Indice

Pagina

1 Nozioni di Base

1.1 Relazioni Binarie

Riportiamo la definizione di *relazione binaria* su uno o due insiemi, che sarà utile per definire formalmente il concetto di *grafo*, fondamentale all'interno di questo elaborato:

Definizione 1.1. Chiameremo *relazione binaria* su A, B qualsiasi sottoinsieme del prodotto cartesiano $A \times B$.
Chiameremo *relazione binaria* su A qualsiasi sottoinsieme del prodotto cartesiano $A \times A$.
Diremo che R mette *in relazione* u, v se $(u, v) \in R$. In questo caso potremo usare la notazione uRv.

Definizione 1.2. Chiameremo *insieme immagine* di un elemento di A l'insieme definito come $R(x) = \{y \in B : xRy\}$.

Alcune relazioni binarie mostrano proprietà fondamentali, che presentiamo nella definizione seguente:

Definizione 1.3. Sia R una relazione binaria su A. Diremo che R è

- *Riflessiva* se $\forall x \in A, xRx$;
- *Simmetrica* se $xRy \implies yRx$ $(x, y \in A)$
- *Transitiva* se $(xRy \land yRz) \implies xRz$ $(x, y, z \in A)$

Esempio 1.1. La relazione "$\leq$" sui naturali è riflessiva e transitiva, ma non simmetrica.
La relazione "$=$" ($a = b \iff$ "a, b sono lo stesso numero") sui naturali è simmetrica, riflessiva e transitiva.

Definizione 1.4. Una relazione binaria riflessiva, simmetrica e transitiva si dice *relazione di equivalenza.*
Una relazione di equivalenza divide un insieme in *classi di equivalenza* all'interno delle quali tutte le coppie di elementi sono in relazione.

In alcune situazioni risulta conveniente definire la più piccola relazione che dispone di una certa proprietà, e che contiene una relazione binaria R. Chiameremo una relazione costruita in questo modo "*chiusura*":

Definizione 1.5. Sia R una relazione binaria su A. Definiamo le seguenti chiusure:

– *Riflessiva*: $R_r = R \cup \{(x,x)\ \forall x \in A\}$

– *Simmetrica*: $R_s = R \cup \{(y,x)\ \forall x,y : (x,y) \in R\}$

– *Transitiva*: $R_t = R \cup \{(x,z)\ \forall x,z : \exists y : (x,y) \in R \wedge (y,z) \in R\}$

Esempio 1.2. La chiusura riflessiva della relazione "$<$" (minore stretto) è la relazione "$\leq$".

Nel seguito sarà fondamentale la definizione seguente:

Definizione 1.6. Sia R una relazione binaria su $A \times B$. Chiameremo *relazione inversa* di R o *contro-immagine* di un elemento di B l'insieme definito come $R^{-1}(y) = \{x \in A : xRy\}$.

Useremo frequentemente la notazione "$R^{-1}(X)$" o "$R(X)$" dove X è un insieme. In questo caso l'insieme risultante è dato dall'unione di tutti gli insiemi $R(x)$ o $R^{-1}(x)$, per ogni $x \in X$.
Introduciamo la seguente notazione:

Definizione 1.7. Sia A un insieme. Denoteremo con "$|A|$" la *cardinalità* di A, cioè il numero dei suoi elementi.
Analogamente, data una relazione binaria R, denoteremo con $|R|$ il numero delle coppie messe in relazione da R.

1.2 Grafi

Con queste premesse possiamo definire un *grafo* come segue:

Definizione 1.8. Sia V un insieme finito non vuoto. Sia E una relazione binaria su V. Chiameremo *grafo diretto* o *grafo orientato* la coppia $G = (V, E)$. Con questa notazione:

– V è l'insieme dei *nodi* o *vertici*;

– E è una relazione binaria che mette in relazione alcuni dei nodi di G.

Esempio 1.3. Il grafo di Figura 1 è descritto dalla coppia

– $V = \{a, b, c, d, e\}$

– $E = \{(a,b), (a,d), (b,c), (d,c), (c,e), (d,e)\}$

Nel seguito utilizzeremo ampiamente la seguente terminologia:

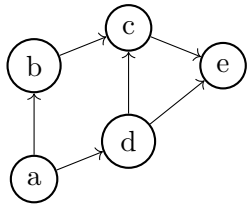


Figura 1: Rappresentazione grafica di un grafo

Definizione 1.9. Sia $G = (V, E)$ un grafo. Diremo che un nodo $u \in V$ è una *foglia* se $\nexists v \in V : uEv$. Diremo che u è *parente* di v e che v è *figlio* di u se uEv.

Un grafo è quindi un insieme di elementi (i *nodi*) accoppiato con un insieme di relazioni tra questi elementi (gli *archi* o *rami*). È naturale associare questo concetto all'idea di percorso: ogni grafo è definito da un insieme di nodi ed un insieme di *cammini* che consentono di spostarsi da un nodo ad un altro. La seguente definizione sorge in modo spontaneo da questa interpretazione:

Definizione 1.10. Sia $G = (V, E)$ un grafo. Siano $u, v \in V$. Diremo che v *è raggiungibile* da u, o in alternativa *esiste un cammino* da u a v, o ancora uE_tv (la t in pedice sta per "*transitivo*"), se esiste una sequenza finita di nodi x_n di lunghezza $K : x_0 = u, x_K = v, x_nEx_{n+1}$.

L'esistenza di un cammino tra nodi fornisce un criterio immediato per partizionare un grafo in gruppi di nodi. Diamo innanzitutto la seguente definizione:

Definizione 1.11. Diremo che un grafo (V, E) è *fortemente connesso* se $\forall v_1, v_2 \in V, v_1E_tv_2$.

Possiamo individuare facilmente i sottografi massimali fortemente connessi:

Definizione 1.12. Le *componenti fortemente connesse* (*strongly connected components*, *SCC*) di un grafo (V, E) sono le classi di equivalenza della relazione E_t [2, Appendice B].

Conseguenza di questa definizione è che i nodi contenuti in una stessa componente fortemente connessa sono mutuamente raggiungibili.

Esempio 1.4. Nel grafo di Figura 2 le SCC sono rappresentate con forme diverse: $\{a, b, c, d\}, \{e\}$.

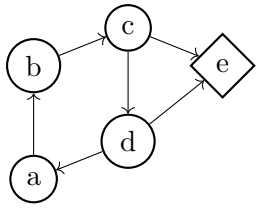


Figura 2: SCC di un grafo

Dato un grafo, possiamo definire il partizionamento dei nodi in SCC come segue:

Definizione 1.13. Sia $G = (V, E)$ un grafo. Definiamo il grafo $G^{SCC} = (V^{SCC}, E^{SCC})$ delle componenti fortemente connesse:

- $V^{SCC} = \{C : \text{“}C \text{ è una classe di equivalenza per } E_t \text{ su } V\text{”}\}$
- $E^{SCC} = \{(A, B) \in V^{SCC} \times V^{SCC} : A \neq B, \exists m \in A, n \in B : mEn\}$

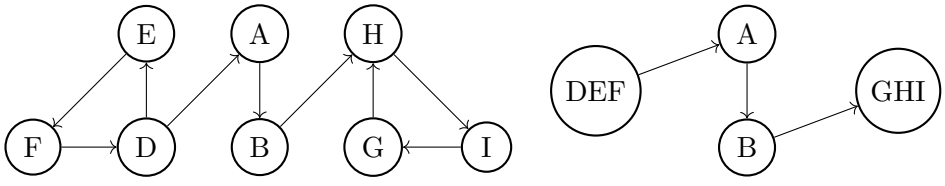
Riportiamo la seguente proprietà immediata:

Proposizione 1.1. *Sia G^{SCC} il grafo delle SCC di un grafo G generico. Allora G^{SCC} è aciclico.*

Dimostrazione. Supponiamo per assurdo che in G^{SCC} esista un ciclo. Allora tutti i nodi di V^{SCC} facenti parte del ciclo sono mutuamente raggiungibili (percorrendo il ciclo). Quindi tutti i nodi fanno parte della stessa SCC, ma questo è assurdo. □

Esempio 1.5. La Figura 3.a rappresenta un grafo generico, la Figura 3.b rappresenta il suo grafo delle componenti fortemente connesse associato.

Dato un grafo generico possiamo determinare la partizione in SCC sfruttando un algoritmo avente complessità lineare $\Theta(|V| + |E|)$ [8]. L'algoritmo non verrà trattato in questo elaborato.



(a) Un grafo (b) Corrispondente grafo delle SCC

Figura 3: Un grafo ed il corrispondente grafo delle SCC

1.3 Insiemi

1.3.1 Cenni di teoria degli insiemi

In generale supporremo validi gli assiomi su cui si fonda la teoria degli insiemi ZFC, ad eccezione dell'Assioma di Fondazione. Formuliamo innanzitutto i seguenti assiomi:

Assioma 1.1 (di estensionalità)**.** Due insiemi sono uguali $\iff$ contengono gli stessi elementi.

Assioma 1.2 (di fondazione)**.** Ogni insieme non vuoto contiene un elemento disgiunto dall'insieme stesso.

Del primo faremo un uso esplicito nel seguito. Il secondo, per motivi che saranno evidenti nella sezione seguente, risulta limitante nell'ambito trattato in questo elaborato. Introduciamo la seguente definizione:

Definizione 1.14. Diremo che un insieme è *ben-fondato* se non contiene se stesso. Altrimenti diremo che è *non-ben-fondato*.

Esempio 1.6. L'insieme $\Omega = \{\Omega\}$ è non-ben-fondato. L'insieme $A = \{1, 2, 3\}$ è ben-fondato.

Riportiamo una formulazione equivalente dell'Assioma 1.2:

Assioma (1.2 bis)**.** $\forall A$ la relazione "$\in$" è ben-fondata su A [6, Chapter III.4]

Da questa questa formulazione risulta evidente l'impossibilità, in ZFC, di costruire insiemi non-ben-fondati.
Rinunciando all'Assioma 1.2 si ottiene un sistema di assiomi che ammette l'esistenza di insiemi non-ben-fondati; tuttavia questo sistema non è più sufficiente a descrivere in modo esaustivo l'aritmetica per mezzo di operazioni su insiemi. Per ovviare a questa mancanza si introduce l'Assioma AFA, che verrà presentato e discusso nella sezione seguente.

1.3.2 Rappresentazione di insiemi tramite grafi

In alcuni casi risulta conveniente fornire un'interpretazione insiemistica della nozione di grafo vista sopra. Introduciamo innanzitutto una nozione fondamentale:

Definizione 1.15. Sia $G = (V, E)$ un grafo orientato. Sia $u \in V : \forall v \in V, uE_tv$, cioè ogni nodo di G è raggiungibile da u. Allora la terna (V, E, u) si dice *accessible pointed graph*, o *APG*.

Per rappresentare un insieme tramite un grafo è necessario definire un processo denominato *decorazione*:

Definizione 1.16. Chiameremo *decorazione* di un APG l'assegnazione di un insieme ad ogni suo nodo. In tal caso associamo la relazione "$\in$" alla relazione di raggiungibilità "E", ovvero $aEb \iff b \in a$.

Possiamo ora dare la seguente definizione:

Definizione 1.17. Chiameremo *immagine* (o *picture* in [1]) di un insieme A la coppia composta da un APG (G, v) e da una decorazione, in cui a v è associato A.

Vale la seguente proposizione, dimostrata in [1]:

Proposizione 1.2. *Ad un APG aciclico è possibile associare un'unica decorazione.*

Questo risultato non è stato dimostrato nel caso di un APG contenente almeno un ciclo. Per questo motivo proponiamo il seguente assioma:

Assioma 1.3 (AFA, Anti-Foundation-Axiom)**.** Ogni APG possiede un'unica decorazione.

L'assioma AFA ha un'ovvia conseguenza:

Corollario 1.1. *Ogni APG è immagine di un unico insieme.*

Esempio 1.7. In Figura 4 sono rappresentati alcuni insiemi sotto forma di APG.

1.4 Bisimulazione

In questa sezione introdurremo la definizione di bisimulazione ed alcune proprietà immediate. In seguito esamineremo la relazione tra la teoria degli insiemi e la bisimulazione.

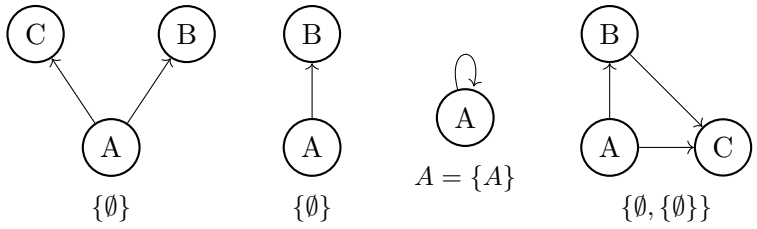


Figura 4: Rappresentazione di insiemi tramite grafi

1.4.1 Definizione e risultati generali

Definizione 1.18. Siano $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ due grafi. Diremo che una relazione binaria $R : V_1 \times V_2$ è una *bisimulazione* su G_1, G_2 se $\forall a \in V_1, b \in V_2$ valgono congiuntamente le seguenti proprietà:

- $aRb, aE_1a' \implies \exists b' \in V_2 : (a'Rb' \wedge bE_2b')$
- $aRb, bE_2b' \implies \exists a' \in V_1 : (a'Rb' \wedge aE_1a')$

Possiamo definire in modo analogo una bisimulazione su un unico grafo G, ponendo $G_1 = G_2 = G$.

Definiamo un'importante caratteristica di una coppia qualsiasi di grafi, che verrà sfruttata ampiamente nel seguito

Definizione 1.19. Siano $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ due grafi. Diremo che sono *bisimili* se $\exists R : V_1 \times V_2 : R$ è una bisimulazione su G_1, G_2.
Diremo che due APG $(G_1, v_1), (G_2, v_2)$ sono *bisimili* se G_1, G_2 sono bisimili e vale v_1Rv_2 per almeno una bisimulazione su G_1, G_2.

Osservazione 1.1. Una bisimulazione può non essere riflessiva, simmetrica, nè transitiva.

Esempio 1.8. La relazione $aRb \iff$ "a, b sono lo stesso nodo" su un grafo qualsiasi è una bisimulazione riflessiva, simmetrica e transitiva.
La relazione $R = \emptyset$ è una bisimulazione su un grafo qualsiasi, ma non è riflessiva.
La relazione $R = \{(a,a),(b,b),(c,c),(d,d),(a,b),(b,c),(c,d)\}$ sul grafo $G = (\{a,b,c,d\}, \{(a,b),(b,c),(c,d),(d,d)\})$ è una bisimulazione, ed è solamente riflessiva.

Dalla definizione di bisimulazione possiamo dedurre una proprietà interessante di una qualsiasi sua chiusura:

Teorema 1.1. *Sia R una bisimulazione sul grafo G. La sua chiusura riflessiva, simmetrica o transitiva è ancora una bisimulazione su G.*

Dimostrazione. Consideriamo separatamente le tre relazioni R_r, R_s, R_t, rispettivamente la chiusura riflessiva, simmetrica e transitiva:

R_r Per definizione $R \subset R_r$, quindi è sufficiente dimostrare che R_r è una bisimulazione quando gli argomenti $u, v \in V$ non sono distinti.
Sia $u \in V$. Chiaramente per definizione di R_r si ha uR_ru. Se $\exists u' \in V : uEu'$ allora (sempre per definizione di R_r) si ha $u'R_ru'$.

R_s Per definizione $R \subset R_s$, quindi è sufficiente dimostrare che R_s è una bisimulazione quando per gli argomenti $u, v \in V$ si ha uRv ma non vRu.
Sia $(u, v) \in V \times V$. Allora

$$uR_sv \implies uRv \vee vRu$$

Supponiamo ad esempio che vRu.

$$\begin{aligned} &\implies \forall v' \in V : (vEv')\ \exists u' \in V : (uEu' \wedge v'Ru') \\ &\implies u'R_sv' \end{aligned}$$

e

$$\begin{aligned} &\implies \forall u' \in V : (uEu')\ \exists v' \in V : (vEv' \wedge v'Ru') \\ &\implies u'R_sv' \end{aligned}$$

cioè sono dimostrate le due condizioni caratteristiche della bisimulazione.
La dimostrazione è analoga se uRv.

R_t Per definizione $b \subset R_t$, quindi è sufficiente dimostrare che R_t è una bisimulazione quando per gli argomenti $u, v, z \in V$ si ha uRv, vRz ma non uRz.
Sia $(u, v, z) \in V \times V \times V$ con questa proprietà. Allora $\forall u' \in V : uEu' \implies \exists v' \in V : vEv' \wedge u'Rv'$. Inoltre $\exists z' : zEz' \wedge v'Rz'$.
Riordinando si ha $u'Rv', v'Rz'$. Allora per definizione di b_t, $u'R_tz'$.
In modo speculare si ottiene la seconda condizione caratteristica della bisimulazione.

□

Da questa proposizione si deduce il seguente corollario, che risulta dall'applicazione iterativa delle tre chiusure viste in precedenza:

Corollario 1.2. *Ad ogni bisimulazione R è possibile associare una bisimulazione $\widetilde{R} : R \subset \widetilde{R} \ \wedge\ \widetilde{R}$ è una relazione di equivalenza.*

Concludiamo la sezione relativa ai risultati generali sulla bisimulazione con la seguente proposizione, che sarà utile nel seguito:

Proposizione 1.3. *Siano R_1, R_2 due bisimulazioni su G_1, G_2. Allora $R = R_1 \cup R_2$ è ancora una bisimulazione.*

Dimostrazione. Siano $u, v : uRv$. Sia $u' : uEu'$. Allora deve essere $uR_1v \vee uR_2v$. Ma quindi $\exists v' : (vEv' \wedge u'R_{1|2}v')$. □

1.4.2 Bisimulazione massima

Definiamo ora il concetto di *bisimulazione massima*, che gioca un ruolo chiave nella risoluzione dei problemi considerati in questo elaborato:

Definizione 1.20. Diremo che una bisimulazione R_M su G_1, G_2 è *massima* se $\forall R :$ "R è una bisimulazione su G_1, G_2" si ha $uRv \implies uR_Mv$.

Osservazione 1.2. Sia R_M la bisimulazione massima su un grafo G. Allora per qualsiasi altra bisimulazione R su G si ha $|R_M| > |R|$.

Naturalmente la bisimulazione massima dipende dai due grafi presi in esame. Possiamo dedurre alcune caratteristiche in modo molto semplice:

Proposizione 1.4. *Valgono le seguenti proprietà:*

1. *La bisimulazione massima su due grafi G_1, G_2 è unica;*
2. *La bisimulazione massima è una relazione di equivalenza.*

Dimostrazione. Le proprietà seguono dal Corollario 1.2 e dall'Osservazione 1.3:

1. Supponiamo per assurdo che esistano due bisimulazioni massime R_{M_1}, R_{M_2}. La loro unione è ancora una bisimulazione, che è "più massima" delle supposte bisimulazioni massime.
2. Se per assurdo la bisimulazione massima non fosse una relazione di equivalenza, potremmo considerare la sua chiusura riflessiva, simmetrica e transitiva, che sarebbe "più massima" ed anche una relazione di equivalenza.

□

Naturalmente il concetto di *bisimulazione massima* può essere definito anche su unico grafo G. Questo caso si rivelerà di grande interesse nel seguito. Per ora dimostriamo il seguente risultato:

Teorema 1.2. *Sia G un grafo (finito). Allora $\exists R_M$ la bisimulazione massima su G.*

Dimostrazione. Può esistere solamente un numero finito di relazioni binarie su G, e questo numero fornisce un limite superiore al numero massimo di bisimulazioni su G. Allora possiamo considerare l'unione di questo numero finito di bisimulazioni, che sarà chiaramente la bisimulazione massima. □

1.4.3 Interpretazione insiemistica della bisimulazione

Mostriamo ora una conseguenza diretta dell'Assioma di Estensionalità e di AFA:

Teorema 1.3. *Due APG rappresentano lo stesso insieme* $\iff$ *sono bisimili.*

Dimostrazione. Dimostriamo separatamente le due implicazioni:

($\Longrightarrow$) Osserviamo innanzitutto che la relazione binaria $\equiv$ su V_A, V_B definita come segue:

$a \equiv b \iff$ "le decorazioni di A, B associano ad a, b lo stesso insieme"

è una bisimulazione sui grafi $G_A = (V_A, E_A), G_B = (V_B, E_B)$.
Chiaramente se $a \equiv b, aEa'$ si ha:

- $a' \in a$, associando ad a, a' gli insiemi che rappresentano secondo la decorazione (l'unica) considerata;
- a, b rappresentano lo stesso insieme.

Quindi per l'Assioma di Estensionalità $\exists b' \in b : b' = a'$, cioè bEb' e $a' \equiv b'$. Si procede specularmente per la seconda condizione caratteristica della bisimulazione.
La relazione $\equiv$ è una bisimulazione sugli APG A, B quando si assume per ipotesi che A, B rappresentino lo stesso insieme.

($\Longleftarrow$) Sia R una bisimulazione su A, B. Consideriamo la decorazione d_A (l'unica) di A. Vogliamo estrapolarne una decorazione per B. Dalla possibilità di operare questo procedimento, dall'Assioma di Estensionalità e da AFA potremo dedurre l'uguaglianza degli insiemi rappresentati.
Definiamo l' applicazione d, che ad ogni nodo di B associa un insieme:

$$d(v) = d_A(u), \text{con } u \text{ nodo di } A : uRv$$

Osservazione. Per ogni nodo v di B deve esistere almeno un nodo u di $A : uRv$ perchè si suppone che i due APG siano bisimili.

Dimostriamo che d è una decorazione di B. In altre parole vogliamo dimostrare che per ogni nodo v di B l'insieme $d(v)$ contiene tutti e soli gli insiemi $d(v')$ $\forall v'$ nodo di $B : vEv'$.

– Supponiamo per assurdo che tra i figli di v "manchi" il nodo corrispondente ad un elemento $X \in d(v)$. Poichè la decorazione di A è ben definita, il nodo u di $A : uRv$ deve avere un figlio corrispondente a X, che chiameremo u'. Ma $uRb, uEu' \implies \exists v' : vEv', u'Rv'$. Cioè $d(v') = X$. Dunque il nodo mancante è stato identificato.

– Supponiamo per assurdo che tra i figli di v ci sia un nodo "in più", ovvero un nodo $v' : vEv', d(v') = Y$ con $Y \notin d(v)$. Ma allora, sempre considerando u il nodo di $A : uRv$, dovrebbe esistere un $u' : uEu', u'Rv'$, cioè $d_A(u') = d(v') = Y$. Ma allora $Y \in d_A(u) = d(v)$, deduzione che è chiaramente in contrasto con l'ipotesi.

E' possibile che ci siano due nodi di a_1, a_2 di A ed un nodo b di B per cui vale $a_1Rb \wedge a_2Rb$. In questo caso la decorazione definita è ambigua. Per questo motivo correggiamo la formulazione di d come segue:

$$d(v) = X, \text{ con } X \text{ l'insieme associato al nodo } u \text{ di } A : uRv,$$
$$\text{con } u \text{ preso casualmente tra i nodi di } A \text{ bisimili a } v.$$

Per AFA esiste un'unica decorazione di b, quindi si deve avere, alternativamente, per ogni nodo v di B:

– $\exists! u$ nodo di $A : uRv$;

– $\forall u$ nodo di $A : uRv$, la decorazione di A associa a tutti gli u lo stesso insieme.

Dunque l'ambiguità è risolta.

□

Tenendo conto di quanto affermato nella sezione 1.3.2, il Teorema 1.3 dimostra che la bisimulazione può sostituire la relazione di uguaglianza tra insiemi quando questi sono rappresentati con APG [3].
Dopo questa considerazone possiamo dare la seguente definizione:

Definizione 1.21. Sia R una bisimulazione su G che sia anche una relazione di equivalenza. Definiamo un nuovo grafo $G_R = (VR, E_R)$ come in [4], che chiameremo *contrazione rispetto alla bisimulazione R di G*:

– $VR = \{A = \{m \in V : \forall n \in A, mRn\}\}$

– $[m]_R E_R [n]_R \iff \exists c \in [n]_R : mEc$

Si definisce *classe del nodo a* rispetto alla bisimulazione R, con la notazione $[a]_R$, il nodo di VR in cui viene inserito il nodo a.

La Definizione 1.21 è di fondamentale importanza per la seguente osservazione:

Proposizione 1.5. *Sia G un grafo, e sia G_R come nella Definizione 1.21, per una bisimulazione R qualsiasi. Allora G, G_R sono bisimili.*

Dimostrazione. Sia $\equiv \subset V \times VR$ la relazione binaria definita come segue:

$$m \equiv M \iff M = [m]_R$$

Vogliamo dimostrare che tale relazione è una bisimulazione sui grafi G, G_R. Supponiamo che $x \equiv X$, e che xEy per qualche $y \in V$. Chiamiamo $Y \coloneqq [y]_R$. Allora, per la Definizione 1.21, si ha XEY. Inoltre vale banalmente $y \equiv Y$.
Per dimostrare la seconda condizione caratteristica della bisimulazione, supponiamo che $x \equiv X$, e che XEY per qualche $Y \in VR$. Sempre per la Definizione 1.21 deve esistere un $y \in Y : (y \equiv Y \wedge xEy)$. □

La Proposizione 1.5 ha una conseguenza ovvia, che risulta evidente per il Teorema 1.3:

Corollario 1.3. *Sia R una bisimulazione che sia anche una relazione di equivalenza. Allora l'APG (G, v) e l'APG $(G_R, [v]_R)$ rappresentano lo stesso insieme.*

Quindi risulta naturale sfruttare le proprietà della bisimulazione per minimizzare la rappresentazione di insiemi, considerando che è sufficiente una bisimulazione sulla rappresentazione iniziale per ottenere una rappresentazione equivalente. Definiamo una relazione d'ordine sulle rappresentazioni:

Definizione 1.22. Diremo che la rappresentazione (G_a, v_a) di un insieme è *minore* della rappresentazione equivalente (G_b, v_b) se $|Va| < |Vb|$.
Diremo che una rappresentazione è *minima* se non esiste una rappresentazione equivalente minore.

Osservazione 1.3. La *contrazione per bisimulazione* è una rappresentazione minore (o eventualmente uguale) di quella iniziale.

Concludiamo la sezione con il seguente risultato, che stabilisce in modo univoco la bisimulazione prescelta per minimizzare la rappresentazione di un dato insieme:

Teorema 1.4. *Sia (G, v) un APG rappresentante un insieme. Sia R_M la bisimulazione massima su (G, v). Allora la contrazione per bisimulazione indotta da R_M su (G, v) fornisce la rappresentazione minima dell'insieme.*

Dimostrazione. Supponiamo per assurdo che esista una bisimulazione R_V su (G, v) che fornisce una contrazione avente un numero di nodi strettamente inferiore alla contrazione indotta da R_M. Ma questo implica che esistono almeno due nodi di G che sono in relazione secondo R_V e non secondo R_M. Chiaramente questa deduzione è in contrasto con il fatto che R_M è la bisimulazione massima.
Supponiamo per assurdo che, dopo la contrazione indotta da R_M, sia possibile trovare una nuova bisimulazione R_O su $(G_{R_M}, [v]_{R_M})$ che induca una contrazione avente un numero di nodi strettamente inferiore a quello di $(G_{R_M}, [v]_{R_M})$. Chiaramente $R_O \subset VR_M \times VR_M$.
Definisco una nuova bisimulazione $R_{\widetilde{M}} \subset V \times V$ tale che

$$xR_{\widetilde{M}}y \iff (xR_My \vee [x]_{R_M} R_O [y]_{R_M})$$

Per definizione di bisimulazione massima bisogna avere $R_{\widetilde{M}} \subset R_M$, quindi non è possibile che la contrazione indotta da R_O sia una rappresentazione minore di quella indotta da R_M. □

1.5 Relational stable coarsest partition

In questa sezione introduciamo il concetto di *RSCP* o *Relational Stable Coarsest Partition* (*partizione "più rozza" stabile rispetto ad una relazione*). Nel seguito del lavoro evidenzieremo il legame tra questo problema e quello della determinazione della bisimulazione massima, che viene sfruttato dagli algoritmi che tratteremo nel seguito in quanto la formulazione di RSCP lo rende un problema più facilmente approcciabile dal punto di vista algoritmico di quanto lo sia quello della determinazione della bisimulazione massima.
Cominciamo con alcune definizioni su cui struttureremo la formulazione del problema:

Definizione 1.23. Sia A un insieme finito. Sia $X = \{x_1, \ldots, x_n\}$ con $x_i \subseteq A \ \forall i \in \{1, \ldots, n\}$. Diremo che X è una *partizione* di A se:

$$\bigcup_{i=1}^{n} x_i = A \qquad \wedge \qquad x_i \cap x_j = \emptyset \ \forall i, j \in \{1, \ldots, n\} \qquad \wedge \qquad x_i \neq \emptyset \ \forall i$$

Inoltre diremo che gli insiemi x_i sono i *blocchi* della partizione X. Se $a \in x_i$ useremo la notazione $[a]_X = x_i$.

Osservazione 1.4. Ogni insieme A ha una partizionamento banale, consistente in un unico blocco contenente tutti gli elementi dell'insieme, che indicheremo con $\widetilde{A}$.

Esempio 1.9. TROVA ESEMPIO

Definizione 1.24. Siano X_1, X_2 due partizioni dello stesso insieme. Diremo che X_2 *rifinisce* X_1 se $\forall x_1 \in X_1,\ \forall x_2 \in X_2$ si ha $x_2 \subseteq x_1$.

Esempio 1.10. Nell'esempio precedente la partizione X_2 rifinisce la partizione X_1, ed entrambe rifiniscono la partizione banale $\widetilde{A}$.

Il partizionamento di insiemi finiti assume interesse nell'ambito trattato in questo lavoro quando si impone la seguente condizione sui blocchi:

Definizione 1.25. Sia A un insieme, ed R una relazione binaria su A. Sia X una partizione su A, e sia $S \subseteq A$. Diremo che X è stabile rispetto alla coppia (S, A) se $\forall x \in X$ vale $x \subseteq R^{-1}(S) \vee x_i \cap R^{-1}(S) = \emptyset$.
Inoltre diremo che X è *stabile* rispetto ad R se è stabile rispetto a qualsiasi coppia formata da R e da un blocco di X.

Esempio 1.11. Sia A come nell'Esempio 1.9. TROVA ESEMPIO

In altre parole per ogni coppia di blocchi di una partizione stabile si hanno due alternative:

- Tutti gli elementi del primo blocco sono in relazione con almeno un elemento del secondo blocco;
- Nessuno degli elementi del primo blocco è in relazione con qualche elemento del secondo blocco.

Vale la seguente osservazione banale:

Osservazione 1.5. Sia X una partizione qualsiasi di un insieme U, e siano $S, T \subseteq U$. Sia R una relazione binaria su U. Supponiamo X stabile rispetto alle coppie $(R, S), (R, T)$. Allora:

1. Qualsiasi rifinitura di P resta stabile rispetto alla coppia (f, S);
2. P è stabile rispetto alla coppia $(f, S \cup T)$.

Dimostrazione. Dimostriamo separatamente i due enunciati:

1. Chiaramente se per un blocco qualsiasi $x \in X$ vale $x \subseteq R^{-1}(S) \vee x \cap R^{-1}(S) = \emptyset$, la stessa relazione vale per qualsiasi sottoinsieme di x;

2. Sia $x \in X$. Chiaramente vale $R^{-1}(S), R^{-1}(T) \subseteq R^{-1}(S \cup T)$. Allora, se almeno uno tra $R^{-1}(S), R^{-1}(T)$ contiene l'immagine di x si avrà $x \subseteq R^{-1}(S \cup T)$, altrimenti $x \cap R^{-1}(S \cup T)$ dovrà chiaramente essere vuoto.

□

Con queste premesse possiamo definire il problema della determinazione della RCSP:

Definizione 1.26. Sia A un insieme, sia R una relazione binaria su A. Sia S una partizione di A. Chiameremo RSCP(R, S) di A la partizione di A stabile rispetto ad R, che rifinisce S e che contiene il minor numero di blocchi (per questo motivo si dice che è la *più rozza*).
Useremo la notazione "RSCP(R)" se la partizione iniziale è quella banale proposta nella Definizione 1.4.

Esempio 1.12. Sia A come nell'Esempio 1.9, e sia R come nell'Esempio 1.11. TROVA ESEMPIO

La seguente definizione ha grande importanza pratica per il problema della determinazione della RSCP:

Definizione 1.27. Sia $\widetilde{S} = \{S_1, \ldots, S_n\}$ una partizione qualsiasi di un insieme finito S. Sia $f : SES$, e sia $A \subset S$. Diremo che A è uno *splitter* di S rispetto a f se

$$\exists S_x \in \widetilde{S} : \qquad S_x \cap f^{-1}(A) \neq \emptyset \ \wedge \ S_x \not\subseteq f^{-1}(A)$$

In tal caso definiamo la seguente funzione:

$$\text{Split}(A, \widetilde{S}) = \{S_x \cap f^{-1}(A), S_x - f^{-1}(A) \quad \forall S_x \in \widetilde{S}\} - \{\emptyset\}$$

Valgono le seguenti osservazioni sulla funzione appena introdotta:

Osservazione 1.6. Se A non è uno *splitter* di $\widetilde{S}$ si ha $\widetilde{S} = \text{Split}(A, \widetilde{S})$.

Osservazione 1.7. Se $\widetilde{S}$ è stabile rispetto ad un insieme A, Split$(B, \widetilde{S})$ è stabile rispetto ad A, B. In altre parole la funzione "Split" preserva la stabilità.

Dimostrazione. Osservando che "Split" può solamente dividere i blocchi, e non mescolarli, la dimostrazione è banale. □

Proponiamo il seguente risultato sulla funzione "Split", che useremo nel seguito:

Teorema 1.5. *Consideriamo la funzione "Split" proposta nella Definizione 1.27:*

1. *La funzione è monotona rispetto al secondo argomento, cioè se $\widetilde{S}$ è una rifinitura di S, allora Split$(A, \widetilde{S})$ rifinisce Split(A, S);*
2. *La funzione è commutativa rispetto al primo argomento: Split$(A,$ Split$(B, S)) =$ Split$(B,$ Split$(A, S))$.*

Dimostrazione. Dimostriamo separatamente gli enunciati:

1. I blocchi di Split$(A, \widetilde{S})$ sono del tipo $x - R^{-1}(A)$ oppure $x \cap R^{-1}(A)$, dove x è un blocco di $\widetilde{S}$. I blocchi di Split(A, S) sono del tipo $y - R^{-1}(A)$ oppure $y \cap R^{-1}(A)$, dove y è un blocco di S. Poichè $\forall x \in \widetilde{S}\ \exists y \in S : x \subseteq y$ si ha $x - R^{-1}(A) \subseteq y - R^{-1}(A)$ e $x \cap R^{-1}(A) \subseteq y \cap R^{-1}(A)$;
2. Conseguenza della commutatività delle operazioni "$-$" e "$\cap$" su insiemi.

□

1.6 Equivalenza tra RSCP e bisimulazione massima

Dimostriamo innanzitutto il seguente risultato preliminare presentato in [4]:

Proposizione 1.6. *Sia $G = (V, E)$. Sia X una partizione di V stabile rispetto a E. Allora la relazione binaria R su V definita come:*

$$aRb \iff [a]_X = [b]_X$$

è una bisimulazione su G.

Dimostrazione. Siano $a, b \in G : aRb$, e sia $a' : aEa'$. Poichè X è stabile, si ha che $[a]_X \subseteq E^{-1}([a']_X)$. Quindi $\exists b' \in [a']_X : bEb'$.
L'altra condizione caratteristica della bisimulazione si dimostra in modo speculare. □

In altre parole, una qualsiasi partizione stabile rispetto a E induce su un grafo una bisimulazione che può essere ricavata in modo banale.
Dimostriamo un altro risultato, che in un certo senso è l'opposto di quello appena presentato:

Proposizione 1.7. *Sia R una bisimulazione su G che sia anche una relazione di equivalenza. Allora la partizione i cui blocchi sono le classi di equivalenza di R è stabile rispetto a E.*

Dimostrazione. Se per assurdo X non fosse stabile esisterebbero due blocchi $x_1, x_2 : x_1 \cap E^{-1}(x_2)$ non è ne x_1 nè $\emptyset$. Chiamiamo A questo insieme.
Gli elementi a di A sono i nodi in $x_1 : \nexists b \in x_2 : aEb$. Ma poichè questi a e gli $x \in x_1 - A$ si trovano all'interno dello stesso blocco x_1 deve valere xRa.
Sia $x \in x_1 - A$, ed $y \in x_2 : xEy$. Poichè $\forall a \in A$ vale xRa, allora $\exists a' : aEa', a'Ry$, cioè $[a']_X = [y]_X = x_2$. Quindi A deve necessariamente essere vuoto. □

Cioè una bisimulazione induce un partizionamento stabile rispetto a E dei nodi del grafo. Vale il seguente corollario:

Corollario 1.4. *Determinare la bisimulazione massima su un grafo $G = (V, E)$ e trovare RSCP(E) di V sono problemi equivalenti.*

Dimostrazione. Dimostriamo separatamente che la bisimulazione ricavata dalla RSCP(E) è massima, e che la partizione ricavata dalla bisimulazione massima è la RSCP(E).

- Sia R_M la bisimulazione massima su G. Per la Proposizione 1.4 è una relazione di equivalenza. Per la Proposizione 1.7 è possibile determinare una partizione X stabile rispetto a E.
  Supponiamo per assurdo che X non sia RSCP(E) di V, quindi esiste una partizione $\widetilde{X}$ stabile rispetto a E che ha meno blocchi di quanti ne ha X. Ma per la Proposizione 1.6 da $\widetilde{X}$ è possibile ricavare una bisimulazione $\widetilde{R}$ su G. Ma quindi $|\widetilde{R}| > |R|$, che è assurdo.

- Sia X la RSCP(E) di V. Supponiamo per assurdo che la bisimulazione R ricavata da X come nella Proposizione 1.6 non sia massima. Allora deve esistere un'altra bisimulazione $\widetilde{R}$ che sia massima. Ma da questa si può ricavare, come nella Proposizione 1.7, una partizione $\widetilde{X}$ stabile rispetto a E per cui vale $|\widetilde{X}| \leq |X|$. Ma questo è assurdo.

Quindi l'equivalenza è dimostrata. □

2 Algoritmi risolutivi

In questa sezione esamineremo alcuni algoritmi risolutivi proposti per la risoluzione del problema della determinazione della massima bisimulazione su un grafo. Come risulterà evidente nel seguito viene sfruttata ampiamente l'equivalenza tra bisimulazione massima e RCSP dimostrata nella sezione precedente.

2.1 Minimizzazione di automi a stati finiti

Esaminiamo innanzitutto l'algoritmo risolutivo per una versione semplificata del problema della determinazione della bisimulazione massima, ovvero la minimizzazione di un'automa a stati finiti, presentato in [5] nel 1971. Sebbene questa soluzione non sia generale, ha fornito alcuni spunti fondamentali per l'ideazione di algoritmi risolutivi più completi, che verranno presentati nel seguito del lavoro.

2.1.1 Alcune nozioni fondamentali

Innanzitutto definiamo il concetto di *automa*, chiaramente centrale nella descrizione del problema:

Definizione 2.1. Consideriamo i seguenti oggetti:

- Un insieme finito I detto *insieme degli ingressi*;
- Un insieme finito S detto *insieme degli stati*, ad ogni stato è associata un'unica uscita;
- Un insieme finito $F \subseteq S$ detto *insieme degli stati finali*;
- Una funzione $\delta : S \times IES$ detta *funzione di trasferimento*.

Chiameremo $A = (S, I, \delta, F)$ *automa*. Useremo la notazione $Out(x)$ per indicare l'output corrispondente allo stato x.

Possiamo rappresentare un'automa con una tabella degli stati, in cui ogni riga rappresenta uno stato, ogni colonna un ingresso, ed ogni cella contiene il nuovo stato del sistema quando, nel momento in cui il sistema si trova nello stato corrispondente alla riga, si inserisce l'ingresso corrispondente alla colonna.In alternativa possiamo utilizzare una rappresentazione grafica, in cui ogni stato è descritto da un cerchio contenente il nome dello stato, e le transizioni tra stati sono descritte da frecce, sulle quali viene specificato

l'ingresso che ha innescato la transizione.

Esempio 2.1. Nella Tabella 1 è rappresentato un'automa che cambia stato (AEB, BEC) solamente se l'ingresso è "1", e lo stato non è "C". In qualsiasi altro caso lo stato non cambia.

	0	1
a[0]	a	b
b[0]	b	c
c[1]	c	c

Tabella 1: Rappresentazione tabellare di un'automa

Esempio 2.2. Nella Figura 5 è rappresentato lo stesso automa dell'Esempio 2.1.

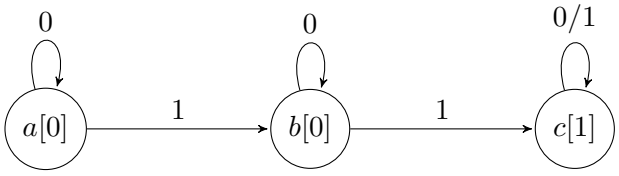


Figura 5: Rappresentazione grafica di un'automa

In alcuni automi è possibile individuare stati che "si comportano in modo simile". Informalmente, il sistema si comporta in modo simile quando riceve in input un ingresso qualsiasi, e si trova in uno degli stati presi in esame. Diamo un esempio di questa situazione:

Esempio 2.3. Consideriamo gli stati dell'automa rappresentato graficamente nella Figura 6. Supponiamo di accorpare gli stati B, C in un unico

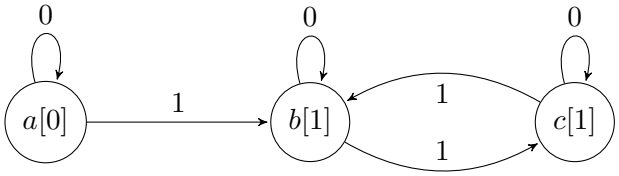


Figura 6: Automa contenente stati equivalenti

stato, che chiamiamo B'. Se ci interessiamo solamente alla sequenza di output ed all'eventuale raggiungimento di uno stato finale, il nuovo automa risulta indistinguibile dal primo.

Proponiamo la definizione formale di equivalenza tra stati che viene utilizzata in [5]. Nel seguito ne dedurremo un'altra, che consente di stabilire un parallelo con gli argomenti trattati nella Sezione 1.

Definizione 2.2. Sia I^* l'insieme di tutte le sequenze di input di lunghezza finita. Sia $\delta^* : S \times I^*$ la funzione di transizione "iterata". Diremo che due stati x, y sono equivalenti (con la notazione "$x \sim y$") se e soltanto se valgono congiuntamente le seguenti condizioni:

1. $Out(x) = Out(y)$;
2. $\forall i^* \in I^*,\ \delta^*(x, i^*) \in F \iff \delta^*(y, i^*) \in F$.

È evidente che individuare gli stati equivalenti consente di minimizzare il sistema, preservando al tempo stesso i risultati ottenuti. In questo lavoro non illustreremo cosa comporta questa definizione, e perchè è importante che per qualsiasi stringa di lunghezza finita si giunga ad uno stato finale partendo da due stati supposti equivalenti.

Osservazione 2.1. La relazione "$\sim$" così definita è una relazione di equivalenza sull'insieme degli stati di un'automa.

La seguente osservazione sarà utile per formulare il problema con la terminologia esposta nella Sezione 1.5:

Osservazione 2.2. Due stati x, y sono equivalenti nel senso della definizione 2.2 se e solo se

$$\forall i \in I, \quad \delta(x, i) \sim \delta(y, i) \tag{1}$$

e $Out(x) = Out(y)$.

Dimostrazione. Supponiamo che $\exists i \in I : \delta(x, i) \not\sim \delta(y, i)$. Allora, ad esempio:

$$\exists i^* \in I^* : \delta^*(\delta(x, i), i^*) = \delta^*(x, ii^*) \in F,\ \delta^*(\delta(y, i), i^*) = \delta^*(y, ii^*) \notin F$$

Quindi, per l'esistenza della stringa ii^* si ha $x \not\sim y$. La dimostrazione è speculare se $\exists i^* \in I^* : \delta^*(x, ii^*) \notin F, \delta^*(y, ii^*) \in F$.
Ora supponiamo che valga la (1). E' chiaro che $\forall i^* \in I^*, \forall i \in I$ si ha che

$$\delta^*(x, ii^*) \sim \delta^*(y, ii^*)$$

e quindi $x \sim y$. □

Osservazione 2.3. La relazione "$\sim$" con la formulazione dell'Osservazione 2.2 è una bisimulazione sull'insieme degli stati di un'automa, se si considera la relazione binaria $E \coloneqq \bigcup_{i \in I, x \in S} \{(x, \delta(x,i))\}$.

Dimostrazione. Supponiamo che $x \sim y$. Sia xEx', cioè $\exists i \in I : x' = \delta(x,i)$. Sia $y' = \delta(y,i) \implies yEy'$. Allora, per l'Osservazione 2.2, $x' \sim y'$.
Chiaramente lo stesso argomento vale in modo speculare. □

Osserviamo che nella definizione di "E" si perde un'informazione importante, cioè il fatto che per $i \in I$ fissato $\delta_i(x) \coloneqq \delta(x,i)$ è una funzione, ovvero l'insieme immagine di ogni $x \in S$ ha cardinalità 1. L'algoritmo di Hopcroft sfrutta questa proprietà nel procedimento che consente di migliorare l'algoritmo banale che verrà discusso nel seguito del lavoro.
Possiamo definire la *minimizzazione per stati equivalenti* di un'automa a stati finiti:

Definizione 2.3. Sia $A = (S, I, \delta, F)$ un'automa. Chiameremo *minimizzazione per stati equivalenti* l'automa $A = (\widetilde{S}, I, \widetilde{\delta}, \widetilde{F})$ dove

- $\widetilde{S}$ = "l'insieme delle classi di equivalenza di $\sim$ su S";
- $\widetilde{\delta} : (\widetilde{S} \times I) E \widetilde{S}, \quad \delta(i,x) = y \implies \widetilde{\delta}(i, \widetilde{x}) = \widetilde{y}$, dove $x, y \in S, i \in I$, e $\widetilde{x}, \widetilde{y} \in \widetilde{S}$ sono rispettivamente le classi di equivalenza di "$\sim$" a cui appartengono x, y;
- $\widetilde{F}$ = "l'insieme delle classi di equivalenza di $\sim$ su F";

Prima di concludere l'esposizione delle nozioni preliminari è necessario dimostrare un risultato interessante che lega il problema della minimizzazzione di un'automa a stati finiti con quanto è stato presentato nella sezione 1.5. A questo scopo dimostriamo i seguenti lemmi, che consentono di dimostrare tale legame in modo agevole:

Lemma 2.1. *Sia $A = (S, I, F, \delta)$ un'automa. Sia $\widehat{S}$ una partizione di S stabile rispetto alle funzioni $\delta_i, \forall i \in I$. Allora $\forall (x,y) \in S \times S$ tali che $[x]_{\widehat{S}} = [y]_{\widehat{S}}$ si ha che $\forall i^* \in I^*$*

$$[\delta^*(x, i^*)]_{\widehat{S}} = [\delta^*(y, i^*)]_{\widehat{S}}$$

Dimostrazione. Procediamo per induzione su i^*. Inizialmente i due stati si trovano nello stesso blocco. Ora supponiamo che dopo l'inserimento dell' $(n-1)$-esimo simbolo di i^* i due stati x_{n-1}, y_{n-1} in cui si trova l'automa

appartengano ancora allo stesso blocco. La partizione è stabile rispetto alla funzione δ_i, dove i è l' n-esimo simbolo di i^*, quindi tutto il blocco $[x_{n-1}]_{\widehat{S}} = [y_{n-1}]_{\widehat{S}}$ è contenuto all'interno dell'insieme $\delta_i^{-1}([\delta_i(x_{n-1})]_{\widehat{S}}) = \delta_i^{-1}([x_n]_{\widehat{S}})$. Quindi $\delta_i(y_{n-1}) \in \delta_i^{-1}([x_n]_{\widehat{S}})$, e dunque si ha anche $[x_n]_{\widehat{S}} = [y_n]_{\widehat{S}}$. □

Lemma 2.2. *Sia $A = (S, I, F, \delta)$ un'automa. Sia $\widehat{S}$ una partizione di S stabile rispetto alle funzioni $\delta_i, \forall i \in I$. Supponiamo inoltre che per $\widehat{S}$ valga la segente condizione:*

$$\forall (x, f) \in S \times F, \quad [x]_{\widetilde{S}} = [f]_{\widetilde{S}} \implies x \in F$$

Allora $\forall X \in \widehat{S},\ \forall (x, y) \in X \times X$ si ha $x \sim y$.

Dimostrazione. Supponiamo per assurdo che in un blocco $X \in \widehat{S}$ esistano due stati x, y non equivalenti. Allora deve esistere una stringa $i^* \in I^*$ tale che, ad esempio, $\delta^*(x, i^*) \in F \wedge \delta^*(y, i^*) \notin F$. Ma per il Lemma 2.1 $[\delta^*(x, i^*)]_{\widehat{S}} = [\delta^*(y, i^*)]_{\widehat{S}}$. Chiaramente questo è assurdo per la condizione supposta nell'ipotesi, quindi non possono esistere nello stesso blocco due stati non equivalenti. □

Possiamo dimostrare il seguente risultato:

Teorema 2.1. *Sia A un'automa a stati finiti. Supponiamo che in tale automa ad uno stato finale ed uno non finale non possa essere assegnato un output identico. La minimizzazione per stati equivalenti proposta nella Definizione 2.3 è l'automa avente per stati i blocchi della partizione più grossolana stabile rispetto alle funzioni $\delta_i : SES, \forall i \in I$.*

Dimostrazione. Dimostriamo che la minimizzazione proposta nella Definizione 2.3 è una partizione stabile rispetto alle funzioni δ_i. Supponiamo per assurdo che $\exists i \in I : \widetilde{S}$ non è stabile rispetto a δ_i. Quindi $\exists S_1, S_2 \in S$:

$$S_1 \not\subseteq \delta_i^{-1}(S_2) \ \wedge \ S_1 \cap \delta_i^{-1}(S_2) \neq \emptyset$$

La prima porzione dell'espressione implica che $\exists s \in S_1 : \delta_i(s) \notin S_2$. Ma questo è chiaramente in contrasto con l'Osservazione 2.2, perchè $\forall (u, v) \in S_1 \times S_1$ deve valere $\delta_i(u) \sim \delta_i(v)$, mentre è evidente che, poichè $S_1 \cap \delta_i^{-1}(S_2) \neq \emptyset$, c'è almeno una coppia che non soddisfa questa condizione.
Ora supponiamo che la partizione $\widetilde{S}$ non sia la più grossolana stabile rispetto alle funzioni $\delta_i : SES, \forall i \in I$. Ne deve esistere, quindi, una (stabile) più

grossolana, che chiamiamo $\widetilde{S}$. Chiaramente si ha $|\widehat{S}| < |\widetilde{S}|$, e quindi devono esistere almeno due blocchi $S_1, S_2 \in \widetilde{S}$ ed un blocco $X \in \widehat{S}$ tali che

$$S_1 \cap X \neq \emptyset \ \wedge \ S_2 \cap X \neq \emptyset$$

Ma questo è chiaramente in contrasto con il Lemma 2.2 per come è stata costruita la partizione $\widetilde{S}$, in quanto esistono coppie di stati $(x, y) \in X \times X$ per cui vale $x \in S_1, y \in S_2$, cioè $x \not\sim y$. □

Osserviamo che la condizione richiesta nell'ipotesi del Teorema 2.1, che impone l'assegnazione di un output diverso a stati finali e non finali, è estremamente leggera: dato un'automa qualsiasi è sempre possibile costruirne un altro, che soddisfa tale condizione, in tempo lineare rispetto al numero di stati.

2.1.2 L'algoritmo naive

Innanzitutto, con i risultati della sezione precedente, possiamo progettare il seguente algoritmo banale:

Algoritmo 1: Procedimento banale per la minimizzazzione

```
   Data: S, I, δ
   // Partizione iniziale contenente un unico blocco
1  S̃ := {{s_1, ..., s_n}};
   /* Separiamo gli stati non equivalenti in base
   all'output                                          */
2  S̃ = PartizionaPerOutput(S̃);
3  forall i ∈ I do
4  |  for NS = BlocchiNonStabili(S,δ_i)), NS ≠ ∅ do
   |  |  /* Estraiamo casualmente una coppia di blocchi non
   |  |  stabili                                        */
5  |  |  (X, Y) = NS[0];
6  |  |  X_1 := X − δ_i^{-1}(Y);
7  |  |  X_2 := X ∩ δ_i^{-1}(Y);
   |  |  // Aggiorniamo S
8  |  |  S̃ = (S̃ − {X}) ∪ {X_1, X_2};
9  |  end
10 end
11 return S
```

L'algoritmo termina, perchè la condizione del ciclo diventa sicuramente falsa quando la partizione S è composta da n blocchi, uno per ogni stato, ed

ad ogni iterazione un blocco viene diviso in due blocchi distinti e non vuoti. Inoltre la risposta fornita è corretta, perchè l'algoritmo si ferma appena viene trovata una partizione stabile. Poichè ad ogni iterazione del ciclo definito nella Riga 4 il numero di blocchi aumenta di 1, il risultato è la partizione stabile più grossolana rispetto alle funzioni δ_i. Osserviamo inoltre che, se la partizione è stabile rispetto a δ_i dopo una certa iterazione del ciclo definito nella Riga 3, allora resta stabile fino alla fine dell'esecuzione.
Consideriamo la complessità dell'algoritmo:

- La Riga 2 ha complessità $\Theta(|S|)$;
- Il ciclo della Riga 3 viene eseguito $\Theta(|I|)$ volte;
- La funzione "BlocchiNonStabili" ha complessità $O(|S|^2)$;
- Il ciclo nella Riga 4 viene eseguito $O(|S|)$ volte;
- Il contenuto del ciclo nella Riga 4 ha complessità $O(|S|)$ (con gli opportuni accorgimenti).

Quindi la complessità dell'algoritmo è

$$T_{alg_1}(|S|,|I|) = \Theta|I|\left[O(|S|^2 + O(|S|) * O(|S|))\right] = \Theta(I)O(|S|^2)$$

Se consideriamo costante il numero di ingressi: $T_{alg_1}(|S|) = O(|S|^2)$.

2.1.3 L'algoritmo di Hopcroft

Riportiamo l'algoritmo di Hopcroft, presentato in [5] nel 1971. Esso migliora la procedura presentata nella sezione precedente, in quanto ha complessità *loglineare*. Forniremo un commento allo pseudocodice, in modo da spiegare il procedimento in modo dettagliato. In seguito analizzeremo formalmente l'algoritmo, proporremo e commenteremo la dimostrazione della correttezza e della complessità.

Algoritmo 2: Algoritmo di Hopcroft

Data: S, I, δ, F

begin
  forall $(s,i) \in S \times I$ **do**
    $\delta^{-1}(s,i) \coloneqq \{t : \delta(t,i) = s\}$;
  $B(1) \coloneqq F, B(2) \coloneqq S - F$;
  /* $\forall i \in I$ costruiamo l'insieme degli stati in $B(j)$ aventi controimmagine non vuota rispetto a δ_i */
  forall $j \in \{1,2\}$ **do**
    forall $i \in I$ **do**
      $i(j) = \{s : s \in B(j) \wedge \delta^{-1}(s,i) \neq \emptyset\}$;
  /* k è il numero di blocchi della partizione. Aumenta dopo le rifiniture */
  $k \coloneqq 2$;
  /* Per ogni ingresso i creiamo un insieme $L(i)$ contenente l'indice j che minimizza $|i(j)|$ */
  forall $i \in I$ **do**
    if $|i(1)| \leq |i(2)|$ **then**
      $L(i) = \{1\}$;
    else
      $L(i) = \{2\}$;
  while $\exists i \in I : L(i) \neq \emptyset$ **do**
    Seleziona un $j \in L(i)$. $L(i) = L(i) - \{j\}$;
    /* Per ogni blocco $B(m)$ per cui il procedimento ha senso, usiamo $i(j)$ come *splitter* */
    forall $m \leq k : \exists t \in B(m) : \delta(t,i) \in i(j)$ **do**
      $B^{'}(m) \coloneqq \{u \in B(m) : \delta(u,i) \in i(j)\}$;
      $B^{''}(m) \coloneqq B(m) - B^{'}(m)$;
      $B(m) = B^{'}(m)$, $B(k+1) \coloneqq B^{''}(m)$;
      forall $a \in I$ **do**
        $a(m) = \{s : s \in B(m) \wedge \delta^{-1}(s,a) \neq \emptyset\}$;
        $a(k+1) \coloneqq \{s : s \in B(k+1) \wedge \delta^{-1}(s,a) \neq \emptyset\}$;
        if $|a(m)| \leq |a(k+1)|$ **then**
          $L(a) = L(a) \cup \{m\}$;
        else
          $L(a) = L(a) \cup \{k+1\}$;
      $k = k + 1$;

Nella Riga 3 definiamo δ^{-1}, che consente l'accesso in tempo costante all'insieme degli stati che conducono ad un determinato stato conseguentemente ad un determinato ingresso.
Inizialmente la partizione contiene due blocchi: gli stati finali e quelli non finali. Di conseguenza le rifiniture successive della partizione manterranno sempre separati stati finali e non finali.
Per evitare accessi inutili (che incrementerebbero i termini costanti dell'algoritmo) nella Riga 7 definiamo $i(j)$ per $i \in I, j \in \{1,2\}$ come l'insieme degli stati s nel blocco $B(j)$ per cui esiste qualche stato t tale che $\delta(t,i) = s$.
All'interno del ciclo della Riga 9 definiamo gli insiemi $L(i)$ $\forall i \in I$, che contengono gli indici dei blocchi che verranno usati come *splitter* nel seguito del procedimento. Osserviamo che all'interno di questo insieme vengono inseriti gli indici dei blocchi per cui la cardinalità di "$i(\cdot)$" è minima. Come dimosteremo nel seguito, questa tecnica

In [7] si osserva che, limitatamente al caso particolare a cui si applica l'algoritmo di Hopcroft, è possibile scegliere gli *splitter* in modo vantaggioso, allo scopo di ridurre il carico di lavoro e dunque la complessità. Riportiamo e commentiamo la dimostrazione:

Osservazione 2.4. Sia S un insieme finito. Sia $f : SES$ una funzione (cioè $\forall s \in S \; |f(s)| = 1$). Sia $\widetilde{S}$ una partizione di S. Sia Q l'unione di alcuni blocchi di $\widetilde{S}$, e sia B un blocco di $\widetilde{S}$, con $B \subseteq Q$. Supponiamo $\widetilde{S}$ stabile rispetto a Q. Allora

$$\text{Split}(B, \widetilde{S}) = \text{Split}(Q - B, \text{Split}(B, \widetilde{S}))$$

Dimostrazione. Vogliamo dimostrare che $Q-B$ non è uno *splitter* di $\text{Split}(B, \widetilde{S})$, cioè che $\forall S_x \in \text{Split}(B, \widetilde{S})$ si ha

$$S_x \subseteq f^{-1}(Q - B) \;\vee\; S_s \cap f^{-1}(Q - B) = \emptyset$$

Ricordando che $\widetilde{S}$ è stabile rispetto a Q, $\widehat{S} \coloneqq \text{Split}(B, \widetilde{S})$ è stabile rispetto a Q e a B, per l'Osservazione 1.7. Di conseguenza per ogni blocco $S_x \in \widehat{S}$ si ha

$$S_x \subseteq f^{-1}(B) \vee S_x \cap f^{-1}(B) = \emptyset$$
$$\wedge$$
$$S_x \subseteq f^{-1}(Q) \vee S_x \cap f^{-1}(Q) = \emptyset$$

Se $S_x \subseteq f^{-1}(B)$ chiaramente $S_x \cap \subseteq f^{-1}(Q - B) = \emptyset$. Se $S_x \cap f^{-1}(Q) = \emptyset$, allora $S_x \cap f^{-1}(Q - B) = \emptyset$. Se $S_x \cap f^{-1}(B) = \emptyset \wedge S_x \subseteq f^{-1}(Q)$ bisogna avere $S_x \subseteq f^{-1}(Q - B)$.
Osserviamo che queste deduzioni valgono solamente se f è una funzione. $\square$

Dal punto di vista computazionale è chiaramente più conveniente usare come *splitter* l'insieme tra B e $Q - B$ avente cardinalità minore. La strategia di Hopcroft ("*process the smaller half*", come viene sintetizzata in [7]) consiste infatti nella selezione degli *splitter* secondo il criterio della cardinalità, a differenza di quanto avviene nell'Algoritmo 1.
Dalla Riga 17 alla Riga 20 viene operato lo "Split". Nel ciclo della Riga 20 vengono aggiornate le strutture dati relative ai nuovi blocchi creati. Osserviamo che ad ogni iterazione del ciclo della Riga 20 vengono creati due nuovi blocchi, anche se il numero di blocchi aumenta soltanto di 1, perchè viene anche "smembrato" un blocco già esistente. Di questi due blocchi se ne sceglie uno da usare come *splitter*, seguendo il criterio illustrato sopra.
Queste operazioni vengono ripetute finchè in $L(i)$ per qualche i resta un blocco da usare come *splitter*.

Analizziamo la correttezza dell'algoritmo. Trascurando la parte iniziale, ad ogni iterazione del ciclo della Riga 14 viene prima rimosso un elemento in $L(i)$, e solamente in seguito ad una rifinitura della partizione ne viene inserito un altro. Poichè non è possibile rifinire all'infinito, l'algoritmo termina.
Il seguente Teorema dimostra la validità del risultato:

Teorema 2.2. *Sia $A = (S, I, \delta, F)$ un'automa. Sia $\widetilde{S}$ la partizione risultante dall'applicazione dell'Algoritmo 3. Siano $x, y \in S$. Allora*

$$x \sim y \iff [x]_{\widetilde{S}} = [y]_{\widetilde{S}}$$

Dimostrazione. Dimostriamo innanzitutto che $[x]_{\widetilde{S}} \neq [y]_{\widetilde{S}} \implies x \not\sim y$. Procediamo per induzione:

- La relazione vale prima del ciclo 14, infatti stati finali e non finali non possono essere equivalenti;
- Supponiamo che sia vero prima di una certa iterazione. Supponiamo che gli stati x, y finiscano in partizioni diverse nelle Righe tra 17 e 20. Questo significa che $[\delta(x,i)]_{\widetilde{S}_n} \neq [\delta(y,i)]_{\widetilde{S}_n}$, dove $\widetilde{S}_n$ è la partizione costruita dall'algoritmo fino all'iterazione considerata. Ma allora, per l'ipotesi induttiva, $\delta(x,i) \not\sim \delta(y,i)$, e quindi x, y non sono equivalenti (per l'Osservazione 2.2).

Questo ragionamento è valido perchè il fatto che due stati si trovino in partizioni differenti dopo una certa iterazione implica che si troveranno in partizioni differenti anche al termine del procedimento.

Dimostriamo ora che $[x]_{\widetilde{S}} = [y]_{\widetilde{S}} \implies x \sim y$. Supponiamo per assurdo che per due stati x, y si abbia $[x]_{\widetilde{S}} = [y]_{\widetilde{S}} \wedge x \not\sim y$. Allora, ad esempio, $\exists i^* \in I^* : \delta^*(x, i^*) \in F, \delta^*(y, i^*) \notin F$. Ma poichè inizialmente poniamo stati inziali e finali in partizioni diverse, deve valere chiaramente $[\delta^*(x, i^*)]_{\widetilde{S}} \neq [\delta^*(y, i^*)]_{\widetilde{S}}$, e quindi procedendo a ritroso sui simboli di i^* si ha

$$[\delta^*(x, i_n^*)]_{\widetilde{S}} \neq [\delta^*(y, i^*)_n]_{\widetilde{S}} \implies [\delta^*(x, i_{n-1}^*)]_{\widetilde{S}} \neq [\delta^*(y, i^*)_{n-1}]_{\widetilde{S}}$$

Per cui si ha chiaramente $[x]_{\widetilde{S}} = [\delta^*(x, i_0^*)]_{\widetilde{S}} \neq [\delta^*(y, i^*)_0]_{\widetilde{S}} = [y]_{\widetilde{S}}$. □

Discutiamo ora la complessità dell'algoritmo. In questo paragrafo non tratteremo i dettagli dell'implementazione, facilmente reperibili in [5], e ci concentreremo unicamente sul contributo del procedimento.
Per costruire δ^{-1} è sufficiente valutare una sola volta ogni stato per ogni ingresso, quindi l'operazione è $\Theta(|S||I|)$. La costruzione delle due partizioni iniziali è $\Theta(|S|)$. La costruzione di $L(i)\ \forall i \in I$ è chiaramente $\Theta(|I|)$. Di conseguenza la complessità delle istruzioni precedenti alla Riga 14 è $\Theta(|S||I|)$. Prima di procedere, consideriamo la seguente Osservazione:

Osservazione 2.5. Sia

$$f_a(x) \coloneqq a\log_2(a) - (a-x)\log_2(a-x) - x\log_2(x) \qquad a > 0,\ 0 < x < a$$

Tale funzione ha massimo in $\frac{a}{2}$, con $f_a(\frac{a}{2}) = a$, ed è strettamente positiva sul dominio considerato.

Dimostrazione. L'osservazione si dimostra con un semplice studio di funzione. □

Consideriamo ora un ingresso $i \in I$ fissato per una determinata iterazione n del ciclo della Riga 14 per cui si abbia la seguente configurazione:

$$\widetilde{S}_n = \{S_1, \ldots, S_m\}, \quad L(i) = \{i_1, \ldots, i_r\}, \quad \{i_{r+1}, \ldots, i_m\} = \{1, \ldots, m\} - L(i)$$

Osserviamo che, se all'inizio dell'iterazione viene selezionato l'indice $x \in L(i)$, la complessità dell'iterazione è $O(|i(x)|)$, cioè il tempo impiegato durante l'esecuzione è maggiorato da $k|i(x)|$, dove k è una costante di propozionalità. Questo avviene perchè il ciclo della Riga 16 è in realtà un ciclo che itera sugli stati $s \in i(x)$, e per ognuno estrae (in tempo costante) $\delta^{-1}(s, i)$. Vogliamo dimostrare che il contributo al tempo di esecuzione di tutte le iterazioni del ciclo in cui si seleziona l'ingresso i considerato sopra, dalla

n-esima fino alla terminazione dell'algoritmo, è maggiorato dalla seguente espressione:

$$T = k \left(\sum_{j=1}^{r} |i(i_j)| \log_2 |i(i_j)| + \sum_{j=r+1}^{m} |i(i_j)| \log_2 \frac{|i(i_j)|}{2} \right)$$

Supponiamo che prima dell'iterazione k-esima (con $k > n$) la maggiorazione sia valida, e dimostriamo che resta valida al termine dell'iterazione. In altre parole è necessario dimostrare che la somma tra T' ed il tempo impiegato per l'esecuzione dell'iterazione, dove T' è un'espressione analoga a T, è minore o uguale di T.
All'inizio di ogni iterazione viene estratto un ingresso a. Si presentano due casi:

- $a \neq i$: poichè T prende in considerazione solamente il tempo impiegato dalle iterazioni in cui viene selezionato i, questa iterazione è esclusa dalla stima. Ciononostante l'iterazione può modificare i blocchi della partizione, e dobbiamo quindi verificare che $T' \leq T$:
  - Se viene modificato un blocco $B(x)$ con $x \in L(i)$, nella stima dobbiamo sostituire un elemento maggiorabile con $b \log_2 b$ con un elemento del tipo $c \log_2 c + (b-c) \log_2 (b-c)$. Per l'osservazione 2.5 si ha $T' < T$;
  - Se invece $x \notin L(i)$, in T' dobbiamo sostituire un elemento del tipo $b \log_2 \frac{b}{2}$ con un elemento del tipo $c \log_2 c + (b-c) \log_2 \frac{b-c}{2}$ (supponendo che $c \leq b - c$, in caso contrario la dimostrazione è simile). Infatti il blocco avente cardinalità $b - c$ fa parte, alla fine dell'iterazione, dell'insieme dei blocchi il cui indice non appartiene ad $L(i)$, ed al termine del ciclo della Riga 20 si ha $c \in L(i)$. Chiaramente si ha $c \leq \frac{b}{2}$, e quindi

$$\begin{aligned} c \log_2 c + (b-c) \log_2 \frac{b-c}{2} &\leq c \log_2 \frac{b}{2} + (b-c) \log_2 \frac{b}{2} \\ &= (c + b - c) \log_2 \frac{b}{2} \\ &= b \log_2 \frac{b}{2}. \end{aligned}$$

- $a = i$: come abbiamo osservato sopra, il tempo impiegato all'interno

dell'iterazione è $O(|i(x)|)$, dove x è l'indice estratto da $L(i)$. Allora

$$T' = k \left(|i(x)| + |i(x)| \log_2 \frac{|i(x)|}{2} + \sum_{\substack{j=1 \\ i_j \neq x}}^{r} |i(i_j)| \log_2 |i(i_j)| + \sum_{j=r+1}^{m} |i(i_j)| \log_2 \frac{|i(i_j)|}{2} \right)$$

dove consideriamo il tempo impiegato per l'iterazione, ed il fatto che l'indice x, al termine dell'iterazione, fa parte dell'insieme dei blocchi il cui indice non appartiene ad $L(i)$. Dimostriamo che $T' \leq T$:

$$\begin{aligned} |i(x)| + |i(x)| \log_2 \frac{|i(x)|}{2} &= |i(x)| + |i(x)| \log_2 |i(x)| - |i(x)| \log_2 2 \\ &= |i(x)| + |i(x)| \log_2 |i(x)| - |i(x)| \\ &= |i(x)| \log_2 |i(x)| \end{aligned}$$

e quindi $T' = T$.

Prima della prima iterazione del ciclo della Riga 14, per un $i \in I$ fissato, (supponendo $|S - F| > |F|$) si ha

$$T_i = k \left(|S - F| \log_2 |S - F| + |F| \log_2 \frac{|F|}{2} \right)$$

che, per l'Osservazione 2.5, si maggiora con $k|S| \log_2 |S|$. Allora la complessità dell'algoritmo è data dalla somma della complessità delle righe precedenti alla 14, ovvero $\Theta(|I||S|)$, e $|I| * O(|S| \log |S|)$, cioè $|I||S| \log |S|$. Considerando costante la cardinalità di I:

$$T_{alg} = O(|S| \log |S|).$$

2.2 L'algoritmo di Paige-Tarjan (PTA)

L'algoritmo che presentiamo in questa sezione (proposto da Paige e Tarjan in [7]) risolve il problema generale presentato all'inizio di questo lavoro, e generalizza il procedimento di Hopcroft sfruttandone l'intuizione.
Descriveremo innanzitutto un algoritmo *naive* avente complessità $O(mn)$, dove m è il numero di coppie messe in relazione dalla relazione binaria R (cioè $|R|$) ed n è la cardinalità dell'insieme A da partizionare. Sulla base di questo applicheremo i miglioramenti sviluppati dall'idea di Hopcroft, che consentiranno di ridurre la complessità ad $O(m \log n)$. Come osserveremo nel seguito, l'idea generale consiste nello scegliere in modo oculato gli *splitter* con cui rifinire la partizione.

2.2.1 L'algoritmo naive

Presentiamo lo pseudocodice per l'algoritmo *naive*:

Algoritmo 3: Algoritmo di Hopcroft

Data: A, R, S

begin

  $\widetilde{S} \coloneqq S$;

  while $\widetilde{S}$ non è stabile rispetto a R **do**

    $Q \coloneqq \bigcup_{j=1}^{n} \widetilde{S}_{i_j}, \quad$ con $\widetilde{S}_{i_j} \in \widetilde{S}$ per $j \in \{1, \ldots, n\}$;

    $\widetilde{S} = \text{Split}(Q, \widetilde{S})$;

  return $\widetilde{S}$;

Bibliografia

[1] Peter Aczel. *Non-well-founded sets*, volume 14 of *CSLI lecture notes series*. CSLI, 1988.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[3] Agostino Dovier, Carla Piazza, and Alberto Policriti. A fast bisimulation algorithm. In *International Conference on Computer Aided Verification*, pages 79–90. Springer, 2001.

[4] Raffaella Gentilini, Carla Piazza, and Alberto Policriti. From bisimulation to simulation: Coarsest partition problems. *Journal of Automated Reasoning*, 31(1):73–103, 2003.

[5] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

[6] Kenneth Kunen. *Set theory an introduction to independence proofs*. Elsevier, 2014.

[7] Robert Paige and Robert E Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[8] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.