

Oggi formalizzeremo ciò che abbiamo visto ieri: la minimizzazione del DFA.

Input: DFA con funzione di transizione totale.

Output: minDFA, cioè quello che ha il numero minimo di stati (nodi) e transizioni (archi), che non necessariamente ha funzione di transizione totale, ed è depurato da dead-states e pozzo.

Questo algoritmo è il più efficiente conosciuto.

Partizionamento: si tratta di effettuare un partizionamento in gruppi, incui elementi siano equivalenti rispetto ad una nozione specifica di equivalenza.

EQUIVALENZA:

$move_d^k$ ← In quanti k -passi di transizione.
è definita per induzione sul valore di k .

$move_d^0(s, \epsilon) = s \rightarrow$ "Applicata ad un certo stato s , considerando la parola vuota ϵ ".

Ricordiamo che i DFA non prevedono ϵ -transizioni. Qui ϵ è usata solo per dare la definizione induttiva.

$move_d^{k+1}(s, wa) = move_d^k(move_d^k(s, w), a)$ Per muoversi in $k+1$ passi in s rispetto alla parola wa .

Sia $D = (S, A, move_d, s_0, F)$ un DFA con $move_d$ totale. Allora due stati $(s, t) \in S$ sono **equivalenti**, e si scrive $s \sim t$, se e solo se per ogni parola w di lunghezza k e sull'alfabeto A , la $move_d^k(s, w) \in F$ se e solo se anche $move_d^k(t, w) \in F$.

\uparrow
è uno stato finale

Questo esplicita formalmente quello che ieri avevamo trattato intuitivamente: due stati sono equivalenti se, con la stessa parola, riesco a raggiungere uno stato finale (non lo stesso stato finale, ma *un certo* stato finale).

La minimizzazione del DFA consiste, da un punto di vista tecnico, nel trovare una partizione degli stati del nostro DFA, rispetto a questa nozione di equivalenza. Cioè, vogliamo mettere, nello stesso sottoinsieme di stati di un DFA, tutti quelli che hanno questa caratteristica.

Noi partiamo dal seguente partizionamento grossolano.

$B_1 = F$ (stati finali)

$B_2 = S \setminus F$ (non finali)

Questi due blocchi rappresentano l'insieme di quello stati che si differenziano per parole di lunghezza 0. Cioè, quello che abbiamo è che:

$\forall s \in B_1, \forall t \in B_2, s \not\sim t$ in quanto:

- $move_d^0(s, \epsilon) = s \in B_1 \in F$ (finale)
- $move_d^0(t, \epsilon) = t \in B_2$, quindi certamente $t \notin F$.

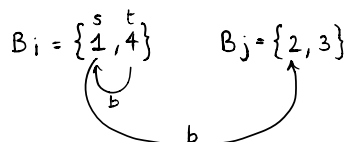
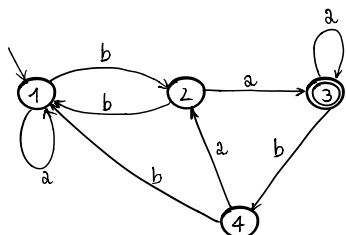
Come facciamo il raffinamento di B_2 ? Quello che dobbiamo ottenere, nel caso in cui riteniamo sia necessario suddividere ancora B_1 e B_2 , sono blocchi che hanno queste caratteristiche:

1. Contengono stati equivalenti,
2. Coppie distinte di blocchi non contengono stati equivalenti (gli stati equivalenti sono obbligati a stare nello stesso blocco).

Cosa facciamo per accertarci di queste due proprietà? Diciamo che:

Dato una certa partizione degli stati (es.: nei blocchi $B_1 \dots B_5$), se, per qualche $B_i \neq B_j$ e per qualche $a \in A, \exists s, t \in B_i$ t.c. e t.c. $move_d(s, a) \in B_j$ & $move_d(t, a) \in B_i$, allora diciamo che B_i è **splittabile** rispetto a (B_j, a) .

MENO FORMALMENTE: Se esistono 2 stati s, t di uno stesso blocco B_i , e un simbolo a , tale per cui tramite a arrivo in uno stato B_i per s , ma in uno stato in un blocco $B_j \neq B_i$ per t , allora...



s va in B_j , t non va in B_j ma in B_i

B_i può essere splittato rispetto a (B_j, a) .

Cosa è lo split di B_i rispetto a (B_j, a) ?

E' ottenuto rimpiazzando B_i con 2 blocchi dati da $\{s \in B_i \text{ t.c. } move_d(s, a) \in B_j\}$ & $\{s \in B_i \text{ t.c. } move_d(s, a) \notin B_j\}$

Algoritmo di partition refinement

INPUT: DFA totale $D = (S, A, move_d, s_0, F)$

OUTPUT: partizione degli stati di S in blocchi di stati equivalenti secondo la definizione di equivalenza.

$$B_1 = F$$

$$B_2 = S \setminus F$$

$$P = \{B_1, B_2\}$$

while $(\exists B_i, B_j \in P \ \& \ \exists a \in A \text{ con } B_i \text{ splittabile rispetto a } (B_j, a))$

$\{ \text{split di } B_i \text{ rispetto a } (B_j, a) \text{ in } P \}$ % ottengo 2 sottoinsiemi: - stati di B_i che tramite a -transizioni raggiungono stati che stanno in B_j
- stati di B_i che tramite a -transizioni raggiungono stati che stanno in B_j

Algoritmo per la minimizzazione di DFA

INPUT: DFA totale

OUTPUT: min DFA

Let $P = \{B_1, \dots, B_n\}$ il risultato del partition refinement di B .

foreach B_i impostare uno stato temporaneo t_i (poi potremo anche buttarlo via) % con questo foreach decidiamo quali sono gli stati

if $B_i \subseteq F$ (e' composto esclusivamente da stati finali) allora t_i è finale % dell'automa che stiamo costruendo

foreach (B_i, B_j, a) t.c. $s_i \in B_i, s_j \in B_j, move_d(s_i, a) = s_j$ % c'è una transizione da s_i a s_j etichettata a

impostare una transizione etichettata a dallo stato temporaneo t_i per B_i allo stato temporaneo t_j per B_j .

foreach dead temporary state t_i , rimuovere t_i e tutte le transizioni da/a t_i .

- Gli stati del minDFA sono gli stati temporanei rimasti (sopravvissuti) a questa fase qui.
- Le transizioni del minDFA sono le transizioni rimaste.
- Lo stato iniziale del minDFA è lo stato che corrisponde a B_i t.c. $s_0 \in B_i$