

# 2009-2010

Caputo Luca - Costante Luca – Davino Cristiano - Di  
Giacomo Ivan – Giordano Antonio - Palo Umberto –  
Vitale Pasquale

**[TEORIA DELLA COMPUTAZIONE 1]**

## Sommario

CAPITOLO 1: AUTOMI .....	3
Introduzione .....	3
Introduzione alle dimostrazioni formali .....	3
Induzioni sugli interi .....	3
I numeri interi come concetti definiti ricorsivamente.....	4
Forme più generali di induzione sugli interi .....	4
Definizione ricorsiva di espressioni aritmetiche.....	4
Induzioni strutturali.....	5
Induzione mutua.....	5
I Quantificatori.....	7
I concetti centrali della teoria degli automi .....	7
Linguaggi.....	9
ESERCIZI CAPITOLO 1 .....	10
Quantificatori.....	10
Esiste.....	10
Per ogni.....	10
Parole.....	10
Operazioni insiemistiche .....	10
CAPITOLO 2 – AUTOMI A STATI FINITI.....	11
Automi a stati finiti deterministici .....	11
Rappresentazioni degli automi.....	11
Automi a stati finiti deterministici (DFA).....	12
Definizione di automa a stati finiti deterministico .....	12
Elaborazione di stringhe in un DFA.....	13
Estensione della funzione di transizione alle stringhe .....	14
Il linguaggio di un DFA .....	16
Automi a stati finiti non deterministici.....	17
Descrizione informale degli automi a stati finiti non deterministici .....	17
Trasformazione da DFA a NFA .....	18
Definizione di automa a stati finiti non deterministico .....	19
La funzione di transizione estesa.....	19
Il linguaggio di un NFA .....	20
Esercizio .....	21
Equivalenza di automi a stati finiti deterministici e non deterministici .....	22

# Teoria della Computazione 1

Caputo L. - Costante L. – Davino C. - Di Giacomo I. – Giordano A. - Palo U. – Vitale P.

Un caso sfavorevole di costruzione per sottoinsiemi.....	25
Principio della piccionaia .....	26
Automi a stati finiti con epsilon-transizioni.....	27
Uso delle $\epsilon$ -transizioni .....	27
ESERCIZI CAPITOLO 2 .....	31
CAPITOLO 3-4 – ESPRESSIONI E LINGUAGGI REGOLARI .....	33
Pumping Lemma.....	33
Operazioni sui linguaggi.....	35
Espressioni Regolari.....	35
Regole di priorità .....	36
Proprietà di chiusura dei linguaggi regolari.....	36
Automi a stati finiti ed espressioni regolari.....	36
Teorema sui linguaggi regolari: dai DFA alle espressioni regolari.....	37
Conversione di espressioni regolari in automi .....	41
CAPITOLO 5 - GRAMMATICHE E LINGUAGGI CONTEXT FREE.....	45
Def(ricorsiva) di reverse: .....	45
Grammatiche .....	46
Derivazioni per mezzo di una grammatica .....	47
Derivazioni a sinistra e a destra.....	48
Proprietà delle grammatiche.....	48
Albero sintattico .....	49
Costruzione di alberi sintattici.....	49
Il Prodotto di un albero sintattico .....	50
ESERCIZI DI ESAME .....	54
ELENCO DEFINIZIONI .....	61
DFA .....	61
NFA .....	61
$\epsilon - NFA$ .....	64
ESPRESSIONI E LINGUAGGI REGOLARI .....	64
Grammatiche e linguaggi CF.....	72

## CAPITOLO 1: AUTOMI

### Introduzione

La teoria degli automi è lo studio di dispositivi astratti di calcolo o "macchine". Negli anni '30, prima dell'avvento dei computer, A. Turing studiò una macchina astratta che aveva tutte le capacità degli elaboratori odierni, almeno per quanto riguarda ciò che possono calcolare. Negli anni '40 e '50 diversi ricercatori studiarono alcuni tipi più semplici di macchine, che oggi sono dette automi a stati finiti. Questi automi, pensati originariamente per fornire un modello del funzionamento cerebrale, risultarono utili per molti altri scopi. Nello stesso periodo, nei tardi anni 60, il linguista N. Chomsky iniziò a studiare le grammatiche formali. Per quanto non siano delle macchine in senso proprio, queste grammatiche sono strettamente collegate agli automi astratti e oggi stanno alla base di alcuni importanti componenti software.

Tutti questi sviluppi teorici hanno un rapporto diretto con quanto gli informatici fanno attualmente. Alcuni concetti, come gli automi a stati finiti e certi tipi di grammatiche formali, vengono usati nella progettazione e realizzazione di importanti tipi di software. Altri concetti, come la macchina di Turing, aiutano a comprendere che cosa ci si può aspettare dal software.

### Introduzione alle dimostrazioni formali

Alcuni studiosi di informatica arrivano a dire che una dimostrazione formale della correttezza di un programma e la scrittura del programma stesso dovrebbero andare di pari passo. Dubitiamo che tale procedimento sia produttivo. D'altro canto c'è chi non lascia alcuno spazio alle dimostrazioni nella disciplina della programmazione, affidandosi spesso allo slogan "se non sei sicuro che il tuo programma sia corretto, eseguilo e vedi".

Per scrivere un'iterazione o una ricorsione corretta bisogna fondarsi su un'ipotesi induttiva. È utile valutare, formalmente o informalmente, che l'ipotesi sia coerente con l'iterazione o la ricorsione. Questo processo di comprensione dei meccanismi di un programma corretto coincide in sostanza con il processo di dimostrazione di teoremi per induzione.

Esiste una forma speciale di dimostrazione, detta induttiva, che è essenziale quando si ha a che fare con oggetti definiti ricorsivamente. Molte delle dimostrazioni induttive più familiari si occupano di interi, ma nella teoria degli automi c'è bisogno anche di dimostrazioni induttive su concetti definiti ricorsivamente come alberi ed espressioni di varia natura, per esempio le espressioni regolari.

### Induzioni sugli interi

Supponiamo di dover dimostrare un enunciato  $S(n)$  su un intero  $n$ . Una soluzione comune consiste nel dimostrare due cose.

1. **La base**, in cui si dimostra  $S(i)$  per un particolare intero  $i$ . Di solito  $i = 0$  oppure  $i = 1$ , ma ci sono esempi in cui si comincia da un  $i$  più alto, magari perché l'enunciato  $S$  è falso per alcuni interi piccoli.
2. **Il passo induttivo**, in cui supponiamo che  $n \geq i$ , dove  $i$  è l'intero di base, e si dimostra che "se  $S(n)$  è vera, allora anche  $S(n + 1)$  lo è".

Intuitivamente queste due parti dovrebbero convincerci che  $S(n)$  è vero per ogni intero  $n$  che sia uguale o maggiore dell'intero di base  $i$ . Il ragionamento è il seguente. Supponiamo che  $S(n)$  sia falso per uno o più di quei numeri interi. Allora dovrebbe esistere un minimo valore di  $n$ , poniamo  $j$ , per il quale  $S(j)$  è falso e tuttavia  $j > i$ . Ma  $j$  non può essere  $i$  perché abbiamo dimostrato (base) che  $S(i)$  è vero; perciò  $j$  deve essere maggiore di  $i$ . Ora sappiamo che  $j - 1 \geq i$  e  $S(j - 1)$  è vero.

## Teoria della Computazione 1

Caputo L. - Costante L. – Davino C. - Di Giacomo I. – Giordano A. - Palo U. – Vitale P.

Proseguendo, nel passo induttivo abbiamo dimostrato che se  $n \geq i$  allora  $S(n)$  implica  $S(n+1)$ . Supponiamo che sia  $n = j - 1$ . Allora sappiamo dal passo induttivo che  $S(j-1)$  implica  $S(j)$ . Poiché sappiamo che  $S(j-1)$  è vero, possiamo concludere che lo è anche  $S(j)$ .

Abbiamo ipotizzato la negazione di quanto volevamo dimostrare; abbiamo cioè supposto che  $S(j)$  fosse falso per un qualche  $j \geq i$ . In ognuno dei casi abbiamo ricavato una contraddizione, così abbiamo ottenuto una dimostrazione per assurdo che  $S(n)$  è vero per ogni  $n \geq i$ .

Purtroppo c'è un sottile difetto logico in questo ragionamento. Il nostro assurdo, che si possa scegliere il minimo  $j \geq i$  per il quale  $S(j)$  è falso, già dipende dalla nostra fiducia nel principio di induzione. In altre parole l'unico modo di dimostrare che si può trovare questo  $j$  è dimostrarlo per meno di un metodo che è essenzialmente una dimostrazione induttiva. Tuttavia la "dimostrazione" discussa sopra è intuitivamente ragionevole e si accorda con la comune comprensione del mondo reale. Generalmente si prende dunque come parte integrante del nostro modo di ragionare il seguente principio.

- Principio di induzione: se dimostriamo  $S(i)$  e dimostriamo che, per ogni  $n \geq i$ ,  $S(n)$  implica  $S(n+1)$ . Allora possiamo concludere  $S(n)$  per ogni  $n > i$ .

## I numeri interi come concetti definiti ricorsivamente

Si è detto che le dimostrazioni induttive sono utili quando l'oggetto su cui si ragiona è definito ricorsivamente. Tuttavia i primi esempi trattati sono induzioni su interi, che normalmente non si considerano definiti ricorsivamente. Ciononostante, esiste una definizione naturale e ricorsiva di numero intero non negativo, e questa definizione si accorda con la maniera in cui si svolgono le induzioni sugli interi: dagli oggetti definiti prima a quelli definiti in seguito.

*BASE: 0 è un intero.*

*INDUZIONE Se  $n$  è un intero, lo è anche  $n + 1$ .*

## Forme più generali di induzione sugli interi

Esistono due importanti generalizzazioni di questo schema.

1. Possiamo usare diversi casi di base. In altre parole dimostriamo, per un  $j > i$ , gli enunciati  $S(i)$ ,  $S(i+1), \dots, S(j)$
2. Nella dimostrazione di  $S(n+1)$  possiamo usare la validità di tutti gli enunciati  $S(i), S(i+1), \dots, S(n)$  piuttosto che usare semplicemente  $S(n)$ . Inoltre, se abbiamo dimostrato i casi di base fino a  $S(j)$ , possiamo supporne che  $n \geq j$  anziché solo  $n \geq i$ .

La conclusione che si può ricavare da questa base e dal passo induttivo è che  $S(n)$  è vero per tutti i valori  $n \geq i$ .

## Definizione ricorsiva di espressioni aritmetiche

1. Un numero reale o una variabile sono espressioni aritmetiche
2. Se  $E_1, E_2$  sono espressioni aritmetiche, allora anche

$E_1+E_2$

$E_1 \cdot E_2$

$E_1-E_2$

$E_1/E_2$

sono espressioni aritmetiche.

## Induzioni strutturali

Nella teoria degli automi esistono diverse strutture definite ricorsivamente su cui si devono dimostrare enunciati. Importanti esempi sono le nozioni familiari di alberi e di espressioni. Similmente alle induzioni, tutte le definizioni ricorsive hanno un caso di base, in cui si definiscono una o più strutture elementari, e un passo induttivo, in cui si definiscono strutture più complesse nei termini di strutture definite in precedenza.

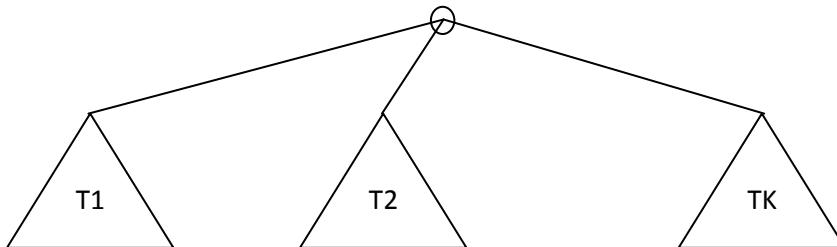
**Esempio** Una definizione ricorsiva di albero.

**Base:** Un singolo nodo è un albero, e tale nodo è la radice dell'albero.

**INDUZIONE:** Se  $T_1, T_2, \dots, T_k$ , sono alberi, allora si può formare un nuovo albero in questo modo.

1. si comincia con un nuovo nodo  $N$ , che è la radice dell'albero
2. si aggiunge una copia degli alberi  $T_1, T_2, \dots, T_k$ ,
3. si aggiungono lati dal nodo  $N$  alle radici di ogni albero  $T_1, T_2, \dots, T_k$ .

La Figura mostra la costruzione induttiva di un albero con radice  $N$  a partire da  $k$  alberi più piccoli.



**Teorema** Ogni albero ha un nodo in più rispetto ai suoi lati.

**DIMOSTRAZIONE:** L'enunciato formale  $S(T)$  che dobbiamo dimostrare per induzione strutturale è: "se  $T$  è un albero e  $T$  ha  $n$  nodi ed  $e$  lati, allora  $n = e + 1$ ".

**BASE:** il caso di base si ha quando  $T$  è un nodo singolo. Allora  $n = 1$  ed  $e = 0$ . Dunque la relazione  $n = e + 1$  è valida.

**INDUZIONE.** Sia  $T$  un albero costituito secondo il passo induttivo della definizione, dal nodo radice  $N$  e da  $k$  alberi più piccoli  $T_1, T_2, \dots, T_k$ . Possiamo supporre che gli enunciati  $S(T_i)$  siano validi per  $i = 1, 2, \dots, k$ . In altre parole, supponendo che  $T_1$  abbia  $n_1$  nodi ed  $e_1$  lati, si ha  $n_1 = e_1 + 1$ .

I nodi di  $T$  sono il nodo  $N$  e tutti i nodi degli alberi  $T_i$ . Esistono dunque  $1 + n_1 + n_2 + \dots + n_k$  nodi in  $T$ . I lati di  $T$  sono i  $k$  lati che sono stati aggiunti esplicitamente nel passo di induzione, più i lati di tutti i  $T_i$ . Perciò  $T$  ha

$$k + e_1 + e_2 + \dots + e_k$$

lati. Se si sostituisce  $e_i + 1$  a  $n_i$  nel conteggio del numero di nodi di  $T$ , si trova che  $T$  ha

$$1 + [e_1 + 1] + [e_2 + 1] + \dots + [e_k + 1]$$

nodi. Poiché ci sono  $k$  termini "+1" possiamo riscrivere:

$$k + 1 + e_1 + e_2 + \dots + e_k$$

Quest'espressione vale esattamente 1 più dell'espressione  $k + e_1 + e_2 + \dots + e_k$  che era stata data per il numero di lati di  $T$ . Perciò  $T$  ha un nodo in più del numero dei suoi lati.

## Induzione mutua

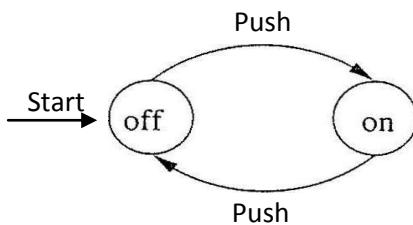
Talvolta non è possibile dimostrare un singolo enunciato per induzione, ma si deve dimostrare congiuntamente un gruppo di enunciati  $S_1(n), S_2(n), \dots, S_k(n)$  per induzione su  $n$ . La teoria degli automi fornisce molte situazioni di questo genere. Nell'Esempio successivo illustriamo un tipico caso in cui si deve spiegare che cosa fa un automa dimostrando un gruppo di enunciati, uno per ogni stato. Tali enunciati indicano per quali sequenze di input l'automa passa in ognuno degli stati.

A rigor di termini, dimostrare un gruppo di enunciati non differisce dal dimostrare la congiunzione (AND logico) di tutti gli enunciati. Per esempio il gruppo di enunciati  $S_1(n), S_2(n), \dots, S_k(n)$  può essere sostituito dal singolo enunciato

$$S_1(n) \text{ AND } S_2(n) \text{ AND } \dots \text{ AND } S_k(n)$$

Tuttavia, quando bisogna dimostrare molti enunciati effettivamente indipendenti, di solito conviene tenere separati gli enunciati e dimostrare per ciascuno la rispettiva base e il passo induttivo. Questo tipo di dimostrazione è detto induzione mutua.

**Esempio:** Consideriamo l'interruttore on/off, presentato come automa.



Dato che la pressione del pulsante cambia lo stato tra on e off, e l'interruttore parte dallo stato off, ci si aspetta che i seguenti enunciati descrivano insieme il funzionamento dell'interruttore.

*S1 (n): l'automa si trova nello stato off dopo n pressioni se e solo se n è pari.*

*S2 (n): l'automa si trova nello stato on dopo n pressioni se e solo se n è dispari*

Sapendo che un numero  $n$  non può essere allo stesso tempo pari e dispari, si potrebbe supporre che S1 implichi S2, e viceversa. Tuttavia ciò che non è sempre vero di un automa è che si trovi in un solo stato. Di fatto l'automa della figura è sempre esattamente in un solo stato, ma questo deve essere dimostrato come parte dell'induzione mutua.

Diamo la base e l'induzione delle dimostrazioni degli enunciati S1(n) e S2(n). Le dimostrazioni dipendono da diverse proprietà degli interi pari e dispari: se si aggiunge o toglie 1 da un intero pari si ottiene un intero dispari, e se si aggiunge o toglie 1 da un intero dispari si ottiene un intero pari.

**BASE:** Per la base scegliamo  $n = 0$ . Dato che ci sono due enunciati, ognuno dei quali deve essere dimostrato in entrambe le direzioni (in quanto S1 ed S2 sono ciascuno enunciati "se-e-solo-se"), in effetti ci sono quattro casi per la base e altrettanti per l'induzione.

1. [S1; se] Dato che 0 è pari, dobbiamo dimostrare che dopo 0 pressioni l'automa della si trova nello stato off. Poiché questo è lo stato iniziale, l'automa si trova effettivamente nello stato off dopo 0 pressioni.
2. [S1; solo-se] L'automa si trova nello stato off dopo 0 pressioni, perciò dobbiamo mostrare che 0 è pari. Ma 0 è pari per la definizione di "pari", dunque non resta altro da dimostrare.
3. [S2; se] L'ipotesi della parte "se" di S2 è che 0 sia dispari. Essendo quest'ipotesi H falsa, qualsiasi enunciato della forma "se H allora C" è vero. Di conseguenza è valida anche questa parte della base.
4. [S2; solo-se] Anche l'ipotesi che l'automa sia nello stato on dopo 0 pressioni è falsa, in quanto l'unico modo di pervenire allo stato on è seguire un arco etichettato Push, il che richiede almeno una pressione sul pulsante. Poiché l'ipotesi è falsa, possiamo concludere nuovamente che l'enunciato se-allora è vero.

**INDUZIONE:** Ora supponiamo che S1(n) e S2 (n) siano vere, e proviamo a dimostrare S1 (n + 1) e S2(n + 1). Anche questa dimostrazione si suddivide in quattro parti.

- ✓ [S1 (n + 1); se] L'ipotesi per questa parte è che  $n + 1$  sia pari. Di conseguenza  $n$  è dispari. La parte "se" dell'enunciato S2(n) dice che dopo  $n$  pressioni l'automa si trova nello stato on. L'arco da on a off recante l'etichetta Push dice che la  $(n + 1)$ -esima pressione farà passare l'automa nello stato off. Ciò completa la dimostrazione della parte "se" di S1 (n + 1).
- ✓ [S1 (n + 1); solo-se] L'ipotesi è che l'automa si trovi nello stato off dopo  $n + 1$  pressioni. L'esame dell'automa indica che l'unico modo di pervenire allo stato off è di trovarsi nello stato on e di ricevere un input Push. Perciò, se ci si trova nello stato off dopo  $n + 1$  pressioni, l'automa deve essersi trovato nello stato on dopo  $n$  pressioni. Allora possiamo ricorrere alla parte "solo-se"

dell'enunciato S2 (n) per concludere che n è dispari. Dunque n + 1 è pari, come si voleva dimostrare per la parte solo-se di S1 (n + 1).

- ✓ [S2(n + 1); se] L'ipotesi per questa parte è che n + 1 sia dispari. Di conseguenza n è pari. La parte "se" dell'enunciato S1(n) dice che dopo n pressioni l'automa si trova nello stato off. L'arco da off a on recante l'etichetta Push dice che la (n + 1)-esima pressione farà passare l'automa nello stato on. Ciò completa la dimostrazione della parte "se" di S2 (n + 1).
- ✓ [S2(n + 1); solo-se] L'ipotesi è che l'automa si trovi nello stato on dopo n + 1 pressioni. L'esame dell'automa indica che l'unico modo di pervenire allo stato on è di trovarsi nello stato off e di ricevere un input Push. Perciò, se ci si trova nello stato on dopo n + 1 pressioni, l'automa deve essersi trovato nello stato off dopo n pressioni. Allora possiamo ricorrere alla parte "solo-se" dell'enunciato S1 (n) per concludere che n è pari. Dunque n + 1 è dispari, come si voleva dimostrare per la parte solo-se di S2 (n + 1).

Dall'esempio si può ricavare il modello di tutte le induzioni mutue.

- Ogni enunciato dev'essere dimostrato separatamente nella base e nel passo induttivo.
- Se si tratta di enunciati se-e-solo-se, allora entrambe le direzioni di ogni enunciato devono essere dimostrate, sia nella base sia nel passo induttivo.

## I Quantificatori

In molti teoremi compaiono enunciati che usano i quantificatori "per ogni" ed "esiste", o varianti simili, come "per qualunque" invece di "per ogni". L'ordine in cui compaiono influisce sul significato dell'enunciato. Spesso è utile vedere gli enunciati con più di un quantificatore come una "partita" tra due giocatori, "per ogni" ed "esiste", che a turno specificano i valori dei parametri menzionati nel teorema. Poiché "per ogni" deve considerare tutte le scelte possibili, generalmente le sue scelte sono lasciate come variabili; "esiste", invece, deve selezionare un valore, che può dipendere dai valori scelti in precedenza. L'ordine dei quantificatori nell'enunciato determina chi parte per primo. Se l'ultimo giocatore che compie una scelta può trovare sempre un valore accettabile, l'enunciato è vero.

Consideriamo per esempio una definizione alternativa di insieme infinito: l'insieme S è infinito se e solo se, per ogni intero n, esiste almeno un sottoinsieme T di S con n elementi. Qui "per ogni" precede "esiste", dunque dobbiamo considerare un intero arbitrario Ts. Ora "esiste" seleziona un sottoinsieme T servendosi della sua conoscenza di n. Per esempio, se S fosse l'insieme degli interi, "esiste" potrebbe scegliere il sottoinsieme T = {1, 2, ..., n}. Questa è una dimostrazione che l'insieme degli interi è infinito.

L'enunciato seguente sembra simile alla definizione di infinito, ma è scorretto perché rovescia l'ordine dei quantificatori: "esiste un sottoinsieme T dell'insieme S tale che, per qualunque n, l'insieme T ha esattamente n elementi". Ora, dato un insieme S come gli interi, il giocatore "esiste" può selezionare qualsiasi insieme T. Poniamo che venga scelto {1, 2, 5}. Per questa scelta il giocatore "per ogni" deve mostrare che T ha n elementi per ogni possibile n. Tuttavia "per ogni" non può farlo. Per esempio l'affermazione è falsa per n = 4 e, di fatto, per ogni n ≠ 3.

## I concetti centrali della teoria degli automi

In questo paragrafo verranno introdotte le definizioni dei termini più importanti che permeano la teoria degli automi. Questi concetti sono "alfabeto" (un insieme di simboli), "stringa" (una lista di simboli di un alfabeto) e "linguaggio" (un insieme di stringhe dallo stesso alfabeto).

**Nozione di non determinismo:** la macchina può scegliere tra diverse opzioni.

**Nozione di codifica:** una struttura X definita in modo ricorsivo (Es. alberi, espressioni booleane, espressioni aritmetiche, espressioni regolari).

**Alfabeti:** un alfabeto è un insieme finito e non vuoto di simboli. Per indicare un alfabeto si usa convenzionalmente il simbolo  $\Sigma$ . Fra gli alfabeti più comuni citiamo:

1.  $\Sigma = \{0,1\}$ , l'alfabeto binario
2.  $\Sigma = \{a, b, c, \dots, z\}$ , l'insieme di tutte le lettere minuscole
3. l'insieme di tutti i caratteri ASCII o l'insieme di tutti i caratteri ASCII stampabili.

**Stringhe:** una stringa (o parola) è una sequenza finita di simboli scelti da un alfabeto. Per esempio 01101 è una stringa sull'alfabeto binario  $\Sigma = \{0,1\}$ . La stringa 111 è un'altra stringa sullo stesso alfabeto.

**La stringa vuota:** la stringa vuota è la stringa composta da zero simboli. Questa stringa, indicata con  $\epsilon$ , è una stringa che può essere scelta da un qualunque alfabeto.

**Insieme delle parole:** su  $\Sigma = \bigcup_{n \geq 0} \Sigma^n = \Sigma^*$

**Lunghezza di una stringa:** spesso è utile classificare le stringhe a seconda della loro lunghezza, vale a dire il numero di posizioni per i simboli della stringa. Per esempio 01101 ha lunghezza 5. Comunemente si dice che la lunghezza di una stringa è "il numero di simboli" nella stringa stessa; quest'enunciato è un'espressione accettabile colloquialmente, ma formalmente non corretta. Infatti ci sono solo due simboli, 0 e 1, nella stringa 01101, ma ci sono 5 posizioni, e la lunghezza della stringa è 5. Tuttavia ci si può aspettare di vedere usato di solito "il numero di simboli" quando invece s'intende il "numero di posizioni".

La notazione standard per la lunghezza di una stringa  $w$  è  $|w|$ . Per esempio  $|011| = 3$  e  $|\epsilon| = 0$ .

(BASE):  $|\epsilon| = 0$

(INDUZIONE): Se  $w \neq \epsilon$  allora  $w = xa$ ,  $x \in \Sigma^*$ ,  $a \in \Sigma$  e  $|w| = |x| + 1$

### Potenze di un alfabeto

Se  $\Sigma$  è un alfabeto possiamo esprimere l'insieme di tutte le stringhe di una certa lunghezza su tale alfabeto usando una notazione esponenziale. Definiamo  $\Sigma^k$  come l'insieme di stringhe di lunghezza  $k$ , con simboli tratti da  $\Sigma$ .

**Esempio:** Si noti che  $\Sigma^0 = \{\epsilon\}$  per qualsiasi alfabeto  $\Sigma$ . In altre parole  $\epsilon$  è la sola stringa di lunghezza 0.

Se  $\Sigma = \{0,1\}$ , allora  $\Sigma^1 = \{0,1\}$ ,  $\Sigma^2 = \{00, 01, 10, 11\}$ .

(BASE):  $\Sigma^0 = \{\epsilon\}$

(INDUZIONE):  $\forall n > 0, \Sigma^n = \{xy \mid x \in \Sigma^{n-1}, y \in \Sigma\}$

### Concatenazione di stringhe

Siano  $x$  e  $y$  stringhe. Allora  $xy$  denota la concatenazione di  $x$  e  $y$ , vale a dire la stringa formata facendo una copia di  $x$  e facendola seguire da una copia di  $y$ . Più precisamente, se  $x$  è la stringa composta da  $i$  simboli,  $x = a_1 a_2 \dots a_i$  e  $y$  è la stringa composta da  $j$  simboli  $y = b_1 b_2 \dots b_j$ , allora  $xy$  è la stringa di lunghezza  $i+j$ :

$$xy = a_1 a_2 \dots a_i b_1 b_2 \dots b_j.$$

La concatenazione è associativa  $(xy)z = x(yz)$ . Invece non è commutativa:

Es. capo, stazione  $\rightarrow$  capostazione

stazione, capo  $\rightarrow$  stazionecapo

Esempio Poniamo  $x = 01101$  e  $y = 110$ . Allora  $xy = 01101110$  e  $yx = 11001101$ . Per qualunque stringa  $w$  sono valide le equazioni  $\epsilon w = w \epsilon = w$ . In altre parole  $\epsilon$  è l'identità per la concatenazione, dato che, concatenata con una qualunque stringa, dà come risultato la stringa stessa (nello stesso modo in cui 0, l'identità per l'addizione, può essere sommato a qualunque numero  $z$  e dà come risultato  $x$ ).

### Prefisso/suffisso di una parola

Sia  $w = abaa$

Suffisso:  $\epsilon$ ,  $a$ ,  $ab$ ,  $aba$ ,  $abaa$

Prefisso:  $\epsilon$ ,  $a$ ,  $aa$ ,  $baa$ ,  $abaa$ .

## Linguaggi

Un insieme di stringhe scelte da  $\Sigma^*$ , dove  $\Sigma$  è un particolare alfabeto, si dice un linguaggio. Se  $\Sigma$  è un alfabeto e  $L \subseteq \Sigma^*$ , allora  $L$  è un linguaggio su  $\Sigma$ . Si noti che un linguaggio su  $\Sigma$  non deve necessariamente includere stringhe con tutti i simboli di  $\Sigma$ ; perciò, una volta che abbiamo stabilito che  $L$  è un linguaggio su  $\Sigma$ , sappiamo anche che è un linguaggio su qualunque alfabeto include  $\Sigma$ .

La scelta del termine "linguaggio" può sembrare strana; d'altra parte i linguaggi comuni possono essere visti come insiemi di stringhe. Un esempio è l'inglese, in cui la raccolta delle parole accettabili della lingua è un insieme di stringhe sull'alfabeto che consiste di tutte le lettere. Un altro esempio è il C, o qualunque altro linguaggio di programmazione, in cui i programmi accettabili sono un sottoinsieme di tutte le possibili stringhe che si possono formare con l'alfabeto del linguaggio. Quest'alfabeto è un sottoinsieme dei caratteri ASCII. L'alfabeto può variare leggermente tra diversi linguaggi di programmazione, ma generalmente include le lettere maiuscole e minuscole, le cifre, la punteggiatura e i simboli matematici.

Nello studio degli automi ci si imbatte in molti altri linguaggi. Alcuni sono esempi astratti, come quelli che elenchiamo.

1. Il linguaggio di tutte le stringhe che consistono di  $n$  0 seguiti da  $n$  1. per  $n \geq 0$ : {e. 01, 0011, 000111,...}.
2. L'insieme delle stringhe con un uguale numero di 0 e di 1: {e, 01, 10, 0011, 0101, 1001, ...}
3. L'insieme dei numeri binari il cui valore è un numero primo: {10, 11, 101, 111, 1011, ...}
4.  $\Sigma^*$ , è un linguaggio per qualunque alfabeto  $\Sigma$ .
5.  $\emptyset$ , il linguaggio vuoto, è un linguaggio rispetto a qualunque alfabeto.
6. Anche  $\{\varepsilon\}$ , il linguaggio che consta della sola stringa vuota, è un linguaggio rispetto a ogni alfabeto.

Si noti che  $\emptyset \neq \{\varepsilon\}$ ; il primo non ha stringhe mentre il secondo ne ha una.

L'unica restrizione di rilievo sui linguaggi è che tutti gli alfabeti sono finiti. Di conseguenza i linguaggi, sebbene possano avere un numero infinito di stringhe, devono consistere di stringhe tratte da un determinato alfabeto finito.

## ESERCIZI CAPITOLO 1

### Quantificatori

#### Esiste

**Esercizio:**  $L = \{w \in \{0,1\}^* \mid \exists x, y, z \in \{0,1\}^* \text{ con } |y| = 2 \text{ t.c. } w \neq xyz \text{ e } y = 00\}$

$\exists x, y \geq 0 \text{ t.c. } x101y = \{w \in \{0,1\}^* \mid w = x00y\} \text{ oppure } \{w \in \{0,1\}^* \mid w = x00y, x, y \in \{0,1\}^*\}$

#### Per ogni

**Esercizio:**  $L' = \{w \in \{0,1\}^* \mid \forall x, y, z \in \{0,1\}^* \text{ con } |y| = 2 \text{ se } w = xyz \text{ allora } y = 00\}$

**Soluzione:**  $= \{0\}^* \cup \{1\}$

### Parole

**Esercizio:** Descrivere la generica parola su  $0^n 1^n \mid n \geq 0 = 0^{n_1} 1^{m_1} 0^{n_2} 1^{m_2} \dots 1^{m_{k-1}} 0^{n_k} 1^{m_k}$

### Operazioni insiemistiche

**Esercizio:**  $\{0,1\}^* \{0^n 1^n \mid n \geq 0\} = \{0^k 1^h \mid h \neq k \text{ e } k, h \geq 0\} \cup \{w \in \{0,1\}^* \mid \exists x, y \in \{0,1\}^* \text{ co } w = x10y\}$

**Esercizio:** Sia:

$$L_1 = \{w \in \{a, b, c\}^* \mid |x|_a = |x|_b = |x|_c \geq 0\}$$

$$L_2 = \{w \in \{a, b, c\}^* \mid |x|_a = |x|_b \geq 0 \text{ e } |x|_c = 1\}$$

$$L_1 = \{\varepsilon, abc, aabbcc, aaabbbccc, aaaabbbbcccc, \dots\}$$

$$L_2 = \{c, abc, aabbc, aaabbcc, aaaabbbbc, \dots\}$$

Chi è:

$$L_1 \setminus L_2 = \{x \in \{a, b, c\}^* \mid |x|_a = |x|_b = |x|_c \geq 2\} \cup \{\varepsilon\}$$

$$L_2 \setminus L_1 = \{x \in \{a, b, c\}^* \mid |x|_a = |x|_b \neq |x|_c = 1\}$$

**Esercizio:**  $\{a, b\}^* \setminus \{a, b\}^2 = \{\varepsilon\} \cup \{a, b\} \cup \bigcup_{n \geq 3} \{a, b\}^n$

## CAPITOLO 2 – AUTOMI A STATI FINITI

### Automi a stati finiti deterministici

Un automa a stati finiti deterministico A (spesso indicato con il suo acronimo DFA) è una quintupla:

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

Dove:

$Q = E'$  un insieme finito di stati

$\Sigma = E'$  un alfabeto (insieme finito di simboli, non vuoto)

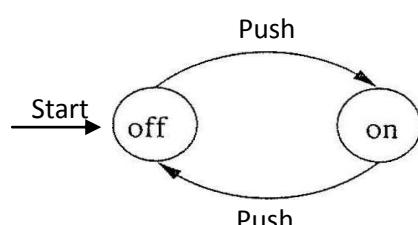
$q_0 = E'$  lo stato iniziale

$F = E'$  Un insieme di stati finali, o accettanti. L'insieme  $F$  è un sottoinsieme di  $Q$ .

$\delta = E'$  una funzione di transizione, che prende come argomento uno stato e un simbolo di input e restituisce uno stato. Questa funzione ha dominio  $Q \times \Sigma$  e codominio  $Q$  da cui  $\delta: Q \times \Sigma \rightarrow Q$ .

Poiché viene utilizzata una funzione (di transizione) ciò vuol dire che il nostro automa deve avere un comportamento deterministico.

Es.



$$\delta: \{\text{OFF}, \text{ON}\} \times \{\text{PUSH}\} \rightarrow \{\text{OFF}, \text{ON}\}$$

Es: ( $\text{Push}$ ,  $\text{Push}$ ,  $\text{Push}$ ) è intuitivamente evidente:  $\text{off} \rightarrow \text{on} \rightarrow \text{off} \rightarrow \text{on}$

**In generale:**

stato iniziale  $q$  simboli dell'alfabeto  $a_1, a_2, \dots, a_n$

$$\delta(q, a_1) = q_1$$

$$\delta(q_1, a_2) = q_2$$

... ...

$$\delta(q_{n-1}, a_n) = q_n$$

### Rappresentazioni degli automi

È possibile descrivere gli automi attraverso 3 rappresentazioni

1. Automa come quintupla  $A = (Q, \Sigma, \delta, q_0, F)$

Es. dell'automa precedente:

$$Q = \{q_0, q_1, q_2\} \quad \Sigma = \{0, 1\} \quad \delta: \{q_0, q_1, q_2\} \times \{0, 1\} \rightarrow \{q_0, q_1, q_2\} \quad q_0 = \text{stato iniziale} \quad F = \{q_2\}$$

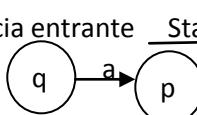
$$\delta(q_0, 1) = q_0 \quad \delta(q_0, 0) = q_1 \quad \delta(q_1, 0) = q_1 \quad \delta(q_1, 1) = q_2 \quad \delta(q_2, 0) = \delta(q_2, 1) = q_2$$

2. Grafo delle transizioni

$Q$ =Insieme dei vertici, lo stato iniziale è indicato da una freccia entrante e lo stato finale da un doppio cerchio , inoltre  $\delta(q, a) = p \Leftrightarrow$

3. Tabella delle transizioni:

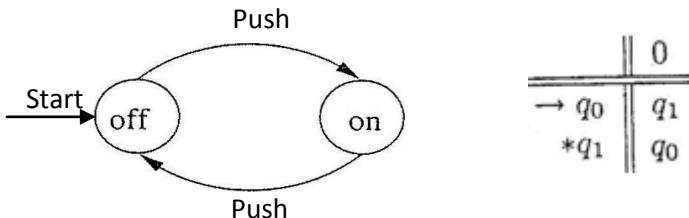
E' una tabella la cui intestazione delle righe è data dagli elementi di  $Q$ , quella delle colonne da  $\Sigma$ .



Gli stati finali sono preceduti da \* e lo stato iniziale da  $\rightarrow$ .

	$\Sigma$									
	a	b	c	...	x	...	z			
Stati	$q_0$			...						
	$q_1$			...						
	$q_2$			...						
..	..	..	..	..						
$q_i$	...	...	...	...	$q_j$			$\delta(q_i, x) = q_j$		
..										
$q_n$										

Es.



## Automi a stati finiti deterministici (DFA)

Cominciamo dalla definizione di automa a stati finiti deterministico, un automa che dopo aver letto una qualunque sequenza di input si trova in un singolo stato. Il termine "deterministico" concerne il fatto che per ogni input esiste un solo stato verso il quale l'automa passa dal suo stato corrente.

### Definizione di automa a stati finiti deterministico

Un automa a stati finiti deterministico consiste dei seguenti componenti.

1. Un insieme finito di stati, spesso indicato con  $Q$ .
2. Un insieme finito di simboli di input, spesso indicato con  $\Sigma$ .
3. Una funzione di transizione, che prende come argomento uno stato e un simbolo di input e restituisce uno stato. La funzione di transizione sarà indicata comunemente con  $\delta$ . Nella rappresentazione grafica informale di automi che abbiamo visto,  $\delta$  è rappresentata dagli archi tra gli stati e dalle etichette sugli archi. Se "q" è uno stato e "a" è un simbolo di input,  $\delta(q, a)$  è lo stato "p" tale che esiste un arco etichettato con "a" da "q" a "p".
4. Uno stato iniziale, uno degli stati in  $Q$ .
  - Più precisamente, il grafo è la descrizione di una funzione di transizione  $\delta$ , e gli archi del grafo sono costruiti per riflettere le transizioni specificate da  $\delta$ .
5. Un insieme di stati finali, o accettanti,  $F$ . L'insieme  $F$  è un sottoinsieme di  $Q$ .

Un automa a stati finiti deterministico verrà spesso indicato con il suo acronimo DFA. La rappresentazione più concisa di un DFA è un'enumerazione dei suoi cinque componenti. Nelle dimostrazioni denotiamo un DFA come una quintupla:

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

dove  $A$  è il nome del DFA,  $Q$  è l'insieme degli stati,  $\Sigma$  i suoi simboli di input,  $\delta$  la sua funzione di transizione,  $q_0$  il suo stato iniziale ed  $F$  il suo insieme di stati accettanti.

### Elaborazione di stringhe in un DFA

La prima cosa che bisogna capire di un DFA è come decide se "accettare" o no una sequenza di simboli di input. Il "linguaggio" del DFA è l'insieme di tutte le stringhe che il DFA accetta. Supponiamo che  $a_1, a_2, \dots, a_n$  sia una sequenza di simboli di input. Si parte dal DFA nel suo stato iniziale,  $q_0$ . Consultando la funzione di transizione  $\delta$ , per esempio  $\delta(q_0, a_1) = q_1$ , troviamo lo stato in cui il DFA entra dopo aver letto il primo simbolo di input  $a_1$ . Il successivo simbolo di input,  $a_2$ , viene trattato valutando  $\delta(q_1, a_2)$ . Supponiamo che questo stato sia  $q_2$ . Continuiamo così, trovando gli stati  $q_3, q_4, \dots, q_n$ , tali che  $\delta(q_{i-1}, a_i) = q_i$  per ogni  $i$ . Se  $q_n$  è un elemento di  $F$ , allora l'input  $a_1, a_2, \dots, a_n$  viene accettato, altrimenti viene rifiutato.

#### Esempio 2.1.

Specifichiamo formalmente un DFA che accetta tutte e sole le stringhe di 0 e di 1 in cui compare la sequenza 01. Possiamo scrivere il linguaggio  $L$  così:

$$\{w \mid w \text{ è della forma } x01y \text{ per stringhe } x \text{ e } y \text{ che consistono solamente di 0 e di 1}\}$$

Una descrizione equivalente, che usa i parametri  $x$  e  $y$  a sinistra della barra verticale, è:

$$\{x01y \mid x \text{ e } y \text{ sono stringhe qualsiasi di 0 e di 1}\}$$

Esempi di stringhe appartenenti al linguaggio includono 01, 11010 e 100011. Esempi di stringhe non appartenenti al linguaggio includono  $\epsilon$ , 0, e 111000.

Che cosa sappiamo di un automa che accetta questo linguaggio  $L$ ? In primo luogo il suo alfabeto di input è  $\Sigma = \{0,1\}$ ; ha un certo insieme di stati,  $Q$ , di cui uno, poniamo  $q_0$ , è lo stato iniziale. Quest'automa deve ricordare i fatti importanti relativi agli input già visti. Per decidere se 01 è una sottostringa dell'input,  $A$  deve ricordare quanto segue.

1. Ha già visto 01? In caso affermativo accetta ogni sequenza di ulteriori input, cioè da questo momento in poi si troverà solo in stati accettanti.
2. Pur non avendo ancora visto 01, l'input più recente è stato 0, cosicché se ora vede un 1 avrà visto 01. Da questo momento può accettare qualunque seguito?
3. Non ha ancora visto 01, ma l'input più recente è nullo (siamo ancora all'inizio) oppure come ultimo dato ha visto un 1. In tal caso  $A$  non accetta finché non vede uno 0 e subito dopo un 1.

Ognuna di queste tre condizioni può essere rappresentata da uno stato. La condizione (3) è rappresentata dallo stato iniziale  $q_0$ . All'inizio bisogna vedere uno 0 e poi un 1. Ma se nello stato  $q_0$  si vede per primo un 1, allora non abbiamo fatto alcun passo verso 01, e dunque dobbiamo permanere nello stato  $q_0$ . In altri termini  $\delta(q_0, 1) = q_0$ . D'altra parte, se ci troviamo nello stato  $q_0$  e vediamo uno 0, siamo nella condizione (2). Vale a dire: non abbiamo ancora visto 01, ma ora abbiamo lo 0. Dunque usiamo  $q_2$  per rappresentare la condizione (2). La transizione da  $q_0$  sull'input 0 è  $\delta(q_0, 0) = q_2$ . Ora consideriamo le transizioni dallo stato  $q_2$ . Se vediamo uno 0, non siamo in una situazione migliore di prima, ma neanche peggiore. Non abbiamo visto 01, ma 0 è stato l'ultimo simbolo incontrato: stiamo ancora aspettando un 1. Lo stato  $q_2$  descrive questa situazione perfettamente, e dunque poniamo  $\delta(q_2, 0) = q_2$ . Se ci troviamo nello stato  $q_2$  e vediamo un input 1, sappiamo che c'è uno 0 seguito da un 1. Possiamo passare a uno stato accettante, che si chiamerà  $q_1$  e corrisponderà alla summenzionata condizione (1). Ossia  $\delta(q_2, 1) = q_1$ .

Infine dobbiamo determinare le transizioni per lo stato  $q_1$ . In questo stato abbiamo già incontrato una sequenza 01, quindi, qualsiasi cosa accada, saremo ancora in una situazione in cui abbiamo visto 01. In altri termini  $\delta(q_1, 0) = \delta(q_1, 1) = q_1$ . Da quanto detto risulta allora  $Q = \{q_0, q_1, q_2\}$ . Come affermato in

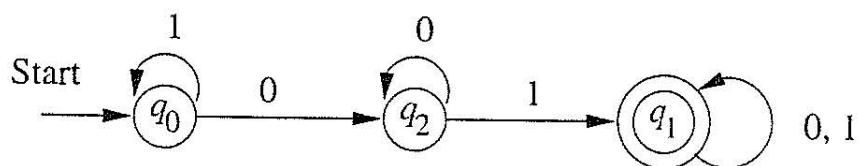
precedenza,  $q_0$  è lo stato iniziale e l'unico stato accettante è  $q_1$ . Quindi  $F = \{q_1\}$ . La definizione completa dell'automa A che accetta il linguaggio L delle stringhe che hanno una sottostringa 01 è

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$$

dove  $\delta$  è la funzione di transizione descritta sopra.

### Esempio 2.2

La Figura mostra il diagramma di transizione per il DFA costruito nell'esempio 2.1. Nel diagramma vediamo i tre nodi che corrispondono ai tre stati. C'è una freccia Start che entra nello stato iniziale  $q_0$ , e l'unico stato accettante,  $q_1$ , è rappresentato da un doppio circolo. Da ogni stato escono un arco etichettato 0 e un arco etichettato 1 (sebbene i due archi siano combinati in un unico arco con una doppia etichetta nel caso di  $q_1$ ). Ogni arco corrisponde a uno dei fatti relativi a  $\delta$ , stabiliti nell'Esempio 2.1.



### Tabelle di transizione

Una tabella di transizione è una comune rappresentazione tabellare di una funzione come  $\delta$ , che ha due argomenti e restituisce un valore. Le righe della tabella corrispondono agli stati, le colonne agli input. La voce all'incrocio della riga corrispondente allo stato  $q$  e della colonna corrispondente all'input  $a$  è lo stato  $\delta(q, a)$ .

### Esempio 2.3

La tabella di transizione relativa alla funzione  $\delta$  dell'Esempio 2.1 è rappresentata dalla Figura, corredata di due informazioni specifiche: lo stato iniziale è indicato da una freccia, gli stati accettanti da un asterisco. Dato che possiamo dedurre gli insiemi degli stati e dei simboli di input dalle intestazioni delle righe e delle colonne, dalla tabella di transizione si ricavano tutte le informazioni necessarie per specificare l'automa a stati finiti in maniera univoca

	0	1
$\rightarrow q_0$	$q_2$	$q_0$
$*q_1$	$q_1$	$q_1$
$q_2$	$q_2$	$q_1$

### Estensione della funzione di transizione alle stringhe

Abbiamo spiegato in modo informale che un DFA definisce un linguaggio: l'insieme di tutte le stringhe che producono una sequenza di transizioni di stati dallo stato iniziale a uno stato accettante. Rispetto al diagramma di transizione, il linguaggio di un DFA è l'insieme delle etichette lungo i cammini che conducono dallo stato iniziale a un qualunque stato accettante.

**Definizione:** Dato un DFA  $A = (Q, \Sigma, q_0, \delta, F)$ , la funzione di transizione estesa  $\hat{\delta}$  è definita nel modo seguente:  $\hat{\delta}: Q \times \Sigma^* \rightarrow Q$ .

È necessario ora precisare la nozione di linguaggio di un DFA. A questo scopo definiamo una funzione di transizione estesa, che descrive che cosa succede quando partiamo da uno stato e seguiamo una sequenza

## Teoria della Computazione 1

Caputo L. - Costante L. – Davino C. - Di Giacomo I. – Giordano A. - Palo U. – Vitale P.

di input. Se  $\delta$  è la funzione di transizione, allora la funzione di transizione estesa costruita da  $\delta$  si chiamerà  $\widehat{\delta}$ . La funzione di transizione estesa è una funzione che prende uno stato  $q$  e una stringa  $w$  e restituisce uno stato  $p$ : lo stato che l'automa raggiunge quando parte nello stato  $q$  ed elabora la sequenza di input  $w$ . Definiamo  $\widehat{\delta}$  per induzione sulla lunghezza della stringa di input come segue.

**BASE**  $\widehat{\delta}(q, \varepsilon) = q$ . In altre parole, se ci troviamo nello stato  $q$  e non leggiamo alcun input, allora rimaniamo nello stato  $q$ .

**INDUZIONE** Supponiamo che  $w$  sia una stringa della forma  $xa$ , ossia “ $a$ ” è l'ultimo simbolo di  $w$  e  $x$  è la stringa che consiste di tutti i simboli eccetto l'ultimo. Per esempio  $w = 1101$  si scomponerebbe in  $x = 110$  e  $a = 1$ . Allora

$$\widehat{\delta}(q, w) = \delta(\widehat{\delta}(q, x), a) \quad (2.1)$$

La (2.1) può sembrare contorta, ma il concetto è semplice. Per computare  $\widehat{\delta}(q, w)$ , calcoliamo prima  $\widehat{\delta}(q, x)$ , lo stato in cui si trova l'automa dopo aver elaborato tutti i simboli di  $w$  eccetto l'ultimo. Supponiamo che questo stato sia  $p$ , ossia  $\widehat{\delta}(q, x) = p$ . Allora  $\widehat{\delta}(q, w)$  è quanto si ottiene compiendo una transizione dallo stato  $p$  sull'input  $a$ , l'ultimo simbolo di  $w$ . In altri termini  $\widehat{\delta}(q, x) = \delta(p, a)$ .

### Esempio 2.4

Costruiamo un DFA che accetti il linguaggio

$$L = \{w \mid w \text{ ha un numero pari di } 0 \text{ e un numero pari di } 1\}$$

Non dovrebbe sorprendere che il compito degli stati di questo DFA sia quello di contare il numero degli 0 e quello degli 1, ma di contarli modulo 2. In altre parole si usa lo stato per ricordare se il numero degli 0 e il numero degli 1 visti fino a quel momento sono pari o dispari. Quindi ci sono quattro stati, che possono essere interpretati come segue:

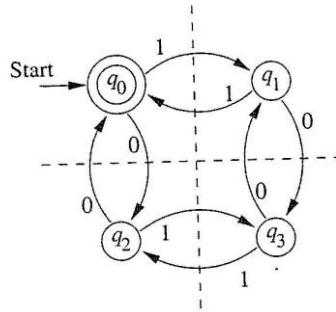
- $q_0$ : sia il numero degli 0 sia il numero degli 1 visti finora sono pari
- $q_1$ : il numero degli 0 visti finora è pari, ma il numero degli 1 è dispari
- $q_2$ : il numero degli 1 visti finora è pari, ma il numero degli 0 è dispari
- $q_3$ : sia il numero degli 0 sia il numero degli 1 visti finora sono dispari.

Lo stato  $q_0$  è nello stesso tempo lo stato iniziale e l'unico stato accettante. È lo stato iniziale perché prima di leggere qualunque input il numero degli 0 e degli 1 visti sono entrambi zero, e zero è pari. È l'unico stato accettante perché descrive esattamente la condizione per la quale una sequenza di 0 e di 1 appartiene al linguaggio  $L$ .

Siamo ora in grado di specificare un DFA per il linguaggio  $L$ :

$$A = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

La funzione di transizione  $\delta$  è descritta dal diagramma di transizione della Figura sotto. Si noti che ogni input 0 fa sì che lo stato attraversi la linea tratteggiata orizzontale. Pertanto, dopo aver visto un numero pari di 0 ci troviamo al di sopra della linea, nello stato  $q_0$  oppure  $q_1$ , mentre dopo averne visto un numero dispari ci troviamo al di sotto, nello stato  $q_2$  oppure  $q_3$ . Analogamente, ogni 1 fa sì che lo stato attraversi la linea tratteggiata verticale. Perciò, dopo aver visto un numero pari di 1 siamo a sinistra, nello stato  $q_0$  oppure  $q_2$ , mentre dopo averne visto un numero dispari siamo a destra, nello stato  $q_1$  o  $q_3$ .



È possibile rappresentare questo DFA anche per mezzo di una tabella di transizione, come illustrato nella Figura sotto. Noi però non siamo interessati solo all'ideazione di questo DFA; il nostro intento è di usarlo per illustrare la costruzione di  $\hat{\delta}$  dalla funzione di transizione  $\delta$ . Supponiamo che l'input sia 110101. Dato che questa stringa ha un numero pari sia di 0 sia di 1, ci aspettiamo che appartenga al linguaggio, cioè che  $\hat{\delta}(q_0, 110101) = q_0$ , dato che  $q_0$  è l'unico stato accettante. Verifichiamo quest'asserzione.

	0	1
*	$q_0$	$q_2$
$q_0$	$q_1$	$q_0$
$q_1$	$q_0$	$q_3$
$q_2$	$q_0$	$q_3$
$q_3$	$q_1$	$q_2$

La verifica comporta il calcolo di  $\hat{\delta}(q_0, w)$  per ogni prefisso  $w$  di 110101, a partire da  $\varepsilon$  e procedendo per aggiunte successive. Il riepilogo di questo calcolo è:

- $\hat{\delta}(q_0, \varepsilon) = q_0$
- $\hat{\delta}(q_0, 1) = \delta(\hat{\delta}(q_0, \varepsilon), 1) = \delta(q_0, 1) = q_1$
- $\hat{\delta}(q_0, 11) = \delta(\hat{\delta}(q_0, 1), 1) = \delta(q_1, 1) = q_0$
- $\hat{\delta}(q_0, 110) = \delta(\hat{\delta}(q_0, 11), 0) = \delta(q_0, 0) = q_2$
- $\hat{\delta}(q_0, 1101) = \delta(\hat{\delta}(q_0, 110), 1) = \delta(q_2, 1) = q_3$
- $\hat{\delta}(q_0, 11010) = \delta(\hat{\delta}(q_0, 1101), 0) = \delta(q_3, 0) = q_1$
- $\hat{\delta}(q_0, 110101) = \delta(\hat{\delta}(q_0, 11010), 1) = \delta(q_1, 1) = q_0$

## Il linguaggio di un DFA

Possiamo ora definire il linguaggio di un DFA  $A = (Q, \Sigma, q_0, \delta, F)$ . Questo linguaggio è indicato con  $L(A)$  ed è definito da

$$L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \text{ è in } F\}$$

In altre parole il linguaggio di  $A$  è l'insieme delle stringhe  $w$  che portano dallo stato iniziale  $q_0$  a uno degli stati accettanti. Se  $L$  è uguale a  $L(A)$  per un DFA  $A$ , allora diciamo che  $L$  è un linguaggio regolare.

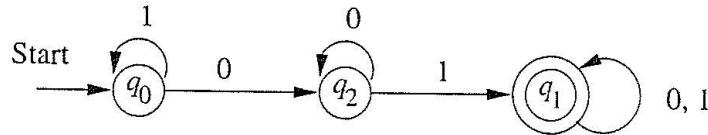
**Esempio 2.5** Come detto in precedenza, se  $A$  è il DFA dell'Esempio 2.1, allora  $L(A)$  è l'insieme di tutte le stringhe di 0 e di 1 che contengono la sottostringa 01. Se invece  $A$  è il DFA dell'Esempio 2.4, allora  $L(A)$  è l'insieme di tutte le stringhe di 0 e di 1 i cui numeri di 0 e di 1 sono entrambi pari.

**Definizione:** un linguaggio  $L \subseteq \Sigma^*$  si dice regolare se esiste un DFA  $A$  t.c.  $L$  è il linguaggio dell'automa, cioè  $L=L(A)$ .

Generalmente, ci sono 2 problemi:

- Problema 1 (ANALISI): Dato un DFA  $A$ , chi è  $L(A)$ ? ci si chiede "Il problema è decidibile?"
- Problema 2 (SINTESI): dato  $L \subseteq \Sigma^*$  esiste (prova INDECIDIBILE)/chi è  $A$  t.c.  $L=L(A)$ ?

NOTA: non per tutti i linguaggi esiste un automa. Se è regolare, allora non è indecidibile. Non tutti i linguaggi sono regolari.



$$L(A) = \{w \in \{0,1\}^* \mid \exists x, y \in \{0,1\}^*: w = x01y\}$$

$$w \in L(A) \Leftrightarrow \delta(q_0, w) = q_2 \Leftrightarrow \exists x, y \in \{0,1\}^*: w = x01y$$

## Automi a stati finiti non deterministici

Un automa a stati finiti non deterministico (NFA, Non-deterministic Finite Automaton) può trovarsi contemporaneamente in diversi stati. Questa caratteristica viene sovente espressa come capacità di "scommettere" su certe proprietà dell'input. Per esempio, quando un automa viene usato per cercare determinate sequenze di caratteri (come: parole chiave) in una lunga porzione di testo, è utile "scommettere" che ci si trova all'inizio di una di tali sequenze e usare una sequenza di stati per verificare, carattere per carattere, che compaia la stringa cercata.

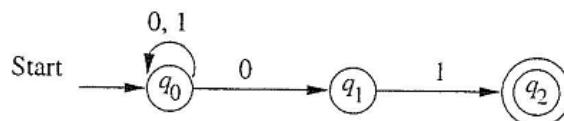
Prima di esaminare le applicazioni, è necessario definire gli automi a stati finiti non deterministici e mostrare che accettano gli stessi linguaggi accettati dai DFA. Come i DFA, gli NFA accettano proprio i linguaggi regolari. Tuttavia ci sono buone ragioni per occuparsi degli NFA. Spesso sono più succinti e più facili da definire rispetto ai DFA; inoltre, anche se è sempre possibile convertire un NFA in un DFA, quest'ultimo può avere esponenzialmente più stati di un NFA.

## Descrizione informale degli automi a stati finiti non deterministici

Come un DFA, un NFA ha un insieme finito di stati, un insieme finito di simboli di input, uno stato iniziale e un insieme di stati accettanti. Ha anche una funzione di transizione che chiameremo  $\delta$ . La differenza tra DFA ed NFA sta nel tipo di  $\delta$ . Per gli NFA,  $\delta$  è una funzione che ha come argomenti uno stato e un simbolo di input (come quella dei DFA), ma restituisce un insieme di zero o più stati (invece di un solo stato, come nel caso dei DFA). Cominceremo da un esempio di NFA e poi passeremo a precisare le definizioni.

### Esempio

$$L = \{w \in \{0,1\}^* \mid \exists x \in \{0,1\}^*: w = x01\}$$

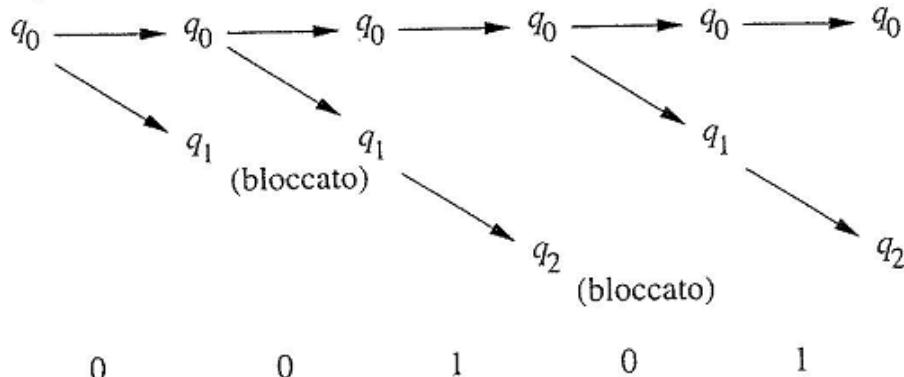


La Figura mostra un automa a stati finiti non deterministico con il compito di accettare tutte e sole le stringhe di 0 e di 1 che finiscono per 01. Lo stato  $q_0$  è lo stato iniziale e possiamo pensare che l'automa si trovi nello stato  $q_0$  (eventualmente insieme ad altri stati) quando non ha ancora "scommesso" che il finale 01 è cominciato. È sempre possibile che il simbolo successivo non sia il primo del suffisso 01, anche se quel simbolo è proprio 0. Dunque lo stato  $q_0$  può operare una transizione verso se stesso sia su 0 sia su 1.

Se però il simbolo successivo è 0, l'NFA scommette anche che è iniziato il suffisso 01. Un altro arco etichettato 0 conduce dunque da  $q_0$  allo stato  $q_1$ . Si noti che ci sono due archi etichettati 0 in uscita da  $q_0$ . L'NFA ha l'opzione di andare verso  $q_0$  oppure verso  $q_1$ , ed effettivamente fa entrambe le cose, come si vedrà quando si preciseranno le definizioni. Nello stato  $q_1$  l'NFA verifica che il simbolo successivo sia 1, e se è così, passa allo stato  $q_2$ , e accetta.

Si osservi che non ci sono archi uscenti da  $q_1$  etichettati 0 e non ci sono affatto archi uscenti da  $q_2$ . In tali situazioni il processo attivo dell'NFA corrispondente a quegli stati semplicemente "muore", sebbene altri

processi possano continuare a esistere. Mentre un DFA ha soltanto un unico arco uscente da ogni stato per ogni simbolo di input, un NFA non ha questi vincoli; nella Figura, per esempio, abbiamo visto casi in cui il numero di archi è zero, uno e due.



La Figura illustra come un NFA elabora gli input. Abbiamo mostrato che cosa succede quando l'automa della Figura precedente riceve la sequenza di input 00101. Esso parte dal solo stato iniziale,  $q_0$ . Quando viene letto il primo 0, l'NFA può passare allo stato  $q_0$  e allo stato  $q_1$ . In questo modo fa entrambe le cose. I due processi sono raffigurati nella seconda colonna della Figura sopra.

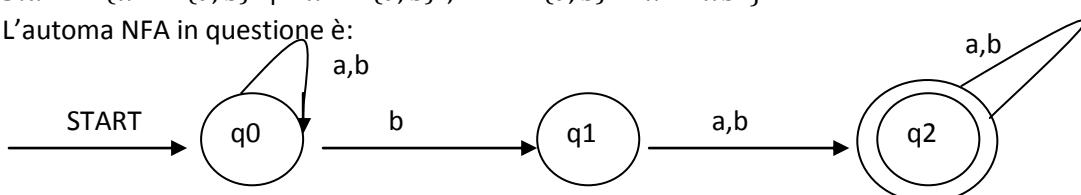
A questo punto viene letto il secondo 0. Lo stato  $q_0$  può nuovamente andare sia verso  $q_0$  sia verso  $q_1$ . Lo stato  $q_1$ , invece, non ha transizioni su 0, perciò "muore". Quando si presenta il terzo input, un 1, bisogna considerare le transizioni sia da  $q_0$  sia da  $q_1$ . Troviamo che  $q_0$  va solamente verso  $q_0$  su 1, mentre  $q_1$  va solamente verso  $q_2$ . Dunque, dopo aver letto 001, l'NFA si trova negli stati  $q_0$  e  $q_2$ . Poiché  $q_2$  è uno stato accettante, l'NFA accetta 001.

Ma l'input non è finito. Il quarto simbolo, uno 0, fa morire il processo associato a  $q_2$ , mentre  $q_0$  va sia in  $q_0$  sia in  $q_1$ . L'ultimo input, un 1, manda  $q_0$  in  $q_0$  e  $q_1$  in  $q_2$ . Dato che ci troviamo di nuovo in uno stato accettante, 00101 viene accettato. L'automa ad ogni passo, scommette che è il penultimo simbolo da leggere per poter accettare con l'ultimo.

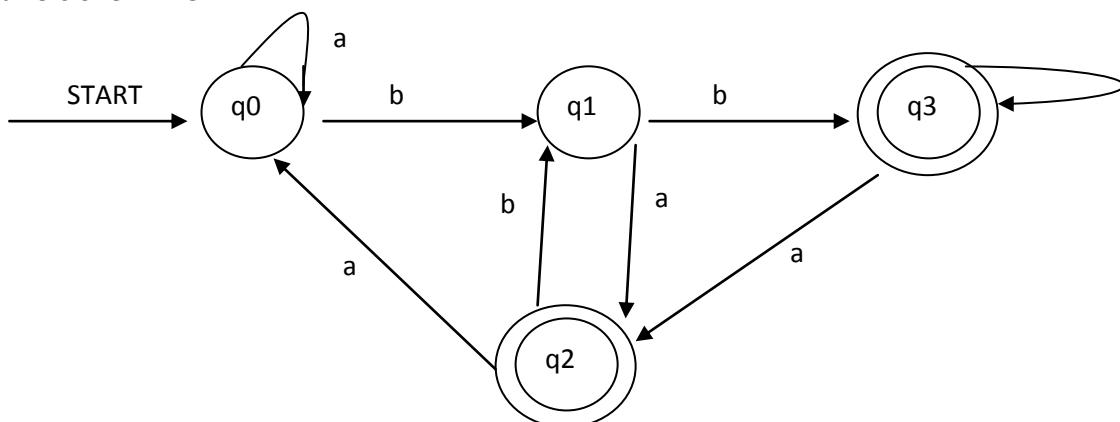
### Trasformazione da DFA a NFA

Sia  $L = \{w \in \{a,b\}^* \mid \exists x \in \{a,b\}^*, \exists z \in \{a,b\}^* : w = xbz\}$

L'automa NFA in questione è:



La versione DFA è



### Definizione di automa a stati finiti non deterministico

Presentiamo ora le nozioni formali relative agli automi a stati finiti non deterministici. Verranno sottolineate le differenze tra i DFA e gli NFA.

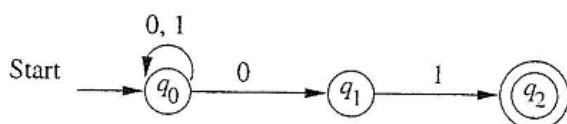
**Definizione:** Un automa finito non deterministico NFA si rappresenta essenzialmente come un DFA:

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

dove:

1.  $Q$  è un insieme finito di stati
2.  $\Sigma$  è un insieme finito e non vuoto di simboli di input
3.  $q_0$ , elemento di  $Q$ , è lo stato iniziale
4.  $F$ , un sottoinsieme di  $Q$ , è l'insieme degli stati finali (o accettanti)
5.  $\delta$ , la funzione di transizione, è la funzione che ha come argomenti uno stato in  $Q$  e un simbolo di input in  $\Sigma$ , e restituisce un sottoinsieme di  $Q$ . Si noti che l'unica differenza tra un NFA e un DFA è nel tipo di valore restituito da  $\delta$ : un insieme di stati nel caso di un NFA e un singolo stato nel caso di un DFA.

$\delta : Q \times \Sigma \rightarrow \wp(Q)$  dove  $\wp(Q) = \text{insieme delle parti di } Q$  (*insieme dei sottinsiemi di*  $Q$ )



L'NFA della Figura può essere specificato formalmente come

$$(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$$

dove la funzione di transizione  $\delta$  è data dalla tabella seguente:

$$\begin{array}{lll} \delta(q_0, 0) = \{q_0, q_1\} & \delta(q_0, 1) = \{q_0\} & \delta(q_1, 0) = \emptyset \\ \delta(q_1, 1) = \{q_2\} & \delta(q_2, 0) = \emptyset & \delta(q_2, 1) = \emptyset \end{array}$$

	0	1
→ q0	{q0, q1}	{q0}
q1	∅	{q2}
*q2	∅	∅

Si osservi che la tabella di transizione può specificare la funzione di transizione tanto per un NFA quanto per un DFA. L'unica differenza è che ogni voce nella tabella di un NFA è un insieme, anche se l'insieme è un singoletto, ossia ha un solo membro. Inoltre, se non c'è alcuna transizione da uno stato su un dato simbolo di input, la voce adeguata è  $\emptyset$ , l'insieme vuoto.

### La funzione di transizione estesa

Come per i DFA, bisogna estendere la funzione di transizione  $\delta$  di un NFA a una funzione  $\hat{\delta}$  che prende uno stato  $q$  e una stringa di simboli di input  $w$ , e restituisce l'insieme degli stati in cui si trova l'NFA quando parte dallo stato  $q$  ed elabora la stringa  $w$ . L'idea è suggerita nella figura di inizio pagina precedente; in sostanza, se  $q$  è l'unico stato nella prima colonna,  $\hat{\delta}(q, w)$  è la colonna di stati successivi alla lettura di  $w$ . Per esempio la Figura indica che  $\hat{\delta}(q_0, 001) = \{q_0, q_2\}$ .

**Definizione:** Dato un automa finito non deterministico NFA  $A = (Q, \Sigma, \delta, q_0, F)$ , la funzione di transizione estesa  $\hat{\delta}$  è la funzione  $\hat{\delta} : Q \times \Sigma^* \rightarrow \wp(Q)$  dove  $\wp(Q)$  definito induttivamente come segue:

**BASE:**  $\forall q \in Q, \hat{\delta}(q, \varepsilon) = \{q\}$ . Se nessun simbolo di input è stato letto, ci troviamo nel solo stato da cui siamo partiti.

**INDUZIONE:** Supponiamo che  $w$  sia della forma  $w = xa$ , dove  $a$  è il simbolo finale di  $w$  e  $x$  è la parte restante.

$$\forall q \in Q, \forall x \in \Sigma^*, \forall a \in \Sigma: \hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q, x)} \delta(p, a)$$

Un NFA può essere rappresentato attraverso:

- una quintupla
- diagramma (o grafo) delle transizioni
- tabella delle transizioni.

**Esempio:** Usiamo  $\hat{\delta}$  per descrivere come l'NFA precedente elabora l' input 00101.

**BASE:**  $\forall q \in Q, \hat{\delta}(q, \varepsilon) = \{q\}$

**INDUZIONE:**  $\forall q \in Q, \forall x \in \Sigma^*, \forall a \in \Sigma: \hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q, x)} \delta(p, a)$

Un compendio dei passi è:

1.  $\hat{\delta}(q_0, \varepsilon) = \{q_0\}$
2.  $\hat{\delta}(q_0, 0) = \bigcup_{p \in \hat{\delta}(q_0, \varepsilon)} \delta(p, 0) = \delta(q_0, 0) = \{q_0, q_1\}$
3.  $\hat{\delta}(q_0, 00) = \bigcup_{p \in \hat{\delta}(q_0, 0)} \delta(p, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
4.  $\hat{\delta}(q_0, 001) = \bigcup_{p \in \hat{\delta}(q_0, 00)} \delta(p, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$
5.  $\hat{\delta}(q_0, 0010) = \bigcup_{p \in \hat{\delta}(q_0, 001)} \delta(p, 0) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
6.  $\hat{\delta}(q_0, 00101) = \bigcup_{p \in \hat{\delta}(q_0, 0010)} \delta(p, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

Il punto (1) è la regola di base. Il punto (2) si ottiene applicando  $\delta$  all'unico stato,  $q_0$ , che si trova nell'insieme precedente: il risultato è  $\{q_0, q_1\}$ . Il punto (3) si ottiene prendendo l'unione, sui due stati dell'insieme precedente, di ciò che si ottiene quando si applica loro  $\delta$  con input 0. In altre parole  $\delta(q_0, 0) = \{q_0, q_1\}$ , mentre  $\delta(q_1, 0) = \emptyset$ . Per il punto (4) prendiamo l'unione di  $\delta(q_0, 1) = \{q_0\}$  e  $\delta(q_1, 1) = \{q_2\}$ . I punti (5) e (6) sono simili a (3) e (4).

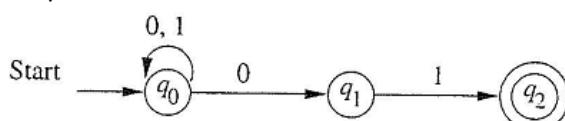
## Il linguaggio di un NFA

Come abbiamo suggerito, un NFA accetta una stringa  $w$  se, mentre si leggono i caratteri di  $w$ , è possibile fare una sequenza di scelte dello stato successivo che porta dallo stato iniziale a uno stato accettante. Il fatto che altre scelte per i simboli di input di  $w$  conducano a uno stato non accettante, oppure non conducano ad alcuno stato (cioè che una sequenza di stati muoia), non impedisce a  $w$  di venire accettato dall'NFA nel suo insieme.

**Definizione:** Sia  $A = (Q, \Sigma, \delta, q_0, F)$  un NFA. Il linguaggio riconosciuto da  $A$ ,  $L(A)$ , è definito da:

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

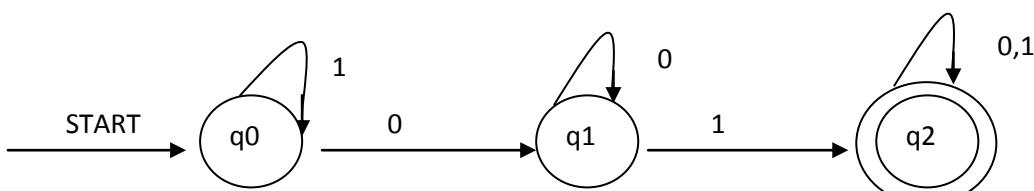
Esempio:



$$L(A) = \{w \in \{0,1\}^* \mid \exists x \in \{0,1\}^*: w=x01\}$$

Dimostrare che il linguaggio del DFA è quello appena descritto non è molto semplice, in quanto siamo nel caso di automi non deterministici.

Vediamo prima un esempio con un automa deterministico:



$$L(A) = \{w \in \{0,1\}^* \mid \exists x, y \in \{0,1\}^*: w=x01y\} = \{w \in \{0,1\}^* \mid \hat{\delta}(q_0, w) = q_2\} (*)$$

Devo dimostrare 3 affermazioni:

$$(1) \hat{\delta}(q_0, w) = q_0 \iff w = 1^k \ k \geq 0$$

$$(2) \hat{\delta}(q_0, w) = q_1 \iff w = 1^k 0^h \ con \ k \geq 0 \ h > 0$$

(3)  $\hat{\delta}(q_0, w) = q_2 \Leftrightarrow w = x01y \quad x, y \in \{0,1\}^*$

La conseguenza è che (\*) è vera.

Dimostriamo che (1)(2) e (3) sono vere usando la mutua induzione (nella lunghezza di w).

Per dimostrare questi enunciati, bisogna considerare in che modo A può raggiungere ciascuno stato; ossia: qual è stato l'ultimo simbolo di input, e in quale stato si trovava A prima di leggere quel simbolo?

Dato che il linguaggio di quest'automa è l'insieme delle stringhe w tali che  $\hat{\delta}(q_0, w)$  contiene q2 (perché q2 è l'unico stato accettante), la dimostrazione dei tre enunciati, in particolare di (3), garantisce che il linguaggio dell'NFA è l'insieme delle stringhe che finiscono per 01. La dimostrazione del teorema è un'induzione su  $|w|$ , la lunghezza di w, a partire dalla lunghezza 0.

**(BASE)** sia  $w = \epsilon$  ( $|w|=0$ ). Allora  $\hat{\delta}(q_0, w) = q_0$  e  $w=1^0$ . Quindi (1) è vera per  $w=\epsilon$ . Inoltre (2) e (3) sono vere per  $w=\epsilon$  perché entrambe le condizioni sono false per  $w=\epsilon$ . Dunque le ipotesi di entrambe le direzioni dell'enunciato se-e-solo-se sono false, e di conseguenza entrambe le direzioni dell'enunciato sono vere.

**(PASSO INDUTTIVO)** Supponiamo che  $w = xa$  con  $x \in \Sigma^*$ ,  $a \in \Sigma$

**(1)** Se  $\hat{\delta}(q_0, w) = \delta(\hat{\delta}(q_0, x), a) = q_0$ , dalla definizione del nostro automa (guardando il grafo) ricavo  $a = 1$ ,  $\hat{\delta}(q_0, x) = q_0$  (per ora ho visto solo 1) per ipotesi induttiva 1,  $x = 1^k$  con  $k \geq 0$  e quindi  $w = xa = 1^k 1$ .

**Viceversa** se  $w = 1^k 1$  allora  $\hat{\delta}(q_0, w) = \hat{\delta}(q_0, 1^k) = \hat{\delta}(\hat{\delta}(q_0, 1^{k-1}), 1) = (\text{per ipotesi ind. di 1}) \hat{\delta}(q_0, 1) = q_0$

**(2)** Supponiamo che  $\hat{\delta}(q_0, w) = \hat{\delta}(q_0, xa) = \delta(\hat{\delta}(q_0, x), a) = q_1$  allora dalla definizione del nostro automa (guardando il grafo), ricavo  $a=0$  e  $\hat{\delta}(q_0, x) \in \{q_0, q_1\}$ . Se  $\hat{\delta}(q_0, x) = q_0$  per l'ipotesi induttiva di (1),  $x = 1^k$  con  $k \geq 0$  mentre se  $\hat{\delta}(q_0, x) = q_1$ , per l'ipotesi induttiva di (2)  $x = 1^k 0^h$  con  $k \geq 0$   $h > 0$  e in ogni caso  $w = 1^k 0^h$  con  $k \geq 0$   $h > 0$ .

**Viceversa** se  $w = 1^k 0^h$  con  $k \geq 0$   $h > 0$  bisogna distinguere i due casi:

**(Caso a)**  $h = 1 \Rightarrow w = 1^k 0 \Rightarrow \hat{\delta}(q_0, w) = \delta(\hat{\delta}(q_0, 1^k), 0) = (\text{per ipotesi induttiva (1)})$

$\delta(q_0, 0) = q_1$  (definizione del nostro automa)

**(Caso b)**  $h > 1 \Rightarrow w = 1^k 0^h \Rightarrow \hat{\delta}(q_0, w) = \delta(\hat{\delta}(q_0, 1^k 0^{h-1}), 0) = (\text{per ipotesi induttiva (2)})$

$\delta(q_1, 0) = q_1$  (definizione del nostro automa)

**(3)** Se  $\hat{\delta}(q_0, w) = q_2$  allora  $w = z01y$  con  $z, y \in \{0,1\}^*$  e se  $w = z01y$  con  $z, y \in \{0,1\}^*$  allora  $\hat{\delta}(q_0, w) = q_2$ .

Se  $\hat{\delta}(q_0, w) = \delta(\hat{\delta}(q_0, x), a) = q_2$  allora  $\hat{\delta}(q_0, x) \in \{q_1, q_2\}$ . Se  $\hat{\delta}(q_0, x) = q_2$  allora per ipotesi induttiva di (3) ho  $x = z01y$  (in poche parole, già computando x sono in q2 ed ora con a resto ancora in q2); di conseguenza  $w = xa = z01ya$ . Se  $\hat{\delta}(q_0, x) = q_1$  allora  $a = 1$  e per ipotesi induttiva (2), ho  $x = z0$ ; quindi  $w = xa = z01$ .

**Viceversa** sia  $w = xa = z01y$  distinguiamo due casi: (I)  $y = \epsilon$ , (II)  $y = y'a$

**(Caso I)**  $y = \epsilon$ .

$\hat{\delta}(q_0, w) = \hat{\delta}(q_0, z01) = \delta(\hat{\delta}(q_0, z0), 1) = (\text{per ipotesi induttiva (2)}) = \delta(q_1, 1) = q_2$

**(Caso II)**  $y = y'a$ .

$\hat{\delta}(q_0, w) = \hat{\delta}(q_0, z01y) = \hat{\delta}(q_0, z01y'a) = \delta(\hat{\delta}(q_0, z01y'), a) = (\text{per ipotesi induttiva (3)}) = \delta(q_2, a) = q_2$

### Esercizio

$$L = \{w \in \{0,1\}^* \mid \forall a_1, a_2, a_3 \in \{0,1\}\}$$

$$\forall x, y \in \{0,1\}^* \text{ t.c. se } w = xa_1 a_2 a_3 y \text{ allora}$$

$$x, y \in \Sigma^* \quad xy = \{w \mid \exists x \in X, y \in Y: w = xy\}$$

$$a_1, a_2, a_3 = \{0,1\}^* \setminus \{w \in \{0,1\}^* \mid \exists x, y \in \{0,1\}^*: w = x111y\}$$

## Equivalenza di automi a stati finiti deterministici e non deterministici

Ci sono molti linguaggi per i quali è più facile costruire un NFA anziché un DFA, come l'esempio delle stringhe che finiscono per 01; eppure, per quanto sorprendente, ogni linguaggio che può essere descritto da un NFA può essere descritto anche da un DFA. Inoltre il DFA ha in pratica circa tanti stati quanti l'NFA, sebbene abbia spesso più transizioni. Nel peggiore dei casi, tuttavia, il più piccolo DFA può avere  $2^n$  stati, mentre il più piccolo NFA per lo stesso linguaggio ha solo  $n$  stati.

La dimostrazione che i DFA possono fare le stesse cose degli NFA si serve di un'importante costruzione, detta "**costruzione per sottoinsiemi**", in quanto comporta la costruzione di tutti i sottoinsiemi dell'insieme di stati dell'NFA. In generale molte dimostrazioni sugli automi richiedono la costruzione di un automa a partire da un altro. È importante considerare la costruzione per sottoinsiemi come esempio di descrizione formale di un automa nei termini degli stati e delle transizioni di un altro, senza conoscere i particolari del secondo automa.

### Famiglia di linguaggi per cui esiste un DFA

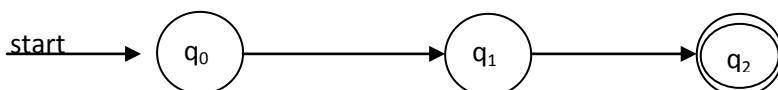
$\{L \subseteq \Sigma^* \mid L \text{ regolari}\} = \{L \subseteq \Sigma^* \mid \exists \text{ DFA } A: L = L(A)\} = (\text{verifica uguaglianza}) = \{L \subseteq \Sigma^* \mid \exists \text{ NFA } A: L = L(A)\}$

### PROVA COSTRUTTIVA

Vediamo che è costoso passare da 3 a 8 stati quindi da  $Q_N$  a  $Q_D$ :

$$n \rightarrow 2^n$$

Esempio:



$$A_N = \{q_0, q_1, q_2\}$$

$$A_D = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$$

B    C    D    E    F    G    H    I

$$Q_D = \wp(Q_N) \quad (\text{ad un insieme di stati associa un solo stato})$$

$$F_N = \{\{q_2\}\}$$

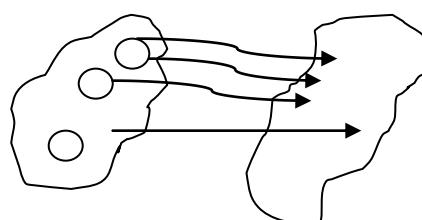
$$F_D = \{\{\{q_2\}\}, \{\{q_0, q_2\}\}, \{\{q_1, q_2\}\}, \{\{q_0, q_1, q_2\}\}\} = \{E, G, H, I\}$$

(Subset Construction)  $A_N = (Q_N, \Sigma, \delta_N, q_N, F_N)$  Definiamo un automa finito deterministico  $A_D = (Q_D, \Sigma, \delta_D, q_D, F_D)$  dove  $Q_D = \wp(Q_N)$ ,  $q_D = \{q_N\}$ ,  $F_D = \{S \in Q_D \mid S \cap F_N \neq \emptyset\}$

Deve contenere almeno uno stato finale  
 $\delta_D: Q_D \times \Sigma \rightarrow Q_D$

$$\forall S \in Q_D, \forall a \in \Sigma$$

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$



Esempio:

$$\delta_D(\{q_0, q_1\}, 0) = \bigcup_{p \in \{q_0, q_1\}} \delta_N(p, 0) = \delta_N(q_0, 0) \cup \delta_N(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$$

### Teorema 1

Per ogni automa finito non deterministico  $A_N = (Q_N, \Sigma, \delta_N, q_N, F_N)$  esiste un automa deterministico  $A_D = (Q_D, \Sigma, \delta_D, q_D, F_D)$  tale che  $L(A_N) = L(A_D)$

**DIMOSTRAZIONE (come la vuole per l'esame)**

La dimostrazione consiste in 3 passi:

**1)Passo 1**

Costruiamo un automa in DFA  $A_D$  a partire da  $A_N$  mediante la "subset construction" cioè  $A_D = (Q_D, \Sigma, \delta_D, q_0, F_D)$  dove  $Q_D = \wp(Q_N)$  (c'è una corrispondenza biunivoca tra gli insiemi degli stati deterministici con quelli non deterministici),

$$q_0 = \{q_N\},$$

$$F_D = \{S \subseteq Q_N \mid S \cap F_N \neq \emptyset\} \text{ e } \delta_D : Q_D \times \Sigma \rightarrow Q_D \text{ è definita da}$$

$$\forall S \subseteq Q_N, \forall a \in \Sigma$$

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$

**2)Passo 2**

$$\text{Dimostrare che per ogni } w \in \Sigma^* \text{ si ha } \widehat{\delta}_D(q_D, w) = \widehat{\delta}_N(q_N, w) \quad (*)$$

La dimostrazione avviene per induzione sulla lunghezza di  $w$ .

**(BASE)** Sia  $w = \varepsilon$  :  $\widehat{\delta}_D(q_D, \varepsilon) = (\text{def. di DFA}) = q_D = (\text{costruzione di DFA}) = \{q_N\} = (\text{def. Sulla funzione estesa di transizione di NFA}) = \widehat{\delta}_N(q_N, \varepsilon)$ . Quindi  $(*)$  è vera per  $w = \varepsilon$

**(PASSO INDUTTIVO)** Sia  $w = xa$   $x \in \Sigma^*$   $a \in \Sigma$

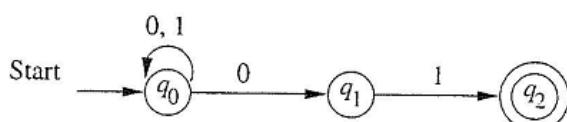
$$\begin{aligned} \widehat{\delta}_D(q_D, w) &= \widehat{\delta}_D(q_D, xa) = (\text{def. di DFA}) = \delta_D(\widehat{\delta}_D(q_D, x), a) = (\text{costr. di DFA}) \\ &= \bigcup_{p \in \widehat{\delta}_D(q_D, x)} \delta_N(p, a) = (\text{ipotesi induttiva}) = \bigcup_{p \in \widehat{\delta}_N(q_N, x)} \delta_N(p, a) = (x \text{ definz di NFA}) \\ &= \widehat{\delta}_N(q_N, xa) = \widehat{\delta}_N(q_N, w) \end{aligned}$$

Quindi  $(*)$  è vera.

**3)Passo 3**

Dimostrazione che  $L(A_D) = L(A_N)$

$$w \in L(A_D) \Leftrightarrow (\text{def. di linguaggio di un automa}) \Leftrightarrow \widehat{\delta}_D(q_D, w) \in F_D \Leftrightarrow (\text{per costruzione di } A_D) \Leftrightarrow \widehat{\delta}_D(q_D, w) \cap F_N \neq \emptyset \Leftrightarrow (\text{per il passo 2}) \Leftrightarrow \widehat{\delta}_N(q_N, w) \cap F_N \neq \emptyset \Leftrightarrow w \in L(A_N)$$

**Esempio**

$$A_D \quad 0 \quad 1$$

$A_D$	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_1\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$



$$\delta_D(\emptyset, a) = \bigcup_{p \in \emptyset} \delta_N(p, a) = \emptyset$$

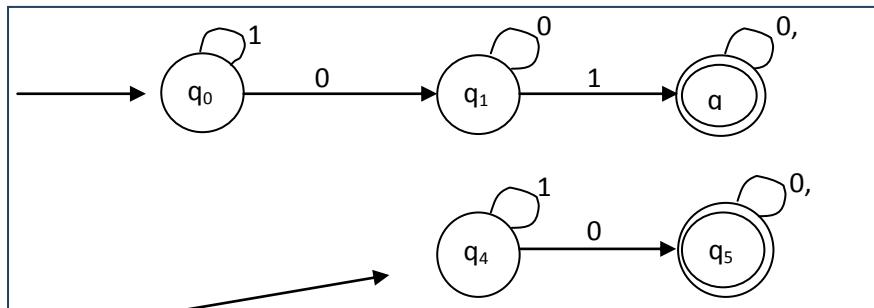
$$\delta_D(\{q_0\}, a) = \bigcup_{p \in \{q_0\}} \delta_N(p, a) = \delta_N(q_0, a)$$

$$\delta_D(\{q_0, q_1\}, a) = \bigcup_{p \in \{q_0, q_1\}} \delta_N(p, a) = \delta_N(q_0, a) \cup \delta_N(q_1, a)$$

$$\delta_D(\{q_0, q_1, q_2\}, a) = \bigcup_{p \in \{q_0, q_1, q_2\}} \delta_N(p, a) = \delta_N(q_0, a) \cup \delta_N(q_1, a) \cup \delta_N(q_2, a)$$

Esistono delle tecniche di ottimizzazione.

E' un automa deterministico?



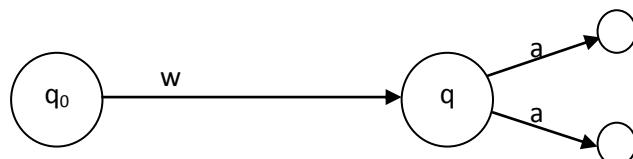
Non influente perché non ci sono cammini che raggiungono gli stati partendo da quello iniziale. Si eliminano gli stati che non sono raggiungibili dallo stato iniziale.

Uno stato  $q$  in un DFA  $A = \{Q, \Sigma, \delta, q_0, F\}$  è raggiungibile dallo stato iniziale se esiste una parola  $w \in \Sigma^*$  tale che  $\hat{\delta}(q_0, w) = q$

#### Algoritmo

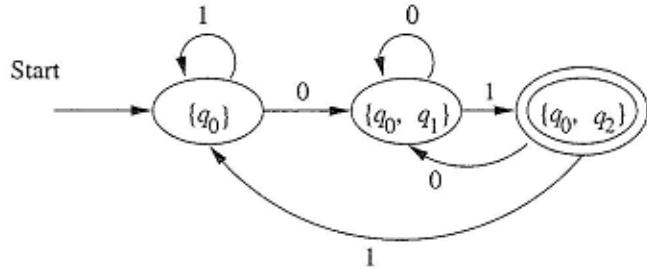
**(BASE)** Lo stato iniziale è raggiungibile ( $\hat{\delta}(q_0, \varepsilon) = q_0$ )

**(INDUZIONE)** Se  $q$  è raggiungibile allora lo è anche  $\hat{\delta}(q, a)$  per ogni  $a \in \Sigma$



Ottimizzazione della tabella precedente eliminando gli stati con valore  $\emptyset$

$A_D$	0	1
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$* \{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$



### Teorema 2

Sia  $A = (Q, \Sigma, \delta, q_0, F)$  un DFA, esiste un NFA  $A_N = (Q_N, \Sigma, \delta_N, q_N, F_N)$  tale che  $L(A) = L(A_N)$

È evidente in quanto lo possiamo vedere come un NFA che può fare una sola scelta.

**(Sketch)** Sia  $A = (Q, \Sigma, \delta, q_0, F)$  un DFA, considero l' NFA  $A_N = (Q_N, \Sigma, \delta_N, q_N, F_N)$  definito come segue  $Q_N = Q$ ,  $q_N = q_0$ ,  $F_N = F$ ,  $\forall q \in Q_N = Q \quad \forall a \in \Sigma \quad \delta_N(q, a) = \{\delta(q, a)\}$

Si può dimostrare che  $L(A_N) = L(A)$

### Corollario

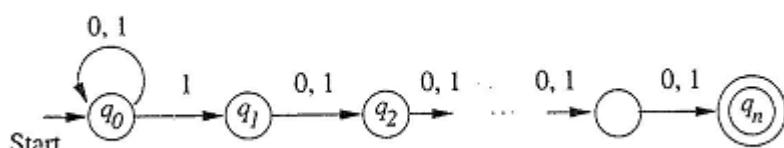
Un linguaggio è riconosciuto da un DFA se e solo se è riconosciuto da un NFA

### Un caso sfavorevole di costruzione per sottoinsiemi

Come già detto, nella pratica è normale che il DFA abbia approssimativamente lo stesso numero di stati dell'NFA a partire dal quale è stato costruito. Tuttavia è possibile una crescita esponenziale del numero degli stati; tutti i  $2^n$  stati del DFA che si possono costruire da un NFA di  $n$  stati potrebbero risultare accessibili. Il seguente esempio non raggiunge il limite, ma illustra un caso in cui il più piccolo DFA equivalente a un NFA di  $n+1$  stati ha  $2^n$  stati.

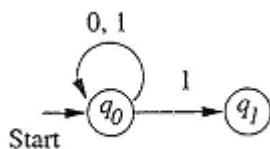
$L(N)$  è l'insieme di tutte le stringhe di 0 e di 1 tali che l' $n$ -esimo simbolo dalla fine sia 1. Intuitivamente un DFA  $D$  che accetti questo linguaggio deve ricordare gli ultimi  $n$  simboli che ha letto. Poiché ci sono  $2^n$  sequenze distinte di lunghezza  $n$ , se  $D$  avesse meno di  $2^n$  stati, ci sarebbe uno stato  $q$  raggiunto dopo aver letto due sequenze distinte di  $n$  bit.

(\*\*)

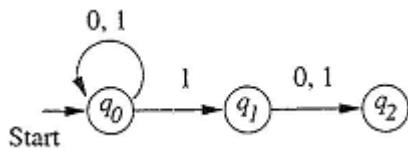


NFA ha  $n+1$  stati

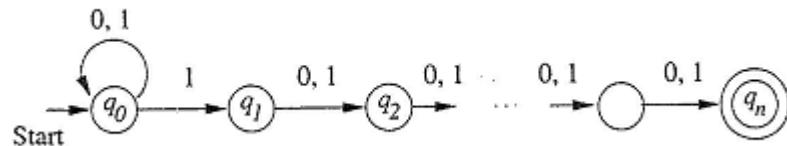
$n = 1$



$n=2$



n



Un DFA che riconosce lo stesso linguaggio di  $A_N$  non può avere meno di  $2^n$  stati.

Riconosce il linguaggio delle stringhe il cui n-esimo carattere è 1 {leggendo la stringa da destra contando i caratteri }

Un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  che sia t.c.  $L(A) = L(A_n)$  non può avere meno di  $2^n$  stati. Supponiamo per assurdo che esista un DFA A t.c.  $L(A) = L(A_n)$  con un numero di stati  $< 2^n$ .

### Principio della piccionaia

$$|\{0,1\}^n| = 2^n$$

x	y
a <sub>1</sub>	b <sub>1</sub>
a <sub>2</sub>	b <sub>2</sub>
a <sub>3</sub>	b <sub>3</sub>
.	.
.	.
.	.
a <sub>n</sub>	b <sub>k</sub>

In parole povere, se si hanno più piccioni che cellette e ogni piccione si ricovera in una celletta, allora ci deve essere almeno una celletta contenente più di un piccione. Nel nostro caso i "piccioni" sono le sequenze di n bit e le "cellette" sono gli stati. Poiché esistono meno stati che sequenze, a uno stato devono essere assegnate due sequenze.

Il principio della piccionaia può sembrare ovvio, ma dipende dal fatto che il numero delle cellette è finito. Funziona perciò per automi a stati finiti, in cui gli stati corrispondono alle cellette della piccionaia. Non si può applicare invece ad altri tipi di automi che hanno un numero infinito di stati.

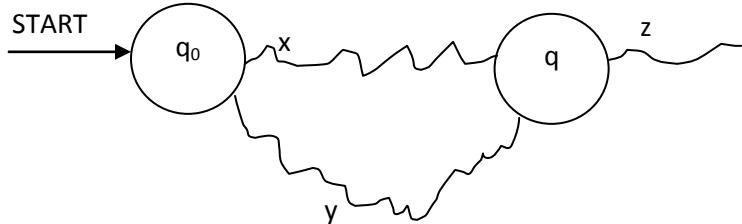
Per capire perché è essenziale che il numero delle cellette sia finito, si consideri la situazione infinita in cui le cellette corrispondono agli interi 1, 2, ... . Numeriamo i piccioni 0, 1, 2, ... , in modo che ci sia un piccione in più rispetto alle cellette. Possiamo allora mandare il piccione i nella celletta i + 1 per ogni i > 0. Così ogni piccione ha la sua celletta e non ci sono due piccioni obbligati a condividere lo stesso spazio.

Se considero gli insiemi degli stati a partire dagli stati iniziali a partire dalla stringa n

$$\{\hat{\delta}(q_0, x) \mid x \in \{0,1\}^n\} \subseteq Q$$

$$|\{\hat{\delta}(q_0, x) \mid x \in \{0,1\}^n\}| \leq 2^n$$

per il principio della piccionaia esistono almeno 2 stringhe  
 $x, y \in \{0,1\}^n, x \neq y \text{ t.c. } \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$



i

$$x = a_1 \dots 0 \dots a_n$$

$$y = b_1 \dots 1 \dots b_n$$

$$xz = a_1 \dots 0 \dots a_n 0^{i-1}$$

$$yz = b_1 \dots 1 \dots b_n 0^{i-1}$$

con  $z = 0^{i-1}$ . Inoltre, puntualizziamo che  $xz \notin L(A)$  mentre  $yz \in L(A)$

*PARENTESI:*  $\forall u, v \in \Sigma^*, \forall q \in Q$  abbiamo  $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$

$\hat{\delta}(q_0, xz) = \hat{\delta}(\hat{\delta}(q_0, x), z) = \hat{\delta}(\hat{\delta}(q_0, y), z) = \hat{\delta}(q_0, yz) \notin F$  e questo è ASSURDO perché in questo caso entrambi accettano  $xz \in L(A)$  e  $yz \in L(A)$ .

## Automi a stati finiti con epsilon-transizioni

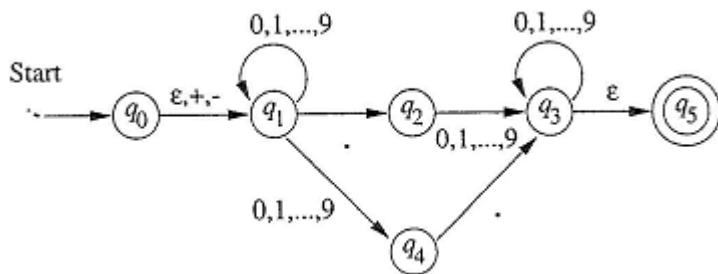
Presentiamo ora un'altra estensione degli automi a stati finiti. La novità consiste nell'ammettere transizioni sulla stringa vuota  $\epsilon$ . È come se l'NFA potesse compiere una transizione spontaneamente, senza aver ricevuto un simbolo di input. Come il non determinismo, anche questa nuova possibilità non amplia la classe dei linguaggi accettati dagli automi a stati finiti, ma offre una certa comodità notazionale.

Abbiamo 3 tipi di automi:

1. Automi finiti deterministici DFA
2. Automi finiti non deterministici NFA
3. Automi finiti con  $\epsilon$ -transizioni  $\epsilon$ -NFA

## Uso delle $\epsilon$ -transizioni

Cominceremo da una trattazione informale degli  $\epsilon$ -NFA, servendoci di diagrammi di transizione con etichette  $\epsilon$ . Negli esempi che seguono, gli automi accettano le sequenze di etichette lungo cammini dallo stato iniziale a uno stato accettante, considerando come invisibili le occorrenze di  $\epsilon$ , che quindi non contribuiscono a formare le stringhe associate ai cammini.



La Figura mostra un  $\epsilon$ -NFA che accetta numeri decimali (in notazione anglosassone) formati da:

1. un segno + o -, facoltativo
2. una sequenza di cifre
3. un punto decimale
4. una seconda sequenza di cifre.

Una delle due sequenze di cifre (punti (2) e (4)), ma non entrambe, può essere vuota. Di particolare interesse è la transizione da  $q_0$  a  $q_1$  su  $\epsilon$ , + o -. Lo stato  $q_1$  rappresenta quindi la situazione in cui sono stati letti il segno, se presente, ed eventualmente delle cifre, ma non il punto decimale. Lo stato  $q_2$  rappresenta la situazione in cui è stato letto il punto decimale, preceduto o no da cifre già lette. In  $q_4$  abbiamo letto almeno una cifra, ma non ancora il punto decimale. Lo stato  $q_3$  si interpreta così: sono stati letti il punto decimale e almeno una cifra, prima o dopo di quello. L'automa può restare in  $q_3$  e leggere nuove cifre, oppure può "scommettere" che le cifre sono terminate e spostarsi autonomamente nello stato accettante  $q_5$ .

**(DEFINIZIONE)** Possiamo rappresentare un  $\epsilon$ -NFA esattamente come un NFA, salvo che per un aspetto: la funzione di transizione deve incorporare informazioni sulle transizioni etichettate da  $\epsilon$ . Formalmente denotiamo un  $\epsilon$ -NFA (o automa finito con  $\epsilon$ -transizioni)  $A = (Q, \Sigma, \delta, q_0, F)$ , dove tutti i componenti si interpretano come per gli NFA:  $Q$ =insieme finito degli stati,  $q_0 \in Q$  stato iniziale,  $F \subseteq Q$ ,  $F$  insieme degli stati finali, mentre  $\delta$  è una funzione che richiede come argomenti:

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \wp(Q)$$

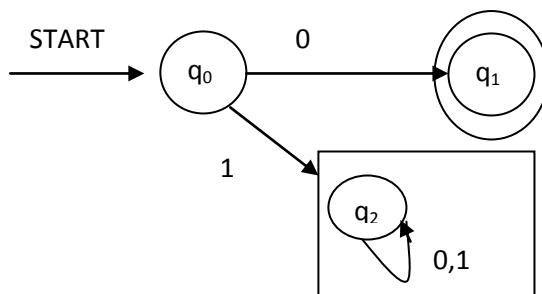
**TEOREMA** Un linguaggio  $L$  è riconosciuto da un automa finito con  $\epsilon$ -transizioni se e solo se  $L$  è riconosciuto da un' automa finito deterministico DFA.

N.B. Di tutto ciò a noi interessa solo la definizione e il teorema.

### Esercizio

Definire un' automa finito deterministico  $A$  t.c.  $L(A)=L$  dove  $L=\{ \dots \}$  (Lo stato pozzo può essere omesso)

$$L=\{0, x \mid x \in \{0,1\}^*\}$$

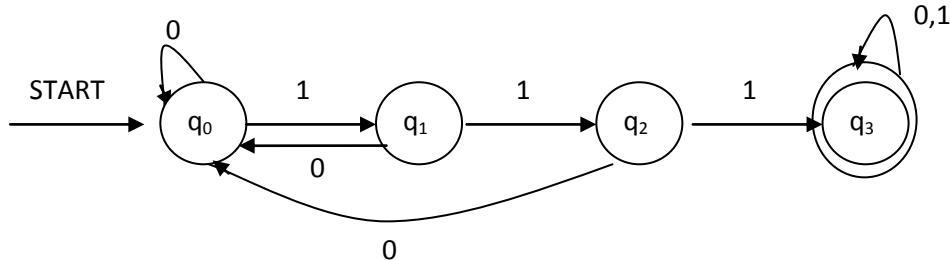


### Esercizio

Definire un DFA  $A$  t.c.

$L(A) = \{w \in \{0,1\}^* \mid \forall a_1, a_2, a_3 \in \{0,1\} \text{ t.c. } \exists x, y \in \{0,1\}^* \text{ w: } xa_1a_2a_3y, \text{ la stringa } a_1a_2a_3 \neq 111 \text{ (contiene almeno uno 0)}\} = L = 0,1^* \setminus \{111\}$

$L' = \{w \in \{0,1\}^* \mid \exists x, y \in \{0,1\}^*: w = x111y\}$

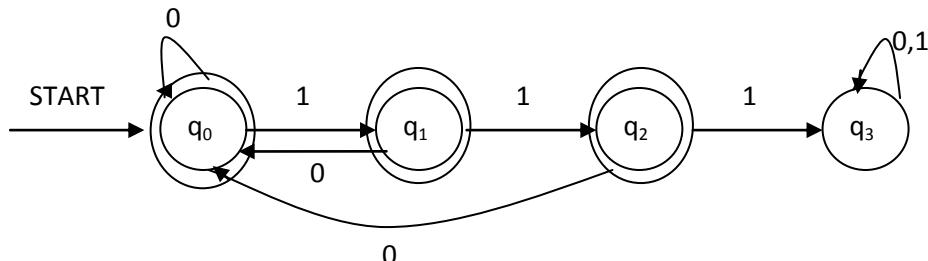


$$A' = \{Q', \Sigma, \delta', q_0, F'\} \quad L(A') = L'$$

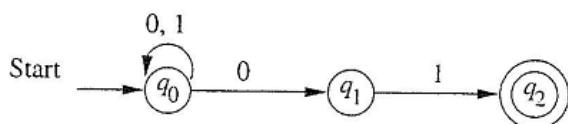
$$w \in L(A') \leftrightarrow \hat{\delta}(q_0, w) \in F'$$

$$w \notin L(A') \leftrightarrow \hat{\delta}(q_0, w) \notin F'$$

Per dimostrare questo, costruiamo il complemento dell'automa sopra



### Esercizio



$$L(A) = \{w \in \{0,1\}^* \mid \exists x \in \{0,1\}^*: w = x01\}$$

PROVA

$$1) \forall w \in \{0,1\}^* \quad q_0 \in \hat{\delta}(q_0, w)$$

$$2) q_1 \in \hat{\delta}(q_0, w) \leftrightarrow \exists x \in \{0,1\}^*: w = x0$$

$$3) w \in L(A) \leftrightarrow q_2 \in \hat{\delta}(q_0, w) \leftrightarrow \exists z \in \{0,1\}^*: w = z01$$

La dimostrazione avviene per mutua induzione sulla  $|w|$

(BASE): Sia  $w = \varepsilon \hat{\delta}(q_0, \varepsilon) = q_0$  1) vera poiché  $w \notin \delta(q_0, w)$  e poiché non termina per 0 è vera 2) e 3) (perché entrambe le cond. sono false per  $w = \varepsilon$ )

(PASSO INDUTTIVO) Sia  $w = xa$   $x \in \{0,1\}^*$ ,  $a \in \Sigma$

1.  $q_0 \in \hat{\delta}(q_0, w) = \hat{\delta}(q_0, xa) = \bigcup_{p \in \hat{\delta}(q_0, x)} \delta(p, a)$  devo applicare su  $x$  quindi per ipotesi induttiva 1)  $q_0 \in \hat{\delta}(q_0, x)$  quindi la 1) è vera
2.  $w = x0 \quad \hat{\delta}(q_0, w) = \hat{\delta}(q_0, x0) = \bigcup_{p \in \hat{\delta}(q_0, x)} \delta(p, 0)$  che per ipotesi induttiva 1),  $q_0 \in \hat{\delta}(q_0, w)$  quindi  $q_1 \in \{q_0, q_1\} = \delta(q_0, 0) \subseteq \hat{\delta}(q_0, w)$   
Supponiamo che  $q_1 \in \hat{\delta}(q_0, w) = \hat{\delta}(q_0, xa) = \bigcup_{p \in \hat{\delta}(q_0, x)} \delta(p, a)$   
Guardo il diagramma dell'automa  
 $q_1 \in \delta(p, a) \rightarrow a = 0 \rightarrow w = x0 \rightarrow$  la 2) è vera
3.  $\exists z \in \{0,1\}^* \quad w = z01 \quad \hat{\delta}(q_0, w) = \hat{\delta}(q_0, z01) = \bigcup_{p \in \hat{\delta}(q_0, z0)} \delta(p, 1)$   
 $q_1 \in \hat{\delta}(q_0, z0)$  (per ipo. ind. di 2)  $q_2 \in \delta(q_1, 1) \subseteq \hat{\delta}(q_0, w)$   
Viceversa se  $q_2 \in \hat{\delta}(q_0, w) = \hat{\delta}(q_0, xa) = \bigcup_{p \in \hat{\delta}(q_0, x)} \delta(p, a)$  Guardo il diagramma dell'automa  
Necessariamente  $a=1$ ,  $q_1 \in \hat{\delta}(q_0, x) \rightarrow$  (ipo. ind. 2)  $x = y0$ . Quindi  $w = xa = x1 = y01$  e 3) è vera.

## ESERCIZI CAPITOLO 2

**Esercizio:**

Sia

$$\begin{aligned}
 L &= \{w \in \{0,1\}^* \mid \text{ogni blocco di 3 caratteri consecutivi in } w \text{ contiene almeno uno } 0\} = \\
 &= \{w \in \{0,1\}^* \mid \forall a_1, a_2, a_3 \in \{0,1\}^*, \exists x, y \in \{0,1\}^* \text{ t.c. } w = x a_1 a_2 a_3 y \text{ e si ha } a_1 a_2 a_3 \\
 &\quad = z_1 0 z_2 \text{ con } z_1, z_2 \in \Sigma^*\} = \\
 &= \{w \in \{0,1\}^* \mid \forall a_1, a_2, a_3 \in \{0,1\}^*, \exists x, y \in \{0,1\}^* \text{ t.c. } w = x a_1 a_2 a_3 y \text{ si ha } a_1 a_2 a_3 \\
 &\quad = \{000, 001, 010, 011, 100, 101, 110\}
 \end{aligned}$$

**Esercizio 2.2.2:**

Abbiamo definito  $\widehat{\delta}$  scomponendo la stringa di input in una stringa seguita da un singolo simbolo (nella parte induttiva,  $\widehat{\delta}(q, w) = \delta(\widehat{\delta}(q, x), a)$ ). Informalmente possiamo pensare invece che  $\widehat{\delta}$  descriva ciò che capita lungo un cammino con una certa stringa di etichette; in tal caso non dovrebbe essere rilevante in che modo scomponiamo la stringa di input nella definizione di  $\widehat{\delta}$ . Mostrate che in effetti  $\widehat{\delta}(q, xy) = \widehat{\delta}(\widehat{\delta}(q, x), y)$  per qualunque stato  $q$  e qualunque coppia di stringhe  $x$  e  $y$ . Suggerimento: svolgete un'induzione su  $(|y|)$ .

**Esercizio 2.2.3:**

Dimostrate che, per qualunque stato  $q$ , stringa  $x$  e simbolo di input  $a$ ,  $\widehat{\delta}(q, ax) = \widehat{\delta}(\delta(q, a), x)$ . Suggerimento: usate l'Esercizio 2.2.2.

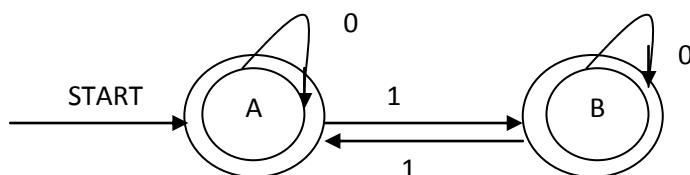
**Esercizio 2.2.10:**

Considerate il DFA con la seguente tabella di transizione:

	0	1
$\xrightarrow{} A$	A	B
$*B$	B	A

Descrivete in termini informali il linguaggio accettato da questo DFA e dimostrate per induzione sulla lunghezza della stringa di input che la descrizione è corretta. Suggerimento: nel porre l'ipotesi induttiva è consigliabile formulare un enunciato su quali input portano a ciascuno stato, e non solo su quali input portano allo stato accettante.

**Soluzione**



L'automa accetta l'input se contiene un numero dispari di 1.

$$L(A) = \{w \in \{0,1\}^* \mid \exists h \geq 0 : |w| = 2h + 1\}$$

**Esercizio 2.2.11:**

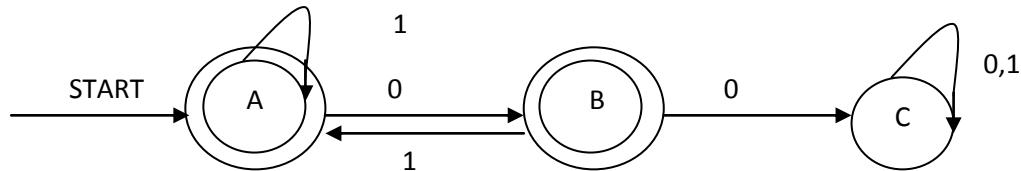
Considerate il DFA con la seguente tabella di transizione:

	0	1
$\xrightarrow{} *A$	B	A
$*B$	C	A
C	C	C

Descrivete in termini informali il linguaggio accettato da questo DFA e dimostrate per induzione sulla lunghezza della stringa di input che la descrizione è corretta. Suggerimento: nel porre l'ipotesi induttiva è consigliabile formulare un enunciato su quali input portano a ciascuno stato, e non solo su quali input portano allo stato accettante.

**Soluzione**

L'automa non accetta le stringhe che hanno 2 zero 00 consecutivi.



$$\{0,1\} \setminus L(A) = \{w \in \{0,1\}^* \mid \exists x, y \in \{0,1\}^* \text{ t.c. } w = x00y\}$$

$$L(A) = \{0,1\}^* \setminus \{w \in \{0,1\}^* \mid \exists x, y \in \{0,1\}^* \text{ t.c. } w = x00y\}$$

## CAPITOLO 3-4 – ESPRESSIONI E LINGUAGGI REGOLARI

Le espressioni regolari sono strettamente legate agli automi a stati finiti non deterministici e possono rappresentare una comoda alternativa alla notazione degli NFA per descrivere componenti software.

Dimostreremo che le espressioni regolari sono in grado di definire tutti e soli i linguaggi regolari.

Da una descrizione dei linguaggi fondata su un tipo di macchina, gli automi a stati finiti deterministicci e no, volgiamo ora l'attenzione a una descrizione algebrica: le "espressioni regolari". Scopriremo che le espressioni regolari definiscono esattamente gli stessi linguaggi descritti dalle varie forme di automa: i linguaggi regolari. Le espressioni regolari offrono comunque qualcosa in più rispetto agli automi: un modo dichiarativo di esprimere le stringhe da accettare. Perciò le espressioni regolari servono da linguaggio di input per molti sistemi che trattano stringhe.

### Pumping Lemma

Sia  $L$  un linguaggio regolare ( $L \subseteq \Sigma^*$ ).  $\exists n > 0$  (intero positivo dipendente da  $L$ ) tale che  $\forall w$  tale che  $w \in L$ ,  $|w| \geq n$ ,  $\exists x, y, z$  tali che:

- 1)  $w = xyz$
- 2)  $|xy| \leq n$
- 3)  $y \neq \epsilon$
- 4)  $\forall k \geq 0, xy^k z \in L$

### Dimostrazione

(Ipotesi) Supponiamo che  $L$  è un linguaggio regolare  $\rightarrow \exists$  un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  tale che  $L(A) = L$ .

Affermo che per  $n = |Q|$  l'enunciato è vero. Infatti:

sia  $w = a_1, \dots, a_m$ , con  $a_i \in \Sigma$ ,  $m \geq n$ , e tale che  $w \in L$ , cioè  $\hat{\delta}(q_0, w) \in F$

Considero  $a_1, \dots, a_m$  e  $p_0 = \hat{\delta}(q_0, \epsilon) = q_0$

$\hat{\delta}(q_0, a_1) = p_1$

$\hat{\delta}(q_0, a_1 a_2) = p_2$

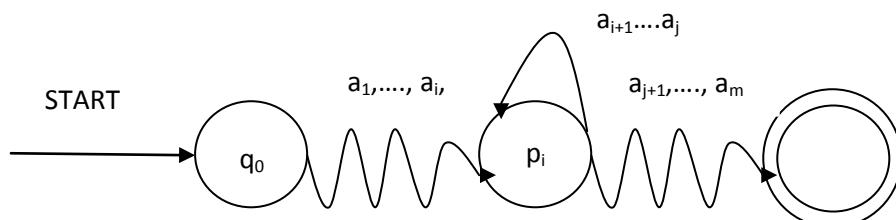
...

$\hat{\delta}(q_0, a_1, \dots, a_n) = p_n$

Cioè  $\hat{\delta}(q_0, a_1, \dots, a_i) = p_i$   $0 \leq i \leq n$

Dunque pi è lo stato in cui si trova A dopo aver letto i primi i simboli di  $w$ . Per il principio della piccionaia, dato che ci sono solo  $n$  stati, gli  $n+1$  stati pi per  $i=0, 1, \dots, n$  non possono essere tutti distinti. Ci sono perciò due interi distinti  $i$  e  $j$  tali che  $\hat{\delta}(q_0, a_1, \dots, a_i) = p_i = p_j = \hat{\delta}(q_0, a_1, \dots, a_j)$  con  $0 \leq i < j \leq n$

$w = a_1, \dots, a_i, a_{i+1}, \dots, a_j, a_{j+1}, \dots, a_m$



Scomponiamo  $w$  con:

Poniamo  $x = a_1, \dots, a_i$      $y = a_{i+1}, \dots, a_j$      $z = a_{j+1}, \dots, a_m$

## Teoria della Computazione 1

Caputo L. - Costante L. - Davino C. - Di Giacomo I. - Giordano A. - Palo U. - Vitale P.

Risulta  $w = (a_1, \dots, a_i) (a_{i+1}, \dots, a_j) (a_{j+1}, \dots, a_m) = xyz$  (Proposizione 1)

Inoltre  $|xy| = |a_1, \dots, a_i, a_{i+1}, \dots, a_j| = j-i \leq n$  (Proposizione 2)

$|y| = |a_{i+1}, \dots, a_j| = j-i > 0 \rightarrow y \neq \epsilon$  (Proposizione 3)

Si noti che  $x$  può essere vuota (quando  $i=0$ ), così come  $z$  (quando  $j=n=m$ ). viceversa,  $j$  non può essere vuota perché  $i$  è strettamente minore di  $j$ .

Proviamo la proposizione 4, cioè  $\forall k \geq 0, xy^k z \in L$

Passo preliminare Dimostriamo (per induzione su  $k$ ) che risulta

$$\forall k \geq 0 \quad \hat{\delta}(q_0, x) = \hat{\delta}(q_0, xy) = \hat{\delta}(q_0, xy^k) \quad (*)$$

**(Base)** Se  $k=0$ , l'automa passa dallo stato iniziale  $q_0$  (che è anche  $p_0$ ) a  $p_i$  sull'input  $x$ .

$\hat{\delta}(q_0, x) = \hat{\delta}(q_0, a_1, \dots, a_i) = p_i$ . Poiché  $p_i$  coincide con  $p_j$ ,  $p_i = p_j = \hat{\delta}(q_0, a_1, \dots, a_j) = \hat{\delta}(q_0, xy)$  (\*) è vero per  $k=0$

$$\begin{aligned} (\text{Passo Induttivo}) \quad k > 0 \text{ e } z = \epsilon \quad \hat{\delta}(q_0, xy^k) &= \hat{\delta}(\hat{\delta}(q_0, xy^{k-1}), y) = \hat{\delta}(\hat{\delta}(q_0, x), y) = \hat{\delta}(q_0, xy) \\ &\uparrow \quad \uparrow \quad \uparrow \\ \text{Per una proprietà} &\quad \text{Per ipotesi} & \text{Per una proprietà} \\ \text{della funzione di} &\quad \text{induttiva} & \text{della funzione di} \\ \text{transizione in un DFA} & & \text{transizione in un DFA} \end{aligned}$$

Se  $p_j = p_i = \hat{\delta}(q_0, a_1, \dots, a_i) = \hat{\delta}(q_0, x)$  (\*) è vero per  $k>0$

$$\begin{aligned} \forall k \geq 0 \quad \hat{\delta}(q_0, xy^k z) &= \hat{\delta}(\hat{\delta}(q_0, xy^k), z) = \hat{\delta}(\hat{\delta}(q_0, xy), z) = \hat{\delta}(q_0, xyz) = \hat{\delta}(q_0, w) \in F \quad (\text{perché per ipotesi} \\ &\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\ \text{Per una proprietà} &\quad \text{Per (*)} & \text{Per una proprietà} \\ \text{della funzione di} & & \text{della funzione di} \\ \text{transizione in un DFA} & & \text{transizione in un DFA} \end{aligned}$$

Siccome  $\forall k \geq 0, \hat{\delta}(q_0, xyz) \in F$ , concludiamo che  $\forall k \geq 0, xy^k z \in L$  (Proposizione 4) (fine dimostrazione)

$L$  regolare  $\rightarrow L$  verifica il Pumping Lemma

$L$  verifica il Pumping Lemma  $\rightarrow L$  regolare

Conclusione: Esistono linguaggi non regolari

**Esempio** Sia  $L = \{w \in \{0,1\}^* \mid |w|_0 = |w|_1\}$  Dimostriamo che  $L$  non è regolare.

Supponiamo per assurdo che  $L$  sia regolare. Quindi  $L$  verifica il P.L., e sia  $n$  la costante del PL associata ad  $L$ ,  $n > 0$ .

Sia  $w = 0^n 1^n$  Risulta  $w = 0^n 1^n \subseteq L$  e  $|w| \geq n$

Per il PL esistono  $x, y, z \in \{0,1\}^*$  tali che  $w = 0^n 1^n = xyz$  e sono verificate le proposizioni 2 3 e 4 del PL.

Siccome  $|xy| \leq n$  (prop 2) e  $y \neq \epsilon$  (prop 3), possiamo affermare che  $\exists t > 0$  tale che  $y = 0^t$

Deve risultare che  $xy^k z \in L, \forall k \geq 0$ . In particolare per  $k=0$  risulta  $xz \in L$ .

Poiché  $xz = x0^t z = w = 0^n 1^n \rightarrow xz = 0^{n-t} 1^n \notin L$  poiché  $n-t < n$  essendo  $t > 0$  ASSURDO  $\rightarrow L$  non è regolare

**Esempio** Sia  $L = \{1^p \mid p \text{ numero primo}\}$

**Dimostriamo che  $L$  non è regolare.**

Supponiamo che  $L$  sia regolare, e sia  $n$  la costante del PL associata a  $L$ .

Sia  $w = 1^p$  con  $p > n+1$ ,  $p$  primo. Allora  $w \in L$  e  $|w| \geq n$

## Teoria della Computazione 1

Caputo L. - Costante L. - Davino C. - Di Giacomo I. - Giordano A. - Palo U. - Vitale P.

Siccome  $L$  verifica il PL, esistono  $x, y, z \in \{1\}^*$  tali che  $w = 1^* = xyz$  e le proposizioni 2,3 e 4 del PL sono verificate.

Siccome  $y \neq \epsilon$ ,  $y = 1^q$  con  $q > 0$ ,  $q \leq n$ . Per la condizione 4 del PL,  $\forall k \geq 0, xy^kz \in L$ . In particolare per  $k=p-q$  ottengo  $xy^{p-q}z \in L \rightarrow |xy^{p-q}z|$  è un numero primo. Ma  $|xy^{p-q}z| = |xz| + |y^{p-q}| = (p-q) + q(p-q) = (p-q)(q+1)$ ;

Inoltre  $q > 0 \rightarrow q+1 > 1$ .  $p-q \geq p-n > 1$  perché  $p > n+1$  **ASSURDO** (poiché abbiamo ottenuto una fattorizzazione di  $p$  e nessuno dei due fattori è 1 o il numero stesso)  $\rightarrow L$  non è regolare

## Operazioni sui linguaggi

- Insiemistiche ( $U, \cap, \setminus$ )
- Concatenazione (o prodotto) di due linguaggi
- Chiusura di Kleene (o stella o star) di un linguaggio

**(DEFINIZIONE)** Siano  $L \subseteq \Sigma^*$ ,  $M \subseteq \Sigma^*$  due linguaggi, il prodotto di  $L$  e  $M$ , denotato con  $L \cdot M$  è definito da

$$L \cdot M = LM = \{w \in \Sigma^* \mid \exists l \in L, m \in M: w = lm\}$$

Esempio  $L = \{010, 1, \epsilon\}$   $M = \{00, 10\}$   $LM = \{01000, 01010, 100, 110, 00, 10\}$

Proprietà  $\{\epsilon\} M = M = M \{\epsilon\}$

Esempio  $L = \{a^n b^n \mid n \geq 0\}$   $M = \{cb, bb\}$   $LM = \{a^n b^n cb, a^n b^n bb \mid n \geq 0\}$

Linguaggio delle stringhe che contengono 0  $\{0, 1\}^* \{0\} \{0, 1\}^*$

**(DEFINIZIONE)** Potenza di un linguaggio  $L \subseteq \Sigma^*$  (Base)  $L^0 = \{\epsilon\}$  (Induzione)  $\forall n \geq 0 L^n = L^{n-1} \cdot L$

**(DEFINIZIONE)** Chiusura di Kleene di  $L \subseteq \Sigma^*$  Sia  $L \subseteq \Sigma^*$ , definiamo  $L^* = \bigcup_{n \geq 0} L^n$

## Espressioni Regolari

Qualsiasi algebra si forma a partire da alcune espressioni elementari, di solito costanti e variabili. Le algebre permettono poi di costruire ulteriori espressioni applicando un determinato insieme di operatori alle espressioni elementari e alle espressioni costruite in precedenza. Abitualmente è necessario anche un metodo per raggruppare gli operatori con i loro operandi, ad esempio attraverso le parentesi. Per esemplificare, l'algebra aritmetica comunemente nota parte da costanti come gli interi e i numeri reali, più le variabili, e costruisce espressioni più complesse con operatori aritmetici come + e x.

L'algebra delle espressioni regolari segue questo schema, usando costanti e variabili che indicano linguaggi e operatori per le tre operazioni: unione, punto e star. Possiamo descrivere le espressioni regolari in maniera ricorsiva, come segue. In questa definizione, non solo descriviamo che cosa sono le espressioni regolari lecite, ma per ogni espressione regolare  $E$  descriviamo il linguaggio che rappresenta, indicandolo con  $L(E)$ .

**DEF:** Un' espressione regolare su di un alfabeto  $\Sigma$  è definita ricorsivamente nel seguente modo:

**BASE:**  $\emptyset, \epsilon$ , a sono espressioni regolari,  $\forall a \in \Sigma$ .

**INDUZIONE:** Se  $E_1$  ed  $E_2$  sono espressioni regolari allora:

1.  $E_1 + E_2$  è un' espressione regolare
2.  $E_1 \cdot E_2$  è un' espressione regolare
3.  $E_1^*$  è un' espressione regolare
4.  $(E_1)$  è un' espressione regolare

## Teoria della Computazione 1

Caputo L. - Costante L. - Davino C. - Di Giacomo I. - Giordano A. - Palo U. - Vitale P.

Per ottenere tutte le stringhe di zero o più occorrenze di 01, usiamo ora l'espressione regolare  $(01)^*$ . Si osservi che mettiamo prima tra parentesi 01 per evitare di confonderci con l'espressione  $01^*$ , il cui linguaggio è formato da tutte le stringhe consistenti di uno 0 e di un qualunque numero di 1. Lo star ha la precedenza sul punto e perciò l'argomento dello star viene selezionato prima di compiere concatenazioni. Scriviamo un'espressione regolare per l'insieme delle stringhe che consistono di 0 e 1 alternati.

Tuttavia  $L((01)^*)$  non è esattamente il linguaggio che vogliamo, in quanto include solo le stringhe di 0 e 1 alternati che cominciano per 0 e finiscono per 1, mentre dobbiamo anche considerare la possibilità che ci sia un 1 all'inizio e uno 0 alla fine. Un modo può essere quello di costruire altre tre espressioni regolari che trattino le altre tre possibilità. In altre parole  $(10)^*$  rappresenta le stringhe alternate che cominciano per 1 e finiscono per 0,  $0(10)^*$  può essere usato per stringhe che cominciano e finiscono per 0, e  $1(01)^*$  si impiega per stringhe che cominciano e finiscono per 1. L'espressione regolare completa è

$$(01)^* + (10)^* + 0(10)^* + 1(01)^*.$$

### Regole di priorità

- 1) \* ha priorità più elevata rispetto a ( · e +)
- 2) · dopo priorità di \*
- 3) + più bassa priorità
- 4) () per cambiare l'ordine delle priorità

### Proprietà di chiusura dei linguaggi regolari

1. **L'unione di due linguaggi regolari è regolare.** Siano L ed M linguaggi sull'alfabeto  $\Sigma$ . Allora  $L \cup M$  è il linguaggio che contiene tutte le stringhe che si trovano in L o in M oppure in entrambi.
2. **L'intersezione di due linguaggi regolari è regolare.** Siano L ed M linguaggi sull'alfabeto  $\Sigma$ . Allora  $L \cap M$  è il linguaggio che contiene tutte le stringhe che si trovano sia in L che in M.
3. **Il complemento di un linguaggio regolare è regolare.** Siano L sull'alfabeto  $\Sigma$ . Allora  $\bar{L}$  è l'insieme delle stringhe in  $\Sigma^*$  che non sono in L.
4. **La differenza di due linguaggi regolari è regolare.** Siano L ed M linguaggi sull'alfabeto  $\Sigma$ . Allora  $L - M$  è il linguaggio che contiene tutte le stringhe che si trovano sia in L ma non in M.
5. **L'inversione di un linguaggio regolare è regolare**
6. **La chiusura (star) di un linguaggio regolare è regolare**
7. **La concatenazione di linguaggi regolari è regolare**

### Automi a stati finiti ed espressioni regolari

Descrivere un linguaggio per mezzo di un'espressione regolare è fondamentalmente diverso dal farlo per mezzo di un automa finito; eppure le due notazioni rappresentano di fatto la stessa classe di linguaggi, quelli che abbiamo denominato "regolari". Abbiamo già dimostrato che gli automi a stati finiti deterministici e i due tipi di automa a stati finiti non deterministici, con e senza  $\epsilon$ -transizioni, accettano la stessa classe di linguaggi. Per provare che le espressioni regolari definiscono la stessa classe, dobbiamo dimostrare due cose.

1. Ogni linguaggio definito da uno di questi automi è definito anche da un'espressione regolare. Per questa dimostrazione possiamo assumere che il linguaggio sia accettato da un DFA.
2. Ogni linguaggio definito da un'espressione regolare è definito da uno di questi automi. Per questa parte della dimostrazione è più semplice mostrare che esiste un NFA con  $\epsilon$ -transizioni che accetta lo stesso linguaggio.

Il linguaggio  $L(E)$ , denotato da un' espressione regolare E su  $\Sigma$ , è il linguaggio per  $\Sigma$  definito ricorsivamente come segue:

**BASE:**

- Se  $E = \emptyset$  allora  $L(E) = L(\emptyset) = \emptyset$
- Se  $E = \epsilon$  allora  $L(E) = L(\epsilon) = \{ \epsilon \}$
- Se  $E = a$  allora  $L(E) = L(a) = \{ a \}$

**INDUZIONE:**

- Se  $E_3 = E_1 + E_2$  con  $E_1$  ed  $E_2$  espressioni regolari, allora  $L(E_3) = L(E_1) \cup L(E_2)$
- Se  $E_3 = E_1 \cdot E_2$  con  $E_1$  ed  $E_2$  espressioni regolari, allora  $L(E_3) = L(E_1) \cdot L(E_2)$
- Se  $E_1 = E^*$  con  $E$  espressione regolare, allora  $L(E_1) = L(E)^*$
- Se  $E_1 = (E)$  con  $E$  espressione regolare, allora  $L(E_1) = L(L(E))$

**Teorema sui linguaggi regolari: dai DFA alle espressioni regolari**

Distinguiamo 2 teoremi sui linguaggi regolari

**Teorema 1**

Sia  $A$  un DFA, esiste un' espressione regolare  $E$  tale che  $L(A) = L(E)$

$$\{L \subseteq \Sigma^* \text{ t.c. } \exists \text{ un DFA } A \text{ t.c. } L=L(A)\} \subseteq \{L \subseteq \Sigma^* \text{ t.c. } \exists \text{ un E.R. } E \text{ t.c. } L=L(E)\}$$

**Teorema 2**

Sia  $E$  un' espressione regolare, esiste un DFA  $A$  tale che  $L(E) = L(A)$

$$\{L \subseteq \Sigma^* \text{ t.c. } \exists \text{ un E.R. } E \text{ t.c. } L=L(E)\} \subseteq \{L \subseteq \Sigma^* \text{ t.c. } \exists \text{ un DFA } A \text{ t.c. } L=L(A)\}$$

Teorema1 e Teorema2  $\rightarrow \{L \subseteq \Sigma^* \text{ t.c. } \exists \text{ un E.R. } E \text{ t.c. } L=L(E)\} = \{L \subseteq \Sigma^* \text{ t.c. } \exists \text{ un DFA } A \text{ t.c. } L=L(A)\}$

**Corollario 1**

Un linguaggio  $L$  è regolare se e solo se esiste un' espressione regolare  $E$  tale che  $L=L(E)$

**Corollario 2**

Il linguaggio  $L$  è riconosciuto da un DFA se e solo se  $L$  è denotato da un' espressione regolare  $E$ .

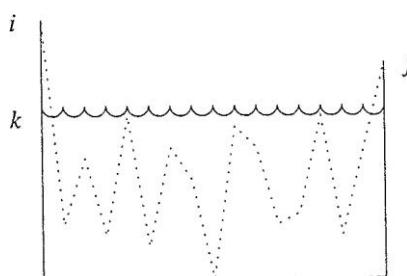
**DIMOSTRAZIONE TEOREMA 1**

Sia  $A$  un DFA  $(Q, \Sigma, \delta, q_0, F)$ . Scelgo di denotare gli stati di  $A$   $Q$ , mediante numeri interi  $\{1, 2, \dots, n\}$  per un intero  $n$ , quindi  $|Q|=n$ . Suppongo che lo stato iniziale sia 1.

**PASSO PRELIMINARE**

Voglio far vedere che esiste un' espressione regolare  $E$  che denota il linguaggio. Usando la notazione  $R_{i,j}^{(k)}$  come nome dell'espressione regolare il cui linguaggio è l'insieme di stringhe  $w$  t.c.  $w$  sia l'etichetta di un cammino dallo stato  $i$  allo stato  $j$  in  $A$ , ed il cammino non abbia alcun nodo intermedio il cui numero sia maggiore di  $k$ .

Si noti che i punti di partenza e di arrivo del cammino non sono intermedi; dunque non richiede che  $i$  e  $j$  siano inferiori o uguali a  $k$ .



Vado da  $i$  a  $j$  senza passare per i nodi, il cui indice, stabilito all'inizio su  $|Q|$  sia maggiore di  $k$ .

$$R_{i,j}^{(k)} = \{w \in \Sigma^* \text{ t.c. } \hat{\delta}(i,w)=j \text{ e } \forall x, y \text{ t.c. } w=xy \text{ e } x \neq \epsilon \text{ e } x \neq w \text{ } \hat{\delta}(i,x) \leq k\}$$

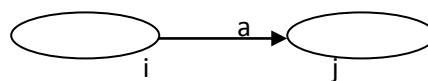
$i, j \in \{1, 2, \dots, n\}$  e  $0 \leq k \leq n \rightarrow$  DIMOSTRO PER INDUZIONE SU K.

**BASE: K=0.**

Dato che tutti gli stati sono numerati da 1 in su, la restrizione comporta che il cammino non abbia nessuno stato intermedio.

$$R_{i,j}^{(0)} = \{w \in \Sigma^* \text{ t.c. } \hat{\delta}(i,w)=j \text{ e } \forall x, y \text{ t.c. } w=xy \text{ e } x \neq \epsilon \text{ e } x \neq w \text{ } \hat{\delta}(i,x) \leq 0\} \rightarrow \text{Può contenere la parola di un carattere} \rightarrow w=a \rightarrow \{w \in \Sigma^* \text{ t.c. } \hat{\delta}(i,w)=j \text{ e } w \in \Sigma \cup \{\epsilon\}\}$$

- **CASO 1:  $w=a$   $\hat{\delta}(i,a)=j \rightarrow$**

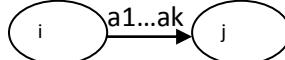


- **CASO 2:  $\hat{\delta}(i, \epsilon) = j \rightarrow$  Cammino di lunghezza 0(zero) che consiste solamente nel nodo i**

Distinguo due casi:

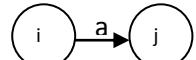
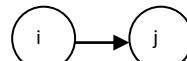
a)  $i \neq j$

- a.  $R_{i,j}^{(0)} = \emptyset$  allora non esiste un arco  $(i,j)$  NON ESISTE
- b.  $R_{i,j}^{(0)} = \{a\}$  allora esiste un arco  $(i,j)$  la cui etichetta è  $a$
- c.  $R_{i,j}^{(0)} = a_1 + a_2 + \dots + a_k$  allora esistono simboli  $\{a_1, a_2, \dots, a_k\}$  che etichettano archi dallo stato  $i$  allo stato  $j$ .



Per  $R_{i,j}^{(0)}$  esiste l'espressione regolare  $R_{i,j}^{(0)}$  che denota questo linguaggio:

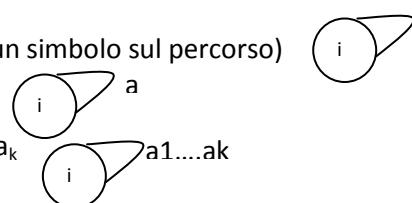
- ) Se  $R_{i,j}^{(0)} = \emptyset$  allora espressione regolare =  $\emptyset$
- ) Se  $R_{i,j}^{(0)} = \{a\}$  allora espressione regolare =  $a$
- ) Se  $R_{i,j}^{(0)} = \{a_1, a_2, \dots, a_k\}$  allora espressione regolare =  $(a_1 + a_2 + \dots + a_k)$



b)  $i=j$

I cammini leciti sono il cammino di lunghezza zero e tutti i cicli da  $i$  a se stesso

- a.  $R_{i,j}^{(0)} = \{\epsilon\} \rightarrow$  Cammino di lunghezza zero (Nessun simbolo sul percorso)
- b.  $R_{i,j}^{(0)} = \{\epsilon, a\} \rightarrow$  Un solo simbolo sul percorso
- c.  $R_{i,j}^{(0)} = \{\epsilon, a_1, a_2, \dots, a_k\} \rightarrow$  simboli da  $a_1, a_2, \dots, a_k$



- ) Se  $R_{i,j}^{(0)} = \{\epsilon\}$  allora espressione regolare =  $\epsilon$

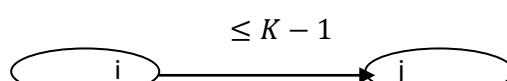
- ) Se  $R_{i,j}^{(0)} = \{\epsilon, a\}$  allora espressione regolare =  $\epsilon + a$

- ) Se  $R_{i,j}^{(0)} = \{\epsilon, a_1, a_2, \dots, a_k\}$  allora espressione regolare =  $(\epsilon + a_1 + a_2 + \dots + a_k)$

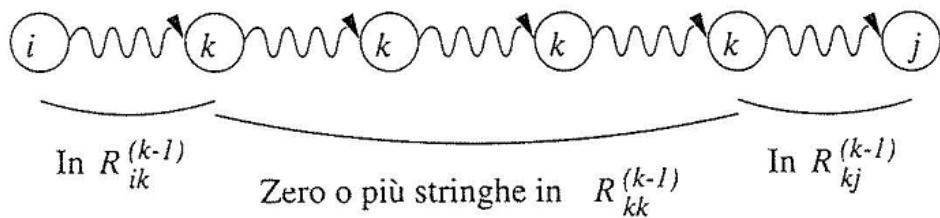
**PASSO INDUTTIVO:** (Suppongo vero l'enunciato per  $k-1$  e lo dimostro per  $k$ )

Supponiamo che esista un cammino dallo stato  $i$  allo stato  $j$  che non attraversa nessuno stato superiore a  $k$ . Bisogna prendere in considerazione due casi:

1. Il cammino non passa per lo stato  $k$ . In questo caso l'etichetta del cammino appartiene al linguaggio di  $R_{i,j}^{(k-1)}$



2. Il cammino passa per lo stato K almeno una volta. Allora è possibile scomporlo in segmenti come segue:



- a. Il primo segmento va dallo stato  $i$  allo stato  $k$  senza passare per  $k$   $R_{i,k}^{(k-1)}$
- b. L'ultimo va da  $k$  a  $j$  senza passare per  $k$   $R_{k,j}^{(k-1)}$
- c. Tutti i segmenti intermedi vanno da  $k$  a  $k$  senza passare per  $k$   $R_{k,k}^{(k-1)}$

**N.B. Se il cammino passa attraverso  $k$  una sola volta non esistono segmenti mediani, ma solo un cammino da  $i$  a  $k$  e da  $k$  a  $j$ .**

L'etichetta per il cammino definito nel caso b è:

$$(R_{i,k}^{(k-1)}) \cdot (R_{k,k}^{(k-1)})^* (R_{k,j}^{(k-1)})$$

Con il seguente significato:

- a.  $(R_{i,k}^{(k-1)})$  → Rappresenta la parte del cammino che giunge allo stato  $k$  la prima volta
- b.  $(R_{k,k}^{(k-1)})^*$  → Rappresenta la porzione che va da  $K$  a se stesso zero o più volte
- c.  $(R_{k,j}^{(k-1)})$  → Rappresenta la parte del cammino che lascia  $K$  per l'ultima volta e va in  $j$

Combinando le espressioni per i due casi otteniamo che:

$$(R_{i,j}^{(k)}) = (R_{i,j}^{(k-1)}) + (R_{i,k}^{(k-1)}) \cdot (R_{k,k}^{(k-1)})^* (R_{k,j}^{(k-1)}) \rightarrow \text{per le etichette di tutti i cammini dallo stato } i \text{ allo stato } j \text{ che non passano per gli stati più alti di } k.$$

**Per definizione di Espressione Regolare: Esiste un espressione regolare per  $R_{i,j}^{(k)}$**

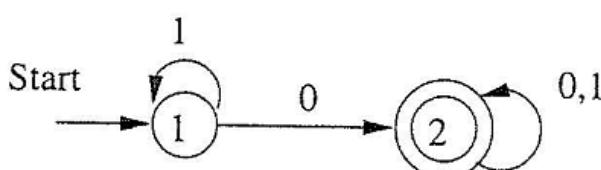
Infine abbiamo che  $R_{i,j}^{(n)}$  è per ogni  $i$  e  $j$

Possiamo assumere che lo stato 1 sia lo stato iniziale, mentre gli stati accettanti possono essere qualunque insieme di stati. L'espressione regolare per il linguaggio dell'automa è allora la somma (unione) di tutte le espressioni  $R_{i,j}^{(n)}$  in cui  $j$  sia uno stato accettante (finale).

$$L(A) = \bigcup_{j \in F} R_{1,j}^{(n)} \rightarrow L(A) = \sum_{j=1; j \in F}^n R_{1,j}^{(n)} \rightarrow \text{ESPRESSONE REGOLARE } L(A)$$

#### Esempio di costruzione

Convertiamo il DFA della Figura in un'espressione regolare. Questo DFA accetta tutte le stringhe che contengono almeno uno 0. Per capirne la ragione, si noti che l'automa va dallo stato iniziale 1 allo stato accettante 2 non appena vede l'input 0. L'automa rimane allora nello stato 2 su tutte le sequenze di input.



Per esempio  $R_{11}^{(0)}$  ha il termine  $\epsilon$  perché gli stati iniziale e finale sono gli stessi: lo stato 1. Ha il termine 1 perché esiste un arco dallo stato 1 allo stato 1 su input 1. Come ulteriore esempio,  $R_{12}^{(0)}$  è 0 perché esiste un arco etichettato 0 dallo stato 1 allo stato 2. Non ci sono termini  $\epsilon$  in quanto gli stati iniziale e finale sono diversi. Come terzo esempio, abbiamo che  $R_{21}^{(0)} = \emptyset$  perché non esiste nessun arco dallo stato 2 allo stato 1.

Ora dobbiamo passare alla parte induttiva, ossia dobbiamo costruire espressioni più complesse che prima prendono in considerazione cammini che passano attraverso lo stato 1 e poi cammini che passano attraverso gli stati 1 e 2, cioè cammini arbitrari. La regola per il calcolo delle espressioni  $R_{ij}^{(1)}$  è un caso della regola generale data nella parte induttiva del Teorema:

$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)}(R_{11}^{(0)})^* R_{1j}^{(0)}$$

Il prospetto della Tabella riporta le espressioni computate per sostituzione diretta nella formula sopra, seguite da espressioni semplificate di cui possiamo dimostrare, attraverso un ragionamento ad hoc, che rappresentano lo stesso linguaggio dell'espressione più complessa.

$$R_{11}^{(1)} = R_{11}^{(0)} + R_{11}^{(0)}(R_{11}^{(0)})^* R_{11}^{(0)} = \epsilon + 1 + (\epsilon + 1)(\epsilon + 1)^*(\epsilon + 1) = 1^*$$

$$R_{12}^{(1)} = R_{12}^{(0)} + R_{11}^{(0)}(R_{11}^{(0)})^* R_{12}^{(0)} = 0 + (\epsilon + 1)(\epsilon + 1)^*(0) = 0 + 1^*0 = 1^*0$$

$$R_{21}^{(1)} = R_{21}^{(0)} + R_{21}^{(0)}(R_{11}^{(0)})^* R_{11}^{(0)} = \emptyset + \emptyset(\epsilon + 1)^*(\epsilon + 1) = \emptyset$$

$$R_{22}^{(1)} = R_{22}^{(0)} + R_{21}^{(0)}(R_{11}^{(0)})^* R_{12}^{(0)} = \epsilon + 0 + 1 + \emptyset(\epsilon + 1)^*(0) = \epsilon + 0 + 1$$

Per esempio consideriamo  $R_{12}^{(1)}$ . La sua espressione è  $R_{12}^{(0)} + R_{11}^{(0)}(R_{11}^{(0)})^* R_{12}^{(0)}$ , che si ottiene dalla regola specificata sopra sostituendo i = 1 e j = 2.

Per comprendere la semplificazione, si osservi il principio generale che se R è una qualunque espressione regolare, allora  $(\epsilon + R)^* = R^*$ . La giustificazione è che ambedue i membri dell'equazione descrivono il linguaggio costituito da qualsiasi concatenazione di zero o più stringhe di  $L(R)$ . Nel nostro caso abbiamo  $(\epsilon + 1)^* = 1^*$ ; ambedue le espressioni denotano un numero arbitrario di 1. Inoltre  $(\epsilon + 1)1^* = 1^*$ . Ancora una volta si può osservare che entrambe le espressioni denotano "qualunque numero di 1". Dunque l'espressione originale  $R_{12}^{(1)}$  è equivalente a  $0 + 1^*0$ . Tale espressione denota il linguaggio che contiene la stringa 0 e tutte le stringhe formate da uno 0 preceduto da un numero arbitrario di 1. Questo linguaggio è indicato anche dall'espressione più semplice  $1^*0$ .

La semplificazione di  $R_{11}^{(1)}$  è simile alla semplificazione di  $R_{12}^{(1)}$  appena considerata.

La semplificazione di  $R_{21}^{(1)}$  e  $R_{22}^{(1)}$  dipende da due regole relative a  $\emptyset$ . Per qualunque espressione regolare R, vale quanto segue:

1.  $\emptyset R = R\emptyset = \emptyset$ . In altre parole,  $\emptyset$  è un annichilatore per la concatenazione; è esso stesso il risultato della concatenazione con una qualsiasi espressione a sinistra o a destra. Tale regola ha un senso perché, se vogliamo che una stringa sia nel risultato di una concatenazione, dobbiamo trovare stringhe da entrambi gli argomenti della concatenazione. Se uno degli argomenti è  $\emptyset$ , è impossibile trovare una stringa per quell'argomento.
2.  $\emptyset + R = R + \emptyset = R$ . Ossia  $\emptyset$  è l'identità per l'unione; quando compare in un'unione, il risultato è l'altra espressione.

Di conseguenza un'espressione come  $\emptyset (\epsilon+1)^* (\epsilon + 1)$  può essere sostituita da  $\emptyset$ . Le ultime due semplificazioni dovrebbero essere chiare.

Calcoliamo ora l'espressione  $R_{ij}^{(2)}$ . La regola induttiva applicata con  $k = 2$  dà:

$$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)}(R_{22}^{(1)})^*R_{2j}^{(1)}$$

Se sostituiamo nella formula sopra riportata le espressioni semplificate della Tabella sopra, otteniamo le espressioni della Tabella sotto. Tale figura mostra anche le semplificazioni che seguono gli stessi principi descritti per la Tabella sopra.

	Per sostituzione diretta	Semplificata
$R_{11}^{(2)}$	$1^* + 1^*0(\epsilon + 0 + 1)^*\emptyset$	$1^*$
$R_{12}^{(2)}$	$1^*0 + 1^*0(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$	$1^*0(0 + 1)^*$
$R_{21}^{(2)}$	$\emptyset + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*\emptyset$	$\emptyset$
$R_{22}^{(2)}$	$\epsilon + 0 + 1 + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$	$(0 + 1)^*$

L'espressione regolare finale equivalente all'automa della figura è costruita prendendo l'unione di tutte le espressioni in cui il primo stato è lo stato iniziale e il secondo stato è quello accettante. In questo esempio, con 1 come stato iniziale e 2 come unico stato accettante, abbiamo bisogno solamente dell'espressione R. Tale espressione è  $1^*0(0 + 1)^*$ , ed è di facile interpretazione. Il suo linguaggio consiste di tutte le stringhe che cominciano con zero o più 1, poi presentano uno 0, e poi qualunque stringa di 0 e 1. In altri termini, il linguaggio è "tutte le stringhe di 0 e 1 con almeno uno 0".

### Conversione di espressioni regolari in automi

**(TEOREMA)** Per ogni **espressione regolare E** esiste un  $\epsilon$ -NFA A tale che  $L(E) = L(A)$ . Ogni linguaggio definito da una espressione regolare è definito anche da un automa a stati finiti.

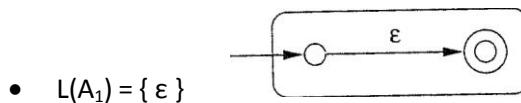
**(DIMOSTRAZIONE)** La dimostrazione è un'induzione strutturale sull'espressione E.

Cominciamo costruendo gli automi per le espressioni di base: singoli simboli,  $\epsilon$ ,  $\emptyset$ . Mostriamo poi come combinare questi automi in automi più grandi che accettano l'unione, la concatenazione o la chiusura dei linguaggi accettati dagli automi più piccoli. Tutti gli automi che costruiamo sono  $\epsilon$ -NFA con un unico stato accettante.

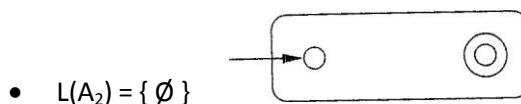
Per ogni espressione regolare E esiste un  $\epsilon$ -NFA A tale che:

1. A ha un solo stato finale  $q_f$ ;
2. Non esistono archi uscenti dallo stato finale  $q_f$ , ( $\delta(q_f, a) = \emptyset \forall a$  lettera);
3. Non esistono archi entranti nello stato iniziale  $q_0$ , ( $\forall$  stato q,  $\forall$  lettera a,  $\delta(q, a) \neq q_0$ );
4.  $L(E) = L(A)$ .

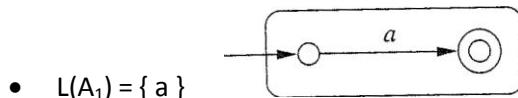
**(BASE)** Devo dimostrare che esiste un  $\epsilon$ -NFA A tale che le proprietà dalla "1" alla "4" siano vere per  $E = \emptyset$  oppure per  $E = \epsilon$  oppure  $E = a$ ,  $a \in \Sigma$ .



Il linguaggio dell'automa è  $\{ \epsilon \}$ , dato che l'unico cammino dallo stato iniziale verso uno stato accettante è etichettato  $\epsilon$ .



Il linguaggio dell'automa è  $\{ \emptyset \}$ , non esistono cammini dallo stato iniziale a allo stato accettante.



Il linguaggio dell'automa è  $\{ a \}$ , rappresentato dall'automa per l'espressione regolare a.

E' facile verificare che tutti questi automi soddisfano le condizioni dalla "1" alla "3".

**(PASSO INDUTTIVO)** Per definizione l'espressione regolare E è costruita da sottoespressioni  $E_1, E_2$  o da  $E_1$ .

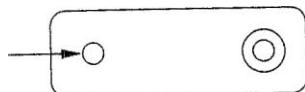
Per definizione di espressione regolare abbiamo i seguenti 4 casi:

1.  $E = E_1 + E_2$
2.  $E = E_1 \cdot E_2$
3.  $E = E_1^*$
4.  $E = (E_1)$

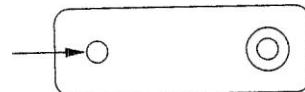
#### Caso 1:

$E = E_1 + E_2$  con  $E_1, E_2$  espressioni regolari.

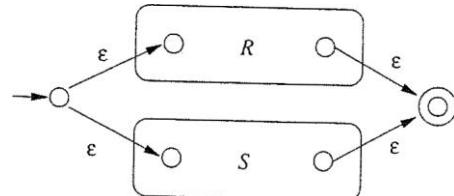
Per ipotesi induttiva esiste un  $\epsilon$ -NFA  $A_1$  tale che da "1" a "3" sono vere e  $L(A_1) = L(E_1)$ .



Esiste un  $\epsilon$ -NFA  $A_2$  tale che tale che da "1" a "3" sono vere e  $L(A_2) = L(E_2)$ .



Allora,  $L(A) = L(A_1) \cup L(A_2) = L(E_1) \cup L(E_2) = L(E_1 + E_2) = L(E)$



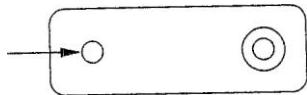
L'espressione è  $R + S$  per due espressioni più piccole  $R$  ed  $S$ . In altre parole, partendo dal nuovo stato iniziale, possiamo andare nello stato iniziale dell'automa per  $R$  oppure in quello dell'automa per  $S$ . Raggiungiamo poi lo stato accettante di uno di questi automi seguendo un cammino etichettato da una stringa che si trova rispettivamente in  $L(R)$  oppure  $L(S)$ . Una volta raggiunto lo stato accettante dell'automa per  $R$  o  $S$ , possiamo seguire uno degli archi  $\epsilon$  verso lo stato accettante del nuovo automa. Il linguaggio dell'automa è perciò  $L(R) \cup L(S)$ .

#### Caso 2: $E = E_1 \cdot E_2$ con $E_1, E_2$ espressioni regolari.

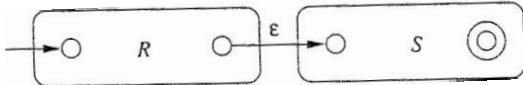
Per ipotesi induttiva esiste un  $\epsilon$ -NFA  $A_1$  tale che da "1" a "3" sono vere e  $L(A_1) = L(E_1)$ .



Esiste un  $\epsilon$ -NFA  $A_2$  tale che da "1" a "3" sono vere e  $L(A_2) = L(E_2)$ .



Allora  $L(A) = L(A_1)$   $L(A_2) = L(E_1)$   $L(E_2) = L(E_1 E_2) = L(E)$



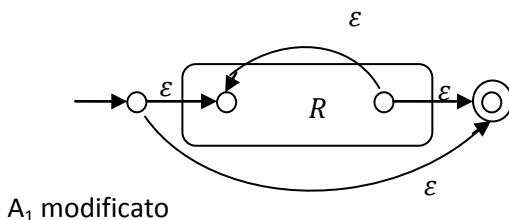
L'espressione è  $RS$  per due espressioni più piccole  $R$  ed  $S$ . Si noti che lo stato iniziale del primo automa diventa il nuovo stato iniziale, e lo stato accettante del secondo automa diventa lo stato accettante dell'automa complessivo. L'idea è che un cammino dallo stato iniziale a quello accettante deve attraversare prima l'automa per  $R$  seguendo un cammino etichettato da una stringa in  $L(R)$ , e poi quello per  $S$  seguendo un cammino etichettato da una stringa in  $L(S)$ . Quindi i cammini nell'automa sono tutti e soli quelli etichettati dalle stringhe in  $L(R)L(S)$ .

**Caso 3:**  $E = E_1^*$  con  $E_1$  espressione regolare.

Per ipotesi induttiva esiste un  $\epsilon$ -NFA  $A_1$  tale che da "1" a "3" sono vere e  $L(A_1) = L(E_1)$ .



Ricordiamo  $L(A) = L(A_1^*) = L(E_1^*) = L(E)$



L'espressione è  $E^*$  per un'espressione più piccola  $E$ . L'automa che offre due tipi di percorso.

- Direttamente dallo stato iniziale allo stato accettante lungo un cammino etichettato  $\epsilon$ . Tale cammino fa accettare  $\epsilon$ , che si trova in  $L(R)$  a prescindere da  $R$ .
- Verso lo stato iniziale dell'automa per  $R$ , attraverso tale automa una o più volte, e poi verso lo stato accettante. Questo insieme di cammini ci permette di accettare stringhe in  $L(R)$ ,  $L(R)L(R)$ ,  $L(R)L(R)L(R)$ , e così via, coprendo così tutte le stringhe in  $L(R^*)$ , eccetto eventualmente  $\epsilon$ , già coperta dall'arco diretto verso lo stato accettante.

**Caso 4:**  $E = (E_1)$  con  $E_1$  espressione regolare.

Ricordiamo che  $L(E) = L((E_1)) = L(E_1)$ .

Quindi  $A = (A_1)$  verifica le condizioni da "1" a "4"

Infatti  $A_1$  verifica le condizioni da "1" a "3" per ipotesi induttiva e  $L(A_1) = L(E_1) = L((E_1)) = L(E)$ .

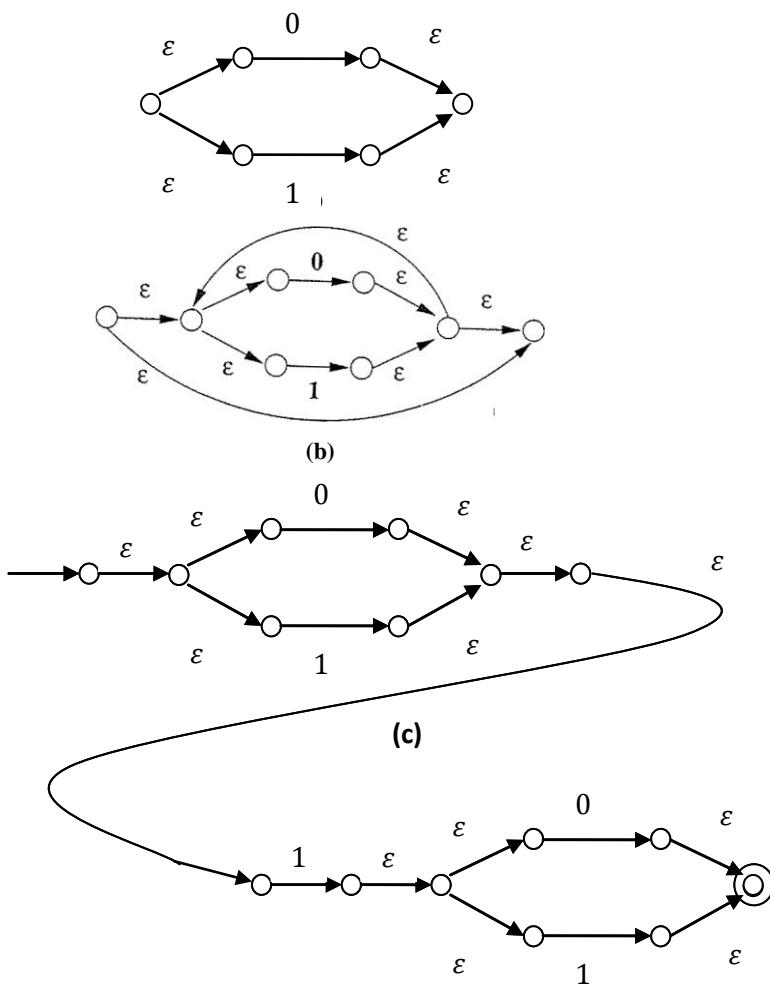
L'espressione è  $(E)$  per un'espressione più piccola  $E$ . Dato che le parentesi non cambiano il linguaggio definito dall'espressione, l'automa per  $R$  serve anche come automa per  $(R)$ .

**Un Esempio:** Convertiamo l'espressione regolare  $(0+1)^*1(0+1)$  in un  $\epsilon$ -NFA. Il primo passo è costruire un automa per  $0 + 1$ . Usiamo due automi costruiti secondo la base della dimostrazione precedente, uno con

etichetta 0 sull'arco e uno con etichetta 1. Questi due automi vengono poi combinati usando la costruzione per l'unione del caso 1 del passo induttivo. Il risultato è rappresentato nella Figura (a).

In un secondo passo applichiamo alla Figura (a) la costruzione per la chiusura vista nel caso 3 del passo induttivo. Il risultato è illustrato dalla (b). Gli ultimi due passi comportano l'applicazione della costruzione per la concatenazione, caso 2 del passo induttivo. In primo luogo connettiamo l'automa della Figura (b) all'automa destinato ad accettare solamente la stringa 1. Tale automa è un'altra applicazione della costruzione base, l'automa  $L(A_1) = \{ a \}$  con etichetta 1 sull'arco. Si noti che dobbiamo creare un *nuovo* automa per riconoscere 1; non possiamo usare l'automa che riconosce 1, già parte della Figura (a). Il terzo automa nella concatenazione è di nuovo un automa per  $0 + 1$ . Ancora una volta dobbiamo creare una copia dell'automa della Figura (a): non possiamo usare la copia che è diventata parte della Figura (b).

L'automa completo è rappresentato nella Figura (c).



## CAPITOLO 5 - GRAMMATICHE E LINGUAGGI CONTEXT FREE

Cominciamo col presentare informalmente la notazione delle grammatiche libere dal contesto, attraverso un esempio. Consideriamo il linguaggio delle parole palindrome. Una stringa è *palindroma* se si può leggere indifferentemente da sinistra a destra e da destra a sinistra, come per esempio otto o madamimadam ("Madam, I'm Adam", la prima frase che si suppone Eva abbia udito nel Giardino dell'Eden).

In altri termini una stringa  $w$  è palindroma se e solo se  $L_{PAL} = \{w \in \{0,1\}^* \mid w = w^R\}$  dove  $w^R$  reverse di  $w$ , ( $abb^R = bba$ ).

### Def(ricorsiva) di reverse:

(base):

- $0^R = 0$ ;
- $1^R = 1$
- $\epsilon^R = \epsilon$

(passo induttivo):

- $w = aw'$ ;
- $w^R = (aw')^R = (w')^Ra, \forall a \in \{0,1\}, w' \in \{0,1\}^*$

Per semplificare le cose descrivremo solo le stringhe palindrome sull'alfabeto  $\{0, 1\}$ ; questo linguaggio comprende stringhe come  $0110, 1101, 1$  e  $\epsilon$ , ma non  $011$  o  $0101$ .

È facile verificare che il linguaggio  $L_{pal}$  delle palindrome di 0 e 1 non è un linguaggio regolare. Per farlo ci serviamo del *pumping lemma*. Se  $L_{pal}$  è regolare, sia  $n$  la costante associata, e consideriamo la stringa palindroma  $w = 0^n 10^n$ . Per il lemma possiamo scomporre  $w$  in  $w = xyz$ , in modo tale che  $y$  consista di uno o più 0 dal primo gruppo. Di conseguenza  $xz$ , che dovrebbe trovarsi in  $L_{pal}$  se  $L_{pal}$  fosse regolare, avrebbe meno 0 a sinistra dell'unico 1 rispetto a quelli a destra. Dunque  $xz$  non può essere palindroma. Abbiamo così confutato l'ipotesi che  $L_{pal}$  non sia un linguaggio regolare.

Per stabilire in quali casi una stringa di 0 e 1 si trova in  $L_{pal}$  possiamo avvalerci di una semplice definizione ricorsiva.

La base della definizione stabilisce che alcune stringhe semplici si trovano in  $L_{pal}$ ; si sfrutta poi il fatto che se una stringa è palindroma deve cominciare e finire con lo stesso simbolo. Inoltre, quando il primo e l'ultimo simbolo vengono rimossi, la stringa risultante deve essere palindroma.

### Def. ricorsiva

(BASE)  $\epsilon, 0$  e  $1$  sono palindrome.

(INDUZIONE) Se  $w$  è palindroma. Io sono anche  $0w0$  e  $1w1$ .

$w \in \{0,1\}^*$  è palindroma se e solo se  $w \in \{0,1,\epsilon\}$  oppure  $w = 0w0$  e  $w = 1w1$ .

Una grammatica libera dal contesto è una notazione formale per esprimere simili definizioni ricorsive di linguaggi. Una grammatica consiste di una o più variabili che rappresentano classi di stringhe, ossia linguaggi. Nell'esempio ci occorre una sola variabile,  $P$ , che rappresenta l'insieme delle palindrome, cioè la classe delle stringhe che formano il linguaggio  $L_{pal}$ . Opportune regole stabiliscono come costruire le stringhe in ogni classe. La costruzione può impiegare simboli dell'alfabeto, stringhe di cui si conosce già l'appartenenza a una delle classi, oppure entrambi gli elementi.

Le regole che definiscono le palindrome, espresse nella notazione delle grammatiche libere dal contesto, sono riportate di seguito:

1.  $P \rightarrow \epsilon$
2.  $P \rightarrow 0$
3.  $P \rightarrow 1$
4.  $P \rightarrow 0P0$
5.  $P \rightarrow 1P1$

Le prime tre regole costituiscono la base e indicano che la classe delle palindrome include le stringhe  $\epsilon$ , 0 e 1. Nessuno dei membri destri delle regole (le porzioni che seguono le frecce) contiene una variabile; questo è il motivo per cui formano la base della definizione.

Le ultime due regole costituiscono la parte induttiva. Per esempio la regola 4 dice che, se si prende una qualunque stringa  $w$  dalla classe  $P$ , anche  $0w0$  si trova nella classe  $P$ . Analogamente la regola 5 indica che anche  $1w1$  è in  $P$ .

## Grammatiche

**(DEFINIZIONE)** Una **grammatica Context-Free**  $G$  è una quadrupla  $G = (V, T, P, S)$  dove:

- $V$  è l'insieme *finito* delle variabili
- $T$  è l'insieme *finito* dei terminali,  $T \cap V = \emptyset$
- $S$  è un elemento di  $V$  (start simbol)

$P \subseteq V \times (V \cup T)^*$ ,  $P$  insieme finito,  $P$  produzione di  $G$ . Un elemento di  $P$  è rappresentato da  $A \rightarrow \alpha$ ,  $A \in V$ ,  $\alpha \in (V \cup T)^*$ .

**Esempio:**  $G_{PAL} = \{V, T, P, S\}$  dove:

$$\begin{aligned} V &= \{S\} \\ T &= \{0, 1, \epsilon\} \end{aligned}$$

$$P = \{P \rightarrow \epsilon, P \rightarrow 0, P \rightarrow 1, P \rightarrow 0P0, P \rightarrow 1P1\}$$

**(DEFINIZIONE DEVIAZIONE DIRETTA)** Sia  $G = (V, T, P, S)$  una grammatica C.F., siano  $\alpha, \beta \in (V \cup T)^*$ . Diremo che si deriva direttamente  $\beta$  in  $G$  e scriveremo  $\alpha \Rightarrow \beta$  se esistono  $\gamma_1, \gamma_2 \in (V \cup T)^*$  e  $A \in P$  tale che  $\alpha \Rightarrow \gamma_1 A \gamma_2$  e  $\beta \Rightarrow \gamma_1 \gamma_2$ .

### Esempio

Esaminiamo una CFG che rappresenta, semplificandole, le espressioni in un tipico linguaggio di programmazione. Dapprima ci limitiamo agli operatori + e \*, corrispondenti a somma e moltiplicazione. Ammettiamo che gli operandi siano identificatori, ma al posto dell'insieme completo di identificatori tipici (una lettera seguita da zero o più lettere e cifre) accettiamo solo le lettere a e b e le cifre 0 e 1. Ogni identificatore deve iniziare per a o b e può continuare con una qualunque stringa in {a, b, 0, 1}\*. In questa grammatica sono necessarie due variabili. La prima, che chiameremo  $E$ , rappresenta le espressioni. È il simbolo iniziale e rappresenta il linguaggio delle espressioni che stiamo definendo. L'altra variabile,  $I$ , rappresenta gli identificatori. Il suo linguaggio è regolare; è il linguaggio dell'espressione regolare

$$(a+b)(a+b+0+1)^*$$

Eviteremo di usare direttamente le espressioni regolari nelle grammatiche, preferendo piuttosto un insieme di produzioni che abbiano lo stesso significato dell'espressione in esame.

La grammatica per le espressioni è definita formalmente da  $G = (\{E,I\}, T, P, E)$ , dove  $T$  è l'insieme di simboli  $\{+, *, (, ), a, b, 0, 1\}$  e  $P$  è l'insieme di produzioni nella Figura 5.2. L'interpretazione delle produzioni è la seguente.

1.  $E \rightarrow I$
2.  $E \rightarrow E + E$
3.  $E \rightarrow E * E$
4.  $E \rightarrow (E)$
5.  $I \rightarrow a$
6.  $I \rightarrow b$
7.  $I \rightarrow Ia$
8.  $I \rightarrow Ib$
9.  $I \rightarrow I0$
10.  $I \rightarrow I1$

Figura 5.2 Una grammatica libera dal contesto per espressioni semplici.

La regola (1) è la regola di base per le espressioni, e afferma che un'espressione può essere un singolo identificatore. Le regole dalla (2) alla (4) descrivono il caso induttivo per le espressioni. La regola (2) afferma che un'espressione può essere formata da due espressioni connesse dal segno  $+$ ; la regola (3) dice la stessa cosa per il segno di moltiplicazione. La regola (4) dichiara che, se si prende una qualunque espressione e la si racchiude fra parentesi, il risultato è ancora un'espressione.

Le regole dalla (5) alla (10) descrivono gli identificatori. La base è data dalle regole (5) e (6), secondo le quali  $a$  e  $b$  sono identificatori. Le rimanenti quattro regole sono il caso induttivo, e affermano che se abbiamo un identificatore possiamo farlo seguire da  $a$ ,  $b$ ,  $0$  oppure  $1$ , e il risultato è comunque un altro identificatore.

### Derivazioni per mezzo di una grammatica

Supponiamo che  $G = (V, T, P, S)$  sia una grammatica CF. Sia  $\alpha A\beta$  una stringa di terminali e variabili, dove  $A$  è una variabile. In altre parole  $\alpha$  e  $\beta$  sono stringhe in  $(V \cup T)^*$ , e  $A$  è in  $V$ .

Sia  $A \rightarrow \gamma$  una produzione di  $G$ . Allora scriviamo  $\alpha A\beta \Rightarrow \alpha\gamma\beta$

Si noti che un passo di derivazione sostituisce una variabile in un punto qualsiasi della stringa con il corpo di una delle sue produzioni.

Possiamo estendere la relazione  $\Rightarrow$  fino a farla rappresentare zero, uno o più passi di derivazione, analogamente a come la funzione di transizione  $\delta$  di un automa a stati finiti si estende a  $\hat{\delta}$ . Per le derivazioni usiamo il simbolo  $\xrightarrow{*}$  per denotare "zero o più passi", come segue:

**(BASE):** Per qualsiasi stringa  $\alpha$  di terminali e variabili,  $\alpha \xrightarrow{*} \alpha$ . In altri termini: qualunque stringa deriva se stessa.

**(INDUZIONE):** Se  $\alpha \xrightarrow{*} \beta$  e  $\beta \Rightarrow \gamma$ , allora  $\alpha \xrightarrow{*} \gamma$  con  $\alpha, \beta, \gamma \in (V \cup T)^*$ . Ossia, se  $\alpha$  può diventare  $\beta$  in zero o più passi, e un passo ulteriore trasforma  $\beta$  in  $\gamma$ , allora  $\alpha$  può diventare  $\gamma$ .

Detto in altri termini,  $\alpha \xrightarrow{*} \beta$  significa che esiste una sequenza di stringhe  $\gamma_1, \gamma_2, \dots, \gamma_n$  con  $n \geq 1$ , tale che:

1.  $\alpha = \gamma_1;$
2.  $\beta = \gamma_n;$
3. per  $i = 1, 2, \dots, n-1$ , abbiamo  $\gamma_i \Rightarrow \gamma_{i+1}$ .

## Derivazioni a sinistra e a destra

Per ridurre il numero di scelte possibili nella derivazione di una stringa, spesso è comodo imporre che a ogni passo si sostituisca la variabile all'estrema sinistra con il corpo di una delle sue produzioni. Tale derivazione viene detta *derivazione a sinistra (leftmost derivation)*, e si indica tramite le relazioni  $\xrightarrow{lm}$  e  $\xrightarrow{lm}^*$ , rispettivamente per uno o molti passi. Se la grammatica  $G$  in esame non è chiara dal contesto, possiamo porre il nome  $G$  sotto la freccia.

Analogamente è possibile imporre che a ogni passo venga sostituita la variabile più a destra con uno dei suoi corpi. In tal caso chiamiamo la derivazione *a destra (rightmost)*, e usiamo i simboli  $\xrightarrow{rm}$  e  $\xrightarrow{rm}^*$  per indicare rispettivamente uno o più passi di derivazione a destra. Qualora non fosse evidente, il nome della grammatica può comparire anche in questo caso sotto i simboli.

Esempio. Derivazione a sinistra della stringa  $a^*(a+b00)$ ,

$$\begin{aligned} E &\xrightarrow{lm} E * E \xrightarrow{lm} I * E \xrightarrow{lm} a * E \xrightarrow{lm} a * (E) \xrightarrow{lm} a * (E + E) \xrightarrow{lm} a * (I + E) \xrightarrow{lm} a * (a + E) \xrightarrow{lm} a * (a + I) \xrightarrow{lm} \\ &\quad \xrightarrow{lm} a * (a + I0) \xrightarrow{lm} a * (a + I00) \xrightarrow{lm} a * (a + b00) \end{aligned}$$

Possiamo inoltre riassumere la derivazione a sinistra scrivendo  $E \xrightarrow{lm} a * (a + b00)$ , oppure esprimerne alcuni passi tramite espressioni come  $E * E \xrightarrow{lm} a * (E)$ .

Esiste una derivazione a destra che applica le stesse sostituzioni per ogni variabile, sebbene in ordine diverso. Eccola:

$$\begin{aligned} E &\xrightarrow{rm} E * E \xrightarrow{rm} E * (E) \xrightarrow{rm} E * (E + E) \xrightarrow{rm} E * (E + I) \xrightarrow{rm} E * (E + I0) \xrightarrow{rm} E * (E + I00) \xrightarrow{rm} \\ &\quad \xrightarrow{rm} E * (E + b00) \xrightarrow{rm} E * (I + b00) \xrightarrow{rm} E * (a + b00) \xrightarrow{rm} I * (a + b00) \xrightarrow{rm} a * (a + b00) \end{aligned}$$

Questa derivazione permette di concludere  $E \xrightarrow{rm} a * (a + b00)$ .

**(DEFINIZIONE)** Sia  $G = (V, T, P, S)$  una grammatica C.F.. Il linguaggio  $L(G)$  generato dalla grammatica  $G$  è definito da:

$$L(G) = \{ w \in T^* \mid S \xrightarrow{*} w \}$$

**(DEFINIZIONE)** Un linguaggio  $L \subseteq T^*$  è context-free se esiste una grammatica  $G$  context-free tale che  $L=L(G)$

**(DEFINIZIONE)** Se  $\alpha \in (V \cup T)^*$  e  $S \xrightarrow{*} \alpha$  (in  $G = (V, T, P, S)$ ) allora  $\alpha$  è una forma sentenziale (sinistra\destra)

## Proprietà delle grammatiche

### Proprietà 1

Sia  $G = (V, T, P, S)$  una grammatica C.F., siano  $A \in V$ ,  $\alpha \in (V \cup T)^*$ . Se  $A \xrightarrow{*} \alpha$ ,  $\forall \gamma_1, \gamma_2 \in (V \cup T)^*$  risulta  $\gamma_1 A \gamma_2 \Rightarrow \gamma_1 \alpha \gamma_2$ .

Esempio:

$$\begin{array}{ll} E \xrightarrow{*} ab & E \Rightarrow I \Rightarrow Ib \Rightarrow ab \\ E + (E + E) \xrightarrow{*} E + (E + ab) & \end{array}$$

### Proprietà 2

Sia  $G = (V, T, P, S)$  una grammatica C.F., siano  $x_1, \dots, x_k \in (V \cup T)$ , sia  $w \in T^*$ . Se  $x_1 \dots x_k \xrightarrow{*} w$  (in  $n$  passi) allora esistono  $w_1, \dots, w_k \in T^*$  tali che  $w = w_1 \dots w_k$ ,  $\forall j \in \{1, \dots, k\}$   $x_j \xrightarrow{*} w_j$  (in  $n_j$  passi,  $n_j \leq n$ ) e se  $x_j \in T$  allora  $x_j = w_j$ .

Esempio:

$$L_{pal} = \{ w \in \{0, 1\}^* \mid w = w^R \}$$

$$G_{pal} = (V, T, P, S) \quad V = \{S\}, T = \{0, 1\}, P = \{S \rightarrow 0|1|\varepsilon|0S0|1S1\}$$

$$01S10 \xrightarrow{*} 01011010$$

$$x_1 = 0, x_2 = 1, x_3 = S, x_4 = 1, x_5 = 0$$

$$w_1 = 0, w_2 = 1, w_3 = 0110, w_4 = 1, w_5 = 0$$

$$S = x_3 \xrightarrow{*} 0110$$

**Esercizio:** Dimostrare che  $L(G_{pal}) = L_{pal}$

Dim.: Dimostriamo che  $L_{pal} \subseteq L(G_{pal})$ . Devo dimostrare che  $\forall j \in L_{pal}$  risulta  $S \xrightarrow{*} w$  e lo dimostro per induzione su  $|w|$ .

(BASE) Sia  $w = 0$  oppure  $w = 1$  oppure  $w = \varepsilon$ . Risulta  $S \xrightarrow{*} 0, S \xrightarrow{*} 1, S \xrightarrow{*} \varepsilon$  e quindi la base è provata

(PASSO INDUTTIVO) Sia  $w \in L_{pal}$  e  $|w| > 1$ . Allora per definizione  $w = 1x1$  dove  $x \in L_{pal}$  e  $|x| < |w|$ . Per ipotesi induttiva  $S \xrightarrow{*} x$ . Se  $w = 0x0$  risulta, per la proprietà 1,  $S = 0S0 \xrightarrow{*} 0x0 = w$ . Se  $w = 1x1$  risulta, per la proprietà 1,  $S = 1S1 \xrightarrow{*} 1x1 = w$ . Quindi  $L_{pal} \subseteq L(G_{pal})$ .

Viceversa, dimostriamo che Quindi  $L(G_{pal}) \subseteq L_{pal}$ , cioè  $\forall w \in L(G_{pal}), w \in L_{pal}$ , ovvero  $\forall w$  tale che  $S \xrightarrow{*} w$  si ha che  $w \in L_{pal}$  e di mostro [ $S \xrightarrow{*} w$  implica  $w \in L_{pal}$ ] per induzione sul numero dei passi della derivazione di  $w$  in  $S$ .

(BASE) Se  $S \xrightarrow{*} w$  in un passo cioè  $S \Rightarrow w$  e quindi, per derivazione,  $w = 0$  oppure  $w = 1$  oppure  $w = \varepsilon$ . Quindi  $w \in L_{pal}$ . La base è dimostrata.

(PASSO INDUTTIVO)  $S \xrightarrow{*} w$  in  $n$  passi con  $n > 1$ . Allora  $S \Rightarrow 0S0 \xrightarrow{*} w$  oppure  $S \Rightarrow 1S1 \xrightarrow{*} w$ . Applichiamo la proprietà 2, risulta  $w = 0x0$  oppure  $w = 1x1$  con  $S \xrightarrow{*} x$  in  $n'$  passi con  $n' \leq n$  e  $x \in T^*$ . Per ipotesi induttiva  $x \in L_{pal}$  e quindi per definizione  $w \in L_{pal}$ . Quindi  $L_{pal} = L(G_{pal})$ .

## Albero sintattico

La rappresentazione ad albero delle derivazioni si è rivelata particolarmente utile. L'albero mostra in modo chiaro come i simboli di una stringa terminale sono raccolti in sottostringhe, ciascuna appartenente al linguaggio di una delle variabili della grammatica. Forse è ancora più importante che un albero di questo tipo, detto "albero sintattico" (parse tree), sia la struttura dati ideale per rappresentare il programma sorgente in un compilatore. La struttura ad albero del programma sorgente facilita la traduzione del programma stesso in codice eseguibile, delegando in modo naturale il processo di traduzione a funzioni ricorsive. In questo paragrafo presentiamo gli alberi sintattici e dimostriamo che la loro esistenza è strettamente legata alle derivazioni e alle inferenze ricorsive. Studieremo poi la questione dell'ambiguità nelle grammatiche e nei linguaggi, che costituisce un'applicazione importante degli alberi sintattici. In alcune grammatiche una stringa terminale può avere più di un albero sintattico; ciò rende una grammatica inadatta a un linguaggio di programmazione perché il compilatore non sarebbe in grado di stabilire la struttura di certi programmi, e quindi non potrebbe dedurre con sicurezza il codice eseguibile appropriato.

## Costruzione di alberi sintattici

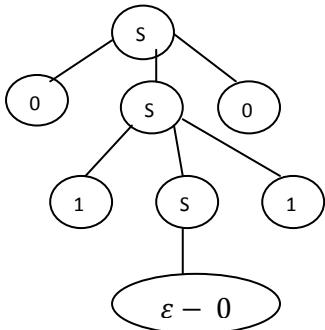
Fissiamo una grammatica  $G = (V, T, P, S)$ . Gli alberi sintattici di  $G$  sono alberi che soddisfano le seguenti condizioni.

1. Ciascun nodo interno è etichettato da una variabile in  $V$ .
2. Ciascuna foglia è etichettata da una variabile, da un terminale, o da  $\varepsilon$ . Se una foglia è etichettata  $\varepsilon$ , deve essere l'unico figlio del suo genitore.
3. Se un nodo interno è etichettato  $A$  e i suoi figli sono etichettati, a partire da sinistra,  $X_1, X_2, \dots, X_k$  allora  $A \rightarrow X_1, X_2, \dots, X_k$ , è una produzione in  $P$ . Si noti che un  $X$  può essere  $\varepsilon$  solo nel caso in cui è l'etichetta di un figlio unico, e quindi  $A \rightarrow \varepsilon$  è una produzione di  $G$ .

### Definizione: Frontiera

La frontiera di un albero sintattico è uguale alla concatenazione delle etichette associate alle foglie dell'albero T (lette da sinistra a destra). La frontiera è un elemento di  $(V \cup T)^*$ .

$$G_{PAL} = (\{S\}, \{0,1\}, S, \{S \rightarrow 0|1|\varepsilon|0S0|1S1\})$$

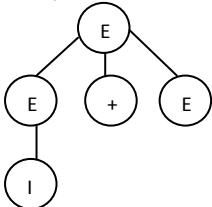


La frontiera è 01010. Se ci fosse (nel l'ultimo cerchio in basso)  $\varepsilon$  allora la frontiera sarebbe 0110.

Esempio: La radice è etichettata dalla variabile  $E$ . La produzione applicata alla radice è  $E \rightarrow E + E$ : i tre figli della radice hanno, a partire da sinistra, le etichette  $E$ ,  $+$ , ed  $E$ . Per il figlio più a sinistra della radice si applica la produzione  $E \rightarrow I$ : il nodo ha un solo figlio, etichettato  $I$

$$G = (\{E, I\}, \{0, 1, a, b, +, *, (, )\}, E, P)$$

$$P = \{I \rightarrow a|b|I0|I1, E \rightarrow E + E|E * E|(E)|I\}$$



$I+E$  è la frontiera

$$E \stackrel{*}{\Rightarrow} I + E \text{ ovvero } E \Rightarrow E + E \Rightarrow I + E$$

### Il Prodotto di un albero sintattico

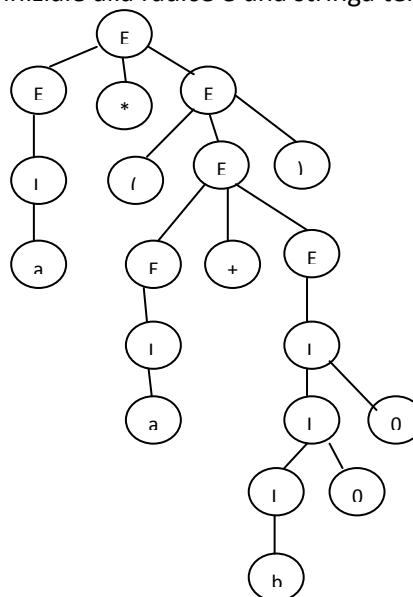
Se concateniamo le foglie di un albero sintattico a partire da sinistra otteniamo una stringa, detta il *prodotto* dell'albero, che è sempre una stringa derivata dalla variabile della radice. Dimostreremo questa asserzione fra poco. Di particolare importanza sono gli alberi sintattici che soddisfano queste due condizioni.

1. Il prodotto è una stringa terminale. In questo caso tutte le foglie sono etichettate da un terminale o da  $\varepsilon$ .
2. La radice è etichettata dal simbolo iniziale.

Questi sono gli alberi sintattici i cui prodotti sono stringhe nel linguaggio della grammatica associata. Fra breve dimostreremo che si può descrivere il linguaggio di una grammatica anche come insieme dei prodotti degli alberi sintattici che hanno il simbolo iniziale alla radice e una stringa terminale come prodotto.

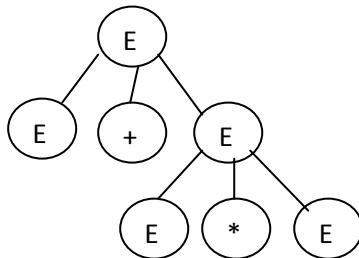
Esempio:

$$E \stackrel{*}{\Rightarrow} a * (a + b00)$$

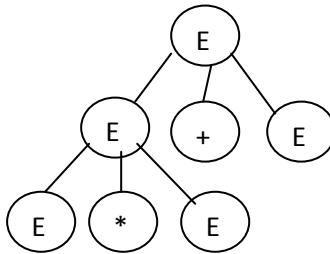


La frontiera è  $a^*(a+b00)$

$$E \Rightarrow E + E \Rightarrow E + E * E$$



Ma si può avere anche:



In questo modo si avrebbero problemi in pratica in quanto si raggiungerebbe una stessa risultato in due modi differenti. Le grammatiche all'interno dei compilatori sono specificate in modo che evitino questo problema.

#### TEOREMA:

Sia  $G=(V,T,P,S)$  una grammatica context-free. Sono equivalenti le affermazioni seguenti:

1.  $A \xrightarrow{*} w \quad A \in V, w \in T^*$
2.  $\exists$  un parse tree (albero sintattico) di radice  $A$  e frontiera  $w, w \in T^*$
3.  $A \xrightarrow{LM}^* w, w \in T^*$

#### DIMOSTRAZIONE:

1)  $\Rightarrow$  2) : la 1) implica la 2) ovvero (se 1 allora 2).

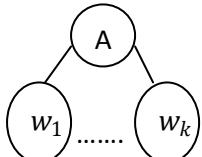
La dimostrazione viene effettuata per induzione. (enunciato: la parola  $w$  si può derivare e vogliamo dimostrare che esiste il parse tree).

Supponiamo che  $A \xrightarrow{*} w$ , con  $w \in T^*, A \in V$ . Dimostriamo che esiste un albero sintattico di radice  $A$  e frontiera  $w$  per induzione sul numero di passi nella derivazione di  $w$  da  $A \xrightarrow{*} w$ .

(BASE)

Supponiamo  $A \Rightarrow w$  e dimostriamo che esiste l'albero sintattico di radice  $A$  e frontiera  $w$ .

Siccome  $A \Rightarrow w$ , per definizione  $\exists$  la produzione ( $P$ )  $A \rightarrow w_1, w_2, \dots, w_k = w, w \in P \quad w_j \in T$  e



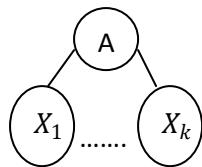
è un albero sintattico la cui frontiera è  $w$ .

#### (PASSO INDUTTIVO)

Supponiamo  $A \xrightarrow{*} w \quad w \in T^*$  in  $n$  passi con  $n > 1$  e assumiamo vero l'enunciato per derivazione in numero di passi  $<$  di  $n$ .

Siccome  $A \xrightarrow{*} w$  in  $n$  passi con  $n > 1$ .

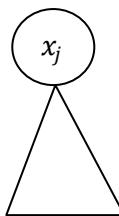
$A \Rightarrow X_1, X_2, \dots, X_k \xrightarrow{*} w$  con  $A \rightarrow X_1, X_2, \dots, X_k \in P \quad X_j \in T \cup V$



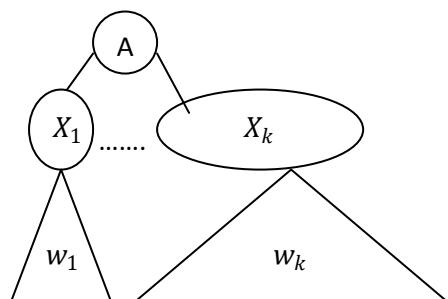
Applichiamo la proprietà 2) delle grammatiche context-free.

Quindi  $w = w_1, w_2, \dots, w_k$  dove  $x_j \Rightarrow w_j$  in  $n' < n$  passi con  $1 \leq j \leq k$  e se  $X_j \in T$  allora  $X_j = w_j$

Dimostrazione 1)  $\Rightarrow$  2) nel caso se  $X_j \notin T$ , per ipotesi induttiva



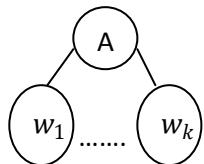
Quindi:



E ho trovato un parse tree di radice A e frontiera w. Quindi ho dimostrato 1)  $\Rightarrow$  2)

**Dimostriamo** che 2)  $\Rightarrow$  3) per induzione sull'altezza dell'albero sintattico.

(BASE) Sia  $T$  un albero sintattico di radice A, frontiera  $w \in T^*$ , e altezza 1.

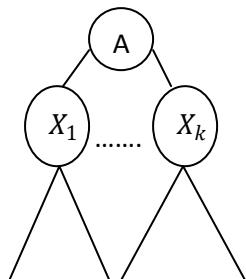


con  $w_j \in T$  e  $w_1, w_2, \dots, w_k = w$

Per definizione di parse tree  $A \rightarrow w_1, w_2, \dots, w_k \in P$ . Quindi  $A \Rightarrow w_1, w_2, \dots, w_k$  cioè  $A \xrightarrow{LM}^* w_1, w_2, \dots, w_k = w$

(PASSO INDUTTIVO)

$\exists$  un parse tree di radice A, frontiera  $w \in T^*$  di altezza  $n > 1$



Siccome è un parse tree allora  $A \rightarrow X_1, X_2, \dots, X_k \in P$ .

Inoltre se  $w_j \neq x_j$  allora  $w_j$  è frontiera di un parse tree di radice  $x_j$  e di altezza minore di n.

Per ipotesi induttiva  $x_j \xrightarrow{LM}^{*(\text{proprietà 1})} w_j$ .

Siccome  $A \rightarrow X_1, X_2, \dots, X_k \in P$  allora  $A \xrightarrow{LM} X_1, X_2, \dots, X_k \xrightarrow{LM} w_1 x_2 \dots x_k \xrightarrow{LM}^* \dots \xrightarrow{LM} w$  (per induzione, usando la (1)).

Quindi  $A \xrightarrow[LM]^* w$  che dimostra anche 3)  $\Rightarrow 1$ .

Definizione:

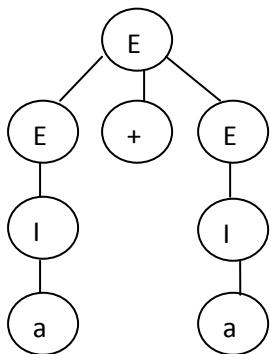
Una grammatica context-free  $G = (V, T, P, S)$  è ambigua se esiste  $w \in T^*$  e due alberi sintattici di radice  $S$  e frontiera  $w$ .

(Proprietà)

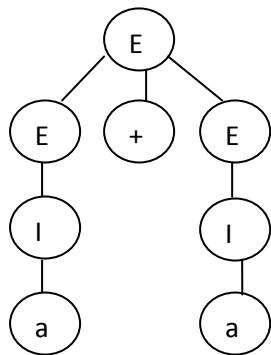
Una grammatica context-free  $G = (V, T, P, S)$  è ambigua se e solo se una stringa di terminali ( $w \in T^*$ ) che permette due o più derivazioni *leftmost* da  $S$ .

$$\begin{aligned} E &\xrightarrow[LM]{} E + E \xrightarrow[LM]{} I + E \xrightarrow[LM]{} a + E \xrightarrow[LM]{} a + I \xrightarrow[LM]{} a + a \\ E &\xrightarrow[RM]{} E + E \xrightarrow[RM]{} E + I \xrightarrow[RM]{} E + A \xrightarrow[RM]{} I + a \xrightarrow[RM]{} a + a \end{aligned}$$

- LM



- RM



**ESERCIZI DI ESAME**

Teoria della Computazione (Prof.ssa De Felice)

2004/05/06

Prove scritte appelli precedenti - 2004/05/06

**AUTOMI/ESPRESSIONI REGOLARI**

1. Sia  $L \subseteq \Sigma^*$ . Dimostrare che se  $L$  è un linguaggio di cardinalità finita allora  $M = \{x \mid \exists n \in \mathbb{N}, \exists y \in L : y = x^n\}$  è un linguaggio regolare. Dimostrare che se  $L$  è un linguaggio regolare allora  $M' = \{x \mid \exists a \in \Sigma, \exists y \in L : x = ay\}$  è un linguaggio regolare.
2. Sia  $\Sigma = \{0, 1\}$ . Per ogni  $w \in \Sigma^*$ , si consideri il linguaggio  $Pref(w) = \{wx \mid x \in \Sigma^*\}$  delle parole su  $\Sigma$  che iniziano per  $w$ .
  - a) Disegnare il grafo delle transizioni di un automa finito *deterministico* che riconosce  $Pref(w)$  e che abbia al più  $|w| + 1$  stati. (Lo stato "pozzo" può essere ignorato.)
  - b) Dimostrare che non esiste un automa finito deterministico che riconosce  $Pref(w)$  e che abbia un numero di stati minore di  $|w| + 1$ .
3. Per ogni  $w \in \{a, b\}^*$  denotiamo con  $|w|_a$  il numero di occorrenze della lettera  $a$  in  $w$  e con  $|w|_b$  il numero di occorrenze della lettera  $b$  in  $w$ .
  - 1) Definire un automa deterministico  $A$  con due stati (escludendo un eventuale stato trappola) il cui linguaggio accettato sia  $L(A) = \{w \in \{a, b\}^* \mid |w|_a \neq |w|_b \pmod{2}\}$ .
  - 2) Definire un automa deterministico  $A$  con due stati (escludendo un eventuale stato trappola) il cui linguaggio accettato sia  $L(A) = \{w \in \{a, b\}^* \mid \text{in } w \text{ ogni occorrenza di } a \text{ è seguita da un'occorrenza di } b\}$ .
4. Sia  $\Sigma = \{a, b\}$  e, per ogni  $w \in \{a, b\}^*$ , denotiamo con  $|w|_a$  il numero di occorrenze della lettera  $a$  in  $w$  e con  $|w|_b$  il numero di occorrenze della lettera  $b$  in  $w$ . Dire se la seguente uguaglianza è vera o falsa. È necessario giustificare formalmente la risposta data. Risposte non giustificate saranno valutate zero.
 
$$\{xy \mid x, y \in \Sigma^*, |x|_a = |y|_b\} = \Sigma^*.$$

5. Si consideri l'operazione che ai linguaggi  $A$  e  $B$  associa  $A^{-1}B = \{y \mid \exists x \in A : xy \in B\}$  e quella che a essi associa  $A_{-1}B = \{x \mid \forall y \in B : xy \in A\}$ . Definire  $A^{-1}B$  quando  $A = a^*$ ,  $B = a^*b$  e dire se  $A^{-1}B$  è un linguaggio regolare. Definire  $A_{-1}B$  quando  $A = B = a^*ba^*$  e dire se  $A_{-1}B$  è un linguaggio regolare. Giustificare le risposte.
6. Sia  $L$  un linguaggio formale sull'alfabeto  $\Sigma$ . Dimostrare l'affermazione seguente:

$$L^* \text{ ha cardinalità infinita} \iff L \neq \emptyset \text{ e } L \neq \{\epsilon\}$$

7. Siano  $A_1 = (Q_1, \Sigma, \delta_1, q_{(0,1)}, F_1)$ ,  $A_2 = (Q_2, \Sigma, \delta_2, q_{(0,2)}, F_2)$  due automi finiti deterministici. Definiamo  $A_{(1,2)} = (Q_1 \times Q_2, \Sigma, q_0, \hat{\delta}, F_1 \times F_2)$ , dove  $\hat{\delta}$  è la funzione da  $(Q_1 \times Q_2) \times \Sigma$  in  $Q_1 \times Q_2$  tale che  $\hat{\delta}((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$  e si è posto  $q_0 = (q_{(0,1)}, q_{(0,2)})$ .

- a) Dimostrare che per ogni  $w \in \Sigma^*$  risulta

$$\hat{\delta}((q_{(0,1)}, q_{(0,2)}), w) = (\hat{\delta}_1(q_{(0,1)}, w), \hat{\delta}_2(q_{(0,2)}, w)).$$

8. Sia  $\Sigma = \{0, 1\}$ . Per ogni  $w \in \Sigma^*$ , si consideri il linguaggio  $Suf(w) = \{xw \mid x \in \Sigma^*\}$  delle parole su  $\Sigma$  che terminano per  $w$ .
  - a) Disegnare il grafo delle transizioni di un automa finito *non deterministico* che riconosce  $Suf(w)$  e che abbia al più  $|w| + 1$  stati.
  - b) Disegnare il grafo delle transizioni di un automa finito *deterministico* che riconosce  $Suf(w)$  e che abbia al più  $|w| + 1$  stati.

- c) Dimostrare che non esiste un automa finito deterministico che riconosce  $Suf(w)$  e che abbia un numero di stati minore di  $|w| + 1$ .
9. Dato l'automa non deterministico  $A$  la cui tavola delle transizioni è riportata di seguito, disegnare il diagramma delle transizioni di  $A$  e determinare il linguaggio  $L(A)$  riconosciuto da  $A$ .

	$a$	$b$
$\rightarrow q_0$	$q_1$	$q_2$
$q_1$	$\emptyset$	$\{q_0, q_1\}$
$* q_2$	$q_2$	$\emptyset$

10. Determinare il diagramma delle transizioni e il linguaggio  $L(A)$  riconosciuto dall'automa non deterministico  $A$  la cui tavola delle transizioni è riportata di seguito.

	$a$	$b$
$\rightarrow q_0$	$q_1$	$q_1$
$q_1$	$q_0$	$q_2$
$* q_2$	$\emptyset$	$\emptyset$

11. Dimostrare che se  $L$  è un linguaggio di cardinalità finita allora  $M = \{x \mid \exists n \in \mathbb{N}, \exists y \in L : x = y^n\}$  è un linguaggio regolare.
12. Definire un automa deterministico  $A$  il cui linguaggio accettato sia il linguaggio definito dall'espressione regolare  $E = (000)^*1 + (00)^*1$  (cioè tale che  $L(A) = L(E)$ ).
13. Determinare il diagramma delle transizioni e il linguaggio  $L(A)$  riconosciuto dall'automa non deterministico  $A$  la cui tavola delle transizioni è riportata di seguito.

	$a$	$b$
$\rightarrow q_0$	$\emptyset$	$q_1$
$* q_1$	$q_2$	$\emptyset$
$q_2$	$q_1$	$q_1$

14. Definire un automa deterministico  $A$  il cui linguaggio accettato sia il linguaggio definito dall'espressione regolare  $E = (bb)^*c + (ba)^*c$  (cioè tale che  $L(A) = L(E)$ ).
15. Determinare il diagramma delle transizioni e il linguaggio  $L(A)$  riconosciuto dall'automa non deterministico  $A$  la cui tavola delle transizioni è riportata di seguito.

	$a$	$b$
$\rightarrow q_0$	$\emptyset$	$q_1$
$q_1$	$q_2$	$q_4$
$q_2$	$\emptyset$	$q_3$
$q_3$	$q_1$	$\emptyset$
$* q_4$	$q_4$	$\emptyset$

16. Determinare il diagramma delle transizioni e il linguaggio  $L(A)$  riconosciuto dall'automa non deterministico  $A$  la cui tavola delle transizioni è riportata di seguito.

	$a$	$b$
$\rightarrow q_0$	$q_1$	$q_1$
$q_1$	$\emptyset$	$q_2$
$* q_2$	$q_0$	$q_0$

17. Rispondere alle seguenti domande, giustificando la risposta. Risposte non giustificate saranno valutate zero.
- a)  $\{x x x \mid x \in \{a, b\}^*\}$  è regolare?
- b)  $\{x c y \mid x, y \in \{a, b\}^*\}$  è regolare?
- c)  $\{a^n b^n \mid 0 \leq n \leq 481\}$  è regolare?

18. Definire un automa deterministico  $A$  il cui linguaggio accettato sia il linguaggio definito dall'espressione regolare  $E = (aa)^*b + (ab)^*a$  (cioè tale che  $L(A) = L(E)$ ).

19. Determinare il diagramma delle transizioni e il linguaggio  $L(A)$  riconosciuto dall'automa non deterministico  $A$  la cui tavola delle transizioni è riportata di seguito.

	a	b
$\rightarrow q_0$	$q_1$	$q_0$
$q_1$	$q_2$	$q_0$
$* q_2$	$q_2$	$q_3$
$q_3$	$q_2$	$\emptyset$

20. Definire un automa deterministico  $A$  il cui linguaggio accettato sia il linguaggio definito dall'espressione regolare  $E = 011(00)^* + 01^*$  (cioè tale che  $L(A) = L(E)$ ).

21. Determinare il diagramma delle transizioni e il linguaggio  $L(A)$  riconosciuto dall'automa non deterministico  $A$  la cui tavola delle transizioni è riportata di seguito.

	a	b
$\rightarrow q_0$	$q_2$	$q_1$
$q_1$	$q_1$	$q_0$
$* q_2$	$q_3$	$q_2$
$q_3$	$\emptyset$	$q_2$

22. Definire un automa deterministico  $A$  il cui linguaggio accettato sia il linguaggio definito dall'espressione regolare  $E = ab(aa)^* + (ab)^*$  (cioè tale che  $L(A) = L(E)$ ).

23. Determinare il diagramma delle transizioni e il linguaggio  $L(A)$  riconosciuto dall'automa non deterministico  $A$  la cui tavola delle transizioni è riportata di seguito.

	0	1
$\rightarrow * q_0$	$q_1$	$\emptyset$
$q_1$	$q_0$	$q_2$
$* q_2$	$q_1$	$q_2$

24. Definire un automa deterministico  $A$  il cui linguaggio accettato sia il linguaggio definito dall'espressione regolare  $E = (000^* + 111^*)^*$  (cioè tale che  $L(A) = L(E)$ ).

25. Sia  $L$  il linguaggio delle parole  $w$  in  $\{0,1\}^+$  che hanno 1 in ogni posizione dispari.

- a) Definire un'espressione regolare  $E$  tale che il linguaggio  $L(E)$  associato a  $E$  sia  $L$ , cioè  $L(E) = L$ .
- b) Definire un automa finito deterministico  $A$  tale che il linguaggio  $L(A)$  associato ad  $A$  sia  $L$ , cioè  $L(A) = L$ .

26. Siano dati i linguaggi  $L = \{0^n1^n \mid n \geq 0\}$ ,  $L^R = \{1^n0^n \mid n \geq 0\}$ . Il linguaggio  $L\{0,1\}^*L^R$  è regolare? Giustificare la risposta (cioè in caso affermativo esibire un automa per il linguaggio, in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

27. Il linguaggio  $L = \{w \in \{0,1\}^* \mid \forall z_1, z, z_2 \in \{0,1\}^*, \text{ se } w = z_1zz_2 \text{ e } |z| = 3 \text{ allora } |z|_0 \leq 1\}$  è regolare? Giustificare la risposta (cioè in caso affermativo esibire un automa per il linguaggio, in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

28. Il linguaggio  $L = \{w \in \{0,1\}^* \mid \exists k \geq 0, x, z \in \{0,1\}^* \text{ tali che } |w| = 2k, w = x11z\}$  è regolare? Giustificare la risposta (cioè in caso affermativo esibire un automa che riconosce  $L$  o un'espressione regolare che rappresenta  $L$ , in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

## ESPRESSIONI REGOLARI

1. Date due espressioni regolari  $E_1$  e  $E_2$ , diciamo che  $E_1 = E_2$  se il linguaggio  $L(E_1)$  associato a  $E_1$  coincide con il linguaggio  $L(E_2)$  associato con  $E_2$ . Inoltre, denotiamo con  $E_1^+$  l'espressione regolare che soddisfa la relazione  $L(E_1^+) = (L(E_1))^+$ . Per ognuna delle seguenti relazioni dire se essa è vera o falsa. È necessario giustificare formalmente la risposta data.

- $b(a^*b)^*a^* = (ba^*)^*$
- $(a+b)^*a^* = (a+b)^*$
- $(b^*a)^+ = (a+b)^*aa$

2. Date due espressioni regolari  $E_1$  e  $E_2$ , diciamo che  $E_1 = E_2$  se il linguaggio  $L(E_1)$  associato a  $E_1$  coincide con il linguaggio  $L(E_2)$  associato con  $E_2$ . Inoltre, denotiamo con  $E_1^+$  l'espressione regolare che soddisfa la relazione  $L(E_1^+) = (L(E_1))^+$ . Per ognuna delle seguenti relazioni dire se essa è vera o falsa. È necessario giustificare formalmente la risposta data.

- $a(b^*a)^*b^* = (ab^*)^+$
- $(a+b)^*(bb)^* = (a+b)^*$
- $(b^*a)^+ = (a+b)^*b^* + b^*$

3. Date due espressioni regolari  $E_1$  e  $E_2$ , diciamo che  $E_1 = E_2$  se il linguaggio  $L(E_1)$  associato a  $E_1$  coincide con il linguaggio  $L(E_2)$  associato con  $E_2$ .

Per ognuna delle seguenti relazioni dire se essa è vera o falsa. È necessario giustificare formalmente la risposta data. Risposte non giustificate saranno valutate zero.

- $(a+b)^* = a^*(b^+a^*)^*$
- $(a+b)^* = a^* + a^*b(a+b)^*$

4. Date due espressioni regolari  $E_1$  e  $E_2$ , diciamo che  $E_1 = E_2$  se il linguaggio  $L(E_1)$  associato a  $E_1$  coincide con il linguaggio  $L(E_2)$  associato con  $E_2$ .

- a) Per ognuna delle seguenti relazioni dire se essa è vera o falsa. È necessario giustificare formalmente la risposta data. Risposte non giustificate saranno valutate zero.

- 1)  $\emptyset^* = \epsilon^*$ ,
- 2)  $(0+1)^* = 0^* + 1^*$ ,
- 3)  $(000^* + 111^*) = (00 + 11)^*$ .

- b) Dimostrare che per ogni intero non negativo  $n$ , risulta  $(01+0)^n 0 = 0(10+0)^n$ . Utilizzare questo risultato per provare che  $(01+0)^* 0 = 0(10+0)^*$ .

5. Sia  $\Sigma$  un alfabeto (finito e non vuoto). Un fattore  $y$  di  $w \in \Sigma^*$  è una parola tale che  $w = xyz$ , con  $x, z \in \Sigma^*$ . Esibire un'espressione regolare  $E$ , utilizzando solo gli operatori di concatenazione e di chiusura di Kleene, tale che  $L(E)$  sia il linguaggio delle parole  $w \in \{a, b\}^*$  che non hanno  $ab$  come fattore.

## PUMPING LEMMA e PROPRIETÀ DI CHIUSURA

1. Indicare quali tra i seguenti linguaggi sono regolari.

- a)  $A = \{0^n 1^m \mid n, m \geq 0\}$ ,
- b)  $B = \{xx \mid x \in A\}$ .

Giustificare le risposte (cioè in caso affermativo esibire un automa o una espressione regolare per il linguaggio, in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

2. Sia  $L = \{0^n 1^n \mid n \geq 0\}$ . Indicare quali tra i seguenti linguaggi sono regolari.

- a)  $0^* L$ ,
- b)  $L 1^*$ ,
- c)  $0^* L \cup L 1^*$ .

Giustificare le risposte (cioè in caso affermativo esibire un automa o una espressione regolare per il linguaggio, in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

3. Sia  $L_d(\Sigma) = \{w \in \Sigma^* \mid \exists k \geq 0 \text{ tale che } |w| = 2k + 1\}$ . Indicare quali tra i seguenti linguaggi sono regolari.

- a)  $L_1 = \{wxw' \mid w, w' \in \{0, 1\}^*, |w| = |w'|, x \in L_d(\{0, 1\})\}$ .
- b)  $L_2 = \{wxw' \mid w, w' \in \{0, 1\}^*, |w| = |w'|, x \in L_d(\{a, b\})\}$ .
- c)  $L_2 \cup L_d(\{a, b\})$ .

Giustificare le risposte (cioè in caso affermativo esibire un automa o una espressione regolare per il linguaggio, in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

4. Sia  $L = \{0^n 1^n \mid n \geq 0\}$ . Indicare quali tra i seguenti linguaggi sono regolari.

- a)  $L_1 = \{0^n 1^m \mid n - m < 5\}$ .
- b)  $L_2 = \{0^n 1^m \mid n + m < 5\}$ .
- c)  $H(L) = \{x \mid \exists y \mid |x| = |y| \text{ e } xy \in L\}$ .

Giustificare le risposte (cioè in caso affermativo esibire un automa o una espressione regolare per il linguaggio, in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

5. È noto che la classe dei linguaggi regolari è chiusa rispetto alle operazioni di unione, intersezione, prodotto, complemento e differenza (cioè se  $L, M$  sono regolari lo sono anche  $L \cup M, L \cap M, LM, \Sigma^* \setminus M, L \setminus M$ ). È anche noto che il linguaggio  $\{0^n 1^n \mid n \geq 0\}$  non è regolare. Utilizzare questi risultati per provare che il linguaggio  $L = \{0, 1\}^* \{10\} \{0, 1\}^* \cup \{0^h 1^k \mid h, k \in \mathbb{N}, h \neq k\}$  non è regolare.

6. È noto che la classe dei linguaggi regolari è chiusa rispetto alle operazioni di unione, intersezione, prodotto, complemento e differenza (cioè se  $L, M$  sono regolari lo sono anche  $L \cup M, L \cap M, LM, \Sigma^* \setminus M, L \setminus M$ ). È anche noto che il linguaggio  $\{a^n b a^n \mid n \geq 0\}$  non è regolare. Utilizzare questi risultati per provare che il linguaggio  $L = \{a\}^* \cup \{a, b\}^* b \{a\}^* b \{a, b\}^* \cup \{a^h b a^k \mid h, k \in \mathbb{N}, h \neq k\}$  non è regolare.

7. È noto che la classe dei linguaggi regolari è chiusa rispetto alle operazioni di unione, intersezione, prodotto, complemento e differenza (cioè se  $L, M$  sono regolari lo sono anche  $L \cup M, L \cap M, LM, \Sigma^* \setminus M, L \setminus M$ ). Utilizzare questa proprietà per provare che il linguaggio  $L = \{0^n 1^n \mid n \geq 0\} \cup \{w \in \{0, 1\}^* \mid \exists h \in \mathbb{N} : |w| = 2h + 1\}$  non è regolare.

8. Indicare quali tra i seguenti linguaggi sono regolari.

- a)  $L_1 = \{(01^n)^n \mid n \geq 0\}$ ,
- b)  $L_2 = \{(01^n)^n (0^* 1^*)^* \mid n \geq 0\}$ ,
- c)  $L_3 = \{(01)^n \mid n \geq 0\}$ .

Giustificare le risposte (cioè in caso affermativo esibire un automa o una espressione regolare per il linguaggio, in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

9. Dimostrare, utilizzando il pumping lemma (per i linguaggi regolari), che il linguaggio  $\{w\phi(w) \mid w \in \{a, b\}^*\}$  sull'alfabeto  $\{a, b, 0, 1\}$  non è regolare, dove  $\phi(w)$  è la parola che si ottiene sostituendo in  $w$  ogni  $a$  con 0 e ogni  $b$  con 1 (cioè  $\phi : \{a, b\}^* \rightarrow \{0, 1\}^*$  è l'applicazione definita dalle relazioni:  $\phi(a) = 0, \phi(b) = 1, \phi(\epsilon) = \epsilon$ , per ogni  $a_1, \dots, a_n \in \{a, b\}$ ,  $\phi(a_1 \cdots a_n) = \phi(a_1) \cdots \phi(a_n)$ ).

10. Dimostrare, utilizzando il pumping lemma (per i linguaggi regolari), che il linguaggio  $\{w\phi(w) \mid w \in \{a\}^*\}$  sull'alfabeto  $\{a, b\}$  non è regolare, dove  $\phi(w)$  è la parola che si ottiene sostituendo in  $w$  ogni  $a$  con  $bb$  (cioè  $\phi : \{a\}^* \rightarrow \{b\}^*$  è l'applicazione definita dalle relazioni:  $\phi(a) = bb, \phi(\epsilon) = \epsilon$  e, con  $a_i = a$  per  $1 \leq i \leq n$ ,  $\phi(a_1 \cdots a_n) = \phi(a_1) \cdots \phi(a_n)$ ).

11. Sia  $\phi : \{0, 1\}^* \rightarrow \{0, 1\}^*$  l'applicazione definita dalle relazioni:  $\phi(0) = 00, \phi(1) = 11, \phi(\epsilon) = \epsilon$  e, per ogni  $a_1, \dots, a_n \in \{0, 1\}$ ,  $\phi(a_1 \cdots a_n) = \phi(a_1) \cdots \phi(a_n)$ . Il linguaggio  $L = \{w\phi(w) \mid w \in \{0, 1\}^*\}$  è regolare? Giustificare la risposta (cioè in caso affermativo esibire un automa a stati finiti che riconosce il linguaggio, in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

12. Sia  $\Sigma$  un alfabeto (finito e non vuoto) e, per ogni  $w \in \Sigma^*, a \in \Sigma$  denotiamo con  $|w|_a$  il numero di occorrenze della lettera  $a$  in  $w$ . Dato un linguaggio  $L$ ,  $L \subseteq \Sigma^*$ , sia:

$$Per(L) = \{y \in \Sigma^* \mid \exists w \in L : \forall a \in \Sigma \mid |w|_a = |y|_a\}.$$

- a) Definire  $Per(L)$  per  $L = (01)^*$ .
- b) È vero che se  $L$  è regolare, allora  $Per(L)$  è regolare?

13. Giustificare formalmente la risposta data al punto b) dell'esercizio precedente.

14. Siano dati i linguaggi  $L = \{a^n b^n \mid n \geq 0\}$ ,  $M = \{ww \mid w \in \{a, b\}^*\}$ . Il linguaggio  $L\{0, 1\}^*M$  è regolare? Giustificare la risposta (cioè in caso affermativo esibire un automa che riconosce  $L$  o un'espressione regolare che rappresenta  $L$ , in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

## GRAMMATICHE

1. Definire una grammatica context-free  $G$  il cui linguaggio generato  $L(G)$  sia:

$$L(G) = \{0^n 1^n \mid n \geq 0\} \cup \{w \in \{0, 1\}^* \mid \exists h \in \mathbb{N} : |w| = 2h + 1\}.$$

2. Sia  $G = (V, T, P, S)$  una grammatica context free, dove  $V = \{S, A\}$ ,  $T = \{a, b, c\}$  e

$$P = \{S \rightarrow aAb, S \rightarrow bS, S \rightarrow Sa, A \rightarrow aAb, A \rightarrow c\}.$$

Determinare il linguaggio  $L(G)$  generato dalla grammatica.

3. Sia  $G = (V, T, P, S)$  una grammatica context free, dove  $V = \{S, A, B\}$ ,  $T = \{0, 1\}$  e

$$P = \{S \rightarrow 0S, S \rightarrow 1A, S \rightarrow \epsilon, A \rightarrow 0B, B \rightarrow 1B, B \rightarrow \epsilon\}.$$

Determinare il linguaggio  $L(G)$  generato dalla grammatica.

4. Sia  $G = (V, T, P, S)$  una grammatica context free, dove  $V = \{S, A, B\}$ ,  $T = \{a, b\}$  e

$$P = \{S \rightarrow AB, S \rightarrow ABS, A \rightarrow aA, A \rightarrow a, B \rightarrow bA\}.$$

Determinare il linguaggio  $L(G)$  generato dalla grammatica.

5. Sia  $\phi : \{0, 1\}^* \rightarrow \{0, 1\}^*$  l'applicazione definita dalle relazioni:  $\phi(0) = 00$ ,  $\phi(1) = 11$ ,  $\phi(\epsilon) = \epsilon$  e, per ogni  $a_1, \dots, a_n \in \{0, 1\}$ ,  $\phi(a_1 \dots a_n) = \phi(a_1) \dots \phi(a_n)$ . Sia  $w^R$  l'inversa di  $w \in \{0, 1\}^*$  cioè  $\epsilon^R = \epsilon$ ,  $(a_1 \dots a_n)^R = a_n \dots a_1$ . Il linguaggio  $L = \{w\phi(w^R) \mid w \in \{0, 1\}^*\}$  è context-free? Giustificare la risposta (cioè in caso affermativo esibire una grammatica o un automa pushdown per il linguaggio, in caso negativo esporre l'argomento che fornisce risposta negativa). Risposte non giustificate saranno valutate zero.

6. Determinare il linguaggio  $L(G)$  generato dalla grammatica context-free  $G = (V, T, P, S)$ , dove  $V = \{S\}$ ,  $T = \{a, b\}$  e  $P = \{S \rightarrow aSb, S \rightarrow Sa, S \rightarrow bS, S \rightarrow \epsilon\}$ .

7. Definire una grammatica  $G$  context-free il cui linguaggio generato sia

$$L = \{a^n b^n c^m d^m \mid n, m \geq 0\}.$$

8. Definire una grammatica context-free  $G$  il cui linguaggio generato  $L(G)$  sia:

$$L(G) = \{0, 1\}^* \{10\} \{0, 1\}^* \cup \{0^h 1^k \mid h, k \in \mathbb{N}, h \neq k\}.$$

9. Definire una grammatica context-free  $G$  il cui linguaggio generato  $L(G)$  sia:

$$L(G) = \{a^h b a^k \mid h, k \in \mathbb{N}, h \neq k\}.$$

10. Sia  $G = (V, T, P, S)$  una grammatica context free, dove  $V = \{S, A, B\}$ ,  $T = \{a, b\}$  e

$$P = \{S \rightarrow BbA, B \rightarrow aAbB, B \rightarrow \epsilon, A \rightarrow aA, A \rightarrow \epsilon\}.$$

Determinare il linguaggio  $L(G)$  generato dalla grammatica.

11. Sia  $G = (V, T, P, S)$  una grammatica context free, dove  $V = \{S, A, B\}$ ,  $T = \{0, 1\}$  e

$$P = \{S \rightarrow 0A, S \rightarrow 1B, A \rightarrow 0A, A \rightarrow 0S, A \rightarrow 1B, B \rightarrow 1, B \rightarrow 0\}.$$

Determinare il linguaggio  $L(G)$  generato dalla grammatica.

12. Sia  $\Sigma = \{0, 1\}$ . Per ogni  $w \in \Sigma^*$ , denotiamo con  $\bar{w}$  la parola ottenuta da  $w$  cambiando ogni 0 in 1 e ogni 1 in 0 (esempio:  $w = 1011$ ,  $\bar{w} = 0100$ ). Denotiamo inoltre con  $w^R$  l'inversione di  $w$ , definita dalle relazioni:  $\epsilon^R = \epsilon$ ,  $(a_1 \dots a_n)^R = a_n \dots a_1$ , con  $a_1, \dots, a_n \in \Sigma$ . Consideriamo il linguaggio  $L = \{w \in \Sigma^* \mid w^R = \bar{w}\}$ . Definire una grammatica context free  $G = (V, T, P, S)$  tale che  $L = L(G)$  sia il linguaggio generato dalla grammatica.

13. Sia  $L$  un linguaggio context-free e sia  $G = (V, T, P, S)$  una grammatica che genera  $L$ . Dare un esempio di linguaggio infinito  $M$  tale che  $MLM$  sia ancora context-free e definire una grammatica context-free  $G'$  che genera  $MLM$ .

14. Sia  $G = (V, T, P, S)$  una grammatica context free, dove  $V = \{S, A, B, C\}$ ,  $T = \{a, b\}$  e

$$P = \{S \rightarrow A, S \rightarrow B, A \rightarrow CAC, A \rightarrow a, B \rightarrow CBC, B \rightarrow b, C \rightarrow a, C \rightarrow b\}.$$

Determinare il linguaggio  $L(G)$  generato dalla grammatica e definire un automa finito deterministico  $\mathcal{A}$  con al più quattro stati, il cui linguaggio  $L(\mathcal{A})$  coincida con  $L(G)$ .

15. • Definire una grammatica context free  $G_1$  tale che  $L(G_1) = \{a^i b^j c^k d^\ell \mid i, j, k, \ell \geq 1, i = j, k = \ell\}$ , dove  $L(G_1)$  è il linguaggio generato da  $G_1$ .  
 • Definire una grammatica context free  $G_2$  tale che  $L(G_2) = \{a^i b^j c^k d^\ell \mid i, j, k, \ell \geq 1, i = \ell, j = k\}$ , dove  $L(G_2)$  è il linguaggio generato da  $G_2$ .

16. • Definire una grammatica context free  $G_1$  tale che  $L(G_1) = \{a^n b^k a^n \mid k, n \geq 0\}$ , dove  $L(G_1)$  è il linguaggio generato da  $G_1$ .  
 • Definire una grammatica context free  $G_2$  tale che  $L(G_2) = \{a^n b^{2n} c^k \mid k, n \geq 0\}$ , dove  $L(G_2)$  è il linguaggio generato da  $G_2$ .

17. Sia  $\Sigma = \{a, b\}$  e, per ogni  $w \in \{a, b\}^*$ , denotiamo con  $|w|_a$  il numero di occorrenze della lettera  $a$  in  $w$  e con  $|w|_b$  il numero di occorrenze della lettera  $b$  in  $w$ . Inoltre, sia  $Pref(w) = \{x \in \Sigma^* \mid \exists y \in \Sigma^* : xy = w\}$  e  $Suff(w) = \{y \in \Sigma^* \mid \exists x \in \Sigma^* : xy = w\}$ . Infine, siano  $L = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$ ,  $Pref(L) = \bigcup_{w \in L} Pref(w)$  e  $Suff(L) = \bigcup_{w \in L} Suff(w)$ . Rispondere alle domande seguenti giustificando formalmente le risposte. Risposte non giustificate saranno valutate zero.

- $Pref(L)$  è regolare o non regolare ma context-free?
- $Suff(L)$  è regolare o non regolare ma context-free?

18. Sia  $G = (V, T, P, S)$  una grammatica context free, dove  $V = \{S, A\}$ ,  $T = \{a, b\}$  e

$$P = \{S \rightarrow aSb, S \rightarrow bA, S \rightarrow Aa, A \rightarrow bA, A \rightarrow aA, A \rightarrow \epsilon\}.$$

Determinare il linguaggio  $L(G)$  generato dalla grammatica.

19. Sia  $G = (V, T, P, S)$  una grammatica context free, dove  $V = \{S, A\}$ ,  $T = \{0, 1\}$  e

$$P = \{S \rightarrow 0S, S \rightarrow 1A, S \rightarrow \epsilon, A \rightarrow 1S, A \rightarrow 0A, A \rightarrow \epsilon\}.$$

Determinare il linguaggio  $L(G)$  generato dalla grammatica.

20. Sia  $G = (V, T, P, S)$  una grammatica context free, sia  $L = L(G)$  il linguaggio generato da  $G$ . Definire una grammatica context free  $G'$  tale che  $L(G') = L^* = (L(G))^*$ .

## ELENCO DEFINIZIONI

- ✓ **(Def. Induzione):** Supponiamo di dover dimostrare un enunciato  $S(n)$  su un intero  $n$ . Una soluzione comune consiste nel dimostrare due cose.
  3. *La base*, in cui si dimostra  $S(i)$  per un particolare intero  $i$ . Di solito  $i = 0$  oppure  $i = 1$ , ma ci sono esempi in cui si comincia da un più alto, magari perché l'enunciato  $S$  è falso per alcuni interi piccoli.
  4. Il passo induttivo, in cui supponiamo che  $n \geq i$ , dove  $i$  è l'intero di base, e si dimostra che "se  $S(n)$  è vera, allora anche  $S(n + 1)$  lo è".
- ✓ **(Def. Induzione mutua):** si deve dimostrare congiuntamente un gruppo di enunciati  $S_1(n), S_2(n), \dots, S_k(n)$  per induzione su  $n$ .
- ✓ **(Def. alfabeto):** un alfabeto è un insieme finito e non vuoto di simboli.
- ✓ **(Def. Stringhe):** una stringa (o parola) è una sequenza finita di simboli scelti da un alfabeto.
- ✓ **(Def. stringa vuota):** la stringa vuota è la stringa composta da zero simboli. Questa stringa, indicata con  $\epsilon$ , è una stringa che può essere scelta da un qualunque alfabeto.
- ✓ **(Def. Lunghezza di una stringa):** il numero di posizioni per i simboli della stringa.
- ✓ **(Def. Potenza di un alfabeto):** Definiamo  $\sum^k$  come l'insieme di stringhe di lunghezza  $k$ , con simboli tratti da  $\sum$ .

### DFA

- ✓ **(Def. Di DFA):** un automa che dopo aver letto una qualunque sequenza di input si trova in un singolo stato. Il termine "deterministico" concerne il fatto che per ogni input esiste un solo stato verso il quale l'automa passa dal suo stato corrente. Nelle dimostrazioni denotiamo un DFA come una quintupla:  $A = (Q, \Sigma, q_0, \delta, F)$
- ✓ **(Def. Funzione di transizione DFA):** Una funzione di transizione, che prende come argomento uno stato e un simbolo di input e restituisce uno stato.
- ✓ **(Def. Funzione di transizione estesa DFA):** Dato un DFA  $A = (Q, \Sigma, q_0, \delta, F)$ , la funzione di transizione estesa  $\hat{\delta}$  è definita nel modo seguente:  $\hat{\delta}: Q \times \Sigma^* \rightarrow Q$ . La funzione di transizione estesa è una funzione che prende uno stato  $q$  e una stringa  $w$  e restituisce uno stato  $p$ : lo stato che l'automa raggiunge quando parte nello stato  $q$  ed elabora la sequenza di input  $w$ .
- ✓ **(Def. Di linguaggio di un DFA):** Il "linguaggio" del DFA è l'insieme di tutte le stringhe che il DFA accetta. Sia il DFA  $A = (Q, \Sigma, q_0, \delta, F)$ . Il linguaggio è indicato con  $L(A)$  ed è definito da  $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$ . In altre parole il linguaggio di  $A$  è l'insieme delle stringhe  $w$  che portano dallo stato iniziale  $q_0$  a uno degli stati accettanti. Se  $L$  è uguale a  $L(A)$  per un DFA  $A$ , allora diciamo che  $L$  è un linguaggio regolare.
- ✓ **(Def. Di linguaggio regolare):** un linguaggio  $L \subseteq \Sigma^*$  si dice regolare se esiste un DFA  $A$  t.c.  $L$  è il linguaggio dell'automa, cioè  $L=L(A)$ .

### NFA

- ✓ **(Def. NFA):** Un automa a stati finiti non deterministico (NFA, Non-deterministic Finite Automaton) può trovarsi contemporaneamente in diversi stati. Questa caratteristica viene sovente espressa come capacità di "scommettere" su certe proprietà dell'input. Un automa finito non deterministico NFA si rappresenta essenzialmente come un DFA:  $A = (Q, \Sigma, \delta, q_0, F)$
- ✓ **(Def. Funzione di transizione NFA)** La funzione di transizione  $\delta$  è la funzione che ha come argomenti uno stato in  $Q$  e un simbolo di input in  $\Sigma$ , e restituisce un sottoinsieme di  $Q$ .  $\delta : Q \times \Sigma \rightarrow \wp(Q)$  dove  $\wp(Q) = \text{insieme delle parti di } Q$  (*insieme dei sottinsiemi di*  $Q$ )

- ✓ **(Def. Funzione di transizione estesa NFA):** Dato un automa finito non deterministico NFA  $A = (Q, \Sigma, \delta, q_0, F)$ , la funzione di transizione estesa  $\hat{\delta}$  è la funzione  $\hat{\delta}: Q \times \Sigma^* \rightarrow \wp(Q)$  dove  $\wp(Q)$  definito induttivamente come segue:

BASE:  $\forall q \in Q, \hat{\delta}(q, \varepsilon) = \{q\}$ . Se nessun simbolo di input è stato letto, ci troviamo nel solo stato da cui siamo partiti.

INDUZIONE: Supponiamo che  $w$  sia della forma  $w = xa$ , dove  $a$  è il simbolo finale di  $w$  e  $x$  è la parte restante.

$$\forall q \in Q, \forall x \in \Sigma^*, \forall a \in \Sigma: \hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q, x)} \delta(p, a)$$

Inoltre  $\forall u, v \in \Sigma^*, \forall q \in Q$  abbiamo  $\hat{\delta}(q, uv) = \hat{\delta}(\hat{\delta}(q, u), v)$

- ✓ **(Def. Linguaggio di un NFA):** Sia  $A = (Q, \Sigma, \delta, q_0, F)$  un NFA. Il linguaggio riconosciuto da  $A$ ,  $L(A)$ , è definito da:  $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$ .
- ✓ **(TEOREMA DA NFA A DFA):** Per ogni automa finito non deterministico  $A_N = (Q_N, \Sigma, \delta_N, q_N, F_N)$  esiste un automa deterministico  $A_D = (Q_D, \Sigma, \delta_D, q_D, F_D)$  tale che  $L(A_N) = L(A_D)$

## DIMOSTRAZIONE

La dimostrazione consiste in 3 passi:

1)Passo 1: Costruiamo un automa in DFA  $A_D$  a partire da  $A_N$  mediante la “subset construction” cioè  $A_D = (Q_D, \Sigma, \delta_D, q_D, F_D)$  dove

$Q_D = \wp(Q_N)$  (c’è una corrispondenza biunivoca tra gli insiemi degli stati deterministici con quelli non deterministici),

$q_0 = \{q_N\}$ ,

$F_D = \{S \subseteq Q_N \mid S \cap F_N \neq \emptyset\}$

$\delta_D: Q_D \times \Sigma \rightarrow Q_D$  è definita da  $\forall S \subseteq Q_N, \forall a \in \Sigma \quad \delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$

### 2)Passo 2

Dimostrare che per ogni  $w \in \Sigma^*$  si ha  $\hat{\delta}_D(q_D, w) = \hat{\delta}_N(q_N, w)$  (\*)

La dimostrazione avviene per induzione sulla lunghezza di  $w$ .

**(BASE)** Sia  $w = \varepsilon : \hat{\delta}_D(q_D, \varepsilon) = (\text{def. di DFA}) = q_D = (\text{costruzione di DFA}) = \{q_N\} = (\text{def. Sulla funzione estesa di transizione di NFA}) = \hat{\delta}_N(q_N, \varepsilon)$ . Quindi (\*) è vera per  $w = \varepsilon$

**(PASSO INDUTTIVO)** Sia  $w = xa \quad x \in \Sigma^* \quad a \in \Sigma$

$$\begin{aligned} \hat{\delta}_D(q_D, w) &= \hat{\delta}_D(q_D, xa) = (\text{def. di DFA}) = \delta_D(\hat{\delta}_D(q_D, x), a) = (\text{costr. di DFA}) \\ &= \bigcup_{p \in \hat{\delta}_D(q_D, x)} \delta_N(p, a) = (\text{ipotesi induttiva}) = \bigcup_{p \in \hat{\delta}_N(q_N, x)} \delta_N(p, a) \\ &= (x \text{ definz di NFA}) = \hat{\delta}_N(q_N, xa) = \hat{\delta}_N(q_N, w) \end{aligned}$$

Quindi (\*) è vera.

### 3)Passo 3

Dimostrazione che  $L(A_D) = L(A_N)$

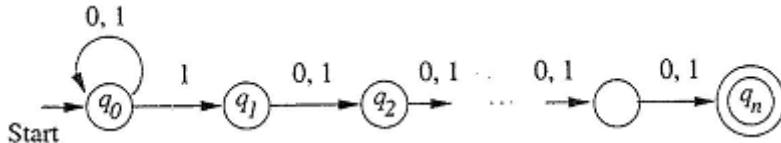
$w \in L(A_D) \Leftrightarrow (\text{def. di linguaggio di un automa}) \Leftrightarrow \hat{\delta}_D(q_D, w) \in F_D \Leftrightarrow (\text{per costruzione di } A_D) \Leftrightarrow \hat{\delta}_D(q_D, w) \cap F_N \neq \emptyset \Leftrightarrow (\text{per il passo 2}) \Leftrightarrow \hat{\delta}_N(q_N, w) \cap F_N \neq \emptyset \Leftrightarrow w \in L(A_N)$

- ✓ **(TEOREMA DA DFA A NFA):** Sia  $A = (Q, \Sigma, \delta, q_0, F)$  un DFA, esiste un NFA  $A_N = (Q_N, \Sigma, \delta_N, q_N, F_N)$  tale che  $L(A) = L(A_N)$ . È evidente in quanto lo possiamo vedere come un NFA che può fare una sola scelta.

**(Sketch)** Sia  $A = (Q, \Sigma, \delta, q_0, F)$  un DFA, considero l’ NFA  $A_N = (Q_N, \Sigma, \delta_N, q_N, F_N)$  definito come segue  $Q_N = Q, q_N = q_0, F_N = F, \forall q \in Q_N = Q \quad \forall a \in \Sigma \quad \delta_N(q, a) = \{\delta(q, a)\}$

Si può dimostrare che  $L(A_N) = L(A)$

- ✓ **Corollario:** Un linguaggio è riconosciuto da un DFA se e solo se è riconosciuto da un NFA.
- ✓ **(Nota):** Se un NFA ha  $n$  stati, nel caso peggiore il corrispondente DFA ha  $2^n$  stati. Consideriamo l'NFA della figura.



Sia  $L(N)$  è l'insieme di tutte le stringhe di 0 e di 1 tali che l' $n$ -esimo simbolo dalla fine sia 1.

Intuitivamente un DFA  $D$  che accetti questo linguaggio deve ricordare gli ultimi  $n$  simboli che ha letto.

Poiché ci sono  $2^n$  sequenze distinte di lunghezza  $n$ , se  $D$  avesse meno di  $2^n$  stati, ci sarebbe uno stato  $q$  raggiunto dopo aver letto due sequenze distinte di  $n$  bit, poniamo  $a_1a_2...a_n$  e  $b_1...b_n$ .

Dato che le sequenze sono diverse, devono differire in una posizione, per esempio ai diversi da  $b_i$ . Supponiamo (per simmetria), che  $a_i=1$  e  $b_i=0$ , rendendo  $x$  accettante e  $y$  no.

i

$$x = a_1 \dots 1 \dots a_n$$

$$y = b_1 \dots 0 \dots b_n$$

### Principio della piccionaia

$$|\{0,1\}^n| = 2^n$$

<b>x (cellette)</b>	<b>y (piccioni)</b>
$a_1$	$b_1$
$a_2$	$b_2$
$a_3$	$b_3$
.	.
.	.
.	.
$a_n$	$b_k$

In parole povere, se si hanno più piccioni che cellette e ogni piccione si ricovera in una celletta, allora ci deve essere almeno una celletta contenente più di un piccione. Nel nostro caso i "piccioni" sono le sequenze di  $n$  bit e le "cellette" sono gli stati. Poiché esistono meno stati che sequenze, a uno stato devono essere assegnate due sequenze.

Il principio della piccionaia può sembrare ovvio, ma dipende dal fatto che il numero delle cellette è finito. Funziona perciò per automi a stati finiti, in cui gli stati corrispondono alle cellette della piccionaia. Non si può applicare invece ad altri tipi di automi che hanno un numero infinito di stati.

Per capire perché è essenziale che il numero delle cellette sia finito, si consideri la situazione infinita in cui le cellette corrispondono agli interi  $1, 2, \dots$ . Numeriamo i piccioni  $0, 1, 2, \dots$ , in modo che ci sia un piccione in più rispetto alle cellette. Possiamo allora mandare il piccione  $i$  nella celletta  $i + 1$  per ogni  $i > 0$ . Così ogni piccione ha la sua celletta e non ci sono due piccioni obbligati a condividere lo stesso spazio.

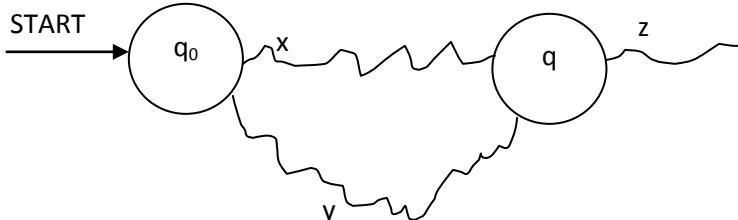
Se considero gli insiemi degli stati a partire dagli stati iniziali a partire dalla stringa  $n$

$$\{\hat{\delta}(q_0, x) \mid x \in \{0,1\}^n\} \subseteq Q$$

$$|\{\hat{\delta}(q_0, x) \mid x \in \{0,1\}^n\}| \leq 2^n$$

per il principio della piccionaia esistono almeno 2 stringhe

$$x, y \in \{0,1\}^n, x \neq y \text{ t.c. } \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$$



i

$$x = a_1 \dots a_n$$

$$y = b_1 \dots b_n$$

$$xz = a_1 \dots a_{n-1} 0^{i-1}$$

$$yz = b_1 \dots b_{n-1} 0^{i-1}$$

con  $z = 0^{i-1}$ . Inoltre, puntualizziamo che  $xz \in L(A)$  mentre  $yz \notin L(A)$

$\hat{\delta}(q_0, xz) = \hat{\delta}(\hat{\delta}(q_0, x), z) = \hat{\delta}(\hat{\delta}(q_0, y), z) = \hat{\delta}(q_0, yz) \notin F$  e questo è ASSURDO perché in questo caso entrambi accettano  $xz \in L(A)$  e  $yz \in L(A)$ .

### ***ε – NFA***

- ✓ **(Def. ε – NFA):** È come se l'NFA potesse compiere una transizione spontaneamente, senza aver ricevuto un simbolo di input. Possiamo rappresentare un  $\epsilon$ -NFA esattamente come un NFA, salvo che per un aspetto: la funzione di transizione deve incorporare informazioni sulle transizioni etichettate da  $\epsilon$ . Formalmente denotiamo un  $\epsilon$ -NFA (o automa finito con  $\epsilon$ -transizioni)  $A = (Q, \Sigma, \delta, q_0, F)$ , dove tutti i componenti si interpretano come per gli NFA:  $Q$ =insieme finito degli stati,  $q_0 \in Q$  stato iniziale,  $F \subseteq Q$ ,  $F$  insieme degli stati finali, mentre  $\delta$  è una funzione che richiede come argomenti:  $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \wp(Q)$
- ✓ **(TEOREMA ε-NFA)** Un linguaggio  $L$  è riconosciuto da un automa finito con  $\epsilon$ -transizioni se e solo se  $L$  è riconosciuto da un' automa finito deterministico DFA.

### **ESPRESSIONI E LINGUAGGI REGOLARI**

- ✓ **(Def. Pumping Lemma):** Sia  $L$  un linguaggio regolare ( $L \subseteq \Sigma^*$ ).  
 $\exists n > 0$  (intero positivo dipendente da  $L$ ) tale che  $\forall w \in L, |w| \geq n,$   
 $\exists x, y, z$  tali che ci permettono di scomporre  $w = xyz$

- 1)  $|xy| \leq n$
- 2)  $y \neq \epsilon$
- 3)  $\forall k \geq 0, xy^k z \in L$

In altre parole, possiamo sempre trovare una stringa non vuota  $y$ , non troppo distante dall'inizio di  $w$ , da "replicare", cioè da ripetere quante volte vogliamo o anche cancellare (è il caso di  $k=0$ ) senza uscire dal linguaggio  $L$ .

#### **Dimostrazione**

(Ipotesi) Supponiamo che  $L$  è un linguaggio regolare  $\rightarrow \exists$  un DFA  $A = (Q, \Sigma, \delta, q_0, F)$  tale che  $L(A) = L$ .

Affermo che per  $n = |Q|$  l'enunciato è vero. Infatti:

sia  $w = a_1, \dots, a_m$ , con  $a_i \in \Sigma$ ,  $m \geq n$ , e tale che  $w \in L$ , cioè  $\hat{\delta}(q_0, w) \in F$

$$\hat{\delta}(q_0, \varepsilon) = p_0 = q_0$$

$$\hat{\delta}(q_0, a_1) = p_1$$

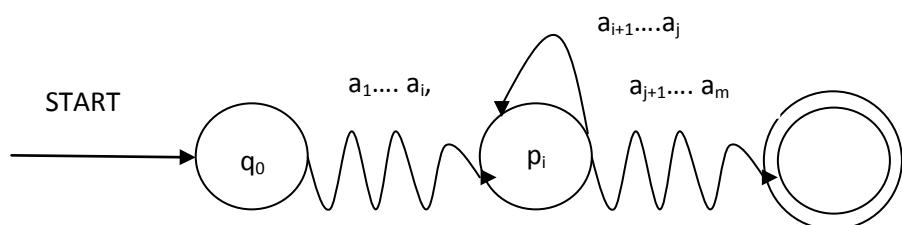
$$\hat{\delta}(q_0, a_1 a_2) = p_2$$

...

$$\hat{\delta}(q_0, a_1 \dots a_n) = p_n$$

$$\text{Cioè } \hat{\delta}(q_0, a_1 \dots a_i) = p_i \quad 0 \leq i \leq n$$

Dunque  $p_i$  è lo stato in cui si trova A dopo aver letto i primi  $i$  simboli di  $w$ . Per il principio della piccionaia, dato che ci sono solo  $n$  stati, gli  $n+1$  stati  $p_i$  per  $i=0, 1, \dots, n$  non possono essere tutti distinti. Ci sono perciò due interi distinti  $i$  e  $j$  tali che  $\hat{\delta}(q_0, a_1 \dots a_i) = p_i = p_j = \hat{\delta}(q_0, a_1 \dots a_j)$  con  $0 \leq i < j \leq n$ .



Scomponiamo  $w$  con:

$$\text{Poniamo } x = a_1 \dots a_i \quad y = a_{i+1} \dots a_j \quad z = a_{j+1} \dots a_m$$

In altre parole  $x$  porta a  $p_i$  la prima volta,  $y$  riporta da  $p_i$  a  $p_j$  (dato che  $p_i$  e  $p_j$  coincidono), mentre  $z$  conclude  $w$ .

$$\text{Risulta } w = (a_1 \dots a_i)(a_{i+1} \dots a_j)(a_{j+1} \dots a_m) = xyz \quad (\text{Proposizione 1})$$

$$\text{Inoltre } |xy| = |a_1, \dots, a_i, a_{i+1} \dots a_j| = j-i \leq n \quad (\text{Proposizione 2})$$

$$|y| = |a_{i+1} \dots a_j| = j-i > 0 \rightarrow y \neq \varepsilon \quad (\text{Proposizione 3})$$

Si noti che  $x$  può essere vuota (quando  $i=0$ ), così come  $z$  (quando  $j=n=m$ ). viceversa,  $j$  non può essere vuota perché  $i$  è strettamente minore di  $j$ .

Proviamo la proposizione 4, cioè  $\forall k \geq 0, xy^k z \in L$

Passo preliminare Dimostriamo (per induzione su  $k$ ) che risulta

$$\forall k \geq 0 \quad \hat{\delta}(q_0, x) = \hat{\delta}(q_0, xy) = \hat{\delta}(q_0, xy^k) \quad (*)$$

**(Base)** Se  $k=0$ , l'automa passa dallo stato iniziale  $q_0$  (che è anche  $p_0$ ) a  $p_i$  sull'input  $x$ .

$\hat{\delta}(q_0, x) = \hat{\delta}(q_0, a_1 \dots a_i) = p_i$ . Poiché  $p_i$  coincide con  $p_j$ ,  $p_i = p_j = \hat{\delta}(q_0, a_1 \dots a_i) = \hat{\delta}(q_0, xy) = \hat{\delta}(q_0, xy^0)$  è vero per  $k=0$

$$\begin{aligned} \text{(Passo Induttivo)} \quad k > 0 \text{ e } z = \varepsilon \quad \hat{\delta}(q_0, xy^k) &= \hat{\delta}(\hat{\delta}(q_0, xy^{k-1}), y) = \hat{\delta}(\hat{\delta}(q_0, x), y) = \hat{\delta}(q_0, xy) \\ &\uparrow \quad \uparrow \quad \uparrow \\ &\text{Per una proprietà} \quad \text{Per ipotesi} \quad \text{Per una proprietà} \\ &\text{della funzione di} \quad \text{induttiva} \quad \text{della funzione di} \\ &\text{transizione in un DFA} \quad \text{transizione in un DFA} \end{aligned}$$

Se  $p_j = p_i = \hat{\delta}(q_0, a_1 \dots a_i) = \hat{\delta}(q_0, x)$  (\*) è vero per  $k>0$

$$\begin{aligned} \forall k \geq 0 \quad \hat{\delta}(q_0, xy^k z) &= \hat{\delta}(\hat{\delta}(q_0, xy^k), z) = \hat{\delta}(\hat{\delta}(q_0, xy), z) = \hat{\delta}(q_0, xyz) = \hat{\delta}(q_0, w) \in F \\ &\uparrow \quad \uparrow \quad \uparrow \\ &\text{Per una proprietà} \quad \text{Per (*)} \quad \text{Per una proprietà} \\ &\text{della funzione di} \quad \text{della funzione di} \quad \text{della funzione di} \\ &\text{transizione in un DFA} \quad \text{transizione in un DFA} \quad \text{transizione in un DFA} \end{aligned}$$

(perché per ipotesi  $w \in L$ )

Siccome  $\forall k \geq 0, \hat{\delta}(q_0, xyz) \in F$ , concludiamo che  $\forall k \geq 0, xy^kz \in L$  (Proposizione 4) (fine dimostrazione)

$L$  regolare  $\rightarrow L$  verifica il Pumping Lemma

$L$  verifica il Pumping Lemma  $\rightarrow L$  regolare

Conclusione: Esistono linguaggi non regolari

- ✓ **(Def. Prodotto di linguaggi)** Siano  $L \subseteq \Sigma^*$ ,  $M \subseteq \Sigma^*$  due linguaggi, il prodotto di  $L$  e  $M$  è denotato con  $L \cdot M$
- ✓ **(Def. potenza di linguaggi)** Potenza di un linguaggio  $L \subseteq \Sigma^*$  (Base)  $L^0 = \{\epsilon\}$  (Induzione)  $\forall n \geq 0, L^n = L^{n-1} \cdot L$
- ✓ **(Def. Chiusura di Kleene)** Chiusura di Kleene di  $L \subseteq \Sigma^*$  Sia  $L \subseteq \Sigma^*$ , definiamo  $L^* = \bigcup_{n \geq 0} L^n$
- ✓ **(Def. Espressione Regolare):** Un' espressione regolare su di un alfabeto  $\Sigma$  è definita ricorsivamente nel seguente modo:

**BASE:**  $\emptyset, \epsilon$ , a sono espressioni regolari,  $\forall a \in \Sigma$ .

**INDUZIONE:** Se  $E_1$  ed  $E_2$  sono espressioni regolari allora:

$E_1 + E_2$  è un' espressione regolare

$E_1 \cdot E_2$  è un' espressione regolare

$E_1^*$  è un espressione regolare

$(E_1)$  è un' espressione regolare

- ✓ **(Def. Regole di priorità di un'ER):**

- + \* ha priorità più elevata rispetto a ( $\cdot$  e  $+$ )
- +  $\cdot$  dopo priorità di \*
- +  $+$  più bassa priorità
- +  $()$  per cambiare l'ordine delle priorità

- ✓ **(Def. Proprietà di chiusura dei linguaggi regolari)**

8. L'unione di due linguaggi regolari è regolare. Siano  $L$  ed  $M$  linguaggi sull'alfabeto  $\Sigma$ . Allora  $L \cup M$  è il linguaggio che contiene tutte le stringhe che si trovano in  $L$  o in  $M$  oppure in entrambi.
9. L'intersezione di due linguaggi regolari è regolare. Siano  $L$  ed  $M$  linguaggi sull'alfabeto  $\Sigma$ . Allora  $L \cap M$  è il linguaggio che contiene tutte le stringhe che si trovano sia in  $L$  che in  $M$ .
10. Il complemento di un linguaggio regolare è regolare. Siano  $L$  sull'alfabeto  $\Sigma$ . Allora  $\bar{L}$  è l'insieme delle stringhe in  $\Sigma^*$  che non sono in  $L$ .
11. La differenza di due linguaggi regolari è regolare. Siano  $L$  ed  $M$  linguaggi sull'alfabeto  $\Sigma$ . Allora  $L - M$  è il linguaggio che contiene tutte le stringhe che si trovano sia in  $L$  ma non in  $M$ .
12. L'inversione di un linguaggio regolare è regolare
13. La chiusura (star) di un linguaggio regolare è regolare
14. La concatenazione di linguaggi regolari è regolare

- ✓ **(Def. Linguaggio di un'ER):** Il linguaggio  $L(E)$ , denotato da un' espressione regolare  $E$  su  $\Sigma$ , è il linguaggio per  $\Sigma$  definito ricorsivamente come segue:

**BASE:**

- Se  $E = \emptyset$  allora  $L(E) = L(\emptyset) = \emptyset$
- Se  $E = \epsilon$  allora  $L(E) = L(\epsilon) = \{ \epsilon \}$
- Se  $E = a$  allora  $L(E) = L(a) = \{ a \}$

**INDUZIONE:**

- Se  $E_3 = E_1 + E_2$  con  $E_1$  ed  $E_2$  espressioni regolari, allora  $L(E_3) = L(E_1) \cup L(E_2)$
- Se  $E_3 = E_1 \cdot E_2$  con  $E_1$  ed  $E_2$  espressioni regolari, allora  $L(E_3) = L(E_1) \cdot L(E_2)$
- Se  $E_1 = E^*$  con  $E$  espressione regolare, allora  $L(E_1) = L(E)^*$
- Se  $E_1 = (E)$  con  $E$  espressione regolare, allora  $L(E_1) = L((E))$

✓ **(Teorema 1 : da DFA a ER)** Sia  $A$  un DFA, esiste un' espressione regolare  $E$  tale che  $L(A) = L(E)$ .

$$\{L \subseteq \Sigma^* \text{ t.c. } \exists \text{ un DFA } A \text{ t.c. } L = L(A)\} \subseteq \{L \subseteq \Sigma^* \text{ t.c. } \exists \text{ un E.R. } E \text{ t.c. } L = L(E)\}$$

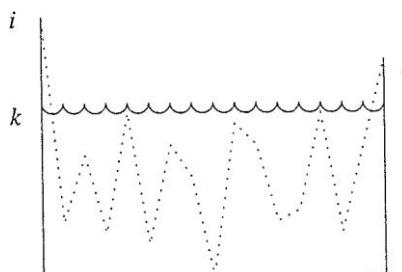
### DIMOSTRAZIONE

Sia  $A$  un DFA  $(Q, \Sigma, \delta, q_0, F)$ . Scelgo di denotare gli stati di  $A$   $Q$ , mediante numeri interi  $\{1, 2, \dots, n\}$  per un intero  $n$ , quindi  $|Q|=n$ . Suppongo che lo stato iniziale sia 1.

#### PASSO PRELIMINARE

Voglio far vedere che esiste un' espressione regolare  $E$  che denota il linguaggio. Usando la notazione  $R_{i,j}^{(k)}$  come nome dell'espressione regolare il cui linguaggio è l'insieme di stringhe  $w$  t.c.  $w$  sia l'etichetta di un cammino dallo stato  $i$  allo stato  $j$  in  $A$ , ed il cammino non abbia alcun nodo intermedio il cui numero sia maggiore di  $k$ .

Si noti che i punti di partenza e di arrivo del cammino non sono intermedi; dunque non richiede che  $i$  e  $j$  siano inferiori o uguali a  $k$ .



Vado da  $i$  a  $j$  senza passare per i nodi, il cui indice, stabilito all'inizio su  $|Q|$  sia maggiore di  $k$ .

$$R_{i,j}^{(k)} = \{w \in \Sigma^* \text{ t.c. } \hat{\delta}(i,w)=j \text{ e } \forall x, y \text{ t.c. } w=xy \text{ e } x \neq \epsilon \text{ e } x \neq w \text{ e } \hat{\delta}(i,x) \leq k\}$$

$i, j \in \{1, 2, \dots, n\}$  e  $0 \leq k \leq n$  -> DIMOSTRO PER INDUZIONE SU K.

**BASE:**  $K=0$ .

Dato che tutti gli stati sono numerati da 1 in su, la restrizione comporta che il cammino non abbia nessuno stato intermedio.

$$R_{i,j}^{(0)} = \{w \in \Sigma^* \text{ t.c. } \hat{\delta}(i,w)=j \text{ e } \forall x, y \text{ t.c. } w=xy \text{ e } x \neq \epsilon \text{ e } x \neq w \text{ e } \hat{\delta}(i,x) \leq 0\} \rightarrow \text{Può contenere la parola di un carattere} \rightarrow w=a \rightarrow \{w \in \Sigma^* \text{ t.c. } \hat{\delta}(i,w)=j \text{ e } w \in \Sigma \cup \{\epsilon\}\}$$



- **CASO 2:**  $\hat{\delta}(i, \epsilon) = j \rightarrow$  Cammino di lunghezza 0(zero) che consiste solamente nel nodo  $i$

**Distinguo due casi:**

c)  $i \neq j$

- a.  $R_{i,j}^{(0)} = \emptyset$  allora non esiste un arco  $(i,j)$  NON ESISTE

The diagram shows two oval states. The left state is labeled 'i' below it. An arrow labeled 'i' points from 'i' to the right state, which is labeled 'j' below it.
- b.  $R_{i,j}^{(0)} = \{a\}$  allora esiste un arco  $(i,j)$  la cui etichetta è  $a$

The diagram shows two oval states. The left state is labeled 'i' below it. An arrow labeled 'a' points from 'i' to the right state, which is labeled 'j' below it.
- c.  $R_{i,j}^{(0)} = a_1 + a_2 + \dots + a_k$  allora esistono simboli  $\{a_1, a_2, \dots, a_k\}$  che etichettano archi dallo stato  $i$  allo stato  $j$ .

The diagram shows two oval states. The left state is labeled 'i' below it. An arrow labeled 'a1,...,ak' points from 'i' to the right state, which is labeled 'j' below it.

Per  $R_{i,j}^{(0)}$  esiste l'espressione regolare  $R_{i,j}^{(0)}$  che denota questo linguaggio:

-) Se  $R_{i,j}^{(0)} = \emptyset$  allora espressione regolare =  $\emptyset$

-) Se  $R_{i,j}^{(0)} = \{ a \}$  allora espressione regolare = a

-) Se  $R_{i,j}^{(0)} = \{ a_1, a_2, \dots, a_k \}$  allora espressione regolare =  $(a_1 + a_2 + \dots + a_k)$

#### d) $i=j$

I cammini leciti sono il cammino di lunghezza zero e tutti i cicli da  $i$  a se stesso

a.  $R_{i,j}^{(0)} = \{\epsilon\} \rightarrow$  Cammino di lunghezza zero (Nessun simbolo sul percorso)

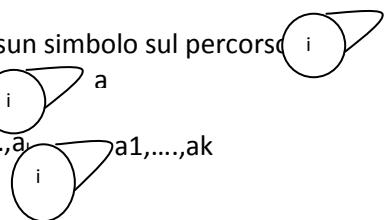
b.  $R_{i,j}^{(0)} = \{\epsilon, a\} \rightarrow$  Un solo simbolo sul percorso

c.  $R_{i,j}^{(0)} = \{\epsilon, a_1, a_2, \dots, a_k\} \rightarrow$  simboli da  $a_1, a_2, \dots, a_k$

-) Se  $R_{i,j}^{(0)} = \{\epsilon\}$  allora espressione regolare =  $\epsilon$

-) Se  $R_{i,j}^{(0)} = \{\epsilon, a\}$  allora espressione regolare =  $\epsilon + a$

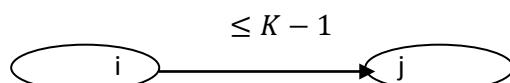
-) Se  $R_{i,j}^{(0)} = \{\epsilon, a_1, a_2, \dots, a_k\}$  allora espressione regolare =  $(\epsilon + a_1 + a_2 + \dots + a_k)$



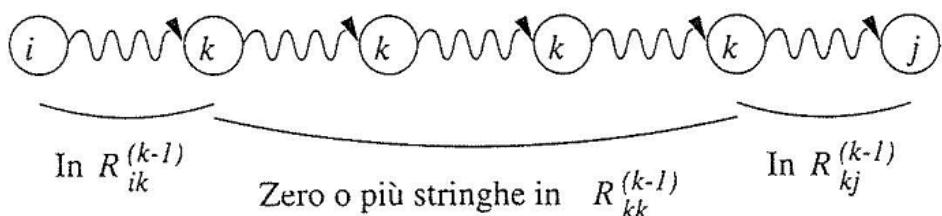
#### PASSO INDUTTIVO:(Suppongo vero l'enunciato per k-1 e lo dimostro per k)

Supponiamo che esista un cammino dallo stato  $i$  allo stato  $j$  che non attraversa nessuno stato superiore a  $k$ . Bisogna prendere in considerazione due casi:

- Il cammino non passa per lo stato  $k$ . In questo caso l'etichetta del cammino appartiene al linguaggio di  $R_{i,j}^{(k-1)}$



- Il cammino passa per lo stato  $k$  almeno una volta. Allora è possibile scomporlo in segmenti come segue:



- Il primo segmento va dallo stato  $i$  allo stato  $k$  senza passare per  $k$   $R_{i,k}^{(k-1)}$
- L'ultimo va da  $k$  a  $j$  senza passare per  $k$   $R_{k,k}^{(k-1)}$
- Tutti i segmenti intermedi vanno da  $k$  a  $k$  senza passare per  $k$   $R_{k,j}^{(k-1)}$

**N.B. Se il cammino passa attraverso  $k$  una sola volta non esistono segmenti mediani, ma solo un cammino da  $i$  a  $k$  e da  $k$  a  $j$ .**

L'etichetta per il cammino definito nel caso b è:

$$(R_{i,k}^{(k-1)}) \cdot (R_{k,k}^{(k-1)})^* (R_{k,j}^{(k-1)})$$

Con il seguente significato:

- d.  $(R_{i,k}^{(k-1)}) \rightarrow$  Rappresenta la parte del cammino che giunge allo stato k la prima volta
- e.  $(R_{k,k}^{(k-1)})^* \rightarrow$  Rappresenta la porzione che va da K a se stesso zero o più volte
- f.  $(R_{kj}^{(k-1)}) \rightarrow$  Rappresenta la parte del cammino che lascia K per l'ultima volta e va in j

Combinando le espressioni per i due casi otteniamo che:

$(R_{i,j}^{(k)}) = (R_{i,j}^{(k-1)}) + (R_{i,k}^{(k-1)}) \cdot (R_{k,k}^{(k-1)})^* (R_{k,j}^{(k-1)}) \rightarrow$  per le etichette di tutti i cammini dallo stato  $i$  allo stato  $j$  che non passano per gli stati più alti di  $k$ .

**Per definizione di Espressione Regolare: Esiste un'espressione regolare per  $R_{i,j}^{(k)}$**

Infine abbiamo che  $R_{i,j}^{(n)}$  e per ogni  $i$  e  $j$

Possiamo assumere che lo stato 1 sia lo stato iniziale, mentre gli stati accettanti possono essere qualunque insieme di stati. L'espressione regolare per il linguaggio dell'automa è allora la somma (unione) di tutte le espressioni  $R_{1,j}^{(n)}$  in cui  $j$  sia uno stato accettante (finale).

$$L(A) = \bigcup_{j \in F} R_{1,j}^{(n)} \rightarrow L(A) = \sum_{j=1; j \in F}^n R_{1,j}^{(n)} \rightarrow \text{ESPRESSONE REGOLARE } L(A)$$

- ✓ **(Corollario 1)** Un linguaggio  $L$  è regolare se e solo se esiste un'espressione regolare  $E$  tale che  $L=L(E)$
- ✓ **(Corollario 2)** Il linguaggio  $L$  è riconosciuto da un DFA se e solo se  $L$  è denotato da un'espressione regolare  $E$ .
- ✓ **(TEOREMA Conversione di espressioni regolari in automi)** Per ogni espressione regolare  $E$  esiste un  $\epsilon$ -NFA  $A$  tale che  $L(E) = L(A)$ . Ogni linguaggio definito da una espressione regolare è definito anche da un automa a stati finiti.

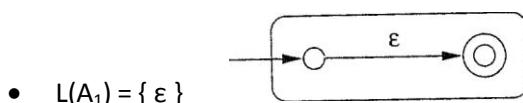
**(DIMOSTRAZIONE)** La dimostrazione è un'induzione strutturale sull'espressione  $E$ .

Cominciamo costruendo gli automi per le espressioni di base: singoli simboli,  $\epsilon$ ,  $\emptyset$ . Mostriamo poi come combinare questi automi in automi più grandi che accettano l'unione, la concatenazione o la chiusura dei linguaggi accettati dagli automi più piccoli. Tutti gli automi che costruiamo sono  $\epsilon$ -NFA con un unico stato accettante.

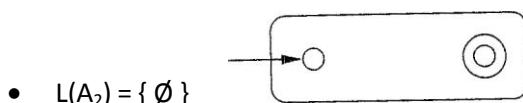
Per ogni espressione regolare  $E$  esiste un  $\epsilon$ -NFA  $A$  tale che:

1.  $A$  ha un solo stato finale  $q_f$ ;
2. Non esistono archi uscenti dallo stato finale  $q_f$ , ( $\delta(q_f, a) = \emptyset \forall a \text{ lettera}$ );
3. Non esistono archi entranti nello stato iniziale  $q_0$ , ( $\forall \text{ stato } q, \forall \text{ lettera } a, \delta(q, a) \neq q_0$ );
4.  $L(E) = L(A)$ .

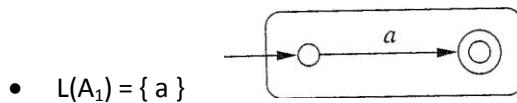
**(BASE)** Devo dimostrare che esiste un  $\epsilon$ -NFA  $A$  tale che le proprietà dalla "1" alla "4" siano vere per  $E = \emptyset$  oppure per  $E = \epsilon$  oppure  $E = a, a \in \Sigma$ .



Il linguaggio dell'automa è  $\{\epsilon\}$ , dato che l'unico cammino dallo stato iniziale verso uno stato accettante è etichettato  $\epsilon$ .



Il linguaggio dell'automa è  $\{ \emptyset \}$ , non esistono cammini dallo stato iniziale a allo stato accettante.



- $L(A_1) = \{ a \}$

Il linguaggio dell'automa è  $\{ a \}$ , rappresentato dall'automa per l'espressione regolare  $a$ .

E' facile verificare che tutti questi automi soddisfano le condizioni dalla "1" alla "3".

**(PASSO INDUTTIVO)** Per definizione l'espressione regolare  $E$  è costruita da sottoespressioni  $E_1, E_2$  o da  $E_1$ .

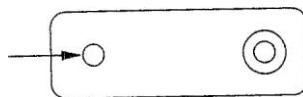
Per definizione di espressione regolare abbiamo i seguenti 4 casi:

1.  $E = E_1 + E_2$
2.  $E = E_1 \cdot E_2$
3.  $E = E_1^*$
4.  $E = (E_1)$

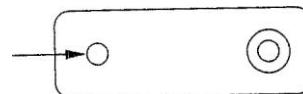
#### Caso 1:

$E = E_1 + E_2$  con  $E_1, E_2$  espressioni regolari.

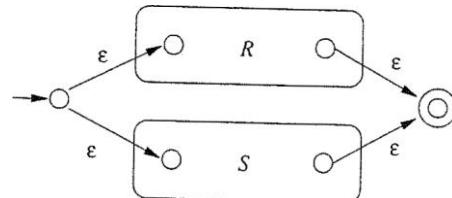
Per ipotesi induttiva esiste un  $\epsilon$ -NFA  $A_1$  tale che da "1" a "3" sono vere e  $L(A_1) = L(E_1)$ .



Esiste un  $\epsilon$ -NFA  $A_2$  tale che tale che da "1" a "3" sono vere e  $L(A_2) = L(E_2)$ .



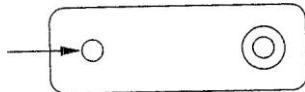
Allora,  $L(A) = L(A_1) \cup L(A_2) = L(E_1) \cup L(E_2) = L(E_1 + E_2) = L(E)$



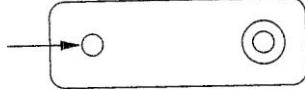
L'espressione è  $R + S$  per due espressioni più piccole  $R$  ed  $S$ . In altre parole, partendo dal nuovo stato iniziale, possiamo andare nello stato iniziale dell'automa per  $R$  oppure in quello dell'automa per  $S$ . Raggiungiamo poi lo stato accettante di uno di questi automi seguendo un cammino etichettato da una stringa che si trova rispettivamente in  $L(R)$  oppure  $L(S)$ . Una volta raggiunto lo stato accettante dell'automa per  $R$  o  $S$ , possiamo seguire uno degli archi  $\epsilon$  verso lo stato accettante del nuovo automa. Il linguaggio dell'automa è perciò  $L(R) \cup L(S)$ .

#### Caso 2: $E = E_1 \cdot E_2$ con $E_1, E_2$ espressioni regolari.

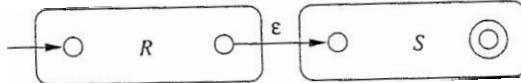
Per ipotesi induttiva esiste un  $\epsilon$ -NFA  $A_1$  tale che da "1" a "3" sono vere e  $L(A_1) = L(E_1)$ .



Esiste un  $\epsilon$ -NFA  $A_2$  tale che da "1" a "3" sono vere e  $L(A_2) = L(E_2)$ .



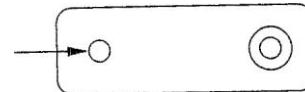
Allora  $L(A) = L(A_1)$   $L(A_2) = L(E_1)$   $L(E_2) = L(E_1 E_2) = L(E)$



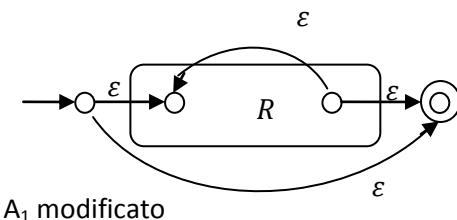
L'espressione è  $RS$  per due espressioni più piccole  $R$  ed  $S$ . Si noti che lo stato iniziale del primo automa diventa il nuovo stato iniziale, e lo stato accettante del secondo automa diventa lo stato accettante dell'automa complessivo. L'idea è che un cammino dallo stato iniziale a quello accettante deve attraversare prima l'automa per  $R$  seguendo un cammino etichettato da una stringa in  $L(R)$ , e poi quello per  $S$  seguendo un cammino etichettato da una stringa in  $L(S)$ . Quindi i cammini nell'automa sono tutti e soli quelli etichettati dalle stringhe in  $L(R)L(S)$ .

**Caso 3:**  $E = E_1^*$  con  $E_1$  espressione regolare.

Per ipotesi induttiva esiste un  $\epsilon$ -NFA  $A_1$  tale che da "1" a "3" sono vere e  $L(A_1) = L(E_1)$ .



Ricordiamo  $L(A) = L(A_1^*) = L(E_1^*) = L(E)$



L'espressione è  $E^*$  per un'espressione più piccola  $E$ . L'automa che offre due tipi di percorso.

- (c) Direttamente dallo stato iniziale allo stato accettante lungo un cammino etichettato  $\epsilon$ . Tale cammino fa accettare  $\epsilon$ , che si trova in  $L(R)$  a prescindere da  $R$ .
- (d) Verso lo stato iniziale dell'automa per  $R$ , attraverso tale automa una o più volte, e poi verso lo stato accettante. Questo insieme di cammini ci permette di accettare stringhe in  $L(R)$ ,  $L(R)L(R)$ ,  $L(R)L(R)L(R)$ , e così via, coprendo così tutte le stringhe in  $L(R^*)$ , eccetto eventualmente  $\epsilon$ , già coperta dall'arco diretto verso lo stato accettante.

**Caso 4:**  $E = (E_1)$  con  $E_1$  espressione regolare.

Ricordiamo che  $L(E) = L((E_1)) = L(E_1)$ .

Quindi  $A = (A_1)$  verifica le condizioni da "1" a "4"

Infatti  $A_1$  verifica le condizioni da "1" a "3" per ipotesi induttiva e  $L(A_1) = L(E_1) = L((E_1)) = L(E)$ .

L'espressione è  $(E)$  per un'espressione più piccola  $E$ . Dato che le parentesi non cambiano il linguaggio definito dall'espressione, l'automa per  $R$  serve anche come automa per  $(R)$ .

## Grammatiche e linguaggi CF

- ✓ **(Def. Grammatica CF):** Una grammatica Context-Free  $G$  è una quadrupla  $G = (V, T, P, S)$  dove:
  - $V$  è l'insieme *finito* delle variabili
  - $T$  è l'insieme *finito* dei terminali,  $T \cap V = \emptyset$
  - $S$  è un elemento di  $V$  (start symbol)

$P \subseteq V \times (V \cup T)^*$ ,  $P$  insieme finito,  $P$  produzione di  $G$ . Un elemento di  $P$  è rappresentato da  $A \rightarrow \alpha$ ,  $A \in V$ ,  $\alpha \in (V \cup T)^*$ .
- ✓ **(Def. DEVIAZIONE DIRETTA)** Sia  $G = (V, T, P, S)$  una grammatica C.F., siano  $\alpha, \beta \in (V \cup T)^*$ . Diremo che si deriva direttamente  $\beta$  in  $G$  e scriveremo  $\alpha \Rightarrow \beta$  se esistono  $\gamma_1, \gamma_2 \in (V \cup T)^*$  e  $A \Rightarrow \gamma \in P$  tale che  $\alpha \Rightarrow \gamma_1 A \gamma_2$  e  $\beta \Rightarrow \gamma_1 \gamma_2$ .
- ✓ **(Def. Derivazioni per mezzo di una grammatica)** Supponiamo che  $G = (V, T, P, S)$  sia una grammatica CF. Sia  $\alpha A \beta$  una stringa di terminali e variabili, dove  $A$  è una variabile. In altre parole  $\alpha$  e  $\beta$  sono stringhe in  $(V \cup T)^*$ , e  $A$  è in  $V$ .  
 Sia  $A \xrightarrow{\gamma}$  una produzione di  $G$ . Allora scriviamo  $\alpha A \beta \Rightarrow \alpha \gamma \beta$   
 Si noti che un passo di derivazione sostituisce una variabile in un punto qualsiasi della stringa con il corpo di una delle sue produzioni.  
 Possiamo estendere la relazione  $\Rightarrow$  fino a farle rappresentare zero, uno o più passi di derivazione, analogamente a come la funzione di transizione  $\delta$  di un automa a stati finiti si estende a  $\hat{\delta}$ . Per le derivazioni usiamo il simbolo  $\xrightarrow{*}$  per denotare "zero o più passi", come segue:  
**(BASE):** Per qualsiasi stringa  $\alpha$  di terminali e variabili,  $\alpha \xrightarrow{*} \alpha$ . In altri termini: qualunque stringa deriva se stessa.  
**(INDUZIONE):** Se  $\alpha \xrightarrow{*} \beta$  e  $\beta \Rightarrow \gamma$ , allora  $\alpha \xrightarrow{*} \gamma$  con  $\alpha, \beta, \gamma \in (V \cup T)^*$ . Ossia, se  $\alpha$  può diventare  $\beta$  in zero o più passi, e un passo ulteriore trasforma  $\beta$  in  $\gamma$ , allora  $\alpha$  può diventare  $\gamma$ .  
 Detto in altri termini,  $\alpha \xrightarrow{*} \beta$  significa che esiste una sequenza di stringhe  $\gamma_1, \gamma_2, \dots, \gamma_n$  con  $n \geq 1$ , tale che:
  4.  $\alpha = \gamma_1;$
  5.  $\beta = \gamma_n;$
  6. per  $i = 1, 2, \dots, n-1$ , abbiamo  $\gamma_i \Rightarrow \gamma_{i+1}$ .
- ✓ **(Def. Left most):** Per ridurre il numero di scelte possibili nella derivazione di una stringa, spesso è comodo imporre che a ogni passo si sostituisca la variabile all'estrema sinistra con il corpo di una delle sue produzioni. Tale derivazione viene detta *derivazione a sinistra (leftmost derivation)*, e si indica tramite le relazioni  $\xrightarrow{lm}$  e  $\xrightarrow{lm}^*$ , rispettivamente per uno o molti passi. Se la grammatica  $G$  in esame non è chiara dal contesto, possiamo porre il nome  $G$  sotto la freccia.
- ✓ **(Def. right most):** Analogamente è possibile imporre che a ogni passo venga sostituita la variabile più a destra con uno dei suoi corpi. In tal caso chiamiamo la derivazione *a destra (rightmost)*, e usiamo i simboli  $\xrightarrow{rm}$  e  $\xrightarrow{rm}^*$  per indicare rispettivamente uno o più passi di derivazione a destra. Qualora non fosse evidente, il nome della grammatica può comparire anche in questo caso sotto i simboli.
- ✓ **(Def. Di linguaggio generato da una grammatica)** Sia  $G = (V, T, P, S)$  una grammatica C.F.. Il linguaggio  $L(G)$  generato dalla grammatica  $G$  è definito da:  $L(G) = \{ w \in T^* \mid S \xrightarrow{*} w \}$
- ✓ **(Def. Linguaggio CF)** Un linguaggio  $L \subseteq T^*$  è context-free se esiste una grammatica  $G$  context-free tale che  $L=L(G)$
- ✓ **(Def. Forma sentenziale)** Se  $\alpha \in (V \cup T)^*$  e  $S \xrightarrow{*} \alpha$  (in  $G = (V, T, P, S)$ ) allora  $\alpha$  è una forma sentenziale (sinistra\destra)

- ✓ **(Proprietà 1):** Sia  $G = (V, T, P, S)$  una grammatica C.F., siano  $A \in V$ ,  $\alpha \in (V \cup T)^*$ . Se  $A \xrightarrow{*} \alpha$ ,  $\forall \gamma_1, \gamma_2 \in (V \cup T)^*$  risulta  $\gamma_1 A \gamma_2 \Rightarrow \gamma_1 \alpha \gamma_2$ .
- ✓ **(Proprietà 2):** Sia  $G = (V, T, P, S)$  una grammatica C.F., siano  $x_1, \dots, x_k \in (V \cup T)$ , sia  $w \in T^*$ . Se  $x_1 \dots x_k \xrightarrow{*} w$  (in  $n$  passi) allora esistono  $w_1, \dots, w_k \in T^*$  tali che  $w = w_1 \dots w_k$ ,  $\forall j \in \{1, \dots, k\}$   $x_j \xrightarrow{*} w_j$  (in  $n'$  passi,  $n' \leq n$ ) e se  $x_j \in T$  allora  $x_j = w_j$ .
- ✓ **(Def. Costruzione di alberi sintattici):** Fissiamo una grammatica  $G = (V, T, P, S)$ . Gli alberi sintattici di  $G$  sono alberi che soddisfano le seguenti condizioni.
  1. Ciascun nodo interno è etichettato da una variabile in  $V$ .
  2. Ciascuna foglia è etichettata da una variabile, da un terminale, o da  $\epsilon$ . Se una foglia è etichettata  $\epsilon$ , deve essere l'unico figlio del suo genitore.
  3. Se un nodo interno è etichettato  $A$  e i suoi figli sono etichettati, a partire da sinistra,  $X_1, X_2, \dots, X_k$  allora  $A \rightarrow X_1, X_2, \dots, X_k$ , è una produzione in  $P$ . Si noti che un  $X$  può essere  $\epsilon$  solo nel caso in cui è l'etichetta di un figlio unico, e quindi  $A \rightarrow \epsilon$  è una produzione di  $G$ .
- ✓ **(Def. Frontiera):** La frontiera di un albero sintattico è uguale alla concatenazione delle etichette associate alle foglie dell'albero  $T$  (lette da sinistra a destra). La frontiera è un elemento di  $(V \cup T)^*$ .
- ✓ **(Def. Prodotto di un albero sintattico):** Se concateniamo le foglie di un albero sintattico a partire da sinistra otteniamo una stringa, detta il *prodotto* dell'albero, che è sempre una stringa derivata dalla variabile della radice. Dimostreremo questa asserzione fra poco. Di particolare importanza sono gli alberi sintattici che soddisfano queste due condizioni.
  - Il prodotto è una stringa terminale. In questo caso tutte le foglie sono etichettate da un terminale o da  $\epsilon$ .
  - La radice è etichettata dal simbolo iniziale.
- ✓ **(TEOREMA):** Sia  $G = (V, T, P, S)$  una grammatica context-free. Sono equivalenti le affermazioni seguenti:
  1.  $A \xrightarrow{*} w$   $A \in V$ ,  $w \in T^*$
  2. Esiste un parse tree (albero sintattico) di radice  $A$  e frontiera  $w$ ,  $w \in T^*$
  3.  $A \xrightarrow[LM]{*} w$ ,  $w \in T^*$

#### DIMOSTRAZIONE:

1)  $\Rightarrow$  2) : la 1) implica la 2) ovvero (se 1 allora 2).

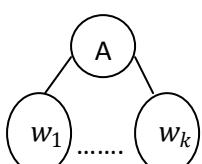
La dimostrazione viene effettuata per induzione. (enunciato: la parola  $w$  si può derivare e vogliamo dimostrare che esiste il parse tree).

Supponiamo che  $A \xrightarrow{*} w$ , con  $w \in T^*$ ,  $A \in V$ . Dimostriamo che esiste un albero sintattico di radice  $A$  e frontiera  $w$  per induzione sul numero di passi nella derivazione di  $w$  da  $A \xrightarrow{*} w$ .

(BASE)

Supponiamo  $A \Rightarrow w$  e dimostriamo che esiste l'albero sintattico di radice  $A$  e frontiera  $w$ .

Siccome  $A \Rightarrow w$ , per definizione  $\exists$  la produzione (P)  $A \rightarrow w_1, w_2, \dots, w_k = w$ ,  $w \in P$   $w_i \in T$  e



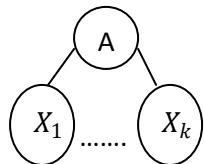
è un albero sintattico la cui frontiera è  $w$ .

(PASSO INDUTTIVO)

Supponiamo  $A \xrightarrow{*} w$   $w \in T^*$  in  $n$  passi con  $n > 1$  e assumiamo vero l'enunciato per derivazione in numero di passi  $<$  di  $n$ .

Siccome  $A \xrightarrow{*} w$  in  $n$  passi con  $n > 1$ .

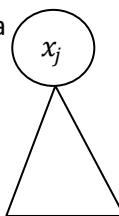
$$A \Rightarrow X_1 X_2 \dots X_k \xrightarrow{*} w \text{ con } A \rightarrow X_1 X_2 \dots X_k \in P \quad X_j \in T \cup V$$



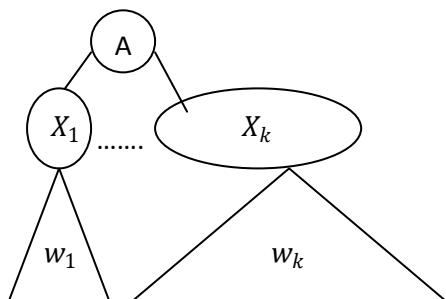
Applichiamo la proprietà 2) delle grammatiche context-free.

Quindi  $w = w_1 w_2 \dots w_k$  dove  $x_j \Rightarrow w_j$  in  $n' < n$  passi con  $1 \leq j \leq k$  e se  $X_j \in T$  allora  $X_j = w_j$

Dimostrazione 1)  $\Rightarrow$  2) nel caso se  $X_j \notin T$ , per ipotesi induttiva



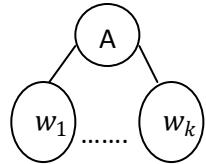
Quindi:



E ho trovato un parse tree di radice A e frontiera w. Quindi ho dimostrato 1)  $\Rightarrow$  2)

**Dimostriamo** che 2)  $\Rightarrow$  3) per induzione sull'altezza dell'albero sintattico.

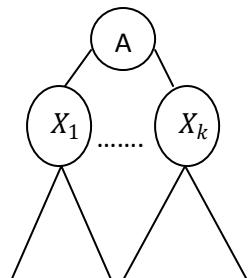
(BASE) Sia  $T$  un albero sintattico di radice A, frontiera  $w \in T^*$ , e altezza 1.



con  $w_j \in T$  e  $w_1 w_2 \dots w_k = w$ . Per definizione di parse tree  $A \rightarrow w_1 w_2 \dots w_k \in P$ . Quindi  $A \Rightarrow w_1 w_2 \dots w_k$  cioè  $A \xrightarrow{LM}^* w_1 w_2 \dots w_k = w$

(PASSO INDUTTIVO)

$\exists$  un parse tree di radice A, frontiera  $w \in T^*$  di altezza  $n > 1$



Siccome è un parse tree allora  $A \rightarrow X_1 X_2 \dots X_k \in P$ .

Inoltre se  $w_j \neq x_j$  allora  $w_j$  è frontiera di un parse tree di radice  $x_j$  e di altezza minore di n.

Per ipotesi induttiva  $x_j \xrightarrow{LM}^* w_j$ .

Siccome  $A \rightarrow X_1X_2 \dots X_k \in P$  allora  $A \xrightarrow{LM} X_1X_2 \dots X_k \xrightarrow{LM} w_1X_2 \dots X_k \xrightarrow{LM}^* \dots \xrightarrow{LM} w$  (per induzione, usando la (1)).

Quindi  $A \xrightarrow{LM}^* w$  che dimostra anche 3)  $\Rightarrow 1)$ .

Definizione:

Una grammatica context-free  $G = (V, T, P, S)$  è ambigua se esiste  $w \in T^*$  e due alberi sintattici di radice  $S$  e frontiera  $w$ .

(Proprietà)

Una grammatica context-free  $G = (V, T, P, S)$  è ambigua se e solo se una stringa di terminali ( $w \in T^*$ ) che permette due o più derivazioni *leftmost* da  $S$ .