Repaso. Lenguajes formales

Profesor
Federico Peinado

Elaboración del material
José Luis Sierra
Federico Peinado

Ingeniería en Informática
Facultad de Informática – Universidad Complutense de Madrid
Curso 2009-2010

Lenguajes formales

- Un **lenguaje formal** es un conjunto (finito o infinito) de cadenas finitas de símbolos primitivos
  - Ej: El lenguaje “Número” es simplemente el conjunto infinito de cadenas finitas formadas con los dígitos *0, 1, 2, 3, 4, 5, 6, 7, 8 y 9*
- Dichas cadenas están formadas gracias a un **alfabeto** y a una **gramática** que están formalmente especificados
  - El alfabeto es un *conjunto finito no vacío* de símbolos
  - La gramática es un *conjunto finito* de reglas para formar cadenas finitas juntando símbolos del alfabeto
  - A cada cadena de símbolos de un lenguaje formal se le llama **fórmula bien formada** (o palabra) del lenguaje

Clasificación de gramáticas formales

- Chomsky clasificó jerárquicamente las gramáticas formales que generan lenguajes formales, en estos tipos:
  - **Tipo 3**: Gramáticas **regulares** que generan lenguajes regulares
  - **Tipo 2**: Gramáticas **incontextuales** que generan lenguajes incontextuales
  - **Tipo 1**: Gramáticas **contextuales** que generan lenguajes contextuales
  - **Tipo 0**: Gramáticas **libres** que generan lenguajes sin ningún tipo de restricción
- Cuanto menor es el tipo, mayor es el *poder expresivo* del lenguaje generado y más complejidad tiene su *tratamiento por parte de una máquina*

- Un **lenguaje regular** es un lenguaje formal que tiene estas características:
  - Puede ser <u>descrito</u> mediante una **expresión regular** (expresar de forma compacta cómo son todas las cadenas de símbolos que le pertenecen)
  - Puede ser <u>generado</u> mediante una **gramática regular** (obtener todas las cadenas de símbolos que le pertenecen)
  - Puede ser <u>reconocido</u> mediante un **autómata finito** (saber si una cadena de símbolos pertenece a él o no)

- *¡Todas estas características facilitan mucho su tratamiento computacional, por eso nos interesan los lenguajes regulares!*

Expresiones regulares (ERs)

- El conjunto de **expresiones regulares** sobre un alfabeto A se denomina **ER(A)** y sólo contiene expresiones formadas mediante estas reglas:
  - $\varnothing \in$ ER(A) y denota el lenguaje {}, siendo $\varnothing$ el vacío
  - $\lambda \in$ ER(A) y denota el lenguaje $\{\lambda\}$, siendo λ la cadena vacía
  - Si $x \in A$, $x \in$ ER(A) y denota el lenguaje {x}
  - Si H$\in$ ER(A) y K $\in$ ER(A), con lenguajes denotados L_H y L_K
    - (H | K) $\in$ ER(A) y denota el lenguaje $L_H \cup L_K$
      (Conjunto de todas las cadenas de H o de K)
    - (HK) $\in$ ER(A) y denota el lenguaje L_{HK} siendo
      $L_{HK} = \{hk \text{ tal que } h \in L_H \text{ y } k \in L_K\}$
      (Conjunto de todas las concatenaciones posibles de una cadena de H y otra de K)
    - H* $\in$ ER(A) y denota el lenguaje L_{H^*} siendo
      $L_{H^*} = \{\lambda\} \cup \{a\alpha \text{ tal que } a \in L_H \text{ y } \alpha \in L_{H^*}\}$
      (Conjunto de todas las concatenaciones sucesivas posibles de cadenas de H)
    - Los paréntesis **()** asocian operadores a cadenas de símbolos. Si no aparecen, repetir ***** es más prioritario que concatenar y concatenar más prioritario que alternar |

Extensiones a la notación de las expresiones regulares

- Estas **extensiones** no amplían la expresividad, pero hacen *mucho más cómodo* expresar lenguajes con ERs
  - $H? \equiv H \mid \lambda$ (Se llama opcionalidad **?** y tiene la misma prioridad que repetición *)
  - $H+ \equiv H(H)^*$ (Se llama cierre positivo **+** y tiene la misma prioridad que repetición *)
  - $[x_n - x_m] \equiv x_n \mid x_{n+1} \mid x_{n+2} \mid \ldots \mid x_m$ (Se llama rango **[-]** y sólo se usa para alfabetos *totalmente ordenados*)
    - *Ej: Intervalos de números naturales como [3-9] o de caracteres ASCII como [f-m]*
  - \x ≡ carácter x (Se llama escape de metacaracteres **** y se usa para incluir en el lenguaje definido caracteres que actúan como operadores, es decir son *metacaracteres*, en las expresiones regulares)
    - *Ej: * para denotar el carácter **
    - *Ej: \\\\ para denotar el carácter *
    - *…*

Ejemplos de expresiones regulares

- Descripción del lenguaje de las cadenas que empiezan por una “a” y continúan con “a’s” y “b’s”
  - a(a|b)*
- Descripción del lenguaje de las cadenas que empiezan por “a”, continúan con “b’s” y “c’s” y terminan en “d”
  - a(b|c)*d
- Descripción del lenguaje de las cadenas formadas por *trozos de cadena* que pueden empezar (o no) por una “a” y continúan con un número que tenga al menos un dígito; además terminan siempre en asterisco *
  - (a?[0-9]+)**

- Las **gramáticas formales** se definen con una *tupla* $\langle T, N, n_0, P \rangle$ siendo:
  - T el alfabeto de símbolos **terminales**
    (Símbolos que forman parte directamente de las cadenas del lenguaje)
  - N el alfabeto de símbolos **no terminales**
    (Símbolos más abstractos que representan posibles partes de las cadenas del lenguaje)
  - $n_0 \in N$, el no terminal inicial o **axioma**
  - P el conjunto de reglas de producción o **producciones** de la gramática
    - Puede representarse como $\boldsymbol{\alpha_x\ n\ \alpha_y \rightarrow \beta}$ ó $\boldsymbol{\alpha_x\ n\ \alpha_y ::= \beta}$ donde $n \in N$ y $\alpha_x, \alpha_y, \beta \in (T \cup N)^*$

Gramáticas regulares

- Las **gramáticas regulares** son de uno de estos dos tipos:
  - Son **gramáticas regulares a derechas**, es decir, todas sus producciones siguen una de estas tres formas:
    - $n \rightarrow \lambda$
    - $n \rightarrow t$
    - $n \rightarrow t\ n'$

    donde $t \in T$ y $n, n' \in N$
  - Son **gramáticas regulares a izquierdas**, es decir, todas sus producciones siguen una de estas tres formas:
    - $n \rightarrow \lambda$
    - $n \rightarrow t$
    - $n \rightarrow n'\ t$

    donde $t \in T$ y $n, n' \in N$

- Los **autómatas finitos** se definen con una *tupla* $<E, e_o, A, t, F>$ siendo:
  - E el conjunto *finito* y *no vacío* de **estados posibles**
  - $e_o \in E$, el **estado inicial** del autómata
  - A el **alfabeto** de entrada que acepta el autómata
  - t, la **función de transición de estados**
  - $F \subseteq E$, el conjunto de **estados finales**
- Hay dos tipos de autómatas finitos
  - **Autómatas finitos deterministas** (AFDs)
    - $t \in E \times A \rightarrow E$

      (Con cada símbolo de entrada se pasa de un estado del autómata a otro)
  - **Autómatas finitos no deterministas** (AFNDs)
    - $t \in E \times (A \cup \{\lambda\}) \rightarrow \wp(E)$

      (Con algún símbolo de entrada, o con la cadena vacía, se pasa de uno de los estados del autómata a otro conjunto no vacío de estados - $\wp$ significa partición -)

Comportamiento de un autómata finito

- Sirve para **reconocer cadenas de símbolos** de un lenguaje regular, para lo que:
  1. Parte del estado inicial
  2. Recibe *uno a uno* los símbolos de la cadena de entrada
     - En un AFND este paso a veces se ignora, pudiendo ocurrir una **transición espontánea** (λ-transición)
  3. Aplica la función de transición para cambiar su estado
     - Un AFND puede estar *en varios estados a la vez*
  4. Si quedan símbolos por procesar, vuelve al paso 2
  5. Si no quedan símbolos por procesar…
     - Si se ha alcanzado algún estado final la cadena es reconocida como ***perteneciente*** al lenguaje [Fin]
     - Si *no* se ha alcanzado ningún estado final la cadena es rechazada por ser ***no perteneciente*** al lenguaje [Fin]

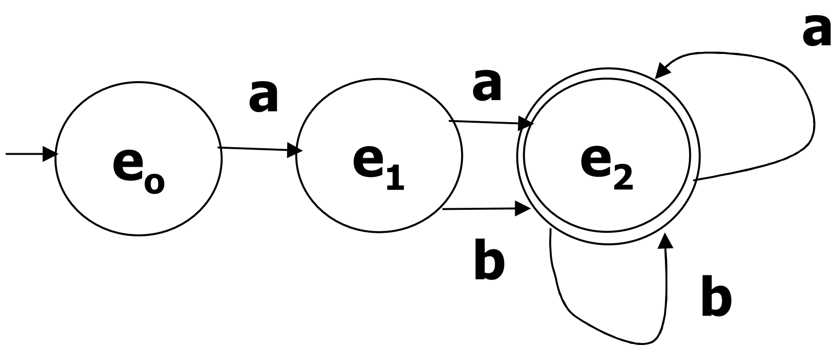
Notaciones para autómatas finitos

- Tabla de transición de estados

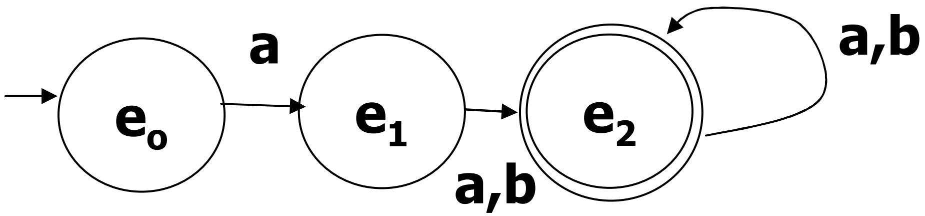
Estados/Símbolos	x	y	…
$\mathbf{e_o}$	e_i	-	…
$\mathbf{e_1}$	e_j	e_k	…
…	…	…	…

- La **permanencia** en un estado se representa como una transición de un estado otra vez al mismo estado
- Se pueden dejar **transiciones sin definir** (función de transición de estados parcial), dejando huecos o con guión **-**
  - Significa saltar a un *estado final de error* y rechazar la cadena por ser ***no perteneciente*** al lenguaje

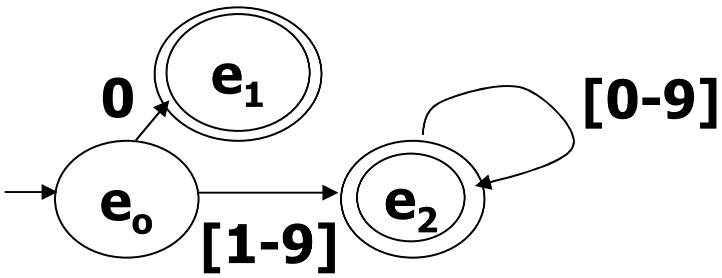
- Diagrama de transición de estados



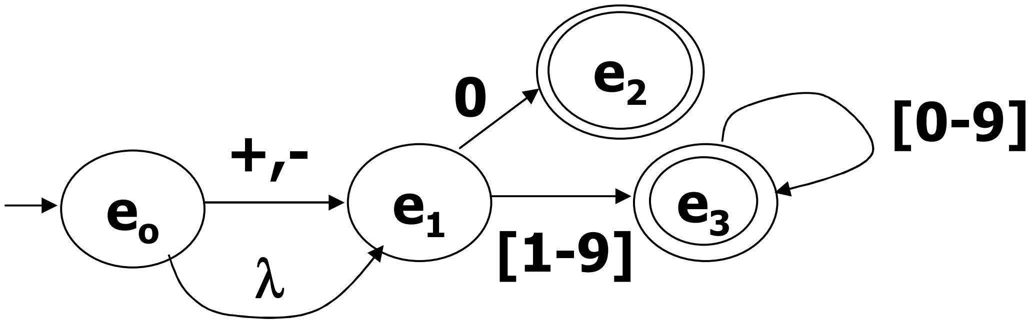
- Si varias transiciones van de un mismo estado inicial a un mismo estado final, con distintos símbolos, se pueden *juntar*



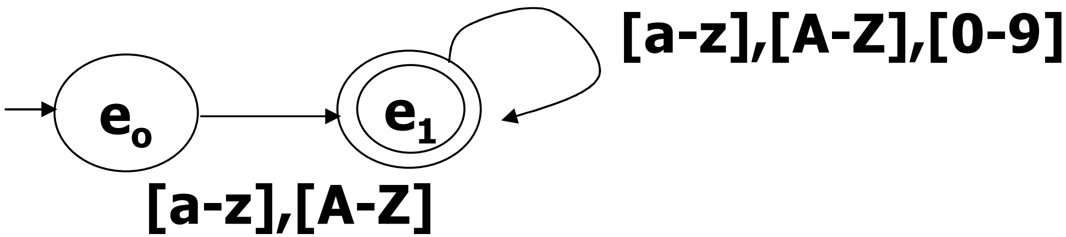
Ejemplos de autómatas finitos



Números naturales



Números naturales con signo



Identificadores (básicos)

- Las expresiones regulares (ERs), las gramáticas regulares, los autómatas finitos deterministas (AFDs) y los no deterministas (AFNDs) son **equivalentes en cuanto a expresividad**
  - Las ERs describen, las gramáticas regulares generan y los AFs permiten reconocer **cualquier lenguaje regular**
  - Existen *demostraciones forma*les para convertir de ER a AFND, de AFND a AFD y de AFD a ER, así como para comparar las ERs con las gramáticas regulares

Lenguajes incontextuales

- Un **lenguaje incontextual** es un lenguaje formal que tiene estas características:
  - Puede ser <u>generado</u> mediante una **gramática incontextual** (obtener todas las cadenas de símbolos que le pertenecen)
  - Puede ser <u>reconocido</u> mediante un **autómata con pila** (saber si una cadena de símbolos pertenece a él o no)

- *Aunque son más complejos que los regulares, estas características facilitan su tratamiento computacional, por eso también nos interesan los lenguajes incontextuales*

- Las **gramáticas incontextuales** tienen producciones **n** $\rightarrow \alpha$ donde n $\in$ N y $\alpha \in$ (T $\cup$ N)*

- Ejemplo: El lenguaje de los números binarios

```
Gbin

Terminales: 0, 1
No terminales: bits, bit
Axioma: bits
Producciones:
  bits → bits bit
  bits → bit
  bit  → 0
  bit  → 1
```

- Las cadenas $\in$ $(T \cup N)^*$ se llaman **formas sentenciales**
- Las cadenas $\in$ T^* se llaman **sentencias** o frases
- Una gramática incontextual G genera en $(T \cup N)^*$ **relaciones de derivación inmediata** $\Rightarrow_G$
  - $\alpha \Rightarrow_G \beta$ (β es derivable inmediatamente de α) si y sólo si se dan estas tres condiciones:
    - $\alpha \equiv \alpha_o n \alpha_1$
    - $\beta \equiv \alpha_o \beta_0 \alpha_1$
    - $n \rightarrow \beta_0 \in P$

    siendo α_o, α_1 y $\beta_0 \in (T \cup N)^*$ y $n \in N$
- Las **relaciones de derivación** $\Rightarrow_G$***** son el resultado de hacer el *cierre reflexivo-transitivo* de $\Rightarrow_G$
  - $\alpha \Rightarrow_G^* \beta$ (β es derivable de α) si y sólo si existe una *secuencia de derivaciones inmediatas* que nos permita llegar a β partiendo de α

Ejemplo de derivación

G_{bin}

`bits` ⇒ `bits bit` ⇒ `bits bit bit` ⇒
`bit bit bit` ⇒ `1 bit bit` ⇒ `1 0 bit` ⇒ `1 0 1`

Más sobre gramáticas formales (incontextuales)

- El **lenguaje generado** por una gramática G es L(G)

  $L(G) = \{\alpha \text{ tal que } \alpha \in T^* \text{ y } n_0 \Rightarrow^*_G \alpha\}$

  (Todas las posibles sentencias derivables del axioma de la gramática n_0)

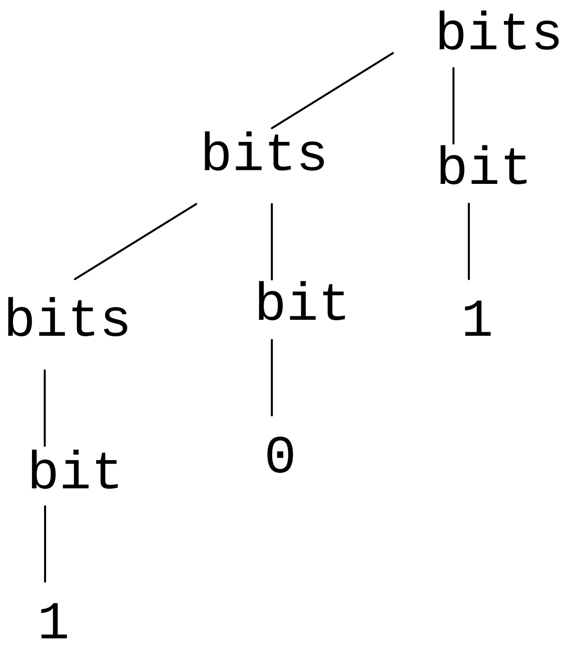
- Si $\alpha \Rightarrow_G^* \beta$ puede haber sólo una secuencia de derivación inmediata (**derivación de β desde α**) o varias posibles
  - Cada una se representa así: $\alpha \Rightarrow_G \alpha_o$, $\alpha_o \Rightarrow_G \alpha_1$, …, $\alpha_n \Rightarrow_G \beta$
  - O en forma compacta: $\alpha \Rightarrow_G \alpha_o \Rightarrow_G \alpha_1 \Rightarrow_G \ldots \Rightarrow_G \beta$
  - **Derivación *por la izquierda***: aquella en la que cada derivación inmediata $\alpha_x \Rightarrow_G \alpha_y$ usa la producción $n \rightarrow \beta_0 \in P$ para el n situado lo más a la izquierda posible en α_x
  - **Derivación *por la derecha***: aquella en la que cada derivación inmediata $\alpha_x \Rightarrow_G \alpha_y$ usa la producción $n \rightarrow \beta_0 \in P$ para el n situado lo más a la derecha posible en α_x

Árboles de derivación

- Un **árbol de derivación** de una gramática G cumple:
  - Los nodos están *etiquetados* con elementos $\in$ T $\cup$ N $\cup$ {λ}
  - Los hijos de los nodos están *ordenados*
  - Se forma de la siguiente manera:
    - Un único nodo etiquetado con n_0 es árbol de derivación
    - Si X es árbol de derivación, h uno de sus nodos hoja etiquetado con n $\in$ N y n $\rightarrow$ α una de sus producciones, se puede construir otro árbol de derivación X' así:
      - Si α es λ,se añade a h un hijo etiquetado con λ
      - Si no, se añaden a h tantos hijos como símbolos tenga α, etiquetados en orden con dichos símbolos
  - Si todas sus hojas están etiquetadas con λ o terminales, el árbol de derivación se llama **estructura** de la sentencia formada al concatenar las etiquetas de dichas hojas
    - Todas las sentencias tienen **al menos una estructura**
    - Una sentencia *puede* tener **varias estructuras distintas**
      - Si ocurre en algún caso, ¡**<u>la gramática G es ambigua!</u>**

Ejemplo de árbol de derivación (estructura)

$\mathbf{G_{bin}}$



Ejemplo de gramática incontextual ambigua

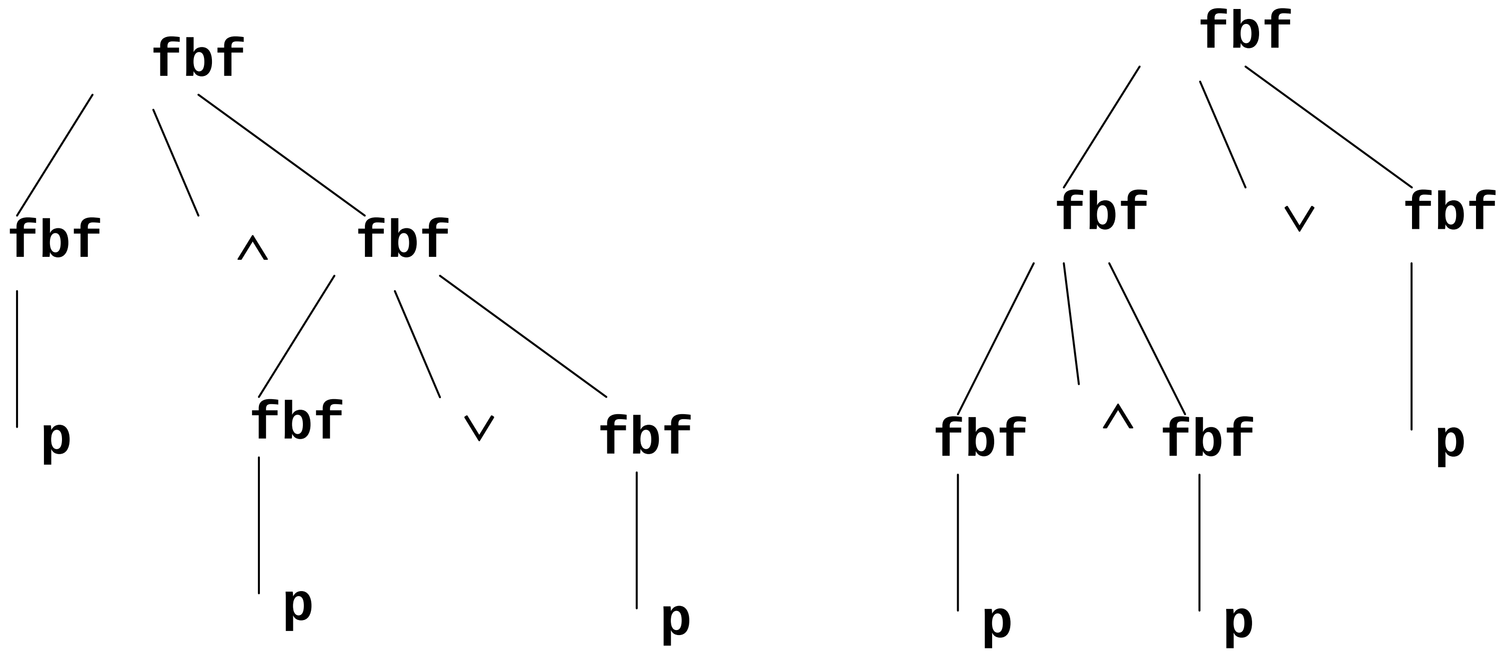
- Ejemplo: El lenguaje de las fórmulas bien formadas (fbf) de la lógica de primer orden
  - p representa cualquier predicado booleano

G_{fbf}

```
Terminales: p, ∧, ∨, ¬
No terminales: fbf
Axioma: fbf
Producciones:
  fbf → p
  fbf → ¬ fbf
  fbf → fbf ∧ fbf
  fbf → fbf ∨ fbf
```

Ejemplo de gramática incontextual ambigua

p ∧ p ∨ p



Basta con encontrar dos estructuras distintas para una misma sentencia de un lenguaje para demostrar que su gramática es ambigua

- Los **autómatas a pila** se definen con una *tupla* $<A, P, E, p_o, e_o, t, F>$ siendo:
  - A el **alfabeto de entrada** que acepta el autómata
  - P el **alfabeto de la pila**
  - E el conjunto *finito* y *no vacío* de **estados posibles**
  - $p_o \in P$, el **símbolo inicial** de la pila
  - $e_o \in E$, el **estado inicial** del autómata
  - t, la **función de transición de estados**
    - $t \in E \times (A \cup \{\lambda\}) \times P \rightarrow \wp(E \times P^*)$

      (Con algún símbolo de entrada, o con la cadena vacía, y teniendo en cuenta la cima de la pila, se pasa de uno de los estados del autómata a otro conjunto no vacío de estados, sustituyendo el símbolo de la cima de la pila por otros símbolos del alfabeto de la pila → el símbolo más a la derecha de ellos es la cima de la pila)
  - $F \subseteq E$, el conjunto de **estados finales**
- Comportamiento similar al de un autómata finito:
  - El comportamiento es *no determinista*, salvo que t siempre lleve a un único estado

Ejemplo de autómata a pila

AP_{bin}(equivalente a G_{bin})

$A = \{0,\ 1\}$
$P = \{0,\ 1,\ \mathrm{bits},\ \mathrm{bit},\ \$\}$
$E = \{x_0,\ x_1,\ x_2\}$
$p_0 = \$$
$e_0 = x_0$
Definición parcial de t:
$t(<x_o,\lambda,\$>) = \{<x_1,\$\ \mathrm{bits}>\}$
$t(<x_1,\lambda,\mathrm{bits}>) = \{<x_1,\mathrm{bit}>,\ <x_1,\mathrm{bit\ bits}>\}$
$t(<x_1,\lambda,\mathrm{bit}>) = \{<x_1,0>,\ <x_1,1>\}$
$t(<x_1,0,0>) = \{<x_1,\lambda>\}$
$t(<x_1,1,1>) = \{<x_1,\lambda>\}$
$t(<x_1,\lambda,\$>) = \{<x_2,\$>\}$
$F = \{x_2\}$

Documentación técnica de un lenguaje de programación

- Conjunto de reglas que especifican y permiten verificar la corrección de las sentencias del lenguaje y que están *orientadas a los programadores* que quieren conocer con exactitud su sintaxis (principalmente)
  - La notación gramatical es útil desde el punto de vista del desarrollador de procesadores de lenguaje, pero no desde el punto de vista de sus usuarios

- Formalismos más utilizados por ser compactos o visuales:
  - **Notación BNF** (Backus-Naur Form)
  - **Notación EBNF** (Extended Backus-Naur Form)
  - **Diagramas sintácticos**

- Todos ellos pueden expresar cualquier *lenguaje incontextual* (la base de la sintaxis de cualquier lenguaje de programación) así que *podemos usar el que queramos*

Notación BNF

- **TERMINAL** Símbolo (*ej. una palabra*) del lenguaje a definir
  (se escribe en letras mayúsculas)

- **<no terminal>** Símbolo que se define en términos de otros símbolos (tanto terminales como no terminales)
  (se escribe en letras minúsculas y entre <>)

- **Regla de producción** Descripción de un símbolo no terminal como equivalente a 1) una combinación de terminales y no terminales, o 2) a la *cadena vacía*

  (Un mismo no terminal puede tener varias reglas de producción)

- **Metasímbolo** Símbolo propio de la notación BNF, está reservado y no puede utilizarse en ningún otro símbolo
  - **::=** Equivalencia
    (lo de la izquierda *equivale* a lo de la derecha; es una regla de producción)
  - **|** Alternativa
    (lo de la izquierda *o* lo de la derecha)

Ejemplo en notación BNF

- Sintaxis de los números enteros positivos en notación BNF

<numero entero> ::= <signo opcional> <secuencia dígitos>

<signo opcional> ::= + | <nada>

<secuencia dígitos> ::= <dígito> | <dígito> <secuencia dígitos>

<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<nada> ::=

Notación EBNF

- Añade metasímbolos nuevos y cambia la forma de presentar las cosas

BNF	**EBNF**
TERMINAL	"terminal"
<no terminal>	No-terminal
Metasímbolo **::=** Equivalencia **|** Alternativa	Metasímbolo **::=** Equivalencia **|** Alternativa **(…)** Agrupación **[…]** Aparición opcional **{…}** Aparición 0, 1 o más veces (*son ERs a la derecha de cada producción*)
Recursividad permitida	Recursividad NO permitida (*se suple con {…}*) Si algún símbolo del lenguaje coincide con un metasímbolo, el símbolo del lenguaje se pone entre 'comillas simples'

Ejemplo en notación EBNF

- Sintaxis de los números enteros positivos en notación EBNF

Numero-entero ::= [Signo] Secuencia-dígitos

Signo ::= "+"

Secuencia-dígitos ::= Dígito {Dígito}

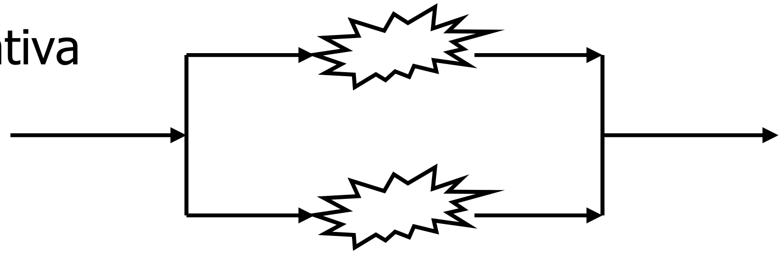
Dígito ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

TERMINAL

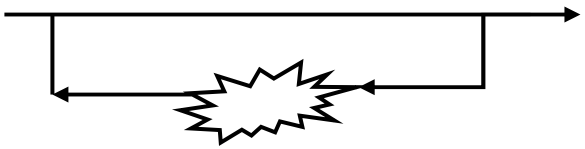
No Terminal

**En las reglas de producción el no terminal de la izquierda se deja sin recuadro*

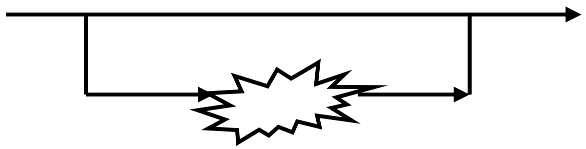
Alternativa



Aparición 0, 1 o más veces

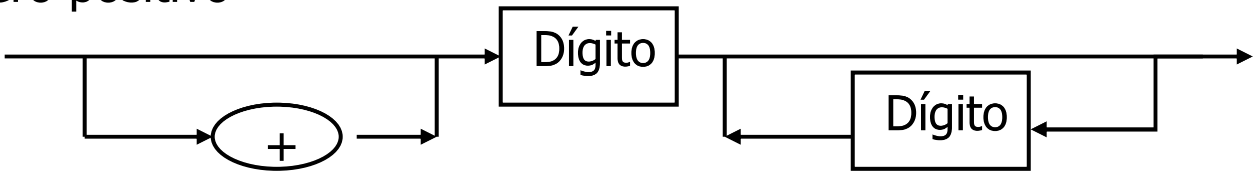


Aparición opcional

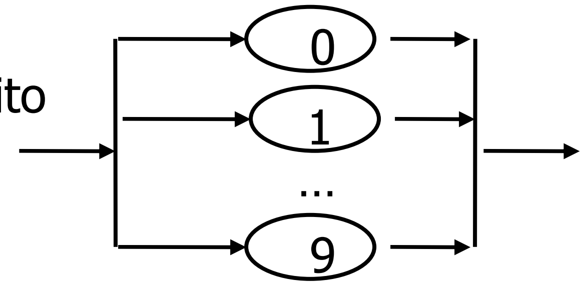


- Sintaxis de los números enteros positivos en notación de diagramas sintácticos

Nº entero positivo



Dígito



Críticas, dudas, sugerencias...

Federico Peinado
www.federicopeinado.es

- Maneras más organizadas de expresar una gramática incontextual *(¡recomendables para no cometer errores en la definición del lenguaje!)*

- **Forma normal de Chomsky**
  - Gramática incontextual G con todas sus producciones expresadas según una de estas dos fórmulas:
    - $n \rightarrow n_o n_1$
    - $n \rightarrow t$

    siendo $t \in T$ y $n, n_o, n_1 \in N$

- **Forma normal de Greibach**
  - Gramática incontextual G con todas sus producciones expresadas según esta fórmula:
    - $n \rightarrow t\,\alpha$

    siendo $t \in T$ y $\alpha \in N^*$