1 Algoritmi e automi

Vi sono vari modi di formalizzare la nozione di algoritmo. Tale nozione non e' assoluta, bensi' relativa ad un *modello di calcolo*. Una formalizzazione che cattura alcune (ma non tutte) le proprieta' che dovrebbe avere un algoritmo deterministico e' data dalla nozione di automa deterministico. Per tener conto della complessita' associamo ad ogni transizione dell' automa con costo a valori in un monoide ordinato.

Definition 1.1 (Monoide ordinato) *Un* **monoide ordinato** *e' un'algebra con predicati* $(M, \leq, +, 0)$

- *dove* M *e' un insieme non vuoto,* $\leq$ *e' un predicato binario,* $+$ *e' una operazione binaria, e* 0 *e' una costante;*
- $(M, \leq)$ *e' un ordine parziale, cioe'* $\leq$ *soddisfa la proprieta' riflessiva, transitiva ed anti-simmetrica:*
- $(M, +, 0)$ *e' un monoide, cioe' l'operazione* $+$ *e' associativa* $x + (y + z) = (x + y) + z$, *e* $x + 0 = x = 0 + x$;
- *ed inoltre* 0 *e' il minimo* $0 \leq x$, *e* $+$ *e' monotona* $x \leq x' \land y \leq y' \supset x + y \leq x' + y'$.

Definition 1.2 (Automa) *Fissati due insiemi* I *e* O, *ed un* monoide ordinato $(M, \leq, +, 0)$, *un automa deterministico* A *con input* I *ed output* O *e pesi in* M *(o piu' semplicemente un automa) e' dato da:*

- *un insieme* S_A *di stati interni*
- *una funzione di transizione pesata* $t_A : I + S_A \to (S_A + O) \times M$.

Introduciamo anche le seguenti nozioni derivate:

- $\tilde{S}_A \stackrel{\Delta}{=} I + S_A + O$ *e' l'insieme degli stati.*
- $\xrightarrow[A]{m} \subseteq \tilde{S}_A \times \tilde{S}_A$ *e' la relazione di transizione (pesata), cioe'* $s \xrightarrow[A]{m} s' \stackrel{\Delta}{\iff} t_A(s) = (s', m)$.
- $(s_i | 0 \leq i \leq n) \in \tilde{S}_A^+$ *e' una computazione (finita) di* A *da* s_0 *a* s_n *con peso* $m \stackrel{\Delta}{\iff}$ *esiste, necessariamente unica,* $(m_i | 0 \leq i < n) \in M^n$ *t.c.* $\forall i < n. s_i \xrightarrow[A]{m_i} s_{i+1}$ *e* $m = \sum_{0 \leq i < n} m_i$.
- $\xrightarrow[A]{m}\!\!\!\!\!\rightarrow \subseteq \tilde{S}_A \times \tilde{S}_A$ *e' la* chiusura transitiva *di* $\xrightarrow{m}$ *cioe'* $s \xrightarrow[A]{m}\!\!\!\!\!\rightarrow s' \stackrel{\Delta}{\iff}$ *esiste una computazione non banale (cioe'* $n > 0$*) di* A *da* s *a* s' *con peso* m.
- $f_A : I \rightharpoonup O$ *e' la funzione (parziale) calcolata da* A, *cioe'* $f_A(x) = y \stackrel{\Delta}{\iff}$ *esiste* $m \in M$ *t.c.* $x \xrightarrow[A]{m}\!\!\!\!\!\rightarrow y$
- $T_A : I \rightharpoonup M$ *e' la funzione complessita' esatta di* A, *cioe'* $T_A(x) = m \stackrel{\Delta}{\iff}$ *esiste* $y \in O$ *t.c.* $x \xrightarrow[A]{m}\!\!\!\!\!\rightarrow y$

Remark 1.3 Vi sono vari modi di scegliere il monoide ordinato M, che riflettono quale uso di *risorsa* vogliamo analizzare. Se siamo interessati alla complessita' tempo useremo il monoide $(N, \leq, +, 0)$ dei numeri naturali con l'addizione, o il monoide $(R^+, \leq, +, 0)$ dei reali non negativi, o il monoide $(\overline{N}, \leq, +, 0)$ dei numeri naturali estesi con ∞. La scelta dell'addizione come operazione del monoide corrisponde al fatto che il tempo per effettuare due transizioni, una dopo l'altra', si ottiene sommando i tempi. Se siamo interessati alla complessita' spazio (di memoria) useremo il monoide $(N, \leq, \max, 0)$ dei numeri naturali con l' operazione di massimo (monoidi analoghi si possono definire su R^+ e $\overline{N}$). La scelta del massimo come operazione del monoide corrisponde al fatto che lo spazio necessario per effettuare due transizioni, una dopo l'altra', si ottiene prendendo il massimo tra gli spazi richiesti, poiche' la memoria necessaria per la prima transizione puo' essere riutilizzata dalla seconda. Se siamo interessati sia alla complessita' spazio che tempo, basta prendere il prodotto dei monoidi ordinati $(N, \leq, +, 0)$ e $(N, \leq, \max, 0)$, cioe' $(N \times N, \leq, \ldots)$. Se non siamo interessati alla complessita', basta prendere il monoide banale, che ha solo l'elemento 0.

Se un algoritmo lo rappresentiamo mediante un automa con input I ed output O, e' naturale rappresentare un problema mediante una relazione $P \subseteq I \times O$ che ad ogni input (istanza del problema) associa un insieme di output (soluzioni) accettabili. Useremo indistintamente la notazione $x\ P\ y$ e $P(x, y)$ per indicare che $(x, y) \in P$. Il primo requisito che deve soddisfare un algoritmo e' la correttezza.

Definition 1.4 (Correttezza) *Dato un automa* A *ed un problema* P *diciamo che*

- A **risolve parzialmente** $P \stackrel{\Delta}{\iff} \forall x \in I.\ f_A(x)$ *definito implica* $x\ P\ f_A(x)$;

- A **risolve totalmente** $P \stackrel{\Delta}{\iff} \forall x \in I.\ f_A(x)$ *definito e* $x\ P\ f_A(x)$.

In generale, se A risolve totalmente P, allora lo risolve parzialmente. Tuttavia, quando la funzione f_A e' totale, vale anche il viceversa. La definizione di correttezza non e' molto sensata quando vi sono $x \in I$ per i quali non esistono $y \in O$ t.c. $P(x,y)$, tuttavia ci possiamo sembre ridurre al caso in cui I e' il dominio della relazione P.

Il requisito di correttezza ignora completamento la complessita' T_A di un algoritmo. Tuttavia, disponendo di due algoritmi corretti, cioe' che risolvono un dato problema P, e' naturale preferire l'algoritmo con minore complessita' (per questo e' importante che su M vi sia una relazione d'ordine $\leq$). In certi casi (p.e. applicazioni real-time) e' importante specificare gia' a livello di problema quanto si e' disposti a tollerare in termini di costi, fissando un limite superiore $T: I \to M$. Molto spesso all'insieme I (ed analogamente per O) si associa una funzione $|\_|: I \to N$, che fornisce la **dimensione** $|x|$ di una *istanza* x del problema. In questo caso e' piu' semplice specificare un limite superiore $T: N \to M$ in funzione della dimensione dell'input.

Definition 1.5 *Dato un automa A, e due funzioni $|\_|: I \to N$ e $T: N \to M$ (che assumiamo monotona), diciamo che T e' una* **maggiorazione** *della complessita' di A (nel caso peggiore)* $\stackrel{\Delta}{\iff} \forall x \in I.\ T_A(x) \leq T(|x|)$.

Si noti che abbiamo richiesto che T sia una funzione totale, e quindi anche T_A deve essere totale se maggiorata da T.

Un algoritmo lo si puo' descrivere a diversi livelli di astrazione. In particolare, quello che e' considerato un passo di computazione ad un *livello di astrazione*, corrispondera' ad uno o piu' passi di computazione ad un maggior livello di dettaglio. La nozione di simulazione tra automi formalizza la corrispondenza tra un algoritmo ed un suo *raffinamento*.

Definition 1.6 (Simulazione tra automi) *Dati due automi A e B con input I ed output O e pesi in M, una simulazione di A tramite B e' una relazione $R \subseteq S_B \times S_A$ t.c.*

- $t\ \tilde{R}\ s \xrightarrow[A]{n} s'$ *implica* $\exists t' \in \tilde{S}_B, m \in M$ *t.c.* $t \xrightarrow[B]{m} t'\ \tilde{R}\ s'$ *e* $m \leq n$

dove $\tilde{R}$ e' la relazione relazione $\Delta_I + R + \Delta_O \subseteq \tilde{S}_A \times \tilde{S}_B$.

In genere si suppone che R sia il grafo di una funzione parziale surgettiva da S_B ad S_A, cioe' per ogni t esiste al piu' un s t.c. $t\ R\ s$, e per ogni s esiste almeno un t t.c. $t\ R\ s$. In termini piu' intuitivi, avere una simulazione di A tramite B vuol dire che ogni transizione $s \xrightarrow[A]{n} s'$ di A puo' essere *mimata* da una transizione composita di B di costo non superiore ad n.

Proposition 1.7 *Se esiste una simulazione di A tramite B, allora i due algoritmi calcolano la stessa funzione parziale, cioe' $f_A(x) = f_B(x)$, e la complessita' di B e' maggiorata da quella di A, cioe' $T_B(x) \leq T_A(x)$.*

Quindi se abbiamo dimostrato che un algoritmo A ad alto livello (o che fa' uso di pseudo-istruzioni) e' *corretto*, cioe' risolve il problema P con un costo maggiorato da $T: N \to M$, allora ogni *implementazione corretta* di A, cioe' un algoritmo B per cui esiste una simulazione di A tramite B, risolve il problema P.

1.1 Invarianti e terminazione

Dato un automa A ed un problema P, diamo un metodo (detto dell' invariante) per dimostrare la correttezza parziale.

Proposition 1.8 *A risolve parzialmente P se esiste $Inv \subseteq I \times \tilde{S}_A$ t.c.*

- $\forall x \in I. Inv(x,x)$
- $\forall x \in I,\ s,s' \in \tilde{S}_A,\ m \in M. Inv(x,s) \land s \xrightarrow[A]{m} s' \supset Inv(x,s')$
- $\forall x \in I,\ y \in O. Inv(x,y) \supset P(x,y)$

Proof Per definizione di f_A, si deve dimostrare che se esiste una computazione $(s_i|0 \leq i \leq n)$ da $x \in I$ ad $y \in O$, allora vale $P(x,y)$. E' facile dimostrare, per induzione su i e dalla definizione di invariante, che $Inv(x,s_i)$. In particolare, si avra' $Inv(x,y)$ e quindi $P(x,y)$. ∎

Se siamo interessati a dimostrare anche la correttezza totale, e' sufficiente dimostrare che la funzione f_A e' totale.

Proposition 1.9 *$f_A: I \rightharpoonup O$ e' totale se esiste $p: S_A \to N$ t.c. $\forall s,s' \in S_A,\ m \in M. s \xrightarrow[A]{m} s' \supset p(s) > p(s')$.*

Example 1.10 Applicare le idee generali per dimostrare la correttezza dell'algoritmo di sorting mediante selezione. Trattare le sequenze come un tipo di dato astratto. Considerare prima un algoritmo A ad alto livello, in cui l'estrazione del minimo elemento da una sequenza e' considera una operazione elementare. Darne quindi una implementazione B a piu' basso livello, e dimostrandone la correttezza esibendo una simulazione di A tramite B.