Dispensa del corso di Sistemi a eventi discreti

Riccardo Bartolomioli
Sebastiano Fregnan

2019-2020

Indice

1 | Segnali discreti e Sistemi

1.1 Definizione di segnale discreto

Si consideri il segnale della forma $e\colon \mathbb{R} \to \{assente\} \cup X$, dove X è un insieme qualsiasi di valori. Intuitivamente questo segnale è discreto se è assente per la maggior parte del tempo, e si possono contare in ordine i tempi in cui è presente (cioè non assume il valore *assente*). Ogni volta che è presente, si ha un evento discreto.

Per esempio, se e è presente per tutti i numeri razionali t, allora non diciamo che il segnale è discreto, perchè i tempi in cui è presente non possono essere contati in ordine (essi non sono una *successione* di eventi istantanei nel tempo, bensì un *insieme* di eventi istantanei nel tempo). Formalmente, se $T \subseteq \mathbb{R}$ è l'insieme dei tempi in cui e è presente, cioè $T = \{\, t \in \mathbb{R} \mid e(t) \neq assente \,\}$, il segnale e è detto ***discreto*** se c'è una funzione iniettiva $f\colon T \to \mathbb{N}$ che preserva l'ordine, cioè $\forall t_1, t_2 \in T$ se $t_1 \leq t_2$ allora $f(t_1) \leq f(t_2)$. L'esistenza di tale funzione iniettiva garantisce che possiamo contare gli eventi secondo un ordine temporale.

Un segnale si dice ***puro*** se ad ogni istante di tempo è assente (non c'è nessun evento in quell'istante) o presente (c'è un evento in quell'istante), cioè non specifica un valore, ma solo l'informazione di essere presente o assente in un certo istante di tempo.

Esempio

Si consideri il segnale puro $x\colon \mathbb{R} \to \{presente, assente\}$ dato da

$$\forall t \in \mathbb{R}, x(t) = \begin{cases} presente & \text{se } t \text{ è un intero non-negativo} \\ assente & \text{altrimenti} \end{cases}$$

Questo segnale è discreto?
Sì. Dobbiamo costruire una funzione $f\colon T \to \mathbb{N}$ iniettiva e che preserva l'ordine. Poichè in questo caso l'insieme dei tempi in cui il segnale x è presente, è $T = \mathbb{N}$, basta prendere f la funzione identità che è iniettiva e preserva banalmente l'ordine.

Esempio

Si consideri il segnale puro $y\colon \mathbb{R} \to \{presente, assente\}$ dato da

$$\forall t \in \mathbb{R}, y(t) = \begin{cases} presente & \text{se } t = 1 - \frac{1}{n} \text{ per ogni intero positivo } n \\ assente & \text{altrimenti} \end{cases}$$

Questo segnale è discreto?
Sì. Dobbiamo costruire una funzione $f\colon T \to \mathbb{N}$ iniettiva e che preserva l'ordine. Poichè in questo caso l'insieme dei tempi in cui il segnale y è presente, è $T = \{1 - \frac{1}{1}, 1 - \frac{1}{2}, 1 - \frac{1}{3}, \dots\}$ possiamo definire f come

$$\forall t \in T\colon f(t) = n, \text{ dove } t = 1 - \frac{1}{n}$$

che è iniettiva e preserva l'ordine.

Esempio

Si consideri il segnale w che si ottiene dalla fusione dei due segnali precedenti x e y, cioè

$$\forall t \in \mathbb{R}, w(t) = \begin{cases} presente & \text{se } x(t) \text{ è } presente \text{ o } y(t) \text{ è } presente \\ assente & \text{altrimenti} \end{cases}$$

Questo segnale è discreto?
No. Per essere discreto dovrebbe esserci una funzione $f\colon T \to \mathbb{N}$ iniettiva e che preserva l'ordine. Si noti che $1 \in T$ e che $1 - \frac{1}{n} \in T$, $\forall n \in \mathbb{N}$, $n > 0$; inoltre $1 > 1 - \frac{1}{n}$, $\forall n \in \mathbb{N}$, $n > 0$. Se f preserva l'ordine deve

essere vero che

$$f(1) > f(1 - \frac{1}{n})$$

ma $f(1 - \frac{1}{n})$ non ha un maggiorante in $\mathbb{N}$, poichè $1 - \frac{1}{n}$ non ha un maggiorante < 1, per $n \in \mathbb{N}$, $n > 0$. Quindi tale f non può esistere. La conclusione è che la proprietà dei segnali discreti non è preservata dalla composizione.

1.2 Sistemi

Con il termine ***sistema fisico*** si indica, tipicamente, una porzione di spazio fisicamente o logicamente distinita dal restante universo, con il quale il sistema interagisce. Dal sistema fisico noi cerchiamo un modello matematico (***sistema dinamico*** o semplicemente ***sistema***) che costituisca, per l'applicazione in questione, un ragionevole compromesso tra l'esigenza di modellare i fenomeni che lo interessano nel modo più preciso possibile e il bisogno di disporre di un modello computazionalmente trattabile.

Per i sistemi adotteremo due tipi di modello: i ***modelli ingresso/uscita*** ed i ***modelli di stato*** (questi ultimi verranno definiti più avanti). In entrambi i casi si assume che il sistema sia soggetto ad un sollecitazione in input a cui risponde con un uscita.

In generale, possiamo classificare i sistemi ingresso/uscita in due categorie sulla base dell'input/output:

sistemi traduttivi (o ***combinatori***) la cui uscita in ogni istante dipende *solo* dall'input in quell'istante; formalmente li descriviamo come una funzione:

$$\mathit{Sistema\ traduttivo}\colon \mathit{Valori} \to \mathit{Valori} \tag{1.1}$$

sistemi reattivi (o ***sequenziali***) che trasforma segnali in altri segnali (generalmente verranno utilizzati segnali con dominio nel tempo). Formalmente li descriviamo come una funzione:

$$\mathit{Sistema\ reattivo}\colon [\mathit{Tempo} \to \mathit{Valori}] \to [\mathit{Tempo} \to \mathit{Valori}] \tag{1.2}$$

Tuttavia è possibile dimostrare facilmente che ogni sistema traduttivo sia in realtà realizzabile tramite un sistema reattivo, infatti considerando un sitema traduttivo f ed uno reattivo F possiamo affermare che: $\forall x \in [\mathit{Tempo} \to \mathit{Valori}]$, $\forall y \in \mathit{Tempo}$, $(F(x))(y) = f(x(y))$ (in pratica un sistema traduttivo corrisponde ad una configurazione temporale di un sistema reattivo). Tra le due categorie sussiste dunque la seguente relazione:

$$\mathit{Sistemi\ traduttivi} \subset \mathit{Sistemi\ reattivi} \tag{1.3}$$

Esistono sistemi reattivi con alcune caratteristiche peculiari e tra essi troviamo i seguenti:

- sistemi privi di memoria;
- sistemi con ritardo;
- sistemi causali;
- sistemi con memoria finita;
- sistemi con memoria infinita.

1.2.1 Sistemi privi di memoria

Un sistema reattivo F è privo di memoria *sse* esiste un sistema traduttivo f tale che $\forall x \in [\mathit{Tempo} \to \mathit{Valori}]$, $\forall y \in \mathit{Tempo}$, $(F(x))(y) = f(x(y))$.

Esempio

Ecco alcuni esempi di sistemi reattivi privi di memoria:

Normalize

$$Normalize\colon [\mathbb{R}^+ \to \mathbb{R}] \to [\mathbb{R}^+ \to \mathbb{R}]$$
$$\text{tale che } \forall x \in [\mathbb{R}^+ \to \mathbb{R}],\ \forall y \in \mathbb{R}^+,$$
$$(Normalize(x))(y) = x(y) - 50$$

Trunc

$$Trunc\colon [\mathbb{R}^+ \to \mathbb{R}] \to [\mathbb{R}^+ \to \mathbb{R}]$$
$$\text{tale che } \forall x \in [\mathbb{R}^+ \to \mathbb{R}],\ \forall y \in \mathbb{R}^+,$$
$$(Trunc(x))(y) = \begin{cases} 256 & \text{se } x(y) > 256 \\ x(y) & \text{se } -256 \leq x(y) \leq 256 \\ -256 & \text{se } x(y) < -256 \end{cases}$$

Quantize

$$Quantize\colon [\mathbb{N}_0 \to \mathbb{R}] \to [\mathbb{N}_0 \to ComputerInts]$$
$$\text{tale che } \forall x \in [\mathbb{N}_0 \to \mathbb{R}], \forall y \in \mathbb{N}_0,$$
$$(Quantize(x))(y) = \begin{cases} 256 & \text{se } \lfloor x(y) \rfloor > 256 \\ \lfloor x(y) \rfloor & \text{se } -256 \leq \lfloor x(y) \rfloor \leq 256 \\ -256 & \text{se } \lfloor x(y) \rfloor < -256 \end{cases}$$

Negate

$$Negate\colon [\mathbb{N}_0 \to Bools] \to [\mathbb{N}_0 \to Bools]$$
$$\text{tale che } \forall x \in [\mathbb{N}_0 \to Bools],\ \forall y \in \mathbb{N}_0,$$
$$(Negate(x))(y) = \neg x(y)$$

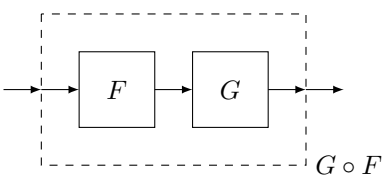
Identity system (Id)

$$Id\colon [Tempo \to Valori] \to [Tempo \to Valori]$$
$$\text{tale che } \forall x \in [Tempo \to Valori],$$
$$Id(x) = x$$

Constant system (Const)

$$\text{Per ogni costante } c \in Valori,$$
$$Const\colon [Tempo \to Valori] \to [Tempo \to Valori]$$
$$\text{tale che } \forall x \in [Tempo \to Valori],\ \forall y \in Tempo,$$
$$(Const(x))(y) = c$$

È importante notare che qualsiasi ***composizione*** di sistemi privi di memoria è a sua volta unico sistema privo di memoria, un esempio:



se F e G sono sistemi reattivi privi di memoria, allora anche $G \circ F$ sarà privo di memoria.

1.2.2 Sistemi con ritardo

Definiamo i sistemi con ritardo nel seguente modo:

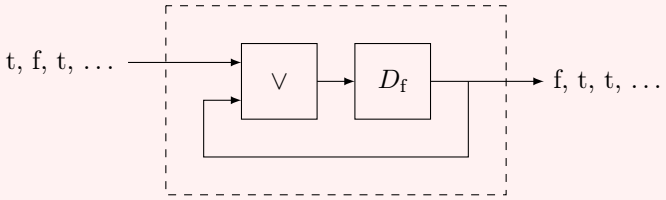
$$Delay_c\colon [Tempo \to Valori] \to [Tempo \to Valori]$$
$$\text{tale che } \forall x \in [Tempo \to Valori], \forall y \in Tempo,$$
$$(Delay_c(x))(y) = \begin{cases} c & \text{se } y < 1 \\ x(y-1) & \text{se } y \geq 1 \end{cases}$$

Questo modello di sistemi è applicabile sia per valori di ritardo temporale nel continuo, sia nel discreto. In particolare: se il sistema lavora su un'insieme di valori discreto e finito allora esso necessiterà di una memoria finita. In caso contrario (insieme continuo e/o illimitato) la memoria del sistema dovrà essere infinita.

L'importanza di questo tipo di sistemi sta nel fatto che essi rendono possibile l'introduzione di ***cicli*** all'interno del diagramma a blocchi.

Esempio

Nel seguente esempio viene riportato il comportamento del sistema, rispetto ad una sequenza di booleani in ingresso, mostrando la sequenza degli output:



In generale, per la composizione di sistemi possiamo affermare che:

- una rappresentazione in diagramma a blocchi di un sistema *traduttivo* è *valida* se
  - tutte le sue componenti sono anch'esse dei sistemi traduttivi;
  - non sono presenti cicli;
- una rappresentazione in diagramma a blocchi di un sistema *reattivo* è *valida* se
  - tutte le sue componenti sono sistemi privi di memoria o sono sistemi con ritardo;
  - ogni ciclo contiene almeno un sistema con ritardo.

1.2.3 Sistemi causali

Un sistema reattivo F è ***causale*** (o ***implementabile***) *sse*:

$$\forall x, y \in [Tempo \to Valori], \forall z \in Tempo,$$
$$\text{se } (\forall t \in Tempo,\ t \leq z \implies x(t) = y(t))$$
$$\text{allora } (F(x))(z) = (F(y))(z)$$

1.2.4 Sistemi con memoria finita e infinita

Sistemi con memoria finita sono naturalmente implementati come ***automi a stati finiti*** (o ***automi a transizoni finite***). Sistemi con memoria numerabile invece sono naturalmente implementati come ***automi a stati infiniti*** (o ***automi a transizioni infinite***). Sistemi con memoria non numerabile invece non sono naturalmente implementabili attraverso un automa a stati.

2 | Automi a stati finiti deterministici

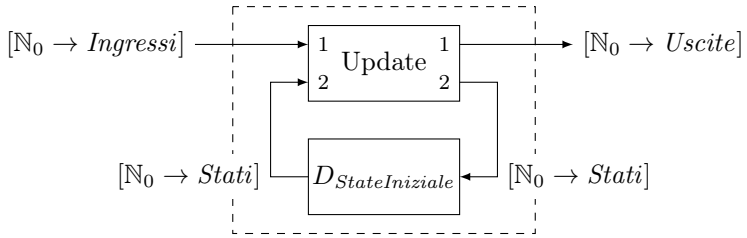
Come abbiamo appena visto, i sistemi sono funzioni che trasformano i segnali. Il dominio e il codominio di queste funzioni sono entrambi spazi di segnale, il che complica notevolmente la specifica delle funzioni. Un'ampia classe di sistemi può essere caratterizzata usando il concetto di ***stato*** e l'idea che un sistema si *evolva* attraverso una sequenza di ***cambiamenti di stato*** o ***transizioni di stato***. Tali caratterizzazioni sono chiamate ***modelli dello spazio degli stati.***

Un modello dello spazio degli stati descrive proceduralmente un sistema, fornendo una sequenza di operazioni passo-passo per l'evoluzione di un sistema. Esso mostra come il segnale di ingresso guida i cambiamenti nello stato, e come viene prodotto il segnale di uscita.

È quindi una descrizione imperativa. L'implementazione di un sistema descritto da un modello dello spazio degli stati in software o hardware è semplice. L'hardware o il software devono semplicemente eseguire in sequenza i passaggi indicati dal modello. Viceversa, dato un pezzo di software o hardware, è spesso utile descriverlo usando un modello dello spazio degli stati, che offre migliori analisi rispetto a descrizioni più informali.

2.1 Struttura dell'automa

Un automa a stati costruisce il segnale di uscita un valore alla volta, partendo dall'osservazione il segnale in ingresso (anch'esso un valore alla volta).



A livello formale, un automa a stati deterministico è definito da una 5-tupla:

$$MSFD = (Stati, Ingressi, Uscite, Update, StatoIniziale) \tag{2.1}$$

Definiamo ora le varie componenti:

- *Stati* è lo spazio (insieme) degli stati. Un automa a stati è finito se l'insieme degli stati è *finito*;
- *Ingressi* è l'insieme di tutti i possibili valori in ingresso;
- *Uscite* è l'insieme di tutti i possibili valori in uscita;
- $StatoIniziale \in Stati$ è lo stato iniziale;
- $Update: Stati \times Ingressi \to Stati \times Uscite$, è la funzione di aggiornamento (o ***funzione di transizione***), essa associa ad ogni coppia "stato attuale-ingresso" *una e una sola* coppia "stato successivo-uscita".

> ***Osservazione:*** ogni sistema privo di memoria (sez. 1.2.1) può essere implementato da un ASFD con un unico stato.

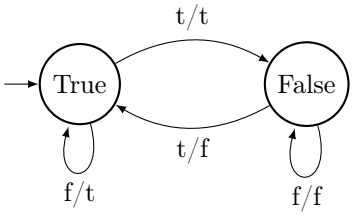
> ***Osservazione:*** ogni sistema causale può essere implementato da un automa a stati; ogni sistema che può essere implementato da un automa a stati è causale.

Esistono due differenti rappresentazioni della funzione di transizione (nei seguenti esempi verrà implementata la funzione di transizione dell'automa per il calcolo della parità dei valori true):

Tabella delle transizioni

Stato attuale	Input	Stato successivo	Output
True	t	False	t
True	f	True	t
False	t	True	f
False	f	False	f

Diagramma delle transizioni



Un diagramma delle transizioni è una rappresentazione per grafi di un automa, ed è quindi costituito da due componenti fondamentali:

- ***Nodi***, che rappresenteranno gli ***stati*** dell'automa;
- ***Archi***, che rappresenteranno le singole ***transizioni***.

Un automa a stati finiti detrministico per essere tale, deve soddisfare alcune proprietà (date del fatto che *Update* è una funzione):

Determinismo Per ogni stato e input, esiste *al massimo* un arco uscente (per ogni stato non possono esistere più di una transzione possibile per lo stesso input);

Ricettività Per ogni stato e input, esiste *almeno* un arco uscente (per ogni stato deve essere descritto almeno una transizione per ogni input).

Osservazione: uno stato q di un automa M è irraggiungibile sse non compare in nessuna esecuzione possibile di M, il che vuol dire che non esistono percorsi dallo stato iniziale verso lo stato q. Esso dunque può essere senza apportare cambiamenti alla funzione di input/output di M.

Macchina di Moore Viene chiamata macchina di Moore un automa a stati finiti i cui output sono definiti *solamente* in base allo stato corrente.

Macchina di Mealy Viene chiamata macchina di Mealy un automa a stati finiti i cui output sono definiti in base allo stato e all'input corrente.

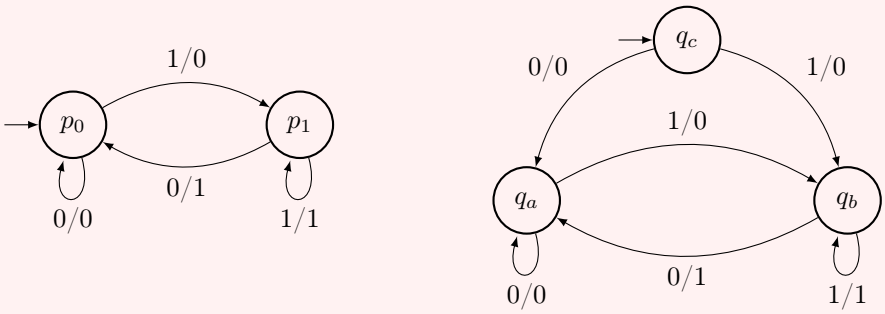
Equivalenza tra automi Un sistema tempo-discreto può avere diverse implementazioni in automi a stati. In generale possiamo dare una definizione di equivalenza tra automi: due automi a stati M_1 e M_2 sono equivalenti sse implementano la stessa funzione di input/output, quindi sse:

1. $Ingressi[M_1] = Ingressi[M_2]$;
2. $Uscite[M_1] = Uscite[M_2]$;
3. $\forall x \in [\mathbb{N}_0 \to Ingressi]$, $M_1(x) = M_2(x)$ con $M_i(x) \in [\mathbb{N}_0 \to Uscite]$.

L'equivalenza tra due automi è un concetto che si riferisce dunque ai soli segnali di input ed output: due automi equivalenti produrranno lo stesso segnale in output a partire dallo stesso segnale in input.

Esempio

Due automi equivalenti:



2.2 Bisimulazione

Teorema

Due macchine a stati M_1 e M_2 sono equivalenti sse esiste una ***bisimulazione*** tra M_1 e M_2 (equivalenza $\iff$ bisimulazione).

Con bisimulazione si intende un insieme di relazioni binarie B tra $Stati[M_1]$ e $Stati[M_2]$, definito come $B \subseteq Stati[M_1] \times Stati[M_1]$ tale che:

1. $(StatoIniziale[M_1], StatoIniziale[M_2]) \in B$;
2. $\forall p \in Stati[M_1]$, $\forall q \in Stati[M_2]$, se $(p, q) \in B$ allora $\forall x \in Ingressi[M_1]$:
   I. $Uscite[M_1](p, x) = Uscite[M_2](q, x)$;
   II. $(NextState[M_1](p, x), NextState[M_2](q, x)) \in B$.

   (Per ogni valore in input, p e q producono lo stesso valore di output, ed i rispettivi stati successivi sono a loro volta relazionati tra loro.)

Riprendendo gli automi visti in precedenza: $B = \{(p_0, q_c), (p_0, q_a), (p_1, q_b)\}$.

2.3 Minimizzazione

L'obbiettivo della minimizzazione è quella di ottenere a partire da una automa M, un altro bisimile con il minor numero di stati possibile. Applicando infatti un algoritmo di minimizzazione su M otterremmo un automa N con le seguenti caratteristiche:

1. M e N sono bisimili (ovvero esiste una bisimulazione tra M e N);
2. Per ogni automa N' che è bisimile a M:
   I. N' possiede al minimo lo stesso numero di stati di N;
   II. Se N' ha lo stesso numero di stati di N, allora N' differisce da N solo per il nome degli stati.

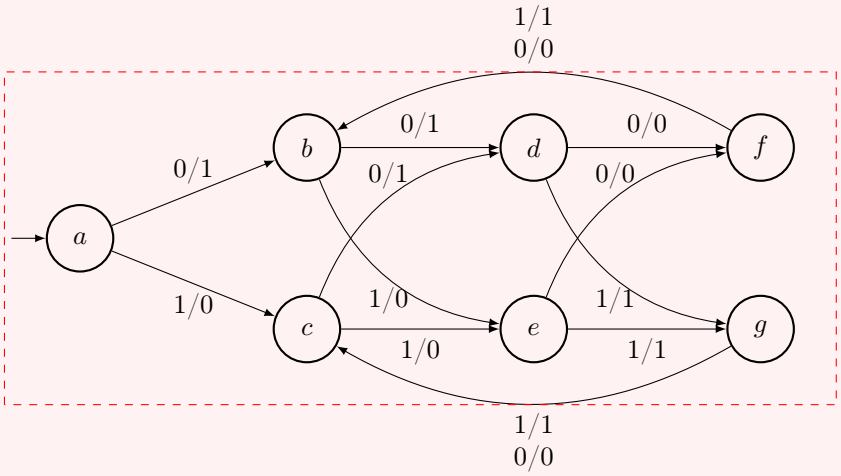
Algoritmo di minimizzazione Definiamo Q come l'insieme di tutti gli stati di M:

1. Costruiamo un insieme P, i cui elementi sono insiemi di stati, in questo modo (impostiamo inizialmente $P = \{Q\}$):
   I. Ripetere finchè possibile la procedura di ***output split***: presi un insieme di stati $R \in P$ e due stati $s_1, s_2 \in R$, se $\exists x \in Ingressi$ tale che $Uscite(s_1, x) \neq Uscite(s_2, x)$ allora i due stati apparterranno ad insiemi diversi (split di R);
   II. Ripetere finchè possibile la procedura di ***next-state split***: presi due insiemi di stati $R, R' \in P$ e due stati $s_1, s_2 \in R$, se $\exists x \in Ingressi$ tale che $NextState(s_1, x) \in R'$ e $NextState(s_2, x) \notin R'$ allora i due stati apparterranno ad insiemi diversi (split di R, nota: R ed R' possono coincidere);
2. Ogni insieme di P rappresenta un singolo stato del più piccolo automa a stati bisimile a M.

Esempio

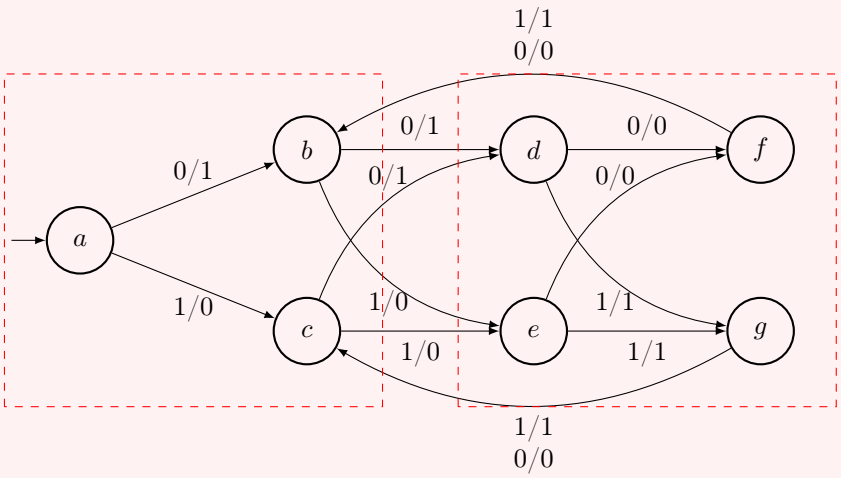
Vediamo un esempio di esecuzione dell'algoritmo per il seguente automa:

$$P = \{Q\} = \{\{a, b, c, d, e, f, g\}\}$$



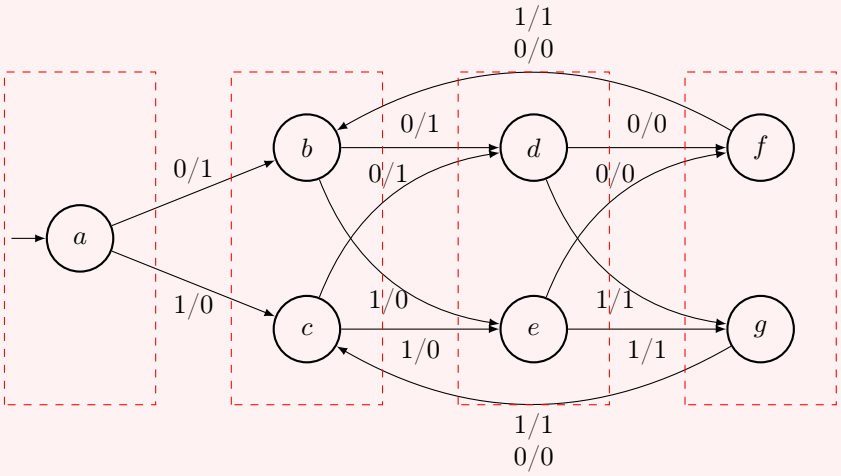
Esecuzione ripetuta dell'***output split:***

$$P = \{Q\} = \{\{a, b, c\}, \{d, e, f, g\}\}$$

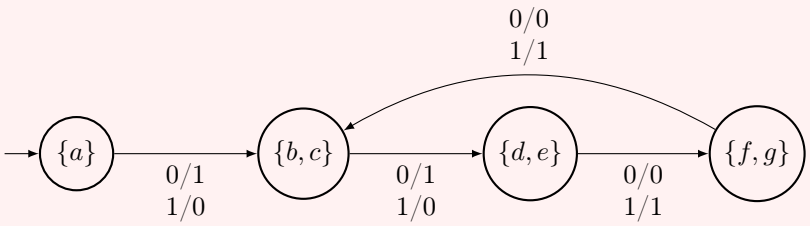


Esecuzione ripetuta del ***next-state split:***

$$P = \{Q\} = \{\{a\}, \{b, c\}, \{d, e\}, \{f, g\}\}$$



L'automa bisimile minimo risulta quindi:



Teorema

Esiste una bisimulazione tra gli automi a stati M_1 e M_2 sse esiste un ***isomorfismo*** tra i loro automi minimizzati.

In conclusione, due automi sono equivalenti se le loro forme minime sono ***isomorfe*** (quindi differiscono al massimo per i nomi degli stati).

Osservazione: ogni diagramma a blocchi di n automi a stati, con i rispettivi insiemi di stati $Stati_1, Stati_2, \ldots, Stati_n$, può essere implementato da un singolo automa a stati, il cui spazio degli stati è dato da $Stati_1 \times Stati_2 \times \cdots \times Stati_n$. Questo automa prende il nome di "automa prodotto".

3 | Automi a stati finiti non deterministici

3.1 Introduzione

Fino a questo momento abbiamo trattato sistemi deterministici, ovvero automi che per ogni segnale in input, associano un unico segnale in output. Esiste tuttavia un classe di sistemi che associano, ad ogni segnale in input, uno o più segnali in output: si tratta della classe di automi ***non-deterministici***.

Tali sistemi quindi non saranno più descritti da funzioni, ma da relazioni:

$$SistemaNonDet \subseteq [Tempo \to Ingressi] \times [Tempo \to Uscite] \tag{3.1}$$

tale che $\forall x \in [Tempo \to Ingressi]$, $\exists y \in [Tempo \to Uscite]$, $(x, y) \in SistemaNonDet$. Ogni coppia $(x, y) \in SistemaNonDet$ è detta ***comportamento*** del sistema reattivo non-deterministico (il comportamento è definito dunque dalla sola coppia "segnale input-segnale output").
Un sistema S_1 si dice che ***raffina*** il sistema S_2 sse:

1. $Tempo[S_1] = Tempo[S_2]$;
2. $Ingressi[S_1] = Ingressi[S_2]$;
3. $Uscite[S_1] = Uscite[S_2]$;
4. $Comportamenti[S_1] \subseteq Comportamenti[S_2]$;

I sistemi S_1 e S_2 sono ***equivalenti*** sse:

1. $Tempo[S_1] = Tempo[S_2]$;
2. $Ingressi[S_1] = Ingressi[S_2]$;
3. $Uscite[S_1] = Uscite[S_2]$;
4. $Comportamenti[S_1] = Comportamenti[S_2]$;

Osservazione: sistemi reattivi tempo-discreti causali deterministici possono essere implementati da automi a stati deterministici; sistemi reattivi tempo-discreti causali non-deterministici possono essere implementati da automi a stati non-deterministici.

3.2 Struttura dell'automa

Rispetto alla formalizzazione degli automi a stati finiti deterministici (2.1) vengono apportate alcune variazioni. La 5-tupla diventa infatti:

$$MSFND = (Stati, Ingressi, Uscite, PossUpdate, PossStatiIniziali) \tag{3.2}$$

Definiamo ora le varie componenti:

- *Stati* è lo spazio (o insieme) degli stati. Un automa a stati è finito se l'insieme degli stati è finito;
- *Ingressi* è l'insieme di tutti i possibili valori in ingresso;
- *Uscite* è l'insieme di tutti i possibili valori in uscita;
- $PossStatiIniziali \subseteq Stati$ è l'insieme degli stati iniziali possibili;
- $PossUpdates\colon Stati \times Ingressi \to \wp(Stati \times Uscite) \setminus \emptyset$ è la funzione di aggiornamento (o ***funzione di transizione***), essa associa per ogni coppia "stato attuale-ingresso" una o più possibili coppie "stato successivo-uscita" (rispetto agli MSFD viene rispettata solo la proprietà di ricettività del sistema, infatti ogni coppia "stato-ingresso" deve avere almeno una transizione associata, per questo si elimina l'insieme vuoto).

Osservazione: nelle macchine a stati deterministiche, per ogni flusso in input, esiste esattamente una esecuzione, invece in quelle non-deterministiche esistono una o più possibili esecuzioni. Ogni esecuzione genera un flusso in output, perciò ogni esecuzione da origine ad un comportamento.

Una macchina a stati è ***progressiva*** quando l'evoluzione è definita per ogni ingresso, cioè la funzione è definita come

$$States \times Inputs \Rightarrow \mathcal{P}(States \times Outputs) \setminus \emptyset$$

dove $\mathcal{P}$ rappresenta l'insieme potenza e l'insieme vuoto impone che sia progressiva.

3.3 Simulazione tra automi

Teorema

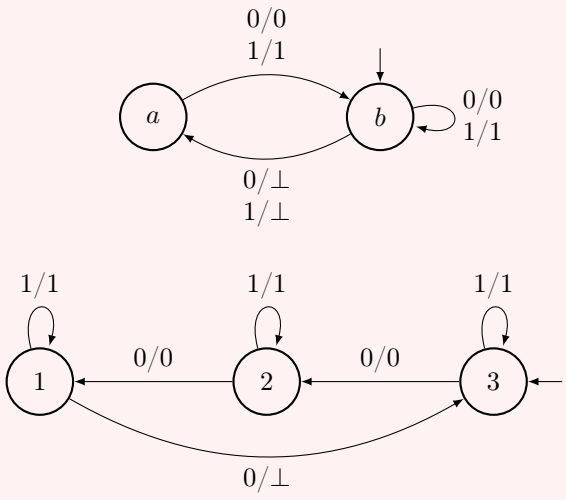
Siano M_1 e M_2 due automi a stati non-deterministici. M_1 raffina M_2 se esiste una ***simulazione*** di M_1 per M_2. Diremo quindi che M_2 simula M_1 (tale affermazione non è simmetrica, in quanto M_1 non può simulare M_2).

Assumiamo che $Ingressi[M_1] = Ingressi[M_2]$ e $Uscite[M_1] = Uscite[M_2]$. Formalmente, una relazione $S \subseteq Stati[M_1] \times Stati[M_2]$ è detta ***simulazione*** sse:

1. $\forall p \in PossInitialStates[M_1]$, $\exists q \in PossInitialStates[M_2]$ tale che $(p, q) \in S$;
2. $\forall p \in Stati[M_1]$, $\forall q \in Stati[M_2]$, se $(p, q) \in S$ allora: $\forall x \in Ingressi$, $\forall y \in Uscite$, $\forall p' \in Stati[M_1]$, se $(p', y) \in PossUpdates[M_1](p, x)$ allora $\exists q' \in Stati[M_2]$ tale che $(q', y) \in PossUpdates[M_2](q, x)$ e $(p', q') \in S$.

Esempio

Vediamo un esempio di simulazione tra i seguenti automi:



In questo esempio la simulazione è definita: $S = \{(3, b), (2, b), (1, b), (3, a)\}$.

3.4 Automi output-deterministici

All'interno della classe degli automi non-deterministici possiamo identifcare due speciali casi di automi a stati, e lo facciamo nel seguente modo:

Deterministico se c'è un solo stato iniziale[1], e per ogni stato e ogni input esiste un solo stato successivo (in sintesi, per ogni ***segnale in ingresso*** esiste una sola possibile esecuzione dell'automa);

[1] $|PossStatoIniziale| = 1$

Output-deterministico se c'è un solo stato iniziale, e per ogni stato e ogni coppia ingresso-uscita esiste un solo stato successivo (in sintesi per ogni ***comportamento***, esiste una sola possibile esecuzione dell'automa).

Se M_2 è un automa ouput-deterministico, allora può essere trovata una simulazione S di M_1 per M_2 nel seguente modo:

1. Se $p \in \mathit{PossStatiIniziali}[M_1]$ e $\mathit{PossStatiIniziali}[M_2] = \{q\}$ allora $(p, q) \in S$;
2. Se $(p, q) \in S$, $(p', y) \in \mathit{PossUpdates}[M_1](p, x)$ e $\mathit{PossUpdates}[M_2](q, x) = \{q'\}$ allora $(p', q') \in S$.

Osservazione: sono valide tutte le seguenti affermazioni:

- Se M_2 è una MSFD allora M_1 è simulato da M_2 sse M_1 è equivalente a M_2. Con 'equivalenza' possiamo intendere che M_1 raffina M_2 e viceversa.
- Se M_2 è una MSF output-deterministico allora M_1 è simulato da M_2 sse M_1 raffina M_2 ma non viceversa.
- Se M_2 è una MSFND, allora M_1 è simulato da M_2 implica che M_1 raffina M_2 ma non viceversa (M_1 può raffinare M_2 anche se non è simulato da quest'ultimo).

$$\text{Deterministico} \implies \text{Output-deterministico} \implies \text{Non-deterministico}$$

Una simulazione tra automi deterministici e output-deterministici può dunque essere trovata molto agevolamente, ma non possiamo dire lo stesso per automi output-deterministici e non-deterministici. Fortunamente è possibile per ogni automa non-deterministico trovare una sua versone equivalente output-deterministica.

3.5 Output-determinazione

Per ogni automa a stati non-deterministico M, possiamo trovare un automa a stati output-deterministico outdet(M) che è equivalente a M (questa operazione è chiamata "costruzione del sottoinsieme"). A questo punto, per verifcare se M_1 raffina M_2 basta verificare se M_1 è simulato da outdet(M_2). Possiamo dunque affermare che:

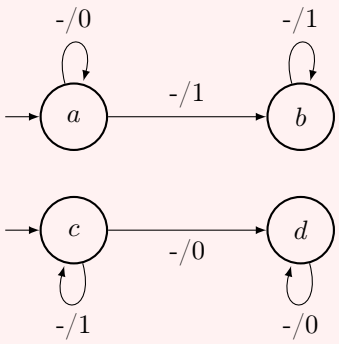
$$M_1 \text{ raffina } M_2 \iff M_1 \text{ raffina outdet}(M_2) \iff M_1 \text{ è simulato da outdet}(M_2)$$

Algoritmo di output-determinazione La costruzione del sottoinsieme si effettua nel seguente modo, sia $\mathit{Ingressi}[\text{outdet}(M)] = \mathit{Ingressi}[M]$ e $\mathit{Uscite}[\text{outdet}(M)] = \mathit{Uscite}[M]$:

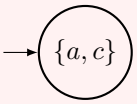
1. $\mathit{StatoIniziale}[\text{outdet}(M)] = \mathit{PossStatiIniziali}[M]$ (gli stati iniziali di M formeranno un unico stato iniziale di outdet(M));
2. $\mathit{Stati}[\text{outdet}(M)] = \{\mathit{StatoIniziale}[\text{det}(M)]\}$ (inizialmente l'insieme degli stati di outdet(M) sarà formato dal solo stato iniziale);
3. Ripetere finché è possibile aggiungere nuove transizioni a outdet(M):
   I. Scegliere uno stato $P \in \mathit{Stati}[\text{outdet}(M)]$ e una coppia $(x, y) \in \mathit{Ingressi} \times \mathit{Uscite}$;
   II. Creare $Q = \{\, q \in \mathit{Stati}(M) \mid \exists p \in P, (q, y) \in \mathit{PossUpdates}[M](p, x) \,\}$, ovvero l'insieme degli stati q di M che sono il possibile risultato (con uscita y) di una transizione dell'automa a partire da uno stato p (e ingresso x) tale che $p \in P$;
   III. Se $Q \neq \emptyset$ allora viene aggiunto agli stati di outdet(M) e viene aggiunta la transizione $\mathit{Update}[\text{outdet}(M)](P, x) = (Q, y)$.

Esempio

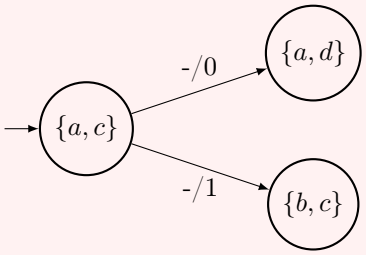
Vediamo un esempio di esecuzione dell'algoritmo per il seguente automa M:



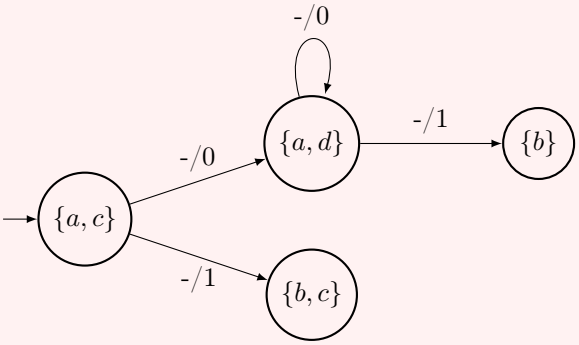
Esecuzione del passo 1 (e passo 2):



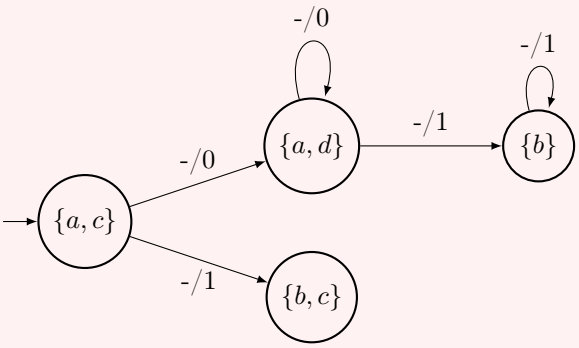
Esecuzione iterata del passo 3 per lo stato $P = \{a, c\}$:



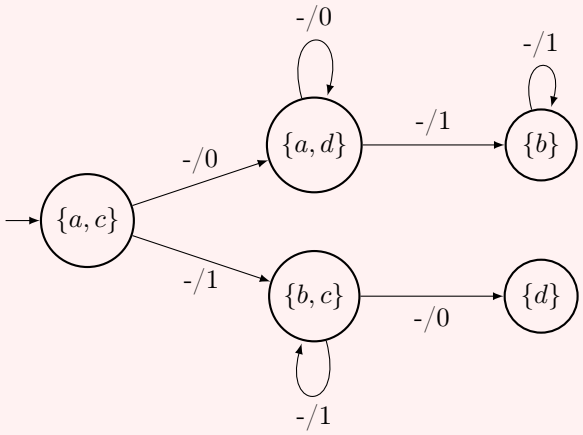
Esecuzione iterata del passo 3 per lo stato $P = \{a, d\}$:



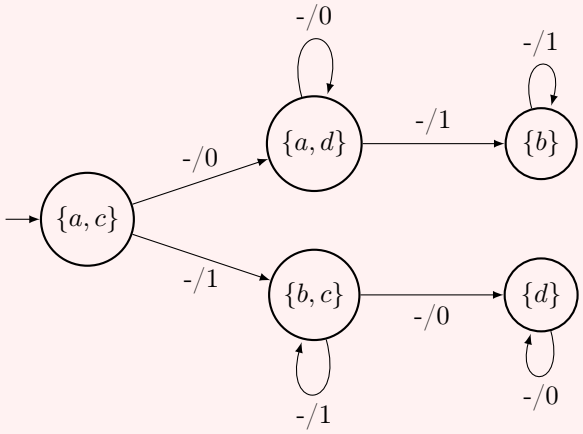
Esecuzione iterata del passo 3 per lo stato $P = \{b\}$:



Esecuzione iterata del passo 3 per lo stato $P = \{b, c\}$:



Esecuzione iterata del passo 3 per lo stato $P = \{d\}$:



Per un dato sistema reattivo S:

- Se esiste un automa a stati non-deterministico che implementa S con n stati, allora esiste un automa a stati output-deterministico che implementa S con 2^n stati;
- Potrebbe non esserci un automa a stati output-deterministico che implementa S con meno di 2^n stati;
- Potrebbe non esserci un unico automa a stati non-deterministico che implementa S con il minor numero di stati (due automi possono essere equivalenti senza essere isomorfi).

3.6 Bisimulazione

Una relazione binaria $B \subseteq \mathit{Stati}[M_1] \times \mathit{Stati}[M_2]$ è una ***bisimulazione*** tra M_1 e M_2 sse:

1. $\forall p \in \mathit{PossStatiIniziali}[M_1]$, $\exists q \in \mathit{PossStatiIniziali}[M_2]$ tale che $(p, q) \in B$;
2. $\forall p \in \mathit{Stati}[M_1]$, $\forall q \in \mathit{Stati}[M_2]$, se $(p, q) \in B$ allora:
   $\forall x \in \mathit{Ingressi}, \forall y \in \mathit{Uscite}, \forall p' \in \mathit{Stati}[M_1]$, se $(p', y) \in \mathit{PossUpdates}[M_1](p, x)$ allora $\exists q' \in \mathit{Stati}[M_2]$ tale che:
   I. $(q', y) \in \mathit{PossUpdates}[M_2](q, x)$;
   II. $(p', q') \in B$;
3. $\forall q \in \mathit{PossStatiIniziali}[M_2]$, $\exists p \in \mathit{PossStatiIniziali}[M_1]$ tale che $(p, q) \in B$;
4. $\forall p \in \mathit{Stati}[M_1]$, $\forall q \in \mathit{Stati}[M_2]$, se $(p, q) \in B$ allora:
   $\forall x \in \mathit{Ingressi}, \forall y \in \mathit{Uscite}, \forall q' \in \mathit{Stati}[M_2]$, se $(q', y) \in \mathit{PossUpdates}[M_2](q, x)$ allora $\exists p' \in \mathit{Stati}[M_1]$ tale che:
   I. $(p', y) \in \mathit{PossUpdates}[M_1](q, x)$;

II. $(p', q') \in B$.

Osservazione: Se M_1 e M_2 sono due MSFD allora possiamo dire che:

$$M_1 \text{ e } M_2 \text{ sono bisimili} \iff M_1 \text{ è equivalente a } M_2 \tag{3.3}$$

Se M_1 e M_2 sono due MSFND allora possiamo dire che:

$$M_1 \text{ e } M_2 \text{ sono bisimili} \implies M_1 \text{ è equivalente a } M_2 \tag{3.4}$$

3.7 Minimizzazione

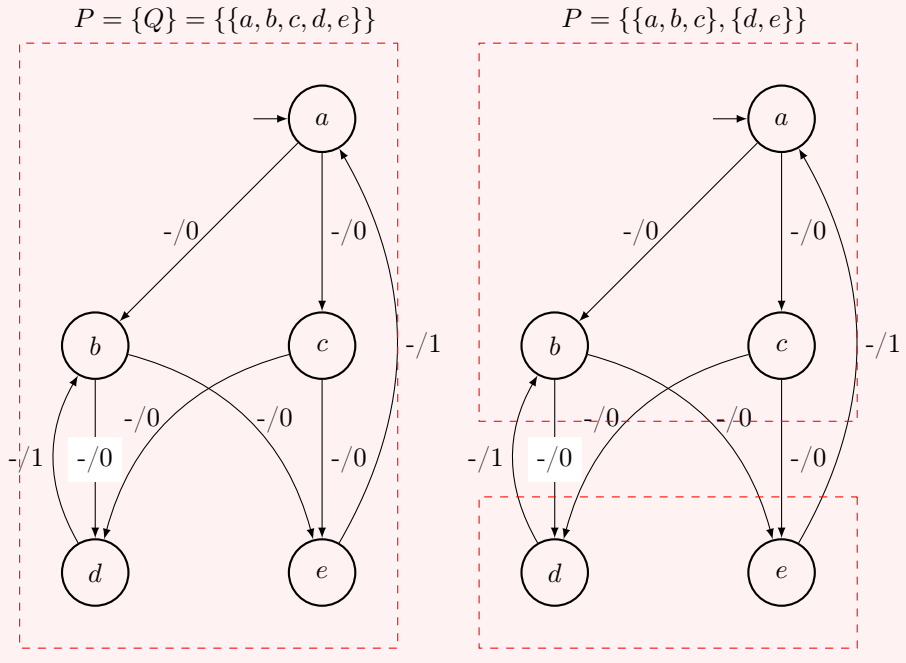
Lo scopo della minimizzazione è quello di ottenere, a partire da un automa a stati non-deterministico M, un automa minimize(M) con il minor numero di stati e bisimile a M.

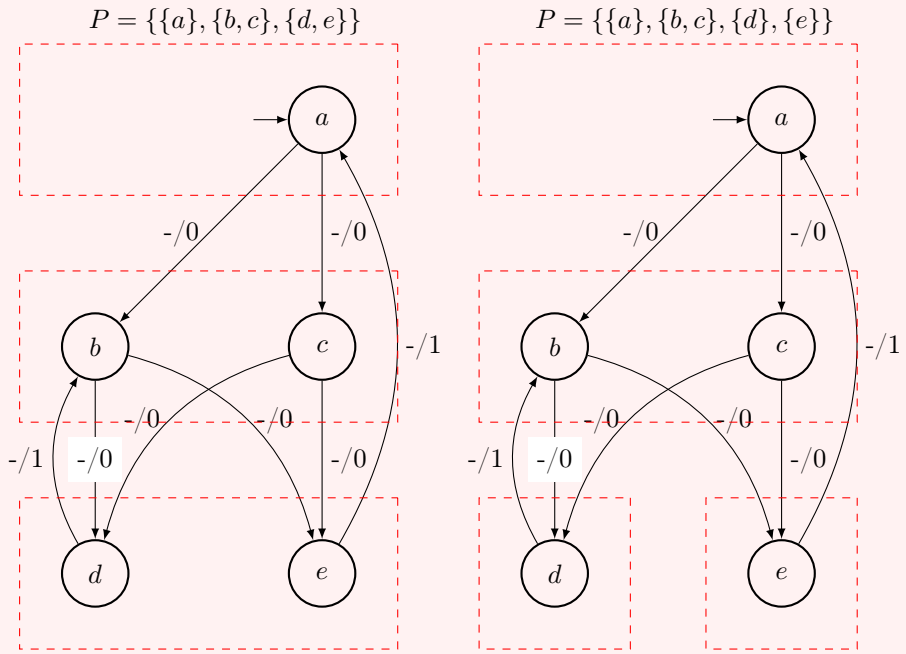
Algoritmo di minimizzazione Descriviamo ora tutti i passaggi dell'algoritmo di minimizzazione per automi a stati non-deterministici:

1. Sia Q l'insieme di tutti gli stati raggiungibili di M;
2. Costruire un insieme di stati P nel seguente modo:
   I. Inizialmente $P = \{Q\}$
   II. Ripetere finché possibile "split P" come segue:
   presi due insiemi di stati $R, R' \in P$, se $r_1 \in R$ per un certo ingresso $x \in \mathit{Ingressi}$ porta ad uno stato $r' \in R'$ con una certa uscita $y \in \mathit{Uscite}$ e invece $r_2 \in R$ per quella stessa coppia ingresso-uscita non porta a nessuno stato $r' \in R'$, allora:
   (a) $R_1 = \{\, r \in R \mid \exists r' \in R', (r', y) \in \mathit{PossUpdates}[M](r, x) \,\}$;
   (b) $R_2 = R \setminus R_1$;
   (c) $P = (P \setminus \{R\}) \cup \{R_1, R_2\}$;
3. Un volta fatto questo, ogni insieme in P rappresenta un singolo stato della più piccola macchina a stati non-deterministica, bisimilare a M.

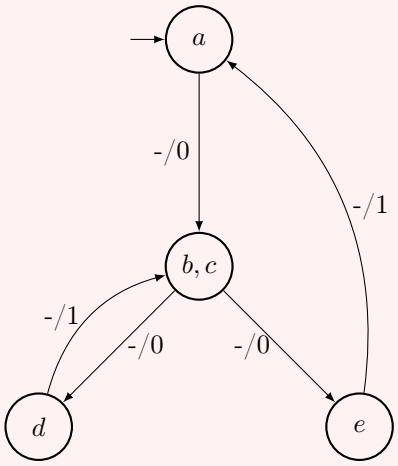
Esempio

Prendiamo in esame il seguente automa non-deterministico M:





L'automa a stati minimo bisimilare a M risulta dunque:



3.8 Automi pseudo-nondeterministici

Una macchina a stati finiti nondeterministica è detta ***pseudo nondeterministica*** se per ogni tripla ingresso/stato/uscita esiste al massimo uno stato successivo.

4 | Reti di Petri

4.1 Definizione

Possiamo definire le reti di Petri come un grafo *bipartito* (i nodi sono di due tipi e gli archi vanno sempre da un tipo all'altro) *orientato* (ogni arco ha un verso) *pesato* (ogni arco ha un peso), attraverso una 5-tupla $N = (P, T, A, w, x_0)$, dove:

- P è un insieme finito di stati chiamati ***posti***;
- T è un insieme finito di chiamati ***transizioni***;
- A è un insieme di archi, $A \subseteq (P \times T) \cup (T \times P)$;
- w è una funzione che associa un "peso" ad ogni arco, $w \colon A \to \mathbb{N}$;
- $\vec{x}_0 = [x(p_1), \ldots, x(p_n)]$ è il vettore di marcatura iniziale, $\vec{x}_0 \in \mathbb{N}^{|P|}$, che mantiene le informazioni dei ***token*** nella rete.

L'insieme $I(t) = \{\, p \in P \mid (p,t) \in A \,\}$ è l'insieme dei ***posti di input*** della transizione $t \in T$, cioè l'insieme degli stati che raggiungono t. L'insieme $O(t) = \{\, p \in P \mid (t,p) \in A \,\}$ è l'insieme dei ***posti di output*** della transizione $t \in T$, cioè l'insieme degli stati raggiunti da t. Una marcatura è una descrizione della dsitribuzione dei token nella rete in un certo istante.

Una transizione t è ***abilitata*** nello stato $\vec{x}$ se vale la ***regola di scatto***, cioè se

$$x(p) \geq w(p,t), \forall p \in I(t)$$

ovvero nel caso in cui i token siano in numero sufficiente per passare ai prossimi stati attraverso un arco con peso minore.

Formalmente la ***funzione di transizione*** $G : (\mathbb{N}^n \times T) \to \mathbb{N}^n$ è definita come

$$G(\vec{x}, t) = \begin{cases} \vec{x}' & \text{se } \forall p \in I(t),\ x(p) \geq w(p,t) \\ \vec{x} & \text{altrimenti} \end{cases}$$

$$\vec{x}' = [x'(p_1), \ldots, x'(p_n)]$$

$$\text{con } x'(p_i) = x(p_i) - w(p_i, t) + w(t, p_i)\ \forall i \in [0, n)$$

cioè quando una transizione $t \in T$ è abilitata, essa scatta consumando token nei suoi stati in input $I(t)$, in base agli archi entranti, e posiziona altri token nei suoi stati in output $O(t)$, in base agli archi uscenti; i token così distribuiti sono descritti nella nuova marcatura $\vec{x}'$.

Uno stato $\vec{y}$ è ***immediatamente raggiungibile*** da $\vec{x}$ se esiste una transizione $t \in T$ tale che $G(\vec{x}, t) = \vec{y}$. Viene detto ***insieme di raggiungibilità*** il più piccolo insieme di stati $R(\vec{x})$ definito come

1. $\vec{x} \in R(\vec{x})$
2. se $\vec{y} \in R(\vec{x})$ e $\vec{z} = G(\vec{y}, t)$ per qualche $t \in T$, allora $\vec{z} \in R(\vec{x})$

cioè è il più piccolo insieme di stati che si possono raggiungere a partire da $\vec{x}$.

Data una rete di Petri con n transizioni ed m stati, un vettore $\vec{u} = [0, \ldots, 1, \ldots, 0]$ di lunghezza n posto a 1 solo nel i-esimo elemento è detto ***vettore di scatto*** (*firing vector*) per la i-esima transizione. Definiamo la ***matrice di incidenza*** $\mathcal{A}$ come la matrice $n \times m$ formata dagli elementi

$$a_{i,j} = w(t_i, p_j) - w(p_j, t_i)$$

e l'equazione di stato può essere scritta come

$$\vec{x}' = \vec{x} + \vec{u}\mathcal{A}$$

che ci permette quindi di valutare il prossimo stato con una semplice moltiplicazione in base a quale transizione vogliamo abilitare.

Una rete di Petri è un ottimo modello per rappresentare esecuzioni concorrenti e può dunque rappresentare un'automa a stati finiti qualsiasi.

4.2 Analisi e proprietà

4.2.1 Boundedness

Un posto $p \in P$ in una rete di Petri $N = (P, T, A, w, \vec{x}_0)$ è ***k-bounded*** o ***k-safe*** se

$$\forall \vec{y} \in R(\vec{x}_0) : y(p) \leq k.$$

Una rete di Petri è *k-bounded* o *k-safe* se tutti i posti $p \in P$ sono *k-bounded*.

4.2.2 Conservazione (conservation)

Una rete di Petri è ***strettamente conservativa*** se

$$\forall \vec{y} \in R(\vec{x}_0) : \sum_{p \in P} y(p) = \sum_{p \in P} x_0(p).$$

Una rete di Petri con n posti è ***conservativa rispetto ad un vettore di pesi*** $\vec{\gamma} = [\gamma_1, \ldots, \gamma_n]$ con $\gamma_i \in \mathbb{N}$ se

$$\forall p \in P : \sum_{i=1}^{n} \gamma_i x(p) = const \quad \text{con } \vec{x} \in R(\vec{x}_0).$$

Una rete di Petri è *conservativa* se è conservativa rispetto ad un vettore di pesi maggiori di zero per ogni posto.

4.2.3 Vitalità (liveness)

Data rete di Petri con $\vec{x}$ stato raggiungibile da $\vec{x}_0$, si dice

L0-live (viva a livello 0) nello stato $\vec{x}$ una transizione t che non può scattare in nessuno stato raggiungibile da $\vec{x}$ (cioè è in deadlock);

L1-live (viva a livello 1) nello stato $\vec{x}$ una transizione t che potenzialmente può scattare dato $\vec{x}$ (cioè è abilitata a scattare);

L2-live (viva a livello 2) nello stato $\vec{x}$ una transizione t se per ogni $n \in \mathbb{N}$ esiste una sequenza di scatto in cui t scatta n volte;

L3-live (viva a livello 3) nello stato $\vec{x}$ una transizione t se esiste un'infinita sequenza di scatto in cui t scatta infinite volte;

L4-live (viva a livello 4) nello stato $\vec{x}$ una transizione t che è *L1-live* in ogni $\vec{y} \in R(\vec{x})$.

Una rete di Petri è *viva a livello i* se ogni sua transizione t è viva a livello i.

4.2.4 Persistenza (persistence)

Due transizioni sono ***persistenti l'una rispetto l'altra*** se, quando entrambe sono abilitate, lo scatto di una non disabilita l'altra.

Una rete di Petri è *persistente* se qualsiasi coppia di transizioni sono persistenti l'una rispetto all'altra.

4.2.5 Copertura (coverability)

Data una rete di Petri e due stati arbitrari $\vec{x}$ e $\vec{y}$, lo stato $\vec{x}$ ***copre*** lo stato $\vec{y}$ se in $\vec{x}$ vegono abilitate tutte le transizioni abilitate in $\vec{y}$, cioè

$$\forall p \in P : x(p) \geq y(p).$$

Lo stato $\vec{x}$ ***copre strettamente*** lo stato $\vec{y}$ se $\vec{x}$ copre $\vec{y}$ e in aggiunta almeno uno stato ha più token, cioè

$$\exists p \in P : x(p) > y(p).$$

Sia $\vec{x} \in R(\vec{x}_0)$; uno stato $\vec{y}$ è *copribile* da $\vec{x}$ sse $\exists \vec{x}' \in R(\vec{x}) : \forall p \in P,\ \vec{x}'(p) \geq y(p)$.

4.2.6 Albero di copertura

Data una rete di Petri $N = (P, T, A, w, \vec{x}_0)$, un suo albero di copertura è un albero dove

- ogni ramo rappresenta una transizione $t \in T$;
- ogni nodo rappresenta uno stato "ω-arricchito";
- il nodo radice è il vettore $\vec{x}_0$;
- ogni nodo foglia è uno stato ω-arricchito nel quale non ci sono transizioni abilitate;
- ogni *nodo duplicato* è uno stato ω-arricchito che già esiste altrove nell'albero;
- ogni ramo è un arco t che connette due nodi $\vec{x}$ e $\vec{y}$ nell'albero sse lo scatto di t nello stato $\vec{x}$ porta a $\vec{y}$.

Algoritmo dell'albero di copertura Data una rete di Petri $N = (P, T, A, w, \vec{x}_0)$:

1. Impostiamo $L = \vec{x}_0$, lista dei nodi aperti;
2. Finché $L \neq \emptyset$, estraiamo un nodo $\vec{x}$ da L e
   I. Se $\forall t \in T : G(\vec{x}, t) = \vec{x}$, allora $\vec{x}$ è un nodo terminale [vai allo step 2]
   II. altrimenti per ogni $\vec{x}' \in G(\vec{x}, t)$, con $t \in T$ e $\vec{x} \neq \vec{x}'$
      I. Se $x(p) = \omega$, allora imposta $x'(p) = \omega$;
      II. Se $\vec{y}$ è un nodo duplicato, tale che $\vec{x}'$ copre $\vec{y}$ ed esiste un percorso da $\vec{y}$ a $\vec{x}'$, allora imposta $\forall p \in P. x'(p) > y(p) : x'(p) = \omega$;
      III. Se $\vec{y}$ non è un nodo duplicato, allora imposta $L = L \cup \vec{x}'$.

Boundness e safety Una rete di Petri è k-bounded se nel suo albero di copertura non appare mai il simbolo ω; se l'albero contiene un ω, possono essere identificati cicli di transizioni per eccedere a un dato k-bound, ma l'albero non ci infoma su quanti cicli siano necessari.

Conservazione Una rete di Petri è conservativa se per ogni nodo del suo albero di copertura è valida la definizione di conservazione rispetto ad un vettore di pesi; se in uno stato è presente il simbolo ω, il suo peso corrispondente γ_i sarà 0.

Per trovare il ***vettore di conservazione***, imposto $\gamma_i = 0$ per ogni p_i unbounded; per b posti bounded ed r nodi nell'albero di conservazione, impostiamo r equazioni con $b + 1$ variabili incognite

$$\sum_{r}^{i=0} \gamma_i x(p_i) = C$$

e risolviamo il sistema per un certo vettore γ con un certo C.

Copertura e raggiungibilità Il problema della copertura può essere risolto ispezionando l'albero di copertura: la più corta sequenza di transizioni che porta ad uno stato coprente può infatti essere trovata efficientemente tramite algoritmi sugli alberi. Il problema della raggiunbilità in generale non può essere risolto tramite l'albero di copertura.

5 | Signal Transition Graph

MISSING

6 | Linguaggi

6.1 Definizioni

Alfabeto Dati dei ***simboli*** σ appartenenti ad un qualsiasi dominio, chiamiamo ***alfabeto*** l'insieme E in cui sono contenuti. Definiamo anche E^* come la ***chiusura di Kleene*** dell'alfabeto E, ovvero l'insieme di tutte le concatenazioni possibili tra gli elementi di E; in parole povere, E^* è l'insieme di tutte le stringhe rappresentabili usando l'alfabeto E, compresa la stringa vuota ε.

Stringa Una sequenza finita $s \in E^*$ di simboli $\sigma \in E$ viene detta ***stringa*** o *parola*. Diamo delle nozioni intuitive sulle stringhe: per due stringhe $s_1 = \sigma_2\sigma_3\sigma_1\sigma_1\sigma_5$ ed $s_2 = \sigma_5\sigma_4$ definiamo

appartenenza $\sigma_i \in s_1$ la notazione per dire che σ_i è presente in s_1;

lunghezza $|s_1|$ il numero di simboli (incluse ripetizioni) in s_1, quindi $|s_1| = 5$;

sottostringa $s \in s_1$ se s è presente in s_1, quindi $\sigma_3\sigma_1\sigma_1$ è sottostringa di s_1;

prefisso p una sottostringa di s_1 che parte dal suo primo simbolo, quindi $\sigma_2\sigma_3\sigma_1$ è un prefisso di s_1;

suffisso t una sottostringa di s_1 che finisce al suo ultimo simbolo, quindi $\sigma_1\sigma_5$ è un suffisso di s_1;

concatenazione s_1s_2 il porre s_2 come suffisso a s_1;

ripetizione σ^n la notazione per $\sigma\sigma\ldots\sigma$ n volte.

Linguaggio Dato un alfabeto E, definiamo ***linguaggio*** un qualsiasi sottoinsieme $L \subseteq E^*$ e definiamo *parole del linguaggio* tutte le parole s tali che $s \in L$. Per i linguaggi valgono le stesse nozioni intuitive date per le stringhe, ovviamente adattate al contesto, in particolare vediamo i prossimi due concetti.

Insieme dei prefissi Dato un linguaggio L, l'insieme dei suoi prefissi $\overline{L}$ è definito come

$$\overline{L} := \{s \in E^* : \exists t \in E^* \text{ t.c. } st \in L\}$$

e rappresenta l'insieme delle stringhe che portano ad un qualsiasi stato dell'automa; notiamo quindi che vale *sempre* $L \subseteq \overline{L}$ (poiché tutte le stringhe $s \in L$ sono anche prefissi per $\varepsilon \in E^*$). Un linguaggio L è detto ***chiuso a prefisso*** se $L = \overline{L}$.

Esempio

Se $L = \{abc, cde\}$ allora $\overline{L} = \{\varepsilon, a, ab, abc, c, cd, cde\}$.

Post-linguaggio Dato un linguaggio L, il post-linguaggio L/s dopo la stringa $s \in L$ è definito come

$$L/s := \{t \in E^* : \exists s \in E^* \text{ t.c. } st \in L\}$$

in pratica è l'insieme dei suffissi che il linguaggio L permette alla stringa s di avere.

Linguaggi senza conflitti Dati due linguaggi L_1 ed L_2, essi si dicono *senza conflitti* se

$$\overline{L_1 \cap \mathcal{L}_2} = \overline{L_1} \cap \overline{L_2}$$

cioè se le stringhe raggiungibili da entrambi i linguaggi sono proprio quelle in comune tra quelle raggiungili dai singoli linguaggi. Questo ci dice che entrambi i linguaggi hanno un "percorso" di stringhe in comune.

Linguaggi regolari Un linguaggio K è detto ***regolare*** se esiste un automa a stati finiti deterministico G che lo ***marca***, cioè se

$$\mathcal{L}_m(G) = K$$

Un linguaggio regolare $K = \mathcal{L}(G)$ è ***chiuso*** (cioè produce un altro linguaggio regolare $K' = \mathcal{L}(G')$) sotto le seguenti operazioni

Chiusura di Kleene K^*: per costruire G' viene creato un nuovo stato iniziale marcato e, insieme ad ogni stato marcato di G, vengono collegati tramite ε-transizioni ad ogni stato iniziale di G. Gli stati iniziali di G vengono poi rimossi, mantenendo come unico stato iniziale quello aggiunto;

Complementazione K^{comp}: viene complementato G, quindi $G' = G^{comp}$.

Due linguaggi regolari $K_1 = \mathcal{L}(G_1)$ e $K_2 = \mathcal{L}(G_2)$ sono *chiusi* (cioè producono un altro linguaggio regolare $K = \mathcal{L}(G)$) sotto le sequenti operazioni

Unione $K_1 \cup K_2$: per costruire G viene creato un nuovo stato iniziale collegato tramite ε-transizioni agli stati iniziali degli automi G_1 e G_2;

Concatenazione K_1K_2: per costruire G tutti gli stati di G_1 vengono smarcati e collegati tramite ε-transizioni agli stati iniziali di G_2;

Intersezione $K_1 \cap K_2$: si usa il prodotto degli automi, quindi $G = G_1 \times G_2$.

Proiezioni naturali Una proiezione naturale è una funzione nella forma

$$P_i : (E_1 \cup E_2)^* \to E_i^* \quad \text{con } i = 1, 2$$

definite come

$$P_i(\varepsilon) = \varepsilon$$
$$P_i(\sigma) = \begin{cases} \sigma & \text{se } \sigma \in E_i \\ \varepsilon & \text{se } \sigma \notin E_i \end{cases}$$
$$P_i(s\sigma) = P_i(s)P_i(\sigma) \text{ per } s \in (E_1 \cup E_2)^*, \sigma \in E_1 \cup E_2$$

cioè è una funzione che, in base al suo selettore i, "filtra" una stringa composta da simboli di due alfabeti E_1 ed E_2 per estrarne una stringa appartenente ad uno solo dei due, cioè ad E_i; in pratica cancella tutti i simboli non appartenenti a E_i.

Possiamo estendere la nozione di proiezione ai linguaggi: dato $L \subseteq (E_1 \cup E_2)^*$, la sua proiezione sull'alfabeto E_i è il linguaggio formato dalle proiezioni delle stringhe di L proiettabili su E_i^*, cioè

$$P_i(L) = \{s \in E_i^* : \exists t \in L \text{ t.c. } P_i(t) = s\}.$$

in pratica filtro ogni stringa di L in modo che appartenga solo ad E_i^*.

Proiezione naturale inversa Possiamo definire anche l'operazione inversa di una proiezione, che data una stringa s in un alfabeto E_i ritorna l'insieme delle stringhe che se proiettate su E_i ottengono s, cioè

$$P_i^{-1}(s) = \{t \in (E_1 \cup E_2)^* : P_i(t) = s\}.$$

Anche le proiezioni inverse sono estese ai linguaggi come

$$P_i^{-1}(L_i) = \{s \in (E_1 \cup E_2)^* : \exists t \in L_i \text{ t.c. } P_i(s) = t)\}$$

per $L_i \subseteq E_i^*$. Notiamo inoltre che $P_i[P_i^{-1}(L_i)] = L_i$ ma che $L \subseteq P_i^{-1}[P_i(L)]$ poiché ad esempio $L \in E_1^*$ potrebbe originariamente non contenere tutte le stringhe possibili usando tutto l'alfabeto E_2.

6.2 Automi

Un automa è quindi una struttura matematica che ha la capacità di generare un determinato linguaggio. Nel contesto dei sistemi a eventi discreti, non parleremo più di simboli ma di ***eventi*** e le stringhe saranno dunque chiamate ***tracce***.

Automa Un automa è un tupla di sei elementi $G = (X, E, f, \Gamma, x_0, X_m)$ dove:

- X è l'insieme degli stati;
- E è l'insieme di eventi associati alle transizioni in G;
- $f : X \times E \to X$ è la ***funzione di transizione***; scrivendo $f(x, e) = y$ diciamo che esiste una transizione etichettata con l'evento e che porta dallo stato x allo stato y;
- $\Gamma : X \to 2^E$ è la ***funzione degli eventi attivi***; $\Gamma(x)$ indica l'insieme degli eventi e accettabili nello stato x, cioè per i quali $f(x, e)$ è definita;
- x_0 è lo stato iniziale;
- $X_m \subseteq X$ è l'insieme degli ***stati marcati*** o *finali*.

Notiamo che potrebbero esserci alcuni stati non raggiungibili tramite nessuna stringa s dallo stato iniziale x_0, e quindi sono totalmente inutili: possiamo eliminare tali stati e tutte le loro transizioni per rendere l'automa ***accessibile***, ovvero raggiungibile in ogni suo stato. Da ora, senza perdita di generalità, assumeremo che ogni automa sia accessibile. Inoltre, possiamo fare un ragionamento analogo sugli stati che non raggiungono uno stato finale: eliminare questi stati rendendo l'automa ***coaccessibile*** lascia inalterato $\mathcal{L}_m(G)$, ma sicuramente modifica $\mathcal{L}(G)$, in particolare rende $\mathcal{L}(G) = \overline{\mathcal{L}_m(G)}$. Se un automa è sia accessibile che coaccessibile viene detto ***potato*** (o ***trim***). Un automa coaccessibile è molto utile per modellare i ***deadlock***, ovvero situazioni che si verificano quando un automa è ***bloccante***, cioè quando $\mathcal{L}(G) \neq \overline{\mathcal{L}_m(G)}$.

Linguaggio generato Il linguaggio generato da un automa è definito come l'insieme di tutte le tracce $s = \sigma_1\sigma_2\dots$ per cui $f(f(f(x_0, \sigma_1), \sigma_2), \dots) = y$, cioè come

$$\mathcal{L}(G) = \{s \in E^* : f(x_0, s) \text{ è definita}\}.$$

Notiamo che $\mathcal{L}(G)$ è *sempre* un linguaggio chiuso a prefisso, per definizione.

Linguaggio marcato Il linguaggio marcato da un automa è definito come l'insieme delle tracce che raggiungono uno stato marcato dell'automa, cioè come

$$\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) : f(x_0, s) \in X_m\}.$$

In generale quindi abbiamo che

$$\mathcal{L}_m(G) \subseteq \overline{\mathcal{L}_m(G)} \subseteq \mathcal{L}(G).$$

Equivalenza tra automi Due automi G_1 e G_2 si dicono equivalenti se vale che

$$\mathcal{L}(G_1) = \mathcal{L}(G_2) \wedge \mathcal{L}_m(G_1) = \mathcal{L}_m(G_2).$$

cioè se accettano le stesse tracce e terminimano per le stesse tracce.

6.2.1 Operazioni sugli automi

Complementazione di un automa Dato l'automa

$$G = (X, E, f, \Gamma, x_0, X_m)$$

che marca il linguaggio $K \subseteq E^*$, il suo complemento è l'automa

$$G^{comp} = (X \cup \{x_d\}, E, f_{tot}, \Gamma, x_0, (X \cup \{x_d\}) \setminus X_m)$$

che marca il linguaggio $E^* \setminus K$, dove

$$f_{tot}(x, e) = \begin{cases} f(x, e) & \text{se } e \in \Gamma(x) \\ x_d & \text{altrimenti} \end{cases}$$

x_d è uno stato pozzo (*dump*).

Chiaramente $\mathcal{L}(G^{comp}) = E^*$ e $\mathcal{L}_m(G^{comp}) = E^* \setminus \mathcal{L}_m(G)$, come desiderato.

Prodotto di automi Dati due automi

$$G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1}) \qquad G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$$

il loro prodotto è l'automa risultato

$$G_1 \times G_2 = (X_1 \times X_2, E_1 \cap E_2, f_{1\times 2}, \Gamma_{1\times 2}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

dove

$$f_{1\times 2}\big((x_1, x_2), e\big) = \begin{cases} \big(f(x_1, e), f_2(x_2, e)\big) & \text{se } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \text{indefinita} & \text{altrimenti} \end{cases}$$
$$\Gamma_{1\times 2}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$$

in pratica si creano tutte le coppie (x_1, x_2) possibili di stati tra i due automi e si crea la transizione etichettata σ verso la coppia (y_1, y_2) se e solo se esiste la transizione etichettata σ da x_1 a y_1 e da x_2 a y_2, effettivamente estraendo le transizioni "in comune" tra gli automi.

Il prodotto di automi gode delle seguenti proprietà :

1. $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$
2. $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$

che sono esattamente le definizioni che poi vengono effettivamente utilizzate negli algoritmi per gli automi a stati finiti.

Composizione parallela di automi Dati due automi

$$G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1}) \qquad G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$$

la loro composizione parallela è l'automa risultato

$$G_1 \| G_2 = (X_1 \times X_2, E_1 \cup E_2, f_{1\|2}, \Gamma_{1\|2}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

dove

$$f\big((x_1, x_2), e\big) = \begin{cases} \big(f_1(x_1, e), f_2(x_2, e)\big) & \text{se } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \big(f_1(x_1, e), x_2\big) & \text{se } e \in \Gamma_1(x_1) \setminus E_2 \\ \big(x_1, f(x_2, e)\big) & \text{se } e \in \Gamma_2(x_2) \setminus E_1 \\ \text{indefinita} & \text{altrimenti} \end{cases}$$
$$\Gamma_{1\|2}(x_1, x_2) = \Gamma_1(x_1) \cup \Gamma_2(x_2)$$

La composizione parallela gode delle seguenti proprietà:

1. $\mathcal{L}(G_1 \| G_2) = P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)]$
2. $\mathcal{L}_m(G_1 \| G_2) = P_1^{-1}[\mathcal{L}_m(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)]$
3. $G_1 \| G_2 = G_2 \| G_1$

6.2.2 Determinazione

Dato un automa a stati finiti non deterministico con ε-transizioni

$$G_{nd} = (X, E \cup \{\varepsilon\}, f_{nd}, \Gamma, X_0, X_m)$$

si può trovare un automa a stati finiti deterministico

$$\det(G) = (X', E, f, \Gamma, x_0', X_m')$$

che è a lui equivalente.

Algoritmo di determinazione

1. Impostiamo lo stato iniziale x'_0 come l'unione degli insiemi di stati Q che, a partire da uno stato iniziale $x_0 \in X_0$ accentando ε^*, contengono uno stato iniziale in X_0, quindi

$$x'_0 = \bigcup_{x_0 \in X_0} \{Q | \exists r \in X_0 \text{ t.c. } f(x_0, \varepsilon^*) = Q \wedge r \in Q\};$$

2. Aggiungo lo stato iniziale all'insieme degli stati, $X' = \{x'_0\}$

3. Per ogni stato $x' \in X'$, aggiungo iterativamente ad X' tutti gli insiemi di stati raggiunti tramite le tracce $s = e\varepsilon^*$ a partire dai singoli $x \in x'$, con $e \in E$, cioè $\forall x' \in X'$ applico

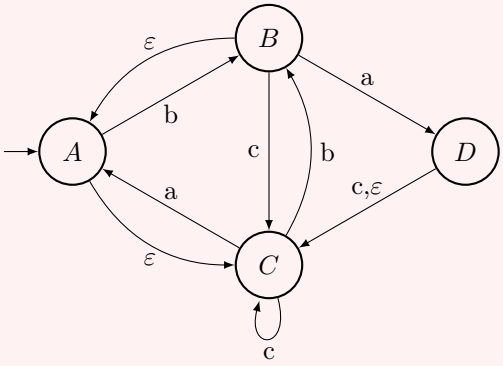
$$X' \leftarrow X' \cup \bigcup_{x \in x'} f(x, e\varepsilon^*)$$

4. Vengono marcati tutti gli stati $x' \in X'$ che contengono almeno uno stato marcato $x_m \in X_m$.

L'algoritmo può essere eseguito utilizzando le tabelle di transizione, dove →A indica uno stato iniziale e (B) uno stato finale.

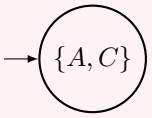
Esempio

Vediamone un esempio per il seguente automa G:



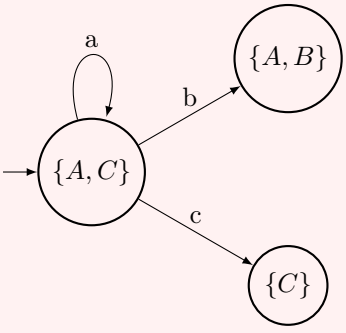
S\I	a	b	c	ε^*
→A	-	B	-	A,C
B	D	-	C	A,B
(C)	A	B	C	C
(D)	-	-	C	C,D

Esecuzione del passo 1 (e passo 2), $X' = \{\{A, C\}\}$:



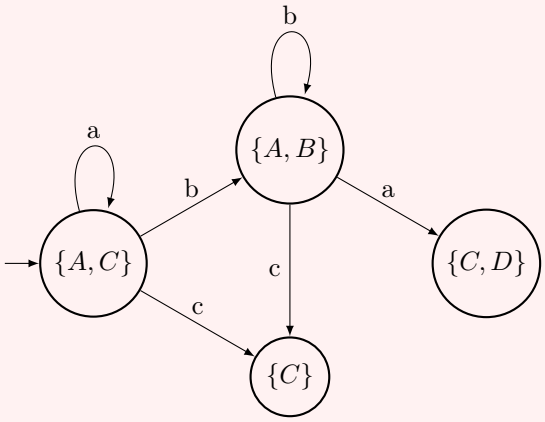
$$\begin{aligned} x'_0 &= \{f(A, \varepsilon^*) = \{A, C\} \wedge A \in \{A, C\}\} \\ &= \{A, C\} \end{aligned}$$

Esecuzione iterata del passo 3 per lo stato $x' = \{A, C\} \in X'$, aggiunge $\{A, B\}$ e $\{C\}$:



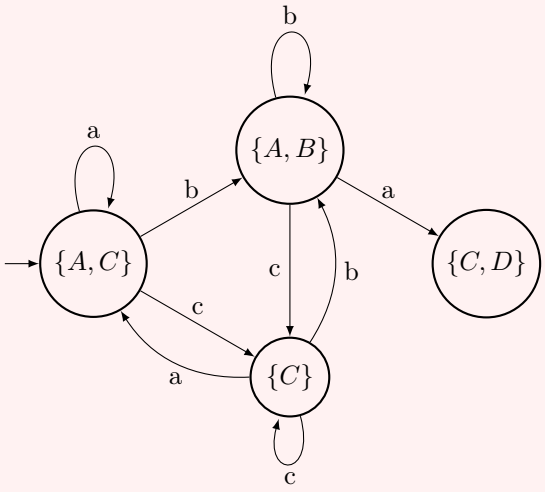
S\I	aε^*	bε^*	cε^*
→({A,C})	A,C	A,B	C

Esecuzione iterata del passo 3 per lo stato $x' = \{A, B\}$, aggiunge $\{C, D\}$:



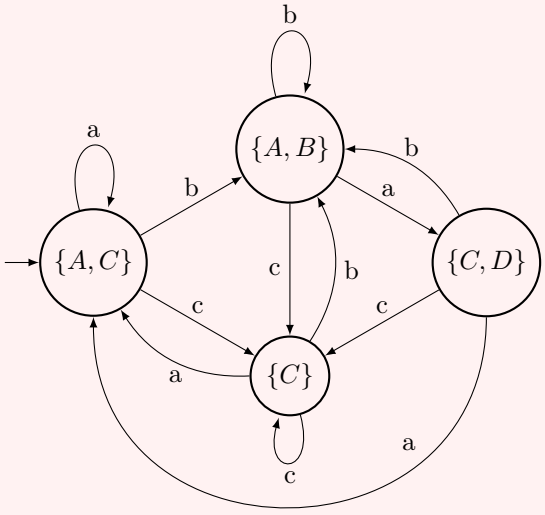
S\I	aε*	bε*	cε*
→{A,C}	A,C	A,B	C
{A,B}	C,D	A,B	C
{C}			

Esecuzione iterata del passo 3 per lo stato $x' = \{C\}$:



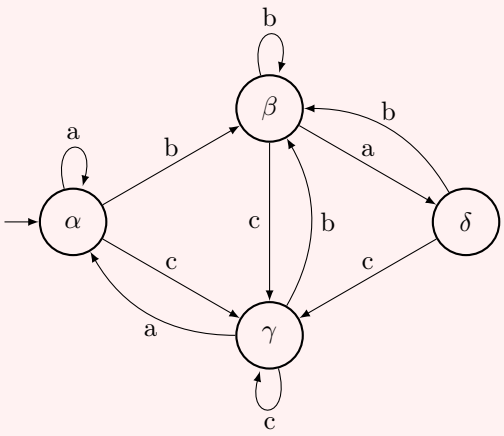
S\I	aε*	bε*	cε*
→{A,C}	A,C	A,B	C
{A,B}	C,D	A,B	C
{C}	A,C	A,B	C
{C,D}			

Esecuzione iterata del passo 3 per lo stato $x' = \{C, D\}$:



S\I	aε*	bε*	cε*
→{A,C}	A,C	A,B	C
{A,B}	C,D	A,B	C
{C}	A,C	A,B	C
{C,D}	A,C	A,B	C

Si possono ora rinominare gli stati ed eliminare ε dall'alfabeto:



S\I	a	b	c
→(α)	α	β	γ
β	δ	β	γ
(γ)	α	β	γ
(δ)	α	β	γ

6.3 Automa osservatore

Dato un sistema a eventi discreti modellato da un automa *possibilmente* non deterministico $G_{nd} = (X, E \cup \{\varepsilon\}, f_{nd}, \Gamma, X_0, X_m)$, possiamo dividere gli eventi come $E = E_o \cup E_{uo}$, dove:

- E_o sono gli ***eventi osservabili***, cioè l'insieme degli eventi che possiamo misurare (ad esempio, un sensore che registra il lampeggio di un LED);
- E_{uo} sono gli ***eventi non osservabili***, comprese le ε-transizioni.

Vogliamo ora stimare lo stato di G_{nd} solamente in base alle tracce degli eventi osservabili, utilizzando un ***osservatore***.

Algoritmo di costruzione di un osservatore Sia $G_{nd} = (X, E \cup \{\varepsilon\}, f_{nd}, \Gamma, x_0, X_m)$ un automa non deterministico dato sugli eventi $E = E_o \cup E_{uo}$.
L'osservatore $G_{obs} = (X_{obs}, E_o, f_{obs}, x_{0,obs}, X_{m,obs})$ per G_{nd} è l'automa che ne ignora tutti gli eventi E_{uo}, ed è costruito seguendo:

1. Rimpiazzare tutte le transizioni di G_{nd} su eventi in E_{uo} con ε-transizioni;
2. Impostiamo $X_{obs} = 2^X \setminus \emptyset$;
3. Per ogni stato $x \in X$ si definisce $UR(x) = f_{nd}(x, \varepsilon)$, detto l'***insieme raggiungibile di stati non osservabili*** (*unobservable reach*), cioè l'insieme di tutti gli stati raggiunti da x tramite una ε-transizione;
   Per un insieme B definiamo $UR(B) = \bigcup_{x \in B} UR(x)$;
4. Impostiamo $x_{0,obs} = UR(x_0)$
5. Per ogni $S \subseteq X$ e $e \in E$ definiamo
$$f_{obs}(S, e) = UR(\{x \in X : \exists x_e \in S[x \in f_{nd}(x_e, e)]\})$$
6. Impostiamo $X_{m,obs} = \{S \subseteq X : S \cap X_m \neq \emptyset\}$.

Un algoritmo molto più semplice è il seguente

1. Rimpiazzare tutte le transizioni di G_{nd} su eventi in E_{uo} con ε-transizioni;
2. Eseguire l'algoritmo di determinazione.

L'automa G_{obs} così ottenuto ha le sequenti proprietà:

- G_{obs} è un automa deterministico sugli eventi E_o;
- $\mathcal{L}(G_{obs}) = P_o[\mathcal{L}(G_{nd})]$ con P_o proiezioni naturale su E_o;
- $\mathcal{L}_m(G_{obs}) = P_o[\mathcal{L}_m(G_{nd})]$;

e le ultime due ci mostrano come un automa non-deterministico abbia lo stesso potere di modellazione di un automa deterministico.

6.4 Controllo supervisore

6.4.1 Definizione

Un sistema a eventi discreti può evolvere senza rispettare un *comportamento legale*, cioè avendo la possibilità di ricevere eventi non controllabili; tali sistemi possono non soddisfare le proprietà di *safety* o *nonbloccanza*, in quanto tali eventi possono portare il sistema fuori dal comportamento voluto.

Comportamento legale Definiamo il *comportamento legale* di un sistema come il linguaggio *ammissibile*

$$L_a \subseteq \mathcal{L}(G)$$

formato dalle stringhe che permettono al sistema di soffisfare la proprietà di *safety*; se il sistema presenta deadlock, cioè non viene soddisfatta la proprietà di *nonbloccanza*, si userà invece

$$L_{am} \subseteq \mathcal{L}_m(G)$$

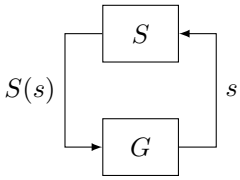
in quanto gli eventi non controllabili potrebbero impedire di raggiungere gli stati marcati. Sia L_a che L_{am} sono linguaggi arbitrariamente definiti in base alle necessità del progettista del sistema, e quindi avranno appunto delle proprietà che vogliamo siano rispettate all'interno del sistema, ad esempio per $L_a = \{a^m b a^n | m \geq n \geq 0\}$ vogliamo avere una stringa in cui il simbolo b divide due porzioni di cui la prima maggiore o uguale della seconda.

Eventi controllabili e non Dividiamo gli eventi E di un sistema in $E = E_c \cup E_{uc}$ dove

- E_c sono gli eventi ***controllabili***, cioè che possono essere evitati, disabilitati, impediti (ad esempio la richiesta di aumentare la velocità di un aereo già al suo massimo);
- E_{uc} sono gli eventi ***non controllabili***, cioè che non possono essere evitati in nessun modo (ad esempio errori hardware, prelazionamenti, guasti, tick del clock, ...).

6.4.2 Controllo centralizzato in totale osservabilità degli eventi

Diciamo che, dato un sistema G con eventi controllabili E_c e non controllabili E_{nc}, esso ha di base un ***comportamento incontrollato***, il quale viene controllato da un ***supervisore*** (o *controllore*) esterno S, posto in un anello chiuso con G; tale sistema controllato è denotato S/G.



Supervisore Dato un sistema a eventi discreti G, un *supervisore* (o controllore) è una funzione nella forma $S : \mathcal{L}(G) \to 2^E$ che data una traccia $s \in \mathcal{L}(G)$ ritorna l'insieme degli eventi che il supervisore permette a G di accettare quando si trova nello stato $y = f(x_0, s)$. L'insieme di eventi $S(s)$ è detto l'***azione di controllo*** raggiunta s, mentre un particolare supervisore S è la ***policy di controllo*** per il sistema G.

Per ogni traccia s, l'insieme degli ***eventi abilitati***, è dato da

$$S(s) \cap \Gamma(f(x_0, s)),\ \forall s \in \mathcal{L}(G)$$

cioè G può eseguire, tra gli eventi accettabili nello stato $f(x_0, s)$, solo quelli presenti anche in $S(s)$, cioè quelli permessi dal supervisore.

Un supervisore è *ammissibile* (cioè ben formato e logicamente coerente) se

$$\forall s \in \mathcal{L}(G),\ E_{uc} \cap \Gamma(f(x_0, s)) \subseteq S(s)$$

cioè se il supervisore S permette a G nello stato $f(x_0, s)$ di accettare anche gli eventi incontrollabili già accettabili da G in quello stato; il supervisore cioè può essere costruito solo se non cerca di impedire eventi incontrollabili.

Linguaggi del sistema controllato Possiamo quindi vedere il sistema S/G come un automa a se state, il cui linguaggio generato è definibile ricorsivamente come

$$\mathcal{L}(S/G) = \{\varepsilon\} \cup \{s\sigma | s \in \mathcal{L}(S/G) \wedge \sigma \in S(s) \wedge s\sigma \in \mathcal{L}(G)\}$$

il quale è chiuso a prefisso per definizione e $\mathcal{L}(S/G) \subseteq \mathcal{L}(G)$.

Il linguaggio marcato da S/G invece è definito come

$$\mathcal{L}_m(S/G) = \mathcal{L}(S/G) \cap \mathcal{L}_m(G)$$

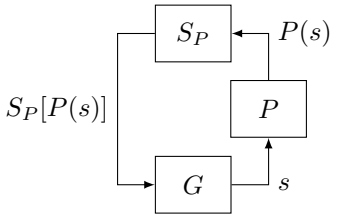
ed è cioè l'insieme delle tracce marcate di G che "sopravvivono" al controllo di S. In generale abbiamo che

$$\emptyset \subseteq \mathcal{L}_m(S/G) \subseteq \overline{\mathcal{L}_m(S/G)} \subseteq \mathcal{L}(S/G) \subseteq \mathcal{L}(G).$$

Diciamo che S è ***nonbloccante*** se S/G è *nonbloccante*, per esempio se $\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)}$, altrimenti S è detto ***bloccante***.

6.4.3 Controllo centralizzato in parziale osservabilità degli eventi

Poniamoci ora nel contesto in cui S non possa "osservare" tutti gli eventi generabili da G, quindi nel caso in cui gli eventi siano partizionabili come $E = E_o \cup E_{uo}$ (dove gli eventi non osservabili sono interpretabili come "interni" al sistema o non registrati dai sensori disponibili). In questo caso, per modellare la situazione si antepone la proeizione naturale P al nuovo supervisore ad S_P, che quindi potrà controllare solo gli eventi osservabili E_o filtrati dalle tracce s prodotte da G.



P-supervisore Dato un sistema a eventi discreti G con eventi non osservabili, un *P-supervisore* è una funzione nella forma $S_P : P[\mathcal{L}(G)] \to 2^E$, cioè è un supervisore che può agire soltanto in base agli eventi osservabili di G. Ogni volta che si verifica un evento osservabile, l'*azione di controllo* viene aggiornata *istantaneamente* dopo a quell'evento e rimane immutata fino all'arrivo di altro evento osservabile.

Data una traccia $t = t'\sigma$ con $\sigma \in E_o$, $S_P(t)$ è l'azione di controllo che viene applicata a tutte le tracce di $\mathcal{L}(G)$ ottenute da $P^{-1}(t')\{\sigma\}$ e a tutte le continuazioni non osservabili di tali tracce (poiché S_P mantiene l'azione di controllo fino al prossimo evento osservabile). In ogni caso, $S_P(t)$ può disabilitare eventi non osservabili e dunque evitare alcune di queste continuazioni non osservabili. Definiamo quindi

$$L_t = P^{-1}(t')\{\sigma\}(S_P(t) \cap E_{uo})^* \cap \mathcal{L}(G)$$

come l'insieme delle tracce di $\mathcal{L}(G)$ che sono effettivamente soggette all'azione di controllo $S_P(t)$ quando S_P controlla G.

Un P-supervisore è *ammissibile* se, come per un ordinario supervisore, non cerca di disabilitare eventi non controllabili, che in questo caso significa che

$$\forall t = t'\sigma \in P[\mathcal{L}(G)],\ E_{uc} \cap \Big[\bigcup_{s \in L_t} \Gamma(f(x_0, s))\Big] \subseteq S_P(t).$$

dove il termine tra parentesi rappresenta tutti le possibili continuazione in $\mathcal{L}(G)$ di tutte le tracce su $S_P(t)$ agisce. Il linguaggio generato e marcato da S_P/G sono definiti allo stesso modo di quelli per S/G (ovviamente nelle espressioni si avrà S_P anziché S).

6.5 Teoremi di esistenza

Date le definizioni precedenti e le considerazioni fatte, possiamo stipulare i seguenti teoremi, essenziali per l'intera teoria dei sistemi.

6.5.1 Controllabilità

La *condizione di controllabilità* è un concetto centrale nel controllo supervisore e può essere intesa come "se non puoi evitare un evento, allora dovrebbe essere legale". Più formalmente, diamo la definizione di

Controllabilità rispetto a un linguaggio

Dati K ed $M = \overline{M} = \mathcal{L}(G)$ linguaggi su E con eventi non controllabili E_{uc}, diciamo che K *è controllabile rispetto a* M *ed* E_{uc} se $\forall s \in \overline{K},\ \forall \sigma \in E_{uc}$ si ha

$$s\sigma \in M \implies s\sigma \in \overline{K}$$

che è equivalente a dire $\bar{K}E_{uc} \cap M \subseteq \bar{K}$.

Osserviamo che la controllabilità è una proprietà della chiusura a prefisso di un linguaggio, quindi K è controllabile se e solo se $\overline{K}$ è controllabile.

Teorema di controllabilità
(o di esistenza di un controllore)

Dato un sistema a eventi discreti G dove $E_{uc} \subseteq E$ è l'insieme dei suoi eventi incontrollabili, con $K \subseteq \mathcal{L}(G)$ sottolinguaggio *non vuoto* per G, diciamo che esiste un supervisore S tale che

$$\mathcal{L}(S/G) = \bar{K}$$

se e solo se vale

controllabilità: K è controllabile rispetto a $\mathcal{L}(G)$ e E_{uc}.

Il teorema è provato costruttivamente e quindi, se la condizione di controllabilità è soddisfatta, ci permette di costruire il supervisore che ottiene il comportamento richiesto. Tale supervisore S ha azione di controllo

$$S(s) = [E_{uc} \cap \Gamma(f(x_0, s))] \cup \{\sigma \in E_c : s\sigma \in \bar{K}\}.$$

Se fosse necessario controllare un linguaggio marcato, dovremmo considerare la chiusura di $\mathcal{L}_m(G)$, il che aggiungerebbe un'altra condizione. Vediamo formalmente questo concetto nel seguente teorema:

Teorema di controllabilità non-bloccante
(o di esistenza di un controllore non-bloccante)

Dati un sistema ad eventi discreti G dove $E_{uc} \subseteq E$ è l'insieme degli eventi incontrollabili, con $K \subseteq \mathcal{L}_m(G)$ sottolinguaggio *non vuoto* delle tracce che marcano G, diciamo che esiste un supervisore nonbloccante S tale che

$$\mathcal{L}_m(S/G) = K \qquad [\implies \mathcal{L}(S/G) = \bar{K}\ ]$$

se e solo se vale valgono

controllabilità: K è controllabile rispetto a $\mathcal{L}(G)$ e E_{uc}.

$\mathcal{L}_m(G)$-chiusura: K è $\mathcal{L}_m(G)$-chiuso (ad esempio $K = \overline{K} \cap \mathcal{L}_m(G)$).

La dimostrazione è costruttiva, come la precedente, e quindi il supervisore S avrà azione di controllo

$$S(s) = [E_{uc} \cap \Gamma(f(x_0, s))] \cup \{\sigma \in E_c : s\sigma \in \bar{K}\}\ .$$

6.5.2 Controllabilità e Osservabilità

Prendiamo ora in esame il caso in cui alcuni eventi non fossero osservabili e descriviamo la *condizione di osservabilità*, che va intesa come "se non puoi distinguere due tracce, allora dovrebbero ricevere la stessa azione di controllo", che dal punto di vista degli eventi si traduce in "se devi evitare un evento, allora non dovresti perdere nulla di necessario per il comportamento legale". Più formalmente, diamo la definizione di

Osservabilità rispetto a un linguaggio

Dati K ed $M = \overline{M} = \mathcal{L}(G)$ linguaggi su E con eventi controllabili E_c e osservabili E_o, con P la proiezione naturale da E^* a E_o^*, diciamo che K *è osservabile rispetto a* M, E_o *ed* E_c se $\forall s \in \overline{K}$, $\forall \sigma \in E_c$ si ha

$$(s\sigma \notin \overline{K}) \wedge (s\sigma \in M) \implies P^{-1}[P(s)]\{\sigma\} \cap \overline{K} = \emptyset$$

La parte destra dell'implicazione indentifica tutte le tracce di $\overline{K}$ che hanno la stessa proiezione di s e possono essere continuate con σ. Se tale insieme non è vuoto, allora esistono due tracce s e s' che hanno la stessa proiezione $P(s) = P(s')$ ma tali che $s\sigma \notin \overline{K}$ e $s'\sigma \in \overline{K}$; tali tracce richiederebbero dunque due azioni di controllo diverse (disabilitare σ per s, abilitare σ per s'), ma a causa dell'osservabilità ristretta un P-supervisore non saprebbe distinguere tra s ed s' e non sarebbe quindi possibile costruire un P-supervisore che ottenga esattamente $\overline{K}$.

Possiamo anche verificare l'osservabilità di K costruendo l'osservatore di $H \times G$ (dove $K = \mathcal{L}(H)$ e $M = \mathcal{L}(G)$) e controllando che ogni stato di $(H \times G)_{obs}$ non abbia ***conflitti di controllo***, ovvero l'osservatore non abbiamo uno stato i cui stati interni necessitano di due differenti azioni di controllo sullo stesso evento controllabile.

Come per la controllabilità, notiamo che l'osservabilità è una proprietà della chiusura a prefisso di un linguaggio, quindi K è osservabile se e solo se $\overline{K}$ è osservabile.

Teorema di controllabilità e osservabilità
(o di esistenza di un controllore non-bloccante con osservabilità parziale)

Dato un sistema a eventi discreti G dove $E_{uc} \subseteq E$ è l'insieme dei suoi eventi incontrollabili e E_o l'insieme dei suoi eventi osservabili e P la proiezione naturale da E^* a E_o^*, con $K \subseteq \mathcal{L}(G)$ sottolinguaggio *non vuoto* per G, diciamo che esiste un P-supervisore nonbloccante S_P tale che

$$\mathcal{L}_m(S_P/G) = K$$

se e solo se valgono

controllabilità: K è controllabile rispetto a $\mathcal{L}(G)$ e E_{uc}.

osservabilità: K è osservabile rispetto a $\mathcal{L}(G)$, P e E_c;

$\mathcal{L}_m(G)$-chiusura: K è $\mathcal{L}_m(G)$-chiuso (ad esempio $K = \overline{K} \cap \mathcal{L}_m(G)$).

La dimostrazione è costruttiva, e quindi il P-supervisore S_P avrà azione di controllo

$$S_P(s) = E_{uc} \cup \{\sigma \in E_c : \exists t\sigma \in \overline{K} \text{ t.c. } P(t) = s\} .$$

6.6 Realizzazione del supervisore

Nei problemi reali ci troveremo di fronte ad un ***impianto*** G, un automa già esistente, che vogliamo ci comporti come ad una ***specifica*** H_a, l'automa che vorremmo avere: per forzare questo comportamento *potremmo* essere in grado di costruire un supervisore S in grado di controllare gli eventi di G e farlo comportare come se fosse H_a, che è esattamente quello che vogliamo, proprio usando i teoremi visti sopra.

La *realizzazione* di un supervisore S per l'automa G tale che $\mathcal{L}(S/G) = \overline{K}$ non è altro che la sua rappresentazione come automa, che dovrà appunto riconoscere il linguaggio $\overline{K}$, dunque dato l'automa

$$R = (Y, E, g, \Gamma_R, y_0, Y) \quad \text{con} \quad \mathcal{L}_m(R) = \mathcal{L}(R) = \overline{K}$$

questo sarà l'automa che "codifica" l'azione di controllo $S(s)$, e si avrà

$$S(s) = \Gamma_{R\times G}(g \times f((y_0, x_0), s)) = \Gamma_R(g(y_0, s))$$

con $f \times g$ la funzione di transizione per la composizione $R \times G$. Chiameremo l'automa R la ***realizzazione standard*** di S. In pratica, data la specifica H_a per l'automa G, il supervisore S per G è codificato nell'automa $R = H_a \times G$.

MAYBE SOMETHING'S MISSING

6.7 Gestire l'incontrollabilità

Quando un linguaggio K non è controllabile rispetto ad M ed E_{uc}, possiamo "cercare nei dintorni" del linguaggio K dei linguaggi a lui riconducibili ma controllabili.

6.7.1 Proprietà della controllabilità

Esistono due tipi di linguaggi derivati da K:

- $K^{\uparrow C}$ il *il sottolinguaggio controllabile supremo di* K;
- $K^{\downarrow C}$ il *il sovralinguaggio controllabile prefisso-chiuso infimo di* K.

Abbiamo i seguenti rapporti:

$$\emptyset \subseteq K^{\uparrow C} \subseteq K \subseteq \overline{K} \subseteq K^{\downarrow C} \subseteq M$$

- Se K^1 e K^2 sono controllabili, allora $K^1 \cup K^2$ è controllabile;
- Se K^1 e K^2 sono controllabili, allora $K^1 \cap K^2$ non ha bisogno di essere controllabile;

- Se K^1 e K^2 sono non in conflitto e controllabili, allora $K^1 \cap K^2$ è controllabile.
  (si ricorda che K^1 e K^2 si dicono non in conflitto qualora $\overline{K^1} \cap \overline{K^2} = \overline{K_1 \cap K_2}$);
- Se K^1 e K^2 sono prefisso-chiusi e controllabili, allora lo è anche $K_1 \cap K_2$.

6.7.2 Riguardo il sottolinguaggio supremo

Definiamo la classe dei *sottolinguaggi controllabili di K* come

$$\mathcal{C}_{in}(K) := \{L \subseteq K : \overline{L}E_{uc} \cap M \subseteq \overline{L}\}.$$

$\mathcal{C}_{in}(K)$ è un *poset* (insieme parzialmente ordinato) chiuso sotto unioni arbitrarie e possiede un unico elemento *supremo* definito come:

$$K^{\uparrow C} := \bigcup_{L \in \mathcal{C}_{in}(K)} L$$

1. Nel caso peggiore, $K^{\uparrow C} = \emptyset$, dal momento che $\emptyset \in \mathcal{C}_{in}(K)$;
2. Se K è controllabile, allora $K^{\uparrow C} = K$
3. Osserviamo che $K^{\uparrow C}$ non necessita di essere prefisso-chiuso in generale

Dati due linguaggi $K = \mathcal{L}(H)$ e $M = \mathcal{L}(G)$, l'automa $H_{\uparrow C}$ tale che $\mathcal{L}(H_{\uparrow C}) = K^{\uparrow C}$ è ottenuto costruendo $H_0 = H \times G$ e poi eliminando tutti gli stati (h, g) se non hanno un evento incontrollabile che però è presente in g (è l'algoritmo di potatura applicato su H_0).

6.7.3 Riguardo il sovralinguaggio infimo

Definiamo la classe dei *sovralinguaggi controllabili prefisso-chiusi di K* come

$$\mathcal{CC}_{out}(K) := \{L \subseteq E^* : (K \subseteq L \subseteq M) \wedge (\overline{L} = L) \wedge (\overline{L}E_{uc} \cap M \subseteq \overline{L})\}.$$

$\mathcal{CC}_{out}(K)$ è un *poset* chiuso sotto intersezioni e unioni arbitrarie e possiede un unico elemento *infimo* definito come:

$$K^{\downarrow C} := \bigcap_{L \in \mathcal{CC}_{out}(K)} L = \overline{K}E^*_{uc} \cap M$$

1. Nel caso peggiore, $K^{\downarrow C} = M$, dal momento che $M \in \mathcal{CC}_{out}(K)$.
2. Se K è controllabile, allora $K^{\downarrow C} = \overline{K}$.

Dati due linguaggi $K = \mathcal{L}(H)$ e $M = \mathcal{L}(G)$, l'automa $H_{\downarrow C}$ tale che $\mathcal{L}(H_{\uparrow C}) = K^{\uparrow C}$ è ottenuto dalla definizione $\overline{K}E^*_{uc} \cap M$, costruendo gli automi per $\overline{K}$ e per E^*_{uc}, concatenandoli e infine facendo il prodotto con G.