Linguaggi formali e compilazione

Corso di Laurea in Informatica

A.A. 2012/2013

Linguaggi formali e compilazione

Obiettivo di questa parte del corso

- Descriveremo gli automi a stati finiti, che sono importanti *strumenti modellistici* che trovano amplissima applicazione in molti settori dell'Informatica.
- Vedremo come gli automi finiti siano un formalismo “equivalente” alle espressioni regolari.
- Ci prepareremo a comprendere il funzionamento di strumenti automatici che (come Lex) possono essere di grande ausilio nella realizzazione di compilatori.

Definizione informale

- Un *automa a stati finiti deterministico* (*ASFD*), può essere visto come un calcolatore elementare dotato di stato interno e supporto unidirezionale di input.
- Il funzionamento dell'automa consiste di transizioni di stato a seguito della lettura di un simbolo da un dispositivo di input.
- Ad ogni stato q sono in generale associate azioni (come la stampa di messaggi) che l'automa esegue quando transita in q.
- Gli ASFD sono ampiamente utilizzati in molti settori dell'Informatica, non solo nel contesto dei linguaggi formali.

Descrizione formale

Un *ASFD* M è una quintupla

$$M = (\Sigma, Q, q_0, Q_f, \delta),$$

in cui

- Σ è l'alfabeto di input;
- Q è un insieme finito i cui elementi sono detti *stati* dell'automa;
- q_0 è un elemento speciale di Q, detto *stato iniziale*;
- $Q_f \subseteq Q$ è l'insieme degli *stati finali*, detti anche di *accettazione* dell'input;
- δ è la funzione che determina le *transizioni* di stato. Essa mappa coppie $\langle stato, simbolo \rangle$ in stati: $\delta : Q \times \Sigma \to Q$.

Computazioni di un automa

- Definibili in modo intuitivo come sequenza di *passi*.
- Ad ogni passo, l'automa si trova in uno stato q (inizialmente $q = q_0$), legge un simbolo x dall'input e transita nello stato $\delta(q, x)$.
- La computazione termina al verificarsi di una delle seguenti situazioni:

  mancanza di input: non vi sono più simboli di input, oppure

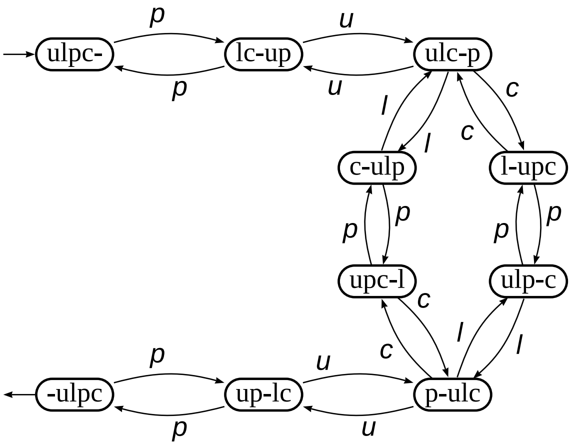
  transizione non specificata: in corrispondenza dello stato attuale e del simbolo letto, la funzione di transizione non è specificata.
- Il numero di transizioni effettuate prima della terminazione è detto *lunghezza* della computazione e ne rappresenta una misura del costo.

Rappresentazione di automi

- Un utile formalismo (molto diffuso perché “intuitivo”) è quello dei *diagrammi di transizione*.
- Un diagramma di transizione è un grafo i cui nodi ed archi rappresentano, rispettivamente, stati e transizioni.
- Ogni arco è etichettato da un simbolo di input.
- Lo stato iniziale viene evidenziato mediante una freccia entrante (e non uscente da alcun altro nodo).
- Gli stati finali sono indicati tramite doppia cerchiatura oppura da una freccia uscente (e non entrante in alcun altro nodo).

Esempio introduttivo

Il lupo, la pecora e il cavolo



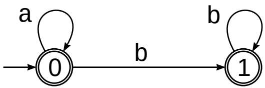
Esempio tratto da Hopcroft, Ullman (1979)

Automi riconoscitori

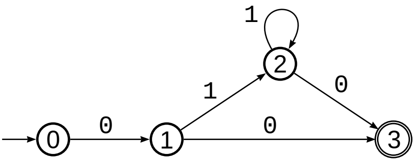
- Un *ASFD riconosce* (o *accetta*) una stringa X in input se la computazione determinata dai caratteri di X termina in uno stato di Q_f per mancanza di input.
- Un *ASFD M riconosce un linguaggio $\mathcal{L}$* se e solo se $\mathcal{L}$ coincide con l'insieme delle stringhe riconosciute da M.
- Ad esempio, nel caso del semplice rompicapo "Lupo, pecora e cavolo", la sequenza (stringa di input) `pulpcup` è riconosciuta dall'automa, mentre la stringa `pulcpup` non lo è.

Esempi

- Il seguente *ASFD* M_5 riconosce il linguaggio $L_5 = \{a^n b^m | n, m \geq 0\} = \mathbf{a}^*\mathbf{b}^*$

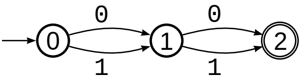


- Il seguente *ASFD* M_3 riconosce il linguaggio $L_3 = \{\mathtt{0}\mathtt{1}^k\mathtt{0} | k \geq 0\} = \mathbf{01}^*\mathbf{0}$

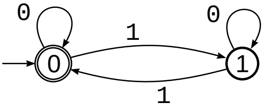


Esempi

- Il seguente *ASFD* M_2 riconosce il linguaggio $L_2 = \{X \in \mathcal{B}^* : |X| = 2\}$



- Il seguente *ASFD* M_{parity} riconosce il cosiddetto *linguaggio parità*, ovvero l'insieme delle stringhe $X \in \mathcal{B}^*$ che contengono un numero pari di 1



Rappresentazione di un ASFD

- La rappresentazione di un ASFD coincide "essenzialmente" con la rappresentazione della funzione di transizione δ.
- Questa può essere semplicemente data come *tabella*, con $m = |Q|$ righe ed $n = |\Sigma|$ colonne.
- Il consumo di memoria $\Theta(nm)$ può essere eccessivo se dalla maggior parte dei nodi del grafo di transizione escono relativamente pochi archi.
- Tuttavia, almeno per il momento, non ci interessiamo al problema dell'efficienza.

Simulazione di automi deterministici

- La simulazione del comportamento di un ASFD $M = (\Sigma, Q, q_0, Q_f, \delta)$ è particolarmente semplice.
- L'algoritmo riceve in ingresso M e l'input X per M e produce l'output che darebbe M su input X.
- L'algoritmo presentato nella diapositiva seguente si riferisce alla simulazione di un generico automa riconoscitore.
- Nella descrizione dell'algoritmo (in pseudocodice) si suppone che:
  - l'input X sia terminato dal carattere \$;
  - tale carattere non appartiene all'alfabeto Σ dell'automa;
  - se, per una determinata coppia stato-simbolo, $\langle q, x \rangle$, la funzione di transizione è indefinita, si pone $\delta(q, x) = \perp$.

Simulazione di un ASFD: algoritmo ASFD-Sim

$q \leftarrow q_0$
$x \leftarrow \text{nextchar}(X)$
while $(x \neq \$)$ **do**
  if $\delta(q, x) \neq \perp$ **then**
    $q \leftarrow \delta(q, x)$
  else
    reject
  $x \leftarrow \text{nextchar}(X)$
if $q \in Q_f$ **then**
  accept
else
  reject

Simulazione di automi deterministici

- Si noti che l'algoritmo è un vero e proprio *interprete*, ancorché molto semplice.
- Infatti, esso prende in input un programma M (l'automa) e un input X per il programma, ed "esegue" M su input X.
- È facile convincersi del fatto che il costo della simulazione è lineare nella lunghezza dell'input (a patto che si possa considerare costante il costo di valutazione della funzione δ, che tipicamente viene implementata mediante una tabella).

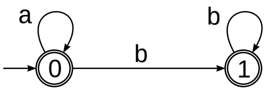
Esercizi proposti

- Per ciascuno dei seguenti linguaggi, si fornisca un ASFD che riconosce il linguaggio.
  - $\{X | X \in \{\mathtt{0},\mathtt{1}\}^*, X \text{ non contiene } \mathtt{0} \text{ adiacenti}\}$
  - $\{X | X \in \{\mathtt{0},\mathtt{1}\}^*, \text{ogni sottostringa di lunghezza 3 in } X \text{ contiene almeno due } \mathtt{1}\}$
  - $\{X | X \in \{\mathtt{a},\mathtt{b},\mathtt{c}\}^*, \text{due qualsiasi caratteri adiacenti in } X \text{ sono fra loro differenti}\}$
- Progettare un ASFD che riconosce il linguaggio descritto dalla seguente *definizione regolare*:

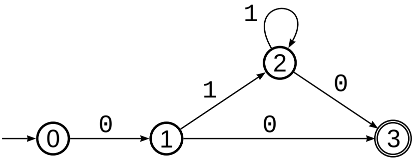
$$
\begin{aligned}
\mathit{NZD} &= 1|2|3|4|5|6|7|8|9 \\
D &= 0|\mathit{NZD} \\
\mathit{IP} &= \mathit{NZD}\,D^*|0 \\
\mathit{OPS} &= -|+|\epsilon \\
M &= \mathit{OPS}\,\mathit{IP}.D^* \\
\mathit{EXP} &= \mathbf{E}\,\mathit{OPS}\,\mathit{IP} \\
\mathit{FLOAT} &= M|M\,\mathit{EXP}
\end{aligned}
$$

Espressioni regolari e automi

- Data un'espressione regolare $\mathcal{E}$, è possibile definire un automa $\mathcal{M}(\mathcal{E})$ che riconosce il linguaggio definito da $\mathcal{E}$, e viceversa.
- Riconsideriamo qualche esempio già visto.
- Il linguaggio $L_5 = \mathbf{a}^*\mathbf{b}^*$ e l'automa M_5

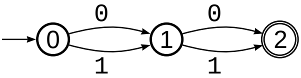


- Il linguaggio $L_3 = \mathbf{01}^*\mathbf{0}$ e l'automa M_3

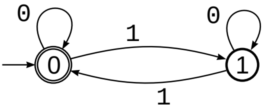


Espressioni regolari e automi

- Il linguaggio $L_2 = (\mathbf{0} + \mathbf{1})(\mathbf{0} + \mathbf{1})$ e l'automa M_2



- Il linguaggio "parità", $L_{parity} = \mathbf{0}^* (\mathbf{1}\mathbf{0}^*\mathbf{1}\mathbf{0}^*)^*$, e l'automa M_{parity}



- Quale automa corrisponde all'espressione regolare $\mathbf{a}^*\mathbf{b}\mathbf{c}^* + \mathbf{c}^*\mathbf{a}^*\mathbf{b}$?

Automi non deterministici

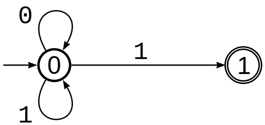
- Data un'espressione regolare $\mathcal{E}$, abbiamo detto (anche se non dimostrato) che esiste un automa $\mathcal{M}(\mathcal{E})$ ad essa "equivalente", cioè che riconosce lo stesso linguaggio definito dall'espressione.
- La trasformazione da espressione ad automa si può automatizzare e tale automatizzazione risulta più" facile se la si suddivide in due passi:
  - dall'espressione regolare ad un *automa non deterministico* equivalente;
  - dall'automa non deterministico all'automa deterministico equivalente.

Definizione di automa non deterministico

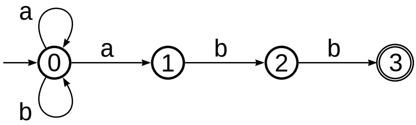
- Un automa a stati finiti si dice *non deterministico* (*ASFND*) se, in almeno uno stato q, la transizione *non è univocamente determinata* dal simbolo di input.
- In altri termini, dallo stato q e con lo stesso simbolo di input l'automa può transitare "non deterministicamente" in più di uno stato diverso.
- Nella definizione formale cambia solo la "funzione" di transizione, che, in un *ASFND*, mappa coppie $\langle$stato,simbolo$\rangle$ in *sottoinsiemi* (anziché elementi) di Q.

Esempi

- Il seguente automa è non deterministico perché nello stato 0 ci sono due transizioni etichettate con il simbolo 1. In altri termini, la funzione di transizione mappa la coppia $\langle 0, 1 \rangle$ nell'insieme $\{0, 1\}$



- Un altro esempio di ASFND:



Riconoscimento di stringhe da parte di un ASFND

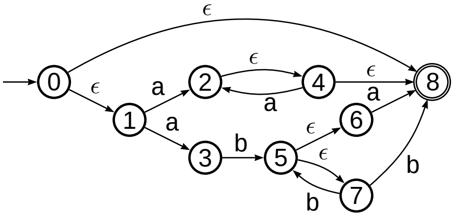
- Si dice che un *ASFND* M riconosce una stringa X se e soltanto se *esiste* una sequenza di transizioni etichettata con i simboli di X che termina in uno stato finale.
- È facile vedere che il primo automa della precedente trasparenza riconosce la stringa in input solo se questa termina con 1.
- Si può anche facilmente dimostrare che, per ogni tale stringa, esiste una sequenza di transizioni (mosse) che porta l'automa nello stato 1.
- Possiamo quindi concludere che l'automa riconosce il linguaggio $(\mathbf{0}|\mathbf{1})^*\mathbf{1}$.

Riconoscimento di stringhe da parte di un ASFND (continua)

- Si noti come nello stato 0, e con input 1, l'automa debba decidere non deterministicamente se transitare nello stato 1 o restare nello stato 0.
- Questo equivale a dire che l'automa deve decidere se è stato letto l'ultimo carattere 1.
- L'automa del secondo esempio riconosce invece il linguaggio $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}$.
- Nello stato 0 e su input a, l'automa deve decidere se quella appena letta è l'ultima a nella stringa di input.

ϵ-transizioni

- Una particolare forma di non determinismo è data dalle cosiddette ϵ-transizioni, cioè transizioni che mappano elementi di $Q \times \{\epsilon\}$ in Q.
- Il concetto si capisce bene osservando il diagramma di transizione: se l'arco che collega due nodi q ed r è etichettato da ϵ, allora l'automa può passare da q ad r "senza consumare input".
- Il seguente diagramma costituisce un primo esempio di ASFND con ϵ-transizioni.

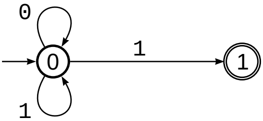


Equivalenza di ASFD e ASFND

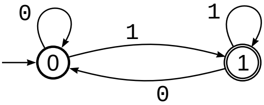
- Un risultato fondamentale nella teoria degli automi (con importanti ripercussioni anche nella costruzione di compilatori) afferma che, se un linguaggio $\mathcal{L}$ è riconoscibile da un ASFND, allora $\mathcal{L}$ è riconoscibile da un ASFD che impiega essenzialmente lo stesso tempo.
- Naturalmente il viceversa è banalmente vero, in quanto gli automi non deterministici generalizzano quelli deterministici.
- Il risultato citato (che dimostreremo) prova quindi che automi finiti deterministici e non deterministici sono equivalenti.

Primi esempi

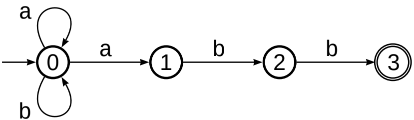
- Automi deterministici equivalenti ai primi ASFND visti come esempio sono illustrati di seguito.
- Automi che riconoscono il linguaggio $(\mathtt{0}|\mathtt{1})^*\mathtt{1}$:
  - Automa non deterministico



  - Automa deterministico equivalente



- Automi che riconoscono il linguaggio $(\mathtt{a}|\mathtt{b})^*\mathtt{abb}$
  - Automa non deterministico



La costruzione dell’automa deterministico: idee

- Gli esempi appena visti sono stati costruiti in modo “ad-hoc”.
- Ciò di cui abbiamo bisogno è invece di un processo automatizzabile per passare da un ASFND $\mathcal{N}$ ad un ASFD $\mathcal{D}$ equivalente ad $\mathcal{N}$.
- Il processo di costruzione che vedremo è noto come *subset construction*.
- L’idea è semplice: dato $\mathcal{N}$, l’automa equivalente in qualche modo “simula” $\mathcal{N}$ tenendo traccia degli stati in cui può trovarsi $\mathcal{N}$ dopo aver letto i simboli di input ($i = 0, 1, \ldots$).

La costruzione dell'automa deterministico: idee (continua)

- Se Q è l'insieme degli stati di $\mathcal{N}$, allora dopo la lettura di i simboli l'automa può trovarsi (in linea di principio) in uno qualunque degli stati di Q.
- Ad esempio, supponiamo che $Q = \{0, 1, 2, 3\}$ e che, dopo aver letto due simboli in input, $\mathcal{N}$ possa trovarsi indifferentemente (o meglio, non deterministicamente) nello stato 1 oppure nello stato 3.
- In questo caso, l'automa deterministico equivalente $\mathcal{D}$ avrà, fra gli altri, uno stato che corrisponde al sottoinsieme $\{1, 3\}$.
- Ovviamente, se $|Q| = m$ allora il numero di stati distinti di $\mathcal{D}$ sarà al più 2^m.

Subset construction

- Sia $\mathcal{N} = (\Sigma, Q, q_0, Q_f, \delta)$ un dato ASFND e sia $\mathcal{D} = (\Sigma, DS, DQ_0, DQ_f, DT)$ l'automa equivalente (che andremo a costruire).
- La costruzione mostra come sono definite le varie componenti di $\mathcal{D}$ (l'alfabeto di input Σ è naturalmente lo stesso).
- La costruzione procede aggiungendo nuovi stati all'insieme DS sulla base delle transizioni che sono possibili, in $\mathcal{N}$, sui vari simboli di input.
- Il primo stato che viene aggiunto a DS rappresenta lo stato iniziale, q_0, di $\mathcal{N}$ unitamente a tutti gli stati che sono raggiungibili da q_0 senza leggere simboli dall'input.

Subset construction (continua)

- L'insieme di tutti gli stati raggiungibili da uno dato stato q mediante ϵ-transizioni (incluso q stesso) viene indicato con ϵ-$CLOSURE(q)$ (ϵ-chiusura di q_0).
- La ϵ-$CLOSURE(q_0)$ rappresenta dunque, in $\mathcal{D}$, l'insieme degli stati nei quali può trovarsi $\mathcal{N}$ senza aver letto alcun simbolo di input.
- In questo modo abbiamo un primo elemento in DS e il processo continua esaminando gli stati già inseriti in DS.
- La ϵ-$CLOSURE(q_0)$ è anche lo stato iniziale (che abbiamo indicato con DQ_0) di $\mathcal{D}$.
- Si noti che, se Q è un insieme di stati, possiamo definire ϵ-$CLOSURE(Q)$ in modo molto naturale come segue:

$$\epsilon\text{-}CLOSURE(Q) = Q \bigcup \left(\bigcup_{q \in Q} \epsilon\text{-}CLOSURE(q) \right).$$

Subset construction (continua)

- Consideriamo ora un generico stato $Q' = \{q_1, q_2, \ldots q_k\} \in DS$ che non sia ancora stato "esaminato", dove naturalmente i q_j sono stati di $\mathcal{N}$.
- Per ogni simbolo $x \in \Sigma$ formiamo dapprima l'insieme Q'' degli stati nei quali può transitare $\mathcal{N}$ partendo da uno stato degli stati $q_j \in Q'$ a seguito della lettura di x.
- In formule:

$$Q'' = \delta(q_1, x) \cup \ldots \cup \delta(q_k, x)$$

- Viene la "tentazione" di affermare che inquesto modo abbiamo ottenuto un altro stato da aggiungere a DS (Q'' appunto), ma c'è ancora qualcosa di cui tenere conto.

Subset construction (continua)

- Infatti, dopo aver eseguito una transizione da q_j a $\delta(q_j, x)$, $\mathcal{N}$ potrebbe, senza leggere ulteriori simboli, muoversi su uno stato collegato a $\delta(q_j, x)$ mediante ϵ-transizioni.
- Quel che dobbiamo fare è dunque considerare l'unione di tutti gli stati contenuti nelle ϵ-*CLOSURE* degli stati di Q''.
- Ma questa è esattamente la $\epsilon\text{-}CLOSURE(Q'')$.
- Questo è precisamente il nuovo stato che viene aggiunto a *DS* (sempre che non sia già presente).
- Inoltre si pone

$$DT(Q', x) = \epsilon\text{-}CLOSURE(Q'').$$

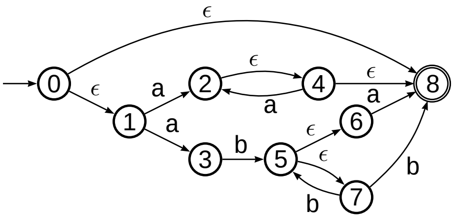
Subset construction (continua)

- Come già osservato, il processo va avanti finché esiste almeno uno stato di DS che non sia ancora stato esaminato.
- Abbiamo comunque la certezza che il processo di aggiunta di stati termini perché il numero di sottoinsiemi distinti di stati di $\mathcal{N}$ è finito (al più aggiungeremo 2^m stati).
- Come ultimo passo dobbiamo individuare quali siano gli stati terminali di $\mathcal{D}$.
- Questi saranno tutti e soli gli stati che “contengono” almeno uno stato finale di $\mathcal{N}$: si pone cioè

$$DQ_f = \{DQ \in DS | \exists q \in DQ \text{ t. c. } q \in Q_f\}$$

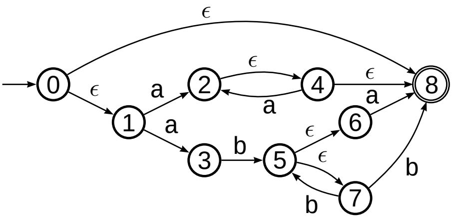
Esempio di subset construction

- Consideriamo il seguente automa non deterministico, già introdotto a proposito delle ϵ-transizioni:



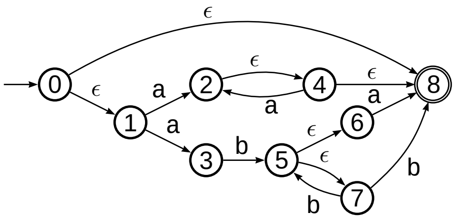
- Se indichiamo con A lo stato iniziale di $\mathcal{D}$, avremo $A = \{0, 1, 8\}$.
- Si noti infatti che $\{0, 1, 8\} = \epsilon\text{-}CLOSURE(0)$.

Esempio di subset construction (continua)



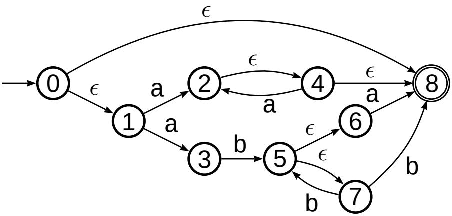
- Esaminiamo ora, a partire dagli stati di A, in quali stati si arriva su input a. Tali stati sono dapprima 2 e 3 ma poi, considerando le ϵ-transizioni, anche gli stati 4 e 8 (vale cioè $\epsilon\text{-}CLOSURE(\{2,3\}) = \{2,3,4,8\}$).
- Poniamo quindi $B = \{2,3,4,8\}$ e $DT(A, \mathrm{a}) = B$.
- L'analisi di A è terminata perché dai corrispondenti stati di $\mathcal{N}$ non esce alcuna transizione etichettata b.

Esempio di subset construction (continua)



- Lo stato $4 \in B$ è l'unico da cui si diparte una transizione etichettata con a.
- Da esso si può ritornare nello stato 2 e quindi, mediante ϵ-transizioni, si può tornare nuovamente in 4 oppure in 8 (cioè ϵ-$CLOSURE(\{2\} = \{2,4,8\})$.
- Poniamo quindi $C = \{2,4,8\}$ e $DT(B,\mathrm{a}) = C$.
- Analogamente, considerando il carattere b di input, avremo ancora un nuovo stato di $\mathcal{D}$, e precisamente $D = \{5,6,7\}$ e $DT(B,\mathrm{b}) = D$.

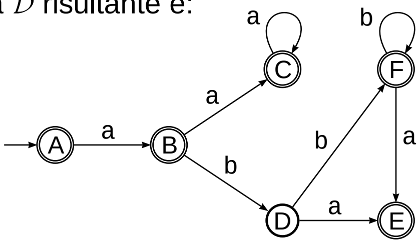
Esempio di subset construction (continua)



- Continuando in questo modo introduciamo dapprima la transizione $DT(C, \mathrm{a}) = C$;
- quindi lo stato $E = \{8\}$ e la transizione $DT(D, \mathrm{a}) = E$;
- quindi lo stato $F = \{5, 6, 7, 8\}$ e la transizione $DT(D, \mathrm{b}) = F$;
- quindi la transizione $DT(F, \mathrm{a}) = E$;
- infine la transizione $DT(F, \mathrm{b}) = F$.

Esempio di subset construction (continua)

- L'automa $\mathcal{D}$ risultante è:



dove

$$
\begin{aligned}
A &= \{0, 1, 8\} \\
B &= \{2, 3, 4, 8\} \\
C &= \{2, 4, 8\} \\
D &= \{5, 6, 7\} \\
E &= \{8\} \\
F &= \{5, 6, 7, 8\}
\end{aligned}
$$

Implementazione della subset construction

- Dobbiamo dapprima decidere come rappresentare i sottoinsiemi di Q, cioè gli elementi di DS.
- Ad esempio, potremmo utilizzare bitmap di $|Q|$ posizioni e rappresentare uno stato mediante un puntatore alla opportuna bitmap.

	0	1	2	3	4	5	6	7	8	9	....					15
A	1	1	0	0	0	0	0	0	1							
B	0	0	1	1	1	0	0	0	1							
C	0	0	1	0	1	0	0	0	1							
D	0	0	0	0	0	1	1	1	0							
E	0	0	0	0	0	0	0	0	1							
F	0	0	0	0	0	1	1	1	1							

Implementazione della subset construction (continua)

- Ogni stato deve inoltre poter essere etichettato con un indicatore binario (diciamo bianco e nero).
- Dobbiamo quindi realizzare la struttura dati DS che “contiene” gli stati e una struttura dati (che sarà una tabella) che rappresenta la funzione di transizione di $\mathcal{D}$,
- Per quanto riguarda DS, diciamo solo che essa deve prevedere operazioni di inserimento e ricerca di stati bianchi.
- La diapositiva seguente illustra l’algoritmo.

Algorithm 1 Subset construction

Inserisci $\epsilon\text{-}CLOSURE(q_0)$ in DS e coloralo di bianco
while esiste uno stato bianco S in DS **do**
  colora S di nero
  for all $x \in \Sigma$ **do**
    $T \leftarrow \{\}$
    for all $q \in S$ **do**
      $T \leftarrow T \cup \delta(q, x)$
    $T \leftarrow \epsilon\text{-}CLOSURE(T)$
    if $T \notin DS$ **then**
      inserisci T in DS e coloralo di bianco
    poni $DT[S, x] = T$

Algorithm 2 Calcolo della ϵ-$CLOSURE(T)$

for all $q \in T$ **do**
    inserisci q in una pila
Poni ϵ-$CLOSURE(T) \leftarrow T$
while La pila non è vuota **do**
    estrai q dalla pila
    if $\delta(q, \epsilon) \neq \perp$ e $\delta(q, \epsilon) = \{q_1, \ldots, q_k\}$ **then**
        for $i = 1, \ldots, k$ **do**
            if $q_i \notin \epsilon$-$CLOSURE(T)$ **then**
                ϵ-$CLOSURE(T) \leftarrow \{q_i\} \cup \epsilon$-$CLOSURE(T)$
                inserisci q_i sulla pila

Un’ultima osservazione sull’efficienza della simulazione

- Abbiamo già osservato come il numero di transizioni di stato di un automa $\mathcal{M}$ sia una buona misura del tempo di calcolo speso da $\mathcal{M}$ su un dato input.
- Un automa non deterministico che riceva in input una stringa di lunghezza *n* esegue, se la stringa è riconosciuta, “almeno” *n* transizioni di stato.
- Le transizioni possono essere di più, se qualcuna è etichettata con ϵ.
- Un automa deterministico equivalente sullo stesso input eseguirà “esattamente” *n* transizioni di stato.
- Ne consegue quindi che l’automa deterministico non è meno efficiente dal punto di vista del tempo.
- Il consumo di spazio, invece (ancorché sempre indipendente dalla dimensione delle stringhe in input) può invece essere decisamente più elevato.