$

Nota

No hay palabras o cadenas repetidas

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la intersección de lenguajes

- **Operación cerrada sobre Σ^*:**

$$\text{si } L_1, L_2 \subseteq \Sigma^* \text{ entonces } L_1 \cap L_2 \subseteq \Sigma^*$$

- **Asociativa:** $L_1 \cap (L_2 \cap L_3) = (L_1 \cap L_2) \cap L_3 = L_1 \cap L_2 \cap L_3$
- **Conmutativa:** $L_1 \cap L_2 = L_2 \cap L_1$
- **Existencia de elemento neutro:** Σ^*

$$L \cap \Sigma^* = \Sigma^* \cap L = L$$

- **Idempotente:** $L \cap L = L$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Diferencia de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 - L_2 = \{x | x \in L_1 \land x \notin L_2\}$$

Ejemplo

Si $L_1 = \{ab, bc, bcc\}$ *y* $L_2 = \{a, bc, abb, ac\}$

entonces $L_1 - L_2 = \{ab, bcc\}$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Diferencia de lenguajes formales)

Si $L_1, L_2 \subseteq \Sigma^*$ *entonces*

$$L_1 - L_2 = \{x | x \in L_1 \wedge x \notin L_2\}$$

Ejemplo

Si $L_1 = \{ab, bc, bcc\}$ *y* $L_2 = \{a, bc, abb, ac\}$

entonces $L_1 - L_2 = \{ab, bcc\}$

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la diferencia de lenguajes

- **Operación cerrada sobre Σ^*:**

  $$\text{si } L_1, L_2 \subseteq \Sigma^* \text{ entonces } L_1 - L_2 \subseteq \Sigma^*$$

- **No asociativa:** $(L_1 - L2) - L_3 \neq L_1 - (L_2 - L_3)$
- **No conmutativa:** $L_1 - L_2 \neq L_2 - L_1$
- **No existencia de elemento neutro**
- **No idempotente:** $L - L = \emptyset$

Especificación de componentes léxicos

Palabras y lenguajes formales

Definición (Complementación de un lenguaje formal)

Si $L \subseteq \Sigma^*$ *entonces*

$$\overline{L} = \Sigma^* - L = \{x | x \in \Sigma^* \wedge x \notin L\}$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Ejemplo

$Si\ \Sigma = \{a, b\}$

$Si\ L = \{\epsilon, a, ab, abb, abbb, \ldots\}$

$entonces\ \overline{L} = \{b, aa, ba, bb, aaa, aab, \ldots\}$

Ejemplo

$\overline{\emptyset} = \Sigma^*$

$\overline{\Sigma^*} = \emptyset$

Especificación de componentes léxicos

Palabras y lenguajes formales

Ejemplo

$$Si\ \Sigma = \{a, b\}$$

$$Si\ L = \{\epsilon, a, ab, abb, abbb, \ldots\}$$

$$entonces\ \overline{L} = \{b, aa, ba, bb, aaa, aab, \ldots\}$$

Ejemplo

$$\overline{\emptyset} = \Sigma^*$$

$$\overline{\Sigma^*} = \emptyset$$

Especificación de componentes léxicos

Palabras y lenguajes formales

Propiedades de la complementación de lenguajes

- **Operación cerrada sobre Σ^*:**

$$\text{si } L \subseteq \Sigma^* \text{ entonces } \overline{L} \subseteq \Sigma^*$$

- **Doble complementación** $\overline{\overline{L}} = L$

Contenido de la sección

Especificación de componentes léxicos

Expresiones Regulares

Definición (Expresión regular)

Expresiones regulares sobre $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$:

1. $\emptyset$ *es una expresión regular*
2. ϵ *es una expresión regular*
3. *Si* $\sigma \in \Sigma$ *entonces* σ *es una expresión regular*
4. *Si* α *y* β *son expresiones regulares entonces también son:*
   - a) $\alpha + \beta$
   - b) $\alpha \cdot \beta$
   - c) (α) *(o* (β)*)*
   - d) $\alpha^* = \sum_{i=0}^{\infty} \alpha^i = \alpha^0 + \alpha^1 + \alpha^2 + \cdots$

Especificación de componentes léxicos

Expresiones Regulares

Notas

- *La palabra vacía se puede representar por* ϵ *o* λ
- *La alternativa se puede representar por* $+$ *o por* $|$
- *El punto de la concatenación se suele omitir*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplos (Expresiones regulares sobre $\Sigma = \{0, 1\}$)

- $\emptyset$, ϵ, 0, 1
- $0 + \epsilon$, $\epsilon + 0$, 0ϵ, $\epsilon 0$
- $1 + \epsilon$, $\epsilon + 1$, 1ϵ, $\epsilon 1$
- $0 + 0$, $0 + 1$, $1 + 0$, $1 + 1$
- 00, 01, 10, 11
- 0^*, 1^*
- $(0 + 1)$, $(0 + 1)^*$, $0^*(0 + 1)1^*$

Especificación de componentes léxicos

Expresiones Regulares

Prioridad de los operadores de las expresiones regulares

+ Máxima prioridad ( )

*

.

- Mínima prioridad +

Especificación de componentes léxicos

Expresiones Regulares

Definición (Lenguaje denotado por una expresión regular)

1. $L(\emptyset) = \emptyset$
2. $L(\epsilon) = \{\epsilon\}$
3. *Si* $\sigma \in \Sigma$ *entonces* $L(\sigma) = \{\sigma\}$
4. *Si* α *y* β *son expresiones regulares sobre* Σ
   a) $L(\alpha + \beta) = L(\alpha) \cup L(\beta)$
   b) $L(\alpha \cdot \beta) = L(\alpha) \cdot L(\beta)$
   c) $L((\alpha)) = L(\alpha)$
   d) $L(\alpha^*) = L(\sum_{i=0}^{\infty} \alpha^i) = \bigcup_{i=0}^{\infty} L(\alpha^i) = \bigcup_{i=0}^{\infty} (L(\alpha))^i$

Especificación de componentes léxicos

Expresiones Regulares

Ejemplos (Lenguajes denotados 1 / 5)

Dado $\Sigma = \{0, 1\}$

- $L(0) = \{0\}$
- $L(0 + 1) = L(0) \cup L(1) = \{0\} \cup \{1\} = \{0, 1\}$
- $L(01) = L(0)L(1) = \{0\}\{1\} = \{01\}$

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Lenguaje denotado 2 / 5)

$$
\begin{aligned}
L(1^*) &= L(\sum_{i=0}^{\infty} 1^i) = \bigcup_{i=0}^{\infty} L(1^i) \\
&= \bigcup_{i=0}^{\infty} (L(1))^i = \bigcup_{i=0}^{\infty} \{1\}^i \\
&= \{\epsilon, 1, 11, 111, \cdots\}
\end{aligned}
$$

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Lenguaje denotado 2 / 5)

$$
\begin{aligned}
L(1^*) &= L(\textstyle\sum_{i=0}^{\infty} 1^i) = \bigcup_{i=0}^{\infty} L(1^i) \\
&= \bigcup_{i=0}^{\infty} (L(1))^i = \bigcup_{i=0}^{\infty} \{1\}^i \\
&= \{\epsilon, 1, 11, 111, \cdots\}
\end{aligned}
$$

Palabras compuestas por cero o más unos

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Lenguaje denotado 3 / 5)

$$
\begin{aligned}
L((0+1)1^*) &= L((0+1))L(1^*) \\
&= L(0+1)L(1^*) \\
&= \{0,1\}\{\epsilon, 1, 11, 111, \cdots\} \\
&= \{0, 01, 011, 0111, \cdots, 1, 11, 111, \cdots\}
\end{aligned}
$$

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Lenguaje denotado 3 / 5)

$$\begin{aligned} L((0+1)1^*) &= L((0+1))L(1^*) \\ &= L(0+1)L(1^*) \\ &= \{0,1\}\{\epsilon, 1, 11, 111, \cdots\} \\ &= \{0, 01, 011, 0111, \cdots, 1, 11, 111, \cdots\} \end{aligned}$$

Palabras que comienza por cero o por uno y van seguidas por cero o más unos

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Lenguaje denotado 4 / 5)

$$
\begin{aligned}
L(0^*(11)0^*) &= L(0^*)L((11))L(0^*) \\
&= \{\epsilon, 0, 00, 000, \cdots\}\{11\}\{\epsilon, 0, 00, 000, \cdots\} \\
&= \{11, 011, 0011, 00011, \cdots\}\{\epsilon, 0, 00, 000, \cdots\} \\
&= \{11, 0110, 00110, \cdots 00110, 001100, \cdots\}
\end{aligned}
$$

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Lenguaje denotado 4 / 5)

$$
\begin{aligned}
L(0^*(11)0^*) &= L(0^*)L((11))L(0^*) \\
&= \{\epsilon, 0, 00, 000, \cdots\}\{11\}\{\epsilon, 0, 00, 000, \cdots\} \\
&= \{11, 011, 0011, 00011, \cdots\}\{\epsilon, 0, 00, 000, \cdots\} \\
&= \{11, 0110, 00110, \cdots 00110, 001100, \cdots\}
\end{aligned}
$$

Palabras que contienen a la cadena 11 y comienzan y terminan por una secuencia de ceros, posiblemente nula

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Lenguaje denotado 5 / 5)

Dado $\Sigma = \{a, b, c, \cdots, z\}$

$$
\begin{aligned}
L(a + b + c + \cdots + z) &= L(a) \cup L(b) \cup L(c) \cup \cdots L(z) \\
&= \{a\} \cup \{b\} \cup \{c\} \cup \cdots \cup \{z\} \\
&= \{a, b, c, \cdots, z\}
\end{aligned}
$$

Especificación de componentes léxicos

Expresiones Regulares

Definición (Definición regular)

- *Identificador* que se asocia a una *expresión regular*
- *Puede ser utilizado para definir nuevas expresiones regulares*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplos

- $letra = (a + b + c + \cdots + z)$
- $cifra = (0 + 1 + \cdots + 9)$
- $guion = -$
- $subrayado = \_$

Especificación de componentes léxicos

Expresiones Regulares

Ejemplos (Identificadores de lenguajes de programación)

- **Pascal:**

$$letra\ (letra\ +\ cifra)^*$$

- **C:**

$$(subrayado\ +\ letra)\ (subrayado\ +\ letra\ +\ cifra)^*$$

- **Cobol:**

$$letra\ (letra\ +\ cifra\ +\ guion\ (letra\ +\ cifra))^*$$

Especificación de componentes léxicos

Expresiones Regulares

Ejemplos (Números en el Lenguaje C)

$$número = parte\_entera\ (\epsilon + parte\_decimal)$$

donde

$$parte\_entera = cifra\ cifra^*$$

$$parte\_decimal$$
$$= punto\ cifra^*(\epsilon + (E + e)(\epsilon + "-" + "+")cifra\ cifra^*)$$

Especificación de componentes léxicos

Expresiones Regulares

Nota (Abreviaturas)

- $\alpha? = \alpha + \epsilon = \epsilon + \alpha$
- $\alpha^{+} = \alpha\ \alpha^{*} = \alpha^{*}\alpha$

Ejemplo (Números en el Lenguaje C)

$$número = parte\_entera\ parte\_decimal?$$

Especificación de componentes léxicos

Expresiones Regulares

Nota (Abreviaturas)

- $\alpha? = \alpha + \epsilon = \epsilon + \alpha$
- $\alpha^{+} = \alpha\ \alpha^{*} = \alpha^{*}\alpha$

Ejemplo (Números en el Lenguaje C)

$$\textit{número} = \textit{parte\_entera}\ \textit{parte\_decimal}?$$

Especificación de componentes léxicos

Expresiones Regulares

Nota

Palabras claves: *existe una expresión regular para cada palabra clave.*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Palabras claves en C 1/3)

- **COMPONENTE LÉXICO**: *IF*
- **Expresión regular**: *if*
- **Paradigma**: *if*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Palabras claves en C 2/3)

- **COMPONENTE LÉXICO**: *WHILE*
- **Expresión regular**: *while*
- **Paradigma**: *while*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Palabras claves en C 3/3)

- **COMPONENTE LÉXICO**: *FOR*
- **Expresión regular**: *for*
- **Paradigma**: *for*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Palabras claves en FORTRAN 1/3)

- **COMPONENTE LÉXICO**: *DO*
- **Expresión regular**: *(D+d)(O+o)*
- **Paradigma**: *DO, Do, dO, do*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Palabras claves en FORTRAN 2/3)

- **COMPONENTE LÉXICO**: *FORMAT*
- **Expresión regular**: *(F+f)(O+o)(R+r)(M+m)(A+a)(T+t)*
- **Paradigma**: *FORMAT, ..., Format, ..., format*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Palabras claves en FORTRAN 3/3)

- **COMPONENTE LÉXICO**: *REAL*
- **Expresión regular**: *(R+r)(E+e)(A+a)(L+l)*
- **Paradigma**: *REAL*, . . . , *Real*, . . . *real*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Palabras claves en Pascal 1/3)

- **COMPONENTE LÉXICO**: *INTEGER*
- **Expresión regular**:
  $(I+i)(N+n)(T+t)(E+e)(G+g)(E+e)(R+r)$
- **Paradigma**: *INTEGER*, ..., *Integer*, ..., *integer*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Palabras claves en Pascal 2/3)

- **COMPONENTE LÉXICO**: *THEN*
- **Expresión regular**: $(T+t)(H+h)(E+e)(N+n)$
- **Paradigma**: *THEN*, ..., *Then*, ..., *then*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Palabras claves en Pascal 3/3)

- **COMPONENTE LÉXICO**: *VAR*
- **Expresión regular**: *(V+v)(A+a)(R+r)*
- **Paradigma**: *VAR*, ..., *Var*, ..., *var*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Operadores aritméticos en C)

COMPONENTE LÉXICO	**Expresión regular**	**Paradigma**
SUMA	+	+
RESTA	-	-
MULTIPLICACIÓN	*	*
DIVISIÓN	/	/
RESTO_DIVISIÓN_ENTERA	%	%

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Operadores aritméticos en FORTRAN)

COMPONENTE LÉXICO	Expresión regular	Paradigma
SUMA	$+$	$+$
RESTA	-	-
MULTIPLICACIÓN	*	*
DIVISIÓN	$/$	$/$
POTENCIA	**	**

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Operadores aritméticos en PASCAL)

COMPONENTE LÉXICO	Expresión regular	Paradigma
SUMA	+	+
RESTA	-	-
MULTIPLICACIÓN	*	*
DIVISIÓN	/	/
RESTO_DIVISIÓN_ENTERA	*mod*	*mod*
COCIENTE_DIVISIÓN_ENTERA	*div*	*div*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Operadores relacionales en C)

COMPONENTE LÉXICO	Expresión regular	Paradigma
MENOR_QUE	<	<
MENOR_IGUAL_QUE	<=	<=
MAYOR_QUE	>	>
MAYOR_IGUAL_QUE	>=	>=
IGUAL	==	==
DISTINTO	!=	!=

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Operadores relacionales en FORTRAN)

COMPONENTE LÉXICO	Expresión regular	Paradigma
MENOR_QUE	*.(L+l)(T+t).*	*.LT., ...lt.*
MENOR_IGUAL_QUE	*.(L+l)(E+e).*	*.LE., ...le.*
MAYOR_QUE	*.(G+g)(T+t).*	*.GT., ...gt.*
MAYOR_IGUAL_QUE	*.(G+g)(E+e).*	*.GE., ...ge.*
IGUAL	*.(E+e)(Q+q).*	*.EQ., ...eq.*
DISTINTO	*.(N+n)(E+e).*	*.NE., ...ne.*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Operadores relacionales en Pascal)

COMPONENTE LÉXICO	Expresión regular	Paradigma
MENOR_QUE	$<$	$<$
MENOR_IGUAL_QUE	$<=$	$<=$
MAYOR_QUE	$>$	$>$
MAYOR_IGUAL_QUE	$>=$	$>=$
IGUAL	$=$	$=$
DISTINTO	$<>$	$<>$

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Operadores lógicos en C)

COMPONENTE LÉXICO	Expresión regular	Paradigma
NEGACIÓN_LÓGICA	!	!
CONJUNCIÓN_LÓGICA	&&	&&
DISYUNCIÓN_LÓGICA	||	||

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Operadores lógicos en FORTRAN)

COMPONENTE LÉXICO	**Expresión regular**	**Paradigma**
NEGACIÓN_LÓGICA	*.(N+n)(O+o)(T+t).*	*.NOT, ...not.*
CONJUNCIÓN_LÓGICA	*.(A+a)(N+n)(D+d).*	*.AND.,... .and.*
DISYUNCIÓN_LÓGICA	*.(O+o)(R+r).*	*.OR., ...or.*

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo (Operadores lógicos en PASCAL)

COMPONENTE LÉXICO	**Expresión regular**	**Paradigma**
NEGACIÓN_LÓGICA	*(N+n)(O+o)(T+t)*	*NOT, . . . not*
CONJUNCIÓN_LÓGICA	*(A+a)(N+n)(D+d)*	*AND, . . . and*
DISYUNCIÓN_LÓGICA	*(O+o)(R+r)*	*OR, . . . or*

Especificación de componentes léxicos

Expresiones Regulares

Nota

- *El analizador sintáctico sólo necesita saber cuál es el componente léxico reconocido*

- *No necesita saber cómo es dicho componente léxico.*

Especificación de componentes léxicos

Expresiones Regulares

Definición (Equivalencia de expresiones regulares)

α *y* β *son* **equivalentes** *si y sólo si denotan el mismo lenguaje:* $L(\alpha) = L(\beta)$

$$\alpha \equiv \beta \Longleftrightarrow L(\alpha) = L(\beta)$$

Especificación de componentes léxicos

Expresiones Regulares

Ejemplo

Se verifica que

$$aa^* \equiv a^*a$$

porque

$$\begin{aligned} L(aa^*) &= L(a)L(a^*) \\ &= \{a\}\{\epsilon, a, aa, \cdots\} \\ &= \{a, aa, aaa, \cdots\} \\ &= \{\epsilon, a, aa, \cdots\}\{a\} \\ &= L(a^*)L(a) \\ &= L(a^*a) \end{aligned}$$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

1.- **Disyunción idempotente**: $\alpha + \alpha = \alpha$

2.- **Disyunción asociativa**: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$

3.- **Disyunción conmutativa**: $\alpha + \beta = \beta + \alpha$

4.- **Concatenación asociativa**: $\alpha\ (\beta\ \gamma) = (\alpha\ \beta)\ \gamma$

5.- **Concatenación no conmutativa**: $\alpha\ \beta \neq \beta\ \alpha$

6.- **Distributiva**: $\alpha\ (\beta\ +\ \gamma) = \alpha\ \beta\ +\ \alpha\ \gamma$

7.- **Elemento neutro de la disyunción**: $\alpha + \emptyset = \emptyset + \alpha = \alpha$

8.- **Elemento neutro de la concatenación**: $\alpha\epsilon = \epsilon\alpha = \alpha$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

1.- **Disyunción idempotente**: $\alpha + \alpha = \alpha$

2.- **Disyunción asociativa**: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$

3.- **Disyunción conmutativa**: $\alpha + \beta = \beta + \alpha$

4.- **Concatenación asociativa**: $\alpha\ (\beta\ \gamma) = (\alpha\ \beta)\ \gamma$

5.- **Concatenación no conmutativa**: $\alpha\ \beta \neq \beta\ \alpha$

6.- **Distributiva**: $\alpha\ (\beta\ +\ \gamma) = \alpha\ \beta\ +\ \alpha\ \gamma$

7.- **Elemento neutro de la disyunción**: $\alpha + \emptyset = \emptyset + \alpha = \alpha$

8.- **Elemento neutro de la concatenación**: $\alpha\epsilon = \epsilon\alpha = \alpha$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

1.- **Disyunción idempotente**: $\alpha + \alpha = \alpha$

2.- **Disyunción asociativa**: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$

3.- **Disyunción conmutativa**: $\alpha + \beta = \beta + \alpha$

4.- **Concatenación asociativa**: $\alpha\ (\beta\ \gamma) = (\alpha\ \beta)\ \gamma$

5.- **Concatenación no conmutativa**: $\alpha\ \beta \neq \beta\ \alpha$

6.- **Distributiva**: $\alpha\ (\beta\ +\ \gamma) = \alpha\ \beta\ +\ \alpha\ \gamma$

7.- **Elemento neutro de la disyunción**: $\alpha + \emptyset = \emptyset + \alpha = \alpha$

8.- **Elemento neutro de la concatenación**: $\alpha\epsilon = \epsilon\alpha = \alpha$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

1.- **Disyunción idempotente**: $\alpha + \alpha = \alpha$

2.- **Disyunción asociativa**: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$

3.- **Disyunción conmutativa**: $\alpha + \beta = \beta + \alpha$

4.- **Concatenación asociativa**: $\alpha\ (\beta\ \gamma) = (\alpha\ \beta)\ \gamma$

5.- **Concatenación no conmutativa**: $\alpha\ \beta \neq \beta\ \alpha$

6.- **Distributiva**: $\alpha\ (\beta\ +\ \gamma) = \alpha\ \beta\ +\ \alpha\ \gamma$

7.- **Elemento neutro de la disyunción**: $\alpha + \emptyset = \emptyset + \alpha = \alpha$

8.- **Elemento neutro de la concatenación**: $\alpha\epsilon = \epsilon\alpha = \alpha$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

1.- **Disyunción idempotente**: $\alpha + \alpha = \alpha$

2.- **Disyunción asociativa**: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$

3.- **Disyunción conmutativa**: $\alpha + \beta = \beta + \alpha$

4.- **Concatenación asociativa**: $\alpha\ (\beta\ \gamma) = (\alpha\ \beta)\ \gamma$

5.- **Concatenación no conmutativa**: $\alpha\ \beta \neq \beta\ \alpha$

6.- **Distributiva**: $\alpha\ (\beta\ +\ \gamma) = \alpha\ \beta\ +\ \alpha\ \gamma$

7.- **Elemento neutro de la disyunción**: $\alpha + \emptyset = \emptyset + \alpha = \alpha$

8.- **Elemento neutro de la concatenación**: $\alpha\epsilon = \epsilon\alpha = \alpha$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

1.- **Disyunción idempotente**: $\alpha + \alpha = \alpha$

2.- **Disyunción asociativa**: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$

3.- **Disyunción conmutativa**: $\alpha + \beta = \beta + \alpha$

4.- **Concatenación asociativa**: $\alpha\ (\beta\ \gamma) = (\alpha\ \beta)\ \gamma$

5.- **Concatenación no conmutativa**: $\alpha\ \beta \neq \beta\ \alpha$

6.- **Distributiva**: $\alpha\ (\beta\ +\ \gamma) = \alpha\ \beta\ +\ \alpha\ \gamma$

7.- **Elemento neutro de la disyunción**: $\alpha + \emptyset = \emptyset + \alpha = \alpha$

8.- **Elemento neutro de la concatenación**: $\alpha\epsilon = \epsilon\alpha = \alpha$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

1.- **Disyunción idempotente**: $\alpha + \alpha = \alpha$

2.- **Disyunción asociativa**: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$

3.- **Disyunción conmutativa**: $\alpha + \beta = \beta + \alpha$

4.- **Concatenación asociativa**: $\alpha\ (\beta\ \gamma) = (\alpha\ \beta)\ \gamma$

5.- **Concatenación no conmutativa**: $\alpha\ \beta \neq \beta\ \alpha$

6.- **Distributiva**: $\alpha\ (\beta\ +\ \gamma) = \alpha\ \beta\ +\ \alpha\ \gamma$

7.- **Elemento neutro de la disyunción**: $\alpha + \emptyset = \emptyset + \alpha = \alpha$

8.- **Elemento neutro de la concatenación**: $\alpha\epsilon = \epsilon\alpha = \alpha$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

1.- **Disyunción idempotente**: $\alpha + \alpha = \alpha$

2.- **Disyunción asociativa**: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$

3.- **Disyunción conmutativa**: $\alpha + \beta = \beta + \alpha$

4.- **Concatenación asociativa**: $\alpha\ (\beta\ \gamma) = (\alpha\ \beta)\ \gamma$

5.- **Concatenación no conmutativa**: $\alpha\ \beta \neq \beta\ \alpha$

6.- **Distributiva**: $\alpha\ (\beta\ +\ \gamma) = \alpha\ \beta\ +\ \alpha\ \gamma$

7.- **Elemento neutro de la disyunción**: $\alpha + \emptyset = \emptyset + \alpha = \alpha$

8.- **Elemento neutro de la concatenación**: $\alpha\epsilon = \epsilon\alpha = \alpha$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

9.- $\alpha\, \emptyset = \emptyset\, \alpha = \emptyset$

10.- $\epsilon^* = \epsilon$

11.- $\emptyset^* = \epsilon$

12.- $\alpha^*\, \alpha^* = \alpha^*$

13.- $\alpha\, \alpha^* = \alpha^*\, \alpha = \alpha^+$

14.- $(\alpha^*)^* = \alpha^*$

15.- $\alpha^* = \epsilon + \alpha\, \alpha^*$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

9.- $\alpha\,\emptyset = \emptyset\,\alpha = \emptyset$

10.- $\epsilon^* = \epsilon$

11.- $\emptyset^* = \epsilon$

12.- $\alpha^*\,\alpha^* = \alpha^*$

13.- $\alpha\,\alpha^* = \alpha^*\,\alpha = \alpha^+$

14.- $(\alpha^*)^* = \alpha^*$

15.- $\alpha^* = \epsilon + \alpha\,\alpha^*$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

9.- $\alpha\ \emptyset\ =\ \emptyset\ \alpha\ =\ \emptyset$

10.- $\epsilon^*\ =\ \epsilon$

11.- $\emptyset^*\ =\ \epsilon$

12.- $\alpha^*\ \alpha^*\ =\ \alpha^*$

13.- $\alpha\ \alpha^*\ =\ \alpha^*\ \alpha\ =\ \alpha^+$

14.- $(\alpha^*)^*\ =\ \alpha^*$

15.- $\alpha^*\ =\ \epsilon\ +\ \alpha\ \alpha^*$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

9.- $\alpha\, \emptyset = \emptyset\, \alpha = \emptyset$

10.- $\epsilon^* = \epsilon$

11.- $\emptyset^* = \epsilon$

$$\begin{array}{rclcl} L(\emptyset^*) & = & (L(\emptyset))^* & = & (\emptyset)^* \\ & = & \bigcup_{i=0}^{\infty} \emptyset^i & = & \emptyset^0 \cup \emptyset^1 \cup \emptyset^2 \cdots \\ & = & \{\epsilon\} \cup \emptyset \cup \emptyset \cdots & & \\ & = & \{\epsilon\} & = & L(\epsilon) \end{array}$$

12.- $\alpha^*\, \alpha^* = \alpha^*$

13.- $\alpha\, \alpha^* = \alpha^*\, \alpha = \alpha^+$

14.- $(\alpha^*)^* = \alpha^*$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

9.- $\alpha \, \emptyset = \emptyset \, \alpha = \emptyset$

10.- $\epsilon^* = \epsilon$

11.- $\emptyset^* = \epsilon$

12.- $\alpha^* \, \alpha^* = \alpha^*$

13.- $\alpha \, \alpha^* = \alpha^* \, \alpha = \alpha^+$

14.- $(\alpha^*)^* = \alpha^*$

15.- $\alpha^* = \epsilon + \alpha \, \alpha^*$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

9.- $\alpha \, \emptyset = \emptyset \, \alpha = \emptyset$

10.- $\epsilon^* = \epsilon$

11.- $\emptyset^* = \epsilon$

12.- $\alpha^* \, \alpha^* = \alpha^*$

13.- $\alpha \, \alpha^* = \alpha^* \, \alpha = \alpha^+$

14.- $(\alpha^*)^* = \alpha^*$

15.- $\alpha^* = \epsilon + \alpha \, \alpha^*$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

9.- $\alpha\ \emptyset\ =\ \emptyset\ \alpha\ =\ \emptyset$

10.- $\epsilon^*\ =\ \epsilon$

11.- $\emptyset^*\ =\ \epsilon$

12.- $\alpha^*\ \alpha^*\ =\ \alpha^*$

13.- $\alpha\ \alpha^*\ =\ \alpha^*\ \alpha\ =\ \alpha^+$

14.- $(\alpha^*)^*\ =\ \alpha^*$

15.- $\alpha^*\ =\ \epsilon\ +\ \alpha\ \alpha^*$

Especificación de componentes léxicos

Expresiones Regulares

Propiedades de las expresiones regulares

9.- $\alpha \, \emptyset = \emptyset \, \alpha = \emptyset$

10.- $\epsilon^* = \epsilon$

11.- $\emptyset^* = \epsilon$

12.- $\alpha^* \, \alpha^* = \alpha^*$

13.- $\alpha \, \alpha^* = \alpha^* \, \alpha = \alpha^+$

14.- $(\alpha^*)^* = \alpha^*$

15.- $\alpha^* = \epsilon + \alpha \, \alpha^*$

Especificación de componentes léxicos

Expresiones Regulares

Capacidad de las expresiones regulares

- Denotan los componentes léxicos
- Pueden denotar
  - Un número **fijo** de repeticiones: *aaaaa*
  - Un número **arbitrario** de repeticiones: a^*
  - Repeticiones no coordinadas

Ejemplo

$$
\begin{aligned}
L_1 &= \{a^i\ b^j \mid i, j \geq 0\} \\
&= \{\epsilon, a, aa, aaa, \cdots b, ab, aab, \cdots\} \\
&= \{\epsilon, a, aa, aaa, \cdots\}\{\epsilon, b, bb, bbb\} \\
&= L(a^* b^*)
\end{aligned}
$$

Especificación de componentes léxicos

Expresiones Regulares

Capacidad de las expresiones regulares

- Denotan los componentes léxicos
- Pueden denotar
  - Un número **fijo** de repeticiones: $aaaaa$
  - Un número **arbitrario** de repeticiones: a^*
  - Repeticiones no coordinadas

Ejemplo

$$
\begin{aligned}
L_1 &= \{a^i\ b^j | i, j \geq 0\} \\
&= \{\epsilon, a, aa, aaa, \cdots b, ab, aab, \cdots\} \\
&= \{\epsilon, a, aa, aaa, \cdots\}\{\epsilon, b, bb, bbb\} \\
&= L(a^* b^*)
\end{aligned}
$$

Especificación de componentes léxicos

Expresiones Regulares

Nota (Limitaciones de las expresiones regulares)

- *No pueden denotar características sintácticas*
- *No pueden denotar repeticiones coordinadas*

Especificación de componentes léxicos

Expresiones Regulares

Nota (Limitaciones de las expresiones regulares)

- *No pueden denotar características sintácticas*
- *No pueden denotar repeticiones coordinadas*

Ejemplo

Lenguaje que no puede ser denotado por una expresión regular

$$L_2 = \{a^i \; b^i | i \geq 0\} = \{\epsilon, ab, aabb, aaabbb, \cdots\}$$

L_2 *no es un lenguaje regular.*

Especificación de componentes léxicos

Expresiones Regulares

Nota (Limitaciones de las expresiones regulares)

- *No pueden denotar características sintácticas*
- *No pueden denotar repeticiones coordinadas*

Ejemplo

$L_2 = \{a^i\ b^i | i \geq 0\}$ *representa a* **muchas estructuras sintácticas** *de los lenguajes de programación:*

+ *Balanceo de paréntesis, llaves o corchetes.*
+ *Paso de parámetros*
+ *Etc.*

Contenido del tema

Contenido de la sección

Reconocimiento de componentes léxicos

Autómatas finitos

COMPONENTES LÉXICOS

Reconocimiento de componentes léxicos

Autómatas finitos

COMPONENTES LÉXICOS

$\downarrow$

EXPRESIONES REGULARES

Reconocimiento de componentes léxicos

Autómatas finitos

COMPONENTES LÉXICOS

↓

EXPRESIONES REGULARES

↓

AFN

Reconocimiento de componentes léxicos

Autómatas finitos

$$
\begin{array}{c}
\text{COMPONENTES LÉXICOS} \\
\downarrow \\
\text{EXPRESIONES REGULARES} \\
\downarrow \\
\text{AFN} \\
\downarrow \\
\text{AFD}
\end{array}
$$

Reconocimiento de componentes léxicos

Autómatas finitos

COMPONENTES LÉXICOS

↓

EXPRESIONES REGULARES

↓

AFN

↓

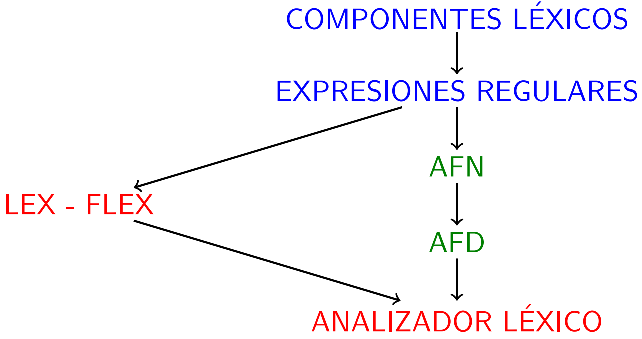
AFD

↓

ANALIZADOR LÉXICO

Reconocimiento de componentes léxicos

Autómatas finitos



Reconocimiento de componentes léxicos

Autómatas finitos

- **Expresiones regulares**: denotan componentes léxicos.
- **Autómatas finitos**: reconocen componentes léxicos:
  - Autómata finito no determinista: AFN
  - Autómata finito determinista: AFD
- Generación de autómatas finitos a partir de expresiones regulares:

Paso 1.- Algoritmo de **Thompson**: genera un AFN a partir de una expresión regular

Paso 2.- Algoritmo de **Construcción de subconjuntos**: genera un AFD a partir de un AFN.

Reconocimiento de componentes léxicos

Autómatas finitos

- **Expresiones regulares**: denotan componentes léxicos.
- **Autómatas finitos**: reconocen componentes léxicos:
  + Autómata finito no determinista: AFN
  + Autómata finito determinista: AFD
- Generación de autómatas finitos a partir de expresiones regulares:

Paso 1.- Algoritmo de **Thompson**: genera un AFN a partir de una expresión regular

Paso 2.- Algoritmo de **Construcción de subconjuntos**: genera un AFD a partir de un AFN.

Reconocimiento de componentes léxicos

Autómatas finitos

- **Expresiones regulares**: denotan componentes léxicos.
- **Autómatas finitos**: reconocen componentes léxicos:
  + Autómata finito no determinista: AFN
  + Autómata finito determinista: AFD
- Generación de autómatas finitos a partir de expresiones regulares:

Paso 1.- Algoritmo de **Thompson**: genera un AFN a partir de una expresión regular

Paso 2.- Algoritmo de **Construcción de subconjuntos**: genera un AFD a partir de un AFN.

Contenido de la sección

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

- Descripción general
- Definición formal
- Función de transición para palabras
- Representación gráfica
- Lenguaje reconocido por un AFD

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

- Descripción general
- Definición formal
- Función de transición para palabras
- Representación gráfica
- Lenguaje reconocido por un AFD

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Definición (Autómata finito determinista: AFD)

- *Dispositivo formal que permite reconocer si una palabra pertenece o no a un lenguaje regular.*
- *También se denomina "máquina reconocedora o aceptadora"*

Reconocimiento de componentes léxicos

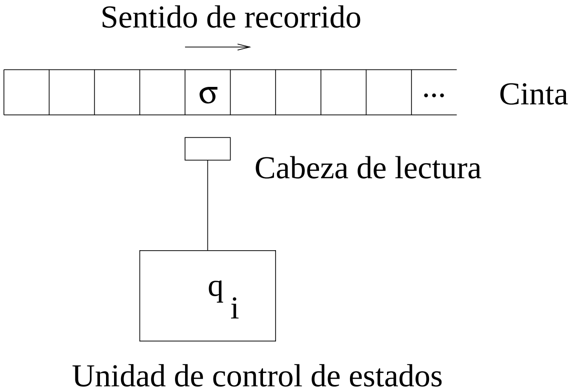
Autómatas finitos deterministas: AFD

Componentes de un AFD

- **Cinta de lectura**:
  - Dividida en celdas.
  - Infinita hacia la derecha.
- **Cabeza de lectura**:
  - Lee el símbolo actual de la cinta.
  - Sólo se puede mover hacia la derecha.
- **Alfabeto de la cinta**
- **Unidad de control de estados**: indica el estado actual.

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD



Componentes básicos de un autómata finito determinista.

Reconocimiento de componentes léxicos

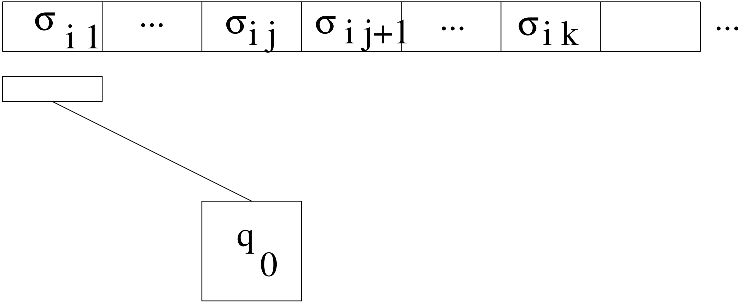
Autómatas finitos deterministas: AFD

Pasos del AFD para reconocer a $x = \sigma_{i_1} \ldots \sigma_{i_j} \sigma_{i_{j+1}} \ldots \sigma_{i_k} \in \Sigma^*$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Pasos del AFD para reconocer a $x = \sigma_{i_1} \ldots \sigma_{i_j} \sigma_{i_{j+1}} \ldots \sigma_{i_k} \in \Sigma^*$

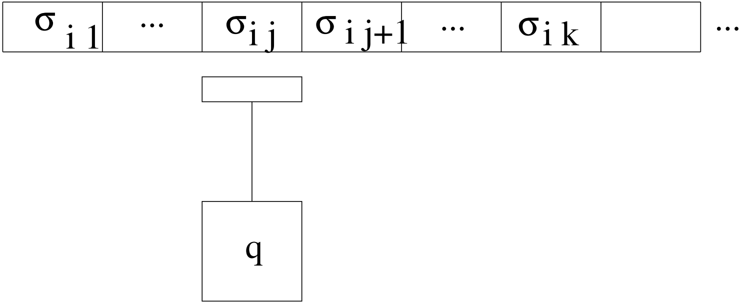


Configuración inicial

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Pasos del AFD para reconocer a $x = \sigma_{i_1} \ldots \sigma_{i_j} \sigma_{i_{j+1}} \ldots \sigma_{i_k} \in \Sigma^*$

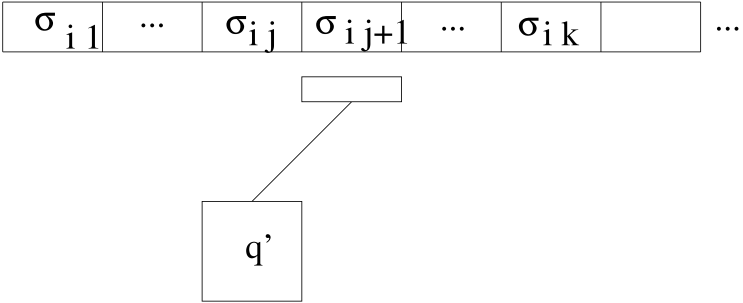


Transición: situación anterior

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Pasos del AFD para reconocer a $x = \sigma_{i_1} \ldots \sigma_{i_j} \sigma_{i_{j+1}} \ldots \sigma_{i_k} \in \Sigma^*$

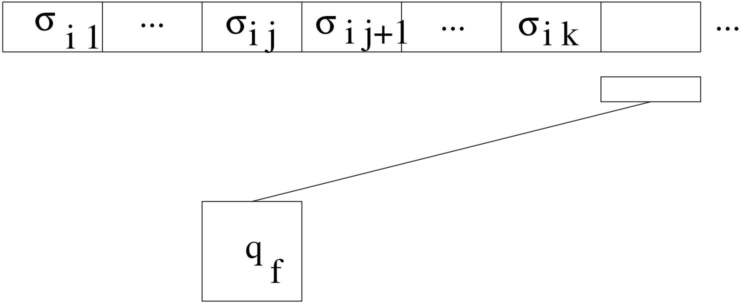


Transición: situación posterior

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Pasos del AFD para reconocer a $x = \sigma_{i_1} \ldots \sigma_{i_j} \sigma_{i_{j+1}} \ldots \sigma_{i_k} \in \Sigma^*$



Configuración final

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

- Descripción general
- Definición formal
- Función de transición para palabras
- Representación gráfica
- Lenguaje reconocido por un AFD

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Definición (Autómata finito determinista)

$$A = (Q, \Sigma, \delta, q_0, F)$$

donde

- Q: *conjunto finito de* **estados**
- Σ: **alfabeto** *de símbolos de entrada*
- δ: **función de transición** *entre estados:*

$$\delta : Q \times \Sigma \longrightarrow Q$$
$$\delta(q, \sigma) = q'$$

- $q_0 \in Q$: *estado* **inicial**
- $F \subseteq Q$: *conjunto de estados* **finales**

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (AFD que reconoce identificadores de COBOL)

	δ	l	d	g
$\rightarrow$	q_0	q_1	q_3	q_3
$\leftarrow$	q_1	q_1	q_1	q_2
	q_2	q_1	q_1	q_3
	q_3	q_3	q_3	q_3

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (AFD que reconoce identificadores de COBOL)

Los componentes del autómata son:

- $Q = \{q_0, q_1, q_2, q_3\}$
- $F = \{q_1\}$
- *El símbolo "$\rightarrow$" indica el estado inicial.*
- *El símbolo "$\leftarrow$" indica los estados finales.*
- $\Sigma = \{l, d, g\}$ *donde:* l = *letra,* d = *dígito y* g = *guion.*

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

- Descripción general
- Definición formal
- Función de transición para palabras
- Representación gráfica
- Lenguaje reconocido por un AFD

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Definición (Función de transición para palabras)

$$\hat{\delta} : Q \times \Sigma^* \longrightarrow Q$$

$$\begin{array}{lcl} \hat{\delta}(q, \epsilon) & = & q \in Q \\ \hat{\delta}(q, x\sigma) & = & \delta(\hat{\delta}(q, x), \sigma) \in Q \quad \forall x \in \Sigma^* \wedge \sigma \in \Sigma \end{array}$$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Notas

- *$\hat{\delta}$ y δ coinciden sobre símbolos de Σ:*

$$\hat{\delta}(q,\sigma) = \hat{\delta}(q,\epsilon\cdot\sigma) = \delta(\hat{\delta}(q,\epsilon),\sigma) = \delta(q,\sigma) \quad \forall\sigma\in\Sigma$$

- $\hat{\delta}(q,xy) = \hat{\delta}(\hat{\delta}(q,x),y) \quad \forall x,y\in\Sigma^*$

- $\hat{\delta}(q,\sigma x) = \hat{\delta}(\hat{\delta}(q,\sigma),x) = \hat{\delta}(\delta(q,\sigma),x)$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo

$$
\begin{aligned}
\hat{\delta}(q_0, x) &= \hat{\delta}(q_0, llgd) = \hat{\delta}(\delta(q_0, l), lgd) \\
&= \hat{\delta}(q_1, lgd) = \hat{\delta}(\delta(q_1, l), gd) \\
&= \hat{\delta}(q_1, gd) = \hat{\delta}(\delta(q_1, g), d) \\
&= \hat{\delta}(q_2, d) = \delta(q_2, d) \\
&= q_1 \in F
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Nota

$x = llgd$ *es* **reconocida** *por el AFD porque* q_1 *es un estado* **final***.*

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Nota

Existe una **notación más simple** *para el reconocimiento de un AFD.*

Ejemplo

$$
\begin{aligned}
(q_0, llgd) &\vdash (q_1, lgd)\\
&\vdash (q_1, gd)\\
&\vdash (q_2, d)\\
&\vdash (q_1, \epsilon)
\end{aligned}
$$

o simplemente $(q_0, llgd) \vdash^* (q_1, \epsilon)$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Nota

Existe una **notación más simple** *para el reconocimiento de un AFD.*

Ejemplo

$$
\begin{array}{rcl}
(q_0, llgd) & \vdash & (q_1, lgd) \\
 & \vdash & (q_1, gd) \\
 & \vdash & (q_2, d) \\
 & \vdash & (q_1, \epsilon)
\end{array}
$$

o simplemente $(q_0, llgd) \vdash^* (q_1, \epsilon)$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Nota

Existe una **notación más simple** *para el reconocimiento de un AFD.*

Ejemplo

$$\begin{aligned}(q_0, llgd) &\vdash (q_1, lgd)\\ &\vdash (q_1, gd)\\ &\vdash (q_2, d)\\ &\vdash (q_1, \epsilon)\end{aligned}$$

o simplemente $(q_0, llgd) \vdash^* (q_1, \epsilon)$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

- Descripción general
- Definición formal
- Representación gráfica
- Lenguaje reconocido por un AFD

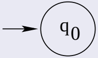
Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Definición (Representación gráfica de un AFD)

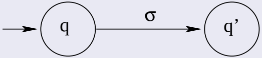
Grafo dirigido:

- *Número de nodos* $=$ *cardinal*(Q).
- *Etiqueta de cada nodo* $\in Q$.
- *Estado inicial:*



- *Estados finales:*

- *Si* $\delta(q, \sigma) = q'$ *entonces*

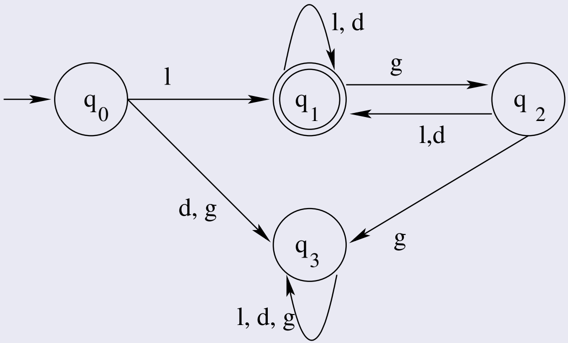


- *Se agrupan las aristas que enlazan los mismos estados.*

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (AFD que reconoce identificadores de COBOL)



Representación gráfica de un AFD.

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

- Descripción general
- Definición formal
- Representación gráfica
- Lenguaje reconocido por un AFD

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Definición (Lenguaje reconocido por un AFD)

$$L(A) = \{x | x \in \Sigma^* \wedge \hat{\delta}(q_0, x) \in F\}$$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Notas

- *Si* $F = Q$ *entonces* $L(A) = \Sigma^*$
- *Si* $F = \emptyset$ *entonces* $L(A) = \emptyset$
- $q_0 \in F$ *si y sólo si* $\epsilon \in L(A)$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo

$$L(A) = L(l(l + d + g(l + d))^*)$$

Lenguaje reconocido por un AFD que reconoce identificadores de COBOL

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (Lenguaje reconocido por un AFD)

Función de transición

	δ	a	b
$\rightarrow$ $\leftarrow$	q_0	q_0	q_1
$\leftarrow$	q_1	q_2	q_1
	q_2	q_2	q_2

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (Lenguaje reconocido por un AFD)

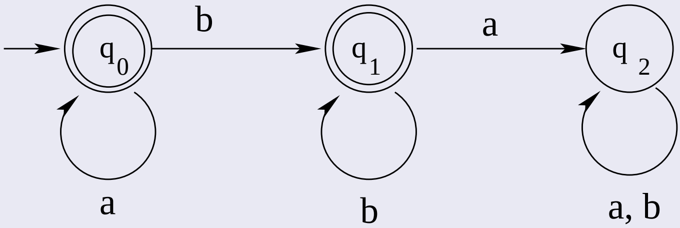
Los componentes del autómata son:

- $Q = \{q_0, q_1, q_2\}$
- $F = \{q_0, q_1\}$
- $\Sigma = \{a, b\}$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (Lenguaje reconocido por un AFD)



$\underline{a \cdots a}$ $\underline{b}$ $\underline{b \cdots b}$

$$L(A) = L(a^* b^*)$$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Nota

El estado q_2 es inútil o superfluo

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (Lenguaje reconocido por un AFD)

$$
\begin{aligned}
\hat{\delta}(q_0, aabb) &= \hat{\delta}(\delta(q_0, a), abb) \\
&= \hat{\delta}(\delta(q_0, a), bb) \\
&= \hat{\delta}(\delta(q_0, b), b) \\
&= \hat{\delta}(q_1, b) \\
&= \delta(q_1, b) \\
&= q_1 \in F
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (Lenguaje reconocido por un AFD)

$$\begin{array}{rcl} (q_0, aabb) & \vdash & (q_0, abb) \\ & \vdash & (q_0, bb) \\ & \vdash & (q_1, b) \\ & \vdash & (q_1, \epsilon) \end{array}$$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (AFD que reconoce identificadores de C)

Función de transición

	δ	*letra*	*subrayado*	*dígito*
$\rightarrow$	q_0	q_1	q_1	q_2
$\leftarrow$	q_1	q_1	q_1	q_1
	q_2	q_2	q_2	q_2

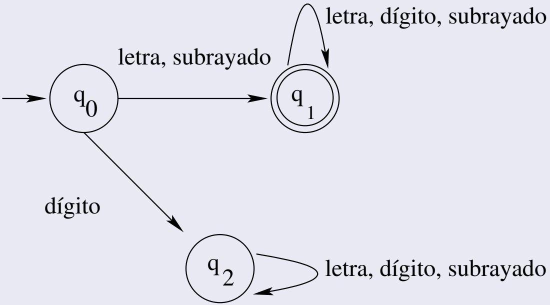
Nota

El estado q_2 es superfluo

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (AFD que reconoce identificadores de C)

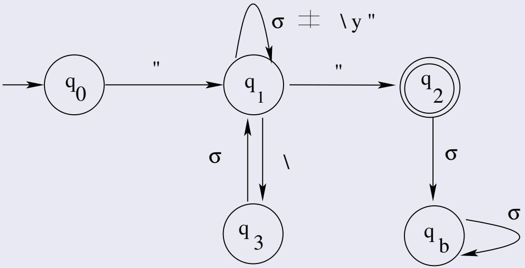


$$L(A) = L((letra + subrayado)(letra + subrayado + dígito)^*)$$

Reconocimiento de componentes léxicos

Autómatas finitos deterministas: AFD

Ejemplo (AFD que reconoce cadenas de caracteres)



$$L(A) = L(''(letra + \cdots)(BARRA'' + letra + \cdots)^{*}'')$$

Contenido de la sección

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

- Descripción general
- Definición formal
- Representación gráfica
- Función de transición para palabras
- Lenguaje reconocido por un AFN
- Equivalencia entre AFN y AFD
- Equivalencia entre expresiones regulares y autómatas finitos

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

- Descripción general
- Definición formal
- Representación gráfica
- Función de transición para palabras
- Lenguaje reconocido por un AFN
- Equivalencia entre AFN y AFD
- Equivalencia entre expresiones regulares y autómatas finitos

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Autómata finito NO determinista: AFN)

- *Un autómota finito es* **no** *determinista si posee alguna de las siguientes transiciones:*
  - **Transición** ϵ*: no lee el símbolo actual pero cambia de estado*
  - **Transición múltiple***: puede cambiar a más de un estado.*
- *Estos tipos de transiciones no son excluyentes.*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

- Descripción general
- Definición formal
- Representación gráfica
- Función de transición para palabras
- Lenguaje reconocido por un AFN
- Equivalencia entre AFN y AFD
- Equivalencia entre expresiones regulares y autómatas finitos

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Autómata finito NO determinista: AFN)

$$A = (Q, \Sigma, \delta, q_0, F)$$

- Q: *conjunto finito de estados*
- Σ: *alfabeto de símbolos de entrada*
- δ: *función de transición entre estados:*
$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \longrightarrow \mathcal{P}(Q)$$
$$\delta(q, \epsilon) \subseteq Q$$
$$\delta(q, \sigma) \subseteq Q$$
- $q_0 \in Q$: *estado inicial*
- $F \subseteq Q$: *conjunto de estados finales*
- $\mathcal{P}(Q)$: *conjunto de las partes de* Q

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Transición (ϵ) trivial)

$$q \in \delta(q, \epsilon) \quad \forall q \in Q$$

Nota

Las transiciones triviales siempre existen y se suponen tácitamente por defecto.

Definición (Transición (ϵ) no trivial)

$$q' \in \delta(q, \epsilon) \quad \wedge \quad q \neq q'$$

Nota

Si existen, estas transiciones se han de indicar expresamente.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Transición (ϵ) trivial)

$$q \in \delta(q, \epsilon) \quad \forall q \in Q$$

Nota

Las transiciones triviales siempre existen y se suponen tácitamente por defecto.

Definición (Transición (ϵ) no trivial)

$$q' \in \delta(q, \epsilon) \quad \wedge \quad q \neq q'$$

Nota

Si existen, estas transiciones se han de indicar expresamente.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Transición (ϵ) **trivial**)

$$q \in \delta(q, \epsilon) \quad \forall q \in Q$$

Nota

Las transiciones triviales siempre existen y se suponen tácitamente por defecto.

Definición (Transición (ϵ) **no** trivial)

$$q' \in \delta(q, \epsilon) \quad \wedge \quad q \neq q'$$

Nota

Si existen, estas transiciones se han de indicar expresamente.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Transición (ϵ) **trivial**)

$$q \in \delta(q, \epsilon) \quad \forall q \in Q$$

Nota

Las transiciones triviales siempre existen y se suponen tácitamente por defecto.

Definición (Transición (ϵ) **no** trivial)

$$q' \in \delta(q, \epsilon) \quad \wedge \quad q \neq q'$$

Nota

Si existen, estas transiciones se han de indicar expresamente.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo

	δ	a	b	ϵ
$\rightarrow$	q_0	$\{q_1\}$	$\emptyset$	$\{q_0, q_1\}$
	q_1	$\emptyset$	$\{q_3, q_4\}$	$\{q_1, q_2\}$
	q_2	$\emptyset$	$\{q_5\}$	$\{q_2\}$
	q_3	$\{q_2\}$	$\emptyset$	$\{q_3\}$
	q_4	$\{q_4, q_5\}$	$\emptyset$	$\{q_2, q_4\}$
$\leftarrow$	q_5	$\emptyset$	$\{q_5\}$	$\{q_3, q_5\}$

Las transiciones triviales se pueden omitir.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Nota

Un AFD puede considerarse un caso especial de AFN:

- *No existen transiciones-ϵ no triviales.*
- *y no existen transiciones múltiples, es decir,*

$$\delta(q, \sigma) = \{q'\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

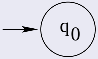
- Descripción general
- Definición formal
- Representación gráfica
- Función de transición para palabras
- Lenguaje reconocido por un AFN
- Equivalencia entre AFN y AFD
- Equivalencia entre expresiones regulares y autómatas finitos

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

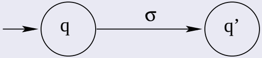
Definición (AFN: representación gráfica)

- *Número de nodos del grafo* $=$ *cardinal*(Q).
- *Etiqueta de cada nodo* $\in Q$.
- *Estado inicial:*

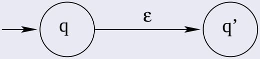


- *Estados finales:*

- *Si* $q' \in \delta(q, \sigma)$ *entonces*



- *Si* $q' \in \delta(q, \epsilon)$ *entonces*

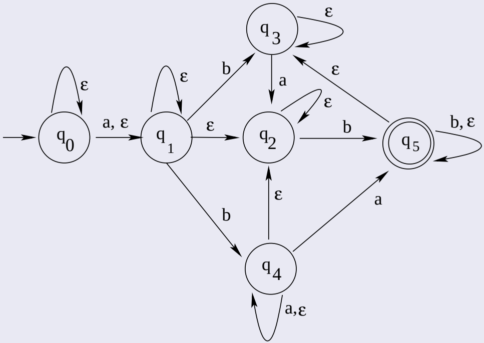


- *Las aristas de las transiciones-*ϵ *triviales se pueden omitir.*
- *Se agrupan las aristas que enlazan los mismos estados.*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo

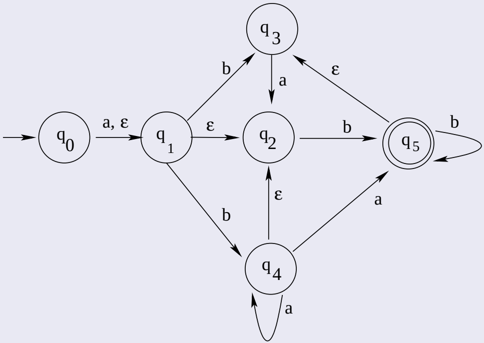


AFN **con** *transiciones triviales*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo



AFN **sin** *transiciones triviales*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

- Descripción general
- Definición formal
- Representación gráfica
- Función de transición para palabras
- Lenguaje reconocido por un AFN
- Equivalencia entre AFN y AFD
- Equivalencia entre expresiones regulares y autómatas finitos

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

- Función de transición para palabras en un AFN:
  - Clausura - ϵ aplicada a estados
  - Clausura - ϵ aplicada a conjuntos de estados
  - Función de transición para palabras: $\hat{\delta}$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Clausura - ϵ aplicada a estados)

$$clausura - \epsilon : Q \longrightarrow \mathcal{P}(Q)$$

Si $q \in Q$

- $+$ $q \in clausura - \epsilon(q)$
- $+$ Si $q' \in clausura - \epsilon(q) \wedge q'' \in \delta(q', \epsilon)$ entonces $q'' \in clausura - \epsilon(q)$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Clausura - ϵ aplicada a estados)

Segunda versión

$$clausura - \epsilon : Q \longrightarrow \mathcal{P}(Q)$$

$$clausura - \epsilon(q) = \{q' | q' \in Q \wedge \exists \text{un camino de } q \text{ a } q' \text{ con las aristas etiquetadas con } \epsilon\}$$

Nota

Siempre se verifica que $q \in clausura - \epsilon(q)$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplos (Clausura-ϵ de los estados del AFN anterior)

$$
\begin{aligned}
clausura - \epsilon(q_0) &= \{q_0, q_1, q_2\} \\
clausura - \epsilon(q_1) &= \{q_1, q_2\} \\
clausura - \epsilon(q_2) &= \{q_2\} \\
clausura - \epsilon(q_3) &= \{q_3\} \\
clausura - \epsilon(q_4) &= \{q_2, q_4\} \\
clausura - \epsilon(q_5) &= \{q_3, q_5\}
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Clausura - ϵ aplicada a conjuntos de estados)

$$clausura - \epsilon : \mathcal{P}(Q) \longrightarrow \mathcal{P}(Q)$$

Si $P \subseteq Q$

$+ \quad P \in clausura - \epsilon(P)$

$+ \quad$ Si $q' \in clausura - \epsilon(P) \wedge q'' \in \delta(q', \epsilon)$
entonces $q'' \in clausura - \epsilon(P)$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Clausura - ϵ aplicada a conjuntos de estados)

Segunda versión

Si $P \subseteq Q$ *entonces*

$$clausura - \epsilon(P) = \bigcup_{q \in P} clausura - \epsilon(q)$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Clausura-ϵ de un conjunto de estados)

$$
\begin{aligned}
clausura-\epsilon(\{q_1,q_3\}) &= \bigcup_{q\in\{q_1,q_3\}} clausura-\epsilon(q)\\
&= clausura-\epsilon(q_1)\cup clausura-\epsilon(q_3)\\
&= \{q_1,q_2\}\cup\{q_3\}\\
&= \{q_1,q_2,q_3\}
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Clausura-ϵ de un conjunto de estados)

$$
\begin{aligned}
clausura - \epsilon(\{q_0, q_5\}) &= \bigcup_{q \in \{q_0, q_5\}} clausura - \epsilon(q) \\
&= clausura - \epsilon(q_0) \cup clausura - \epsilon(q_5) \\
&= \{q_0, q_1, q_2\} \cup \{q_3, q_5\} \\
&= \{q_0, q_1, q_2, q_3, q_5\}
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Función de transición para palabras)

$$\hat{\delta} : Q \times \Sigma^* \longrightarrow \mathcal{P}(Q)$$

$$\hat{\delta}(q, \epsilon) = clausura - \epsilon(q)$$

$$\hat{\delta}(q, x\sigma) = clausura - \epsilon \left(\bigcup_{q' \in \hat{\delta}(q,x)} \delta(q', \sigma) \right)$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Función de transición para palabras)

En particular

$$
\begin{array}{rrcl}
+ & \hat{\delta}(q,\sigma) & = & clausura-\epsilon\left(\bigcup\limits_{q'\in\hat{\delta}(q,\epsilon)} \delta(q',\sigma)\right) \\
 & & = & clausura-\epsilon\left(\bigcup\limits_{q'\in clausura-\epsilon(q)} \delta(q',\sigma)\right)
\end{array}
$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Reconocimiento de una palabra: 1 / 8)

$$
\begin{aligned}
\hat{\delta}(q_0, x) &= \hat{\delta}(q_0, abb) \\
&= clausura - \epsilon \left(\bigcup_{q' \in \hat{\delta}(q_0, ab)} \delta(q', b) \right)
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Reconocimiento de una palabra: 2 / 8)

$$\hat{\delta}(q_0, ab) \quad = \quad clausura - \epsilon \left(\bigcup_{q' \in \hat{\delta}(q_0, a)} \delta(q', b) \right)$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Reconocimiento de una palabra: 3 / 8)

$$\hat{\delta}(q_0, a) \quad = \quad clausura - \epsilon \left(\bigcup_{q' \in \hat{\delta}(q_0, \epsilon)} \delta(q', a) \right)$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Reconocimiento de una palabra: 4 / 8)

$$\begin{aligned}\hat{\delta}(q_0, \epsilon) &= clausura - \epsilon(q_0) \\ &= \{q_0, q_1, q_2\}\end{aligned}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Reconocimiento de una palabra: 5 / 8)

Se sustituye $\hat{\delta}(q_0, \epsilon)$ *en* $\hat{\delta}(q_0, a)$*:*

$$
\begin{aligned}
\hat{\delta}(q_0, a) &= clausura - \epsilon \left(\bigcup_{q' \in \hat{\delta}(q_0, \epsilon)} \delta(q', a) \right) \\
&= clausura - \epsilon \left(\bigcup_{q' \in \{q_0, q_1, q_2\}} \delta(q', a) \right) \\
&= clausura - \epsilon(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)) \\
&= clausura - \epsilon(\{q_1\} \cup \emptyset \cup \emptyset) \\
&= \{q_1, q_2\}
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Reconocimiento de una palabra: 6 / 8)

Se sustituye $\hat{\delta}(q_0, a)$ *en* $\hat{\delta}(q_0, ab)$*:*

$$
\begin{aligned}
\hat{\delta}(q_0, ab) &= clausura - \epsilon\left(\bigcup_{q' \in \hat{\delta}(q_0, a)} \delta(q', b)\right) \\
&= clausura - \epsilon\left(\bigcup_{q' \in \{q_1, q_2\}} \delta(q', b)\right) \\
&= clausura - \epsilon(\delta(q_1, b) \cup \delta(q_2, b)) \\
&= clausura - \epsilon(\{q_3, q_4\} \cup \{q_5\}) \\
&= clausura - \epsilon(\{q_3, q_4, q_5\}) \\
&= \{q_2, q_3, q_4, q_5\}
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Reconocimiento de una palabra: 7 / 8)

Por último, se sustituye $\hat{\delta}(q_0, ab)$ *en* $\hat{\delta}(q_0, abb)$*:*

$$
\begin{aligned}
\hat{\delta}(q_0, abb) &= clausura - \epsilon\left(\bigcup_{q' \in \hat{\delta}(q_0, ab)} \delta(q', b)\right) \\
&= clausura - \epsilon\left(\bigcup_{q' \in \{q_2, q_3, q_4, q_5\}} \delta(q', b)\right) \\
&= clausura - \epsilon(\delta(q_2, b) \cup \delta(q_3, b) \cup \delta(q_4, b) \cup \delta(q_5, b)) \\
&= clausura - \epsilon(\{q_5\} \cup \emptyset \cup \emptyset \cup \{q_5\}) \\
&= clausura - \epsilon(\{q_5\}) \;=\; \{q_3, q_5\}
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Reconocimiento de una palabra: 8 / 8)

$x = abb \in L(A)$ *porque*

$$\hat{\delta}(q_0, abb) \cap F = \{q_3, q_5\} \cap F = \{q_5\} \neq \emptyset$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

- Descripción general
- Definición formal
- Representación gráfica
- Función de transición para palabras
- Lenguaje reconocido por un AFN
- Equivalencia entre AFN y AFD
- Equivalencia entre expresiones regulares y autómatas finitos

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Lenguaje reconocido por un AFN)

$$L(A) = \{x | x \in \Sigma^* \wedge \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

- Descripción general
- Definición formal
- Representación gráfica
- Función de transición para palabras
- Lenguaje reconocido por un AFN
- Equivalencia entre AFN y AFD
- Equivalencia entre expresiones regulares y autómatas finitos

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Necesidad de convertir AFN en AFD

- Calcular $\hat{\delta}(q_0, x)$ en un AFN es muy tedioso.
- Se suele evitar el uso de AFN.
- AFN y AFD tienen la misma capacidad de reconocimiento.
- Paso de AFN a AFD: algoritmo de Construcción de subconjuntos.
- Se ha de extender la definición de la función de transición a subconjuntos de Q.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Definición (Extensión de δ a subconjuntos de Q)

$$\delta : \mathcal{P}(Q) \times (\Sigma \cup \{\epsilon\}) \longrightarrow \mathcal{P}(Q)$$
$$\delta(P, \sigma) = \bigcup_{q \in P} \delta(q, \sigma)$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Aplicación de δ a $p \subseteq Q$)

Sea el AFN anterior y $p = \{q_0, q_3\}$*:*

$$
\begin{aligned}
\delta(p, a) &= \delta(\{q_0, q_3\}, a) \\
&= \bigcup_{q \in \{q_0, q_3\}} \delta(q, a) \\
&= \delta(q_0, a) \cup \delta(q_3, a) \\
&= \{q_1\} \cup \{q_2\} \\
&= \{q_1, q_2\}
\end{aligned}
$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Teorema

Dado un AFN A_N, se puede construir otro AFD A_D **equivalente***:*

$$L(A_N) = L(A_D)$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Demostración

- *Algoritmo de Construcción de subconjuntos*

AFN

↓

Algoritmo de Construcción de Subconjuntos

↓

AFD

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Algoritmo (Construcción de subconjuntos)

- *Entrada:* $A_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$
- *Salida:* $A_D = (Q_D, \Sigma, \delta_D, p_0, F_D)$

Algoritmo (Construcción de subconjuntos)

inicio

$p_0 \leftarrow clausura - \epsilon(q_0)$; $Q_D \leftarrow \{p_0\}$ *y* p_0 *no marcado*

mientras *haya un estado* $p \in Q_D$ *no marcado* **hacer**

Marcar a p

para *cada* $\sigma \in \Sigma$ **hacer**

$p' \leftarrow clausura - \epsilon(\delta_N(p, \sigma))$

si $p' \notin Q_D$ **entonces**

$Q_D \leftarrow Q_D \cup \{p'\}$ *y* p' *no marcado*

fin_si

Definir $\delta_D(p, \sigma) \leftarrow p'$

fin para

fin mientras

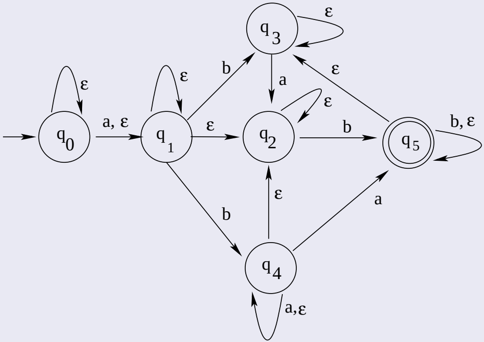
$F_D \leftarrow \{p_i | F_N \cap p_i \neq \emptyset\}$

fin

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

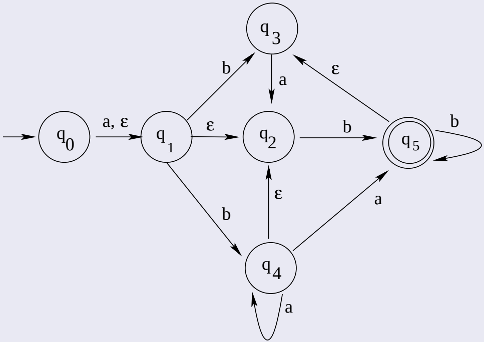


AFN **con** *transiciones triviales*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



AFN **sin** *transiciones triviales*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

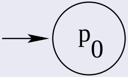
Paso 0: *Estado inicial del AFD:*

$$\begin{aligned} p_0 &= clausura - \epsilon(q_0) = \{q_0, q_1, q_2\} \\ Q_D &= \{p_0\} \end{aligned}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 0: *estado inicial* p_0

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 1: *Transiciones de* $p_0 = \{q_0, q_1, q_2\}$

Se marca el estado p_0*:*

$$Q_D = \{\underline{p}_0\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 1: *Transiciones de* $p_0 = \{q_0, q_1, q_2\}$

$$
\begin{aligned}
clausura - \epsilon(\delta_N(p_0, a)) &= \\
&= clausura - \epsilon(\bigcup_{q \in p_0} \delta_N(q, a)) \\
&= clausura - \epsilon(\bigcup_{q \in \{q_0, q_1, q_2\}} \delta_N(q, a)) \\
&= clausura - \epsilon(\delta_N(q_0, a) \cup \delta_N(q_1, a) \cup \delta_N(q_2, a)) \\
&= clausura - \epsilon(\{q_1\} \cup \emptyset \cup \emptyset) \\
&= clausura - \epsilon(\{q_1\}) = \{q_1, q_2\} = p_1
\end{aligned}
$$

Por tanto, $\boxed{\delta_D(p_0, a) = p_1}$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 1: *Transiciones de* $p_0 = \{q_0, q_1, q_2\}$

Como $p_1 \notin Q_D$*:*

$$Q_D = Q_D \cup \{p_1\} = \{\underline{p}_0, p_1\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 1: *Transiciones de* $p_0 = \{q_0, q_1, q_2\}$

$$
\begin{aligned}
clausura - \epsilon(\delta_N(p_0, b)) &= \\
&= clausura - \epsilon(\bigcup_{q \in p_0} \delta_N(q, b)) \\
&= \ldots \\
&= clausura - \epsilon(\delta_N(q_0, b) \cup \delta_N(q_1, b) \cup \delta_N(q_2, b)) \\
&= clausura - \epsilon(\emptyset \cup \{q_3, q_4\} \cup \{q_5\}) \\
&= clausura - \epsilon(\{q_3, q_4, q_5\}) \\
&= \{q_2, q_3, q_4, q_5\} = p_2
\end{aligned}
$$

$$\boxed{\delta_D(p_0, b) = p_2}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 1: *Transiciones de* $p_0 = \{q_0, q_1, q_2\}$

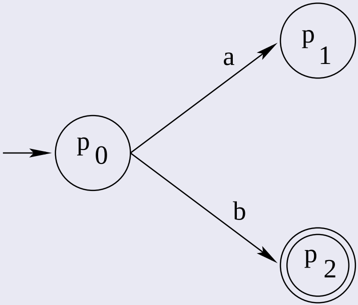
Como $p_2 \notin Q_D$:

$$Q_D = Q_D \cup \{p_2\} = \{\underline{p}_0, p_1, p_2\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 1: *transiciones de* p_0

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 2: *Transiciones de* $p_1 = \{q_1, q_2\}$

Se marca el estado p_1*:*

$$Q_D = \{\underline{p}_0, \underline{p}_1, p_2\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 2: *Transiciones de* $p_1 = \{q_1, q_2\}$

$$
\begin{aligned}
clausura - \epsilon(\delta_N(p_1, a)) &= clausura - \epsilon\Big(\bigcup_{q \in p_1} \delta_N(q, a)\Big) \\
&= \dots \\
&= clausura - \epsilon(\delta_N(q_1, a) \cup \delta_N(q_2, a)) \\
&= clausura - \epsilon(\emptyset \cup \emptyset) \\
&= clausura - \epsilon(\emptyset) = \emptyset
\end{aligned}
$$

Por tanto, $\boxed{\delta_D(p_1, a) = -}$*, es decir, está indefinida.*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 2: *Transiciones de* $p_1 = \{q_1, q_2\}$

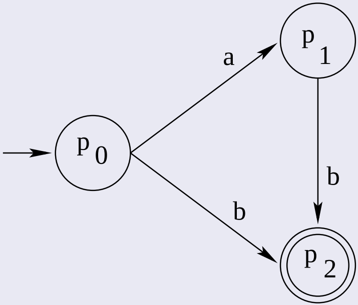
$$
\begin{aligned}
clausura-\epsilon(\delta_N(p_1, b)) &= clausura-\epsilon(\bigcup_{q \in p_1} \delta_N(q, b)) \\
&= \cdots \\
&= clausura-\epsilon(\delta_N(q_1, b) \cup \delta_N(q_2, b)) \\
&= clausura-\epsilon(\{q_3, q_4\} \cup \{q_5\}) \\
&= clausura-\epsilon(\{q_3, q_4, q_5\}) \\
&= \{q_2, q_3, q_4, q_5\} = p_2
\end{aligned}
$$

$$\boxed{\delta_D(p_1, b) = p_2}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 2: *transiciones de p_1*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 3: *Transiciones de* $p_2 = \{q_2, q_3, q_4, q_5\}$

Se marca el estado p_2*:*

$$Q_D = \{\underline{p}_0, \underline{p}_1, \underline{p}_2\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 3: *Transiciones de* $p_2 = \{q_2, q_3, q_4, q_5\}$

$$
\begin{aligned}
clausura-\epsilon(\delta_N(p_2,a)) &= clausura-\epsilon(\bigcup_{q\in p_2}\delta_N(q,a)) \\
&= clausura-\epsilon(\bigcup_{\{q_2,q_3,q_4,q_5\}}\delta_N(q,a)) \\
&= clausura-\epsilon(\delta_N(q_2,a)\cup\cdots\cup\delta_N(q_5,a)) \\
&= clausura-\epsilon(\emptyset\cup\{q_2\}\cup\{q_4,q_5\}\cup\emptyset) \\
&= clausura-\epsilon(\{q_2,q_4,q_5\}) \\
&= \{q_2,q_3,q_4,q_5\} = p_2
\end{aligned}
$$

$$\boxed{\delta_D(p_2,a) = p_2}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 3: *Transiciones de* $p_2 = \{q_2, q_3, q_4, q_5\}$

$$
\begin{aligned}
clausura-\epsilon(\delta_N(p_2, b)) &= clausura-\epsilon(\bigcup_{q \in p_2} \delta_N(q, b)) \\
&= clausura-\epsilon(\bigcup_{\{q_2, q_3, q_4, q_5\}} \delta_N(q, a)) \\
&= clausura-\epsilon(\delta_N(q_2, b) \cup \cdots \cup \delta_N(q_5, b)) \\
&= clausura-\epsilon(\{q_5\} \cup \emptyset \cup \emptyset \cup \{q_5\}) \\
&= clausura-\epsilon(\{q_5\}) \\
&= \{q_3, q_5\} = p_3
\end{aligned}
$$

$$\boxed{\delta_D(p_2, b) = p_3}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 3: *Transiciones de* $p_2 = \{q_2, q_3, q_4, q_5\}$

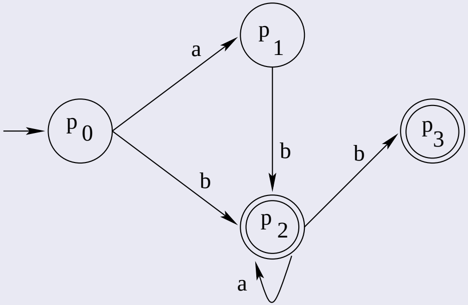
Como $p_3 \notin Q_D$

$$Q_D = \{\underline{p}_0, \underline{p}_1, \underline{p}_2, p_3\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 3: *transiciones de* p_2

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 4: *Transiciones de* $p_3 = \{q_3, q_5\}$

Se marca el estado p_3*:*

$$Q_D = \{\underline{p}_0, \underline{p}_1, \underline{p}_2, \underline{p}_3\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 4: *Transiciones de* $p_3 = \{q_3, q_5\}$

$$
\begin{aligned}
clausura - \epsilon(\delta_N(p_3, a)) &= clausura - \epsilon(\bigcup_{q \in p_3} \delta_N(q, a)) \\
&= \ldots \\
&= clausura - \epsilon(\delta_N(q_3, a) \cup \delta_N(q_5, a)) \\
&= clausura - \epsilon(\{q_2\} \cup \emptyset) \\
&= clausura - \epsilon(\{q_2\}) \\
&= \{q_2\} = p_4
\end{aligned}
$$

$$\boxed{\delta_D(p_3, a) = p_4}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 4: *Transiciones de* $p_3 = \{q_3, q_5\}$

Como $p_4 \notin Q_D$

$$Q_D = \{\underline{p}_0, \underline{p}_1, \underline{p}_2, \underline{p}_3, p_4\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 4: *Transiciones de* $p_3 = \{q_3, q_5\}$

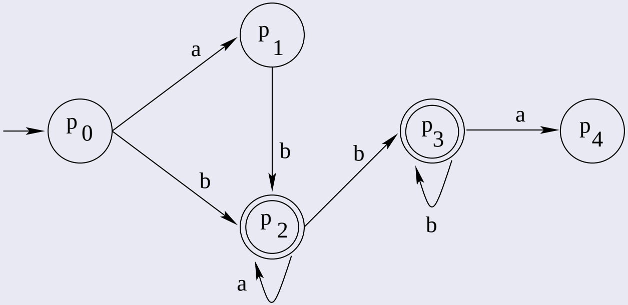
$$
\begin{aligned}
clausura-\epsilon(\delta_N(p_3,b)) &= clausura-\epsilon\Big(\bigcup_{q\in p_3}\delta_N(q,b)\Big)\\
&= \cdots\\
&= clausura-\epsilon(\delta_N(q_3,b)\cup\delta_N(q_5,b))\\
&= clausura-\epsilon(\emptyset\cup\{q_5\})\\
&= clausura-\epsilon(\{q_5\})\\
&= \{q_3,q_5\} = p_3
\end{aligned}
$$

$$\boxed{\delta_D(p_3,b) = p_3}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 4: *transiciones de p_3*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 5: *Transiciones de* $p_4 = \{q_2\}$

Se marca el estado p_4

$$Q_D = \{\underline{p}_0, \underline{p}_1, \underline{p}_2, \underline{p}_3, \underline{p}_4\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 5: *Transiciones de* $p_4 = \{q_2\}$

$$\begin{aligned} clausura-\epsilon(\delta_N(p_4,a)) &= clausura-\epsilon(\bigcup_{q\in p_4}\delta_N(q,a)) \\ &= \cdots \\ &= clausura-\epsilon(\delta_N(q_2,a)) \\ &= clausura-\epsilon(\emptyset) = \emptyset \end{aligned}$$

$$\boxed{\delta_D(p_4,a) = -}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 5: *Transiciones de* $p_4 = \{q_2\}$

$$
\begin{aligned}
clausura - \epsilon(\delta_N(p_4, b)) &= clausura - \epsilon(\bigcup_{q \in p_4} \delta_N(q, b)) \\
&= \cdots \\
&= clausura - \epsilon(\delta_N(q_2, b)) \\
&= clausura - \epsilon(\{q_5\}) \\
&= \{q_3, q_5\} = p_3
\end{aligned}
$$

$$\boxed{\delta(p_4, b) = p_3}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

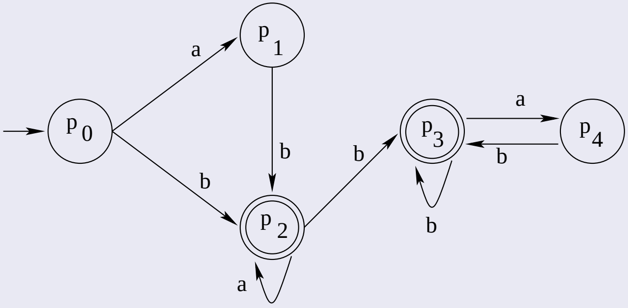
Nota (Algoritmo de Construcción de Subconjuntos)

El algoritmo finaliza al estar marcados todos los estados de Q_D.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 5: *transiciones de p_4*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

La función de transición del AFD es:

	δ	a	b
$\rightarrow$	p_0	p_1	p_2
	p_1	$-$	p_2
$\leftarrow$	p_2	p_2	p_3
$\leftarrow$	p_3	p_4	p_3
	p_4	$-$	p_3

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Los dos únicos estados finales son p_2 *y* p_3 *porque*

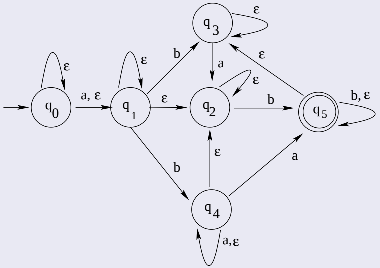
$$p_2 \cap F_N = \{q_2, q_3, q_4, q_5\} \cap \{q_5\} = \{q_5\} \neq \emptyset$$

$$p_3 \cap F_N = \{q_3, q_5\} \cap \{q_5\} = \{q_5\} \neq \emptyset$$

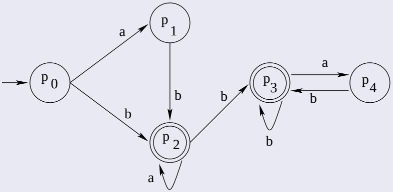
Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



AFN original



AFD construido

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Análisis de $x = abb$ *usando el* AFD *construido:*

$$\begin{array}{rcl} (p_0, abb) & \vdash & (p_1, bb) \\ & \vdash & (p_2, b) \\ & \vdash & (p_3, \epsilon) \end{array}$$

$x \in L(A_D)$ *porque* $p_3 \in F_D$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

- Descripción general
- Definición formal
- Representación gráfica
- Función de transición para palabras
- Lenguaje reconocido por un AFN
- Equivalencia entre AFN y AFD
- Equivalencia entre expresiones regulares y autómatas finitos

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

AFN

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

AFN

$\downarrow$

$\boxed{\text{Algoritmo de Construcción de Subconjuntos}}$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

AFN

$\downarrow$

Algoritmo de Construcción de Subconjuntos

$\downarrow$

AFD

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

EXPRESIÓN REGULAR

↓

AFN

↓

Algoritmo de Construcción de Subconjuntos

↓

AFD

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

EXPRESIÓN REGULAR

↓

Algoritmo de Thompson

↓

AFN

↓

Algoritmo de Construcción de Subconjuntos

↓

AFD

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Teorema

Dada una **expresión regular** α*, se puede construir un* **AFN** A_N *equivalente:*

$$L(\alpha) = L(A_N)$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Demostración

- *Algoritmo de Thompson*

EXPRESIÓN REGULAR

↓

$\boxed{\text{Algoritmo de Thompson}}$

↓

AFN

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Algoritmo (Thompson)

- *Entrada:* **expresión regular** α.
- *Salida:* **AFN** $A_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

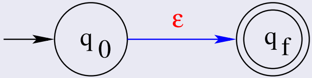
Algoritmo (Thompson)

AFN equivalente a la expresión regular $\emptyset$.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Algoritmo (Thompson)

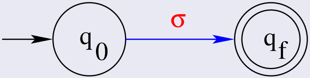


AFN equivalente a la expresión regular ϵ.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Algoritmo (Thompson)

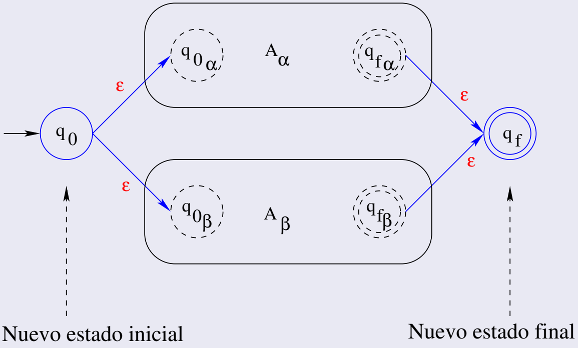


AFN equivalente a la expresión regular σ*.*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Algoritmo (Thompson)

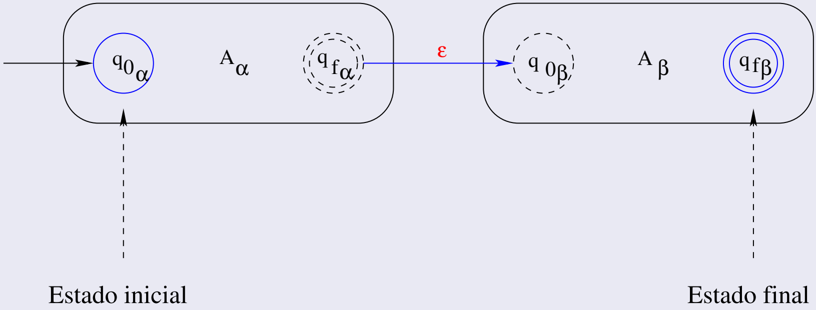


AFN equivalente a la expresión regular $\alpha + \beta$.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Algoritmo (Thompson)

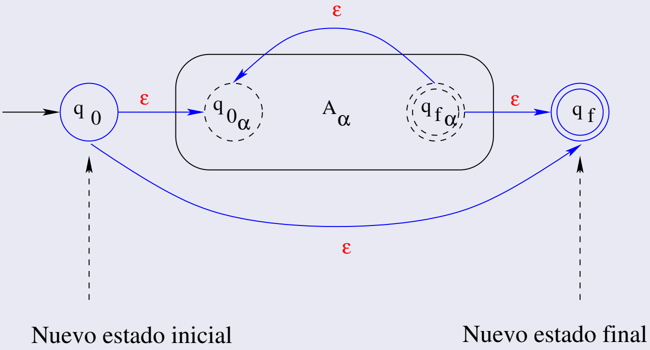


AFN equivalente a la expresión regular $\alpha\beta$.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Algoritmo (Thompson)



AFN equivalente a la expresión regular α^*.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

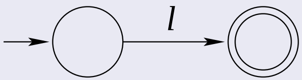
Nota (Algoritmo de Thompson)

$$L((\alpha)) = L(\alpha) = L(A_N)$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Thompson aplicado a $\alpha = l(l+d)^*$)

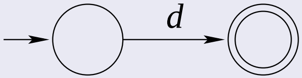


Paso 1: *AFN equivalente a l*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Thompson aplicado a $\alpha = l(l+d)^*$)

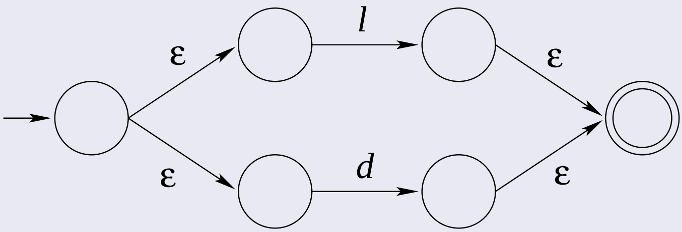


Paso 2: *AFN equivalente a d*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Thompson aplicado a $\alpha = l(l+d)^*$)

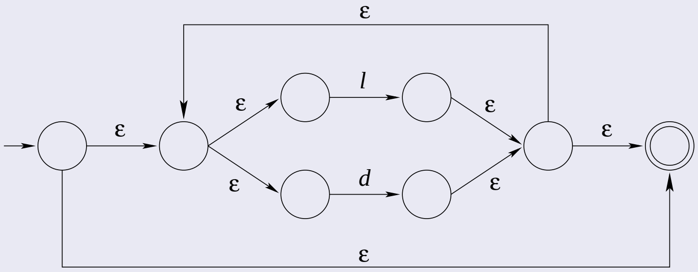


Paso 3: *AFN equivalente a $l + d$*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Thompson aplicado a $\alpha = l(l+d)^*$)

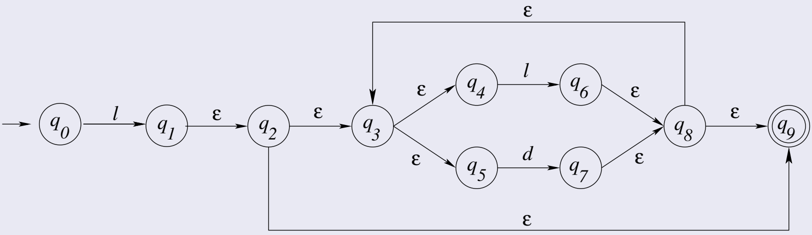


Paso 4: *AFN equivalente a* $(l+d)^*$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Thompson aplicado a $\alpha = l(l+d)^*$)



Paso 5: *AFN equivalente a $\alpha = l(l+d)^*$*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

EXPRESIÓN REGULAR

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

EXPRESIÓN REGULAR

$\downarrow$

$\boxed{\text{Algoritmo de Thompson}}$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

EXPRESIÓN REGULAR

↓

$\boxed{\text{Algoritmo de Thompson}}$

↓

AFN

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

EXPRESIÓN REGULAR

↓

Algoritmo de Thompson

↓

AFN

↓

Algoritmo de Construcción de Subconjuntos

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

EXPRESIÓN REGULAR

$\downarrow$

Algoritmo de Thompson

$\downarrow$

AFN

$\downarrow$

Algoritmo de Construcción de Subconjuntos

$\downarrow$

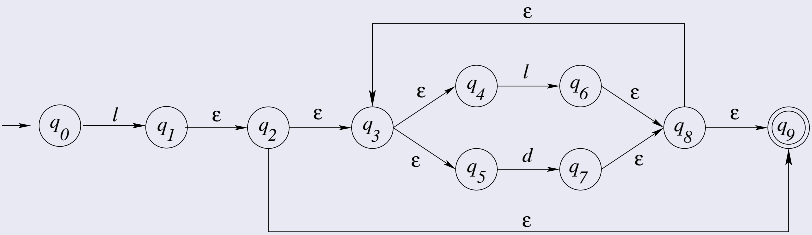
AFD

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de subconjuntos)

AFN equivalente a $\alpha = l(l + d)^*$



Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

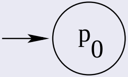
Paso 0: *Estado inicial del AFD:*

$$\begin{aligned} p_0 &= clausura - \epsilon(q_0) = \{q_0\} \\ Q_D &= \{p_0\} \end{aligned}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 0: *estado inicial p_0*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 1: *Transiciones de* $p_0 = \{q_0\}$

Se marca el estado p_0*:*

$$Q_D = \{\underline{p}_0\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 1: *Transiciones de* $p_0 = \{q_0\}$

$$
\begin{aligned}
clausura - \epsilon(\delta_N(p_0, l)) &= \\
&= clausura - \epsilon\Big(\bigcup_{q \in p_0} \delta_N(q, l)\Big) \\
&= clausura - \epsilon\Big(\bigcup_{q \in \{q_0\}} \delta_N(q, l)\Big) \\
&= clausura - \epsilon(\delta_N(q_0, l)) \\
&= clausura - \epsilon(\{q_1\}) \\
&= \{q_1, q_2, q_3, q_4, q_5, q_9\} = p_1
\end{aligned}
$$

Por tanto, $\boxed{\delta_D(p_0, l) = p_1}$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 1: *Transiciones de* $p_0 = \{q_0\}$

Como $p_1 \notin Q_D$*:*

$$Q_D = Q_D \cup \{p_1\} = \{\underline{p}_0, p_1\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 1: *Transiciones de* $p_0 = \{q_0\}$

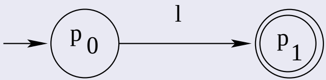
$$
\begin{aligned}
clausura - \epsilon(\delta_N(p_0, d)) &= \\
&= clausura - \epsilon(\bigcup_{q \in p_0} \delta_N(q, d)) \\
&= clausura - \epsilon(\bigcup_{q \in \{q_0\}} \delta_N(q, d)) \\
&= clausura - \epsilon(\delta_N(q_0, d))) \\
&= clausura - \epsilon(\emptyset) \\
&= \emptyset
\end{aligned}
$$

$$\boxed{\delta_D(p_0, d) = -}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 1: *transiciones de p_0*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 2: *Transiciones de* $p_1 = \{q_1, q_2, q_3, q_4, q_5, q_9\}$

Se marca el estado p_1*:*

$$Q_D = \{\underline{p}_0, \underline{p}_1\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 2: *Transiciones de* $p_1 = \{q_1, q_2, q_3, q_4, q_5, q_9\}$

$$
\begin{aligned}
clausura - \epsilon(\delta_N(p_1, l)) &= clausura - \epsilon\Big(\bigcup_{q \in p_1} \delta(q, l)\Big) \\
&= \cdots \\
&= clausura - \epsilon(\delta_N(q_1, l) \cup \cdots \cup \delta_N(q_9, l)) \\
&= clausura - \epsilon(\emptyset \cup \emptyset \cup \emptyset \cup \{q_6\} \cup \emptyset) \\
&= clausura - \epsilon(\{q_6\}) \\
&= \{q_3, q_4, q_5, q_6, q_8, q_9\} = p_2
\end{aligned}
$$

Por tanto, $\boxed{\delta_D(p_1, l) = p_2}$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 2: *Transiciones de* $p_1 = \{q_1, q_2, q_3, q_4, q_5, q_9\}$

Como $p_2 \notin Q_D$*:*

$$Q_D = Q_D \cup \{p_2\} = \{\underline{p}_0, \underline{p}_1, p_2\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 2: *Transiciones de* $p_1 = \{q_1, q_2, q_3, q_4, q_5, q_9\}$

$$
\begin{aligned}
clausura-\epsilon(\delta_N(p_1, d)) &= clausura-\epsilon(\bigcup_{q \in p_1} \delta(q, d)) \\
&= \cdots \\
&= clausura-\epsilon(\delta_N(q_1, d) \cup \cdots \cup \delta_N(q_9, d)) \\
&= clausura-\epsilon(\emptyset \cup \emptyset \cup \emptyset \cup \{q_7\} \cup \emptyset) \\
&= clausura-\epsilon(\{q_7\}) \\
&= \{q_3, q_4, q_5, q_7, q_8, q_9\} = p_3
\end{aligned}
$$

Por tanto, $\boxed{\delta_D(p_1, d) = p_3}$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 2: *Transiciones de* $p_1 = \{q_1, q_2, q_3, q_4, q_5, q_9\}$

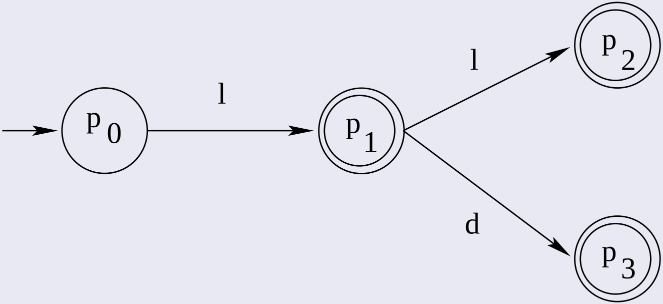
Como $p_3 \notin Q_D$:

$$Q_D = Q_D \cup \{p_3\} = \{\underline{p}_0, \underline{p}_1, p_2, p_3\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 2: *transiciones de* p_1

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 3: *Transiciones de* $p_2 = \{q_3, q_4, q_5, q_6, q_8, q_9\}$

Se marca el estado p_2*:*

$$Q_D = \{\underline{p}_0, \underline{p}_1, \underline{p}_2, p_3\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 3: *Transiciones de* $p_2 = \{q_3, q_4, q_5, q_6, q_8, q_9\}$

$$
\begin{aligned}
clausura-\epsilon(\delta_N(p_2, l)) &= clausura-\epsilon\Big(\bigcup_{q \in p_2} \delta_N(q, l)\Big) \\
&= \cdots \\
&= clausura-\epsilon(\delta_N(q_3, l) \cup \cdots \cup \delta_N(q_9, l)) \\
&= clausura-\epsilon(\emptyset \cup \{q_6\} \cup \emptyset \cup \emptyset \cup \emptyset \cup \emptyset) \\
&= clausura-\epsilon(\{q_6\}) \\
&= \{q_3, q_4, q_5, q_6, q_8, q_9\} = p_2
\end{aligned}
$$

$$\boxed{\delta_D(p_2, l) = p_2}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 3: *Transiciones de* $p_2 = \{q_3, q_4, q_5, q_6, q_8, q_9\}$

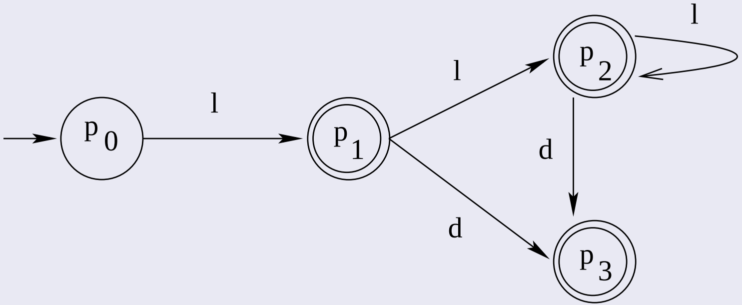
$$
\begin{aligned}
clausura - \epsilon(\delta_N(p_2, d)) &= clausura - \epsilon(\bigcup_{q \in p_2} \delta_N(q, d)) \\
&= \cdots \\
&= clausura - \epsilon(\delta_N(q_3, d) \cup \cdots \cup \delta_N(q_9, d)) \\
&= clausura - \epsilon(\emptyset \cup \emptyset \cup \{q_7\} \cup \emptyset \cup \emptyset \cup \emptyset) \\
&= clausura - \epsilon(\{q_7\}) \\
&= \{q_3, q_4, q_5, q_7, q_8, q_9\} = p_3
\end{aligned}
$$

$$\boxed{\delta_D(p_2, d) = p_3}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 3: *transiciones de p_2*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 4: *Transiciones de* $p_3 = \{q_3, q_4, q_5, q_7, q_8, q_9\}$

Se marca el estado p_3*:*

$$Q_D = \{\underline{p}_0, \underline{p}_1, \underline{p}_2, \underline{p}_3\}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 4: *Transiciones de* $p_3 = \{q_3, q_4, q_5, q_7, q_8, q_9\}$

$$
\begin{aligned}
clausura-\epsilon(\delta(p_3, l)) &= clausura-\epsilon(\bigcup_{q \in p_3} \delta(q, l)) \\
&= clausura-\epsilon(\delta_N(q_3, l) \cup \cdots \cup \delta_N(q_9, l)) \\
&= clausura-\epsilon(\emptyset \cup \{q_6\} \cup \emptyset \cup \emptyset \cup \emptyset \cup \emptyset) \\
&= clausura-\epsilon(\{q_6\}) \\
&= \{q_3, q_4, q_5, q_6, q_8, q_9\} = p_2
\end{aligned}
$$

$$\boxed{\delta_D(p_3, l) = p_2}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Paso 4: *Transiciones de* $p_3 = \{q_3, q_4, q_5, q_7, q_8, q_9\}$

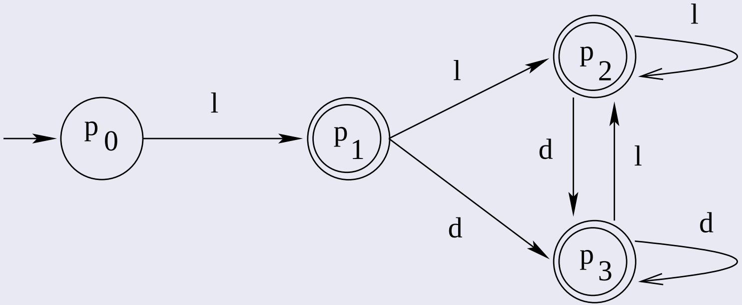
$$
\begin{aligned}
clausura-\epsilon(\delta(p_3,d)) &= clausura-\epsilon\Big(\bigcup_{q\in p_3}\delta(q,d)\Big)\\
&= clausura-\epsilon(\delta_N(q_3,d)\cup\cdots\cup\delta_N(q_8,d))\\
&= clausura-\epsilon(\emptyset\cup\emptyset\cup\{q_7\}\cup\emptyset\cup\emptyset\cup\emptyset)\\
&= clausura-\epsilon(\{q_7\})\\
&= \{q_3,q_4,q_5,q_7,q_8,q_9\} = p_3
\end{aligned}
$$

$$\boxed{\delta_D(p_3,d) = p_3}$$

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



Paso 4: *transiciones de p_3*

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Nota (Algoritmo de Construcción de Subconjuntos)

El algoritmo finaliza al estar marcados todos los estados de Q_D.

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Función de transición del AFD

	δ	l	d
$\rightarrow$	p_0	p_1	$-$
$\leftarrow$	p_1	p_2	p_3
$\leftarrow$	p_2	p_2	p_3
$\leftarrow$	p_3	p_2	p_3

Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)

Estados **finales**: p_1, p_2 y p_3

$$p_1 \cap F_N = \{q_1, q_2, q_3, q_4, q_5, q_9\} \cap \{q_9\} = \{q_9\} \neq \emptyset$$

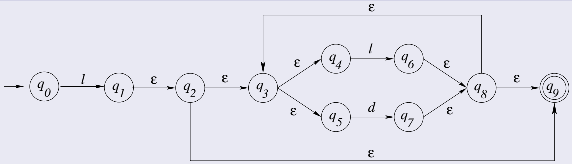
$$p_2 \cap F_N = \{q_3, q_4, q_5, q_6, q_8, q_9\} \cap \{q_9\} = \{q_9\} \neq \emptyset$$

$$p_3 \cap F_N = \{q_3, q_4, q_5, q_7, q_8, q_9\} \cap \{q_9\} = \{q_9\} \neq \emptyset$$

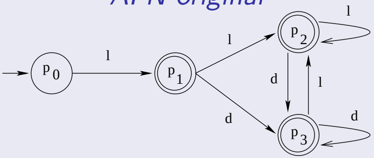
Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



AFN original

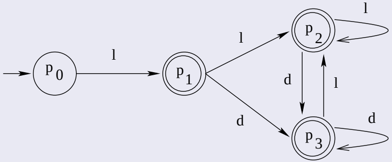


AFD equivalente a $\alpha = l(l+d)^*$

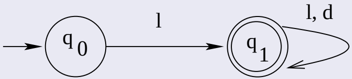
Reconocimiento de componentes léxicos

Autómatas finitos NO deterministas: AFN

Ejemplo (Algoritmo de Construcción de Subconjuntos)



AFD equivalente a $\alpha = l(l+d)^*$



AFD minimizado y equivalente a $\alpha = l(l+d)^*$

Contenido de la sección

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Definición

Dos **autómatas** *son* **equivalentes** *si reconocen el mismo lenguaje:*

$$A \equiv A' \Leftrightarrow L(A) = L(A')$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Razones para minimizar un AFD

- Un lenguaje **regular** puede ser reconocido por **varios** autómatas finitos deterministas **equivalentes**.
- Se debe usar el AFD con el **menor** número de estados.
- La Minimización permite generar el AFD que reconoce un lenguaje regular con el menor número de estados.
- La minimización está basada en una **relación de equivalencia** entre estados.
- El **autómata cociente** de la relación de equivalencia es el **AFD mínimo**.

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Definición (Equivalencia entre estados)

Se dice que dos **estados** $q, q' \in Q$ *son* **equivalentes** (qEq') *si se verifica que:*

$$\forall x \in \Sigma^* \ \ (\hat{\delta}(q,x) \in F \Leftrightarrow \hat{\delta}(q',x) \in F)$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

E es una relación de quivalencia

$\forall x \in \Sigma^* \wedge \forall q, q' \in Q$:

- **Reflexiva**: $qEq\ \hat{\delta}(q,x) \in F \Leftrightarrow \hat{\delta}(q,x) \in F$
- **Simétrica**: $qEq' \Longrightarrow q'Eq$

$$(\hat{\delta}(q,x) \in F \Leftrightarrow \hat{\delta}(q',x) \in F) \Longrightarrow (\hat{\delta}(q',x) \in F \Leftrightarrow \hat{\delta}(q,x) \in F)$$

- **Transitiva**: $qEq' \wedge q'Eq'' \Longrightarrow qEq''$

$$\left.\begin{array}{c} (\hat{\delta}(q,x) \in F \Leftrightarrow \hat{\delta}(q',x) \in F) \\ (\hat{\delta}(q',x) \in F \Leftrightarrow \hat{\delta}(q'',x) \in F) \end{array}\right\} \Rightarrow (\hat{\delta}(q,x) \in F \Leftrightarrow \hat{\delta}(q'',x) \in F)$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Definición (Clase de equivalencia de un estado)

Si $q \in Q$, la **clase de equivalencia** *de q respecto de la relación E se define como:*

$$E[q] = \{q' | qEq'\} = \{q' \mid \forall x \in \Sigma^* \ (\hat{\delta}(q,x) \in F \Leftrightarrow \hat{\delta}(q',x) \in F)\}$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Propiedades de la relación de equivalencia E

- $\forall q \in Q \;\; q \in E[q]$
- Si $q' \notin E[q]$ entonces $E[q'] \cap E[q] = \emptyset$
- $Q = \bigcup_{q \in Q} E[q]$
- $|Q_{|_E}| \leq |Q|$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Definición (Autómata cociente)

$$A_{|_E} = (Q_{|_E}, \Sigma, \delta_{|_E}, E[q_0], F_{|_E})$$

- $Q_{|_E} = \{E[q] | q \in Q\}$
- *La función de transición* $\delta_{|_E}$ *se define como:*

$$\delta_{|_E} : Q_{|_E} \times \Sigma \longrightarrow Q_{|_E}$$

$$\delta_{|_E}(p, \sigma) = p' \in Q_{|_E} \Longleftrightarrow \begin{cases} \exists q, q' \in Q \\ p = E[q] \wedge p' = E[q'] \\ \wedge \ \delta(q, \sigma) = q' \end{cases}$$

- $E[q_0]$ *es el estado inicial y*
- $F_{|_E} = \{E[q_f] | q_f \in F\}$,

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Nota

La función $\hat{\delta}_{|_E}$ *se define de manera similar a* $\hat{\delta}$.

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Teorema

Todo AFD es equivalente a su autómata cociente:

$$L(A) = L(A_{|_E})$$

Demostración

$$\begin{aligned}
x \in L(A) &\Leftrightarrow \hat{\delta}(q_0, x) \in F \\
&\Leftrightarrow \exists q_f \in F \;\; \hat{\delta}(q_0, x) = q_f \\
&\Leftrightarrow p_0 = E[q_0] \wedge \exists q_f \in F \;\; p_f = E[q_f] \wedge \hat{\delta}_{|_E}(p_0, x) = p_f \\
&\Leftrightarrow \hat{\delta}_{|_E}(p_0, x) = p_f \in F_{|_E} \\
&\Leftrightarrow x \in L(A_{|_E})
\end{aligned}$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Teorema

Todo AFD es equivalente a su autómata cociente:

$$L(A) = L(A_{|E})$$

Demostración

$$\begin{aligned}
x \in L(A) &\Leftrightarrow \hat{\delta}(q_0, x) \in F \\
&\Leftrightarrow \exists q_f \in F \;\; \hat{\delta}(q_0, x) = q_f \\
&\Leftrightarrow p_0 = E[q_0] \wedge \exists q_f \in F \;\; p_f = E[q_f] \wedge \hat{\delta}_{|E}(p_0, x) = p_f \\
&\Leftrightarrow \hat{\delta}_{|E}(p_0, x) = p_f \in F_{|E} \\
&\Leftrightarrow x \in L(A_{|E})
\end{aligned}$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Demostración

- *Algoritmo de Construcción del Autómata Cociente*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Algoritmo (Construcción del Autómata Cociente)

- *Entrada:* $A = (Q, \Sigma, \delta, q_0, F)$.
- *Salida:* $A_{|_E} = (Q_{|_E}, \Sigma, \delta_{|_E}, E[q_0], F_{|_E})$, **Autómata cociente**

Algoritmo (Construcción del Autómata Cociente)

inicio

$p_0 \leftarrow Q - F$, $p_1 \leftarrow F$

$Nuevo \leftarrow \{p_0, p_1\}$ *y* p_0 *y* p_1 *no marcados*

mientras $\exists p \in Nuevo$ *no marcado* **hacer**

Marcar a p

para *cada* $\sigma \in \Sigma$ **hacer**

Dividir p en subconjuntos de forma que sus estados

q_i *y* q_j *estarán en el mismo subconjunto*

si $\delta(q_i, \sigma)$ *y* $\delta(q_j, \sigma)$ *pertenecen al mismo subconjunto*

fin para

si *se ha dividido p en subconjuntos*

entonces

Sustituir p por los nuevos subconjuntos

Desmarcar todos los estados de Nuevo

fin si

fin mientras

fin

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplos (Minimización de AFD)

1. *AFD que reconoce identificadores de Pascal.*
2. *AFD que reconoce componentes de arrays.*
3. *AFD que reconoce algunas cadenas de ceros y unos.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

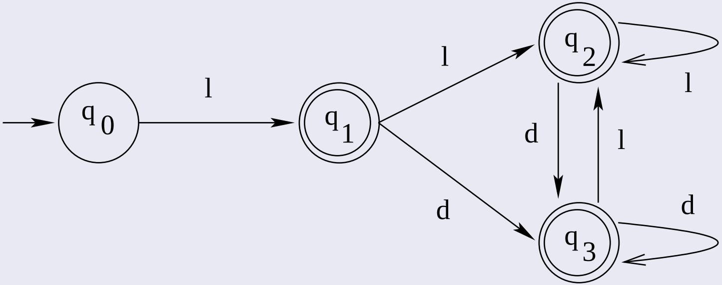
Ejemplos (Minimización de AFD)

1. *AFD que reconoce identificadores de Pascal.*
2. *AFD que reconoce componentes de arrays.*
3. *AFD que reconoce algunas cadenas de ceros y unos.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (1.- Minimización de AFD: identificadores de Pascal)



AFD original que reconoce identificadores de Pascal.

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (1.- Minimización de AFD: identificadores de Pascal)

- *Estados* **no** *finales:* $p_0 = Q - F = \{q_0\}$
- *Estados* **finales***:* $p_1 = F = \{q_1, q_2, q_3\}$
- *Nuevo* $= \{p_0, p_1\}$
- *Los estados de Nuevo* **no** *están* **marcados***.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (1.- Minimización de AFD: identificadores de Pascal)

Paso 1: *Análisis de* $p_0 = \{q_0\}$

- *Se marca* p_0: *Nuevo* $= \{\underline{p}_0, p_1\}$
- p_0 *sólo contiene un estado y no se puede descomponer más.*
- *Transiciones "provisionales" de* p_0:

$$\delta_{|E}(p_0, l) = E[\delta(q_0, l)] = E[q_1] = p_1$$
$$\delta_{|E}(p_0, d) = E[\delta(q_0, d)] = E[-] = -$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (1.- Minimización de AFD: identificadores de Pascal)

Paso 2: *Análisis de* $p_1 = \{q_1, q_2, q_3\}$

- *Se marca* p_1*: Nuevo* $= \{\underline{p}_0, \underline{p}_1\}$
- *Se comprueban que los estados de* p_1 *son homogéneos:*

δ	l	d
q_1	p_1	p_1
q_2	p_1	p_1
q_3	p_1	p_1

- *Transiciones de* p_1*:* $\delta_{|_E}(p_1, l) = p1$, $\delta_{|_E}(p_1, d) = p1$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (1.- Minimización de AFD: identificadores de Pascal)

- *El algoritmo* **finaliza** *porque todos los* **estados** *de Nuevo están* **marcados**.
- *Los estados generados son:*
  - $p_0 = \{q_0\}$
  - $p_1 = \{q_1, q_2, q_3\}$
- *El estado* **inicial** *es* p_0 *porque* $q_0 \in p_0$.
- $F_{|_E} = \{p_1\}$ *porque* $p_1 \cap F = \{q_1, q_2, q_3\} \neq \emptyset$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (1.- Minimización de AFD: identificadores de Pascal)

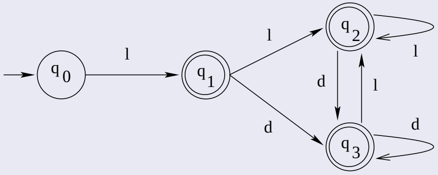
- *Función de transición del autómata minimizado*

δ	l	d
$\rightarrow p_0$	p_1	$-$
$\leftarrow p_1$	p_1	p_1

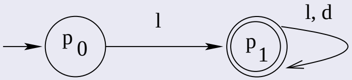
Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (1.- Minimización de AFD: identificadores de Pascal)



AFD original.



AFD minimizado.

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

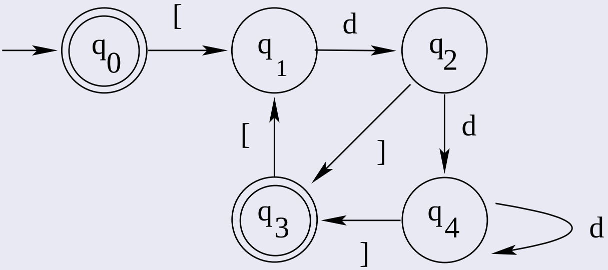
Ejemplos (Minimización de AFD)

1. *AFD que reconoce identificadores de Pascal.*
2. *AFD que reconoce componentes de arrays.*
3. *AFD que reconoce algunas cadenas de ceros y unos.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)



AFD original.

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)

- *Estados* **no** *finales:* $p_0 = Q - F = \{q_1, q_2, q_4\}$
- *Estados* **finales***:* $p_1 = F = \{q_0, q_3\}$
- *Nuevo* $= \{p_0, p_1\}$
- *Los estados* p_0 *y* p_1 **no** *están* **marcados**

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Nota (2.- Minimización de AFD: componentes de arrays)

El estado **inicial** q_0 **pertenece** *a* p_1 *porque también es un estado* **final**

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)

Paso 1: *Análisis de* $p_0 = \{q_1, q_2, q_4\}$

- *Se marca* p_0*: Nuevo* $= \{\underline{p}_0, p_1\}$
- *Se comprueban las transiciones de los estados de* p_0*:*

δ	[	d	]
q_1	$-$	p_0	$-$
q_2	$-$	p_0	p_1
q_4	$-$	p_0	p_1

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)

Paso 1: *Análisis de* $p_0 = \{q_1, q_2, q_4\}$

- q_1 **no** *es equivalente a* q_2 *y* q_4.
- *El antiguo estado* p_0 *se divide en dos nuevos estados:*

$$
\begin{aligned}
p_0 &= \{q_1\} \\
p_2 &= \{q_2, q_4\}
\end{aligned}
$$

- *Nuevo* $= \{p_0, p_1, p_2\}$
- *Todos los estados están desmarcados.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)

Paso 2: **Segundo** *análisis de* $p_0 = \{q_1\}$

- *Se marca* p_0: *Nuevo* $= \{\underline{p}_0, p_1, p_2\}$
- p_0 *sólo contiene un estado y no se puede descomponer más.*
- *Transiciones "provisionales" de* p_0:

$$\delta_{|E}(p_0, [) = E[\delta(q_1, [)] = E[-] = -$$
$$\delta_{|E}(p_0, d) = E[\delta(q_1, d)] = E[q_2] = p_2$$
$$\delta_{|E}(p_0,]) = E[\delta(q_1,])] = E[-] = -$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)

Paso 3: *Análisis de* $p_1 = \{q_0, q_3\}$

- *Se marca* p_1: *Nuevo* $= \{\underline{p}_0, \underline{p}_1, p_2\}$
- *Se comprueban que los estados de* p_1 *son homogéneos:*

δ	[	d	]
q_0	p_0	$-$	$-$
q_3	p_0	$-$	$-$

- *Transiciones "provisionales" de* p_1:

$$\delta_{|E}(p_1, [) = E[\delta(q_0, [)] = E[q_1] = p_0$$
$$\delta_{|E}(p_1, d) = E[\delta(q_0, d)] = E[-] = -$$
$$\delta_{|E}(p_1,]) = E[\delta(q_0,])] = E[-] = -$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)

Paso 4: *Análisis de* $p_2 = \{q_2, q_4\}$

- *Se marca* p_2: *Nuevo* $= \{\underline{p}_0, \underline{p}_1, \underline{p}_2\}$
- *Se comprueban que los estados de* p_1 *son homogéneos:*

δ	[	d	]
q_2	$-$	p_2	p_1
q_4	$-$	p_2	p_1

- *Transiciones de* p_2*:*

$$\delta_{|E}(p_1, [) = E[\delta(q_2, [)] = E[-] = -$$
$$\delta_{|E}(p_1, d) = E[\delta(q_2, d)] = E[q_4] = p_2$$
$$\delta_{|E}(p_1,]) = E[\delta(q_2,])] = E[q_3] = p_1$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)

- *El algoritmo* **finaliza** *porque todos los* **estados** *de Nuevo están* **marcados**.
- *Los estados generados son:*
  - $p_0 = \{q_1\}$
  - $p_1 = \{q_0, q_3\}$
  - $p_2 = \{q_2, q_4\}$
- *El estado inicial es* p_1 *porque* $q_0 \in p_1$.
- $F_{|E} = \{p_1\}$ *porque* $p_1 \cap F = \{q_0, q_3\} \neq \emptyset$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)

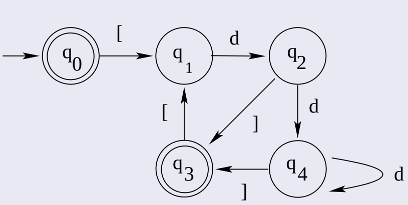
- *Función de transición del autómata minimizado*

δ	[	d	]
p_0	$-$	p_2	$-$
$\rightarrow \; \leftarrow \; p_1$	p_0	$-$	$-$
p_2	$-$	p_2	p_1

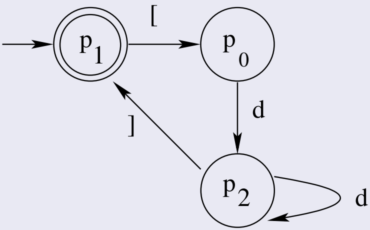
Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (2.- Minimización de AFD: componentes de arrays)



AFN original



AFD minimizado

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

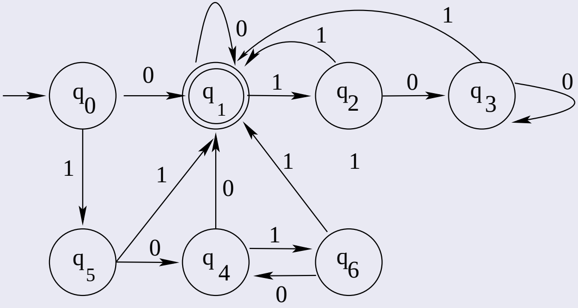
Ejemplos (Minimización de AFD: cadenas de ceros y unos)

1. *AFD que reconoce identificadores de Pascal.*
2. *AFD que reconoce componentes de arrays.*
3. *AFD que reconoce algunas cadenas de ceros y unos.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)



AFD que se va a minimizar.

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

- *Estados* **no** *finales:* $p_0 = Q - F = \{q_0, q_2, q_3, q_4, q_5, q_6\}$
- *Estados* **finales***:* $p_1 = F = \{q_1\}$
- *Nuevo* $= \{p_0, p_1\}$
- *Los estados* p_0 *y* p_1 **no** *están* **marcados**

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso 1: *Análisis de* $p_0 = \{q_0, q_2, q_3, q_4, q_5, q_6\}$

- *Se marca* p_0: *Nuevo* $= \{\underline{p_0}, p_1\}$
- *Se comprueban las transiciones de los estados de* p_0:

δ	0	1
q_0	p_1	p_0
q_2	p_0	p_1
q_3	p_0	p_1
q_4	p_1	p_0
q_5	p_0	p_1
q_6	p_0	p_1

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso número 1: *Análisis de* $p_0 = \{q_0, q_2, q_3, q_4, q_5, q_6\}$

- q_0 *y* q_4 *tienen transiciones diferentes de los otros estados*
- *El antiguo estado* p_0 *se divide en dos nuevos estados:*

$$\begin{array}{lcl} p_0 & = & \{q_0, q_4\} \\ p_2 & = & \{q_2, q_3, q_5, q_6\} \end{array}$$

- *Nuevo* $= \{p_0, p_1, p_2\}$
- *Todos los estados están desmarcados.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso 2: **Segundo** *análisis de* $p_0 = \{q_0, q_4\}$

- *Se marca* p_0: *Nuevo* $= \{\underline{p_0}, p_1, p_2\}$
- *Se comprueban si las transiciones de los estados contenidos en* p_0 *son homogéneas:*

δ	0	1
q_0	p_1	p_2
q_4	p_1	p_2

- *Transiciones "provisionales" de* p_0:

$$\delta_{|E}(p_0, 0) = E[\delta(q_0, 0)] = E[q_1] = p_1$$
$$\delta_{|E}(p_0, 1) = E[\delta(q_0, 1)] = E[q_5] = p_2$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso 3: *Análisis de* $p_1 = \{q_1\}$

- *Se marca* p_1*: Nuevo* $= \{\underline{p}_0, \underline{p}_1, p_2\}$
- p_1 *sólo contiene un estado y no se puede descomponer más.*
- *Transiciones "provisionales" de* p_1*:*

$$\delta_{|_E}(p_1, 0) = E[\delta(q_1, 0)] = E[q_1] = p_1$$
$$\delta_{|_E}(p_1, 1) = E[\delta(q_1, 1)] = E[q_2] = p_2$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso 4: *Análisis de* $p_2 = \{q_2, q_3, q_5, q_6\}$

- *Se marca* p_2: *Nuevo* $= \{\underline{p}_0, \underline{p}_1, \underline{p}_2\}$
- *Se comprueban si son homogéneas las transiciones de los estados contenidos en* p_2:

δ	0	1
q_2	p_2	p_1
q_3	p_2	p_1
q_5	p_0	p_1
q_6	p_0	p_1

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso 4: *Análisis de* $p_2 = \{q_2, q_3, q_5, q_6\}$

- q_2 *y* q_3 *tienen transiciones diferentes de los otros estados*
- *El antiguo estado* p_2 *se divide en dos nuevos estados:*

$$\begin{aligned} p_2 &= \{q_2, q_3\} \\ p_3 &= \{q_5, q_6\} \end{aligned}$$

- *Nuevo* $= \{p_0, p_1, p_2, p_3\}$
- *Todos los estados están desmarcados.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso 5: **Tercer** *análisis de* $p_0 = \{q_0, q_4\}$

- *Se marca* p_0: *Nuevo* $= \{\underline{p}_0, p_1, p_2, p_3\}$
- *Se comprueban si las transiciones de los estados contenidos en* p_0 *son homogéneas:*

δ	0	1
q_0	p_1	p_3
q_4	p_1	p_3

- *Transiciones "provisionales" de* p_0 *han cambiado:*

$$\delta_{|_E}(p_0, 0) = E[\delta(q_0, 0)] = E[q_1] = p_1$$
$$\delta_{|_E}(p_0, 1) = E[\delta(q_0, 1)] = E[q_5] = p_3$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso 6: *Segundo análisis de* $p_1 = \{q_1\}$

- *Se marca* p_1: *Nuevo* $= \{\underline{p}_0, \underline{p}_1, p_2, p_3\}$
- *El estado* p_1 *no se puede dividir porque sólo contiene a* q_1
- *Se comprueba si han cambiado las transiciones "provisionales" de* p_1*:*

$$\delta_{|E}(p_1, 0) = E[\delta(q_1, 0)] = E[q_1] = p_1$$
$$\delta_{|E}(p_1, 1) = E[\delta(q_1, 1)] = E[q_2] = p_2$$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso número 7: **Segundo** *análisis de* $p_2 = \{q_2, q_3\}$

- *Se marca* p_2: *Nuevo* $= \{\underline{p}_0, \underline{p}_1, \underline{p}_2, p_3\}$
- *Se comprueban si son homogéneas las transiciones de los estados contenidos en* p_2:

δ	0	1
q_2	p_2	p_1
q_3	p_2	p_1

- *Los estados* q_2 *y* q_3 *son equivalentes y no se requiere ninguna división.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

Paso número 8: *Análisis de* $p_3 = \{q_5, q_6\}$

- *Se marca* p_3: *Nuevo* $= \{\underline{p}_0, \underline{p}_1, \underline{p}_2, \underline{p}_3\}$
- *Se comprueban si son homogéneas las transiciones de los estados contenidos en* p_3:

δ	0	1
q_5	p_0	p_1
q_6	p_0	p_1

- *Los estados* q_5 *y* q_6 *son equivalentes y no se requiere ninguna división.*

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

- *El algoritmo* **finaliza** *porque todos los* **estados** *de Nuevo están* **marcados**.
- *Los estados generados son:*
  - $p_0 = \{q_0, q_4\}$
  - $p_1 = \{q_1\}$
  - $p_2 = \{q_2, q_3\}$
  - $p_3 = \{q_5, q_6\}$
- *El estado inicial es p_0 porque $q_0 \in p_0$.*
- $F_{|E} = \{p_1\}$ *porque* $p_1 \cap F = \{q_1\} \neq \emptyset$

Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)

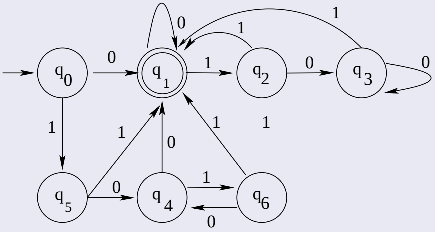
- *Función de transición del autómata minimizado*

	$\delta_{\|E}$	0	1
$\rightarrow$	p_0	p_1	p_3
$\leftarrow$	p_1	p_1	p_2
	p_2	p_2	p_1
	p_3	p_0	p_1

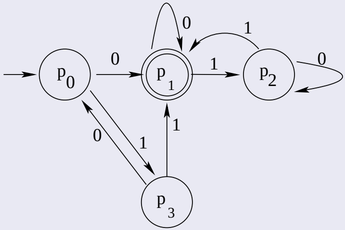
Reconocimiento de componentes léxicos

Minimización de autómatas finitos deterministas

Ejemplo (3.- Minimización de AFD: cadenas de ceros y unos)



AFN original



AFD minimizado

Contenido del tema

Contenido de la sección

Implementación de los analizadores léxicos

Introducción

Definición

Se denomina implementación de un analizador léxico a su codificación en un lenguaje de programación.

Implementación de los analizadores léxicos

Introducción

Nota

Se recuerda que el analizador léxico suele ser una función o procedimiento auxiliar del analizador sintáctico.

Implementación de los analizadores léxicos

Introducción

Tareas del analizador léxico

- Reconocer todos los componentes léxicos:
  - Palabras reservadas
  - Identificadores
  - números
  - Operadores aritméticos, relacionales, etc.
  - Etc.
- Enviar al analizador sintáctico los componentes léxicos reconocidos.
- Procesar los errores léxicos que pueda detectar.

Implementación de los analizadores léxicos

Introducción

Nota (Primer paso para implementar el analizador léxico)

Definir una expresión regular
para denotar cada componente léxico

Implementación de los analizadores léxicos

Introducción

Tipo de reconocimiento de las palabras reservadas

- Implícito
  + Se pre-instalan en la tabla de símbolos.
  + Se procesan inicialmente como identificadores.
  + Se reconocen como palabras claves al buscarlas en la tabla de símbolos.
- Explícito
  + Se reconocen de forma independiente a los identificadores
  + Siempre se utiliza el lexema más largo: if - ifa
  + En caso de igualdad de longitudes, se escoge el componente léxico que se haya especificado en primer lugar.

Implementación de los analizadores léxicos

Introducción

Métodos de implementación del Analizador Léxico

- Codificación manual utilizando un lenguaje de programación.
- Utilización de un generador automático del analizador léxico:
  + lex: lenguaje C y unix.
  + flex: lenguaje C y linux.
  + pclex: lenguaje C y MSDOS.
  + Jlex: lenguaje java y multiplataforma.

Contenido de la sección

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Características

- Se utiliza un lenguaje de programación:
  - Alto nivel: C, C++, Pascal, Java, etc.
  - Bajo nivel: ensamblador.
- Se codifica una función que combina todos los AFDs transformados.

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Lenguaje de programación de alto nivel

- Ventajas
  - + Computacionalmente eficiente.
  - + Permite la detección y recuperación específica de errores.
- Inconvenientes
  - + Requiere un gran esfuerzo de programación.
  - + Las modificaciones pueden ser dificultosas.

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Lenguaje de programación de bajo nivel

- Ventajas
  - + Computacionalmente muy eficiente: permite controlar de forma directa la Entrada / Salida.
  - + Permite la detección y recuperación específica de errores.
- Inconvenientes
  - + Requiere un esfuerzo de programación muy elevado.
  - + Las modificaciones son muy dificultosas.

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

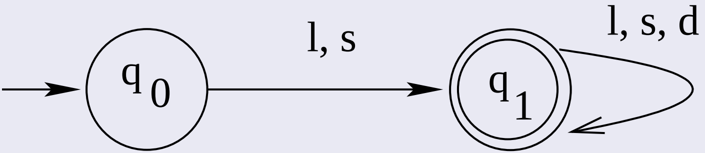
Combinación de los Autómatas Finitos Deterministas transformados

- Se transforma cada AFD para que:
  - reconozca el componente léxico
  - compruebe si es necesario procesar el símbolo que sigue al componente léxico:
    - Si el símbolo es correcto, se devuelve al "buffer" de entrada.
    - Si el símbolo es incorrecto, se procesa el error detectado.
  - devuelva el componente léxico reconocido.
  - y continúe el análisis léxico.
- Se combinan todos los Autómatas Finitos Deterministas transformados.

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (1.- Transformación de AFD: 1/3)

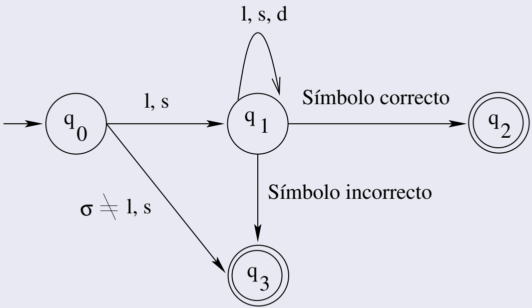


AFD que reconoce identificadores del lenguaje C.

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (1.- Transformación de AFD: 2/3)



AFD transformado.

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (1.- Transformación de AFD: 3/3)

- *El estado q_2 debe:*
  - + *Devolver el componente léxico reconocido.*
  - + *Devolver al "buffer" de entrada el símbolo correcto que no pertenece al identificador de C:*
    - *espacio en blanco*
    - *punto y coma,*
    - *etc.*
- *El estado q_3 debe procesar el error detectado.*

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Nota

Las celdas vacías de la función de transición se completan con "rutinas" de tratamiento de errores:

	δ	$\cdots$	σ_j	$\cdots$
$\rightarrow$	q_0			
	$\cdots$			
	q_i		*Error*	
	$\cdots$			

Error representa una función o procedimiento que permite el tratamiento del error detectado.

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

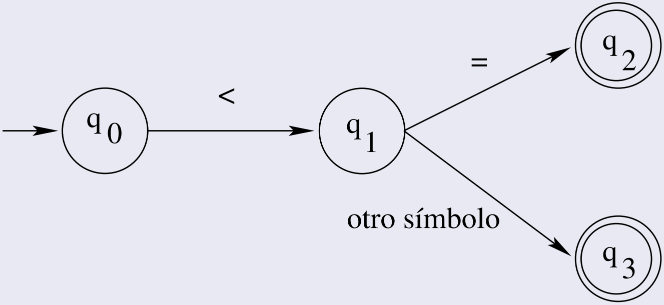
Nota

- *Algunas veces no es necesario comprobar el símbolo que sigue al componente léxico.*
- *Suele ocurrir con los componentes léxicos más simples:*
    - *Punto y coma*
    - *Espacio en blanco, tabular o salto de línea*
    - *Operadores aritméticos*
    - *Etc.*

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (2.- Transformación AFD: 1/2)



AFD transformado que reconoce los componentes léxicos MENOR o MENOR IGUAL.

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (2.- Transformación AFD: 2/2)

- q_2 *reconoce el componente léxico MENOR IGUAL*
  - *No se necesita procesar el símbolo siguiente, porque no ha sido leído.*
- q_3 *reconoce el componente léxico MENOR.*
  - *El* "otro símbolo" *debe ser devuelto al "buffer" de entrada.*

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Estrategias de la codificación manual

- Utilizar directamente las tablas de la función de transición de los AFDs.
- Simular el funcionamiento de los AFDs mediante sentencias de control.

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (Uso de la tabla de la función de transición)

INICIO ESTADO ← INICIAL
LEER (CARÁCTER)

MIENTRAS (FINAL(ESTADO) = FALSO **Y** CARÁCTER ≠ FIN-DE-FICHERO)
  HACER
    ESTADO ← M(ESTADO,CARÁCTER)
    LEER (CARÁCTER)
FIN MIENTRAS

SI ERROR(ESTADO) = VERDADERO **ENTONCES**
    PROCESAR-ERROR(ESTADO)
SI NO
    SI DEVOLVER-CARÁCTER(ESTADO) **ENTONCES**
      DEVOLVER (CARÁCTER)
    FIN SI

    COMPONENTE-LÉXICO(ESTADO)

FIN

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (Uso de la tabla de la función de transición)

```
INICIO ESTADO ← INICIAL
LEER (CARÁCTER)
MIENTRAS (FINAL(ESTADO) = FALSO Y CARÁCTER ≠ FIN-DE-FICHERO)
  HACER
   ESTADO ← M(ESTADO,CARÁCTER)
   LEER (CARÁCTER)
FIN MIENTRAS
SI ERROR(ESTADO) = VERDADERO ENTONCES
   PROCESAR-ERROR(ESTADO)
SI NO
   SI DEVOLVER-CARÁCTER(ESTADO) ENTONCES
     DEVOLVER (CARÁCTER)
   FIN SI
   COMPONENTE-LÉXICO(ESTADO)
FIN
```

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (Simulación con sentencias de control: 1/3)

```
yylex()
{
 int c;
 /* Salta blancos y tabuladores */
 while((c=getchar())==' ' || c=='\t' ); /* Sentencia nula */
 if (c == EOF)
  {
   printf("\n Fin de la ejecución de %s \n", progname);
   return 0;
  }
 else if (c == '.' || isdigit(c))
  { /* El símbolo leído se devuelve al buffer de entrada
  para leerlo como parte de un número */
   ungetc(c,stdin);
   /* Lee el número */
   scanf("%lf",&yylval.val);
   return NUMBER;
  }
```

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (Simulación con sentencias de control: 2/3)

```
else if (isalpha(c))
 { /* comprueba si lee un identificador */
  Symbol *s;
  char sbuf[100],*p=sbuf;
  do {
    *p++=c;
   } while((c=getchar()) != EOF && isalnum(c));
  /* Devuelve el símbolo que no pertenece al identificador */
  ungetc(c,stdin);
  /* Cadena correcta: carácter nulo al final */
  *p='\0';
  /* Si no está en la tabla de símbolos, lo instala */
  if ((s=lookup(sbuf))==0) s = install(sbuf,INDEFINIDA,0.0);

  yylval.sym=s;
  return s->tipo==INDEFINIDA ? VAR : s->tipo;
}
```

Implementación de los analizadores léxicos

Codificación manual del analizador léxico

Ejemplo (Simulación con sentencias de control: 3/3)

```
else if (c=='\n')
 {
  lineno++;
  return FIN;
 }
else if (c==';') return FIN;

/* Devuelve el código ASCII de los demás caracteres */
return c;

}
```

Contenido de la sección

Implementación de los analizadores léxicos

Generación automática del analizador léxico

Características de la generación automática

- Los componentes léxicos se denotan mediante expresiones regulares.
- El generador léxico crea automáticamente el código a partir de las expresiones regulares.
- Generadores léxicos: lex, flex, pclex, jlex, ...

Implementación de los analizadores léxicos

Generación automática del analizador léxico

COMPONENTES LÉXICOS

Implementación de los analizadores léxicos

Generación automática del analizador léxico

COMPONENTES LÉXICOS

$\downarrow$

EXPRESIONES REGULARES

Implementación de los analizadores léxicos

Generación automática del analizador léxico

COMPONENTES LÉXICOS

↓

EXPRESIONES REGULARES

↓

AFN

Implementación de los analizadores léxicos

Generación automática del analizador léxico

COMPONENTES LÉXICOS

↓

EXPRESIONES REGULARES

↓

AFN

↓

AFD

Implementación de los analizadores léxicos

Generación automática del analizador léxico

COMPONENTES LÉXICOS

↓

EXPRESIONES REGULARES

↓

AFN

↓

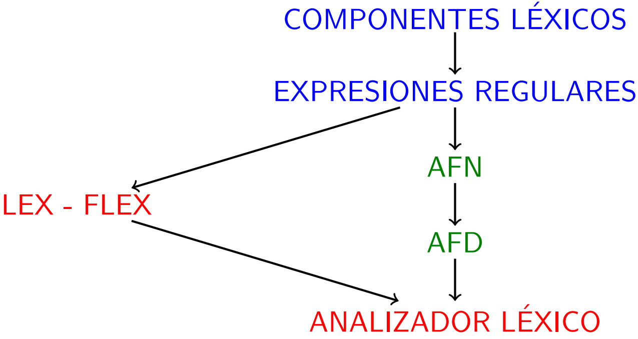
AFD

↓

ANALIZADOR LÉXICO

Implementación de los analizadores léxicos

Generación automática del analizador léxico



Implementación de los analizadores léxicos

Generación automática del analizador léxico

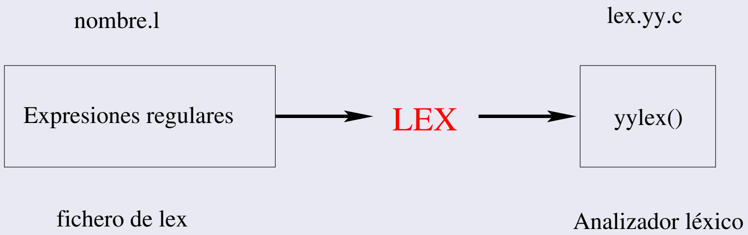
LEX

- Creado por M. E. Lesk y E. Schmidt (Bell Laboratories).
- Genera analizadores léxicos para C, Fortran, Raftor.
- Hay versiones para Unix, Linux, DOS, etc.

Implementación de los analizadores léxicos

Generación automática del analizador léxico

Funcionamiento de LEX



Implementación de los analizadores léxicos

Funcionamiento de LEX

- Unix
  - > lex nombre.l
  - > cc -g lex.yy.c -ll -o nombre.exe
- Linux
  - > flex nombre.l
  - > gcc -g lex.yy.c -lfl -o nombre.exe

Implementación de los analizadores léxicos

Ejecución

- $>$ ./nombre.exe
- Redirigiendo la entrada y la salida

  $>$ ./nombre.exe $<$ fichero_entrada

  $>$ ./nombre.exe $<$ fichero_entrada $>$ fichero_salida
- Usando argumentos desde la línea de comandos

  $>$ ./nombre.exe fichero_entrada

  $>$ ./nombre.exe fichero_entrada fichero_salida

Implementación de los analizadores léxicos

Estructura del fichero de LEX

declaraciones (opcional)

%%

reglas de traducción de las expresiones regulares

%%

funciones auxiliares (opcional)

Implementación de los analizadores léxicos

LEX: expresiones regulares

- Símbolos especiales:
  - |: disyunción
  - ( ): agrupación de expresiones regulares
  - *: repetición de un patrón cero o más veces.
  - +: repetición de un patrón una o más veces.
  - ?: el patrón puede aparecer cero o una vez.
  - “ ”: delimitadores de cadenas
  - .: cualquier carácter distinto del salto de línea (\n).
  - \n: salto de línea
  - $: carácter de final de línea
  - [ ]: delimitadores de clases de caracteres
  - ^ : inicio de línea y complementario de una clase.

Implementación de los analizadores léxicos

Nota (LEX: expresiones regulares)

Si se antepone la barra \ delante de un símbolo especial entonces sólo se representa a sí mismo:

\. $\rightarrow$ *sólo representa el punto.*

Implementación de los analizadores léxicos

Ejemplo (LEX: expresiones regulares 1 / 3)

Expresión regular	Significado
a|b	*a, b*
[ab]	*a, b*
ab	*ab*
ab+	*ab, abb, abbb,* $\cdots$
(ab)+	*ab, abab, ababab,* $\cdots$
*ab**	*a, ab, abb,* $\cdots$
*(ab)**	ϵ*, ab, abab,* $\cdots$
ab{1,3}	*ab, abb, abbb*

Implementación de los analizadores léxicos

Ejemplo (LEX: expresiones regulares 2 / 3)

Expresión regular	**Significado**
[a-z]	$a, b, c, \cdots, z$
[a \-z]	$a, -, z$
[-az]	$-, a, z$
[az-]	$a, z, -$
[a-zA-Z]	$a, b, \cdots, z, A, B, \cdots, Z$
[a-zA-Z0-9]	$a, b, \cdots, z, A, B, \cdots, Z, 0, 1, \cdots, 9$
*[a-zA-Z0-9]**	*cero o más veces* $a, b, c, \cdots, 9$
[a-zA-Z0-9]+	*una o más veces* $a, b, c, \cdots, 9$

Implementación de los analizadores léxicos

Ejemplo (LEX: expresiones regulares 3 / 3)

Expresión regular	Significado
[\t \n]	*espacio en blanco, tabulador y salto de línea*
[^ ab]	*cualquier carácter distinto de a o b*
[a ^ b]	*a, acento circunflejo, b*
a/b	*a sólo si va seguido de b*
a$	*a si va seguido del carácter \n*
a/ \n	*a si va seguido del carácter \n*
^ abc	*abc si está escrito al principio de la línea*
[\40-\176]	*caracteres ASCII imprimibles desde octal 40 (espacio) hasta octal 176 (tilde ˜)*

Implementación de los analizadores léxicos

LEX: Zona de declaraciones (opcional)

- Código extendido de lenguaje C delimitado por **%**{ y } **%**
  + Ficheros de cabecera.
  + Macros.
  + Prototipos de funciones.
  + Variables globales
  + Etc.
- Directivas de lex: %x, %a, %n %o, %p, ...
- Declaración de definiciones regulares.

Implementación de los analizadores léxicos

LEX: Zona de declaraciones (opcional)

- Directivas de lex (tablas internas):
  - %x ESTADO: permite activar reglas condicionales (véase el ejemplo del comentario).
  - %a número: cambia el número de transiciones empaquetadas.
  - %n número: cambia el número de transiciones.
  - %e número: cambia el número de nodos.
  - %p número: cambia el número de posiciones.
  - Etc

Implementación de los analizadores léxicos

Ejemplo (Definiciones regulares 1/2)

numero	`[0-9]`
letra	`[a-zA-Z]`
identificador	`{letra}({letra}|{numero})*`

Implementación de los analizadores léxicos

Ejemplo (Definiciones regulares 2/2)

La definición regular **identificador** *definida como*

$$\{letra\}(\{letra\}|\{numero\})*$$

es transformada en

$$[a-zA-Z]([a-zA-Z]|[0-9])*$$

Implementación de los analizadores léxicos

LEX: Zona de reglas de traducción

expresión regular	sentencia de lenguaje C
expresión regular	{ sentencias de lenguaje C }

Implementación de los analizadores léxicos

Ejemplo (Lex: zona de reglas)

```
%%
[ \t]           { ; } /* saltar los espacios y los tabuladores */
{numero}+\.?|{numero}*\.{numero}+   {
                                     sscanf(yytext,"%lf",&yylval.val);
                                     return NUMBER;
                                    }

\n        {lineno++; return FIN;}
.         {return yytext[0];} /* Devuelve cualquier otro carácter */
```

Implementación de los analizadores léxicos

LEX: resolución de ambigüedades

- Si una secuencia de caracteres se puede emparejar con **varias expresiones regulares**,
  1. tiene preferencia la expresión regular que denote la cadena de caracteres de **mayor longitud**.
  2. si la longitud de la cadena es igual, tiene preferencia la que aparezca en **primer lugar**.

Implementación de los analizadores léxicos

Ejemplo (Lex: resolución de la ambigüedad de **if**)

```
%%
if               { return IF; }
{identificador}  { Symbol *s;
                   if ((s=lookup(yytext)) == 0)
                       s = install (yytext, INDEFINIDA, 0.0);

                   yylval.sym = s;
                   return s->tipo == INDEFINIDA ? VAR : s->tipo;
                  }
```

Implementación de los analizadores léxicos

LEX: comandos especiales

- ECHO: imprime por pantalla el texto reconocido.
- BEGIN: cambia a un estado definido por el programador (véase el ejemplo del *comentario*).
- REJECT: rechaza el texto reconocido para que pueda ser procesado por otra regla (véase el ejemplo de *pink*).

Implementación de los analizadores léxicos

Ejemplo (ECHO)

- *Imprime por pantalla todos los caracteres*

```
%%
.|\n    ECHO;
%%
```

Equivalencia

```
%%
.|\n    printf("%s",yytext);
%%
```

Implementación de los analizadores léxicos

LEX: Zona de funciones auxiliares (opcional)

- Código de funciones auxiliares utilizadas por las reglas de traducción
- También se pueden incluir
  - Ficheros de cabecera.
  - Macros.
  - Prototipos de funciones.
  - Declaración de variables globales
  - Etc.

Implementación de los analizadores léxicos

LEX: variables globales predefinidas

- yytext: cadena que contiene el texto reconocido (tipo: char *)
- yyleng: longitud de yytext (tipo: int)
- yyin:
  - Puntero al fichero de entrada
  - Tipo: FILE *
  - Valor por defecto: **stdin**, el teclado
- yyout:
  - Puntero al fichero de salida
  - Tipo: FILE *
  - Valor por defecto: **stdout**, la pantalla

Implementación de los analizadores léxicos

LEX: funciones predefinidas

- yylex(): contiene el analizador léxico generado por flex o lex
- yymore(): indica a lex que añada el siguiente componente léxico al componente léxico actual (véase el ejemplo *hiper*).
- yywrap(): se ejecuta cuando el analizar léxico encuentra el fin de fichero:
  - Si devuelve 0, el analizador léxico continúa explorando.
  - Si devuelve 1 (valor por defecto), el analizador léxico devuelve un componente léxico nulo para indicar el fin del fichero.
  - Esta función puede ser redefinida por el programador.
- yyless(n): retiene los primeros *n* caracteres de yytext y devuelve el resto al dispositivo de lectura.

Implementación de los analizadores léxicos

Ejemplo (Lex: zona de declaraciones 1/3)

```
%{
#include "macros.h"
#include "hoc3.h"
#include "y.tab.h"
extern char *progname;
extern int lineno;
%}

/* definiciones regulares */
numero          [0-9]
letra           [a-zA-Z]
identificador   {letra}({letra}|{numero})*
```

Implementación de los analizadores léxicos

Ejemplo (Lex: zona de reglas 2/3)

```
%%
[ \t]          { ; } /* saltar los espacios y los tabuladores */
{numero}+\.?|{numero}*\.{numero}+   {
                                     sscanf(yytext,"%lf",&yylval.val);
                                     return NUMBER;
                                    }
{identificador}   { Symbol *s;
                    if ((s=lookup(yytext)) == 0)
                        s = install (yytext, INDEFINIDA, 0.0);
                    yylval.sym = s;
                    return s->tipo == INDEFINIDA ? VAR : s->tipo;
                   }
;        {return FIN ;}
\n       {lineno++; return FIN;}
.        {return yytext[0];} /* Devuelve cualquier otro carácter */
```

Implementación de los analizadores léxicos

Ejemplo (Lex: zona de funciones auxiliares 3/3)

```
/***** Zona de funciones auxiliares *****/
extern FILE *yyin, *yyout;
main(int cantidad, char *palabras[])
 {   switch(cantidad)
   {
    case 2: yyin=fopen(palabras[1],``r'');
        break;
    case 3: yyin=fopen(palabras[1],``r'');
        yyout=fopen(palabras[2],``w'');
   }
yylex();
}
```

Contenido del tema

Contenido de la sección

Detección y recuperación de errores

Clasificación de los errores del proceso de traducción

Tipos de errores

- Invisibles: errores que no pueden ser detectados por el procesador de lenguajes
- Visibles: errores que sí pueden ser detectados

Detección y recuperación de errores

Clasificación de los errores del proceso de traducción

Ejemplo (Error invisible)

Se ha tecleado

$$a \; = \; b + c$$

en vez

$$a \; = \; b * c$$

Detección y recuperación de errores

Clasificación de los errores del proceso de traducción

Errores invisibles

- Suelen ser errores **"conceptuales" o "algorítmicos"**
- **No pueden ser detectados** porque no infrigen ninguna norma del lenguaje de programación.
- Podrían ser detectados si se incluyen técnicas de verificación en el procesador de lenguajes:
  - + Poseen gran complejidad
  - + Su coste computacional es muy elevado
- En la práctica, estos errores son "detectados" y "corregidos" manualmente

Detección y recuperación de errores

Clasificación de los errores del proceso de traducción

Errores visibles

- Pueden ser **detectados**:
  - durante el proceso de traducción
  - o durante la ejecución del programa ejecutable.
- Son producidos porque no se ha tenido suficiente "cuidado" al programar:
  - **falta de comprensión o desconocimiento** de las características del lenguaje
  - o **confusión** con las características de otro lenguaje.
- Se caracterizan por
  - Ser errores de **ortografía**
  - Ser errores que **omiten** requisitos formales del lenguaje de programación.

Detección y recuperación de errores

Clasificación de los errores del proceso de traducción

Errores visibles

Se clasifican según la fase en que son detectados:

- Errores léxicos: no se reconoce un componente léxico correcto.
- Errores sintácticos:
  - Sentencia que no respeta las reglas gramaticales del lenguaje de programación
  - Se origina al procesar un componente léxico "inesperado"
- Errores semánticos: el significado de un componente léxico es incorrecto o inapropiado.
- Errores de ejecución:
  - Funcionamiento incorrecto del programa.
  - No detectables durante el proceso de compilación.

Detección y recuperación de errores

Clasificación de los errores del proceso de traducción

Ejemplos (Errores visibles)

- *Errores léxicos:*
  - + *Componente léxico mal escrito.*
  - + *Componente léxico con símbolos no permitidos.*
- *Errores sintácticos:*
  - + *Sentencias de control incompletas o mal escritas.*
  - + *Parámetros incorrectos, paréntesis no balanceados, ...*
- *Errores semánticos:*
  - + *Identificador usado incorrectamente o no declarado.*
  - + *Valor fuera de rango, ...*
- *Errores de ejecución:*
  - + *Bucles infinitos.*
  - + *Flujo de control incorrecto, ...*

Contenido de la sección

Detección y recuperación de errores

Errores léxicos

Tipos de errores léxicos

Componentes léxicos

- con símbolos ilegales.
- incompletos o muy largos.
- mal escritos: infringen alguna restricción del lenguaje de programación

Detección y recuperación de errores

Errores léxicos

Ejemplos (Errores léxicos 1/3)

- *Componentes léxicos con símbolos ilegales*
  - *Símbolo no permitido en un identificador:* **dato$**
  - *Símbolo no permitido en un número:* **3.17#10**

Detección y recuperación de errores

Errores léxicos

Ejemplos (Errores léxicos 2/3)

- *Componentes léxicos incompletos*
  - *Cadenas de caracteres* **no** *cerradas*

```
/* faltan las comillas finales */
static char *nombre = ``Almudena ... ;
```

  - *Comentarios* **no** *cerrados.*

```
/* Este es un ejemplo maravilloso de comentario
que no tiene cierre final
```

Detección y recuperación de errores

Errores léxicos

Ejemplos (Errores léxicos 3/3)

- *Componentes léxicos mal escritos*

Lenguaje	*Componente léxico*	*Incorrecto*	*Correcto*
C	*IDENTIFICADOR*	*1dato-*	*dato_1*
C	*NÚMERO*	*3.14.15*	*3.1415*
C	*MENOR IGUAL*	=<	<=
Prolog	*MENOR IGUAL*	<=	=<
Fortran	*MENOR IGUAL*	*.le*	*.le.*

Detección y recuperación de errores

Errores léxicos

Características de los errores de los componentes léxicos

- Son provocados por una **escritura incorrecta**.
- Son los más **frecuentes**.
- El Análisis Léxico **no puede detectar** todos los errores de los componentes léxicos.
- Muchos **errores** de los componentes **léxicos** son **detectados** durante **las demás fases.**
- Pueden **provocar errores** sintácticos, semánticos o de ejecución.

Detección y recuperación de errores

Errores léxicos

Ejemplo (1.- Error no detectable durante el análisis léxico)

Palabra reservada mal escrita

```
fi (x >= 0) valor_absoluto = x;
else valor_absoluto = -x;
```

Nota

- *El análisis léxico genera para fi el componente léxico IDENTIFICADOR en vez de IF.*
- *Este error podrá ser detectado durante el análisis sintáctico*

Detección y recuperación de errores

Errores léxicos

Ejemplo (1.- Error no detectable durante el análisis léxico)

Palabra reservada mal escrita

```
fi (x >= 0) valor_absoluto = x;
else valor_absoluto = -x;
```

Nota

- *El análisis léxico genera para fi el componente léxico IDENTIFICADOR en vez de IF.*
- *Este error podrá ser detectado durante el análisis sintáctico*

Detección y recuperación de errores

Errores léxicos

Ejemplo (2.- Error no detectable durante el análisis léxico)

Omisión de símbolo de fin de sentencia

```
/* falta el punto y coma */
if (x >= 0) valor_absoluto = x
  else valor_absoluto = -x;
```

Nota

Este error puede ser detectado durante el análisis sintáctico.

Detección y recuperación de errores

Errores léxicos

Ejemplo (2.- Error no detectable durante el análisis léxico)

Omisión de símbolo de fin de sentencia

```
/* falta el punto y coma */
if (x >= 0) valor_absoluto = x
  else valor_absoluto = -x;
```

Nota

Este error puede ser detectado durante el análisis sintáctico.

Detección y recuperación de errores

Errores léxicos

Ejemplo (3.- Error no detectable durante el análisis léxico)

Índice de un array que posee un valor que fuera de rango

```
a[-999999] = 10;
```

Nota

Este error puede ser detectado durante el análisis semántico.

Detección y recuperación de errores

Errores léxicos

Ejemplo (3.- Error no detectable durante el análisis léxico)

Índice de un array que posee un valor que fuera de rango

```
a[-999999] = 10;
```

Nota

Este error puede ser detectado durante el análisis semántico.

Detección y recuperación de errores

Errores léxicos

Ejemplo (4.- Error no detectable durante el análisis léxico)

Argumento erróneo de una función

```
valor = log(-10);
```

Nota

- *Este error se detectará durante la ejecución.*
- *Podría ser detectado con técnicas de verificación*

Detección y recuperación de errores

Errores léxicos

Ejemplo (4.- Error no detectable durante el análisis léxico)

Argumento erróneo de una función

```
valor = log(-10);
```

Nota

- *Este error se detectará durante la ejecución.*
- *Podría ser detectado con técnicas de verificación*

Contenido de la sección

Detección y recuperación de errores

Tratamiento de los errores

Criterios "generales" para el tratamiento de los errores

- Informar del error
  - Localización: ubicación del error dentro del código.
  - Descripción: motivo o características del error
- Evitar la cascada de errores
  - Informar una sola vez de cada error.
  - Si un error es producido por otro entonces sólo se debe informar del primero.
- Reparar el error, si es posible, e informar de la corrección realizada.
- Continuar con el proceso de traducción para detectar otros posibles errores.

Detección y recuperación de errores

Tratamiento de los errores

Criterios "generales" para el tratamiento de los errores

- Informar del error
  - Localización: ubicación del error dentro del código.
  - Descripción: motivo o características del error
- Evitar la cascada de errores
  - Informar una sola vez de cada error.
  - Si un error es producido por otro entonces sólo se debe informar del primero.
- Reparar el error, si es posible, e informar de la corrección realizada.
- Continuar con el proceso de traducción para detectar otros posibles errores.

Detección y recuperación de errores

Tratamiento de los errores

Criterios "generales" para el tratamiento de los errores

- Informar del error
  - Localización: ubicación del error dentro del código.
  - Descripción: motivo o características del error
- Evitar la cascada de errores
  - Informar una sola vez de cada error.
  - Si un error es producido por otro entonces sólo se debe informar del primero.
- Reparar el error, si es posible, e informar de la corrección realizada.
- Continuar con el proceso de traducción para detectar otros posibles errores.

Detección y recuperación de errores

Tratamiento de los errores

Criterios "generales" para el tratamiento de los errores

- Informar del error
  - Localización: ubicación del error dentro del código.
  - Descripción: motivo o características del error
- Evitar la cascada de errores
  - Informar una sola vez de cada error.
  - Si un error es producido por otro entonces sólo se debe informar del primero.
- Reparar el error, si es posible, e informar de la corrección realizada.
- Continuar con el proceso de traducción para detectar otros posibles errores.

Detección y recuperación de errores

Tratamiento de los errores

Nota

Los criterios **"generales"** *para el tratamiento de los errores son aplicables a* **todos** *los tipos de errores.*

Detección y recuperación de errores

Tratamiento de los errores

Nota (Reparación del error)

- **Siempre** *debe ser revisada después por el programador.*
- *Sólo* **propone** *una solución, que no tiene por qué ser la correcta.*
- *Sólo pretende que el proceso de traducción* **continúe** *... para detectar más errores.*

Contenido de la sección

Detección y recuperación de errores

Métodos de recuperación de los errores léxicos

Métodos de procesamiento de los errores léxicos

- Modo de **pánico o de sincronización**
- Método de **la mínima distancia**

Detección y recuperación de errores

Métodos de recuperación de los errores léxicos

Modo de pánico o de sincronización

- Se **eliminan** los caracteres de la entrada hasta que se encuentra un componente léxico bien formado
- Es **sencillo** de aplicar
- Puede **provocar** errores en el análisis sintáctico al suprimir otros componentes léxicos correctos.
- Es útil en un entorno **interactivo**.

Detección y recuperación de errores

Métodos de recuperación de los errores léxicos

Método de la mínima distancia

- **Supone** que la mayoría de los errores léxicos son provocados por una **única** transformación.
- Se realizan transformaciones relativamente simples:
  + **Eliminar** un carácter: dato$1 $\Longrightarrow$ dato1
  + **Insertar** un carácter: include $\Longrightarrow$ #include
  + **Permutar** dos caracteres: => $\Longrightarrow$ >=
  + **Modificar** un carácter: /# $\Longrightarrow$ /*
- Es **costoso** de implementar.
- Es adecuado para un entorno local o **interactivo**.

Detección y recuperación de errores

Métodos de recuperación de los errores léxicos

Nota

- *Se debe informar de la transformación realizada.*
- *La transformación no pretende corregir el error, sino continuar con el análisis léxico.*
- *Posteriormente, el programador deberá supervisar la transformación realizada.*

PROCESADORES DE LENGUAJES

TEMA II.- ANÁLISIS LÉXICO

Prof. Dr. Nicolás Luis Fernández García

Departamento de Informática y Análisis Numérico
Escuela Politécnica Superior de Córdoba
Universidad de Córdoba