LENGUAJES

Unidad 1: Lógica Proposicional

La regla de inferencia más común se denomina modus ponens y funciona de la siguiente manera: Supongamos A y B son sentencias y asumamos que
A → B es verdadera y A es verdadera
Entonces podemos concluir
B es verdadera

Otra regla es llamada modus tollens y funciona de la siguiente manera: Supongamos A y B son sentencias y asumamos que
A → B es verdadera y B es falsa
Entonces podemos concluir
A es falsa

Falacias lógicas: argumentos que parecen válidos pero no lo son.

Una proposición es una oración que es verdadera o falsa.

Sintaxis: ¿Es esta expresión gramaticalmente correcta?
Semántica: ¿Cuál es el significado de la siguiente expresión?

Una fórmula bien formada es:

- un símbolo de verdad
- una letra proposicional
- la negación de una fbf
- la conjunción de dos fbf
- la disyunción de dos fbf
- la implicación de una fbf a otra fbf
- una fbf rodeada de paréntesis

La forma normal de Backus-Naur (BNF) es una notación usada para describir lenguajes formales. Una especificación de BNF es un sistema de reglas de derivación de la forma <símbolo> ::= <expresión> <símbolo> es un no-terminal, y <expresión> es una

secuencias de símbolos o secuencias separadas por la barra vertical, '|' (indicando una opción). La secuencia de símbolos puede contener otros símbolos n oterminales, símbolos terminales, o ε (la cadena nula).

¿Es posible asociar una tabla de verdad a cada fbf? Sí, pero antes debemos establecer una
jerarquía de precedencia para los conectivos:
¬ (mayor precedencia, aplicar primero)
Λ
V
→ (menor precedencia, aplicar último)
Además Λ, V y → son asociativos a izquierda.

Tautologías, Contradicciones y Contingencias

Una fbf cuyos valores de verdad son siempre verdaderos (v) es llamada tautología.
Una fbf cuyos valores de verdad son siempre falsos (f) es llamada contradicción.
Una fbf cuyos valores de verdad son a veces v y otras f es llamada contingencia.

Propiedades de la equivalencia:

≡ es una relación de equivalencia.
Cualquier sub-fbf de una fbf puede ser reemplazada por una fbf equivalente sin cambiar el valor de verdad de la fbf original.

Negation	Disjunction	Conjunction	Implication
$\neg\neg A \equiv A$	$A \vee \text{True} \equiv \text{True}$ $A \vee \text{False} \equiv A$ $A \vee A \equiv A$ $A \vee \neg A \equiv \text{True}$	$A \wedge \text{True} \equiv A$ $A \wedge \text{False} \equiv \text{False}$ $A \wedge A \equiv A$ $A \wedge \neg A \equiv \text{False}$	$A \rightarrow \text{True} \equiv \text{True}$ $A \rightarrow \text{False} \equiv \neg A$ $\text{True} \rightarrow A \equiv A$ $\text{False} \rightarrow A \equiv \text{True}$ $A \rightarrow A \equiv \text{True}$

Some Conversions	Absorption laws
$A \rightarrow B \equiv \neg A \vee B$ $\neg (A \rightarrow B) \equiv A \wedge \neg B$ $A \rightarrow B \equiv A \wedge \neg B \rightarrow \text{False}$ $\wedge$ and $\vee$ are associative. $\wedge$ and $\vee$ are commutative. $\wedge$ and $\vee$ distribute over each other: $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$ $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$	$A \wedge (A \vee B) \equiv A$ $A \vee (A \wedge B) \equiv A$ $A \wedge (\neg A \vee B) \equiv A \wedge B$ $A \vee (\neg A \wedge B) \equiv A \vee B$
	De Morgan's Laws
	$\neg (A \wedge B) \equiv \neg A \vee \neg B$ $\neg (A \vee B) \equiv \neg A \wedge \neg B$

Formas normales

Toda fbf es equivalente a una forma normal disyuntiva (FND)

Toda fbf es equivalente a una forma normal conjuntiva (FNC)

Un literal es una letra proposicional o su negación.
Ejemplos: P, Q, ┐P, ┐Q
Una conjunción fundamental es un literal o una conjunción de dos o más literales.
Ejemplos: P, P∧┐Q

En general podemos definir 2^n conectivos n-arios. (cantidad de filas de la tabla de verdad)

Una forma normal disyuntiva (FND) es una conjunción fundamental, o la disyunción de dos o más conjunciones fundamentales.

Una forma normal conjuntiva (FNC), es la conjunción de dos o más disyunciones fundamentales.

Formas de transformar a FNC/FND

- Método de la tabla de verdad
- Método por equivalencias lógicas

Una forma normal está completa si cada operación fundamental tiene exactamente n literales, uno para cada una de las n letras proposicionales que aparecen en la fbf.

Un conjunto de operadores es completo si se puede expresar cualquier función de verdad utilizando sólo esos operadores.

Dado que se puede expresar cualquier función de verdad utilizando sólo '¬', 'Λ' y 'V', decimos que el conjunto de operadores {¬ , Λ , V} es completo.
Por la propiedad de substitución, dado que {¬ , Λ, V} es un conjunto completo, de conectivos también lo son los conjuntos {¬ , Λ } y {¬ , V}

Conjeturas y pruebas

Es necesario utilizar algún tipo de razonamiento deductivo para verificar la verdad o falsedad de una conjetura. Cuando una conjetura es probada pasa a ser un teorema.

O podríamos encontrar un contraejemplo para mostrar que la conjetura es falsa, un caso en que P sea verdadera y Q falsa.

Técnicas para probar que una conjetura es verdadera o mostrar que es falsa:

- Mostrar que es falsa utilizando un contraejemplo.
- Prueba exhaustiva.
- Prueba directa.
- Contrapositiva.
- Contradicción.
- Prueba por inducción.

The Proof Rules **(6.7)**

Conjunction (Conj) $\frac{A, B}{A \wedge B}$.	*Simplification (Simp)* $\frac{A \wedge B}{A}$ and $\frac{A \wedge B}{B}$.	
Addition (Add) $\frac{A}{A \vee B}$ and $\frac{A}{B \vee A}$.	*Disjunctive Syllogism (DS)* $\frac{A \vee B, \neg A}{B}$ and $\frac{A \vee B, \neg B}{A}$.	
Modus Ponens (MP) $\frac{A, A \rightarrow B}{B}$.	*Conditional Proof (CP)* $\frac{\text{From } A, \text{ derive } B}{A \rightarrow B}$	
Double Negation (DN) $\frac{\neg \neg A}{A}$ and $\frac{A}{\neg \neg A}$.	*Contradiction (Contr)* $\frac{A, \neg A}{\text{False}}$.	*Indirect Proof (IP)* $\frac{\text{From } \neg A, \text{ derive False}}{A}$.

Derived Rules **(6.8)**

Modus Tollens (MT) (Latin for "mode that denies") $\frac{A \to B, \quad \neg B}{\neg A}$.	*Proof by Cases (Cases)* $\frac{A \vee B, \quad A \to C, \quad B \to C}{C}$.
Hypothetical Syllogism (HS) $\frac{A \to B, \quad B \to C}{A \to C}$.	*Constructive Dilemma (CD)* $\frac{A \vee B, \quad A \to C, \quad B \to D}{C \vee D}$.

Cuando las conjeturas se refieren a una colección finita podemos mostrar que la conjetura es verdadera para cada elemento de la colección. A esto se lo llama prueba exhaustiva.
La prueba puede ser:

Formal (construyendo un argumento válido basado en la lógica).
Menos formal (pero precisa).

Regla de inferencia: Patrón sintáctico que establece que a partir de un conjunto de premisas (hipótesis o antecedentes) podemos derivar una conclusión

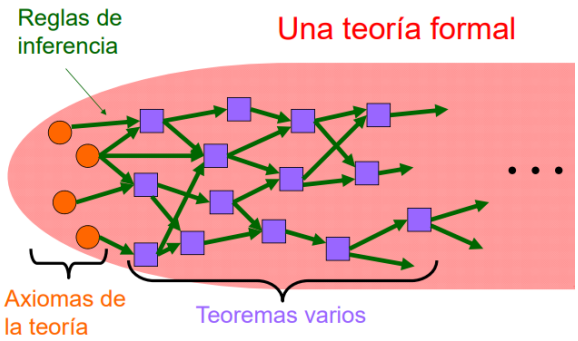
¿Cuándo podemos asegurar que una regla de inferencia preserva la verdad?
Cuando P1 Λ ... Λ Pk → C es una tautología

Un axioma es una fbf que queremos usar como base a partir de la cual podremos razonar. Un axioma es usualmente una fbf que conocemos como verdadera

Una prueba es una secuencia finita de fbf con la propiedad de que cada fbf en la secuencia es:

- un axioma, o
- puede ser inferido de fbf previas en la secuencia

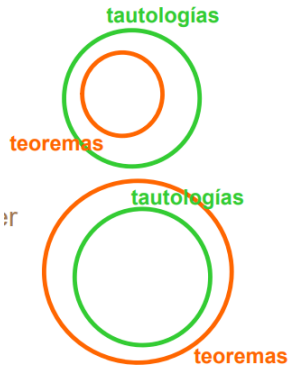
La última fbf en la secuencia es llamada teorema.



Sensatez y completitud

Sensatez: (correctitud/sanidad): Todo

teorema de la teoría debe ser una tautología.
Completitud: Toda tautología debe ser un teorema de la teoría.
Consistente: no es posible deducir una fórmula y su negación.



Unidad 2: Conjuntos y Relaciones

Un conjunto es una colección de elementos u objetos y un elemento es un miembro de un conjunto.

Los elementos del conjunto no tienen impuesto un orden. Listar elementos más de una vez es redundante. Dos conjuntos son iguales si y sólo si contienen los mismos elementos.

Dado los conjuntos A y B, A se dice subconjunto de B si y sólo si todo elemento de A es también elemento de B.

La cardinalidad de un conjunto es el número de elementos dentro del conjunto. La cardinalidad de un conjunto S se denota |S|.

A partir de cada conjunto podemos generar muchos subconjuntos. El conjunto cuyos elementos son todos estos subconjuntos se conoce como el conjunto potencia. Para un conjunto S, usamos p(S) para denotar el conjunto potencia de S.

Para un conjunto con n elementos, el conjunto potencia tiene 2nelementos.

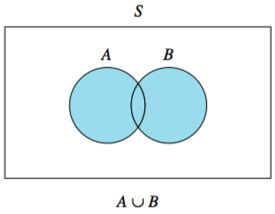
Dos pares ordenados (a,b) y (c,d) son iguales si y sólo si a = c y b = d.

Operaciones

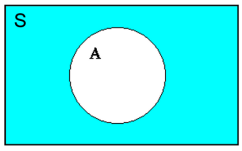
Si para todo x, y en S, xºy ºS, entonces S se dice cerrado bajo la operación º. Si para todo x, y en S, xºy existe y es único, entonces se dice que º está bien definida.

La operación binaria º es una operación binaria sobre un conjunto S si el conjunto es cerrado bajo la operación º y la operación º está bien definida.

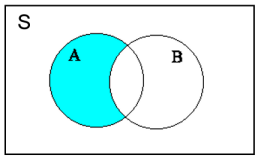
La unión de los conjuntos A y B, denotada AUB es el conjunto que contiene todos los elementos que están en el conjunto A o en el conjunto B. Es decir AUB = {x| x∈A V x∈B}.



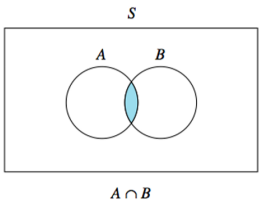
Dados los conjuntos A y B, si A∩B = ∅ , entonces A y B son conjuntos disjuntos. En otras palabras, no existen elementos en A que están también en B.



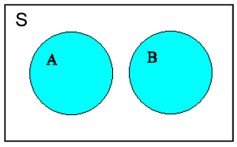
La diferencia A-B es el conjunto de elementos en A que no están en B. Es también conocido como el complemento de B relativo a A. A -B = {x| x∈A Λ x∉B } Notar que A –B = A∩B’



La intersección de los conjuntos A y B, denotada A ∩ B contiene todos los elementos que son comunes a ambos conjuntos, es decir A ∩ B = {x| x∈A Λ x∈B}



Para un conjunto A e (S),el complemento denotado ~A o A’, es el conjunto de todos los elementos que no están en A. A’= {x| x∈S Λ x∉ A}



El producto cartesiano de A y B denotado simbólicamente A X B se define como A X B = {(x,y) | x∈A Λ y∈B }

Propiedades

Propiedad conmutativa (pc)

$A \cup B = B \cup A$ $A \cap B = B \cap A$

Propiedad asociativa (pa)

$A \cup (B \cup C) = (A \cup B) \cup C$

$A \cap (B \cap C) = (A \cap B) \cap C$

Propiedad distributiva (pd)

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

Propiedades de identidad (pi)

$\varnothing \cup A = A \cup \varnothing = A$ $S \cap A = A \cap S = A$

Propiedades de complemento (comp)

$A \cup A' = S$ $A \cap A' = \varnothing$

Ley de doble complemento

$(A')' = A$

Leyes de idempotencia

$A \cup A = A$ $A \cap A = A$

Leyes de absorción

$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$

Representación alternativa de la diferencia

$A - B = A \cap B'$

Inclusión en unión

$A \subseteq A \cup B$

Inclusión de intersección

$A \cap B \subseteq A$

Propiedad transitiva de subconjuntos

Si $A \subseteq B$, y $B \subseteq C$, entonces $A \subseteq C$

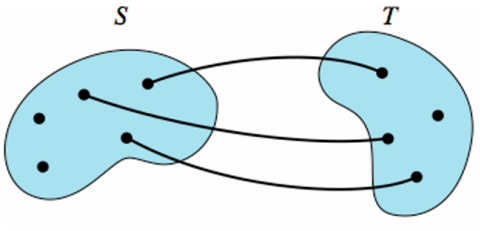
Relaciones binarias

La notación xRy implica que el par ordenado (x,y) satisface la relación R.

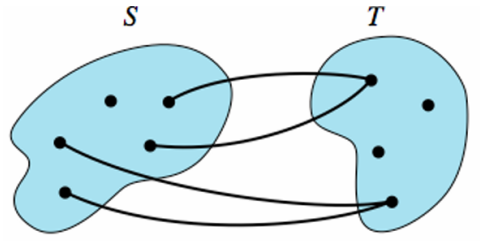
Una relación binaria sobre S es un subconjunto de S X S

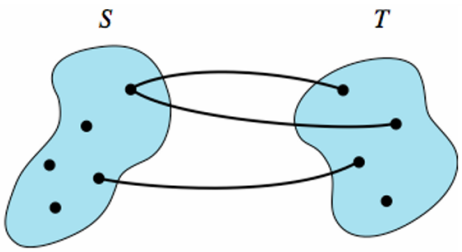
Uno a uno: Si cada primer componente y cada segundo componente aparece sólo una vez en la relación.

Uno a muchos: Si el mismo primer componente aparece con más de un segundo componente.

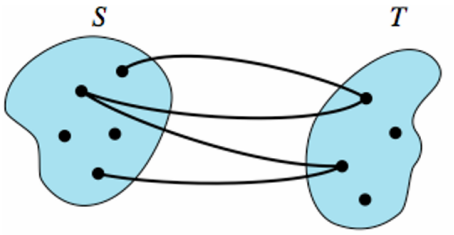


Muchos a uno: Si un segundo componente aparece con más de un primer componente.





Muchos a muchos: Si al menos un primer componente aparece con más de un segundo componente y al menos un segundo componente aparece con más de un primer componente.



Propiedades

Sea R una relación binaria sobre un conjunto S. Entonces:

R reflexiva significa (x∈S—>(x,x)∈R)

R simétrica significa (x∈S, y∈S, (x,y)∈R, (y,x)∈R)

R transitiva significa (x∈S, y∈S, z∈S, (x,y)∈R, (y,z)∈R, (x,z)∈R)

R antisimétrica significa (x∈S, y∈S, (x,y)∈R, (y,x)∈R —> x = y)

R irreflexiva significa (x∈S, (x,x)∉R)

Si R1 es una relación binaria sobre AXB y R2es una relación binaria sobre BxC, entonces la composición de R1y R2 es la relación binaria R1◦R2 sobre AXC definida como a(R1◦R2)c si y sólo si aR1b y bR2c para algún b∈B.

Si R es una relación binaria sobre AxA usamos Rn para denotar la composición de R consigo misma n veces.

R0 ={(a,a)|a∈A}

$R_{n+1} = R_n \circ R$

Una relación binaria R* sobre el conjunto S es la clausura de una relación R sobre S con respecto a la propiedad P si:

R* tiene la propiedad $P \subseteq R^*$

R* es un subconjunto de cualquier otra relación sobre S que incluye R y tiene la propiedad P

Grafos

Un grafo es un conjunto no vacío de nodos y un conjunto de arcos.

Un grafo es un par ordenado (N,A) donde:

N =es un conjunto no vacío de nodos

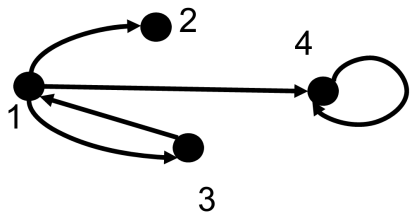
A = es un conjunto de arcos o aristas.

Un grafo dirigido(di-grafo) es par ordenado(N, A) donde:

N = es un conjunto no vacío de nodos

A = es un conjunto de arcos o aristas. Cada arco es un par ordenado (x,y) de nodos donde x es el punto inicial e y es el punto terminal.

Ejemplo: En este ejemplo hay cuatro nodos y cinco arcos. N={1,2,3,4} A = {(1, 2), (1, 4), (1, 3), (3, 1), (4, 4)}



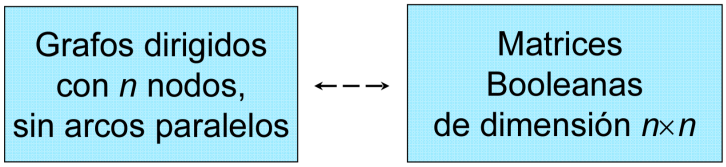
Grafo etiquetado: un grafo cuyos nodos tienen asociada información tal como los nombres de ciudades en un mapa aéreo.

Grafo ponderado: Un grafo donde cada arco tiene asociado un valor numérico o peso. Por ejemplo, un grafo donde se indica las distancias de diferentes rutas en un mapa aéreo.

Multigrafo: un grafo con arcos múltiples, es decir donde dos nodos pueden estar conectados por más de un arco.

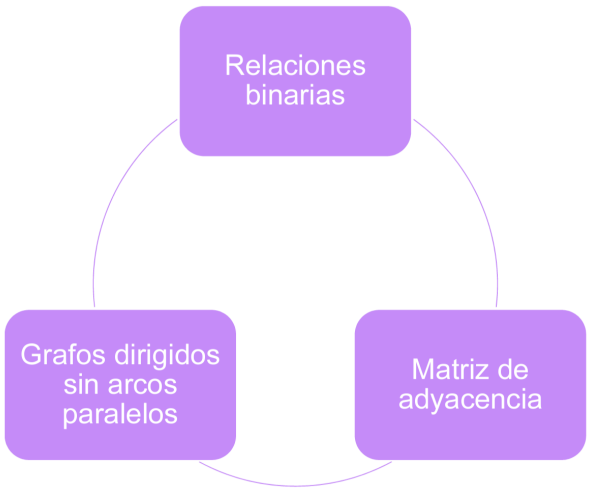
Nos concentraremos en grafos dirigidos, no ponderados y sin arcos paralelos.

La matriz de adyacencia de estos grafos será una matriz booleana (elementos 0s y 1s) de dimensiones nXn, no necesariamente simétrica. Existe una correspondencia uno a uno:



Dado un grafo dirigido G con n nodos y sin arcos paralelos donde N es el conjunto de nodos, podemos definir una relación binaria R como sigue: Si ni, nj ∈ N entonces niRnj↔ existe un arco en G de ni a nj..

Dada una relación binaria R sobre un conjunto S de n elementos podemos construir un grafo G dirigido con n nodos y sin arcos paralelos como sigue: Si ni, nj ∈ S entonces existe un arco en G de ni a nj.. ↔ niRnj



Matriz de adyacencia

Dada una matriz de adyacencia M podemos operarla consigo misma utilizando el producto booleano Θ:

M(0)= I

M(n+1) = M(n)Θ M

R reflexiva: M v M(0)

R simétrica: M v MT

R transitiva (caso finito, n elementos): M v M(2) v M(3) v … v M(n)

Producto booleano Θ: como multiplicar matrices pero en vez de multiplicar hacer conjunción y en vez de sumar hacer disyunción.

Dado un grafo podemos utilizar su matriz de adyacencia para calcular caminos de diferentes longitudes.

Sea M la matriz de adyacencia para la relación R. El algoritmo de Warshall reemplaza M con la matriz de adyacencia correspondiente a la clausura transitiva de R.

Sea M la matriz de adyacencia para un grafo ponderado. El algoritmo de Floyd reemplaza M con la matriz de adyacencia correspondiente a la distancia más corta entre los vértices.

Relación de equivalencia

Una relación binaria sobre un conjunto S que es reflexiva, simétrica y transitiva se llama relación de equivalencia sobre S.

Una partición P de un conjunto S es una colección de subconjuntos Ai$\subseteq$ S, no vacíos y disjuntos, cuya unión es igual a S

1. Ai$\neq\varnothing$ y Ai$\subseteq$ S.
2. La unión de todos los Ai es igual a S. 3.Ai$\cap$ Aj= $\varnothing$, para todo Ai, Aj$\subseteq$ S, tales que i$\neq$ j.

Sea R una relación de equivalencia sobre S y a$\in$S, entonces [a] es el conjunto de elementos de S con los que a se relaciona, y es llamado clase de equivalencia de a:

[a] = {b$\in$S| aRb}

Teorema: Si R es una relación de equivalencia sobre S, entonces R induce una partición P sobre S. Recíprocamente, toda partición P sobre S induce una relación de equivalencia RP.

Sea R una relación de equivalencia. Definimos para todo a$\in$S, [a]p={b$\in$S| (a,b) $\in$R}

[a]p es la clase de equivalencia de a, según la relación R.

El elemento a ∈[a]p se dice un representante de la clase de equivalencia [a]p

El conjunto cociente S/R de S por una relación de equivalencia R es el conjunto de las clases de equivalencia según la relación R.

La función que manda cada elemento de S a su clase de equivalencia se llama aplicación canónica.

¿Dada un relación R, cómo se genera la menor relación de equivalencia R' que contiene a R?Obtenemos la clausura reflexiva, simétrica y transitiva de R

Relaciones de orden

Una relación binaria sobre un conjunto S que es reflexiva, antisimétrica y transitiva se llama relación de orden parcial sobre S.

- orden estricto $x < y$ si y sólo si $x \leq y$ y $x \neq y$
- orden no estricto o reflexivo $x \leq y$ si y sólo si $x < y$ o $x = y$

Un conjunto arbitrario parcialmente ordenado se denota (S, ≤). En casos particulares, ≤ puede representar la relación de "menor o igual", "subconjunto" "divide", etc.

Sea (S, ≤). un conjunto ordenado. Se dice que el elemento b ∈ S cubre al elemento a ∈ S si

1) $a < b$

2) No existe ningún elemento $x \in S$ tal que $a < x < b$.

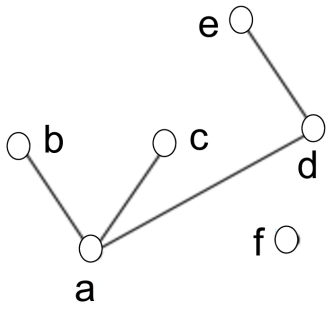
Los diagrama de Hasse son diagramas utilizados para representar visualmente conjuntos parcialmente ordenados con un número finito de elementos.

Dado un conjunto ordenado finito (S,≤): Cada elemento de S es representado por un nodo del diagrama. Si b ∈ S cubre al elemento a ∈ S, el nodo b se coloca por arriba del nodo a y ambos nodos se unen por un segmento de extremosa y b.

Ejemplo

Sea S= {a, b,c,d,e,f}

R = {(a,a), (b,b), (c,c), (d,d), (e,e), (f,f),
(a,b), (a,c), (a,d), (a,e), (d,e)}



Partial Orderings and Equivalence Relations					
Type of Binary Relation	**Reflexive**	**Symmetric**	**Antisymmetric**	**Transitive**	**Important Feature**
Partial ordering	Yes	No	Yes	Yes	Predecessors and successors
Equivalence relation	Yes	Yes	No	Yes	Determines a partition

REPASO PARCIAL 1

Cadenas

Los elementos x e y se dicen comparables si x ≤ y o y ≤ x.

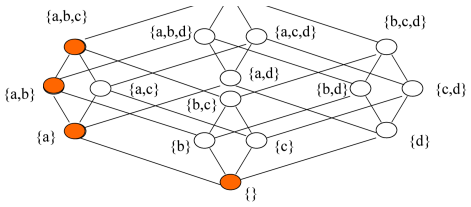
Si x e y no son comparables se dicen incomparables.

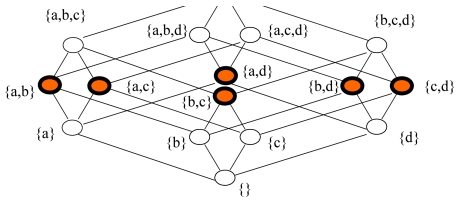
Una cadena (también llamada conjunto totalmente ordenado, linealmente ordenado o simplemente ordenado) es un conjunto parcialmente ordenado en el que todo par de elementos es comparable.

Una anticadena es un conjunto parcialmente ordenado en el que todos los elementos son incomparables.

También llamamos cadena a un subconjunto de un conjunto parcialmente ordenado que a su vez es una cadena.

Y llamamos anticadena a un subconjunto de un conjunto parcialmente ordenado que a su vez es una anticadena.





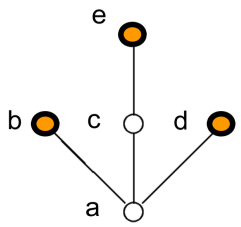
Elementos especiales

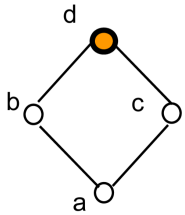
Un elemento m de un conjunto ordenado (S,≤) se dice un elemento maximal de S si:

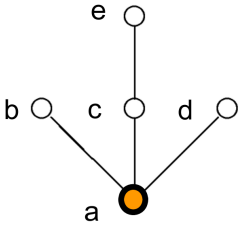
Si x ∈ S es tal que m ≤x entonces x = m

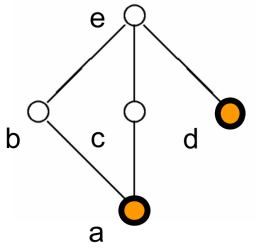
Un elemento m de un conjunto ordenado (S,≤) se dice un elemento minimal de S si:

Si x ∈ S es tal que x ≤m entonces x = m









Ejemplos:

Conjunto N ordenado por la relación ≤: Elemento minimal: 0. No tiene elementos maximales

Conjunto Z ordenado por≤: No existen elementos minimales ni maximales

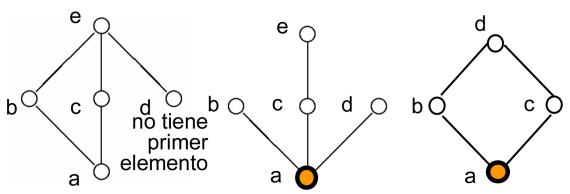
Sea (S,≤) un conjunto ordenado.

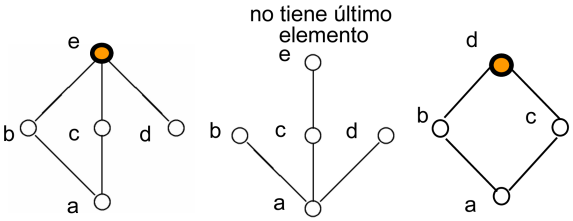
Se dice que un el’emento 0 ∈ S es primer elemento de S si:

0 ≤ x; para todo x ∈ S

Se dice que un elemento 1 ∈ S es último elemento de S si:

x ≤1; para todo x ∈ S

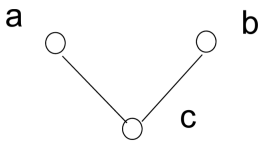




Dados a, b ∈ S se dice que el elemento c ∈ S es el ínfimo de a y b y se nota c = a Λ b si:

- c≤a y c≤b
- Si x ∈ S verifica x≤a y x≤b, entonces x ≤c

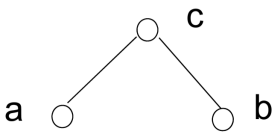
aΛb = c
aΛc = c



Dados a, b ∈ S se dice que el elemento c ∈ S es el supremo de a y b y se nota c = a V b si:

- a≤c y b≤c
- Si x ∈ S verifica a≤x y b≤x, entonces c ≤x

aVb = c
aVc = c



Lema: Todo conjunto ordenado finito tiene, por lo menos, un elemento minimal (maximal).

Lema: Si un conjunto ordenado tiene primer (último) elemento este es único.

Lema: En un conjunto ordenado si existe el ínfimo (supremo) de dos elementos, es único.

Un conjunto ordenado (S,≤), se dice un reticulado inferior si todo par de elementos de S tiene ínfimo.

Un conjunto ordenado (S,≤), se dice un reticulado superior si todo par de elementos de S tiene supremo.

Un conjunto ordenado (S,≤), se dice un reticulado si es un reticulado inferior y superior. Es decir si todo par de elementos de S tiene ínfimo y supremo.

El ordenamiento topológico es un proceso para encontrar un orden total O a partir de un orden parcial R sobre un conjunto finito, de manera tal que O sea una extensión de R. Esto significa que si xRy, entonces xOy (ó $R \subseteq O$)

Este es un proceso de ordenamiento especializado, ya que arrancamos de un orden parcial para llegar a un orden total.

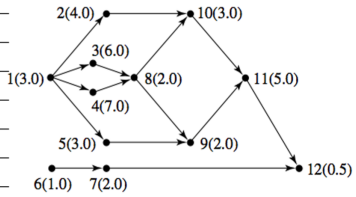
```
TopSort(conjunto finito S; orden parcial sobre S)
i= 1
whileS<> O
  elegir un elemento minimal x i de S;
  S= S- {xi}
  i++
end while
//x1 < x2< … < x nes un orden total que extiende R
write(x1, x2, x3, …, xn)
```

Definimos un orden parcial sobre el conjunto de tareas: x≤y ← → tarea x = tarea y ó tarea x es requisito para la tarea y.

Construimos un diagrama de PERT(program evaluation and review technique) de manera similar a un diagrama de Hasse como sigue:

- Los nodos son tareas.
- Asociamos a los nodos información sobre tiempo requerido para completar la tarea.
- Orientamos el diagrama de tal manera que si x < y, entonces x aparece a la izquierda de y (el diagrama es de izquierda a derecha en lugar de abajo para arriba)
- Camino crítico: camino más largo.

Tarea	pre-requisito	horas para completar
1	ninguna	3.0
2	1	4.0
3	1	6.0
4	1	7.0
5	1	3.0
6	ninguna	1.0
7	6	2.0
8	3,4	2.0
9	5,8	2.0
10	2,8	3.0
11	9,10	5.0
12	7,11	0.5



Unidad 3: Autómatas finitos y lenguajes regulares

Un alfabeto (o vocabulario) Σ es un conjunto finito no vacío de símbolos.

Una cadena (o palabra) sobre Σ es una secuencia finita de símbolos de Σ.

Σ* es el conjunto de todas las cadenas sobre Σ.

Un lenguaje sobre Σ es un subconjunto de Σ*.

Una gramática para un lenguaje puede ser descripta definiendo su proceso generativo.

Gramática

Una gramática estructura de frase (tipo 0) es una 4-tupla

G = (VN,VT,S,P), donde

- VN = conjunto de símbolos no terminales.
- VT= conjunto de símbolos terminales.
- S= símbolo inicial de la gramática.
- P= conjunto finito de producciones de la forma a → b donde a es una cadena sobre VN U VT con al menos un símbolo de VN y b es una cadena sobre VN U VT. (usamos a→λ cuando b es la cadena nula)

Sea G=(VN,VT,S,P) y sean w1 y w2 cadenas sobre VN U VT.

Decimos que w1 deriva directamente w2, y lo notamos w1→ w2 si

- a→b es una producción en P
- w1 contiene una instancia de a
- w2 es obtenida a partir de w1 reemplazando esa instancia de a con b

Si w1,w2,...,wn son cadenas sobre VN U VT y w1→w2, w2→w3,... wn-1→wn, entonces w1 genera (deriva) wn.

Se escribe w1→+ wn si es en 1 o más pasos.

Se escribe w1→* wn si es en 0 o más pasos.

Dada una gramática G, el lenguaje L generado por G, denotado L(G), es el conjunto

L = {w ∈ VT*| S→+ w}. En otras palabras, L es el conjunto de todas las cadenas de terminales generadas a partir de S.

Una vez que una cadena w de terminales ha sido obtenida, ninguna producción puede ser aplicada a w,y w no puede generar otras palabras.

Ejemplo:

Sea L= {a^2n, n ≥ 0}. Una gramática para L es G=(VN,VT,S,P) donde VN= {S}, VT= {a}, y P consiste de las siguientes producciones:

S→λ | aaS

Podemos generar aaaaaa como sigue:

S→ aaS → aaaaS → aaaaaaS → aaaaaa

Clases de gramática

- Estructura de frase (tipo 0). Sin restricciones.
- Sensible al contexto (tipo 1). Para cada producción a→b: |a| ≤ |b| (excepto S→λ).
- Libre de contexto (tipo 2). Para cada producción a→b: a ∈ VN y |a| ≤ |b| (excepto S→λ).
- Regular (tipo 3).Para cada producción a→b: a ∈ VN y b es de la formato tW, donde t ∈ VT y W ∈ VN (excepto S→λ).

Un lenguaje se dice de tipo N si puede ser generado por una gramática de tipo N

El dispositivo computacional más básico se llama autómata finito y se corresponde con los lenguajes de tipo 3

Los lenguajes son conjuntos (finitos o infinitos) de cadenas y por tal motivo podemos operar sobre ellos:

Operaciones:

- unión
- intersección
- complemento
- concatenación
- estrella de Kleene

Sean L1 y L2 dos lenguajes sobre un alfabeto Σ:

L1 U L2 = {w | w ∈ L1 ó w ∈ L2}

L1 ∩ L2 = {w | w ∈ L1 y w ∈ L2}

L1’ = {w | w ∈ Σ* y w ∉ L1}

L1 º 2 = {w1w2| w1 ∈ L1 y w2 ∈ L2}

L1* = {w1w2… wn| wi ∈ L1, n ≥ 0}

Las gramáticas regulares sirven para generar:

- Cualquier lenguaje finito (aunque puede volverse una tarea tediosa).
- Lenguajes infinitos que presentan ciertas regularidades que pueden ser expresada de manera sencilla utilizando las llamadas expresiones regulares (como veremos más adelante).

Ejemplos de lenguajes que pueden ser generados por gramáticas regulares:

- L=∅ El lenguaje vacío.
- L=Σ* para cualquier Σ. El lenguaje universal.
- L={w| w ∈ {a,b}* donde w tiene m a’s seguidas de n b’s para m,n>0}.
- L={w| w ∈ {0,1}* donde w tiene exactamente tres 0’s y tres 1’s}

Las gramáticas libres de contexto tienen ciertas características destacadas:

- Son sencillas, reemplazan un símbolo por vez.
- Pueden ser utilizadas para describir la sintaxis de los lenguajes de programación.
- La derivación de una gramática libre de contexto puede ser vista como un árbol de derivación.

Ejemplo de lenguajes que pueden ser generados por gramáticas libres de contexto:

- Todos los que pueden ser generados por gramáticas regulares
- L ={a^n b^n| n >0}
- L={w|w ∈ {(,)}* donde los paréntesis están balanceados}
- L={w| w ∈ {a,b}* y w=w^r }.

  Observación: Si w = w1w2…wn, wi ∈ {a,b}, entonces wr =wn, wn-1…w1

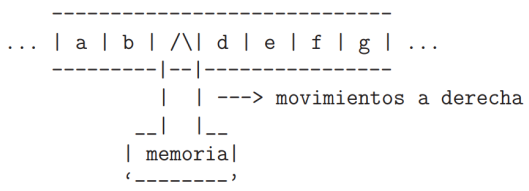
Autómatas finitos y lenguajes regulares

Los autómatas finitos, también denominados máquinas de estados finitos o simplemente autómatas, son entes o máquinas abstractas que estudiaremos desde dos puntos de vista: como dispositivos capaces de determinar la pertenencia de una cadena de símbolos a un lenguaje determinado y como traductores de una cadena en otra. En

ambos casos los autómatas están computando una función, la primera emite una respuesta de tipo si-no y es frecuentemente referida en la jerga computacional como una “función booleana”.

Podemos imaginar un autómata finito como una máquina formada por un conjunto de estados, capaz de leer símbolos de una cinta. Operacionalmente funciona partiendo de un estado y cambiando de estado según el símbolo leído. Prosigue de esta forma hasta haber leído todos los símbolos de la cinta. Un estado puede ser pensado como una secuencia de valores que lo definen unívocamente.

Un autómata finito consta entonces, de una cinta en la que se escribe una cadena de entrada, una memoria que recuerda el estado actual y una cabeza lecto-escritora que luego de leer un símbolo se mueve hacia la derecha y es capaz de decidir dado el estado corriente y el símbolo leído cual es el próximo estado



La memoria central del autómata queda determinada al momento del diseño, ya que el autómata “recuerda” a través de sus estados el conjunto de situaciones que puede tratar. Este conjunto de estados es finito.

Entre los estados siempre podemos distinguir tres categorías, no necesariamente disjuntas: el estado inicial, los estados finales o aceptadores y los estados intermedios. Si una máquina termina de leer todos los símbolos de una cinta y queda en un estado final, decimos que la cadena es aceptada por el autómata. En caso contrario, decimos que la cadena es rechazada por el autómata.

El autómata deberá estar construido respetando estas propiedades y así podrá reconocer al lenguaje L. Ya que los estados le permiten al autómata “recordar” situaciones que se desean diferenciar, es crucial en su diseño saber distinguir el conjunto de situaciones mínimas que se desea controlar.

Un autómata finito es un modelo que captura las características de una computadora.

- Autómatas Finitos Reconocedores (AFR): Son capaces de reconocer lenguajes regulares (exactamente el mismo lenguaje generado por gramáticas de tipo 3)
- Autómatas Finitos Traductores: Toman una entrada y la traducen en una salida
    - Autómatas de Moore: las salidas van asociadas a los estados

- Autómatas de Mealy: las salidas van asociadas a las transiciones

Autómatas Finitos Reconocedores

Un autómata finito reconocedor(AFR) es una quíntupla M = (S, Σ, δ, s0, F), donde:

S es un conjunto finito de estados S≠λ

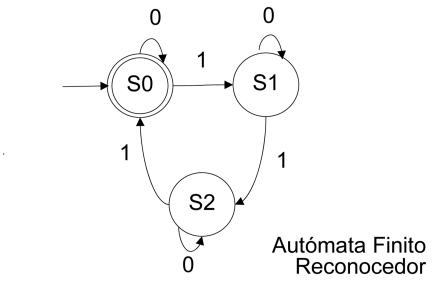
Σ es el alfabeto de entrada

δ: S x Σ →S es la función de transición

s0 es el estado inicial s0 ∈ S

F es el conjunto de estados finales o aceptadores (F⊆ S)

La función de transición δ es una función binaria: de acuerdo a un estado de S y un elemento de Σ, nos indica a que otro estado el autómata puede evolucionar.



Reconoce el lenguaje sobre Σ={0,1} con un número múltiplo de 3 de 1's

Representación de Autómatas mediante Grafos Dirigidos

Un grafo dirigido es una terna (N, A, V) donde N es el conjunto de vértices o nodos, A es el conjunto de arcos y V es la función que asocia a cada arco a ∈ A con un par ordenado de vértices. Aquí utilizaremos multidigrafos para representar autómatas finitos y los llamaremos Diagrama de Estados. Las convenciones a adoptar para tal fin serán las siguientes:

- Existe un nodo n ∈ N del grafo por cada estado s ∈ S.
- Existe un arco a ∈ A tal que V (a)= (si, sk) si y solo si δ(si, a) = sk, donde si, sk ∈ S, a ∈ Σ.
- Debe notarse que cada arco puede tener más de un rótulo.
- El estado inicial será indicado con el símbolo"→" y los estados finales con

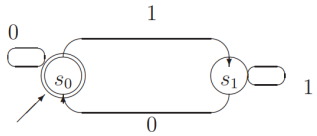


Figura 1.3: AF para reconocer numeros binarios pares.

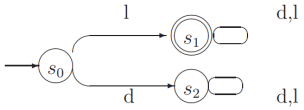


Figura 1.2: AF para reconocimiento de identificadores de Pascal.

un doble círculo.

Otro aspecto de la definición de autómata finito a tener en cuenta es la posibilidad de incorporar
a la definición de la función de transición la posibilidad de usar λ como símbolo a ser procesado
por el autómata.

Autómatas Finitos Traductores

Podría ser util considerar autómatas finitos para computar funciones, donde la función de salida en lugar de ser la aceptación o rechazo de una cadena sea una función más compleja que mapea o traduce cadenas de entrada en cadenas de salida.

Los autómatas finitos son como un modelo de computadora restringida, ya que el objetivo es realizar un calculo a partir de una cadena. Para ello será necesario cambiar en la definición de M el conjunto de estados aceptadores por una función de salida f0; donde f0 está definida del conjunto de estados en el alfabeto de salida. Esto conduce a los denominados Autómatas Finitos Traductores (AFT), de los cuales consideraremos dos clases, los AFT de Moore y los AFT de Mealy.

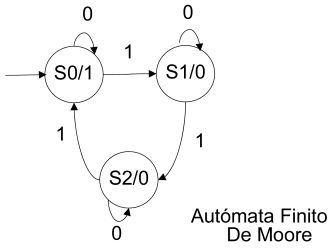
Un autómata finito traductor de Moore es una séxtupla

M = (S, Σ, Z, δ, s0, f0) donde f0 : S → Z. Aquí Z es el alfabeto de salida y puede variar de acuerdo al problema planteado.

Los demás elementos se interpretan como en el caso de un AFR.

La función de salida se indicará en la tabla de estados con una columna extra:

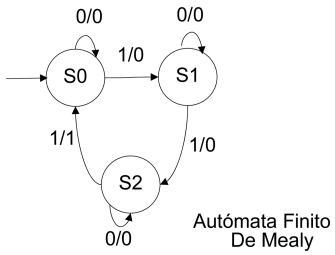
$estados$	$entradas$	$salida$
s_0	$\dots$	z_0
s_1	$\dots$	z_1
$\dots$	$\dots$	$\dots$



Emite 1 mientras el número de 1's leídos hasta el momento sea múltiplo de 3, de lo contrario emite 0

Un autómata finito traductor de Mealy es una séxtupla
M = (S, Σ, Z, δ, s0, f0) donde

f0 : S × Σ → Z y los demás elementos se interpretan como en el caso de AFT de Moore.



Emite 1 cuando el número de 1's leídos se vuelve múltiplo de 3 (≠0), de lo contrario emite 0

REPASO PARCIAL 2

AFRND

Un autómata finito reconocedor no determinista (AFRND) es una quíntupla M = (S, Σ, δ, s0, F), donde:

S es un conjunto finito de estados, S ≠ .

Σ es el alfabeto de entrada.

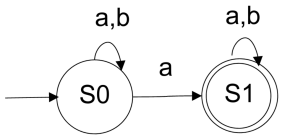
δ: S x Σ →2S es la función de transición.

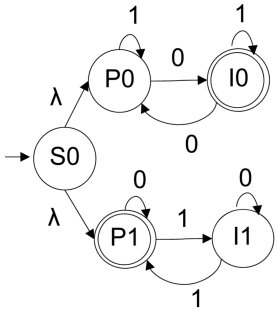
s0 es el estado inicial, s0 ∈ S.

F es el conjunto de estados finales o aceptadores (F S).

Un autómata finito reconocedor no determinista con transiciones λ (AFRND λ) es una quíntupla

M = (S, Σ, δ, s0, F), donde: δ: S x (Σ U {λ}) →2S es la función de transición.





Número impar de ceros o par de unos

El lenguaje reconocido por el AFRND M = (S, Σ, δ, s0, F),se denota L(M) y se define como:

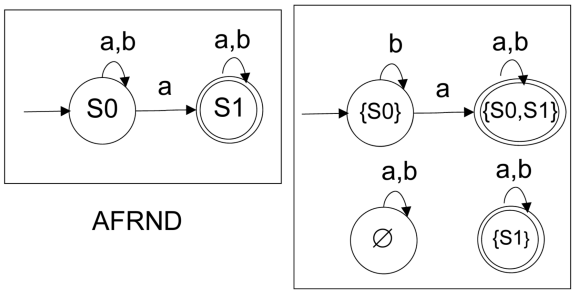
L(M)={w | δ*(s0,w) $\subseteq$ F ≠ ∅ } Es decir, el lenguaje de M es el conjunto de cadenas w tales que comenzando del estado inicial de M nos llevan a algún conjunto de estados que contenga al menos un estado final.

A partir de un AFRND N = (S, Σ, δ, s0, F) obtenemos un AFR determinista D = (Sd, Σ, δd, {s0}, Fd)

Sd = 2S

Fd es el conjunto de subconjuntos Q$\subseteq$ Sd tal que Q $\in$ F ≠∅

Para cada conjunto Q$\subseteq$ S y para cada símbolo a $\in$ Σ



AFRND

AFRD

cadenas sobre Σ = {a,b} que contienen al menos una a.

Algoritmo de minimización

Entrada: un AFD completo M=(S, Σ, δ, s0,f0)

Salida: un AF M' equivalente a M minimizado

1) T ← S -{estados inalcanzables en S}

2) Particionar T en clases formadas por estados 0 equivalentes (aceptadores y no aceptadores)

3) k ← 0

4) Repetir

Determinar clases (k+1)-equiv. como refinamiento de las k-equivalentes

si, sj son k-equivalentes y δ(si,a), δ(sj,a) (comparar de a dos, si recibiendo el mismo input van a un estado de la misma clase)

k ← k+1

Hasta Clases((k+1)-equiv.)=Clases(k-equiv.)

5) Definir M' usando las clases k-equivalentes halladas

TABLE 8.7

Present State	Next State (Present Input 0)	Next State (Present Input 1)	Output
0	3	1	1
1	4	1	0
2	3	0	1
3	2	3	0
4	1	0	1

0-equivalentes:
{0, 2, 4}, {1, 3}
1-equivalentes:
{0}, {2, 4}, {1, 3}
2-equivalentes:
{0}, {2, 4}, {1, 3}

A = {0},
B = {2, 4},
C = {1, 3}

TABLE 8.8

Present State	Next State (Present Input 0)	Next State (Present Input 1)	Output
A	*C*	*C*	1
B	*C*	*A*	1
C	*B*	*C*	0

Expresiones regulares

Las expresiones regulares(ER) sobre un alfabeto Σ son cadenas sobre el alfabeto

A = Σ U {(,),•, +,, λ,∅} definidas recursivamente como sigue:

1. λ es una ER
2. ∅ es una ER
3. Todo símbolo a ∈ Σ es una ER
4. 4. Si α y β son ER, entonces (α), α•β, α+β y α también son ER

Construimos la función L tal que si αes una ER entonces L(α) es el lenguaje representado por α:

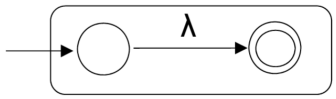
1. L(λ) = {λ }.
2. L(∅) = ∅.
3. Para cada símbolo a Σ, L(a) = {a}.
4. Si α y β son ER, entonces L(α•β) = L(α)•L(β), L(α+β) = L(α) U L(β), L(α*) = L(α)*

Teorema de Kleene

Un lenguaje L puede ser expresado por una ER si y solo si es reconocido por un AF.

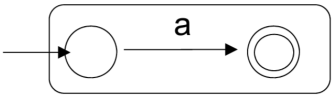
Parte 1: Dada una ER, construimos un AF.

λ

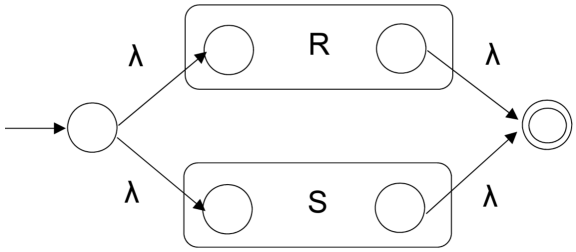


∅

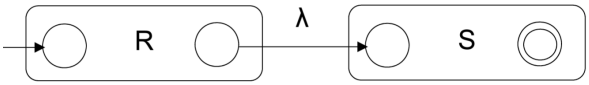
a ∈ Σ



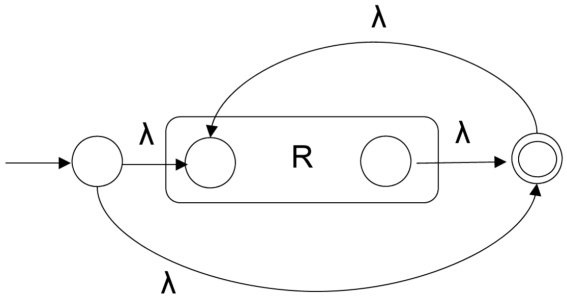
casos base



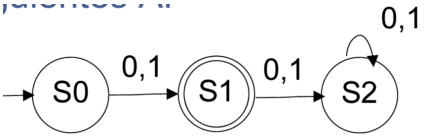
a+b



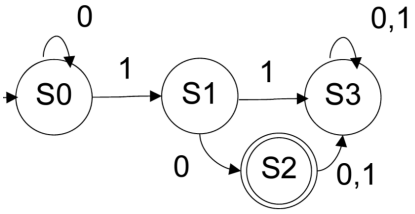
ab



a*



0+1

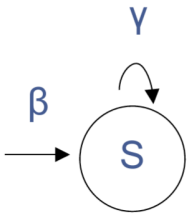


0*10

ejemplos

Parte 2: Dado un AF, construimos una ER.

Lema: Si ω, βy γson ER sobre un alfabeto Σ, γ≠ λ, entonces la ecuación ω= β+ ωγ tiene una única solución y está dada por ω= βγ*



asociar cada estado con una ER

$\omega S = \beta + \omega S\gamma \rightarrow \omega s\gamma^*$

Algoritmo

Entrada: un AF M= (S, Σ, δ, s0 , F)

Salida: una ER que denota el LR reconocido por M

Paso 1: Plantear una ecuación por cada estado, como unión de n términos ωsi= ωsj1• a1+ ωsj2• a2....

Para δ(sj,a)=si, uno de los términos de la ecuación ω si será ωsj•a.

En la ecuación para el estado inicial se agrega el término λ.

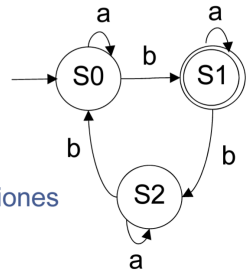
Paso 2: despejar ecuaciones según lema ω=β+ωγ entonces ω = βγ *

Paso 3: La ER es la unión de las soluciones para todos los estados aceptadores del AF

M= (S, Σ, δ, s_0 , F)

S = {s_0, s_1, s_2}, Σ = {a,b}, **F** ={s_1}

δ	a	b
s_0	s_0	s_1
s_1	s_1	s_2
s_2	s_2	s_0



Paso 1: Planteamos ecuaciones

$\omega_{S0} = \omega_{S0}a + \omega_{S2}b + \lambda$

$\omega_{S1} = \omega_{S1}a + \omega_{S0}b$

$\omega_{S2} = \omega_{S2}a + \omega_{S1}b$

Paso 2: ω=β+ωγ entonces ω = βγ *

$\omega_{S0} = \omega_{S0}a + \omega_{S2}b + \lambda \Rightarrow \omega_{S0} = (\omega_{S2}b + \lambda)a^*$
γ=a y β=$\omega_{S2}b + \lambda$

$\omega_{S1} = \omega_{S1}a + \omega_{S0}b \Rightarrow \omega_{S1} = (\omega_{S0}b)a^*$
γ=a y β=$\omega_{S0}b$

$\omega_{S2} = \omega_{S2}a + \omega_{S1}b \Rightarrow \omega_{S2} = (\omega_{S1}b)a^*$
γ=a y β=$\omega_{S1}b$

$\omega_{S1} = (\omega_{S0}b)a^*$

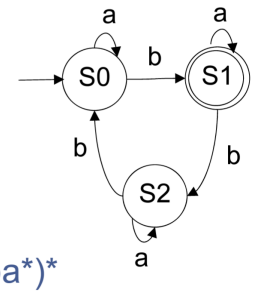
$\omega_{S1} = ((\omega_{S2}b + \lambda)a^*b)a^*$

$\omega_{S1} = (((\omega_{S1}b)a^*b + \lambda)a^*b)a^* = \omega_{S1}ba^*ba^*ba^* + a^*ba^*$

$\omega_{S1} = a^*ba^*(ba^*ba^*ba^*)^*$

Paso 3: La ER es la unión de las soluciones para todos los estados aceptadores del AF.

ω_{S1}=a*ba*(ba*ba*ba*)*



L(M) =a*ba*(ba*ba*ba*)*

Propiedades de los lenguajes regulares

Lema: La clase de los lenguajes aceptados por AF, es decir la clase de los lenguajes regulares, es cerrada bajo:

- unión

  Sean L y M dos lenguajes regulares. Podemos obtener su expresión regular tal que L = L(R) y M = R(S)

  Luego L U M = L(R+S) por definición de la operación + de las expresiones regulares

  L U M es regular

  Los lenguajes regulares son cerrados bajo unión

- complemento

  Sea L = L(A) para algún AFRD A = (S, Σ, δ, s0, F), luego L' = L(B) donde B es el AFRD B = (S, Σ, δ, s0, S-F). Esto es, B es igual que A, pero los estados aceptadores de A son no aceptadores en B y viceversa.

  Entonces w está en L(B) sí y sólo sí δ(So, w) está en S-F que sólo ocurre si w no está en L(A). Como A es un autómata determinista, cada cadena o pertenece a F o a S-F

  Luego, L' es regular

  Los lenguajes regulares son cerrados bajo complemento

- intersección

Sean L y M dos lenguajes regulares. Por DeMorgan L∩M = (L' U M')' y porque los lenguajes regulares son cerrados bajo complemento y unión, utilizamos la identidad para obtener la intersección.

Luego, L∩M es regular

Los lenguajes regulares son cerrados bajo intersección

- concatenación
- estrella de Kleene

Equivalencia entre gramáticas regulares y AF

Si G una gramática regular podemos obtener un autómata M tal que L(M) = L(G), y viceversa, dado el autómata M podemos construir la gramática G.

Parte 1: Dada una gramática G mostraremos como construir un autómata M tal que L(G) = L(M)

Dada G = (VN, VT,S,P) construimos M = (K, VT, δ, S, F) donde K=VN {A} con A nuevo (no perteneciente a VN)

Si P tiene la producción S → λ entonces F ={A,S} si no F = {A}.

Al definir δ debemos considerar los siguientes casos

- si B → a ∈ P entonces:

  {A} ⊆ δ(B,a)

  a∈VT; B∈VN

- si B → aC ∈ P entonces:

  {C}⊆δ(B,a)

  a∈VT; B,C∈VN

Parte 2: Dado un autómata M mostraremos como construir una gramática G tal que L(M) = L(G)

Dado M = (S, Σ, δ, s0, F) un AF determinista construimos G=(S, Σ, s0,P) donde P está formado por

s0 → λ, si s0 ∈ F

B →aC, si δ(B,a) = C

B →a, si δ(B,a) =C y C ∈ F

Ejemplo

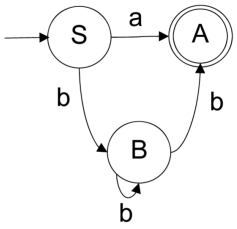
$G=(V_N, V_T,S,P)$

V_N ={S,B}

V_T={a,b}

S → a | bB

B → bB | b



M = (K, V_T, **δ,** S, F)

K ={S,A,B}

F = {A}

δ(S,a)={A}
δ(S,b)={B}
δ(B,b)={A,B}

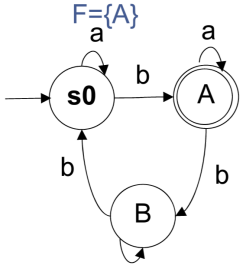
GR → AFR

Ejemplo

M = (S, Σ, **δ, s_0, F**)

Σ ={a,b}

F={A}



AFR → GR

G=(S, Σ, **s_0**,P)

P:

s_0 → a**s_0** | bA| b

A → aA |bB|a

B → b**s_0**| aB

Unidad 4: Introducción a los Modelos de Computación

Primer principio de inducción

Sea P(n) una propiedad definida para el entero n y supongamos

1. P(1) [caso base]
2. Para todo entero positivo k, [paso inductivo]
3. P(k)→ P(k+1) [hipótesis inductiva]

Entonces P(n) para todo entero n a partir de 1.

Los métodos de inducción pueden ser aplicados a estructuras diferentes a los enteros. Por ejemplo matrices, árboles secuencias, etc.

La propiedad crucial es que exista un principio de buen ordenamiento: debe existir una noción de tamaño de tal manera que todos los objetos tengan un tamaño finito y cada conjunto de objetos deberá contener un objeto menor (el más chico de todos).

Árboles binarios

- Árbol binario: cada nodo puede tener hasta dos hijos.
- Árbol binario lleno (full): cada nodo tiene cero o dos hijos.

- Árbol binario completo (complete): cada nivel del árbol binario está completamente lleno, excepto posiblemente el último nivel. Si el último nivel no estuviese lleno, solo podrían faltar nodos del lado derecho de dicho nivel.
- Árbol binario perfecto (perfect): cada nodo tiene cero o dos hijos y todas las hojas tienen la misma profundidad o nivel.

Ejemplo

¿Cuántos nodos tiene un árbol binario perfecto de altura n?

Probar Nodos(n)= 2n+1-1

Prueba:

Tras realizar un análisis descubrimos la siguiente definición recursiva:

Nodos(0) = 1 Nodos(n) = 2*Nodos(n-1) + 1

Caso base: Tomaremos como caso base un árbol de altura 0 (solo contiene el nodo raíz).

Notemos que Nodos(0) = 20+1-1 = 1 y por lo tanto se verifica la propiedad.

Paso inductivo:

Asumamos como HI: Nodos(k)= 2k+1-1

Queremos probar: Nodos(k+1)= 2k+2-1.

Nodos(k+1) = (Def)

2*Nodos(k) +1=(HI)

2*(2k+1-1)+1= 2k+2-1

Segundo principio de inducción

Sea P(n) una propiedad definida para el entero n y supongamos:

1. P(1) [caso base]
2. Si P(r) es verdadero para todo entero positivo r,1≤r≤k implica [hipótesis inductiva]

   P(k+1) es verdadero [paso inductivo]

Entonces P(n) para todo entero n a partir de 1.

Ejemplo

Mostar que podemos sumar cualquier monto de 8¢o más utilizando solo estampillas de 3¢ y 5¢.

Caso base: P(8) 8 = 5+3 También lo vemos para dos casos adicionales: P(9) 9=3+3+3 y P(10) 10=5+5

Hipótesis inductiva: P(r) vale para 8 ≤r ≤ k Paso inductivo: Probemos P(k+1) donde k+1 es al menos 11. Si k+1 ≥11, entonces k+1-3 = k-2≥8.

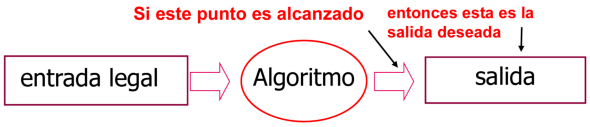
Por hipótesis inductiva P(k-2) es verdadera y por lo tanto k-2 puede ser escrito como suma de 3's y 5's. Sumando 3 obtenemos k+1 como suma de 3's y 5's.

Luego P(k+1) se verifica.

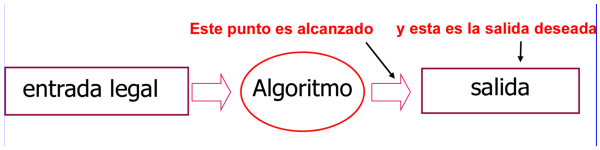
Correctitud de algoritmos

Un algoritmo es correcto si para cualquier entrada (input) legal termina y produce la salida (output) deseada. Las pruebas de correctitud no pueden ser automatizadas.

Correctitud parcial



Correctitud total



Para probar correctitud parcial asociamos una serie de aserciones (sentencias sobre el estado de la ejecución) a puntos específicos del algoritmo.

Precondiciones: aserciones que deben ser válidas antes de ejecutarse un algoritmo o subrutina.

Poscondiciones: aserciones que deben ser válidas luego de ejecutarse un algoritmo o subrutina.

Asignaciones y condicionales

Probar desde abajo hacia arriba, que con la poscondición podemos llegar a la precondición

En condicionales probar que en ambos casos (if y else) se cumple la correctitud

$\{x \neq y\}$
if $x >= y$ **then**
  max $= x$
else
  max $= y$
end if

The desired postcondition reflects the definition of the maximum, ($x > y$ and max $= x$) or ($x < y$ and max $= y$). The two implications to prove are

$\{x \neq y \text{ and } x \geq y\}$ max $= x$ $\{(x > y \text{ and max} = x) \text{ or } (x < y \text{ and max} = y)\}$

and

$\{x \neq y \text{ and } x < y\}$ max $= y$ $\{(x > y \text{ and max} = x) \text{ or } (x < y \text{ and max} = y)\}$

Using the assignment rule on the first case (substituting x for max in the postcondition) would give the precondition

$$(x > y \wedge x = x) \vee (x < y \wedge x = y)$$

Since the second disjunct is always false, this is equivalent to

$$(x > y \wedge x = x)$$

which in turn is equivalent to

$$x > y \quad \text{or} \quad x \neq y \text{ and } x \geq y$$

The second implication is proved similarly.

Ciclos

Encontrar invariante

```
Producto(enteros no negativos x e y)
variables locales: enteros i,j
    i=0
    j=0
    while i<> x do
          j=j+y                    Q: j=i*y
          i=i+1
    end while
    return j
```

Probar invariante por inducción

Prueba de Q: j=i*y utilizando inducción

Caso base:
  Q(0): $j_0=i_0*y$

Hipótesis inductiva:
  Asumamos Q(k): $j_k=i_k*y$

Paso inductivo:
  Probemos Q(k+1): $j_{k+1}=i_{k+1}*y$
    Por el programa $j_{k+1}=j_k+y$, $i_{k+1}=i_k+1$
    Luego j_{k+1} $= j_k+y$ (por programa)
    $= i_k*y+y$ (por HI)
    $= (i_k+1)*y$
    $= i_{k+1}*y$ (por programa)

Verificar que la función computa el valor correcto

Conjuntos contables

Sean A y B conjuntos. Si existe una biyección entre A y B denotaremos este hecho escribiendo
|A| = |B|.
En este caso diremos que A y B tienen el mismo tamaño o la misma cardinalidad o que son equipotentes.

Un conjunto se dice contable si es finito o si existe una biyección entre el mismo y N. En el último caso se dice que el conjunto es infinitamente contable.

En términos de tamaño decimos que un conjunto S es contable si |S| = |{0, 1, . . . , n – 1}| para algún número natural n o |S| = |N|.

Si un conjunto no es contable, decimos que es incontable.

Propiedades de conjuntos contables
a. Todo subconjunto de N es contable.
b. S es contable si y sólo si |S| ≤ |N|.
c. Cualquier subconjunto de un conjunto contable es contable.
d. Cualquier imagen de un conjunto contable es contable.

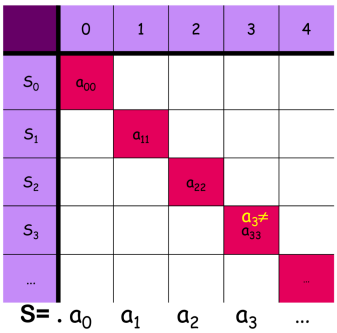
Lema: Sean S0, S1, S2, .. una secuencia de conjuntos contables, entonces la unión de todos ellos es contable.

Diagonalización

Construimos A a partir de una secuencia de elementos diagonales cambiando cada elemento de tal manera que an /= ann para todo n. Entonces A difiere de S en cada elemento n-ésimo.

Por ejemplo, tomando dos elementos x e y ∈ A definimos

$$a_n = \begin{cases} x & si \quad a_{nn} = y \\ y & si \quad a_{nn} \neq y \end{cases}$$



C es contable

1. definir función biyectiva tal que f: N → C (o viceversa)
2. probar biyectividad
3. Usar secuencia de conjuntos contables

Conjuntos infinitamente contables: N, Z, Q, Σ*, NxN, etc.

Ejemplo, Σ* es contable para Σ = {0}

Asumiendo que 0 ∈ N definimos f tal que f: Σ* → N

f(X) = |x| (es decir, la longitud de la cadena)

f(x) es sobreyectiva pues para cualquier n ∈ N siempre hay una cadena de esa longitud, f(x) es inyectiva pues dos cadenas distintas tienen distinta longitud.

Luego, tenemos una biyección entre N y Σ*, por lo tanto Σ* es contable

C es incontable

1. probar por el absurdo
2. usar diagonalización

Conjuntos incontables: R, p(N), Z^w = { w | w es infinita sobre Σ}, p(Σ*), etc.

Ejemplo, qpq R es incontable

Es suficiente probar que U = (0,1) es incontable

Asumamos por el absurdo que U es contable, entonces podemos listar todos los elementos entre 0 y 1 como una secuencia contable r1,r2,r3…. rn. Cada número real

puede ser representado por decimales infinitos, entonces para cada n existe una representación n = 0d0,d1,d2,d3 …. dn, luego por diagonalización podemos concluir que existe un decimal infinito que no esta en la lista. Por ejemplo, elegimos los dígitos 1 y 2 para construir el número s = 0s0s1s2s3… donde

sk = 1 si dkk = 2

2 si dkk ≠ 2

Entonces 0 < s < k y s es diferente de cada rn en la n-ésima posición. S no está en la lista, contrario a la asunción de que podíamos listar todos los elementos de U.

U es incontable

Luego, R es incontable pues U ⊆ R

Límites de la computabilidad

Teorema: El conjunto de programas que pueden ser escritos utilizando lenguajes de programación es infinitamente contable.

Dado que existe un número incontable de funciones N → N no todas son computables. Existen programas para calcular sólo un conjunto contable de estas funciones.

Ejemplo: No es posible computar cualquier real hasta un número arbitrario de cifras decimales debido a que existe un número contable de programas y un número incontable de reales.

- Los números racionales pueden ser computados.
- Algunos números irracionales pueden ser computados (π,√2, etc).