Prima esercitazione sul progetto di automi

Luca Tesei

1 Progettare un automa

In molti esercizi viene richiesto di scrivere un automa che accetti un certo linguaggio dato. Il linguaggio può essere stato specificato sia in maniera formale (con un'espressione regolare o con un'espressione su insiemi) oppure in maniera informale tramite una descrizione a parole e con l'ausilio di esempi.

In entrambi i casi, se si vuole affrontare il problema partendo col piede giusto, è bene assicurarsi di aver capito *esattamente* che tipo di linguaggio viene richiesto. Se questo è vero si è sicuramente in grado di scrivere alcune stringhe del linguaggio e, soprattutto, di individuare i "casi particolari": le stringhe più corte, quelle più anomale, quelle che hanno una struttura precisa che si ripete, ecc.

Quando si è sicuri di aver compreso il linguaggio si può passare al progetto dell'automa. In questa fase bisogna cercare di trovare un modo di combinare gli strumenti che ci dà la teoria degli automi per ottenere le stringhe richieste. A ben guardare gli strumenti sono molto semplici: stati, transizioni con cui consentire o impedire (non mettendo la transizione) cammini, stati di accettazione e una forma molto semplice di ricorsione che ci permette di iterare delle transizioni su un certo stato o su un certo ciclo di stati.

La prima cosa da fare è cercare di trovare un algoritmo con cui si vogliono accettare *tutte e sole* le stringhe del linguaggio. Un errore tipico è quello di preoccuparsi di far accettare all'automa tutte le stringhe del linguaggio e non preoccuparsi di *non* fare accettare stringhe che *non* sono nel linguaggio. Un altro errore tipico è quello di costruire l'automa seguendo alcuni esempi di stringhe e non la definizione di tutto il linguaggio: in questo caso l'automa risultante di solito accetta tutte le stringhe di esempio, ma non ne accetta altre che sono comunque nel linguaggio.

Per evitare questi problemi è bene seguire una metodologia che guidi alla soluzione. La prima raccomandazione di questa metodologia è quella di

partire dagli stati per elaborare un algoritmo. Gli stati di un automa possono astrarre qualunque tipo di informazione, a qualunque livello di dettaglio. Si può decidere, quindi, di rappresentare con uno stato una qualsiasi situazione. Dopo questo primo passo, tenendo ben presente i diversi significati che si sono dati ai diversi stati dell'automa, si raccomanda di passare a scrivere le transizioni tra di essi *in maniera coerente con la logica dell'algoritmo che si è pensato*. Il tocco finale è riconoscere gli stati di accettazione.

Facciamo alcuni esempi classici per illustrare questa metodologia. Si consideri il seguente problema: costruire un automa che accetti tutte e sole le stringhe di $\{0,1\}^*$ che hanno un numero pari di occorrenze del simbolo 1.

Per prima cosa riflettiamo sul linguaggio. Zero è da considerarsi numero pari e quindi sicuramente l'automa deve accettare tutte le stringhe che contengono zero occorrenze del simbolo 1, cioè che contengono solo simboli 0. In particolare può accettare anche la stringa vuota ϵ poiché questa contiene sicuramente zero occorrenze del simbolo 1. Delle stringhe che invece contengono qualche simbolo 1 osserviamo subito che non ci importa come sono dislocati questi 1 all'interno della stessa. In particolare, essi potrebbero essere tutti attaccati oppure occorrere qua e là fra diversi simboli 0 messi a piacere. Non è questo che ci interessa: ci interessa la parità del numero delle loro occorrenze.

Avendo, in questo modo, inquadrato per bene il linguaggio possiamo passare a cercare un algoritmo per l'accettazione delle stringhe descritte. Il fatto che le occorrenze del simbolo 0 non influenzano la nostra decisione sull'accettazione o no della stringa ci induce a pensare che sicuramente, in qualsiasi stato si trovi l'automa, possiamo inserire una transizione uscente etichettata con 0. Questo significa che la vera logica dell'automa deve essere basata solo sulle occorrenze dei simboli 1.

Mettiamoci nei panni dell'automa che riceve una stringa in ingresso e deve decidere se accettarla o no. Pensiamo all'algoritmo che potrebbe seguire. Possiamo pensare che alla prima occorrenza di un simbolo 1 l'automa deve ricordare questa informazione in un certo stato poiché questo significa che, per il momento, c'è un numero dispari (1) di occorrenze di simboli 1. Quindi, se la stringa si conclude qui, o al limite dopo alcune occorrenze del simbolo 0, non possiamo accettarla. Comunque la stringa potrebbe non essere terminata e quindi l'automa deve poter continuare a guardare i prossimi simboli in ingresso.

Ad una eventuale occorrenza successiva di un simbolo 1 l'automa dovrà di nuovo ricordare la cosa poiché, se dopo la prima occorrenza il numero di

simboli 1 era dispari, ora il numero di simboli 1 è pari e quindi, allo stato attuale delle cose, la stringa può essere accettata. Dobbiamo fare in modo che questo avvenga se la stringa si conclude qui oppure se si conclude dopo un certo numero di occorrenze di soli simboli 0.

Se la stringa continua, a questo punto, ci ritroviamo nella situazione iniziale: se arriva un simbolo 1 bisogna cambiare stato e ricordare che attualmente la stringa non può essere accettata. E se arriva un 1 successivamente l'automa deve di nuovo cambiare lo stato e ricordare che attualmente la stringa può essere accettata.

A questo punto è chiara la logica dell'algoritmo: si oscilla tra i due stati fino a quando la stringa non termina e a quel punto si guarda lo stato in cui si è per decidere se accettare o no.

Formalizziamo questa idea con gli strumenti che ci danno gli automi. Innanzitutto, alla luce della nostra idea, definiamo due stati p e d. Lo stato p ricorda l'informazione: "fino a questo momento la stringa di ingresso contiene un numero pari di simboli 1" mentre lo stato d ricorda l'informazione: "fino a questo momento la stringa di ingresso contiene un numero dispari di simboli 1".

All'inizio, quando l'automa non ha ancora letto nessun simbolo della stringa in ingresso, vale l'enunciato associato allo stato p poiché zero occorrenze di 1 sono un numero pari. Pertanto p è il candidato perfetto per essere lo stato iniziale dell'automa.

Passiamo a definire le transizioni fra questi stati stando bene attenti a preservare la logica degli stessi, cioè a fare in modo che l'informazione ad essi associata rimanga sempre vera. Esaminiamo la situazione stato per stato.

Consideriamo lo stato p e mettiamoci nei panni dell'automa che legge il primo simbolo della stringa che gli è rimasta da analizzare. Assumiamo per costruzione che l'informazione associata allo stato sia vera e impegnamoci a scrivere le transizioni in modo tale che resti vera. Siccome l'alfabeto Σ è l'insieme $\{0, 1\}$ l'automa *può* prevedere zero, una o più[1] transizioni uscenti dallo stato d etichettate con 0 o 1.

Consideriamo il simbolo 0. Abbiamo detto che l'occorrenza di tale simbolo non cambia niente rispetto alla logica degli stati. In particolare, in questo caso, si vede benissimo che l'informazione associata a p, cioè "fino a questo momento la stringa di ingresso contiene un numero pari di simboli 1", non è influenzata dall'occorrenza attuale di un simbolo 0, che comunque

[1] Ad esempio se l'automa è non deterministico.

può occorrere liberamente. Pertanto ci sarà una transizione uscente da p etichettata con 0 e lo stato di destinazione deve essere per forza p stesso! Diversamente, infatti, l'automa dovrebbe andare in d asserendo erroneamente di aver letto un numero dispari di 1. Non c'è bisogno di altre transizioni etichettate con 0.

Consideriamo il simbolo 1. Sappiamo che anche 1 può occorrere liberamente in qualsiasi punto della stringa in ingresso, ma ogni volta che occorre lo stato dell'automa deve cambiare. Stiamo assumendo di trovarci nello stato p e quindi, con la transizione etichettata 1, dobbiamo andare in d. Si noti che la logica degli stati è rispettata perché l'informazione associata allo stato d è vera *dopo* l'esecuzione di questa transizione (questo perché noi stiamo assumendo che l'informazione associata a p sia vera *prima* della transizione). Per quanto riguarda lo stato p non ci sono altre considerazioni da fare.

Per lo stato d si fanno gli stessi ragionamenti considerando, però, che questa volta si parte dall'assunzione che siano stati letti un numero dispari di 1. Quindi, all'occorrenza di 0, come in p, l'automa non cambia stato, ma all'occorrenza di 1 l'automa deve ritornare in p per preservare la logica degli stati.

Rimangono solo da definire, a questo punto, gli stati finali. È chiaro che in questo automa l'unico stato che può essere considerato di accettazione è p. Questo perché la consegna dice che l'automa deve accettare *solo* le stringhe con un numero pari di 1 e p è proprio lo stato in cui questo è sempre vero.

Si noti che, in generale, assegnare uno stato s come di accettazione non implica che un automa che si trova in s non possa continuare a leggere una stringa in ingresso e continuare quindi a costruire un cammino. È essenziale invece accertarsi che *se l'automa si ferma in uno stato di accettazione allora la stringa letta fino a quel momento sia sempre una stringa del linguaggio assegnato.*

Nel nostro esempio questo è vero perchè una volta che l'automa è entrato nello stato p la stringa letta fino a quel momento può essere sicuramente accettata. La stringa inoltre può continuare con diverse occorrenze di simboli 0 e rimanere sempre una stringa del linguaggio assegnato. Solo leggendo un 1 si cambia stato e si potrà accettare la stringa solo se proseguendo, alla fine, si ritornerà in p.

L'automa che abbiamo costruito è raffigurato in Figura 1.

Facciamo un altro esempio. Supponiamo di voler scrivere un automa che accetti il linguaggio, sempre sull'alfabeto $\{0, 1\}$, di tutte le stringhe che non contengono mai due simboli 1 consecutivi.

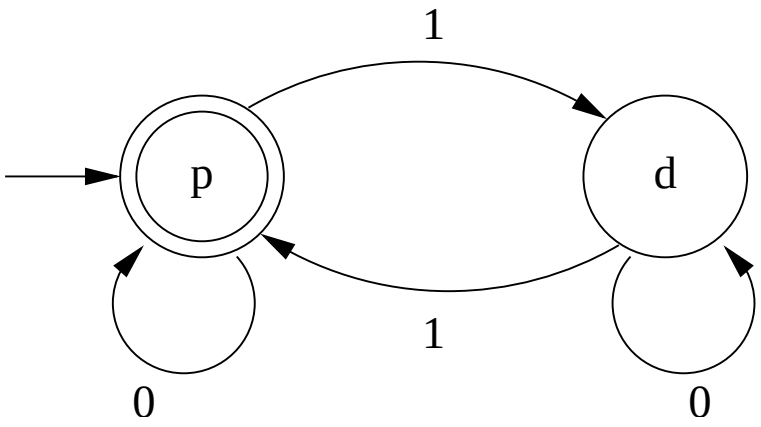

Figure 1: L'automa costruito per riconoscere stringhe con parità di 1.

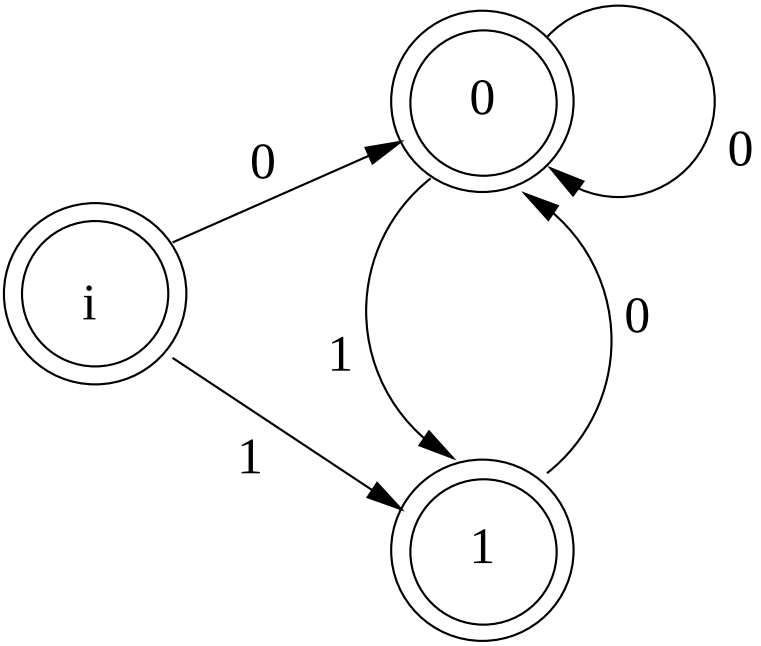

Figure 2: L'automa costruito per stringhe senza 1 consecutivi.

Di nuovo ci va bene qualunque stringa di soli simboli 0 e quindi anche la stringa vuota. Per quanto riguarda i simboli 1 essi di nuovo possono occorrere in qualunque posizione, ma mai due volte di seguito. Per poter realizzare questo vincolo possiamo pensare di ricordare negli stati qual è l'ultimo simbolo che è stato letto nella stringa di ingresso. Essendoci solo due simboli, creiamo due stati: lo stato 0 che ricorda l'informazione: "l'ultimo simbolo letto è 0" e lo stato 1 a cui è associata l'informazione: "l'ultimo simbolo letto è 1".

Come stato iniziale, però, nessuno dei due va bene poiché entrambi presuppongono che ci sia un simbolo letto *precedentemente*. Non c'è problema: possiamo creare tutti gli stati che vogliamo e quindi ne creiamo uno, chiamiamolo i, che serve solo a fare da stato iniziale.

Adesso costruiamo le transizioni di conseguenza. Dallo stato iniziale i partirà una transizione etichettata con 0 verso lo stato 0 e una etichettata con 1 verso lo stato 1. In questo modo l'informazione sugli stati diventa vera al primo passo dell'automa. Facciamo inoltre in modo che non ci sia la possibilità di ritornare nello stato i (questo perché l'abbiamo costruito solo come stato iniziale e un uso diverso potrebbe confondere la nostra progettazione).

Nello stato 0 non ci sono vincoli su quale simbolo può occorrere. Ciò significa che c'è una transizione uscente etichettata con 0 ed una etichettata con 1. Per rispettare la logica degli stati, ovviamente, la prima deve ritornare nello stato 0 mentre la seconda deve andare nello stato 1.

Nello stato 1 bisogna esprimere il vincolo assegnato dal linguaggio.

In generale sappiamo che un automa accetta una stringa se e solo se riesce a costruire per essa almeno un cammino che termina in uno stato di accettazione. Pertanto abbiamo due modi per *non* fare accettare una stringa ad un automa: fare in modo che tutti i suoi cammini terminino in uno stato non di accettazione oppure impedire la costruzione di un cammino per la stringa.

In questo caso utilizziamo la seconda possibilità: semplicemente impediamo all'automa di costruire cammini per stringhe che hanno due simboli 1 consecutivi. Per far questo basta non mettere nessuna transizione uscente dallo stato 1 etichettata con 1. Per il simbolo 0 invece inseriamo la transizione uscente e, per rispettare la logica degli stati, questa transizione deve ritornare nello stato 0.

Manca solo la specifica degli stati finali. In questo caso qualunque stringa che abbia un cammino sull'automa va bene poiché abbiamo escluso tutte e sole le stringhe non accettabili impedendo all'automa di costruire cammini

per esse. Quindi tutti gli stati sono di accettazione.

L'automa è disegnato in Figura 2. Si noti che, da un punto di vista di ottimizzazione, lo stato i è inutile poiché perfettamente equivalente allo stato 0. Pertanto sarebbe possibile scrivere un automa per lo stesso linguaggio con soli due stati. Non essendo stato richiesto un automa minimo non ci dobbiamo preoccupare di questo aspetto. L'automa con lo stato iniziale i in più va benissimo.