Lenguajes Formales

Elvira Mayordomo

Noviembre 2003

DEPARTAMENTO DE INFORMATICA E
INGENIERIA DE SISTEMAS

Tema 1: Expresiones regulares

- Definición
- Ejemplos
- Notaciones

Bibliografía de los temas 1, 2 y 3

Ullman, J.D., Hopcroft, J.E., and Motwani, R. (2001). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

Lewis, H. and Papadimitriou, C. (1981). *Elements of the Theory of Computation.* Prentice-Hall.

Apuntes de Joaquín Ezpeleta para la asignatura de *Compiladores I*, CPS, Universidad de Zaragoza.

Apuntes de Elvira Mayordomo para el curso de doctorado *Introducción al Procesamiento de Lenguaje Natural*, DIIS, Universidad de Zaragoza.

Jurafsky, D. and Martin, J.H. (2000). *Speech and Language Processing*. Prentice Hall.

Expresiones regulares. Definiciones

alfabeto

conjunto finito de símbolos

ejemplos

{0,1}, letras y dígitos

cadena (palabra o string)

secuencia finita de elementos del alfabeto

ejemplos

0010

estadoInicial

v_0

Expresiones regulares. Definiciones

longitud de una cadena

número de símbolos que la componen

ejemplos

|hola| = 4

|123456| = 6

ε = 0 (cadena vacía)

lenguaje

dado un alfabeto, cualquier conjunto de cadenas formadas con dicho alfabeto

ejemplo

Siendo Σ={0,1} {0,111,011,0111,01111,1}

Expresiones regulares. Definiciones

Algo sobre cadenas:

- concatenación
  - `c1=hola, c2=Colega`

    `c1c2=holaColega`
  - ε es el neutro, tando a derecha como a izquierda:

    $\varepsilon c = c\varepsilon = c$
  - $c^0=e$, $c^1=c$, $c^2=cc$, $c^3=ccc$,....
- terminología
  - prefijo, sufijo
  - subcadena

Expresiones regulares. Definiciones

Operadores sobre lenguajes:

- Sean L,M dos lenguajes

unión de lenguajes

$$L \cup M = \{c \mid c \in L \text{ ó } c \in M\}$$

concatenación de lenguajes

$$LM = \{st \mid s \in L \text{ y } t \in M\}$$

Expresiones regulares. Definiciones

Más operadores sobre lenguajes:

potencia de un lenguaje

$L^0 = \{\varepsilon\}$ $\quad L^1 = L$

$L^i = L\ L^{i-1}$ para $i>1$

cerradura de Kleene

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

CERO o MÁS concatenaciones

cerradura positiva

$$L^* = \bigcup_{i=1}^{\infty} L^i$$

UNA o MÁS concatenaciones

Expresiones regulares. Definiciones

Una expresión regular es una fórmula que caracteriza un lenguaje (conjunto de cadenas)

Se usan en compilación, procesamiento de lenguaje natural, búsquedas de cadenas en UNIX y muchos editores de texto (vi, Perl, Emacs, grep, Word, netscape, nedit)

Veremos la sintaxis de Perl (todas son muy similares)

Expresiones regulares. Definiciones

Sea Σ un alfabeto

expresión regular

1) ε es la expresión regular cuyo lenguaje es $L(\varepsilon)=\{\varepsilon\}$

2) Si $a\in\Sigma$, a es la expresión regular cuyo lenguaje es $L(a)=\{a\}$

3) Sean r,s exp. reg. con lenguajes L(r) y L(s)

- (r)|(s) es la exp. reg. cuyo lenguaje es $L(r) \cup L(s)$
- (r)(s) es la exp. reg. cuyo lenguaje es L(r)L(s)
- (r)* es la exp. reg. cuyo lenguaje es (L(r))*

Se pueden quitar los paréntesis cuando estos no sean necesarios

Expresiones regulares. Ejemplos

Ejemplo 1: Sea Σ={a,b}

r	*L(r)*
ab	{ab}
a|b	{a,b}
a*	{ε,a,aa,aaa,aaaa,...}
ab*	{a,ab,ab,abbb,...}
(ab|c)*d	{d,abd,cd,abcd,ababcd,...}

Expresiones regulares. Ejemplos

Ejemplo 2: Sea $\Sigma=\{0,1\}$

`00` `(00)`	la cadena '00'
`(1|10)*`	ε y todas las cadenas que empiezan con '1' y no tienen dos '0' consecutivos
`(0|1)*`	todas las cadenas con 0 ó más '1' ó '0' y ε
`(0|1)*00(0|1)*`	todas las cadenas con al menos 2 '0' consecutivos
`(0|ε)(1|10)*`	todas las cadenas que no tengan dos '0' consecutivos
`(0|1)*011`	todas las cadenas que acaban en '011'

Expresiones regulares. Notaciones

Convenios de notación útiles:

- Si r representa L(r)

(r)+ **representa** L(r)+

una ó más veces **r**

Se cumple que:

$r^* = \varepsilon | r^+$

$r^+ = rr^*$

(r)? **representa** $\{\varepsilon\} \cup L(r)$

0 ó 1 vez

Se cumple que

$r? = r | \varepsilon$

Expresiones regulares. Notaciones

- Formas abreviadas

  [....] expresa elección en un conjunto de elementos del alfabeto

  [aeiou] ≅ a|e|i|o|u

  [0-9] ≅ 0|1|2|3|4|5|6|7|8|9

  \- expresa subrango

  [+-]?[0-9]+ son las constantes enteras

  Ejemplos:

  [a-zA-Z][_0-9a-zA-Z]* identificadores Pascal

Expresiones regulares. Notaciones

- **Formas abreviadas**

  `[^ ...]` expresa el complementario de un conjunto de elementos del alfabeto

`[^aeiou0-9]`	$\cong$	símbolos distintos de vocal o dígito

  `.` representa un símbolo cualquiera del alfabeto

`.[0-9]`	cualquier símbolo seguido de un dígito

Expresiones regulares. Ejemplos en Perl

- En Perl el alfabeto es el formado por los caracteres alfanuméricos (letras, dígitos, espacios, tabuladores, puntuación, etc)
- Las expresiones regulares se escriben entre / /
- La expresión regular ε no se usa en Perl
- Si quiero escribir un carácter "especial" en una exp. reg.

  +, -, *, ., ^, [,], ?, (,), |

  lo escribo con \ delante (excepto si no hay confusión posible)

Expresiones regulares. Ejemplos en Perl

r	ejemplos de cadenas de L(r)
/woodchucks/	links to <u>woodchucks</u> and
/a/	M<u>a</u>ry Ann stopped by Mona's
/Claire says,/	'my gift, please,' <u>Claire says,</u>
/song/	all our pretty <u>song</u>s
/!/	You've done it again<u>!</u>

e.r. (disyunción de símbolos)

r	ejemplos de cadenas de L(r)
/[wW]oodchuck/	Woodchuck
/[abc]/	In uomini, in soldati
/[1234567890]/	in 1989
/[A-Z]/	we should call it 'Drenched
/[a-z]/	my beans were impatient to
/[0-9]/	Chapter 1: Down the Hole

e.r. (complemento de símbolos)

r	ejemplos de cadenas de L(r)
/[^A-Z]/	Oyfn pripetchik
/[^Ss]/	I have no exquisite reason for
/[^\.]/	our resident Djinn
/[e^]/	look up ^ now
/a^b/	look up a^b now

e.r. ?

r	ejemplos de cadenas de L(r)
/woodchucks?/	woodchuck
/colou?r/	colour

e.r. * + .

r	ejemplos de cadenas de L(r)
/a*/	ε, a, aa, aaaa ...
/a+/	a, aa, aaa ...
/[ab]+/	a, b, ab, aa, ba, bb ...
/beg.n/	begin, beg'n, begun ...
/a.*a/	abracadabra ...

anclas (sólo Perl)

^	principio de línea
$	fin de línea
\b	fin de palabra
\B	no fin de palabra

e.r. ejemplo en Perl

Encontrar todas las ocurrencias del artículo "the"

```
/the/
/[Tt]he/
/\b[Tt]he\b/
```

e.r.

Para practicar:

Podéis aprender perl, usar emacs, etc

Ejercicios del tema 1

Escribir las e.r. (notación Perl) que representen:

- El conjunto de palabras con más de 2 vocales.
- Los números que empiecen en 2 y tengan al menos 2 cifras.
- El conjunto de todas las cadenas del alfabeto {a,b} tales que cada a está inmediatamente precedida e inmediatamente seguida de una b.

Nota: **Una palabra es una cadena de letras separada de otras un por espacio en blanco, símbolo de puntuación o fin de línea.**

Tema 2: Autómatas finitos

1) Autómatas finitos

- Generalidades
- Grafo de transiciones asociado a un AF
- Aceptación de una cadena por un AF

2) Conversión de una expresión regular en un AFN

3) Transformación de un AFN en un AFD

4) Minimización de un AFD

Autómatas Finitos. Generalidades

Los autómatas finitos pueden ser utilizados para reconocer los lenguajes expresados mediante exprexiones regulares

Un autómata finito (FA) es una máquina abstracta que reconoce cadenas correspondientes a un lenguaje

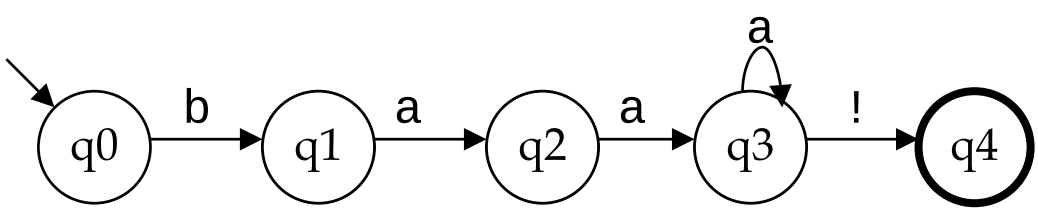
Misión de un FA:

- "reconocer" si una cadena de entrada respeta las reglas determinadas por una expresión regular

Autómatas finitos como grafos

Empezaremos con una visión gráfica del lenguaje de las ovejas

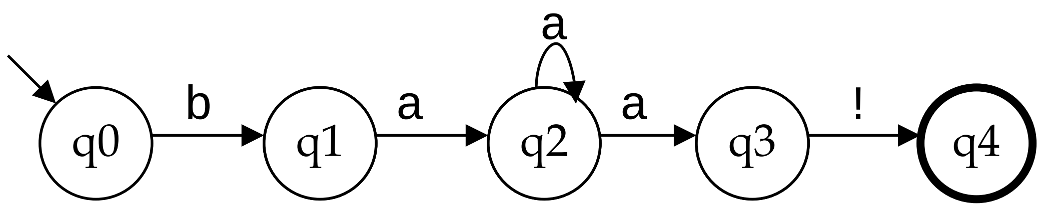
`baa+!`



5 estados, q0 es el inicial, q4 es el final

Autómatas finitos como grafos

Por supuesto, hay alternativas ...



Autómatas Finitos. Definiciones

Autómata Finito Determinista

Un DFA es una 5-tupla (S,Σ,δ,s_0,F) donde:

1) **S : conjunto de estados**

2) Σ : **conjunto de símbolos de entrada**

3) δ : **función de transición**

$$\delta : S \times \Sigma \rightarrow S$$

4) $s_0 \in S$: **estado inicial**

5) $F \subseteq S$: **conjunto de estados finales (o de aceptación)**

Autómatas Finitos. Definiciones

Autómata Finito No Determinista

Un NFA es una 5-tupla (S,Σ,δ,s_0,F) donde:

1) S : conjunto de estados

2) Σ : conjunto de símbolos de entrada

3) δ : función de transición

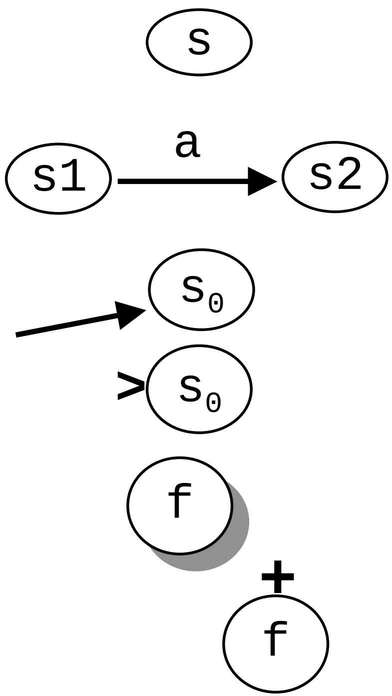
$$\delta : S \times (\Sigma \cup \{\varepsilon\}) \to \mathbf{P}(S)$$

4) $s_0 \in S$: estado inicial

5) $F \subseteq S$: conjunto de estados finales (o de aceptación)

Autómatas Finitos. Grafo de transiciones

Notación gráfica:



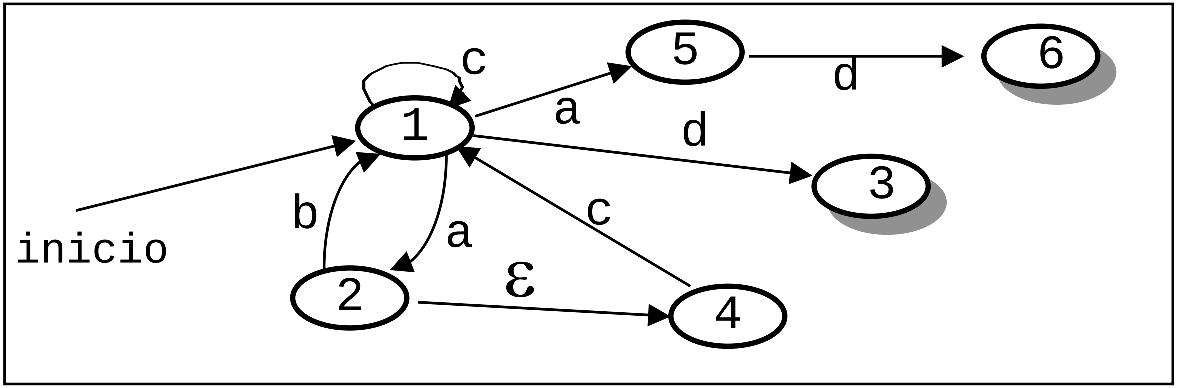
un estado

"transitar" de s1 a s2
cuando se encuentre "a"

s_0 es el estado inicial

f es un estado final
(de aceptación)

NFA como grafo de transiciones



1) S ={1,2,3,4,5,6}

2) Σ ={a,b,c,d}

3) δ(1,c)={1} δ(1,d)={3} δ(1,a)={2,5} δ(2,b)={1}

δ(2,ε)={4} δ(4,c)={1} δ(5,b)={6}

4) s_0 = 1

5) F = {3,6}

Autómatas Finitos. Aceptación

¿Cómo funciona un NFA?

Dada una cadena, debe determinar si la acepta o no

aceptación de una cadena por un NFA

La cadena $c_1 c_2 \ldots c_n$ es **aceptada** por un NFA cuando existe, en el grafo de transiciones, un camino

$$s_0 \xrightarrow{c_1} s_1 \xrightarrow{c_2} s_2 \rightarrow \ldots \rightarrow s_{m-1} \xrightarrow{c_n} s_m$$

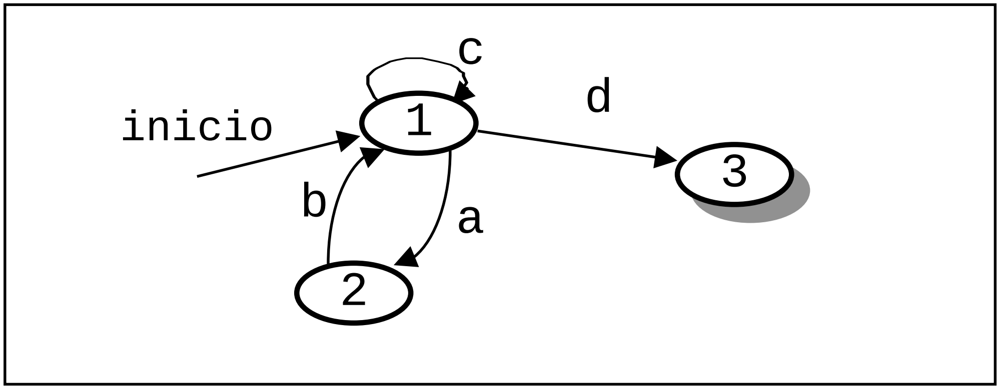
(puede haber ε-movimientos)

de manera que s_m es un estado final (de aceptación)

Autómatas Finitos. Aceptación

Ejemplo:

- **¿Aceptaría el autómata “abcd”? ¿Y ”acd”?**



Autómatas Finitos Deterministas

Un Autómata Finito Determinista (DFA) es un caso particular de NFA

Autómata Finito Determinista

Un DFA es un NFA tal que:

1) ε no etiqueta ningún arco

2) $\delta : S \times \Sigma \rightarrow S$

Es decir:

- toda transición exige un símbolo
- desde un estado, no hay dos arcos etiquetados con el mismo símbolo

Autómatas Finitos Deterministas

Simular un DFA es muy sencillo y eficiente

En lo que sigue:

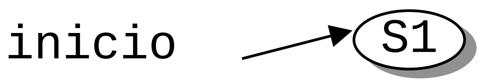
- dada una e.r., generar el NFA
- Convertir el NFA en un DFA y minimizarlo
- Implementar el DFA

Conversión de una expresión regular a NFA

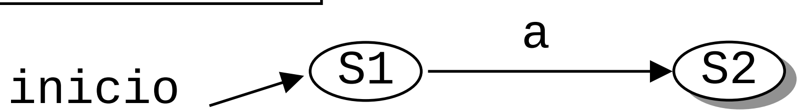
Objetivo: dada una expresión regular, generar un DFA que la reconozca

Método: construcción de Thompson (Bell Labs)

NFA para ε

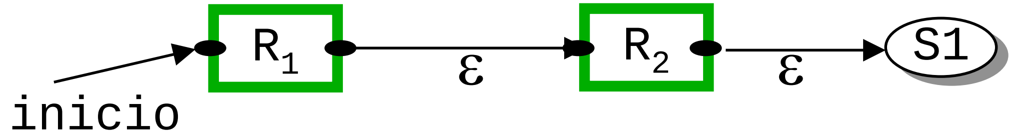


DFA para a∈Σ

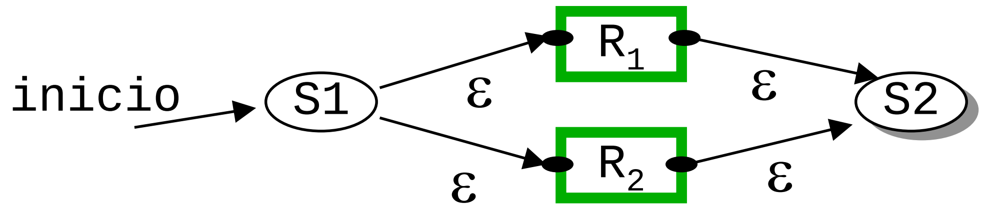


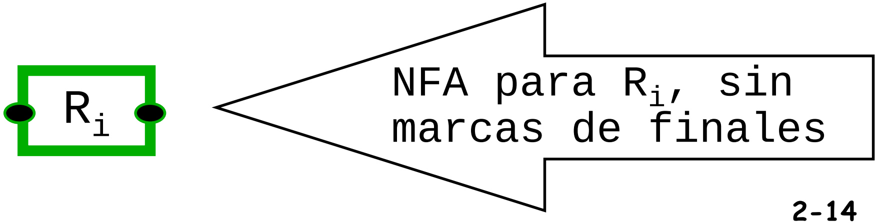
Conversión de una expresión regular a NFA

NFA para la expresión regular R_1R_2



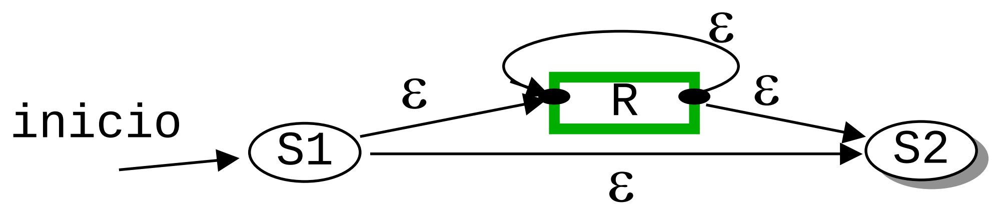
NFA para la expresión regular $R_1|R_2$





Conversión de una e.r. en un NFA

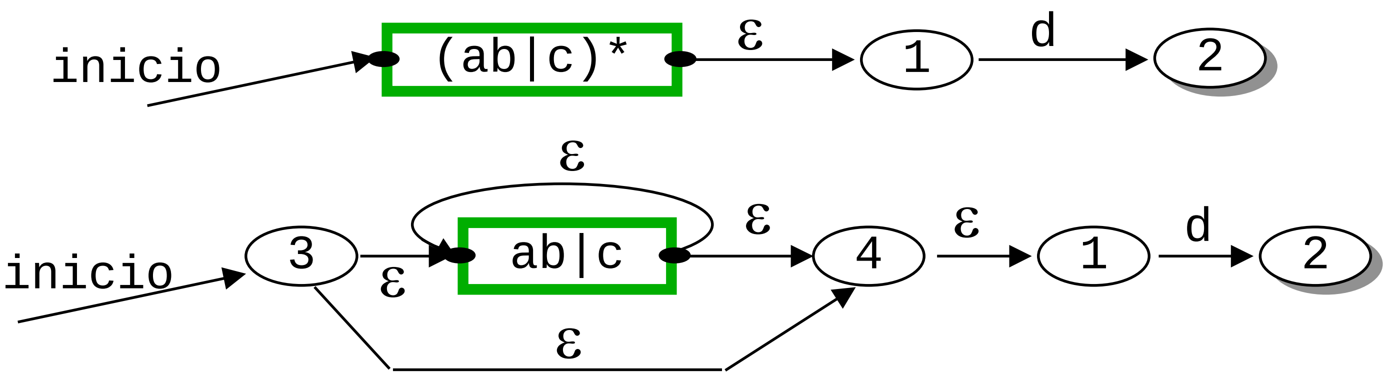
NFA para la expresión regular R*



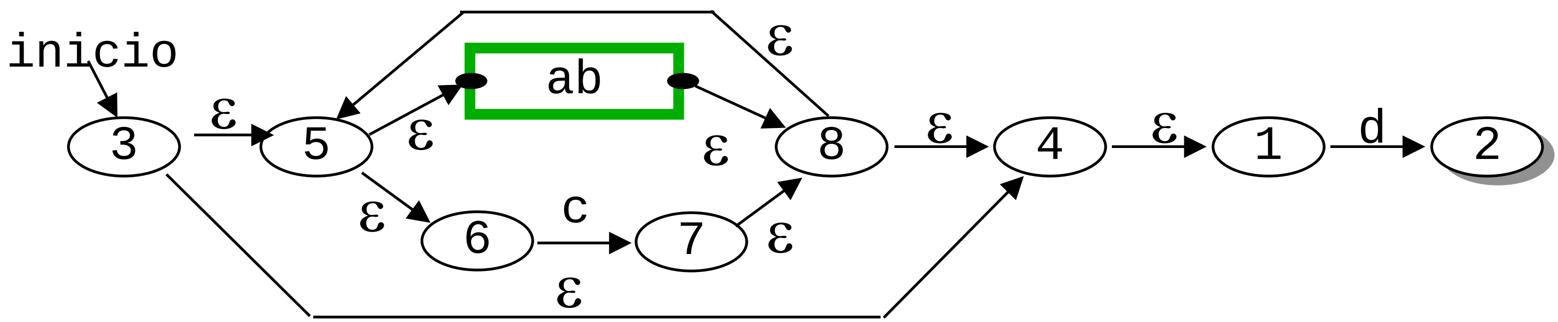
Conversión de una expresión regular a NFA

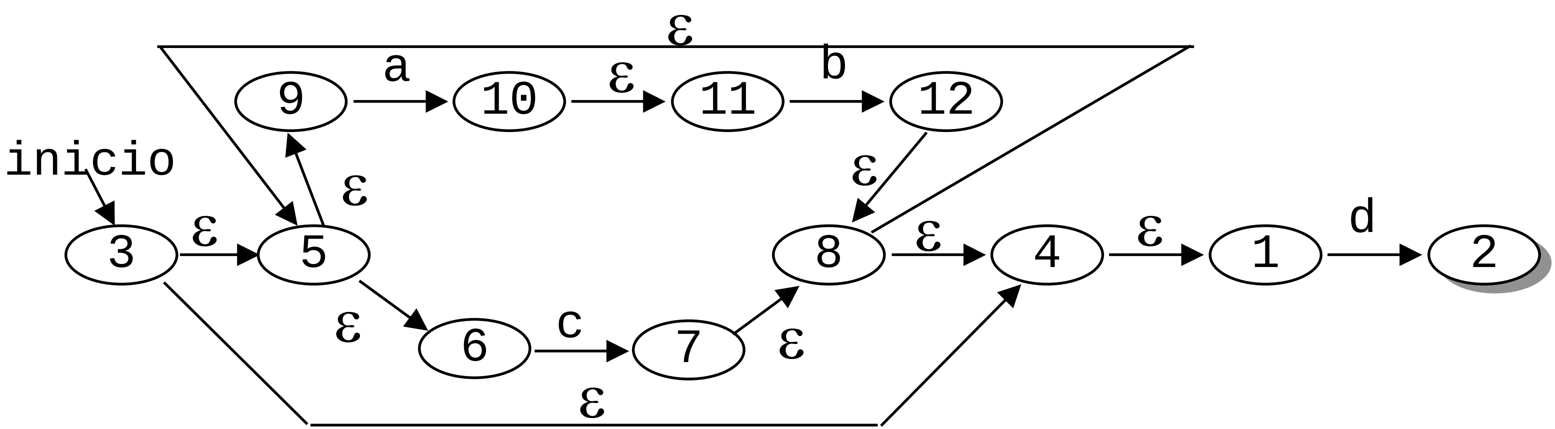
Ejemplo: proceso de construcción para

(ab|c)*d



Conversión de una e. r. en un NFA





Transformación de un NFA en un DFA

Generar un NFA a partir de una e.r. es sencillo

Implementar un DFA es más sencillo y eficiente que implementar un NFA

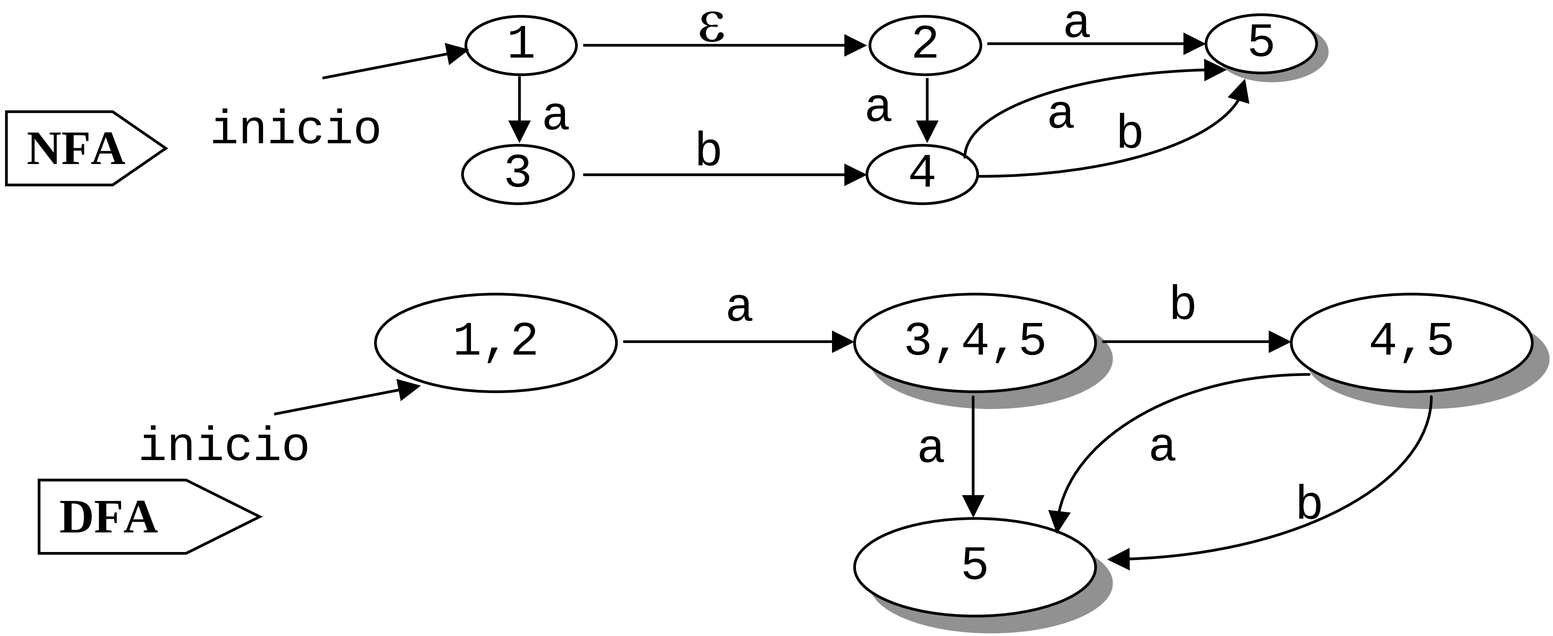
Por lo tanto, es interesante saber generar, a partir de un NFA, un DFA equivalente

El método se basa en la idea de ε-clausura (ver [UllmanHoM 01])

Transformación de un NFA en un DFA

La idea básica es que un estado del DFA agrupa un conjunto de estados del NFA

Ejemplo:



Transformación de un NFA en un DFA

Sea A=(S,Σ,δ,s_0,F)**un NFA**

ε-clausura de $s \in S$

Conjunto de estados de N alcanzables desde s usando transiciones ε

ε-clausura de $T \subseteq S$

$$\bigcup_{s \in T} \varepsilon\text{-clausura}(s)$$

mueve(T,c)

Conjunto de estados a los que se puede llegar desde algún estado de de T mediante la transición c

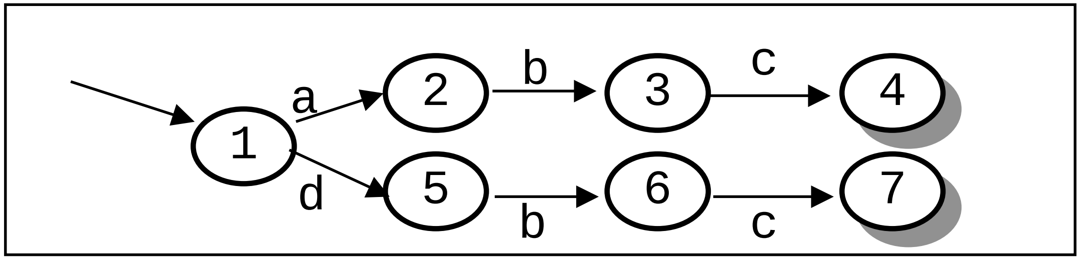
Minimización de DFA

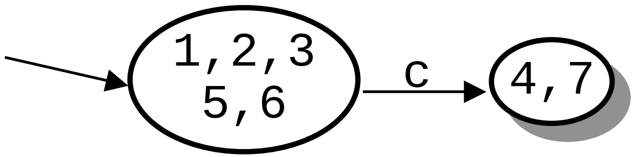
Algunas cuestiones:

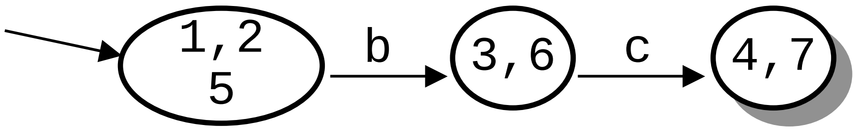
- Como es lógico, cuantos más estados tiene un FA, más memoria es necesaria
- El número de estados del DFA se puede/debe minimizar (ver [UllmanHoM 01])
  - inicialmente, dos estados:
    - uno con los de aceptación
    - otro con los demás
  - sucesivas particiones de estados "globales" que no concuerdan con algún sucesor para un carácter de entrada
- El DFA mínimo equivalente es único

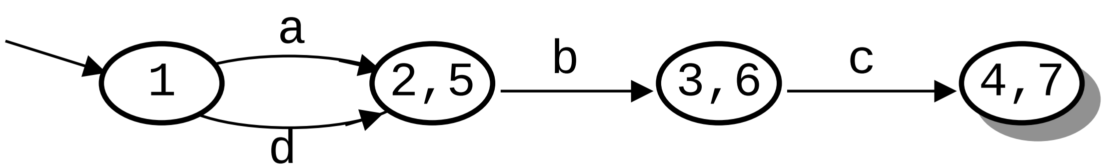
Minimización de DFA

Ejemplo:









Ejercicios del tema 2

Ejercicio 1: Los identificadores para un determinado lenguaje de programación se definen de acuerdo con la siguiente descripción:

Un identificador puede empezar con una letra o un "underscore" (carácter "_"), debe estar seguido por 0 ó más letras, dígitos o underscores, pero con las restricciones siguientes:

1) No pueden aparecer dos underscores seguidos.
2) Un identificador no puede terminar con un underscore.

Además, no hay ninguna limitación en cuanto a la longitud de los identificadores.

1.1) Dibujar el Autómata Finito Determinista que corresponde a los identificadores descritos anteriormente.
Para etiquetar los arcos, en lugar de utilizar caracteres simples utilizar las siguientes clases de caracteres:

letra [a-zA-Z]   digito [0-9]   und "_"

1.2) Dar una expresión regular correspondiente a los identificadores descritos anteriormente.

Ejercicios del tema 2

Ejercicio 2: El libro "Pascal: User Manual and Report" de K. Jensen y N. Wirth, que establece la especificación ISO Pascal Standard, define un comentario del lenguaje como:

(* seguido de cualquier secuencia de 0 ó más caracteres que no contenga *), y terminado por *)

Escribir una expresión regular con sintaxis Perl para los comentarios Pascal así definidos.

Nota: En este enunciado (* quiere decir el carácter (seguido del carácter *, no es una expresión regular.

Tema 3: Gramáticas libres de contexto

1) Introducción

2) Gramáticas. Definiciones y clasificación

3) GLC. Notaciones

4) GLC. Árboles de análisis sintáctico

5) GLC. Derivación a dcha. y a izda.

6) GLC. Ambigüedad y transformación de gramáticas

Gramáticas. Definiciones

El estudio de gramáticas es anterior al de lenguajes de programación

Empezó con el estudio del lenguaje natural

Punto de referencia: Noam Chomsky

gramática

Una gramática G=(N,T,S,P) es una 4-tupla donde:

1) N es el conjunto de No-Terminales

2) T es el de Terminales

$N \cap T = \varnothing$

3) $S \in N$, y se denomina Símbolo Inicial

4) P es el conjunto de Producciones

Gramáticas. Definiciones

El **símbolo inicial** es el único no terminal que se utiliza para generar las cadenas de terminales del lenguaje

Objetivo: generar cadenas de terminales

Una **producción** es una regla que establece una transformación de cadenas

Forma de una producción $\boxed{\alpha \rightarrow \beta}$

Significado: si δ es una cadena, al aplicarle la producción, una aparición de α en δ se sustituye por β

Gramáticas. Definiciones

Establezcamos dos operadores

Sea G=(N,T,S,P) una gramática

$\alpha A \delta$ deriva directamente $\alpha\beta\delta$ en la gramática G

derivación directa

Sean $\alpha, \delta \in$ **(N** $\cup$ **T)* y sea** $A \rightarrow \beta \in$ **P. Entonces**

$$\alpha A \delta \Rightarrow_G \alpha\beta\delta$$

α_1 deriva α_n en cero o más pasos en la gramática G

derivación

Sean $\alpha_1 \ldots \alpha_n \in$ **(N** $\cup$ **T)* t.q.**

$$\alpha_0 \Rightarrow_G \alpha_1 \Rightarrow_G \ldots \Rightarrow_G \alpha_n \text{ , } n >= 0$$

Entonces

$$\alpha_1 \overset{*}{\Rightarrow}_G \alpha_n$$

Gramáticas. Definiciones

Ejemplo: Consideremos la gramática G=(N,T,S,P) donde

- N={ frase,sujeto,predicado, artículo, nombre,verbo,adverbio }
- T={ el, la, está, lejos, cerca, perro, gata }
- S={ frase }
- P={ p1,p2,p3,p4,p5,p6,p7,p8,p9,p10 }

p1: frase → sujeto predicado

p2: sujeto → artículo nombre

p3: predicado → verbo adverbio

p4: artículo → el

p5: artículo → la

p6: nombre → perro

p7: nombre → gata

p8: verbo → está

p9: adverbio → cerca

p10: adverbio → lejos

Gramáticas. Definiciones

Ejemplos de derivaciones

α1: sujeto predicado

predicado→verbo adverbio

α2: sujeto verbo adverbio

verbo → está

α3: sujeto está adverbio

Tenemos: $\boxed{\alpha_1 \Rightarrow_G \alpha_2}$ $\boxed{\alpha_1 \overset{*}{\Rightarrow}_G \alpha_3}$ $\boxed{\alpha_1 \overset{+}{\Rightarrow}_G \alpha_3}$

Gramáticas. Definiciones

Algunas definiciones:

forma de frase

Cualquier cadena que se pueda derivar del símbolo inicial (cadena de terminales y no terminales)

frase

Cualquier forma de frase con sólo elementos terminales

lenguaje definido por una gramática

Conjunto de todas las frases de la gramática

Gramáticas. Definiciones

Es fácil ver que:

- los siguientes elementos pertenecen al lenguaje generado por la gramática:
  - `el perro está cerca`
  - `la perro está lejos`
  - `el gata está cerca`
  - .....
- los siguientes elementos NO pertenecen al lenguaje generado por la gramática:
  - `el perro`
  - `la lejos perro está`
  - .....
- Por desgracia, no siempre será tan sencillo

Ejercicio 1: Sea $G=(N,T,S,P)$ con

- $N=\{A\}$
- $T=\{a,b\}$
- $P=\{A\to aAb,\ A\to ab\}$
- $S=A$

$$L(G)=\{a^n b^n \mid n>=1\}$$

Ejercicio 2: Sea $G=(N,T,S,P)$ con

- $N=\{S,A,B\}$
- $T=\{a,b\}$
- $P=\{\ S\to aB,\ S\to bA,\ A\to a, A\to aS,$
  $A\to bAA,\ B\to b,\ B\to bS,\ B\to aBB\}$
- $S=S$

$$L(G)=\{w\in T^* \mid\ |w|_a=|w|_b\}$$

Ejercicio 3: ¿Cuál es el lenguaje generado por la siguiente gramática?

$S \to [\ S$
$S \to S1$
$S1 \to [\ a\ ]$

Gramáticas. Definiciones

Hemos visto cómo la aplicación de producciones genera las frases del lenguaje

Para compiladores y lenguaje natural, el proceso es en sentido contrario:

- dado una cadena:
  - ¿Pertenece al lenguaje?
  - ¿Cuáles son las producciones aplicadas para derivarla del símbolo inicial?

Este proceso lo lleva a cabo el **analizador sintáctico** ("parser")

Gramáticas. Definiciones

Ejercicio 3: Sea $G=(N,T,S,P)$ con

- N = { `programa, bloque, insts, inst, opas, ident, const, punto`}
- T = {`PROGRAM, BEGIN, END, =, A, B, 1, 0, .` }
- S = `programa`

P=

```
programa → PROGRAM ident punto bloque
bloque  → BEGIN insts END punto
insts  → insts punto inst
insts  → inst
inst   → ident opas const
opas  → =
ident  → A
ident  → B
const  → 1
const  → 0
punto  → .
```

```
PROGRAM A.
BEGIN
     B=1.
     A=0
END.
```

```
PROGRAM A.
BEGIN
     B=1.
     A=0.
END.
```

¿Sintácticamente correctos?

Gramáticas. Clasificación

Una producción "general" tiene la forma

$$\alpha \rightarrow \beta$$

Donde α y β son cadenas de terminales y no terminales

Imponiendo restricciones a las posibles formas de las producciones, se obtienen distintos tipos de gramáticas

- restricciones respecto a qué pueden ser α y β
- restricciones de dónde se puede aplicar la transformación establecida por la producción

Gramáticas. Clasificación

Clasificación de Chomsky

- gramáticas de Tipo 0

libres

- gramáticas de Tipo 1

$$\alpha A \beta \rightarrow \alpha\delta\beta$$

dependientes del contexto

  - $\alpha,\beta,\delta \in (N \cup T)^*$
  - δ no vacío y *A* un no terminal
  - *A* se transforma en δ sólo cuando va precedido por α y seguido por β
  - La producción ya no se aplica siempre, sino sólo en determinados contextos
    - demasiado complicadas de manejar

Gramáticas. Clasificación

- gramáticas de Tipo 2

libres de contexto

$$A \rightarrow \beta$$

  - A es un (único) no terminal
  - Cada vez que aparece A, se puede sustituir por β
  - Se corresponde con la notación BNF
  - Pascal (en la mayoría de sus aspectos) es una gramática de tipo 2

- gramáticas de Tipo 3

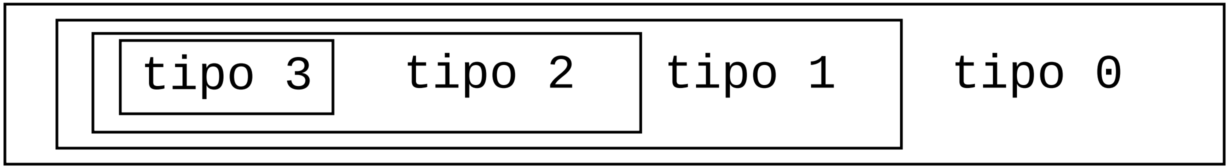
$$A \rightarrow w \qquad A \rightarrow wB$$

  - A ,B son no terminales
  - w es una cadena de terminales
  - poco potentes
  - equivalentes a expresiones regulares

regulares

Gramáticas. Clasificación

En forma de diagrama:



Cada gramática de un tipo es también del tipo anterior

Cuanto menos restrictiva es una gramática, más complejo es su análisis

Las más sencillas son las de Tipo 3, que pueden ser reconocidas por autómatas de estados finitos

GLC. Notación

Por claridad, para GLC usaremos los siguiente convenios:

- consideraremos terminales:
  - primeras minúsculas del abecedario
  - símbolos de operación y puntuación
  - dígitos
  - algunas palabras: perro,begin
- consideraremos NO terminales:
  - primeras mayúsculas del abecedario
  - algunas palabras: sujeto,expresión
  - la S, suele representar el símbolo inicial
- ...,X,Y,Z: representan símbolos (terminales o no terminales)
- letras minúsculas de "en medio" (u,v,..) representan cadenas de terminales

GLC. Notación

- letras griegas minúsculas representan formas de frase (cadenas de símbolos)
  - Así, en una GLC, una producción se escribe siempre como

$$\boxed{A \rightarrow \beta}$$

parte izquierda

parte derecha

- varias producciones con igual parte izda. se pueden agrupar en una producción con alternativas

$$\boxed{\begin{array}{l} A \rightarrow \alpha_1 \\ A \rightarrow \alpha_2 \\ \quad \ldots\ldots \\ A \rightarrow \alpha_n \end{array}} \quad = \quad \boxed{A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n}$$

GLC. Arboles de análisis sintáctico

Considerar la siguiente GLC

```
expr → term | expr + term
term → prim | term * prim
prim → a|b|c
```

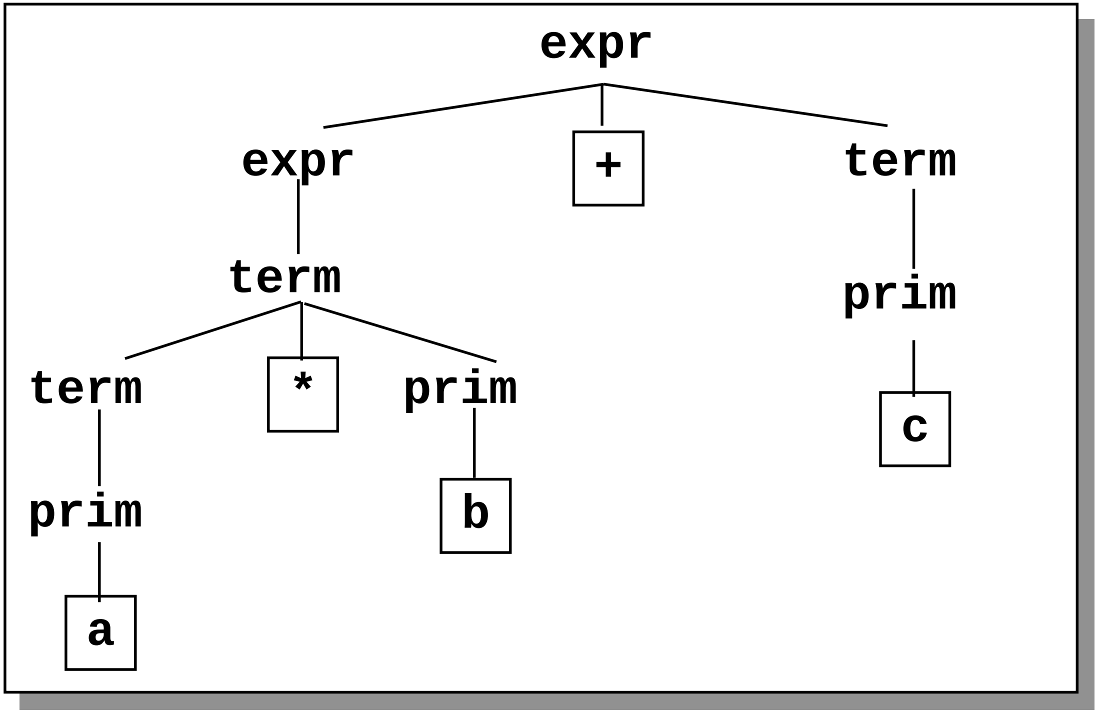
terminales

¿Es a*b+c del lenguaje generado?

a * b + c	
prim * b + c	prim → a
prim*prim + c	prim → b
prim*prim+prim	prim → c
term*prim+prim	term → prim
term*prim+term	term → prim
term+term	term → term*prim
expr+term	expr → term
expr	expr → expr+term

GLC. Arboles de análisis sintáctico

Gráficamente, la derivación se puede representar como un árbol de sintaxis



GLC. Arboles de análisis sintáctico

Características:

- el **nodo raíz** se etiqueta con el símbolo inicial
- cada **nodo no hoja** se etiqueta con un no terminal
- los hijos de un nodo son los **símbolos** (de izda. a dcha.) que aparecen en una de las **producciones** que tienen dicho nodo como parte izda.
- cuando se ha derivado una frase, las hojas del árbol son terminales

La derivación de una frase (o forma de frase) no es fácil

1) Podemos elegir un camino no apropiado (necesitando "backtraking")
2) Una misma frase puede tener más de una derivación posible (gramática ambigua)

GLC. Arboles de análisis sintáctico

Podíamos haber seguido otro camino,
¡¡Que no lleva a nada!!

```
expr → term | expr + term
term → prim | term * prim
prim → a|b|c
```

`a * b + c`	
`prim * b + c`	`prim → a`
`prim*prim + c`	`prim → b`
`prim*prim+prim`	`prim → c`
`prim*term+prim`	`term → prim`
`prim*expr+prim`	`expr → term`
`prim*expr`	`expr → expr+term`
`term*expr`	`term → prim`
`expr*expr`	`expr → term`
`??????????`	`????????????`

GLC. Arboles de análisis sintáctico

A veces, una frase tiene más de un árbol posible: ambigüedad

$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$

id+id*id

id+**id*****id**	
E+**id*****id**	$E \rightarrow$ **id**
E+E***id**	$E \rightarrow$ **id**
E***id**	$E \rightarrow E+E$
E*E	$E \rightarrow$ **id**
E	$E \rightarrow E*E$

id+**id*****id**	
id+**id***E	$E \rightarrow$ **id**
id+E*E	$E \rightarrow$ **id**
id+E	$E \rightarrow E*E$
E+E	$E \rightarrow$ **id**
E	$E \rightarrow E+E$

GLC. Derivación izda. y dcha.

En cada paso en una derivación:

- hay que elegir qué no terminal sustituir
- elegido uno, optar por una de las posibles producciones que lo tengan como parte izda.

Si siempre se sustituye el de más a la izda.

derivación por la izda.

Si siempre se sustituye el de más a la dcha.

derivación por la dcha.

GLC. Derivación izda. y dcha.

Tomemos otra vez $E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$

id+id*id

izda

$$E \underset{mi}{\Rightarrow} E + E \underset{mi}{\Rightarrow} \mathbf{id} + E \underset{mi}{\Rightarrow} \mathbf{id} + E * E \underset{mi}{\Rightarrow} \mathbf{id} + \mathbf{id} * E \underset{mi}{\Rightarrow} \mathbf{id} + \mathbf{id} * \mathbf{id}$$

dcha

$$E \underset{md}{\Rightarrow} E + E \underset{md}{\Rightarrow} E + E * E \underset{md}{\Rightarrow} E + E * \mathbf{id} \underset{md}{\Rightarrow} E + \mathbf{id} * \mathbf{id} \underset{md}{\Rightarrow} \mathbf{id} + \mathbf{id} * \mathbf{id}$$

GLC. Derivación izda. y dcha.

Pero también puede existir más de una derivación por la izda. (o dcha.)

$E \rightarrow E + E \mid E * E \mid (E) \mid -E \mid \mathbf{id}$

id+id*id

izda

$$E \underset{mi}{\Rightarrow} E + E \underset{mi}{\Rightarrow} \mathbf{id} + E \underset{mi}{\Rightarrow} \mathbf{id} + E * E \underset{mi}{\Rightarrow} \mathbf{id} + \mathbf{id} * E \underset{mi}{\Rightarrow} \mathbf{id} + \mathbf{id} * \mathbf{id}$$

izda

$$E \underset{mi}{\Rightarrow} E * E \underset{mi}{\Rightarrow} E + E * E \underset{mi}{\Rightarrow} \mathbf{id} + E * E \underset{mi}{\Rightarrow} \mathbf{id} + \mathbf{id} * E \underset{mi}{\Rightarrow} \mathbf{id} + \mathbf{id} * \mathbf{id}$$

GLC. Ambigüedad

Una gramática es ambígua si existe al menos una frase ambígua

Una frase es ambígua si existe más de un árbol para ella (mi ó md)

- produce indeterminismo
- es importante eliminarla, cuando se pueda

A veces, es posible eliminar la ambigüedad

↓

transformar la gramática en una equivalente que no sea ambigua

GLC. Ambigüedad

Ejemplo: Considerar la instrucción "if" en Pascal

Si la gramática es de la forma

$$\begin{aligned} \textit{inst} \rightarrow\ & \textbf{if}\ \textit{exp}\ \textbf{then}\ \textit{inst} \\ |\ & \textbf{if}\ \textit{exp}\ \textbf{then}\ \textit{inst}\ \textbf{else}\ \textit{inst} \\ |\ & \textit{otras instrucciones} \end{aligned}$$

```
if exp1 then if exp2 then inst2 else inst3
```

1

```
if exp1 then
   if exp2 then
        inst2
     else
        inst3
```

2

```
if exp1 then
   if exp2 then
      inst2
else
   inst3
```

GLC. Ambigüedad

¿Con cuál nos quedamos?

Generalmente, se aplica la misma regla que aplica Pascal:

- cada **else** se empareja con el **then** más próximo
- es decir, la buena versión es la 1

¿Se puede expresar esta regla en una gramática transformada?

- en este caso, sí
- no siempre será posible

```
inst   → instC | instI
instC  → if exp then instC else instC
         | otras instrucciones
instI  → if exp then inst
         | if exp then instC else instI
```

GLC. Ambigüedad

Ejercicio: "Convencerse" de que genera el mismo lenguaje que la anterior y se ha eliminado la ambigüedad

¿Es posible saber si una gramática dada **es ambígua**?

¡¡ NO !! (Hopcroft y Ullman)

Una cuestión interesante: la simplificación de gramáticas

- puesto que puede haber varias gramáticas equivalentes, busquemos alguna más simple

- posibles simplificaciones:
  - eliminación de no terminales inútiles
  - eliminación de producciones-ε
  - eliminación de producciones unitarias

GLC. Transformación de gramáticas

Ejemplo: Considerar la gramática

$$
\begin{array}{lll}
S & \rightarrow & A \mid B \\
A & \rightarrow & a \\
B & \rightarrow & B\ b \\
C & \rightarrow & c
\end{array}
$$

Verifica que:

- el no terminal C no es alcanzable desde S
- el no terminal B no deriva ninguna cadena terminal

Los no terminales B y C se denominan inútiles

No terminales inútiles

- pueden ser eliminados
- se obtiene una gramática equivalente

GLC. Transformación de gramáticas

Un no terminal A es útil si existe

$$\boxed{S \Rightarrow^* \alpha A \beta \Rightarrow^* w}$$

para algún α,β y tal que $w \in T^*$

Dos condiciones necesarias de "utilidad"

1) debe haber alguna cadena de terminales que sea derivable de A
2) A debe aparecer en alguna forma de frase derivable desde S

Proceso:

1) eliminar los no terminales que no deriven ninguna frase
2) eliminar los no terminales que no sean alcanzables desde S

Un no terminal es terminable cuando es capaz de derivar alguna frase

```
Algoritmo  eliminaNoTerminables
                    (E G:GLC; S: G′:GLC)
--Pre: G=(N,T,P,S) t.q. L(G)<>∅
--Post: G′=(N′,T,P′,S) ∧ G′ ≅ G ∧
--      ∀X′∈N′.∃w∈T*.X′⇒* w
Vars viejo,nuevo: conj. de no terminales
Principio
  viejo:={}
  nuevo:={A∈N|A→w∈P,w∈T*}
  Mq viejo<>nuevo
    viejo:=nuevo
    nuevo:=viejo ∪
            {B∈N|B→α∈P,α∈(T ∪ viejo)*}
  FMq
  N′:=nuevo
  P′:={A→w∈P|A∈N′, w∈(N′ ∪ T)*}
Fin
```

GLC. Transformación de gramáticas

Segundo paso: eliminar los símbolos que no sean accesibles desde el símbolo inicial

```
Algoritmo  eliminaNoAccesibles
                  (E G:GLC; S: G':GLC)
--Pre: G=(N,T,P,S) t.q. L(G)<>∅ ∧
--    ∀X∈N.X es terminable
--Post: G'=(N',T',P',S) ∧ G' ≅ G ∧
--      ∀X∈(N'∪T').∃α,β∈(N'∪T')*.
--      S ⇒* αXβ
Vars viejo,nuevo: conj. símbolos gram.
Principio
  viejo:={S}
  nuevo:={X∈(N ∪ T)|S→ αXβ ∈ P}∪{S}
  Mq viejo<>nuevo
    viejo:=nuevo
    nuevo:=viejo ∪
           {Y∈(N ∪ T)|A→ αYβ ∈P,A∈viejo}
  FMq
  <N',T'>:=<nuevo ∩ N, nuevo ∩ T>
  P':={A→ w ∈ P|A∈N',w∈(N'∪T')*}
Fin
```

GLC. Transformación de gramáticas

Un no terminal X se dice **anulable** cuando $X \Rightarrow^{*} \varepsilon$

Transformación interesante:

obtener una gramática equivalente a otra "sin ε"

Una gramática es **sin ε** ssi:

1) no hay ninguna regla $X \rightarrow \varepsilon$

2) como mucho, hay una producción $S \rightarrow \varepsilon$, pero entonces S no aparece en la parte derecha de ninguna otra producción

GLC. Transformación de gramáticas

Otra transformación interesante: eliminación de producciones unitarias

- **una producción de la forma** $A \rightarrow B$ **se dice** unitaria

Conclusión: desarrollo de formas normales

- **formas normales de Chomsky y de Greibach**

GLC. Comparación de gramáticas

Asumamos una gramática para un lenguaje

¿Es correcta?¿"Expresa" el lenguaje que queríamos expresar?

Notar que la gramática es la propia definición del lenguaje

Formas de verificación:

- hacer "algunos" tests para ver resultados
- comparar la equivalencia con una gramática "correcta"
- NO existe un algoritmo general para GLC

Ejercicios del tema 3

Escribir las gramáticas que generen los siguientes lenguajes sobre el alfabeto {a,b}:

- El conjunto de palabras de la forma $a^n b^m$ con $n>m$.
- El conjunto de palabras para las que el número de aes es exactamente el doble del número de bes (por ejemplo, 0 aes y 0 bes, ó 6 aes y 3 bes).

Tema 4: Problemas decidibles y semidecidibles

Bibliografía para este tema

Ullman, J.D., Hopcroft, J.E., and Motwani, R. (2001). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley.

Lewis, H. and Papadimitriou, C. (1981). *Elements of the Theory of Computation.* Prentice-Hall.

Lenguajes. Definiciones

alfabeto

conjunto finito de símbolos

ejemplos

{0,1}, letras y dígitos

cadena (palabra o string)

secuencia finita de elementos del alfabeto

ejemplos

0010

estadoInicial

v_0

Lenguajes. Definiciones

longitud de una cadena

número de símbolos que la componen

ejemplos

|hola| = 4

|123456| = 6

ε = 0 (cadena vacía)

lenguaje

dado un alfabeto, cualquier conjunto de cadenas formadas con dicho alfabeto

ejemplo

Siendo Σ={0,1} {0,111,011,0111,01111,1}

Lenguajes y problemas decisionales

- A cada lenguaje L le podemos asociar el problema

  "Dada x, ¿x está en L?"

- A cada problema con sólo dos respuestas posibles (por ejemplo SI y NO) le podemos asociar el lenguaje:

  {x| x tiene solución SI}

Los problemas con sólo dos respuestas posibles se llaman problemas decisionales

Lenguajes y problemas decisionales

Ejemplo

Identificamos el problema:

Dado x número natural en binario, ¿es x primo?

con el lenguaje sobre el alfabeto {0,1}:

$\{x \mid x \text{ es un número primo en binario}\}$

Programas que reconocen lenguajes

Un programa con p una entrada x puede no terminar (se queda colgado por cualquier razón, por ejemplo, un bucle infinito). Lo denotamos con $p(x)\uparrow$

Si el programa p con entrada x termina escribimos $p(x)\downarrow$

Consideramos programas que tienen una entrada y una salida. La salida es un booleano (TRUE o FALSE)

Si un programa p con entrada x termina, denotamos la salida con $p(x)$

Programas que reconocen lenguajes

Un programa p **reconoce un lenguaje** L si

$$L = \{x \mid p(x) = \text{TRUE}\}$$

ATENCIÓN. Si p reconoce L para una entrada $x \notin L$ pueden ocurrir dos cosas:

- $p(x)\uparrow$
- $p(x) = \text{FALSE}$

Programas que reconocen lenguajes

Un programa p **para siempre** si $\forall x \ \ p(x)\downarrow$

Si p reconoce L y p para siempre entonces para una entrada x:

- si $x \in L$ entonces $p(x)=$ TRUE
- si $x \notin L$ entonces $p(x)=$ FALSE

Lenguajes decidibles y semidecidibles

L es semidecidible si existe un programa que reconoce L

L es decidible si existe un programa que reconoce L y para siempre

- Todos los decidibles son semidecidibles
- Hay semidecidibles no decidibles

Los decidibles corresponden a problemas que se pueden resolver con un programa

Ejemplos

El **problema de parada**:

"Dados p programa y x entrada, ¿ $p(x)\downarrow$? "

NO es decidible (aunque sí es semidecidible)

El problema de las **ecuaciones diofánticas**:

"Dada una ecuación e polinómica con coeficientes enteros, ¿e tiene una solución entera?"

Ejemplo: $x_1x_2-5+x_1=0$, $x^2-2=0$

NO es decidible (aunque sí es semidecidible)

¿Y los lenguajes formales?

Los semidecidibles son los lenguajes que podemos generar con gramáticas de tipo 0

A es decidible si y sólo si A y el complementario de A son ambos semidecidibles

Ejercicio del tema 4

Demostrar que el problema de las ecuaciones diofánticas es semidecidible

Tema 5: Tiempo polinómico versus tiempo exponencial

Bibliografía para los temas 5,6 y 7

GAREY, M. y JOHNSON. D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman. 1978.

P y EXP

P y EXP son conjuntos de problemas decisionales

EXP contiene casi todos los problemas que intentaréis resolver con un algoritmo

P es una parte de EXP: los problemas que se pueden resolver eficientemente o resolubles en la práctica

Definiciones: tiempo

Dado un programa p y una entrada x, $t_p(x)$ es el tiempo que tarda el programa p con entrada x

Dado un programa p y $n \in N$, $T_p(n)$ es el tiempo máximo que tarda el programa p con una entrada de tamaño n

$$T_p(n)=\max \{t_p(x) \mid |x|=n\}$$

Definiciones: DTIME

Dada una función $f: N \rightarrow N$

$O(f) = \{h: N \rightarrow N \mid \exists\ c>0 \text{ tal que } h(m) \leq c\ f(m)\ \forall\ m\}$

($O(f)$ es el conjunto de funciones acotadas por $c\ f$, para alguna constante c)

Dada una función $f: N \rightarrow N$

$DTIME(f(m)) = \{\Pi \mid \Pi$ es un problema decisional y existe un algoritmo q que lo resuelve y cumple $T_q \in O(f)\}$

Definiciones

P es el conjunto de problemas resolubles en tiempo polinómico:

$$P=\cup_{k\in N} DTIME(m^k)$$

EXP es el conjunto de problemas resolubles en tiempo exponencial:

$$EXP=\cup_{k\in N} DTIME(2^{m^k})$$

Atención

- Los tiempos (DTIME(2^{m^k}), DTIME(m^k)) son cotas superiores
- $P \subseteq EXP$
- Se sabe que $P \neq EXP$ (luego $P \subset EXP$)

Problemas resolubles en la práctica

P es el conjunto de problemas resolubles en la práctica

Si Π no está en P se considera no resoluble de forma eficiente

La mayoría de los algoritmos q que se implementan cumplen $T_q(m) \in O(m^k)$ para algún k

Razones para considerar P "lo resoluble eficientemente"

Los problemas naturales que se sabe que están en P tienen algoritmos "rápidos" (que cumplen $T_q(m) \leq c\ m^k$ para $k \leq 3$ y c pequeña)

Los problemas naturales para los que no se conocen algoritmos polinómicos, tampoco tienen algoritmos conocidos con tiempo por debajo de una exponencial 2^{cm}

Pero ... hay unos pocos problemas con algoritmos exponenciales en caso peor que funcionan rápido en la práctica: Programación lineal y Mochila

$$T_q(m) = \max_{|x|=m} t_q(x)$$

Tesis de Turing-Church: Máquinas de Turing

Cada máquina calcula una función a base de acciones elementales (moverse una casilla, cambiar de estado, escribir un símbolo)

Si definimos el tiempo de una máquina M con entrada x como el número de acciones de M con entrada x desde que empieza hasta que se para ...

$$P = \cup_{k\in N} \mathrm{DTIME}_{MT}(m^k)$$

$$EXP = \cup_{k\in N} \mathrm{DTIME}_{MT}(2^{m^k})$$

Se pueden definir P y EXP con máquinas de Turing

Tesis de Turing-Church

Para cada uno de los modelos conocidos con una definición natural de paso, P y EXP corresponden a tiempo polinómico y exponencial respectivamente

Tesis de Turing-Church: Cualquier modelo razonable y secuencial de cálculo da la misma definición de P y EXP

Tesis de Turing-Church. Pero ...

La conjetura anterior está siendo replanteada a la luz de los recientes estudios sobre el computador cuántico

Este modelo, formulado en 1982 por Deutsch y Lloyd es de naturaleza muy distinta a los otros secuenciales

En 1994, Nishino resolvió con este modelo en tiempo polinómico problemas para los que no se conocen algoritmos polinómicos

Hasta el momento no se ha construído ningún computador cuántico

Tema 6: La clase NP y las reducciones

Problemas comprobables en tiempo polinómico

Tenemos un problema de búsqueda, por ejemplo:

"Dado un mapa M, dos puntos u,v, y una longitud máxima D ¿existe un camino de u a v con longitud como máximo D?"

Problemas comprobables en tiempo polinómico

Tenemos un problema de búsqueda, por ejemplo:

"Dado un mapa M, dos puntos u,v, y una longitud máxima D ¿existe un camino de u a v con longitud como máximo D?"

Si me dan un candidato a camino, se puede comprobar rápidamente si es válido o no. El problema:

"Dado un mapa M, dos puntos u,v, una longitud máxima D, y un candidato a camino C ¿C es un camino de u a v con longitud como máximo D?"

El problema del viajante de comercio (TSP)

"Dados:

- n el número de ciudades,
- la matriz de distancias d(i,j) para 1<=i,j<=n,
- una longitud máxima k

¿existe un camino de que pasa por todas las ciudades con longitud como máximo k?"

Se puede "comprobar" eficientemente una posible solución

El problema del viajante de comercio (TSP)

CompTSP:

"Dado n el número de ciudades, la matriz de distancias d(i,j) para 1<=i,j<=n, una longitud máxima k y un candidato a camino $c=(c_1, \ldots, c_n)$"

¿c es un camino de que pasa por todas las ciudades con longitud como máximo k?"

Este problema es la comprobación de TSP y para resolverlo sólo es necesario comprobar que c pasa por todas las ciudades y calcular la suma de los $d(c_i\ c_{i+1})$

Definición

Un problema A es comprobable en tiempo polinómico si existe un problema $B \in P$ y $k \in N$ tales que

$$x \text{ tiene solución SI para } A$$

$$\Leftrightarrow$$

$$\exists y \; (|y| \in O(|x|^k)) \text{ tal que } (x,y) \text{ tiene solución SI para } B$$

B es la "versión comprobación" de A

NP

NP es el conjunto de problemas comprobables en tiempo polinómico

Se sabe que $P \subseteq NP \subseteq EXP$

No se sabe si alguno de los dos contenidos es estricto

Reducciones

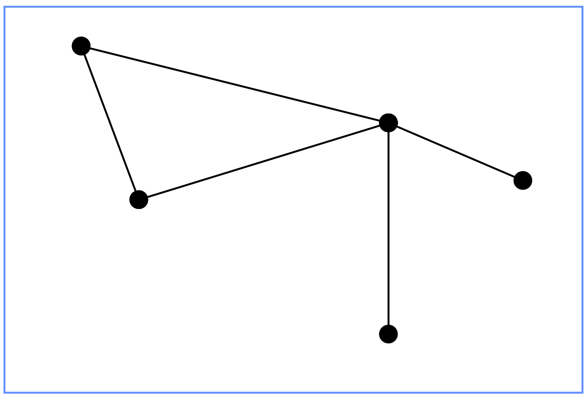
Un problema A se puede reducir a B ($A \leq^p B$) si existe una transformación f de las entradas de A en las entradas de B que cumpla:

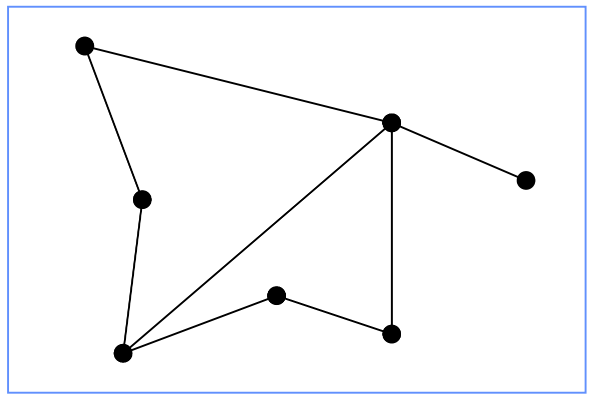
- f es fácil de calcular (tiempo polinómico)
- la solución de el problema A para una entrada x es la misma que la del problema B con entrada f(x) (respeta las soluciones)

Reductiones: ejemplo

HAM es el problema de

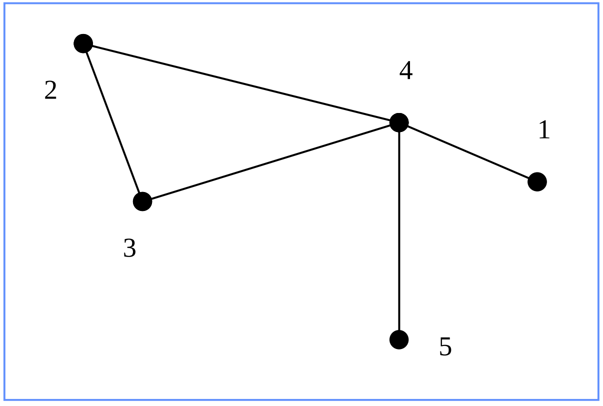
"dados n puntos y algunas conexiones (un grafo) ¿existe un camino que pasa por los n puntos una sola vez?"





Reducciones: ejemplo

Cada entrada de HAM se puede transformar en una entrada equivalente del problema del viajante (TSP):



→

5 ciudades

d(1,4)=d(2,3)=d(2,4)=d(3,4)=d(4,5)=1
resto d(i,j)=100

k=4

HAM se puede reducir a TSP

Reducciones

Se pueden componer:

- **Si $A \leq_p B$ y $B \leq_p C$ entonces $A \leq_p C$**

Sirven para comparar dificultad de los problemas:

- **Si $A \leq_p B$ y $B \in P$ entonces $A \in P$**
- **Si $A \leq_p B$ y $A \notin P$ entonces $A \notin P$**

NP-difíciles y NP-completos

- A es NP-difícil (NP-hard) si para todo $X \in NP$, $X \leq_P A$
  (A es tan difícil como cualquiera de los de NP)

- A es NP-completo si A es NP-difícil y $A \in NP$
  (A es de los más difíciles de NP)

Ejemplos de NP-completos

- HAM y TSP son NP-completos
- Hay una larga lista de NP-completos de todos los temas

Tema 7: NP-completos

Una aplicación práctica

Supón que tu jefe te pide que escribas un algoritmo eficiente para un problema extremadamente importante para tu empresa

Después de horas de romperte la cabeza sólo se te ocurre un algoritmo de “fuerza bruta”, que analizándolo ves que cuesta tiempo exponencial

Te encuentras en una situación muy embarazosa:

> “No puedo encontrar un algoritmo eficiente, me temo que no estoy a la altura”

Una aplicación práctica

Te gustaría poder decirle al jefe:

"No puedo encontrar un algoritmo eficiente, ¡porque no existe!"

Para la mayoría de los problemas, es muy difícil demostrar que son intratables, porque la mayoría de los problemas interesantes que no se saben resolver son NP-completos

- Los NP-completos parecen intratables
- Nadie ha sabido demostrar que los NP-completos son intratables

Una aplicación práctica

Después de estudiar complejidad puedes ser capaz de demostrar que el problema de tu jefe es NP-completo y decirle:

"No puedo encontrar un algoritmo eficiente, pero tampoco pueden un montón de científicos famosos"

O todavía mejor:

"Si pudiera diseñar un algoritmo eficiente para este problema, ¡no estaría trabajando para usted! Me habría ganado el premio de un millón de dólares que da el Instituto Clay"

Una aplicación práctica

Después de la segunda respuesta, tu jefe abandonará la búsqueda

Pero la necesidad de una solución no desaparecerá

Seguramente al demostrar que es NP-completo has aprendido mucho y ahora puedes:

- Olvidar lo de intentar encontrar un algoritmo en tiempo polinómico para el problema
- Buscar un algoritmo eficiente para un problema diferente relacionado con el original
- O bien intentar usar el algoritmo exponencial a ver qué tal funciona con las entradas que te interesan

P versus NP

Ya hemos visto las clases de problemas P y NP:

- En P están los problemas que se pueden resolver en tiempo polinómico
- En NP están los problemas que se pueden comprobar en tiempo polinómico

P versus NP

Sabemos que $P \subseteq NP$

¿P=NP? Esta es una de las preguntas abiertas más importantes en informática

Ver en

http://www.claymath.org/prizeproblems/index.htm

cómo ganar un millón de dólares resolviéndola

P versus NP

¿Cómo la resolverías?

Para intentar $P \neq NP$:

- Demostrar que hay un problema que está en NP pero no en P

Para intentar P=NP:

- Demostrar que todos los problemas de NP se pueden resolver en tiempo polinómico, así que $NP \subseteq P$

P versus NP

Con los NP-completos, esto se simplifica

Para intentar P=NP:

- Demostrar que hay un problema NP-completo que se puede resolver en tiempo polinómico

P versus NP

P ≠ NP es equivalente a "existe un problema NP-completo que no está en P"

P = NP es equivalente a "existe un problema NP-completo que está en P"

Lista de NP-completos

- Existe una larga lista de NP-completos (ver Garey Johnson)

- Añadir un problema a la vista quiere decir que el problema es tan difícil como cualquiera de los que ya están (puedes decirle al jefe "No puedo encontrar un algoritmo eficiente, pero tampoco pueden un montón de científicos famosos")

Lista de NP-completos

Para añadir un problema *C* a la lista hay que:

1) Demostrar que *C* está en NP

2) Hacer una reducción $B \leq^{p} C$ desde un problema B de la lista

Referencias

Lista de NP-completos:

GAREY, M. y JOHNSON. D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman. 1978.

Premio del instituto Clay:

www.claymath.org/prizeproblems/pvsnp.htm

www.claymath.org/prizeproblems/milliondollarminesweeper.htm