UNIVERSITÀ
DEGLI STUDI
DI PALERMO

NDS
NETWORKING AND DISTRIBUTED SYSTEMS
RESEARCH GROUP

Knowledge discovery tramite modelli strutturali basati su RNN

Tesi di Laurea Magistrale in Ingegneria Informatica

Marco Curaba

Relatore: Prof. Salvatore Gaglio
Correlatore: Ing. Marco Ortolani

Indice

Abstract

La **knowledge discovery** è un argomento estremamente attuale il cui interesse pratico è stato colto dalle parole di John Naisbitt *"We are drowning in information but starved for knowledge"* nel suo libro Megatrends. La frase risale a quasi 40 anni fa, eppure coglie in pieno la situazione odierna, in quanto l'ubiquità dei sensori e la dipendenza dell'essere umano dalla tecnologia permettono l'accesso ad una mole di dati sempre in aumento. Basti pensare ai telefoni cellulari i quali da soli possono registrare dati relativi a posizione, movimento, presenza di oggetti vicini e luce ambientale tramite sensori appositi (gps, magnetometri, accelerometri, giroscopi, sensori ad infrarossi) e che sono oggetti d'uso comune. Anche realtà come le *smart homes* e l'*Internet of Things* (IoT) spingono in questa direzione, con lo scopo di mettere le abitazioni e gli oggetti al servizio dell'utente ma con l'effetto secondario di una crescente pervasività dei sensori nella vita quotidiana.
Riprendendo le parole di Naisbitt oggigiorno la vera sfida è rappresentare la *conoscenza* tramite modelli che mettano in evidenza relazioni tra i dati altrimenti nascoste. Si potrebbe essere portati a pensare che la sovrabbondanza di dati sia in questo senso un vantaggio, ma paradossalmente è vero l'opposto. Infatti è difficile distinguere quando i dati sono legati da relazioni realmente esistenti e quando invece il legame sia soltanto apparente e dovuto a similarità statistiche che inevitabilmente emergono dalla mole di dati.

Gli approcci *euristici* cui ci si riferisce come *decision-theoretic* in *Syntactic Pattern Recognition, Applications*, affrontano il problema costruendo un modello indipendente dai dati che rappresenti i pattern tramite vettori di caratteristiche. Un chiaro esempio è dato dalle Artificial Neural Network (ANN) le quali sono un modello ispirato all'apprendimento biologico, oggi molto adoperato in intelligenza artificiale. Questa categoria di metodi è in grado di apprendere i dati ma fallisce nell'evidenziare un legame tra questi ultimi e la struttura del modello e quindi tra la conoscenza e la sua rappresentazione.
Ad essi si contrappongono gli approcci *cognitivi* (*structural* in *Syntactic Pattern Recognition, Applications*) per i quali l'obiettivo è costruire un modello che sia esso stesso la rappresentazione della conoscenza relativa ai dati da cui si è partiti. Passive Learning e Active Learning (AL) sono esempi di questa categoria ed associano la conoscenza a linguaggi regolari, i quali dividono i propri elementi in un insieme finito di classi organizzabili secondo un grafo orientato, il Deterministic Finite Automaton (DFA) il quale verrà quindi associato alla rappresentazione della conoscenza. Gli approcci cognitivi però tendono ad essere particolarmente sensibili agli errori nei dati, soprattutto se questi vengono percepiti come outliers.

L'obiettivo di questa tesi è illustrare una tecnica che combina approcci euristici e cognitivi per la knowledge discovery allo scopo di avere i vantaggi di entrambe le tipologie. Per questo si è scelto di adoperare l'AL che pur essendo un algoritmo cognitivo, richiede la *collaborazione* di un oracolo che verrà quindi costruito a partire da uno degli approcci euristici. Tra gli algoritmi di active learning sono stati scelti L* nato insieme

a questo paradigma di apprendimento (1987) e Observation Pack e TTT che invece nascono in anni più recenti (rispettivamente 2012 e 2015). Tutti e 3 gli algoritmi giungono allo stesso risultato nel caso in cui i dati possiedano una regolarità associabile a quella di un linguaggio regolare. Quando questo non accade i risultati sono approssimazioni che variano a seconda dell'algoritmo scelto. I DFA ottenuti da L* hanno un maggior numero di stati e riconoscono quanti più dati di partenza possibile. Al contrario l'Observation Pack e il TTT arrivano a modelli con un minor numero di stati più adatti a cogliere la legge nascosta. Ciò mostra come la scelta del tipo di algoritmo sia critica e dipenda sia dall'ambito applicativo che dal tipo di risultato desiderato. Come già detto qualunque algoritmo di active learning ha bisogno che gli venga fornito un oracolo, un ente capace di apprendere i dati ma non in grado di estrarre una struttura da essi e dunque non in grado di rappresentare la conoscenza cercata, e a questo scopo è stato deciso di adoperare le Recurrent Neural Network (RNN), un tipo di reti neurali artificiali basato sull'apprendimento di sequenze temporali che tiene conto oltre all'ingresso nell'istante attuale, anche dei valori relativi agli istanti precedenti e dunque un valido candidato per il riconoscimento di linguaggi regolari. La funzione più importante che l'oracolo deve eseguire è indicare se un modello ipotesi corrisponde alla soluzione cercata e in caso negativo fornire una spiegazione del perchè ciò non sia vero tramite un campione che faccia da esempio. L'approccio più diretto per fare ciò tramite una rete neurale consiste nel verificare per un numero arbitrariamente grande di parole del linguaggio se sono classificate correttamente dal modello. Ciò comporta da un lato un eccessivo costo computazionale ripetuto per ogni verifica, e dall'altro il rischio che il modello non rappresenti bene la conoscenza, ma solo il numero ridotto di campioni usati per il controllo. In questa tesi viene illustrato un metodo per la implementazione di un oracolo che adoperi la conoscenza appresa da una rete ricorrente, adoperando assieme ad essa un albero decisionale basato su classificatori binari e mantenendo al contempo una propria rappresentazione strutturale costruendo e aggiornando un proprio DFA, come avviene per l'AL. Il DFA dell'oracolo e quello generato a partire dall'algoritmo di active learning sono costruiti sulla base dello stesso linguaggio ma secondo criteri diversi. Quindi inizialmente saranno differenti ma durante il corso dell'algoritmo queste differenze verranno individuate ed usate per raffinare il modello in errore, in pratica adoperando i due DFA in maniera che si *correggano* reciprocamente. Se quello appreso dalla RNN è un linguaggio regolare allora i due DFA convergeranno verso la struttura che rappresenta correttamente i dati. Tale struttura però non esiste se il linguaggio appreso non è regolare, e dunque si rende necessario fissare un criterio di terminazione che porti a quella che è una approssimazione della rete e del linguaggio.

Gli esperimenti condotti hanno avuto lo scopo di verificare innanzitutto se il metodo è efficace in condizioni ottimali ovvero se il linguaggio dei dati è regolare e se la rete lo ha appreso senza alcun errore. Inoltre si è valutato quanto il modello ottenuto si discosti da quello effettivo in caso di errori nei dati o nell'apprendimento e sempre sotto quest'ottica le differenze tra i tre algoritmi di AL presi in esame.

Capitolo 1

Introduzione

Col termine di **knowledge discovery** si intende l'analisi di grandi quantità di dati grezzi al fine di estrarne nuove informazioni di alto livello. Il suo interesse pratico è riassunto nelle parole di Naisbitt *"We are drowning in information but starved for knowledge"* in [1]. Oggi infatti l'ubiquità dei sensori e la dipendenza dell'essere umano dalla tecnologia rendono facile recuperare moli di dati sempre crescenti da ambienti più o meno controllati, soprattutto se si pensa ad ambiti quali *Data Mining* e *Internet of Things* (IoT). La vera sfida è arrivare alla *conoscenza* tramite modelli che mettano in evidenza relazioni tra i dati altrimenti nascoste. Dai dati infatti non è semplice identificare direttamente leggi o regole generali ed essi contengono una quantità non trascurabile di informazioni ridondanti o inutili. Inoltre spesso i dati tendono a presentare una naturale regolarità sotto forma di pattern ripetuti, in particolar modo se riguardano il comportamento umano o leggi matematiche (fisiche, chimiche), la quale però non viene messa direttamente in evidenza.
Un modello in grado di apprendere questi pattern, rappresentarli e soprattutto di metterli in evidenza permetterebbe di *concretizzare* la conoscenza senza perderne la generalità. Esistono in letteratura molti approcci alla *pattern recognition* (*clustering* dei dati, RNN, HMM...), la maggior parte dei quali però fallisce nel creare un modello con le caratteristiche sopracitate. Il modello adoperato in questa tesi a tale scopo è il Deterministic Finite Automaton (DFA), riconoscitore di **linguaggi regolari**, particolare tipologia di linguaggi formali descritti dalla gerarchia di Chomsky in [2], la cui struttura dipende fortemente dai pattern sintattici appresi. Dunque sotto questa ottica, la conoscenza corrisponderà ai linguaggi regolari nell'ambito di questa tesi il cui punto focale sarà l'estrazione di automi a partire da un insieme di campioni.
Esistono due famiglie di approcci adottabili, l' Active Learning (AL) e il Passive Learning (PL), basate su filosofie contrapposte dell'insegnamento umano.
Il Passive Learning (PL) ha l'insegnante come fulcro dell'insegnamento non permettendo agli studenti di partecipare attivamente alla lezione ma solo di essere presenti e di trarre autonomamente le proprie conclusioni, ed è adoperato principalmente quando il numero di studenti è tale da non permettere che vengano seguiti singolarmente. Per i DFA questo si traduce nel costruire un automa che si sovra-adatti ai campioni noti, procedendo con successive generalizzazioni tramite *merging* di stati. Per quanto ciò si presti molto bene all'estrazione della conoscenza strutturale a partire dai dati e siano stati già effettuati esperimenti nell'ambito del riconoscimento di attività relative a percorsi basandosi sulle loro traiettorie geohash, i quali hanno restituito ottimi risultati, si corre il rischio di avere un algoritmo eccessivamente *data-driven*, finendo per estrapolare non la conoscenza cercata ma solo una visione parziale di essa. Inoltre gli algoritmi di PL si basano sulla esplorazione dello spazio degli stati dei DFA, la quale

diventa costosa all'aumentare della dimensione dell'alfabeto e del numero di stati del DFA target.
Per questo si è deciso di adoperare l'AL, tipologia che invece pone insegnante e studente allo stesso livello e che si basa sul confronto tra queste due figure, incoraggiando interventi e domande da parte dello studente. Il primo framework basato sull'active learning per i linguaggi regolari è quello descritto in [3] e denominato Minimally Adequate Teacher (MAT). In esso vengono definite le caratteristiche di un insegnante o oracolo che dovrà *conoscere* il linguaggio obiettivo e da cui l'allievo estrapolerà il DFA target. In particolare l'allievo costruirà una prima ipotesi che generalizzi molto (un DFA che accetta o che rifiuta qualsiasi sequenza), e procederà per successivi raffinamenti tramite separazione di stati finché l'oracolo non indichi che l'ipotesi è quella corretta. Ciò corrisponde a procedere in maniera *deduttiva* e permette di non *esporre* direttamente l'allievo ai dati, che verranno invece appresi dall'oracolo secondo un algoritmo di tipo diverso. Questo disaccoppiamento permette di definire un metodo che astrae il modello dai dati e di conseguenza consente di confrontare

1. i risultati ottenuti da metodi di apprendimento differenti;

2. la conoscenza relativa a dataset diversi.

Tutto ciò però richiede che il modello con il quale vengono appresi i dati sia *esteso* per poter fare da oracolo all'algoritmo di AL. I metodi di AL infatti garantiscono l'arrivo al linguaggio target se questo esiste ma a patto che l'oracolo abbia determinate caratteristiche, la più complessa delle quali è quella di valutare le ipotesi dell'allievo.
È quindi necessario costruire un oracolo che almeno sia in grado di apprendere i dati in questione e che nel migliore dei casi abbia una struttura che si adatti alla conoscenza celata nei dati, in modo da poterla estrapolare e rappresentare con un DFA. Da questo punto di vista le Artificial Neural Network (ANN) e in particolare le Recurrent Neural Network (RNN) si sono dimostrate particolarmente promettenti, in quanto approcci estremamente popolari e adoperati in letteratura per l'apprendimento di dati che presentano la regolarità accennata prima, come ad esempio il riconoscimento vocale ([4], [5] e [6]), il riconoscimento della scrittura sia in base alla grafia ([7]) che ai movimenti della mano ([8]) e il riconoscimento dell'attività umana ([9]). La loro nascita deriva dall'idea del percettrone descritta in [10], la quale ha stimolato la ricerca, portando a modelli sempre più potenti che possiamo osservare oggi. Esse prendono spunto dai neuroni biologici e ne imitano il comportamento, mimando l'apprendimento del cervello umano, rinforzando o indebolendo i collegamenti tra i neuroni che le compongono per adattarsi meglio ai dati da apprendere.
Pur rappresentando uno strumento molto potente per l'IA, le ANN presentano un insieme di svantaggi non trascurabili:

1. la dipendenza della rete da un insieme di fattori e variabili molto vasto quali

   (a) il trainset e il testset, le loro dimensioni, la loro suddivisione in *batch* e l'ordinamento dei loro campioni

   (b) la *loss function* scelta per la valutazione durante l'addestramento, il tasso di apprendimento, etc...

2. la possibilità che la rete si sovra-adatti agli elementi del trainset;

3. la non immediata intellegibilità della rete stessa.

L'insieme di questi fattori rende le ANN delle *black box* valutabili solo da punti di vista esterni, come ad esempio valori di accuratezza relativi a testset non usati in fase di addestramento. Questo è anche più evidente se si considera la difficoltà oggettiva di riuscire ad individuare che cosa la rete sia addestrata a riconoscere, e per lo stesso motivo anche il confrontare direttamente tra loro due reti risulta impossibile.
In altre parole il tipo di conoscenza fornita da questi approcci si basa sull'*intuizione* e sull'*esperienza* fatta dalla rete in fase di addestramento, non sul *ragionamento*. In [11] l'apprendimento viene descritto come:"processo che cumulativamente aquisisce conoscenza o genera comportamenti e abilità". L'apprendimento delle ANN ricade in particolare nella seconda categoria. Per quanto questo in altri ambiti si dimostri più che sufficiente, manca per la **knowledge discovery** del passo successivo ovvero l'aquisizione della conoscenza stessa e la sua rappresentazione secondo un qualche modello.
Da un lato quindi abbiamo dei modelli in grado di apprendere tramite addestramento grandi quantità di dati che però sono poco interpretabili e non comparabili l'uno con l'altro. Dall'altro invece un approccio che necessita della *collaborazione* di modelli che abbiano già appreso la conoscenza da rappresentare. In questa tesi verranno combinati i due approcci sopra presentati per ottenere i vantaggi di entrambi, adoperando algoritmi di AL con un oracolo basato su ANN. Verrano innanzitutto illustrati tre algoritmi di AL, valutandoli in base ad un insieme di costi indipendenti dal tipo di oracolo adoperato. Quindi verrà descritto un metodo per la costruzione di un oracolo a partire dalla combinazione di più tecniche di deep learning. Infine si esporranno i risultati ottenuti dagli approcci di active learning presi in esame adoperando tale tipo di oracolo.

Capitolo 2

Apprendimento attivo

Nel corso di questo capitolo verrà esposta la parte della tesi relativa all' Active Learning (AL), concentrandoci dunque sui suoi algoritmi e su learner, ignorando per il momento il funzionamento dell'oracolo. Verranno prima ricapitolati alcuni concetti fondamentali riguardanti linguaggi regolari e Deterministic Finite Automaton (DFA), quindi verrà descritto l'AL e le sue basi teoriche per quanto concerne l'apprendimento di strutture semantiche. Infine verranno visti nel dettaglio gli algoritmi presi in esame: **L***, **Observation Pack** e **TTT**.

2.1 Linguaggi regolari, DFA

I linguaggi regolari sono una classe di linguaggi formali, prodotti da grammatiche di livello 3 come definito dalla gerarchia di Chomsky descritta in [2] rappresentata in figura 2.1. Questa gerarchia organizza i linguaggi in classi, ognuna delle quali contiene in sé quelle relative a linguaggi più semplici perché generati con un maggior numero di vincoli. Ogni grammatica ha una espressività maggiore rispetto a quelle di livello inferiore, ma al costo di regole e riconoscitori più complessi. I linguaggi regolari sono i più semplici, e generati da grammatiche con regole tanto restrittive da poter essere riconosciuti da qualunque tipo di riconoscitore della gerarchia.

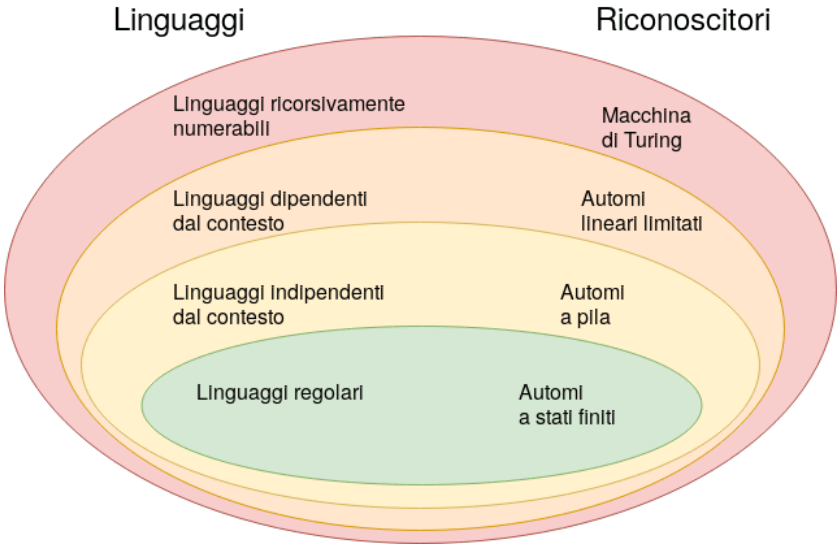

Figura 2.1: Gerarchia di Chomsky

Il riconoscitore più semplice è il DFA che è quindi adatto a rappresentare tali linguaggi senza l'uso di strutture inutilmente complesse o controintuitive, il quale presenta

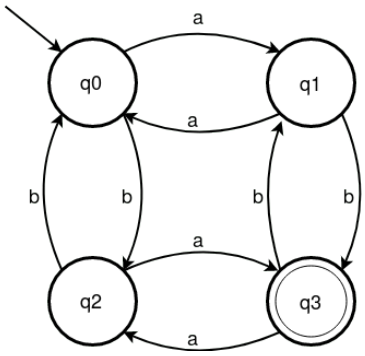

Figura 2.2: DFA che riconosce tutte le parole di "a" e "b" con un numero dispari di a e di b. Lo stato iniziale è $\mathbf{q}_0$ (freccia in ingresso) e l'insieme degli stati accettanti è composto dal solo $\mathbf{q}_3$ (doppia cerchiatura).

inoltre il vantaggio di poter essere appreso a partire da parole del linguaggio tramite gli algorimi che verranno trattati a breve. Nella pratica i DFA sono principalmente adoperati per definire modelli di ricerca (regex) e la struttura lessicale dei linguaggi di programmazione (analisi lessicale), e molto meno per questo tipo di apprendimento.
Come mostrato in figura 2.2 un DFA è caratterizzato da un alfabeto e da un insieme di stati, alcuni dei quali finali o accettanti e di cui uno di partenza. Inoltre ogni stato ha una transizione per ogni simbolo dell'alfabeto verso un altro stato. Verificare se una parola appartiene ad un linguaggio, noto uno dei suoi DFA consiste nel navigare lungo gli stati seguendo le transizioni corrispondenti ai simboli della parola. Di seguito sono presentate alcune definizioni relative a linguaggi regolari e a DFA.

Definition 2.1.1. Stringa d'accesso: Dato un DFA A definito su di un alfabeto Σ, il suo stato iniziale $\mathbf{q}_0$ e un suo stato $\mathbf{q}$, chiamiamo stringa d'accesso a $\mathbf{q}$ qualunque parola $w \in \Sigma^*$ che a partire dallo stato $\mathbf{q}_0$ segua le transizioni di A legate ai simboli di w fino a raggiungere $\mathbf{q}$, per la quale verrà usata la seguente notazione:

$$\mathbf{q} = \mathbf{q}_0[w]$$

Definition 2.1.2. Funzione di output: Dato un linguaggio regolare $\boldsymbol{L}$ definito su di un alfabeto Σ, un suo DFA A e un suo stato $\mathbf{q}$ chiamiamo funzione di output la funzione $f\colon \Sigma^* \to \{0,1\}$ corrispondente al linguaggio, in generale diverso da $\boldsymbol{L}$, che A identifichebbe se $\mathbf{q}$ fosse lo stato iniziale. È evidente che la funzione di output dello stato iniziale $\mathbf{q}_0$ corrisponde al linguaggio $\boldsymbol{L}$ stesso. Per la funzione di output del generico stato $\mathbf{q}_\mathrm{i}$ verrà usata la notazione:

$$\lambda_{\boldsymbol{L}}[\mathbf{q}_\mathrm{i}](s)$$

mentre per la funzione di output dello stato iniziale $\mathbf{q}_0$ verrà usata la notazione semplificata:

$$\lambda_{\boldsymbol{L}}(s)$$

Definition 2.1.3. Congruenza di Nerode: Dato un linguaggio regolare $\boldsymbol{L}$ definito su di un alfabeto Σ, diremo che due parole $w_1, w_2 \in \Sigma^*$ sono **Nerode-congruenti** rispetto al linguaggio $\boldsymbol{L}$ se e solo se:

$$\forall s \in \Sigma^* : \lambda_{\boldsymbol{L}}(w_1 \cdot s) = \lambda_{\boldsymbol{L}}(w_2 \cdot s)$$

Verrà indicata la Nerode-congruenza rispetto al linguaggio $\boldsymbol{L}$ di 2 parole w_1 , w_2 con la notazione:

$$w_1 \cong_{\boldsymbol{L}} w_2$$

2.2 MAT framework

L'apprendimento di linguaggi regolari può essere diviso in due macrocategorie, che devono il nome a due forme di apprendimento dell'essere umano: il Passive Learning (PL) e l'Active Learning (AL). Nel primo caso l'allievo riceve passivamente ogni nozione fornita dall'insegnante senza poter dare il proprio feedback, rendendo l'insegnante il fulcro del processo di apprendimento. L'insegnamento è dunque completamente guidato da quest'ultimo. Nel passaggio a algoritmi software come ad esempio il Regular Positive Negative Inference (RNPI) e il Blue*, egli sarà l'unico agente e in effetti il ruolo dell'allievo sarà completamente assunto dalle strutture dati. Questo approccio in particolare parte dalla soluzione più specifica adoperando una struttura dati denominata prefix tree. Questa consiste in un albero creato a partire dalle parole del training set (insieme che supponiamo prefix-closed) associando uno stato diverso per ogni parola, prefissi compresi. Una tale struttura può sovra-adattarsi ai dati usati nel processo di addestramento, per cui si procede generalizzando tramite successivi merging di stati.
Nell'AL invece l'allievo è a sua volta parte attiva intervenendo per fare domande e avere chiarimenti. In maniera contrapposta rispetto al PL gli algoritmi basati su questo approccio partono dalla soluzione più generale la quale inizialmente può riconoscere qualunque parola o non riconoscerne nessuna e procedono specializzando l'ipotesi per successivi splitting di stati. Questo si riflette nel MAT framework definito in [3]. Nel MAT framework l'insegnante o oracolo (i due termini saranno usati in maniera intercambiabile) deve essere in grado di rispondere a due tipi di domande: la membership e l'equivalence query.

Definition 2.2.1. Membership query: Dato un linguaggio regolare target $\boldsymbol{L}$ definito su di un alfabeto Σ, un oracolo O che rappresenta $\boldsymbol{L}$ tramite le sue strutture interne, non visibili dall'esterno, e una parola $w \in \Sigma^*$ definiamo **membership query** la domanda posta all'oracolo sull'appartenenza o meno di w a $\boldsymbol{L}$, la cui risposta può essere solo **vero** o **falso**.

Definition 2.2.2. Equivalence query: Dato un linguaggio regolare target $\boldsymbol{L}$ definito su di un alfabeto Σ, un oracolo O che rappresenta $\boldsymbol{L}$ tramite le sue strutture interne, non visibili dall'esterno, e un DFA ipotesi A anch'esso definito su Σ definiamo **equivalence query** la domanda posta all'oracolo sulla equivalenza o meno di $\boldsymbol{L}$ e del linguaggio riconosciuto da A, la cui risposta può essere solo **vero** o **falso**. Nel caso in cui la risposta dovesse essere **falso**, O deve fornire un *controesempio* come prova della non equivalenza. Un controesempio è una parola $c \in \Sigma^*$ appartenente ad $\boldsymbol{L}$ ma non riconosciuta da A o viceversa non appartenente ad $\boldsymbol{L}$ e riconosciuta da A.

In genere un oracolo è un qualche tipo di riconoscitore di stringhe per cui tipicamente la membership query risulta meno costosa e più semplice da implementare rispetto all'equivalence query, indipendentemente da come l'oracolo sia stato realizzato. Per rispondere alla membership query l'oracolo dovrà fare ciò per cui è stato pensato, mentre l'implementazione della equivalence query dipende fortemente dall'oracolo adoperato. Ci sono due problemi in cui si incorre nell'implementare l'equivalence query per un oracolo. Inferire un controesempio da strutture dati lontane da un DFA non è banale, e in molti casi vengono approssimate da un serie di membership query. Questo comporta il dover scegliere una condizione di terminazione, sulla base di un numero finito di prove, che può aumentare la probabilità di aver trovato il DFA corretto ma non potrà mai darne la certezza.

2.2.1 Il learning loop

Il *Learning Loop* è l'algoritmo generico di active learning per l'apprendimento di linguaggi regolari. Schematizzato con un maggior dettaglio dallo pseudocodice dell'algoritmo 1, il learning loop può essere così riassunto:

1. fase di creazione della prima ipotesi in seguito alle prime membership query
2. valutazione dell'ipotesi tramite equivalence query, la quale
   (a) restituirà un controesempio se l'ipotesi dovesse rivelarsi errata
   (b) sancirà la fine dell'algoritmo se l'ipotesi dovesse rivelarsi corretta
3. uso del controesempio e di ulteriori membership query per raffinare l'ipotesi, inglobando in essa le nuove informazioni portate dal controesempio
4. ripetizione dal passo 2

Questi passi di alto livello sono realizzati in maniera differente a seconda tipo di approccio di AL adoperato. Si può dimostrare che, se il linguaggio $\boldsymbol{L}$ è regolare e dunque

Algorithm 1 Learning Loop

```
1: procedure LEARN(learner, teacher)
2:     queries ← learner.get_first_queries()
3:     for query in queries do
4:         response ← teacher.membership_query(query)
5:         learner.build_hypothesis(query, response)
6:     end for
7:     hypotesis ← learner.get_hypothesis()
8:     is_correct, counterexample ← teacher.equivalence_query(hypothesis)
9:     while is_correct ≠ true do
10:        queries ← learner.get_queries_for_counterexample(counterexample)
11:        for query in queries do
12:            response ← teacher.membership_query(query)
13:            learner.update_hypothesis(query, response)
14:        end for
15:        hypotesis ← learner.get_hypothesis()
16:        is_correct, counterexample ← teacher.equivalence_query(hypothesis)
17:    end while
18:    return hypothesis
19: end procedure
```

esiste un DFA target e se le ipotesi sono generate in modo da riconoscere correttamente ogni parola restituita come controesempio o da una membership query, questo algoritmo arriva all'ipotesi corretta in un numero finito di passi. Bisogna inoltre sottolineare che gli algoritmi di AL assumono comportamenti differenti in base al tipo di oracolo che si decide di adoperare. Uno stesso DFA può fare da oracolo per il proprio linguaggio, verificando se la parola termina in uno stato accettante come membership query e adoperando un algoritmo chiamato *table filling* come equivalence query. Benchè ciò sia di scarso interesse pratico, perché si arriverebbe tramite il learning loop a ritrovare un DFA già noto, mostra come questo algoritmo possa arrivare ad una soluzione che

generalizzi e ben rappresenti le regole semantiche e la struttura del linguaggio. Gli algoritmi illustrati in questo capitolo (L*, Observation Pack, TTT) infatti trovano tutti il DFA canonico a queste condizioni. Da una prospettiva più realistica durante lo sviluppo di questa tesi si è studiata la possibilità che il linguaggio $\boldsymbol{L}$ non sia noto a priori, ma se ne abbia una visione parziale, limitata ad esempio ad una serie di suoi *campioni*, parole per le quali è nota l'appartenenza o non appartenenza ad $\boldsymbol{L}$. Sono state quindi sperimentate maniere di ottenere oracoli a partire da questi dati. In questo secondo caso, dato che l'oracolo non ha più una conoscenza completa di $\boldsymbol{L}$ si è andati incontro a compromessi per quanto riguarda la correttezza delle soluzioni trovate. Il problema principe è stato il dover simulare una equivalence query scegliendo tra le altre variabili un criterio di terminazione che indicasse quando l' ipotesi dell'algoritmo fosse *abbastanza* corretta. Questo ha dato la possibilità di notare come i tre algoritmi presi in esame restituiscano DFA abbastanza diversi, ognuno dei quali rispecchia le caratteristiche dell'algoritmo che lo ha appreso.

2.3 Classificatori black box

Quale che sia l'algoritmo di AL che scegliamo di adoperare, il passaggio da informazioni recuperate dalle query a DFA non è immediato. Infatti queste informazioni devono comunque essere depositate in apposite strutture dati e mantenute finchè non saranno sufficenti e tali da poter generare la successiva ipotesi. Ovviamente ogni algoritmo di AL ha delle proprie strutture dati diverse da quelle di altri algoritmi, ma tutte si basano su di uno stesso modello matematico, il *classificatore black box*.

Definition 2.3.1. Classificatori black box: Dato un alfabeto Σ, e un sottoinsieme di parole di questo alfabeto che definiamo *insieme dei prefissi corti* definiamo **classificatore black box** o **BBC** una funzione $f : \Sigma^* \to \mathbf{D}$ che associ i prefissi corti agli elementi di un generico *dominio delle classi* $\mathbf{D}$. Un BBC è detto *suffix-based* se ad ogni *classe* $\boldsymbol{c_i} \in \mathbf{D}$ corrisponde un insieme finito di suffissi $\boldsymbol{S_i} \subset \Sigma^*$che chiamiamo *insieme di caratterizzazione* e una funzione $g : \boldsymbol{S_i} \to \mathbf{B}$ definita sull'insieme di caratterizzazione a valori nell'insieme $\{0, 1\}$.

Definition 2.3.2. Insieme di separazione: Dato un alfabeto Σ e un BBC definito su di esso, prese comunque due classi di BBC $\boldsymbol{c_1}$ e $\boldsymbol{c_1}$, definiamo **insieme di separazione** di $\boldsymbol{c_1}$ e $\boldsymbol{c_1}$ l'insieme di tutti i suffissi $\{ s_i \}$ tali che $s_i \in \boldsymbol{S_1}$, $s_i \in \boldsymbol{S_1}$, $g_1(s_i) \neq g_2(s_i)$ con $\boldsymbol{S_1}$ e $\boldsymbol{S_2}$ insiemi di caratterizzazione rispettivamente di $\boldsymbol{c_1}$ e $\boldsymbol{c_2}$ e g_1 e g_2 funzioni rispettivamente delle classi $\boldsymbol{c_1}$ e $\boldsymbol{c_2}$.

Definition 2.3.3. Classificatori black box validi: Dato un alfabeto Σ e un linguaggio $\boldsymbol{L}$ definito su di esso, definiamo **valido** relativamente a $\boldsymbol{L}$ un BBC suffix-based per il quale comunque prese due parole w_1 w_2 , valga:

$$\mathbf{c1} \in \mathbf{D}, \mathbf{c2} \in \mathbf{D}, w_1 \in \mathbf{c1}, w_2 \in \mathbf{c2}$$

$$\mathbf{c1} \neq \mathbf{c2} \Rightarrow w_1 \ncong_{\boldsymbol{L}} w_2$$

Data una qualunque coppia di prefissi corti p_1 e p_2 di un BBC K possiamo quindi stabilire un criterio di equivalenza che dipenda da K, ovvero l'appartenenza dei due prefissi ad una stessa classe di K. Per indicare questa congruenza verrà usata la notazione $p_1 \sim_{\mathbf{k}} p_2$. Se K è valido questa congruenza può essere vista come una versione meno forte della congruenza di Nerode rispetto al linguaggio target $\boldsymbol{L}$. In particolare questo vuol dire che:

$$p_1 \cong_{\boldsymbol{L}} p_2 \Rightarrow p_1 \sim_{\mathbf{k}} p_2$$

e che

$$p_1 \nsim_{\mathbf{k}} p_2 \Rightarrow p_1 \ncong_{\boldsymbol{L}} p_2.$$

Da questo momento in poi ci riferiremo ad *astrazioni di classificatori black box suffix-based validi rispetto ad un linguaggio* semplicemente come classificatori black box se non diversamente indicato.

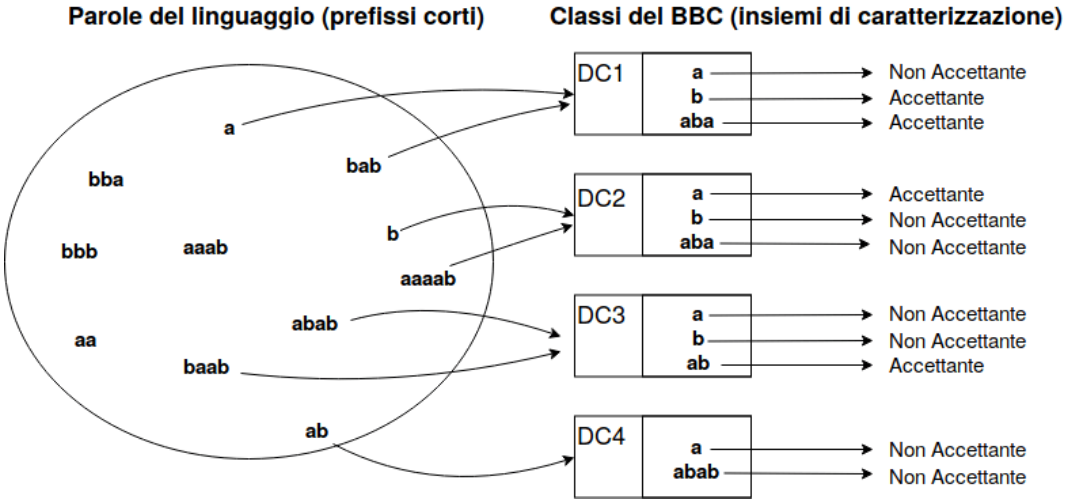

Figura 2.3: Classificatore black box

2.4 Da BBC a DFA

Un classificatore black box può essere usato per rappresentare in maniera sintetica le informazioni raccolte su di un linguaggio regolare. La struttura intermedia fornita dalle classi di un BBC si avvicina infatti a quella dei DFA. In particolare gli stati di un DFA e le classi di un BBC hanno in comune l'essere insiemi di parole dell'alfabeto Σ. Infatti come già detto, ogni stato $\mathbf{q}$ di un DFA si può associare all'insieme di tutte le stringhe d'accesso dello stato mentre ogni classe di un BBC è associabile ai propri prefissi corti. Il lemma seguente evidenzia come un DFA partizioni in classi di equivalenza tramite i suoi stati, tutte le parole di un alfabeto, essendo le parole appartenenti ad uno stesso stato del DFA **Nerode-congruenti** rispetto al linguaggio definito dal DFA. Si noti che il viceversa è vero solo se il DFA è canonico, ovvero se ogni suo stato ha una funzione di output diversa da quelle di tutti gli altri stati e dunque se risulta essere il DFA con meno stati che rappresenti il proprio linguaggio.

Lemma 2.4.1. Dato un linguaggio regolare $\boldsymbol{L}$ e un suo DFA A, scelti comunque uno stato $\mathbf{q} \in \mathbf{Q_A}$ e un suffisso $s \in \Sigma^*$:

$$\forall w_i = p_i \cdot s, w_j = p_j \cdot s : \mathbf{q} = \mathbf{q}_0[p_i] = \mathbf{q}_0[p_j],$$

$$\lambda_{\boldsymbol{L}}(p_i) = \lambda_{\boldsymbol{L}}(p_j) \Leftrightarrow w_i \cong_{\boldsymbol{L}} w_j$$

Dimostrazione:
i seguenti passaggi si limitano a concatenare o separare parole e a sfruttare la funzione di output di stati di DFA

$$\lambda_{\boldsymbol{L}}(w_i) = \lambda_{\boldsymbol{L}}(p_i \cdot s) = \lambda_{\boldsymbol{L}}[\mathbf{q}_0[p_i]](s) = \lambda_{\boldsymbol{L}}[\mathbf{q}](s) =$$
$$= \lambda_{\boldsymbol{L}}[\mathbf{q}_0[p_j]](s) = \lambda_{\boldsymbol{L}}(p_j \cdot s) = \lambda_{\boldsymbol{L}}(w_j)$$

Quindi:

$$\forall s \in \Sigma^* \lambda_{\boldsymbol{L}}(w_i) = \lambda_{\boldsymbol{L}}(w_j)$$

che per definizione (2.1.3) corrisponde a $w_i \cong_{\boldsymbol{L}} w_j$

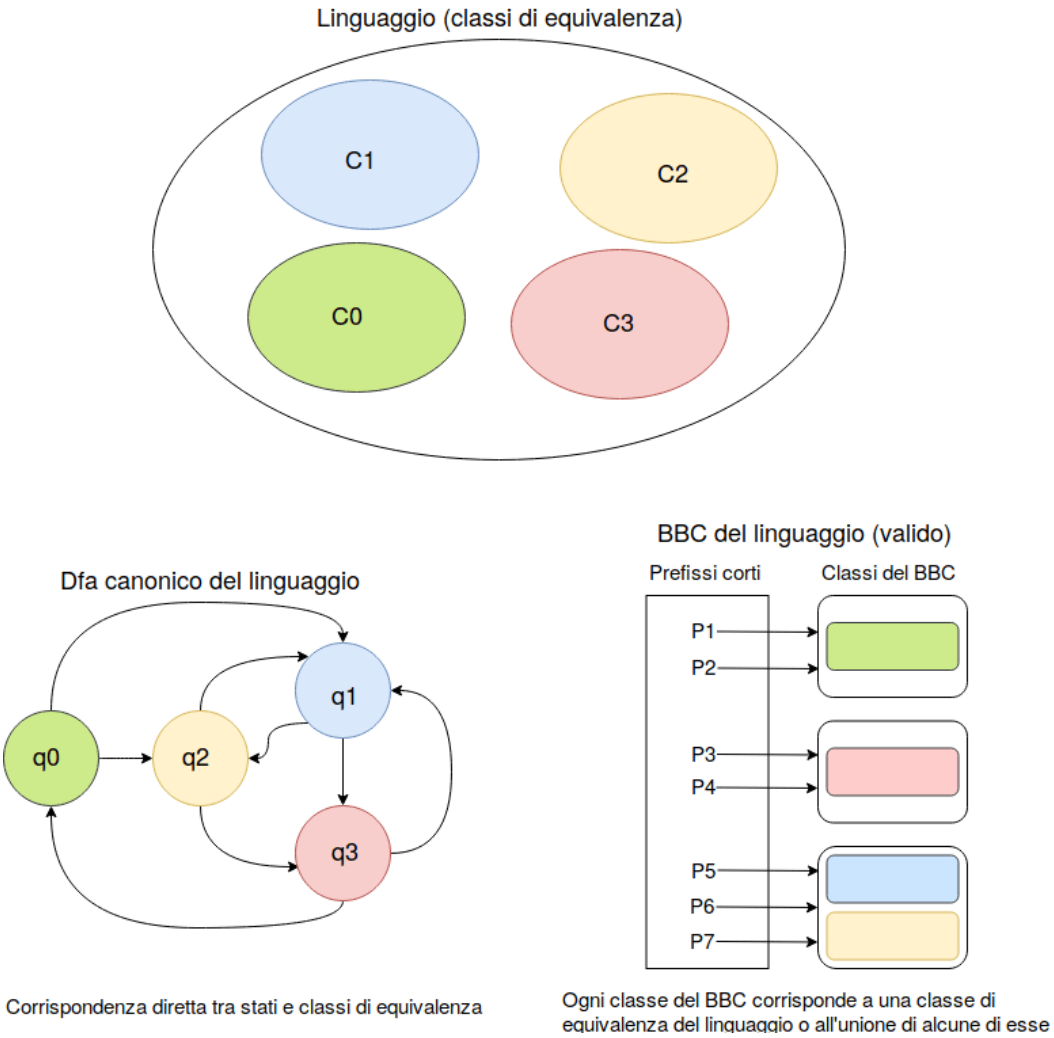

Figura 2.4: Linguaggio regolare, DFA canonico e possibile BBC messi a confronto

Un algoritmo di AL basato sul MAT framework ha quindi come obiettivo il giungere ad un BBC ogni classe contenga elementi tra loro Nerode-congruenti, per poter arrivare al DFA target trasformando le classi in stati. Purtroppo come già visto in generale non è detto che sia così, ed inoltre il verificarlo anche solo per una coppia di prefissi richiederebbe infiniti confronti (uno per ogni suffisso dell'alfabeto Σ). Se il BBC K è valido abbiamo però la certezza di non incorrere in *falsi negativi*, ovvero coppie di parole K-congruenti ma non Nerode-congruenti. Al contrario la validità di K non annulla la possibilità di *falsi positivi*. L'assenza di falsi negativi implica che comunque presa una coppia di prefissi corti Nerode-congruenti, questi verranno mappati da K nella stessa classe, perché se così non fosse avremmo un falso negativo. Per un motivo analogo la presenza dei falsi positivi indica che coppie di prefissi corti non Nerode-congruenti potrebbero comunque essere K-congruenti e quindi finire mappati nella stessa classe. Quindi per un BBC valido ogni classe corrisponde ad uno stato del DFA canonico o all'unione di due o più di essi. Un generico algoritmo di active learning procederà quindi a raffinare un BBC fino a poter costruire un DFA ipotesi che partizioni le classi di equivalenza di $\boldsymbol{L}$. Prima di poter costruire un DFA però è necessario che il BBC

abbia delle proprietà le quali se assenti, vengono ristorate per mezzo di membership query. Quando potrà essere fatta una ipotesi si procederà con una equivalence query. Nel caso in cui questa dovesse dare risultato positivo l'apprendimento terminerebbe, in caso contrario si passerebbe allo studio del controesempio fornito dall'oracolo. In particolare se l'equivalence query dovesse avere risultato negativo vorrebbe dire che DFA ipotesi e target non corrispondono, ma uno o più stati dell'ipotesi devono essere l'unione di stati del target. Il controesempio verrà dunque adoperato dall'algoritmo per identificare almeno uno di tali stati e procedere allo splitting. Questo potrebbe comportare la necessità di ulteriori membership query prima della formulazione di una nuova ipotesi e poi di nuovo ad una equivalence query e così via. Questa serie di splitting "separa" via via le unioni di stati che compongono le classi del BBC e quindi porta sempre più vicino al DFA target.

2.5 Costruzione del DFA

Abbiamo già visto come stati di un DFA e classi di un BBC siano concetti simili in quanto possono essere entrambi associati ad insiemi di parole. Di conseguenza il passare dalle classi di un BBC a gli stati del DFA ipotesi è lineare: per ogni classe distinta verrà creato uno stato. La differenza sostanziale è come questi due enti associno le parole rispettivamente a stati e a classi. Bisogna in particolare trasformare le funzioni di mapping dei BBC in un insieme di transizioni. Le transizioni devono quindi essere "estratte" a partire dalle informazioni contenute nel BBC. Per chiarire il concetto si può ricorrere al seguente esempio pratico per la chiusura di una transizione. Supponendo di voler trovare lo stato di arrivo della transizione relativa al simbolo a con stato di partenza $\mathbf{q_i}$ associato alla i-esima classe del BBC e al prefisso p_i , si procede concatenando p_i e a ottenendo la parola w . Diciamo che la classe dove la parola w viene mappata dal BBC sia la j-esima e che sia qundi associata allo stato $\mathbf{q_j}$. Dunque per lo stato $\mathbf{q_i}$ e il simbolo a la transizione dovrà puntare allo stato $\mathbf{q_j}$
Questo esempio è estremamente semplificato e non tiene conto di alcuni aspetti necessari alla costruzione di DFA ipotesi che riassuma correttamente le tutte la informazioni apprese dal BBC. Di seguito sono definite alcune proprietà dei BBC necessarie a tale scopo.

Definition 2.5.1. Chiusura di un BBC: Dato un BBC K definito sull'alfabeto Σ e l'insieme di tutti i suoi prefissi corti $\mathbf{P}$, diremo che K è chiuso se comunque scelto un suo prefisso corto $p_1 \in \mathbf{P}$ e un simbolo $a \in \Sigma$ esiste sempre $p_2 \in \mathbf{P}$ con $p_2 \sim_{\mathbf{K}} p_1 \cdot a$

Definition 2.5.2. Determinismo di un BBC: Dato un BBC K definito sull'alfabeto Σ e l'insieme di tutti i suoi prefissi corti $\mathbf{P}$, diremo che K è deterministico se comunque scelta una coppia di prefissi corti $p_1, p_2 \in \mathbf{P}$ e un simbolo $a \in \Sigma$ vale $p_1 \cdot a \sim_{\mathbf{K}} p_2 \cdot a$

2.5.1 L'algoritmo

Dato un BBC K chiuso e deterministico è sempre possibile costruire un DFA a partire da esso che classifichi correttamente ogni suo prefisso corto, nel senso che prefissi corti K-congruenti saranno stringhe di accesso dello stesso stato. Per ogni classe del BBC creiamo uno stato del DFA, inoltre scegliamo un prefisso corto che la rappresenti e che verrà usato per la chiusura delle transizioni.
Come mostrato in figura 2.7 per ogni stato $\mathbf{q_i}$ del DFA e per ogni simbolo a dell'alfabeto Σ deve essere calcolata la transizione che parte da $\mathbf{q_i}$ e relativa ad a . Ciò viene fatto verificando per il prefisso corto p_i che abbiamo associato a $\mathbf{q_i}$ di una classe e per

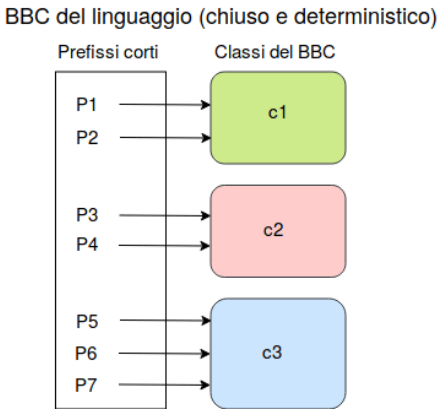

Figura 2.5: Generico BBC chiuso e deterministico, con prefissi corti P1-7 e 3 classi

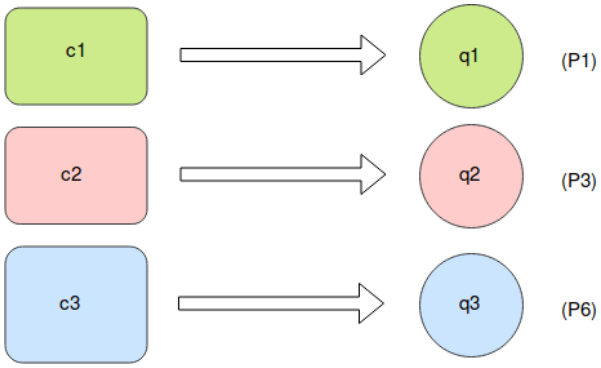

Figura 2.6: Da ogni classe del BBC si costruisce un diverso stato del DFA

ogni simbolo dell'alfabeto, a quale classe corrisponda la parola ottenuta concatenando a alla fine di p_i .
Per la definizione di chiusura di un BBC quale che sia la scelta di p_i e di a deve esistere una classe di K in cui venga mappata la parola $w = p_i \cdot a$, e per la definizione di determinismo di un BBC, data la classe di partenza c_i e il simbolo a , quale che sia la scelta del prefisso p_i tra quelli mappati in c_i , la classe c_j in cui verrà mappato $w = p_i \cdot a$ sarà la stessa.

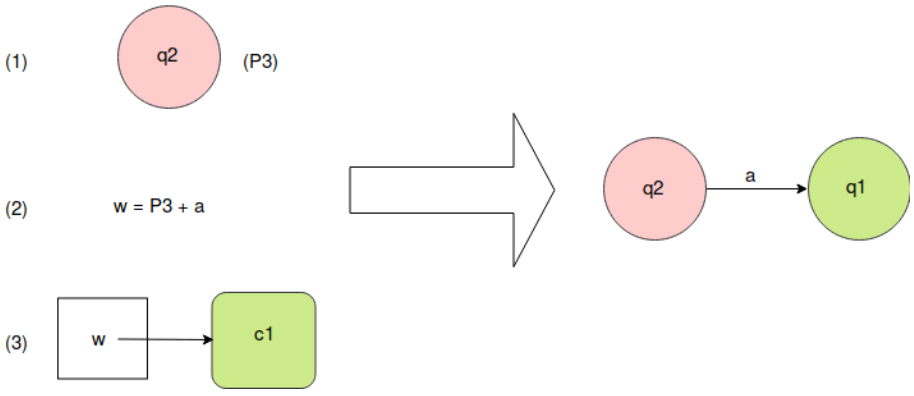

Figura 2.7: Dato il BBC si calcola la transizione che parte dallo stato $\mathbf{q}_2$ relativa al simbolo a

Infine rendiamo accettanti quegli stati relativi alle classi per cui il suffisso ε (che imponiamo essere sempre presente in tutti gli insiemi di caratterizzazione di K) mappa in 1, e scegliamo come stato iniziale quello in cui viene mappato il prefisso corto ε (la parola

vuota). Da notare che in questa costruzione, a parte che per la valutazione degli stati accettanti, non abbiamo tenuto conto delle funzioni g_i di ogni classe e dei relativi insiemi di caratterizzazione. Questo potrebbe essere causa di *inconsistenza*, ovvero di parole classificate diversamente dal DFA e dal BBC, perché già incerti nel BBC.

2.5.2 Consistenza di un BBC

Isberner definisce in [12] i seguenti tipi di inconsistenza di un BBC:

Definition 2.5.3. Inconsistenza relativa alla raggiungibilità: Dato un BBC chiuso e deterministico K, sia A il DFA costruito a partire da esso. Diremo che il la parola $w \in \Sigma^*$ costituisce una **inconsistenza relativa alla raggiungibilità** se e solo se $A[w] \neq K[w]$ ovvero se lo stato $\mathbf{q_i}$ di A che contiene w tra le proprie stringhe di accesso, non corrisponde alla classe in cui K mappa w . Se non esiste parola w che rappresenti una inconsistenza relativa alla raggiungibilità allora diremo che K è *consistente relativamente alla raggiungibilità*

La consistenza relativa raggiungibilità è garantita se l'insieme dei prefissi corti è un insieme prefix-closed, cioè se non esiste un suo elemento che contenga un prefisso non appartenente all'insieme.

Definition 2.5.4. Inconsistenza relativa all'output: Dato un BBC chiuso e deterministico K, sia A il DFA costruito a partire da esso. Diremo che la coppia (p , s), con p prefisso corto e s suffisso facente parte dell'insieme di caratterizzazione della classe di p , costituisce una **inconsistenza relativa all'output** se e solo se $\lambda_{\boldsymbol{L}}[\mathbf{q}_0[p]](s) \neq K(p, s)$ cioè se la funzione di output dello stato $\mathbf{q_i} = \mathrm{A}[\ p\ ]$ e la funzione di output della classe cui appartiene p non restituiscono lo stesso valore per s . Se non esiste coppia (p , s) che rappresenti una inconsistenza relativa all'output allora diremo che K è *consistente relativamente all'output.*

La consistenza relativa all'output è garantita se il BBC *semanticamente suffix-closed*, ovvero se comunque scelto un prefisso corto p_1 , un suffisso s_1 appartenente all'insieme di caratterizzazione di p_1 , e un simbolo $a \in \Sigma$, si ha che il suffisso s_2 per cui vale $s_1 = a \cdot s_2$ appartiene all'insieme di caratterizzazione del prefisso $p_2 = p_1 \cdot a$

Definition 2.5.5. Consistenza relativa all'osservazione: Dato un BBC chiuso e deterministico K, sia A il DFA costruito a partire da esso. K viene detta *consistente all'osservazione* o semplicemente consistente se e solo se comunque scelto un prefisso corto p e un suffisso s appartenente all'insieme di caratterizzazione di p , $\lambda_{\boldsymbol{L}}(p \cdot s) = A[p](s)$. Condizione necessaria e sufficiente affinchè un BBC sia consistente è che sia contemporaneamente consistente relativamente all'output e consistente relativamente alla raggiungibilità.

I DFA ottenuti a partire da BBC consistenti oltre a classificare correttamente i prefissi corti presentano il vantaggio di riconoscere correttamente tutte le parole di cui il BBC è venuto a conoscenza tramite membership query e tutti i controesempi forniti da equivalence query.

2.6 L*

In questa sezione verrà trattato l'algoritmo **L***, prima concretizzazione di algoritmi di active learning basati sul MAT framework, trattato nel dettaglio in [3]. Si vedranno le

strutture dati e come queste implementano l'idea astratta di BBC, in cosa si tradurrà per esse la consistenza e infine come costruire il DFA ipotesi.

2.6.1 Le strutture dati

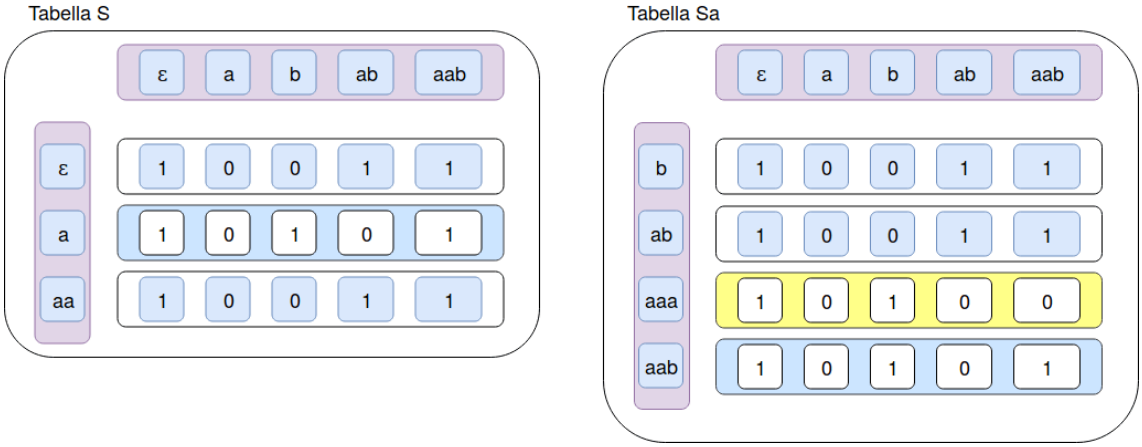

Figura 2.8: Tabelle S ed Sa di L*

Nell'algoritmo L* il Black Box Classifier (BBC) consiste nell'insieme delle due tabelle "S" ed "Sa" mostrate in figura 2.8. L'insieme dei prefissi corti viene mantenuto prefix-closed per assicurarsi la consistenza del BBC. Quindi per ogni nuova parola inserita tra i prefissi corti, verranno aggiunti anche tutti i suoi prefissi non ancora presenti nell'insieme. Si parla inoltre di singolo insieme di caratterizzazione globale, perché questo è comune a tutte le classi, permettendo di rappresentare le informazioni del BBC come una coppia di matrici a due dimensioni. Per mantenere il BBC semanticamente suffix-closed allora è sufficiente che l'insieme di caratterizzazione sia suffix-closed.
Le righe di entrambe le tabelle sono associate a prefissi, mentre ogni colonna corrisponde ad un suffisso appartenente all'insieme di caratterizzazione. In entrambe le tabelle il valore di ogni elemento in posizione (i,j) è quello restituito da una membership query fatta all'oracolo per la parola $w = p_i \cdot s_i$, dove p_i è il prefisso corrispondente all'i-esima riga della tabella, e s_j è il suffisso relativo alla j-esima colonna. In questo caso la funzione che mappa i prefissi nelle classi dipende dai valori delle righe della tabella: dati i prefissi p_i e p_j relativi alle righe i e j, essi apparterranno alla stessa classe di K se e solo se tutti i valori nelle posizioni corrispondenti delle due righe saranno uguali. In figura 2.8 le parole ε, aa, b e ab sono K-congruenti (come evidenziato in figura dalla stessa palette di colori) perché le corrispondenti righe hanno gli stessi valori. Se le due righe dovessero differire per un solo valore allora i due prefissi non sarebbero K-congruenti come è ad esempio per i prefissi corti ε ed a. Inoltre se le due righe differiscono per gli elementi in posizione k-esima allora il suffisso s_k farà parte dell'insieme di separazione delle classi cui appartengono i prefissi. Prendendo di nuovo come esempio ε ed a vediamo che differiscono per i valori in 3ª e 4ª posizione. L'insieme di separazione delle loro due classi sarà allora {b, ab}.
La tabella S ha il duplice scopo della funzione f e delle funzioni g_i di un BBC. Deve associare i prefissi corti noti nelle diverse classi che in questo caso corrispondono a vettori di booleni e deve associare gli elementi dell'insieme di caratterizzazione di ogni classe in {0,1}, ovvero fare da funzione di output per le classi. Nell'esempio in figura 2.8 la tabella S svolge il ruolo della funzione f associando {ε, a, aa} alle classi c_0 e c_1 rappresentate dai vettori [1,0,0,1,1] e [1,0,1,0,1]. Considerando adesso la classe c_0

la tabella S assume il ruolo di g_0 associando l'insieme di caratterizzazione ai valori {0,1} tramite il vettore di riga [1,0,0,1,1]. Avremo quindi $g_0(\varepsilon) \to 1, g_0(a) \to 0$ e così via.
Come già mostrato per poter calcolare le transizioni di un DFA a partire da un BBC chiuso e deterministico bisogna considerare per i prefissi corti ogni a-successore con $a \in \Sigma$. La tabella Sa serve ad assicurarsi che ogni prefisso corto del nostro BBC abbia il proprio a-successore se questo non dovesse già essere presente in S. In figura 2.8 vediamo che la tabella Sa contiene le righe relative alle parole aaa e aab, perché è presente in S il prefisso corto aa per il quale dovranno essere gestiti i successori relativi al simbolo a e a quello b. Invece non è presente in Sa alcuna riga riguardo al prefisso aa perché questo è già presente in S. Possiamo dire che contiene le informazioni *ad un simbolo di distanza* rispetto ad S, per poter calcolare le transizioni durante la costruzione del DFA ipotesi.
Le due tabelle hanno due insiemi separati di prefissi (cioè ogni prefisso appare solo in una tabella) ma esattamente lo stesso insieme di suffissi. Questo si rende necessario perché deve essere possibile confrontare tra loro prefissi appartenenti ad S e ad Sa per i quali devono quindi essere note i valori delle funzioni di output relativi agli stessi suffissi. Se le tabelle formano una BBC chiusa e deterministica, ovvero se l' insieme di prefissi corti è prefix-closed e l' insieme di suffissi è suffix-closed, allora come già detto il BBC risulterà consistente.
Va notato infine il fatto che le due tabelle non mantengono, durante tutta l'esecuzione dell'algoritmo, chiusura e determinismo. Ovvero le operazioni che effettuiamo su di esse possono portare alla non consistenza del BBC, ma questo non costituisce un problema in quanto queste proprietà verranno ripristinate prima di generare qualunque ipotesi, tramite membership query e aggiunte di righe e colonne.

2.6.2 L'algoritmo

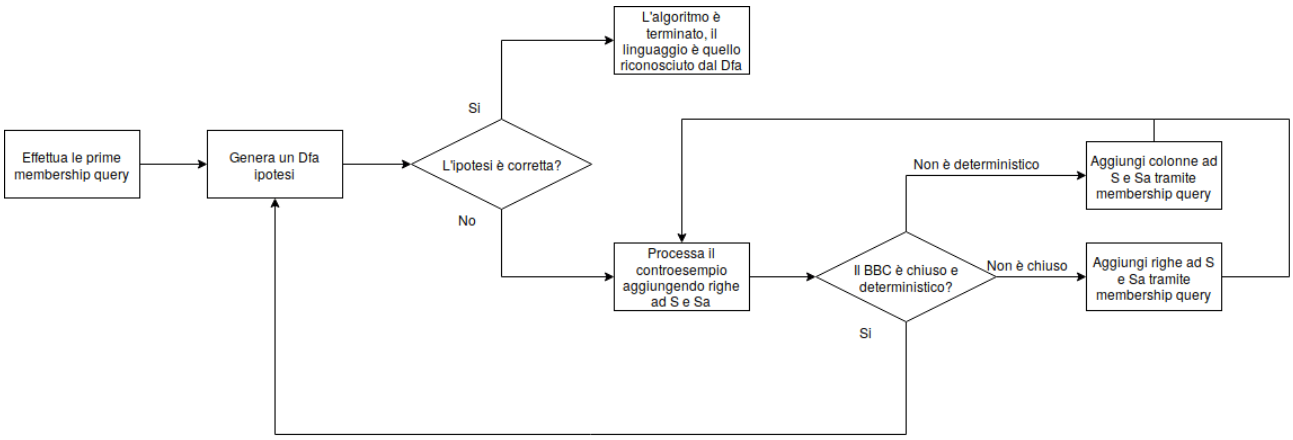

Figura 2.9: Diagramma di flusso dell'algoritmo L*

L'algoritmo L* è descritto sinteticamente dal diagramma di flusso in figura 2.9: dopo la costruzione della prima ipotesi, si passa alla valutazione di questa tramite equivalence query, il che può portare al termine dell'algoritmo se l'ipotesi è corretta o alla generazione di un controesempio da parte dell'oracolo in caso contrario. Il controesempio viene quindi processato portando alla modifica delle tabelle S ed Sa e alla non consistenza del BBC. Si procede quindi con delle equivalence query per riportare il BBC in uno stato chiuso e deterministico per riprovare con nuova equivalence query ricominciando il ciclo.

Costruzione della prima ipotesi

La costruzione della prima ipotesi da parte di L* richiede che si effettui una membership query per ogni simbolo dell'alfabeto e una membership query per la stringa vuota ε. La parola vuota sarà l'unico prefisso corto, e ogni parola di lunghezza 1 $w = a$ con $a \in \Sigma$ sarà usata come a-successore di ε Quindi S conterrà una sola riga corrispondente ad ε mentre Sa conterrà tante righe quanti sono i simboli dell'alfabeto. L'insieme di caratterizzazione sarà composto dal solo suffisso ε, il che comporta che ogni prefisso verrà valutato solo in base alla sua alla sua appartenenza a $\boldsymbol{L}$ e che sia S che Sa avranno una sola colonna relativa a ε. La prima ipotesi per un linguaggio con alfabeto $\Sigma = \{a, b\}$ è mostrata in figura 2.10.
Il BBC creato fino a questo punto è ovviamente prefix-closed, essendo l'insieme dei prefissi corti composto dal solo ε, e suffix-closed visto che lo stesso vale per l'insieme di caratterizzazione. Il BBC deve inoltre essere necessariamente deterministico: come già detto il non determinismo si verifica quando due prefissi appartenenti alla stessa classe hanno due a-successori relativi allo stesso simbolo a che appartengono a classi diverse. Avendo a disposizione il solo prefisso ε trovare due prefissi distinti per cui valga quanto appena detto è impossibile. Al contrario la proprietà di chiusura non è

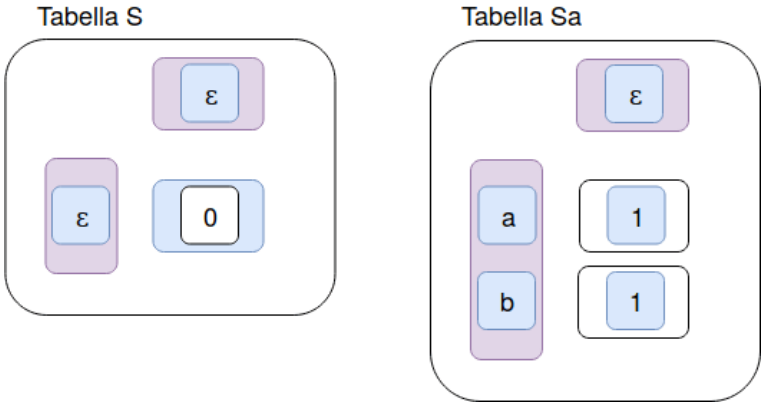

Figura 2.10: Struttura di una possibile prima ipotesi dell'algoritmo L*

garantita. Data la presenza di un'unica colonna gli stati del BBC possibili sono i due relativi ai seguenti vettori: [0] e [1] ma come già detto la presenza di un solo prefisso corto comporta che questo BBC possa avere un solo stato. Uno solo dei due vettori sarà associato a ε e all'unica riga di S e il relativo stato sarà l'unico del BBC. Ogni riga di Sa corrispondente a questo vettore rappresenta una transizione chiusa, ovvero che punta ad uno stato esistente (attualmente l'unico). Potrebbero però esistere righe diverse rispetto a quella di S se banalmente per il linguaggio $\boldsymbol{L}$ si verifica $\lambda_{\boldsymbol{L}}(\varepsilon) \neq \lambda_{\boldsymbol{L}}(a_i)$ per qualche $a_i \in \Sigma$ come avviene nel caso in figura 2.10 dove entrambe le righe di Sa sono diverse da quella in S. In questo caso abbiamo transizioni verso stati non esistenti per cui il BBC risulta non chiuso e si rende necessaria l'aggiunta di righe ad S e Sa come vedremo più avanti.
Infine la chiusura insieme al determinismo, che è garantito in questo caso, ci consentono di costruire il primo DFA ipotesi. Si noti che ogni DFA generato dall'algoritmo riconosce correttamente tutte le parole per cui è stata fatta una membership query o fornite come controesempio, data la consistenza del BBC. Ciò è dovuto al fatto che le proprietà di prefix-closedness e semantic suffix-closedness come vedremo saranno mantenute per tutto l'algoritmo.

Chiusura

Si ricordi che la definizione di chiusura per generici BBC richiede per qualunque prefisso corto p_i e per qualunque simbolo $a \in \Sigma$ che la parola $w = p_i \cdot a$ a-successore di p_i possa essere associata ad una classe del BBC. Dato che tutti gli a-successori sono contenuti in Sa come righe, questo si traduce per L* nel verificare che ogni riga di Sa sia effettivamente uno stato del BBC e cioè che esista per ogni riga di Sa almeno una riga di S identica. In figura 2.11 abbiamo che il BBC risulta non chiuso perché esiste in Sa la riga [1, 1], relativa al successore "b" evidenziato in rosso, che non corrisponde ad alcuna riga in S.

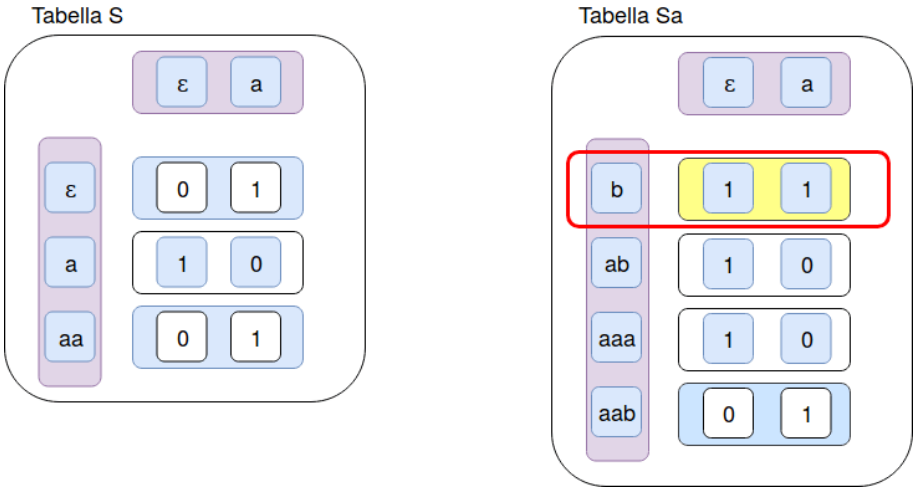

Figura 2.11: Le tabelle S ed Sa costituiscono un BBC non chiuso

Ripristinare la chiusura del BBC per l'algoritmo L* una volta identificato un vettore riga presente in Sa e assente in S è molto semplice: si procede prelevando da Sa la riga in questione ed aggiungendola ad S. Questo in pratica corrisponde ad aggiungere lo *stato mancante* al DFA ipotesi. Inoltre dato che un nuovo prefisso corto è stato aggiunto bisogna, per mantenere le informazioni sulle transizioni, aggiungere ogni suo a-successore all'insieme Sa. In figura 2.12 si vede come la riga di Sa relativa alla parola "b" venga spostata in S, e i suoi a-successori "ba" e "bb", evidenziati in verde in figura, aggiunti ad Sa. Per conoscere i valori degli elementi di riga di questi ultimi si rende necessario effettuare una serie di membership query all'oracolo.
È facile dimostrare come l'insieme dei prefissi corti si mantenga prefix-closed: chiamiamo w il prefisso da aggiungere e supponiamo che il BBC fosse effettivamente prefix-closed prima della sua aggiunzione. Sappiamo che esso era un a-successore di un qualche altro prefisso corto p_i . Dato che p_i faceva parte dei prefissi corti e che il BBC era prefix-closed ogni prefisso di w era già presente in S (a parte lo stesso w). Quindi aggiungere w all'insieme dei prefissi corti non viola la sua proprietà di prefix-closedness, proprietà che come abbiamo detto è necessaria alla consistenza del BBC. Lo stesso ovviamente vale per la semantic suffix-closedness dato che l'insieme di caratterizzazione non viene modificato.
Bisogna notare come la singola aggiunzione di un nuovo prefisso corto non necessariamente porti alla proprietà di chiusura del BBC. Questo è ovvio se si pensa che potrebbero esserci più righe diverse di Sa non presenti in S. Il singolo passaggio è quindi necessario ma non sufficiente. Infine una sola cosa è assicurata sulla proprietà di determinismo dopo questa operazione: se il BBC non era deterministico rimarrà non deterministico. Al contrario se il BBC era deterministico l'operazione potrebbe sia mantenerlo tale che renderlo non deterministico.

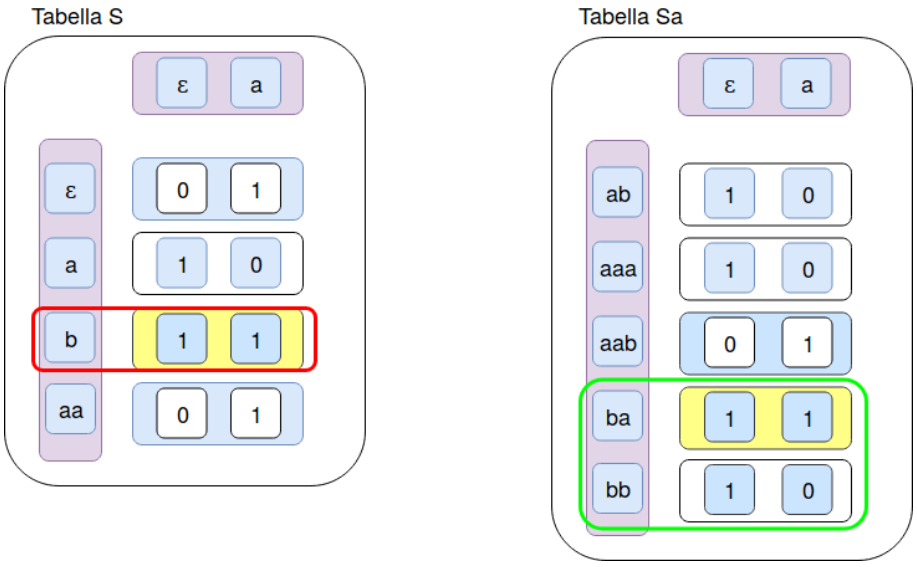

Figura 2.12: L'aggiunta di righe ad S e Sa ha reso chiuso il BBC

Determinismo

La definizione di determinismo per generici BBC richiede che presi comunque due prefissi corti p_i e p_j appartenenti ad una stessa classe del BBC e un simbolo $a \in \Sigma$ che le parole $w_i = p_i \cdot a$ e $w_j = p_j \cdot a$ siano anch'esse associata alla stessa classe del BBC (non necessariamente la stessa di p_i e p_j). Questo si traduce per le tabelle S ed Sa nel trovare una coppia di prefissi corti p_i e p_j ed un simbolo $a \in \Sigma$ tali che le righe di p_i e p_j in S r_{p_i} e r_{p_j} siano uguali e gli a-successori $w_i = p_i \cdot a$ e $w_j = p_j \cdot a$ corrispondano a righe di S o Sa r_{p_i} e r_{p_j} diverse tra loro. In figura 2.13 abbiamo che il BBC risulta non deterministico per via dei prefissi corti "ε" e "aa", corrispondenti a righe uguali di S, in rosso in figura, e del simbolo b: i b-successori "b" e "aab" infatti corrispondono a righe e quindi a classi diverse, evidanziate in figura dai colori verde e blu.

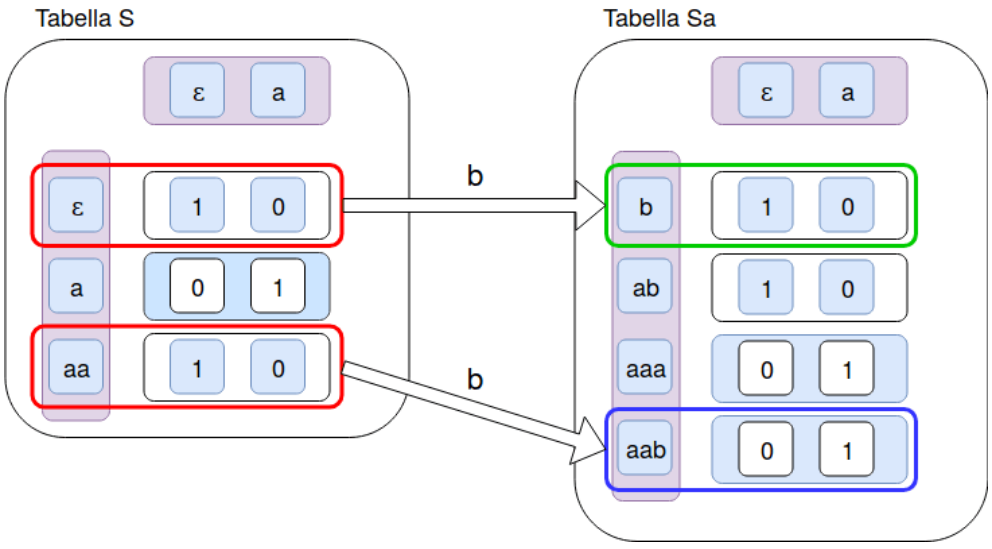

Figura 2.13: Le tabelle S ed Sa costituiscono un BBC non deterministico

In pratica questo consiste nel dire che sono stati individuati dei prefissi corti che il BBC associa erroneamente alla stessa classe. Per rendere un BBC deterministico, una volta individuata una coppia di prefissi corti p_i e p_j e un simbolo a con le caratteristiche

appena viste, bisogna aggiungere una colonna ad S ed Sa in modo da poter separare la classe cui attualmente appartengono p_i e p_j, in due classi distinte. Per fare ciò la nuova colonna dovrà essere relativa ad un *discriminatore* di p_i e p_j per il linguaggio $\boldsymbol{L}$, ovvero ad un suffisso che concatenato a p_i e p_j generi due parole delle quali una ed una soltanto risulti appartenente ad $\boldsymbol{L}$.
Consideriamo w_i e w_j a-successori di p_i e p_j per il simbolo a, che come detto prima risultano appartenere a classi diverse. L'appartenenza a classi diverse implica che le loro righe in S o Sa differiscano per almeno un elemento, per cui deve esistere almeno un discriminatore s tra w_i e w_j. Deve quindi valere:

$$\lambda_{\boldsymbol{L}}(w_i \cdot s) \neq \lambda_{\boldsymbol{L}}(w_j \cdot s).$$

Ricordando che w_i e w_j sono a-successori di p_i e p_j otteniamo:

$$\lambda_{\boldsymbol{L}}(p_i \cdot a \cdot s) \neq \lambda_{\boldsymbol{L}}(p_j \cdot a \cdot s).$$

Infine, chiamando $s_1 = a \cdot s$ la precedente può essere riscritta come:

$$\lambda_{\boldsymbol{L}}(p_i \cdot s_1) \neq \lambda_{\boldsymbol{L}}(p_j \cdot s_1).$$

Abbiamo quindi trovato s_1 discriminatore di p_i e p_j che se aggiunto come nuova colonna di S ed Sa farà si che p_i e p_j vengano associati a classi diverse. In figura 2.14 vediamo un esempio di questa operazione: i prefissi corti "ε" e "aa" che prima appartenevano alla stessa classe, portano tramite il simbolo b alle parole "b" e "aab" separate dal suffisso "ε". Per risolvere il non determinismo in questo caso si aggiunge la nuova colonna $b \cdot \varepsilon = b$ che separa i prefissi corti "ε" e "aa".

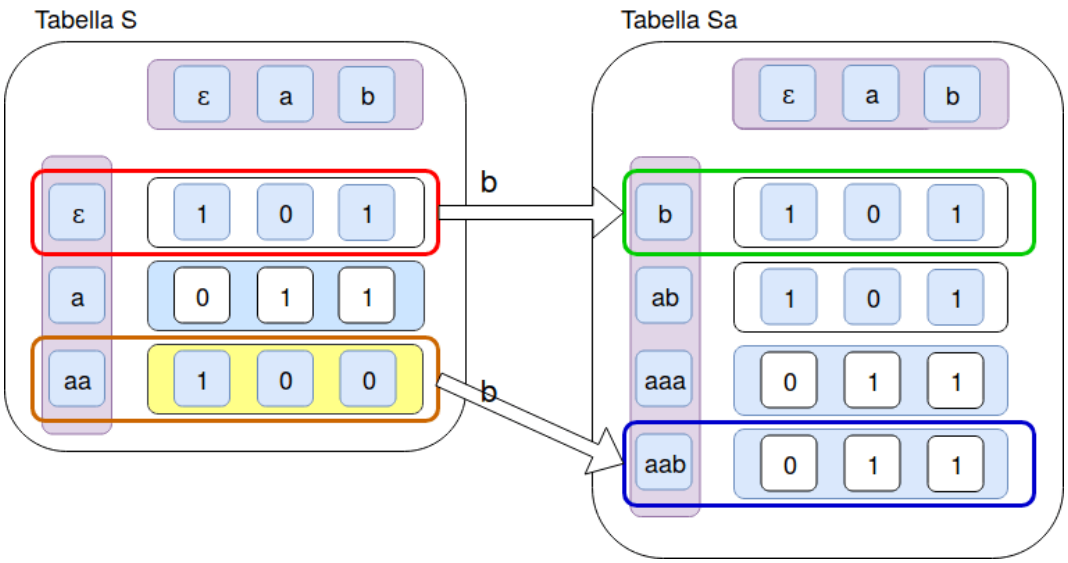

Figura 2.14: L'aggiunta di una colonna ad S e Sa ha reso deterministico il BBC

Questa operazione inoltre non elimina la semantic suffix-closedness del BBC se questa era già presente come è facile dimostrare: il nuovo suffisso s_1 corrisponde infatti alla concatenazione del simbolo a e del suffisso s già presente nell'insieme di caratterizzazione. Quindi se il BBC era già suffix-closed ogni suffisso di s_1 (tranne s_1 stesso fa già parte dell'insieme di caratterizzazione. La semantic suffix-closedness è quindi mantenuta. Ovviamente lo stesso vale per la prefix-closedness del BBC dato che non viene modificato l'insieme dei prefissi.

Anche in questo caso questa operazione potrebbe non bastare ad eliminare il non determinismo, essa è necessaria ma non sufficiente. Potrebbero infatti esserci più coppie di prefissi corti con le caratteristiche viste prima. Infine è bene notare come questa operazione non assicuri affatto il mantenimento del determinismo del BBC

Gestione del controesempio

Come già detto ogni ipotesi proposta dall'algoritmo deve essere validata dall'oracolo, che in caso di responso negativo dovrà restituire un controesempio. Si consideri innanzitutto che l'esistenza di un controesempio indica che il DFA target ha almeno uno stato in più del DFA ipotesi. Questo perché il BBC è valido rispetto al linguaggio $\boldsymbol{L}$, il che comporta che ogni stato dell'ipotesi corrisponda ad un singolo stato o all'unione di due o più stati del DFA target. Se esiste un controesempio quindi deve esistere uno "*stato unione*", che è necessario dividere in due o più stati, e in particolare il controesempio fornito, o uno dei suoi prefissi, è una stringa di accesso a questo stato unione.
Per l'algoritmo L* questo si traduce nell'aggiunta di una nuova classe del BBC. Il controesempio va quindi aggiunto all'insieme dei prefissi corti, insieme ad ogni suo prefisso non ancora presente. Questo oltre a mantenere il BBC prefix-closed assicura che la nuova classe sarà presente nella tabella S. Infine bisognerà aggiornare la tabella Sa aggiungendo una riga per ogni a-successore del controesempio, passo necessario per il calcolo delle transizioni durante la costruzione dell'ipotesi.
Il BBC in seguito all'aggiunzione del controesempio risulterà necessariamente non deterministico: infatti deve esistere almeno una classe del BBC che aveva come successore la classe unione, e che in seguito alla sua divisione contenga due prefissi corti, uno dei quali con un a-successore appartenente alla classe preesistente, e l'altro con a-successore appartenente alla nuova classe. Quindi una classe del BBC punta per lo stesso simbolo $a \in \Sigma$ contemporaneamente a due classi diverse. Tutto ciò può essere visualizzato meglio ragionando in termini di DFA.

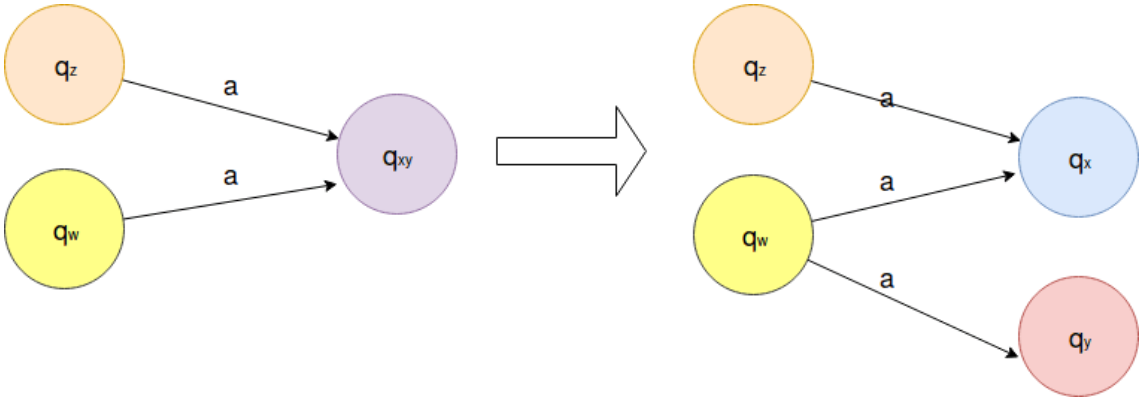

Figura 2.15: Divisione di uno stato dovuta ad un controesempio

Dalla figura 2.15 vediamo come lo stato $\mathbf{q}_{\mathrm{xy}}$ venga suddiviso nei due stati $\mathbf{q}_{\mathrm{x}}$ e $\mathbf{q}_{\mathrm{y}}$. Lo stato $\mathbf{q}_{\mathrm{w}}$ il cui a-successore era $\mathbf{q}_{\mathrm{xy}}$, puntando in seguito alla divisione di $\mathbf{q}_{\mathrm{xy}}$ ad entrambi gli stati per lo stesso simbolo, diventa la causa del non determinismo. Si noti che non tutti gli stati che puntavano allo stato unione saranno necessariamente causa di non determinismo. Infatti ciò non accade per lo stato $\mathbf{q}_{\mathrm{z}}$ dato che evidentemente tutte le transizioni puntavano ad a-successori appartenenti solo a $\mathbf{q}_{\mathrm{x}}$ e non a $\mathbf{q}_{\mathrm{y}}$. Il processare un controesempio assicura che ciò accadrà almeno per uno stato.
Infine bisogna aggiungere che il BBC potrebbe risultare anche non chiuso. Questo è evidente se si considera la possibilità che uno o più a-successori del controesempio, che

come detto andranno a fare parte della tabella Sa, corrispondano a classi non presenti in S.

Costruire il DFA

L'algoritmo di costruzione del DFA ipotesi è analogo a quello del caso generale. La chiusura e il determinismo del BBC assicurano che per ogni stato $\mathbf{q}$ del DFA ipotesi e per ogni simbolo $a \in \Sigma$ esista e sia unica la transizione con $\mathbf{q}$ come stato di partenza e relativa al simbolo a. Come mostrato in figura 2.6 ad ogni classe del BBC che in questo caso corrisponde a vettori riga, verrà associato uno stato del DFA. Quindi per ogni stato $\mathbf{q}$ e simbolo a verrà calcolata la relativa transizione prendendo un prefisso corto p appartenente a $\mathbf{q}$ e verificando a quale stato appartenga l' a -successore di p (figura 2.7). Ricordiamo infine che la scelta del prefisso corto per ogni classe è indifferente per via delle proprietà di chiusura e determinismo.

2.7 Observation Pack

Il secondo algoritmo di active learning preso in esame è l'**observation pack** descritto in [13]. La principale differenza di questo approccio rispetto a L*, differenza che si rispecchia molto anche nelle strutture dati, è che mentre L* manteneva un insieme di caratterizzazione globale, ovvero comune ad ogni classe, per l'observation pack vale esattamente l'opposto. Infatti ogni classe dell'observation pack mantiene il più piccolo insieme di caratterizzazione necessario a differenziarla da tutte le altre.
In particolare per ogni coppia di classi c_1 e c_2 del Black Box Classifier (BBC) si avrà a disposizione un solo discriminatore. Questo non era vero per L*: in quel caso era facile, avendo il BBC un insieme di caratterizzazione globale, che ogni coppia di classi avesse più di un discriminatore. Infatti per l'algoritmo L* l'insieme di separazione tra due classi potrebbe essere costituito dall'intero insieme di caratterizzazione.

2.7.1 Le strutture dati

Nell'observation pack sono adoperati due tipi di alberi: lo **span-tree** e il **discrimination-tree**. Il discrimination-tree è la funzione di classificazione del BBC: esso è adoperato per associare le parole a classi a partire da un insieme di suffissi, approssimando la congruenza di Nerode. I nodi interni del discrimination-tree corrisponderanno infatti a suffissi, mentre i nodi foglia saranno le classi riconosciute. Lo span-tree tiene invece traccia delle transizioni, e genera nel contempo l'ipotesi. Si può infatti considerare lo span-tree come l'albero ricoprente del DFA ipotesi, anche se in effetti l'ipotesi viene generata a partire dai due alberi.
Come per l'algoritmo L* anche in questo caso i concetti di classe del BBC e stato del DFA ipotesi vanno quasi a coincidere, per alcuni versi anche maggiormente rispetto ad L*. Di seguito quindi verrà usato il termine classe per indicare sia i nodi foglia del discrimination-tree (che appunto fa da classificatore del BBC) che l'insieme di nodo foglia (del discrimination-tree) e relativo nodo dello span-tree. Invece il termine stato verrà usato esclusivamente per i nodi dello span-tree.
In figura 2.17 ad esempio con classe c_1 potremmo indicare indifferentemente sia il quadrato con etichetta $\mathbf{q}_1$ che la coppia di cerchio e quadrato etichettati $\mathbf{q}_1$. Con stato $\mathbf{q}_0$ invece indicheremo il solo cerchio $\mathbf{q}_0$.

Lo span-tree

Per capire al meglio il ruolo dello span-tree per questo algorimo è bene rivedere la definizione di albero di copertura di generici grafi orientati (categoria che include i DFA). Nonostante in letteratura span-tree e albero di copertura indichino lo stesso concetto, in questo contesto avranno significati leggermente diversi. Verrà usato il termine span-tree per riferirsi alla struttura dati propria dell'algoritmo Observation Pack e quindi relativa a DFA. Il termine albero ricoprente avrà invece un'accezione più generale e più in linea con la letteratura, riferendosi a generici grafi. Si noti che quando detto implica che lo span-tree sia un particolare albero di copertura.

Definition 2.7.1. Albero di copertura: Dato un grafo orientato G, ovvero un insieme di nodi collegati da archi direzionati, definiamo **albero di copertura** del grafo, un qualunque albero che contenga tutti i nodi di G ma solo un sottoinsieme dei suoi archi.

In particolare l'albero di copertura non potrà contenere insiemi di archi che porterebbero a cicli, perché altrimenti non si tratterebbe più di un albero. In pratica l'albero di copertura di un grafo contiene le informazioni per raggiungere ogni nodo del grafo a partire da un nodo scelto arbitrariamente come radice. Non conterrà però ogni percorso possibile, ma solo un percorso per ogni nodo destinazione.
Inoltre quando gli archi sono distinguibili e ordinabili, come avviene per i DFA dato che i simboli degli archi possono essere disposti ad esempio in ordine alfanumerico, è possibile generare un albero di copertura che rispecchi un qualche criterio. Ad esempio è possibile organizzare l'albero tramite esplorazione in ampiezza o in profondità del grafo.

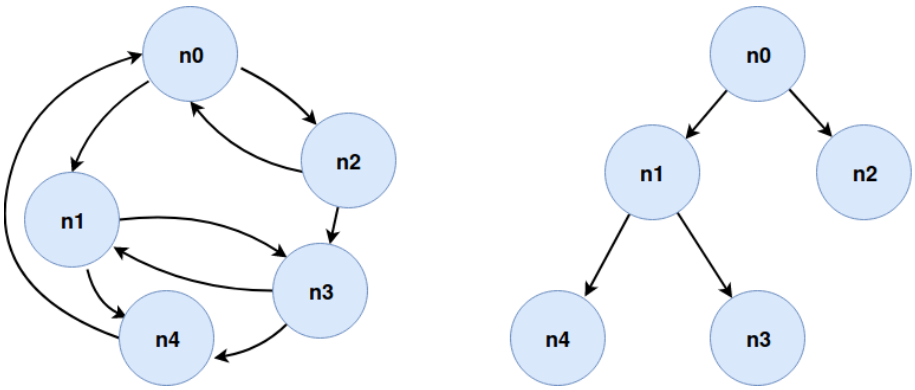

Figura 2.16: Grafo orientato e un possibile albero di copertura con radice n_0

Lo span-tree è l'albero di copertura del DFA ipotesi H, la cui radice è sempre lo stato iniziale di H e la cui organizzazione dipenderà dal linguaggio e dai controesempi forniti. La differenza rispetto al normale albero di copertura è che, mentre di solito si parte dal grafo per arrivare all'albero, in questo caso si partirà dallo span-tree per giungere al DFA ipotesi. Questo è possibile perchè per lo span-tree conserviamo oltre ai suoi rami, cui faremo riferimento col termine **s-transizioni**, altre transizioni che giungono ai nodi del discrimination-tree e che chiameremo **d-transizioni**. Queste ultime sono però caratteristiche solo dello span-tree e non degli alberi di copertura. Quindi per essere precisi lo span-tree è un albero di copertura con l'aggiunta delle d-transizioni.

Costruzione dell'ipotesi

Per rendere più chiaro il ruolo dello span-tree di seguito verrà mostrato un esempio di costruzione di un DFA ipotesi, anche se ancora non sono stati definiti chiusura e determinismo del BBC dell'Observation Pack né visto nel dettaglio il discrimination-tree. Basti sapere che il BBC preso in esame è chiuso e deterministico e che ogni foglia del discrimination-tree corrisponde ad un nodo dello span-tree.

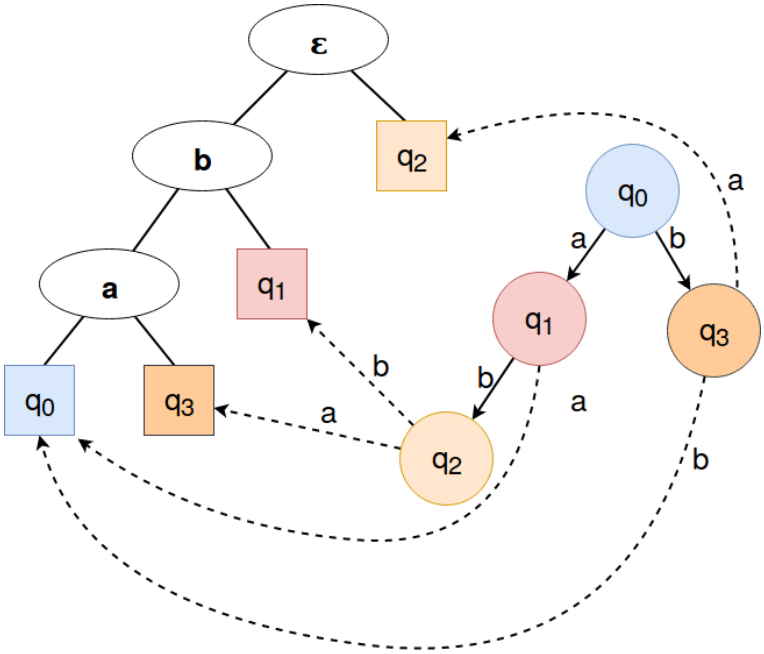

Figura 2.17: Esempio di BBC chiuso e deterministico dell'Observation Pack

In figura 2.17 vediamo come le strutture dati dell'Observation Pack interagiscano: il discrimination-tree associa prefissi a classi e di conseguenza in nodi dello span-tree, mentre quest'ultimo mantiene le informazioni sulle transizioni del DFA ipotesi. Distinguiamo inoltre tra s-transizioni e d-transizioni (in figura mostrate come frecce rispettivamente continue e trattegiate): entrambe partono da nodi dello span-tree ma mentre le prime puntano ancora a nodi dello span-tree le seconde invece a foglie del discrimination-tree.
Possiamo dire che le s-transizioni sono *informazioni certe* che non cambieranno mai durante il corso dell'algoritmo: ad esempio le s-transizioni dello stato $\mathbf{q_0}$ in figura punteranno sempre agli stati $\mathbf{q_1}$ e $\mathbf{q_3}$. Le d-transizioni invece dipendono dalla classificazione del BBC, che come per L* cambia per adattarsi meglio alle membership query e ai controesempi forniti. Dunque lo stato $\mathbf{q_1}$ punta almeno temporaneamente alla classe c_0 tramite la d-transizione "a", perché il discrimination-tree classifica così il suo "a"-successore, ma ciò potrebbe cambiare in seguito ad un controesempio futuro. Data questa rapida lettura dello span-tree possiamo passare alla costruzione vera e propria del DFA ipotesi.
Questo passaggio è particolarmente facile per l'Observation Pack. Infatti l'intera struttura dello span-tree viene semplicemente replicata nel DFA ipotesi: ogni nodo diventerà uno stato, ogni s-transizione una transizione del DFA. L'unica differenza è data dalle d-transizioni: queste copiate nel DFA ipotesi non punteranno più alle classi ma invece ai relativi stati. Questo si concretizza nella figura 2.18 dove ad esempio la d-transizione relativa al simbolo "a" che dallo stato $\mathbf{q_3}$ andava alla classe c_2 punterà invece nel DFA ipotesi allo stato $\mathbf{q_2}$.

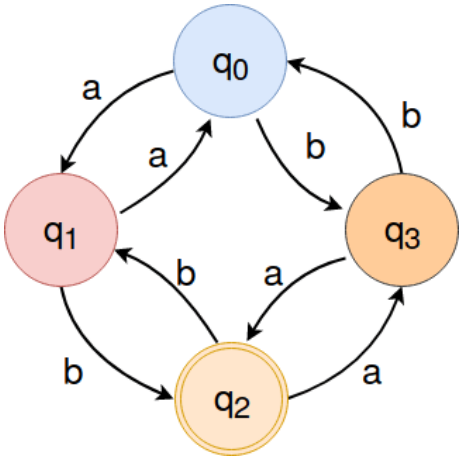

Figura 2.18: Possibile DFA ipotesi dell'Observation Pack

Il discrimination-tree

Il discrimination-tree è per l'Observation Pack il classificatore del BBC. Come anticipato le classi corrispondono ai nodi foglia e ogni prefisso p verrà classificato partendo dal nodo radice del discrimination-tree e eseguendo una serie di membership query. A seconda del risultato della membership query relativamente ad un nodo interno n_i , si procederà verso il sottoalbero destro o sinistro di n_i continuando fino a giungere ad un nodo foglia che rappresenta la classe di quel prefisso.
In particolare ad ogni nodo interno n_i è associato un suffisso s_i che verrà concatenato al prefisso da classificare ottenendo la parola $w_i = p \cdot s_i$. La membership query fatta dal nodo n_i per il prefisso p riguarderà la parola w_i . Consideriamo di voler

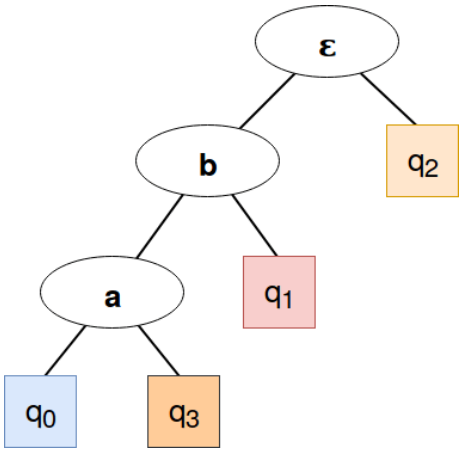

Figura 2.19: Discrimination-tree

classificare il prefisso p = "abaab" tramite l'albero in figura 2.19 adoperando l'oracolo O. Il nodo radice è sempre associato al suffisso ε per cui la prima membership query riguarderebbe la parola $w_0 = p \cdot s_0 = abaab \cdot \varepsilon = abaab$. Supponendo che la membership query di O per w sia negativa procederemo per il sottoalbero sinistro (al contrario se w fosse appartenuta al linguaggio $\boldsymbol{L}$ la scelta sarebbe ricaduta sul sottoalbero destro). Procedendo in questo modo un possibile svolgimento è il seguente:

$$w_0 = abaab \cdot \varepsilon = abaab$$

$$O.mem\_query(w_0) = 0$$

$$w_1 = abaab \cdot b = abaabb$$

$$O.mem\_query(w_1) = 1$$

$$abaab \in c_1$$

In pratica dal nodo radice siamo scesi lungo il sottoalbero destro dato che "abaab" non appartiene ad $\boldsymbol{L}$, dopodichè dal nodo n_b siamo scesi lungo il ramo destro arrivando alla classe c_1 data l'appartenenza di "abaabb" al linguaggio.

Gli insiemi di caratterizzazione

Riprendendo la definizione generale di BBC, l'insieme di caratterizzazione $\boldsymbol{S_i}$ di una classe c_i è un insieme di suffissi tali che per qualunque coppia di prefissi corti $p_1, p_2 \in c_i$ e per qualunque suffisso $s \in \boldsymbol{S_i}$ vale:

$$\lambda_{\boldsymbol{L}}(p_1 \cdot s) = \lambda_{\boldsymbol{L}}(p_2 \cdot s).$$

Nell'Observation pack ogni classe ha un proprio insieme di caratterizzazione, al contrario di quanto accadeva per l'algoritmo L* dove si aveva un insieme di caratterizzazione globale. Anzi l'Observation Pack mantiene per ogni classe l'insieme di caratterizzazione più piccolo possibile in grado di distinguerla da ogni altra classe. Ricordando che le classi sono identificate da foglie del discrimination-tree è evidente che l'insieme di caratterizzazione della classe c_i è dato da tutti gli antenati del relativo nodo foglia. Infatti per giungere alla foglia si è passati per ogni suo antenato rispondendo a quelle membership query proprie dell'insieme di caratterizzazione.
Ad esempio in figura 2.19 abbiamo che la classe c_3 ha insieme di classificazione $\boldsymbol{S_3}$ = {"ε", "a", "b"}. Questo vuol dire che ogni prefisso corto di c_3 forma con quei suffissi rispettivamente due parole non accettate da $\boldsymbol{L}$ (per "ε" e "a") e una parola accettata (per il suffisso "b").

Gli insiemi di separazione

L' insieme di separazione di due classi c_1 e c_2 di un generico BBC corrisponde a tutti quei suffissi s comuni agli insiemi di separazione delle due classi per i quali vale:

$$\lambda_{\boldsymbol{L}}(p_1 \cdot s) \neq \lambda_{\boldsymbol{L}}(p_2 \cdot s)$$

con $p_1 \in c_1$ e $p_1 \in c_1$. Per l'Observation Pack l'insieme di separazione di qualunque coppia di classi si riduce ad un solo elemento cui ci riferiremo come **elemento di separazione**. Ciò è dovuto proprio alla peculiare struttura ad albero binario del discrimination-tree. In particolare l'elemento di separazione tra due qualunque classi di un BBC dell'Observation Pack è il più profondo antenato comune alle due classi. Infatti è il punto di *biforcazione* tra i percorsi per le due diverse classi.
In figura 2.19 l'elemento di separazione tra la classe c_3 e la classe c_1 è dato dal loro antenato comune più in profondità: il nodo etichettato con "b". Infatti dato che c_3 appartiene al suo sottoalbero sinistro mentre c_1 fa parte di quello destro deve valere per quanto detto sopra:

$$\lambda_{\boldsymbol{L}}(p_1 \cdot b) \neq \lambda_{\boldsymbol{L}}(p_3 \cdot b)$$

con p_1 e p_3 generici prefissi corti di c_1 e c_3 .

2.7.2 Proprietà del BBC

Perché tramite l'Observation Pack si giunga al DFA target, il BBC deve avere le proprietà già elencate per l'L*. Queste sono determinismo, chiusura consistenza relativa alla raggiungibilità e consistenza relativa all'output. È bene ricordare che mentre chiusura e determinismo sono condizioni necessarie e sufficienti perché si possa creare un DFA ipotesi a partire dal BBC, non assicurano che tale ipotesi riconosca correttamente ogni parola ottenuta tramite membership o equivalence query fatta all'oracolo. Quindi non diversamente dall'algoritmo L*, l'Observation Pack procederà ad effettuare membership query atte a rendere il BBC chiuso, deterministico e consistente, per poter generare l'ipotesi, verificarla tramite equivalence query e eventualmente ricominciare il tutto.
Prima di procedere è utile dare la seguente definizione:

Definition 2.7.2. Prefisso caratteristico: Dato un BBC dell'Observation Pack e un suo generico stato $\mathbf{q_i}$, chiamiamo **prefisso caratteristico** il prefisso p_i ottenuto dalla concatenazione dei simboli delle s-transizioni da attraversare per giungere dal nodo radice dello span-tree allo stato $\mathbf{q_i}$. L'insieme di tutti i prefissi caratteristici corrisponde all'insieme dei prefissi corti del BBC, ed esiste per ogni stato solamente un prefisso caratteristico.

In pratica ogni classe del BBC contiene un solo prefisso corto il quale diventa l'unico rappresentante per quella classe. Nell'esempio in figura 2.17 i prefissi caratteristici degli stati $\mathbf{q_0}$, $\mathbf{q_1}$, $\mathbf{q_2}$ e $\mathbf{q_3}$ sono rispettivamente "ε", "a", "ab" e "b".

Chiusura

Durante l'Observation Pack il BBC può risultare non chiuso in un solo momento. Ciò avviene quando la gestione di un controesempio porta alla creazione di un nuovo stato $\mathbf{q}$. Per questo nuovo stato non esisteranno inizialmente transizioni uscenti (né s-transizioni né d-transizioni). Saranno dunque necessarie delle membership query per creare delle d-transizioni uscenti da $\mathbf{q}$ e che puntino alle corrette classi del BBC. La

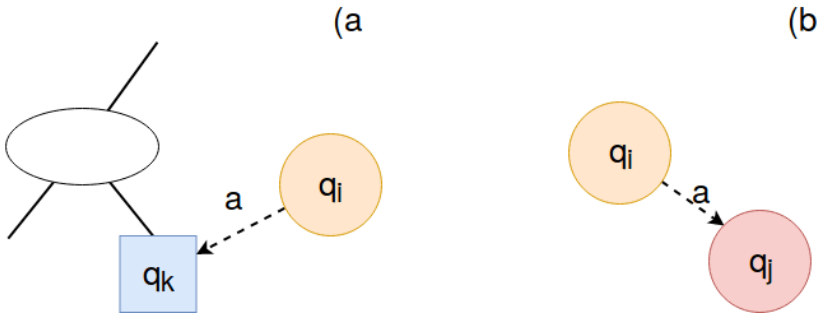

Figura 2.20: Chiusura per i BBC dell'Observation Pack

d-transizione "a" dello stato $\mathbf{q_i}$ che in figura 2.20.a puntava alla classe $\mathbf{q_k}$ diventa una s-transizione che punta al nuovo stato $\mathbf{q_j}$ (fig 2.20.b) in seguito ad un controesempio. Lo stato $\mathbf{q_j}$ è quindi inizialmente senza transizioni. Queste vengono aggiunte andando a verificare gli a-successori del prefisso caratteristico di $\mathbf{q_j}$ per ogni $a \in \Sigma$.

Determinismo

Come vedremo la gestione del controesempio porta alla creazione di nuove classi del BBC. Questo si potrebbe riflette oltre che in un nuovo nodo dello span-tree, nella

crescita del discrimination-tree e nella trasformazione di una delle sue foglie in un nodo interno. Tutti gli stati dello span-tree con transizioni verso quel nodo sono causa di non determinismo.

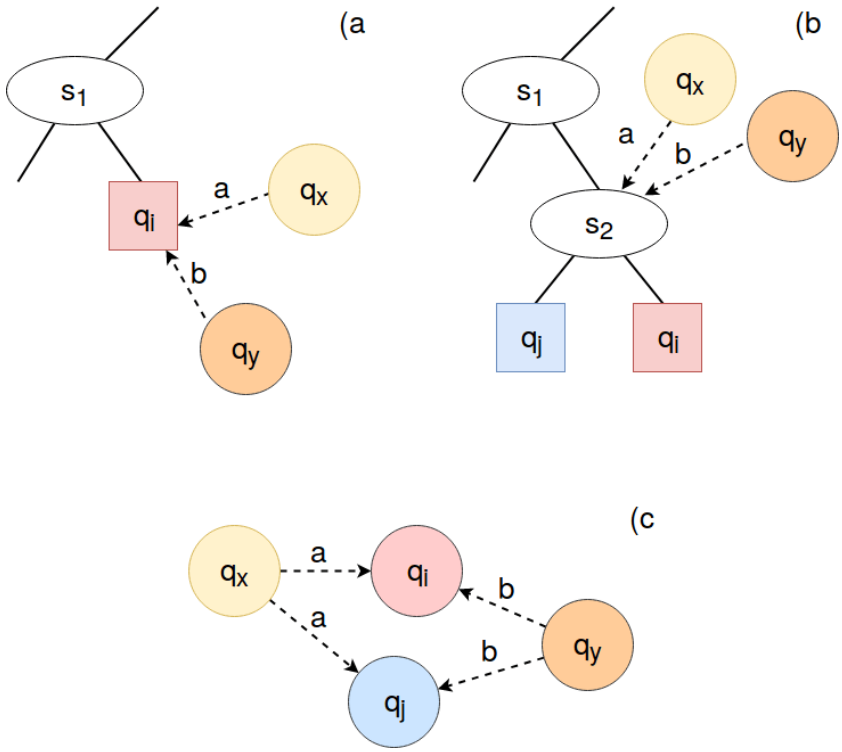

Figura 2.21: Determinismo per i BBC dell'Observation Pack

Ciò è più chiaro se si considera l'esempio in figura. In seguito ad un controesempio la classe $\mathbf{q}_i$, a cui giungono alcune d-transizioni, viene divisa (fig. 2.21.a). Il relativo nodo foglia diviene un nodo interno, le cui foglie corrispondono alle due nuove classi $\mathbf{q}_i$ e $\mathbf{q}_j$(fig. 2.21.b). Le d-transizioni continuano però a puntare al nodo interno. Se provassimo a costruire l'ipotesi sia $\mathbf{q}_i$ che $\mathbf{q}_j$ sarebbero possibili successori di $\mathbf{q}_x$ per la transizione "a" e di $\mathbf{q}_y$ per "b" e non potendo scegliere l'una o l'altra, ognuna di quelle transizioni del DFA ipotesi dovrebbe essere doppia e puntare ad entrambi i relativi stati (fig. 2.21.c), da cui deriva il non determinismo.

Consistenza relativa alla raggiungibilità

Come già visto la consistenza relativa alla raggiungibilità di un BBC chiuso e deterministico implica che qualunque prefisso corto sia associato dal BBC e dal suo DFA ipotesi a una classe e a uno stato tra loro corrispondenti. Si è visto altresì che condizione sufficiente affinchè un BBC chiuso e deterministico sia consistente alla raggiungibilità è che il suo insieme di prefissi corti sia prefix-closed. Questo è il caso per l'Observation Pack. Infatti ogni prefisso corto è anche prefisso caratteristico di uno stato e si ottiene concatenando i simboli delle s-transizioni dello span-tree. Ciò significa che, a parte per il nodo radice il cui prefisso caratteristico è sempre ε, ogni prefisso caratteristico si può ottenere dalla concatenazione di un altro prefisso caratteristico e del simbolo della s-transizione entrante nel relativo nodo. Questo è evidente osservando lo span-tree in figura 2.17.
Dunque per induzione vale che l'insieme di prefissi di un BBC dell'Observation Pack è sempre prefix-closed e dunque il BBC è consistente alla raggiungibilità purché sia chiuso e deterministico.

Consistenza relativa all'output

Come visto nella definizione 2.5.4 l' inconsistenza relativa all'output consiste in una coppia p s , con p generico prefisso corto del BBC e s suffisso appartenente all'insieme di caratterizzazione di p , tale che la parola $w = p \cdot s$ sia riconosciuta differentemente dalle strutture dati del BBC e dal suo DFA ipotesi. Questo problema può verificarsi quando le strutture dati sono *distanti* dal DFA ipotesi, come accade per gli algoritmi tabellari di active learning come ad esempio l'L*.
Abbiamo già visto come invece per l'Obsevation Pack le strutture dati in pratica corrispondano al DFA ipotesi quando il BBC è chiuso e deterministico. Esse sono un DFA per il quale è stata stabilita anche una sorta di *gerarchia degli stati* basata su alcuni suffissi, data dal discrimination-tree. Risulta quindi evidente l'impossibilità di discrepanze tra strutture dati e DFA ipotesi per l'Observation Pack, quale che sia la parola scelta e dunque anche per qualunque coppia di p ed s .
Il BBC dell'Observation Pack è dunque consistente all'output a patto di essere chiuso e deterministico (condizioni necessarie e sufficienti alla costruzione dell'ipotesi).

2.7.3 L'algoritmo

L'Observation Pack non differisce come sequenza di passi ad alto livello dal learning loop e dall'L*. Ciò che lo differenzia e che verrà evidenziato in questa sezione è il come questi passi siano eseguiti nello specifico. In particolare vedremo l'inizializzazione delle strutture dati e la gestione del controesempio avendo già spiegato come costruire il DFA ipotesi.

Inizializzazione delle strutture dati

Il primo passo dell'algoritmo è costruire lo span-tree e il discrimination-tree. Inizialmente entrambi gli alberi sono composti dal solo nodo radice. Lo span-tree sarà composto dal solo stato $\mathbf{q_0}$ senza alcuna transizione, ed associato al prefisso caratteristico ε. Il discrimination-tree avrà il solo nodo radice corrispondente al suffisso ε, al quale verrà aggiunta la classe $\mathbf{q_0}$ come figlio sinistro o destro a seconda dell'appartenenza ad $\boldsymbol{L}$ di ε.

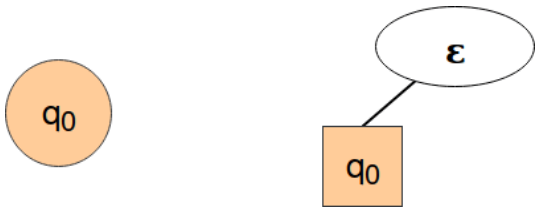

Figura 2.22: Inizializzazione delle strutture dati (1)

In figura 2.22 immaginando di voler costruire un BBC che riconosca il linguaggio $\boldsymbol{L}$ basato sull'alfabeto $\Sigma = \{a, b, c\}$ consideriamo lo stato $\mathbf{q_0}$ non accettante e dunque inserito come foglia sinistra del discrimination-tree. A questo punto devono essere gestite le transizioni uscenti da $\mathbf{q_0}$, una per ogni simbolo $a \in \Sigma$. Ogni simbolo con la stessa accettanza di $\mathbf{q_0}$ per $\boldsymbol{L}$ diventerà un d-transizione verso la classe $\mathbf{q_0}$. Per il primo simbolo con accettanza diversa da $\mathbf{q_0}$ (se questo dovesse esistere) verrà creato il nuovo stato $\mathbf{q_1}$ con relativa classe. Al nuovo stato verrebbe quindi associato il simbolo come prefisso caratteristico e la sua classe andrebbe inserita nel nodo libero del

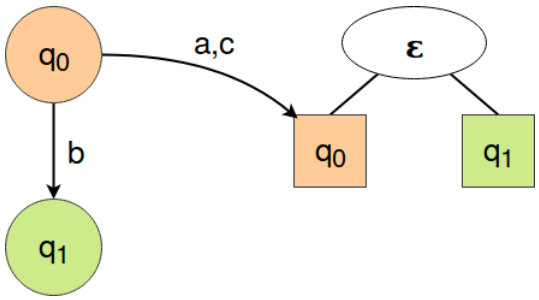

Figura 2.23: Inizializzazione delle strutture dati (2)

discrimination-tree.
Per chiudere le transizioni di $\mathbf{q_0}$ verranno effettuate delle membership query per i simboli "a", "b" e "c". Supponendo che "a" e "c" non appartengano al linguaggio verranno considerati parte di $\mathbf{q_0}$. Avremo quindi delle d-transizioni verso quella classe. Al contrario la parola "b" appartiene al linguaggio, per cui viene creato il nuovo stato $\mathbf{q_1}$ con prefisso caratteristico "b" classificato come foglia destra del discrimination tree (figura 2.23). Dovranno essere quindi calcolate le transizioni uscenti dallo stato $\mathbf{q_1}$ che in ogni caso risulteranno essere d-transizioni e che punteranno alla classe $\mathbf{q_0}$ o alla classe $\mathbf{q_1}$ a seconda dei risultati delle membership query per "ba", "bb" e "bc".
A questo punto, indipendentemente dal numero di simboli dell'alfabeto, avremo pronta la prima ipotesi che consisterà di uno o, come in questo caso, due stati.

Gestione del controesempio

La gestione del controesempio non è di per se una operazione particolarmente complessa per l'Observation Pack. Ma per capire come funzioni e sopratutto perché, è bene chiarire qualche concetto riguardo lo span-tree, i suoi stati e i relativi prefissi caratteristici. Ogni prefisso caratteristico appartiene ad una classe diversa del BBC ed è al contempo una stringa d'accesso per il proprio stato. In particolare vi giungerà solo tramite s-transizioni.
Ed è questo il motivo per cui le s-transizioni non vanno modificate: per evitare che uno stato non abbia più il proprio prefisso caratteristico come stringa di accesso. Infatti è necessario che i prefissi caratteristici appartengano tutti a classi differenti, perché per essi abbiamo trovato tramite query precedenti, suffissi che dimostrano la loro Nerode incongruenza per il linguaggio $\boldsymbol{L}$. In generale però gli stati del DFA ipotesi avranno altre stringhe di accesso oltre al prefisso caratteristico.
Detto questo, la presenza di un controesempio indica l'esistenza di almeno uno stato da dividere, perché quello stato contiene almeno una stringha di accesso non Nerode congruente con il proprio prefisso caratteristico. A partire dal controesempio si dovrà quindi ricavare uno o più nuovi stati da posizionare sulla frontiera dello span-tree, in modo da non modificare alcuna s-transizione. Quindi è necessario trovare la corretta d-transizione che verrà "*promossa*" a s-transizione e che punterà al nuovo stato e a partire dalla quale verrà ricavato il nuovo prefisso caratteristico. Dal contresempio si otterrà quindi il suffisso che separa la nuova classe da quella vecchia e questo verrà aggiunto al discrimination-tree insieme alla nuova classe. Infine si dovranno chiudere le transizioni uscenti dal nuovo stato facendo le necessarie membership query.
Dobbiamo però capire come ottenere tutto ciò da un controesempio. Questo risulta più chiaro se si segue un esempio passo per passo. Supponiamo che il linguaggio target definito sull'alfabeto $\Sigma = \{a, b\}$ sia $\boldsymbol{L}$ e che riconosca solo le parole con un numero dispari di "a" e un numero dispari di "b", il cui DFA canonico è mostrato in figura

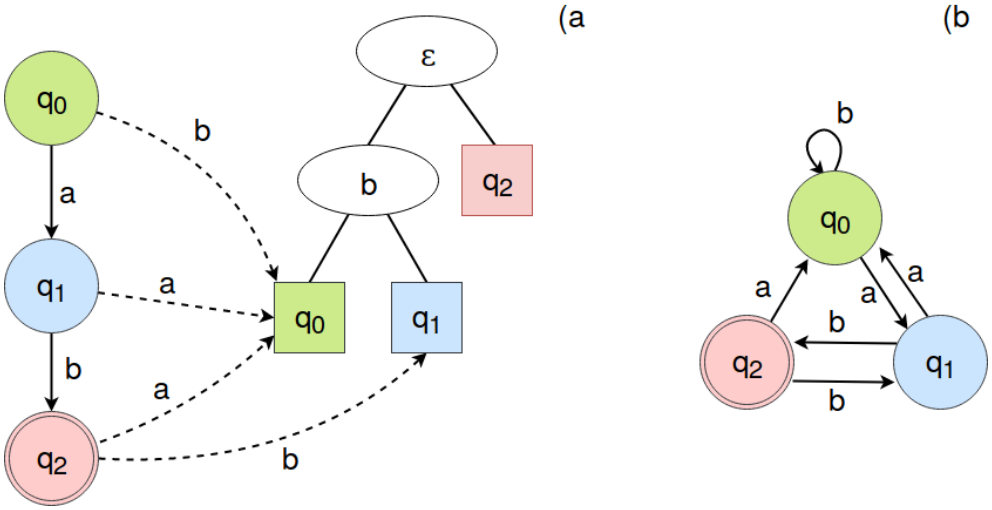

Figura 2.24: BBC dell'observation pack chiuso e deterministico e relativo DFA ipotesi.

2.18. In figura 2.24.a abbiamo le strutture dati dell'Observation Pack e in 2.24.b il relativo DFA ipotesi. L'algoritmo ha già fatto alcune equivalence query ma ancora non è terminato data la non equivalenza di DFA target e DFA ipotesi. Immaginiamo in particolare che l'oracolo decida di restituire il controesempio "aaabaa", accettato dal linguaggio $\boldsymbol{L}$ ma non dall'ipotesi. Il controesempio va scomposto in una forma p - t - s , con p prefisso composto da zero o più simboli, t transizione associata ad un singolo simbolo e s suffisso composto da 1 o più simboli. Chiamiamo con $\mathbf{q}_\mathrm{i}$ lo stato del DFA ipotesi relativo alla stringa di accesso p e $\mathbf{q}_\mathrm{j}$ lo stato relativo alla stringa $p \cdot t$. Il t -successore di $\mathbf{q}_\mathrm{i}$ sarà quindi $\mathbf{q}_\mathrm{j}$. Chiamiamo inoltre c_i e c_j i prefissi caratteristici rispettivamente di $\mathbf{q}_\mathrm{i}$ e $\mathbf{q}_\mathrm{j}$. Perché la scomposizione sia valida (non tutte lo sono), deve valere:

$$\lambda_{\boldsymbol{L}}(c_j \cdot s) \neq \lambda_{\boldsymbol{L}}(c_i \cdot t \cdot s) \quad (1)$$

che implica:

$$c_j \ncong_{\boldsymbol{L}} c_i \cdot t \quad (2).$$

Questo significa trovare lo stato $\mathbf{q}_\mathrm{j}$ per il quale esiste la stringa di accesso $c_i \cdot t$ non Nerode-congruente col suo prefisso caratteristico c_j . Infatti sappiamo dalla (1) che per le due stringhe esiste il separatore s . Lo stato $\mathbf{q}_\mathrm{j}$ non dovrà allora avere più $t \cdot s$ come stringa di accesso, e questo verrà ottenuto facendo sì che la transizione t dello stato $\mathbf{q}_\mathrm{i}$ non punti più a $\mathbf{q}_\mathrm{j}$ ma ad un nuovo stato $\mathbf{q}_\mathrm{k}$ al quale verrà assegnato il prefisso caratteristico $c_k = c_i \cdot t$. Nel nostro esempio procederemo per tentativi, scorrendo tutte le possibili scomposizioni fino a trovarne una che soddisfi la (1). Esistono approcci più efficienti ma useremo comunque questo per la sua semplicità e chiarezza. Partiamo dalla prima scomposizione possibile per il nostro controesempio "aaabaa": ε-a-aabaa. In questo caso

$$p = \varepsilon,\ t = a,\ s = aabaa,$$

$$\mathbf{q}_\mathrm{i} = \mathbf{q}_0,\ c_i = \varepsilon,$$

$$\mathbf{q}_\mathrm{j} = \mathbf{q}_1,\ c_j = a.$$

Bisogna verificare se vale la 1, ma questo è impossibile dato che le due parole ottenute sono uguali. Non può mai verificarsi infatti la seguente:

$$\lambda_{\boldsymbol{L}}(aaabaa) \neq \lambda_{\boldsymbol{L}}(aaabaa)$$

La seconda scomposizione invece è a-a-abaa da cui:

$$p = a,\ t = a,\ s = abaa,$$

$$\mathbf{q_i} = \mathbf{q_1},\ c_i = a,$$

$$\mathbf{q_j} = \mathbf{q_0},\ c_j = \varepsilon.$$

Bisogna dunque verificare se:

$$\lambda_{\boldsymbol{L}}(abaa) \neq \lambda_{\boldsymbol{L}}(aaabaa)$$

Ma dato che entrambe le parole "abaa" e "aaabaa" sono riconosciute da $\boldsymbol{L}$ (entrambe hanno un numero dispari di "a" e di "b") la disuguaglianza non è vera. Lo stesso vale per la scomposizione aa-a-baa. Invece per la scomposizione aaab-a-a abbiamo:

$$p = aaab,\ t = a,\ s = a,$$

$$\mathbf{q_i} = \mathbf{q_2},\ c_i = ab,$$

$$\mathbf{q_j} = \mathbf{q_0},\ c_j = \varepsilon.$$

In questo caso

$$\lambda_{\boldsymbol{L}}(c_j \cdot s) \neq \lambda_{\boldsymbol{L}}(c_i \cdot t \cdot s)$$

corrisponde a:

$$\lambda_{\boldsymbol{L}}(a) \neq \lambda_{\boldsymbol{L}}(abaa)$$

che dà risultato positivo.

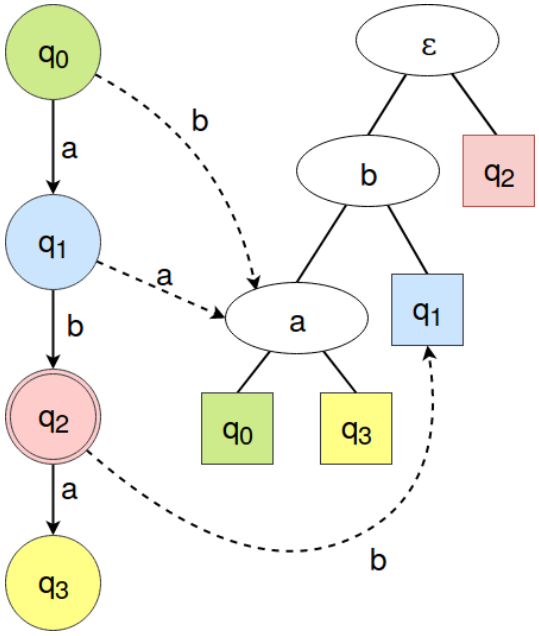

Figura 2.25: BBC dell'Observation Pack cui si è giunti dopo il controesempio "aaabaa". Necessariamente non chiuso né deterministico.

Abbiamo trovato quindi per lo stato $\mathbf{q_0}$ la stringa di accesso $c_i \cdot t = aba$ che non è Nerode-congruente con il prefisso caratteristico di $\mathbf{q_0}$ ε, perché esiste il separatore "a" per i due prefissi. Quindi aba e ε devono appartenere a classi diverse ed essere stringhe di accesso di stati differenti. Ma "ε" è il prefisso caratteristico di $\mathbf{q_0}$ quindi sarà la stringa aba a non dover portare a $\mathbf{q_0}$ ma ad un nuovo stato diverso da tutti

gli altri. Nello span-tree lo stato $\mathbf{q}_2$ non dovrà quindi puntare alla $\mathbf{q}_0$ per la d-transizione "a" come faceva in figura 2.24, ma ad un nuovo stato $\mathbf{q}_3$, il cui prefisso caratteristico sarà $c_3 = c_2 \cdot t = aba$. Per quanto riguarda il discrimination-tree invece avremo che la classe $\mathbf{q}_0$ viene suddivisa tramite il suffisso "a" nelle classi $\mathbf{q}_0$ e $\mathbf{q}_3$. Il risultato è mostrato in figura 2.25. Si noti che non è possibile ottenere immediatamente dopo questa operazione un nuovo DFA ipotesi: il BBC risulta infatti non chiuso per la mancanza di transizioni uscenti dallo stato $\mathbf{q}_3$ e non deterministico, per via delle d-transizioni che puntano ad un nodo interno del discrimination-tree. Sarà dunque necessario riportare il BBC ad uno stato chiuso e deterministico tramite una serie di membership query: "ba" e "aaa" per risolvere il non determinismo, e almeno "abaa" e "abab" per la non chiusura. Infine è bene sottolineare che questa procedura potrebbe portare ad un DFA ipotesi che ancora non riconosce correttamente il controesempio. Ciò dipende dal fatto che per riconoscere il controesempio dato è necessario aggiungere più di uno stato. In questo caso si continuerà a cercando altre scomposizioni del controesempio fino ad arrivare a riconoscerlo.

2.8 Costi

Il terzo algoritmo preso in esame è il **TTT**. Esso prende fortemente spunto dall'Observation Pack, ma presenta delle modifiche che portano ad alcuni sostanziali vantaggi. Per poter capire esattamente le motivazioni che hanno portato al suo sviluppo è necessario che vengano definiti dei costi relativi agli algoritmi di active learning. È evidente che un algoritmo di active learning sarà tanto migliore tanto più in fretta troverà il linguaggio target e quindi quante meno equivalence e membership query effettuerà per riuscirci. Quindi i primi due costi definiti per questi algoritmi sono il **costo in membership query** e il **costo in equivalence query**. In [12] alcune considerazioni portano alla definizione di un ulteriore costo. In particolare il costo in membership query sottointende che ogni membership query abbia lo stesso onere computazionale. In gererale però non è così. Infatti come viene sottolineato in [12], il costo computazionale di una membership query è tipicamente lineare con la lunghezza della parola da controllare e anche se esistono casi pratici in cui le membership query hanno effettivamente un costo costante rispetto alla loro lunghezza, la valutazione di un algoritmo dovrebbe essere indipendente dalla sua realizzazione pratica. Quindi il costo in membership query viene mantenuto e si introduce il **costo in simboli**, corrispondente al numero cumulativo di simboli delle parole per le quali vengono effettuate membership query durante l'intero l'algoritmo. Come vedremo questo costo viene ridotto dal TTT mantenendo il discrimination-tree suffix-closed e riducendo di conseguenza la lunghezza delle parole per cui verranno effettuate membership query. Di seguito sono riportati da [12] i costi asintotici degli algoritmi per quanto riguarda membership query e numero di simboli in funzione della dimensione del DFA canonico del linguaggio target n, della dimensione dell'alfabeto k e della lunghezza del controesempio di lunghezza massima m fortnito dall'oracolo.

Algoritmo	costo in membership query	costo in simboli
L*	$O(kn^2m)$	$O(kn^2m^2)$
Observation Pack	$O(kn^2 + n\log m)$	$O(kn^2m + nm\log m)$
TTT	$O(kn^2 + n\log m)$	$O(kn^2m + nm\log m)$

La tabella mostra come Observation Pack e TTT siano migliori di L* per quanto riguarda i due costi visti. Inoltre il costo in membership query è vicino al limite

minimo teorico di $\Omega(kn^2)$ dimostrato in [14]. Vediamo altresì che il TTT non peggiora per questi costi rispetto all'Observation Pack e migliora notevolmente nel caso in cui l'insegnante sia non benevolo, ovvero fornisca controesempi di lunghezza arbitraria. Il discorso è opposto per quanto riguarda le equivalence query. Infatti mentre per L* questo era una frazione di n , per Observation Pack e TTT è dello stesso ordine di grandezza. In pratica possiamo vedere L* come l'algoritmo più *zelante*, il quale effettua molti più controlli per evitare errori. Ciò si traduce in un maggior numero di membership query tra una ipotesi e la successiva e in un incremento notevole del numero di stati dell'ipotesi. Al contrario la filosofia di Observation Pack e TTT porta ad eseguire il minor numero possibile di membership query, facendo sì che l'ipotesi cresca lentamente portando quindi il numero di equivalence query vicino al limite massimo n . Purtroppo non è possibile abbassare uno dei due costi senza far aumentare il secondo. Da un punto di vista pratico sarebbe interessante avere una misura del costo computazionale basata su tutti questi paramentri, ma questa non può esistere data la natura astratta dell'oracolo e di conseguenza la grande variabilità del costo di singole membership ed equivalence query. Ci si potrebbe aspettare un peggioramento delle prestazioni da parte di Observation Pack e TTT, dato che quale che sia l'oracolo, l'equivalence query risulta molto più onerosa della membership query. Valutiamo però due casi limite:

1. L'equivalence query è simulata da un numero arbitrariamente grande di membership query effettuate su un insieme di parole fisso. Il suo costo, per quanto molto maggiore di una singola membership query è costante e lo indicheremo con C .

2. L'oracolo è un DFA, e l'equivalence query viene eseguita tramite l'algoritmo del *table filling*.

Nel primo caso avremo che il costo complessivo tra equivalence e membership query di L* sarà $O(EC + kn^2m)$, dove con E indichiamo il numero di membership query effettuate da L*. Per il TTT invece sarà $O(EC + kn^2 + n \log m)$, dove chiaramente E avrà un valore diverso. Considerando il caso più favorevole per L* diciamo che questo dovrà effettuare una sola equivalence query, mentre il contrario vale per il TTT che dovrà eseguire esattamente n equivalence query, partendo da una ipotesi con un solo stato e facendola crescere di uno stato alla volta. Otteniamo quindi per L* un costo computazionale di $O(C + kn^2m)$ che viene ridotto dato che C è costante, a $O(kn^2m)$. Il costo computazionale del TTT diventa invece

$$O(nC + kn^2 + n \log m) = O(n(C + kn + \log m)) = O(kn^2 + n \log m)$$

sempre considerando C trascurabile.
Nel secondo caso invece abbiamo che l'equivalence query viene effettuata tramite l'algoritmo del table filling tra DFA ipotesi e DFA target. Il costo computazionale di questo algoritmo è direttamente proporzionale al prodotto del numero di stati dei due DFA. Quindi in entrambi i casi, l'onere computazionale sarà dovuto principalmente all'equivalence query finale, comune ad entrambi gli algoritmi, in cui DFA ipotesi e target avranno lo stesso numero di stati. I costi asintotici per le equivalence query saranno quindi uguali e proporzionali ad n^2 e di nuovo avremo che il TTT presenta un minor costo rispetto ad L* dato che la discriminante sarà data dai costi in membership query. Sottolineamo di nuovo come queste considerazioni non siano di natura generale, ma presentino casi abbastanza realistici, in cui il costo delle equivalence query è costante o lineare nella dimensione dell'ipotesi. Infine è di particolare interesse il fatto che questi algoritmi giungano al linguaggio target se questo è regolare, ma trovano diverse approssimazioni del linguaggio nel caso non sia regolare. La scelta dell'algoritmo quindi non

dipende soltanto dal costo ma anche da quanto fedeli ci vogliamo mantenere all'oracolo e da quanto invece vogliamo arrivare ad un linguaggio più generale e con meno stati.
Un ulteriore costo degli algoritmi di AL è quello in termini di memoria, portato all'attenzione in [15] e [16] dove viene illustrato l'algoritmo DHC, competitivo nei casi in cui la memoria sia il fattore critico. Questo vantaggio è ottenuto al costo di una forte perdita di informazioni che porta a dover ripetere più volte query all'oracolo. Al contrario il TTT riesce a minimizzare l'uso di memoria senza adoperare query ridondanti, come dimostrato in [12].

2.9 TTT

Come anticipato il terzo algoritmo di active learning preso in esame è il **TTT**. Descritto in [12], esso prende fortemente spunto dall'Observation Pack, tanto che lo si può considerare una sua evoluzione. Il suo stesso nome deriva dalle strutture dati che adopera, due delle quali prese direttamente dall'Observation Pack: lo span-**T**ree e il discrimination-**T**ree mentre la terza struttura, il **T**rie è introdotta dall'algoritmo. Nella sezione precedente sono stati descritti i vantaggi dell'algoritmo rispetto al predecessore ovvero la riduzione della lunghezza del controesempio e il costo ottimale in termini di memoria. In questa sezione vedremo come il TTT ottiene questi vantaggi analizzando nel dettaglio una modifica nell'algoritmo che permette l'adozione della terza struttura dati. L'uso del trie per quanto utile nella riduzione del costo in termini di spazio non produce alcun effettivo cambiamento nell'algoritmo e dunque per esso verrà data una rapida descrizione.

2.9.1 Nodi temporanei e nodi finali

Come l'Observation Pack anche il TTT fa uso del discrimination-tree per la classificazione dei suoi prefissi corti. In particolare ogni suo nodo interno del discrimination-tree contiene un suffisso che fa da discriminatore mentre ogni foglia rappresenta una classe del Black Box Classifier (BBC). Ad ogni prefisso viene concatenato il suffisso dei nodi che attraversa e la parola risultante viene valutata tramite una equivalence query per sapere se si dovrà continuare la discesa lungo il figlio destro o sinistro del nodo attuale. Il procedimento viene ripetuto fino a raggiungere un nodo foglia che corrisponderà alla classe cui viene associato, almeno temporaneamente il prefisso.
Nell'Observation Pack l'unica modifica che può essere fatta al discrimination-tree è aggiungere nodi alla sua periferia. Quindi una volta aggiunto, un nodo interno non potrà mai essere eliminato o modificato. Questi suffissi risultano dalla scomposizione del controesempio in forma p - t - s e dunque dipendono dai controesempi restituiti. Nel caso in cui l'oracolo non sia benevolo questi potrebbero essere inutilmente lunghi. La classificazione del prefissi, passando per questi nodi farebbe allora lievitare il costo in simboli.
Per risolvere questo problema nel TTT i nodi vengono distinti in **temporanei** e **finali**. I suffissi dei nodi temporanei sono quelli ottenuti dalla scomposizione dei controesempi e verranno eventualmente sostituiti tramite **finalizzazione** in nodi finali. I suffissi dei nodi finali formano un insieme suffix-closed e questo fa sì che col tempo crescano in maniera al più lineare e permette come vedremo, di adoperare il trie. Inoltre la finalizzazione di un nodo temporaneo potrà essere effettuata solo se il genitore è finale. Questo, insieme al fatto che il nodo radice "ε" del discrimination-tree è finale, ci assicura che partendo dal nodo radice incontreremo solo nodi finali, fino ad arrivare o ad un nodo temporaneo o ad una foglia. A partire dal primo nodo temporaneo non verranno

incontrati nodi finali ma solo altri nodi temporanei fino a raggiungere una foglia. Ciò

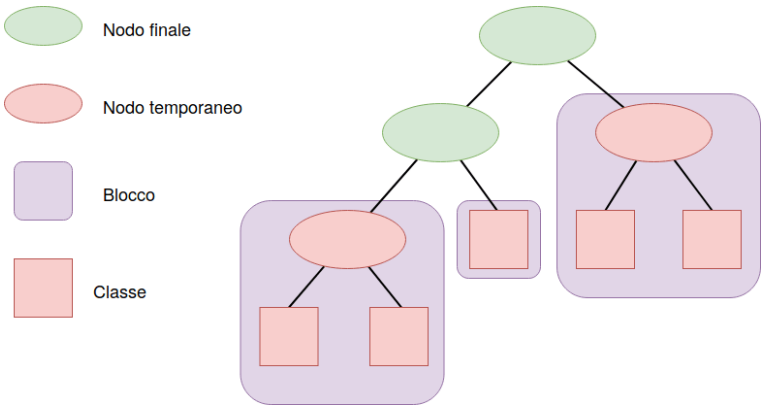

Figura 2.26: Discrimination-tree del TTT. Ogni sottoalbero contente nodi temporanei e foglie appartiene ad un blocco.

viene mostrato in figura 2.26 dove in particolare sono evidenziati i sottoalberi senza alcun nodo finale, radici comprese. Chiamiamo questi sottoalberi **blocchi**. Possiamo vedere i blocchi come foglie dell'*albero finale* o come *macro-classi*. Infatti ogni classe appartiene ad un blocco, e ogni blocco contiene una (nel qual caso diremo che il blocco è minimo) o più classi. Al momento di determinare dove punti una d-transizione scenderemo per il discrimination-tree fermandoci, il più delle volte, non appena incontreremo un blocco. In questo modo eviteremo di effettuare query inutilmente lunghe. Una d-transizione che punti alla radice di un blocco non minimo è però causa di non determinismo perchè quando una d-transizione punta ad un nodo è come se puntasse a tutte le classi che sono sue discendenti. Tramite finalizzazione però ridurremo la dimensione dei blocchi fino a minimizzarli, eliminando il non determinismo.

2.9.2 Finalizzazione

Il processo di finalizzazione riguarda blocchi con due o più foglie. Il suffisso del nodo radice di un blocco separa le classi corrispondenti alle sue foglie a seconda se esse siano suoi discendenti destri o sinistri. Se trovassimo un altro suffisso, più corto, che

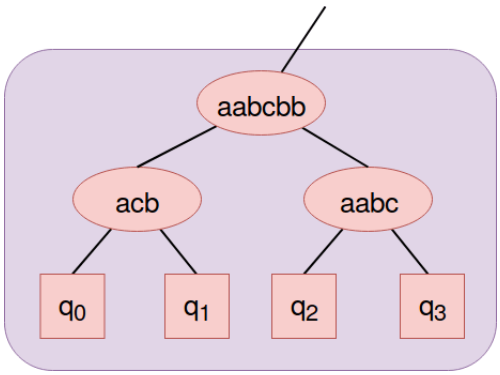

Figura 2.27: Blocco prima della finalizzazione del nodo radice

separi a sua volta le classi lo potremmo sostituire a quello del nodo radice rendendo finale quest'ultimo. Questa operazione porta a dividere il blocco in due sottoblocchi più piccoli. In figura 2.27 abbiamo un blocco che contiene le classi da $\mathbf{q}_0$ a $\mathbf{q}_3$. La radice del blocco "aabcbb" contiene un suffisso molto lungo che separa le classi $\mathbf{q}_0$ e $\mathbf{q}_1$ (figli sinistri) da $\mathbf{q}_2$ e $\mathbf{q}_3$ (figli destri). Supponiamo di aver trovato in qualche

modo il suffisso "aa" che separa a sua volta le classi, non necessariamente alla stessa maniera di "aabcbb". Diciamo ad esempio che $\mathbf{q}_0$ e $\mathbf{q}_2$ vengano classificati come figli sinistri, mentre $\mathbf{q}_1$ e $\mathbf{q}_3$ come figli destri. Possiamo allora finalizzare il nodo

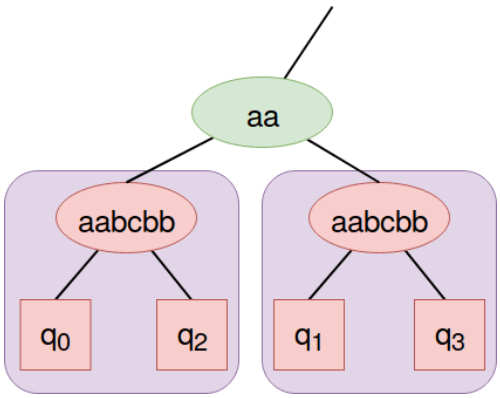

Figura 2.28: Blocco dopo la finalizzazione del nodo radice

"aabcbb" sostituendo il suo suffisso con "aa" e generando due nuovi blocchi i quali conterranno rispettivamente le classi ($\mathbf{q}_0$ e $\mathbf{q}_2$) e ($\mathbf{q}_1$ e $\mathbf{q}_3$). I sottoalberi relativi ai due nuovi blocchi sono estratti dal sottoalbero del blocco originale dal quale vengono prelevati i nodi necessari per separare le foglie suddivise tra i due sottoblocchi. In questo caso entrambi i sottoalberi "ereditano" il nodo radice "aabcbb" necessario a separare le proprie foglie come si vede in figura 2.28. Si noti come non sia detto che la struttura venga mantenuta. Supponiamo ad esempio di aver trovato il suffisso "bc" che separa le classi in $\{\ \mathbf{q}_0,\ \mathbf{q}_1,\ \mathbf{q}_2\}$ e $\{\ \mathbf{q}_3\}$. Otteremmo in questo caso la situazione in figura 2.29. Infine è bene ricordare che i nuovi suffissi non hanno solo il vantaggio di

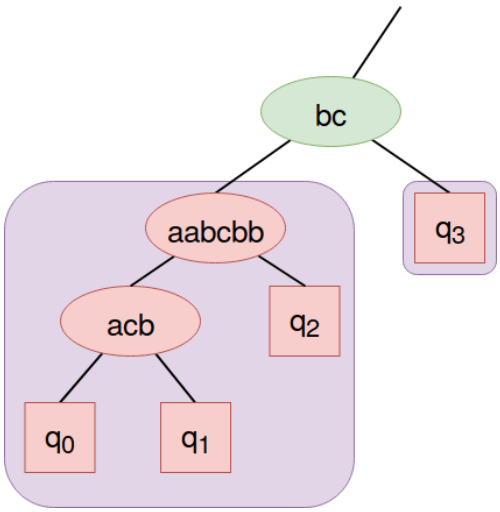

Figura 2.29: Altra possibile finalizzazione

crescere lentamente ma aggiunti all'insieme dei suffissi finali, mantengono la proprietà di suffix-closedness di tale insieme.

2.9.3 Due esempi

Abbiamo detto in che consiste la finalizzazione ma non abbiamo ancora visto come trovare i nuovi suffissi. Come già detto possiamo considerare i blocchi come macro-classi. Questo ci permette di considerare la proprietà di determinismo anche per i

blocchi. L'idea è la stessa del determinismo per le classi di un BBC: per ogni simbolo a dell'alfabeto gli a-successori dei prefissi corti di una classe c_i devono appartenere tutti alla stessa classe c_j. Nel nostro caso per ogni simbolo a dell'alfabeto tutti gli a-successori delle classi di un blocco B_i devono appartenere allo stesso blocco B_j. Se così non fosse, avremmo trovato due classi del blocco che sappiamo essere separate dal discriminatore $s_j = a \cdot s_i$ con s_i discriminatore dei due diversi blocchi puntati, appartenente al più profondo antenato comune ai due blocchi. Quindi avremmo trovato un suffisso s_j ottenuto concatenando un singolo simbolo al suffisso s_i relativo ad un nodo finale in quanto antenato comune a due blocchi diversi. Per questo motivo s_j risulta essere un suffisso valido con il quale finalizzare il nodo radice del nostro blocco. Quindi possiamo procedere a dividere i blocchi del BBC finchè continuiamo a trovarne di non deterministici. Si potrebbe però anche arrivare al punto in cui tutti i blocchi sono deterministici ma non tutti sono già minimi. In questo caso si dovrà necessariamente effettuare una o più membership query relative a nodi interni ad un blocco e quindi temporanei. Vediamo adesso i due casi descritti tramite i seguenti due esempi.

esempio 1

Supponiamo che il linguaggio target $\boldsymbol{L}$ sia di nuovo il linguaggio definito su $\Sigma = \{a, b\}$ delle parole con un numero dispari di "a" e un numero dispari di "b". La costruzione della prima ipotesi procede come per l'Observation Pack. Vengono effettuate le membership query relative alle parole "ε", "a" e "b" e dando tutte esito negativo, portano alle strutture in figura 2.30. L'ipotesi in figura non è corretta e i controesempi più corti

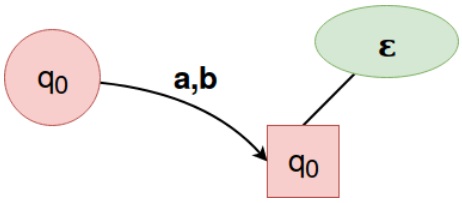

Figura 2.30: Prima ipotesi

sono "ab" e "ba". Poniamoci però nel caso in cui l'oracolo non sia benevolo, e restituisca il controesempio valido ma inutilmente lungo "abaaaa". La prima scomposizione valida è ε-a-baaaa che ci porta alla situazione in figura 2.31 in cui il BBC è non chiuso e non deterministico. Come già detto il non determismo non va risolto immediatamente per

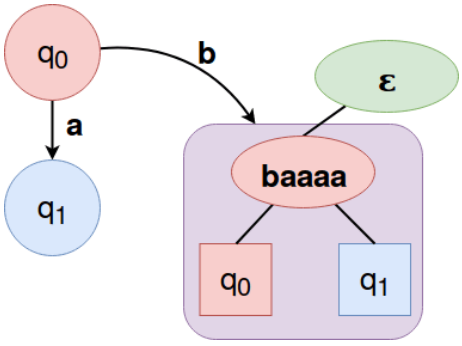

Figura 2.31

cui la transizione "b" di $\mathbf{q_0}$ continuerà a puntare alla radice del blocco. Si procederà invece a chiudere le transizioni dello stato $\mathbf{q_1}$. Per la transizione "a" verrà effettuata la membership query relativa alla parola "aa" ottenuta dalla concatenazione di $c_1 = a$

(prefisso caratteristico di $\mathbf{q}_1$), del simbolo a relativo alla transizione presa in esame e di ε suffisso relativo al nodo radice (finale) del discrimination-tree. Dato che questa membership query dà risultato negativo ("aa" non appartiene al linguaggio), si scenderà per il ramo sinistro arrivando al blocco. Per la transizione "b" la membership query sarà invece relativa alla parola "ab". In questo caso la membership query dà risultato positivo, dunque si scenderà lungo il ramo destro del nodo radice. Dato che il nodo radice non ha figlio destro vengono creati la nuova classe e il nuovo stato $\mathbf{q}_2$ quali b-successori di $\mathbf{q}_1$ e figli destri del nodo radice. Chiudiamo le transizioni di questo nuovo stato effettuando le relative membership query ("aba" e "abb") che danno entrambe esito negativo. Entrambe le transizioni di $\mathbf{q}_2$ punteranno allora al blocco sinistro. Il risultato è in figura 2.32. A questo punto non possiamo creare il DFA

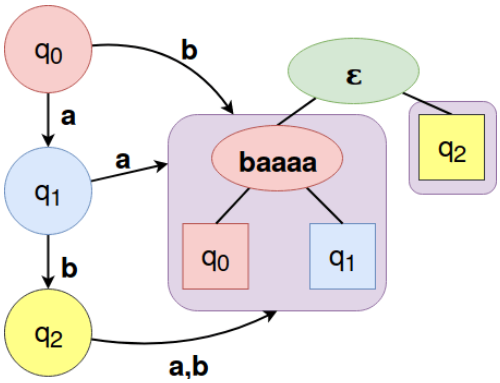

Figura 2.32: Esempio 1: il blocco $B_0 = \{\mathbf{q}_0, \mathbf{q}_1\}$ risulta non deterministico per via delle transizioni b delle sue due classi.

ipotesi in quanto il BBC è ancora non deterministico ma per renderlo tale dovremmo fare delle membership query relative ai nodi temporanei ("baaaa") data la presenza di blocchi non minimi. Ci rendiamo conto però che il blocco $B_0 = \{\mathbf{q}_0, \mathbf{q}_1\}$ è "non deterministico". Infatti per la transizione "b" le sue classi puntano a blocchi differenti. Questo ci permette di individuare un suffisso con cui finalizzare il nodo radice del blocco. In particolare questo suffisso manterrà l'insieme dei suffissi finali suffix-closed e dato che tale insieme è attualmente composto dal solo suffisso ε, il nuovo suffisso sarà al massimo di lunghezza 1.
Per trovare il suffisso dobbiamo vedere quali siano i due blocchi raggiunti e da quale discriminatore finale siano separati. La d-transizione "b" della classe $\mathbf{q}_0$ punta a B_0 . Invece la s-transizione "b" dello stato $\mathbf{q}_1$ punta allo stato $\mathbf{q}_2$ e di conseguenza al blocco $B1 = \{\mathbf{q}_2\}$ che lo contiene. I due blocchi B_0 e B_1 sono separati dal suffisso ε in quanto appartenente al più profondo antenato comune ai due blocchi. In altre parole anche se ancora non sappiamo a quale classe punterà la transizione "b" dello stato $\mathbf{q}_0$, siamo certi che il suffisso ε separerà quella classe dal b-successore di $\mathbf{q}_1$. Di conseguenza le due classi $\mathbf{q}_0$ e $\mathbf{q}_1$ sono separate dal suffisso $b \cdot \varepsilon = b$. Possiamo allora finalizzare il nodo radice del blocco B_0 , sostituendo al suo, il suffisso b e ottenendo il risultato in figura 2.33. A questo punto dato che tutti i blocchi presenti sono minimi è possibile procedere alla prossima ipotesi.

esempio 2

Consideriamo adesso il linguaggio $\boldsymbol{L}$ definito su {a, b} la cui unica parola è "abab". Le prime membership query porteranno nuovamente al BBC in figura 2.30. In questo caso l'unico controesempio possibile è proprio "abab" la cui prima scomposizione valida è

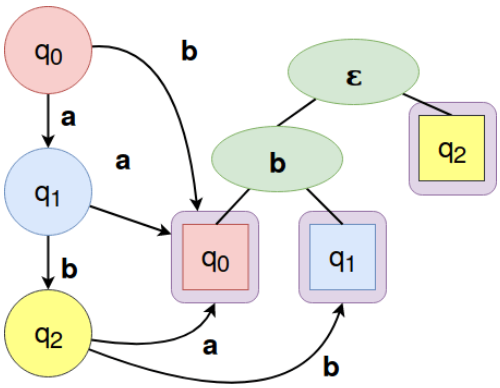

Figura 2.33: Esempio 1: il blocco è stato diviso e i suoi nodi finalizzati.

ε-*a*-*bab*. Questo porta alla creazione del nuovo stato $\mathbf{q}_1$, e modifica il discrimination-tree come mostrato in figura 2.34 dove sono anche state chiuse le transizioni di $\mathbf{q}_1$. Abbiamo quindi che

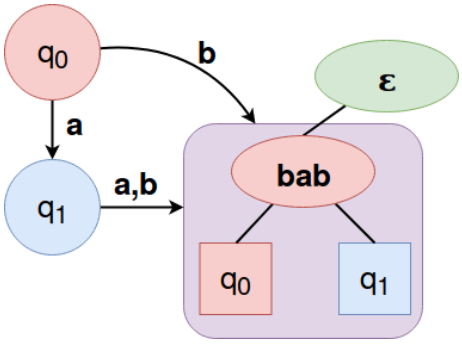

Figura 2.34: Esempio 2: l'unico blocco non può essere minimizzato.

1. Non può essere generata una ipotesi data la presenza di un blocco non minimo.
2. Non è possibile dividere il blocco perché non presenta non determinismo (essendo l'unico blocco tutte le transizioni devono puntare ad esso).
3. Il controesempio ancora non è stato riconosciuto

Purtoppo i primi due problemi non possono essere risolti e per riconoscere il controesempio sarà necessario fare membership query relative a nodi temporanei che invece avremmo voluto evitare. Quindi si continua con la scomposizione del controesempio. La seconda scomposizione possibile è *a*-*b*-*ab*, che comporta il dover valutare la seguente disuguaglianza:

$$\lambda_{\boldsymbol{L}}(c_i \cdot t \cdot ab) \neq \lambda_{\boldsymbol{L}}(c_j \cdot ab)$$

dove ricordiamo c_i è il prefisso caratteristico di $\mathbf{q}_\mathrm{i}$, stato raggiunto dal prefisso a e c_j è il prefisso caratteristico dello stato $\mathbf{q}_\mathrm{j}$ raggiunto invece dalla concatenazione di prefisso e transizione: $a \cdot b = ab$. Dalla figura 2.34 vediamo che lo stato raggiunto dal prefisso a è $\mathbf{q}_1$ e dunque c_i corrisponde ad a. Il prefisso ab invece porta ad un blocco e non ad una classe. Quindi non possiamo sapere a che prefisso corrisponda c_j tra quelli del blocco. Dobbiamo scoprire se la transizione "b" dello stato $\mathbf{q}_1$ punti al figlio destro o sinistro del nodo *bab*. Per farlo dobbiamo effettuare la membership query relativa alla parola *abbab* ottenuta concatenando il prefisso caratteristico di $\mathbf{q}_1$, a, alla transizione b e al nodo *bab*. La membership query dà esito negativo per cui la

classe puntata sarà $\mathbf{q}_0$. Come accadeva per l'Observation Pack la classe viene divisa a partire dal separatore dato dal nuovo suffisso trovato *ab*, ottenendo la nuova classe $\mathbf{q}_2$ come mostrato in figura 2.35 nella quale inoltre sono state chiuse le transizioni relative al nuovo stato. Purtroppo ancora non possiamo dividere il blocco e ancora

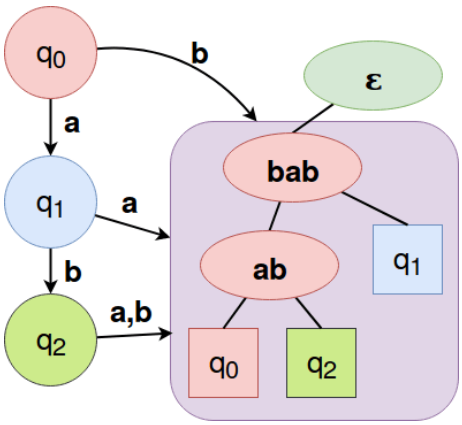

Figura 2.35: Esempio 2: il blocco continua a crescere per mezzo di query non minime.

non riconosciamo il controesempio. In maniera analoga continuiamo con la successiva scomposizione del controesempio *ab-a-b*. Omettendo i singoli passaggi arriviamo alla situazione in figura 2.36. A questo punto riconosciamo il controesempio dato e inoltre

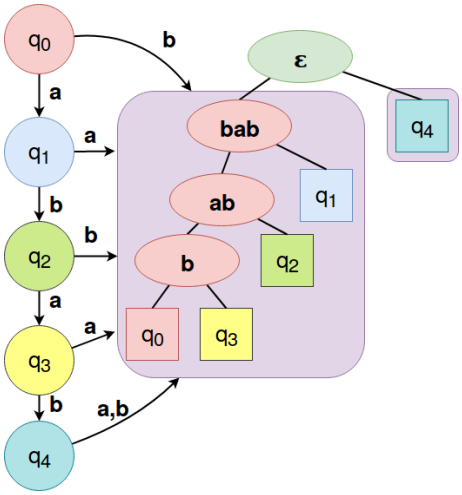

Figura 2.36: Esempio 2: la gestione del controesempio ha portato alla creazione di un nuovo blocco e al non determinismo del primo.

possiamo finalmente dividere il blocco $B_0 = \{\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3\}$. Infatti le classi $\mathbf{q}_0$, $\mathbf{q}_1$ e $\mathbf{q}_2$ tramite *b*-transizioni puntano al blocco B_0, mentre la classe $\mathbf{q}_3$ punta per lo stesso simbolo al blocco $B_1 = \{\mathbf{q}_4\}$. I due blocchi sono separati dal suffisso ε, per cui il suffisso usato per la finalizzazione sarà $b \cdot \varepsilon = b$. Questo porta alla divisione in figura 2.37. Il blocco può essere ulteriormente diviso, infatti le classi $\mathbf{q}_0$ e $\mathbf{q}_1$ per la transizione a puntano ad un blocco diverso da quello puntato da $\mathbf{q}_2$ per la stessa transizione. In questo caso il separatore dei blocchi è b, dunque il nuovo suffisso sarà $a \cdot b = ab$. Il procedimento può essere ripetuto fino ad ottenere 5 blocchi minimi eliminando il non determinismo e permettendo la costruzione dell'ipotesi successiva.

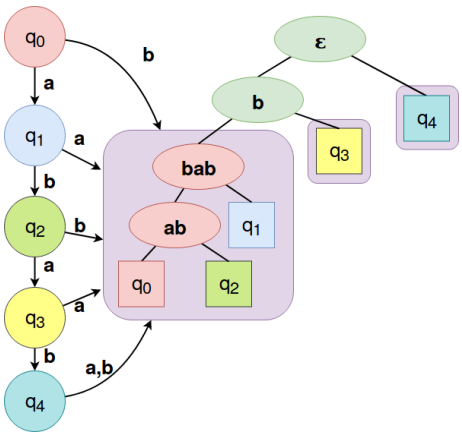

Figura 2.37: Esempio 2: la gestione del controesempio ha portato alla creazione di un nuovo blocco e al non determinismo del primo.

2.9.4 Trie

Oltre a ridurre il numero di membership query e di *accorciare* i controesempi forniti da un oracolo non benevolo, il TTT ha anche il vantaggio di gestire l'occupazione della memoria in maniera ottimale. Questo dipende dalla proprietà di suffix-closedness dei suffissi relativi a nodi finali e all'uso dei **trie**, strutture dati che possono approfittare di tale proprietà dei dati per ridurre l'occupazione di spazio in memoria. I trie definiti in [17], sono alberi nei quali non è mantenuta esplicitamente alcuna informazione, in cui ogni arco diretto sempre dal nodo figlio a quello padre, è associato ad un simbolo di Σ. Ogni nodo corrisponde a un elemento dell'insieme, ma non lo conserva al proprio interno evitando una complessità quadratica. Per trovare il suffisso relativo ad un nodo bisogna risalire fino al nodo radice, mantenendo tutti i simboli relativi ai nodi incontrati. In questo modo si possono rappresentare insiemi suffix-closed (o prefix-closed) senza ripetere simboli relativi a suffissi comuni a più elementi. Ciò è rappresentato in figura 2.38, nella quale sono stati inclusi i suffissi appartenenti ai nodi per maggior chiarezza.

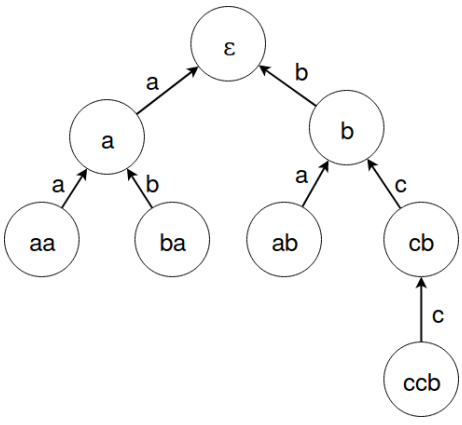

Figura 2.38: Trie relativo all'insieme suffix-closed $\{\varepsilon, a, b, aa, ba, ab, cb, ccb\}$

Quindi per una occupazione migliore dello spazio, la lettura dei suffissi dei nodi finali del discrimination-tree viene gestita tramite un trie. Questo non influisce in alcun modo sull'algoritmo in sè ma come dimostrato in [12], permette la gestione ottimale dello spazio.

Capitolo 3

La conoscenza al centro dell'oracolo: le ANN

Nel capitolo precedente è stato descritto l'Active Learning (AL) da un punto di vista teorico e matematico ed è stata fatta una panoramica di alcuni suoi algoritmi. L'attenzione è sempre stata posta sul learner e sulle sue strutture dati oltre che sull'algoritmo in sé. Al contempo non è stata data alcuna visione interna dell'oracolo né dal punto di vista implementativo né da quello algoritmico, dando per assunto che funzionasse correttamente. L'oracolo è sempre stato considerato da un punto di vista esterno, relativo alle due funzioni usate per interfacciarsi ad esso: la membership query e l'equivalence query. In questo capitolo l'attenzione sarà invece concentrata sui modelli alla base dell'oracolo adoperato in questa tesi. In particolare vedremo reti neurali feed-forward e RNN.
Le reti neurali artificiali nascono dall'idea del *percettrone* descritta in [10], la quale ha stimolato la ricerca, portando a modelli sempre più potenti che possiamo osservare oggi. L'idea alla base é prendere spunto dai neuroni biologici e imitarne il comportamento, mimando l'apprendimento del cervello umano. Quando si impara qualcosa di nuovo avvengono delle trasformazioni tra i collegamenti tra coppie di neuroni che possono essere rinforzati o indeboliti. Più sarà il tempo passato ad apprendere l'attività più i neuroni formeranno una rete in grado di *rispondere bene* a tale attività. Una rete neurale artificiale mima esattamente questo, modificando durante l'addestramento i pesi delle sinapsi che collegano i neuroni per meglio adattarsi ai dati da apprendere.

3.1 Categorie di ANN

Le reti neurali possono essere suddivise in macrocategorie a seconda di quali siano gli aspetti di interesse. Alcune di queste categorie sono il tipo di apprendimento, lo scopo per il quale si addestra la rete e l'output che restituisce e se la rete è statica o dinamica.

3.1.1 Tipo di apprendimento

Ogni ANN prima di poter essere adoperata richiede una fase di addestramento, del quale esistono tre principali tipologie: **supervisionato**, **non supervisionato** e **per rinforzo**.
L'apprendimento supervisionato è il più diretto: alla ANN viene fornito un dataset composto di campioni ed etichette. Ad ogni campione corrisponde una o più etichette (a seconda del numero di caratteristiche di interesse per l'applicazione) in modo che l'ANN possa direttamente accedervi. Per misurare quanto bene la rete generalizzi

e per evitare che si sovra-adatti ai dati, il dataset può essere suddiviso in trainset e testset. La rete quindi si addestra sul trainset con lo scopo di classificare correttamente quanti più campioni possibili, associando ognuno alla propria etichetta. Il testset viene invece usato per verificare la bontà del metodo in termini di accuratezza e sopratutto generalizzazione, mostrando come la rete risponda a dati su cui non è stata addestrata.
L'apprendimento non supervisionato non richiede che venga associata alcuna etichetta agli campioni forniti per l'addestramento. La rete autonomamente dovrà riorganizzarli e classificarli a partire da caratteristiche comuni. Questo tipo di apprendimento ha principalmente lo scopo di scovare relazioni nascoste tra i dati piuttosto che partire da relazioni evidenti date da etichette. Un tipo di ANN che si basa sull'apprendimento non supervisionato sono gli *auto-encoder* adoperati per il riconoscimento di immagini, i quali si dividono in una parte che codifica l'input con un minor numero di *features* a più alto livello, e un'altra che riottenga l'input a partire dalle features per verificare che la perdita di informazioni delle features non sia eccessiva.
L'apprendimento per rinforzo non si basa direttamente su esempi ma su *ricompense* date alla rete a seconda della sua risposta a eventi e mutazioni dell'ambiente. La rete si muoverà dunque cercando di adattarsi per guadagnare il maggior numero possibile di ricompense.
Il tipo di addestramento adoperato per le reti usate come nucleo dell'oracolo in questa tesi, è supervisionato.

3.1.2 Scopo della rete

In [11] vengono identificate 6 diverse categorie a seconda di quale sia l'utilizzo previsto dalla rete:

1. Identificazione di oggetti:
   ANN il cui scopo sia identificare correttamente gli oggetti presenti in una immagine. Questo è uno degli usi principali delle reti neurali, le quali danno ottimi risultati, nonostante la complessità del compito. Arrivare ad informazioni di alto livello a partire dai pixel di una immagine è infatti non banale. Esempi tipici sono le reti convoluzionali, composte di più strati oltre quello completamente connesso, tra cui lo strato convoluzionale che processa l'immagine riducendola di dimensione e che dà il nome alla rete.

2. Clusterizzazione:
   ANN che lavorano su dati grezzi e il cui scopo sia raggruppare i campioni in insiemi detti cluster. Lo scopo è quindi trovare una qualche legge nascosta che regoli l'appartenenza dei campioni ad uno dei cluster. L'apprendimento in questo caso è tipicamente non supervisionato e la classificazione dei campioni dipende unicamente da caratteristiche interne a questi ultimi.

3. Riconoscimento di pattern:
   ANN per l'identificazione di pattern a partire da conoscenza a priori o informazioni statistiche presenti nei pattern.

4. Regressione:
   ANN il cui compito sia stimare la relazione funzionale tra variabili dipendenti e variabili indipendenti, tipicamente per mezzo del valore atteso.

5. Generazione di comportamenti:
   comprende tutte le reti il cui scopo sia generare una serie di azioni in risposta

a degli stimoli presenti o passati. Queste reti sono adoperate nella teoria dei giochi, in cui non si ha il pieno controllo del sistema ma questo è condiviso con un avversario. Il problema in cui si incorre in questo caso è la dimensione dello spazio degli stati del sistema, che dipende ovviamente dal gioco. Non è possibile quindi apprendere la risposta migliore per ogni sequenza di mosse passate. Quello che si fa è invece usare l'apprendimento per rinforzo basandosi su ricompense e punizioni a seconda della bontà dello stato cui si giunge in seguito alle mosse della rete.

6. Acquisizione di conoscenza:
   ANN il cui compito sia il recupero di conoscenza, distinta in [18] in *concettuale* e *comportamentale*, la prima relativa a fatti generali ed oggettivi riguardanti il mondo percepito, e la seconda a reazioni condizionate da eventi o azioni mirate ad uno scopo.

La sesta e ultima categoria e in particolare quella concettuale, si riferisce ad un tipo di conoscenza tipica dell'essere umano strettamente legata al concetto di sequenza (per l'essere umano è più semplice imparare informazioni ordinate secondo una sequenza logica piuttosto che un insieme di dati senza alcun ordine). Quest'ultima sarà la categoria delle ANN che verranno prese in esame in questa tesi.

3.1.3 Dinamicità della rete

Con il termine **dinamicità** intendiamo come la rete possa cambiare dopo l'addestramento, sia in termini di struttura, sia in termini di pesi e output della rete. Una rete **statica** una volta addestrata non potrà più modificare i propri pesi, a meno di non riprendere una nuova *epoca* di apprendimento. Al contrario una rete **dinamica** avrà al suo interno sia valori fissati dall'addestramento, sia variabili modificate anche dagli input. Possiamo quindi dire che questa seconda categoria possiede una sorta di memoria interna, particolarmente utile nel nostro caso perché assimilabile alla "memoria" data dagli stati degli automi. Per questo motivo verrano usate anche reti con elementi dinamici quali le RNN.

Output della rete

Le reti vanno infine divise a seconda del tipo di output che restituiscono. L'output di una rete può consistere di un singolo scalare o di un vettore. Inoltre i singoli elementi dell'uscita possono essere numeri reali, interi o appartenere a qualche altro insieme ancora. Questi fattori dipendono da come l'uscita della rete debba essere interpretata: potremmo volere una rete in grado di individuare la presenza di alcuni tipi di oggetti in una immagine e progettare l'uscita con una dimensione pari al numero di oggetti, in cui l'i-esimo elemento dell'uscita è un numero compreso nell'intervallo [0, 1] e rappresenta la probabilità che l'i-esimo oggetto sia presente nell'immagine passata in input. Dato un insieme di elementi ordinato, un output composto di un solo intero potrebbe invece essere usato per predire l'elemento successivo di una sequenza: data la sequenza come input, l'output andrebbe interpretato come indice corrispondente all'elemento successivo. Infine una rete con un output composto di un solo scalare appartenente all'insieme $\{0, 1\}$ può essere utilizzata come classificatore binario del proprio input. Questo è esattamente il nostro caso, in quanto sarà necessario per ogni parola opportunamente trasformata in input della rete, andare ad indicare se essa appartiene o meno al linguaggio regolare approssimato dalla nostra rete.

3.2 Il neurone

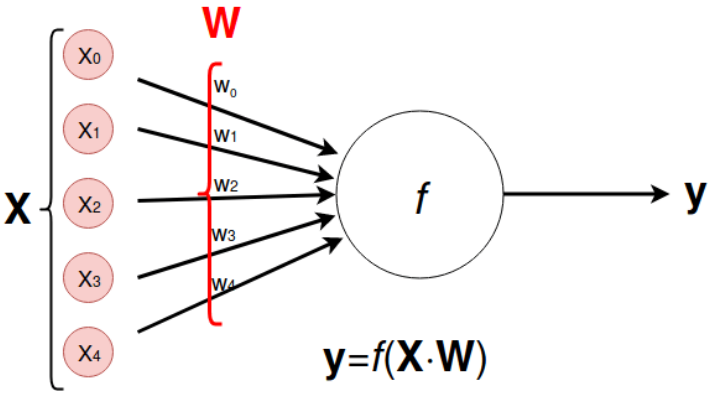

Figura 3.1: Neurone di una ANN

Il neurone è l'unità fondamentale di qualunque tipo di rete neurale ed è caratterizzato dal vettore W relativo ai pesi delle sinapsi in ingresso. Se chiamiamo X il vettore composto dagli output dei neuroni del livello precedente o **vettore degli ingressi** del neurone ed f la sua **funzione di attivazione** possiamo esprimere l'output del neurone dato il suo vettore degli ingressi tramite la seguente formula:

$$y = f(W^{\mathrm{T}} \cdot X)$$

dove con $(\cdot)$ si intende il prodotto scalare di 2 vettori della stessa dimensione.
Esistono diverse funzioni di attivazione alcune delle quali sono mostrate in figura 3.2, tra cui la funzione gradino, la cui discontinuità permette di catturare più fedelmente i campioni dei dataset anche se questi dovessero presentare una scarsa regolarità. Quest'ultima è molto adoperata con l'aggiunta di un parametro detto *soglia* o *bias*, di seguito indicato da α, sottratto al risultato di $W^{\mathrm{T}} \cdot X$. Quindi se il prodotto scalare di W e X supera la soglia, l'output del neurone sarà 1, 0 altrimenti. Una rete composta

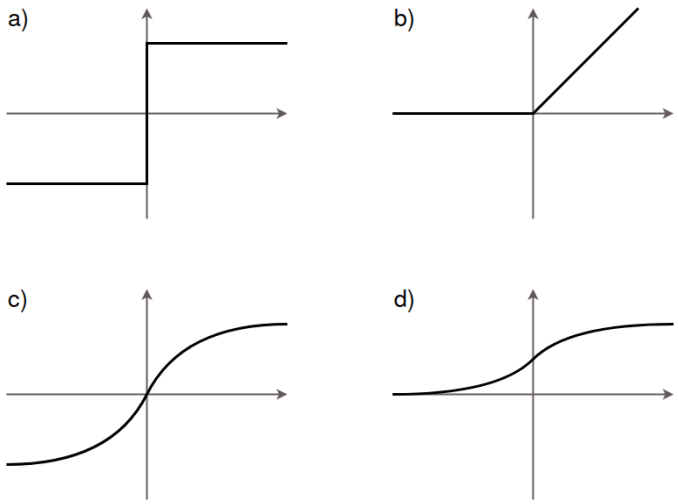

Figura 3.2: 4 possibili funzioni di attivazione:
a) funzione gradino b) relu (Rectifier Linear Unit)
c)funzione logistica d) tangente iperbolica

da un singolo neurone in cui i vettori X e W hanno dimensione n, con funzione di attivazione data dalla funzione soglia, è però molto limitata. Essa infatti sarà solo in grado

di dividere lo spazio n-dimesionale dei vettori di ingresso in 2 emispazi n-dimensionali. Se per chiarezza prendiamo in esame il caso in cui X e W siano bidimensionali, vediamo che questo corrisponde al tracciare una retta.
Infatti $f({W^{\mathrm{T}}}_2 \cdot X_2, \alpha) =$

$$1 \; se \; x_1 w_1 + x_2 w_2 > \alpha$$

$$0 \; altrimenti$$

che corrisponde all'equazione di un semipiano generato a partire da una retta come mostra la figura 3.3. Per questo motivo le funzioni rappresentabili dal singolo neurone

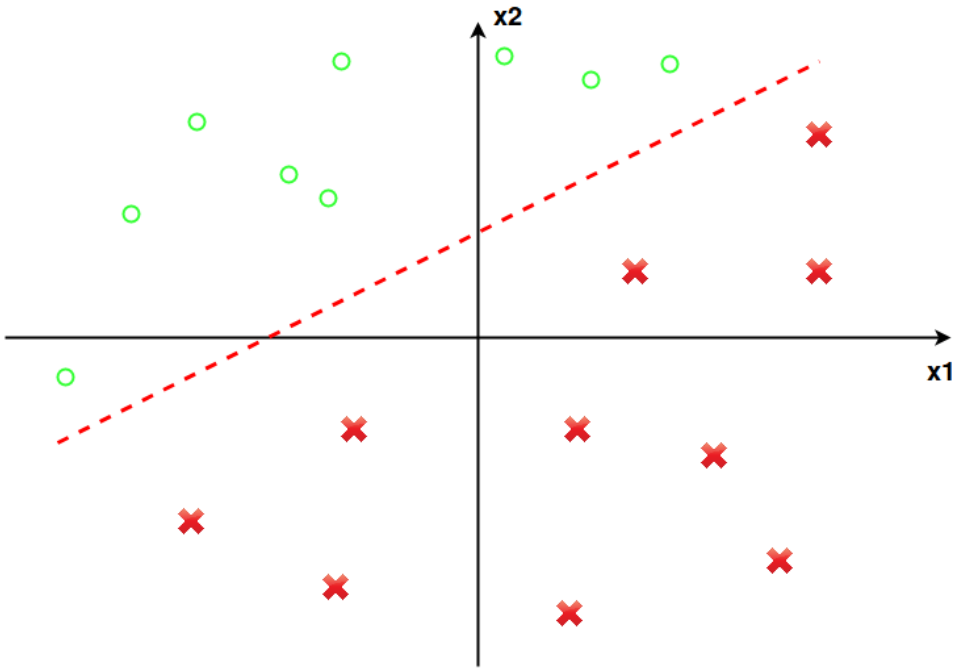

Figura 3.3: La funzione i cui campioni sono divisi in cerchi verdi e croci rosse è linearmente separabile. I 2 tipi di campioni sono distinti nei due semipiani identificati dalla disequazione $x_1 - 2x_2 > -4$.

sono dette *linearmente separabili*. Queste ultime costituiscono una classe estremamente ridotta di funzioni, per cui una rete composta da un solo neurone dà scarsi risultati, a meno che i dati non siano facilmente separabili lungo le loro dimensioni. Al contrario una rete che combini più livelli di neuroni riesce ad adattarsi meglio, specie dopo un *tuning* dei suoi parametri (numero di livelli, neuroni per livello, velocità di apprendimento, etc).

3.3 Reti feed-forward

Le reti neurali con struttura più semplice sono le reti **feed-forward**. Queste hanno una struttura molto rigorosa: tutti i neuroni infatti sono divisi e ordinati in livelli. Ogni neurone ha delle connessioni in ingresso con ogni neurone del solo livello precedente e delle connessioni in uscita con ogni neurone del solo livello successivo. In una rete feed-forward i *segnali* possono viaggiare solo in una direzione. Vista dall'esterno una rete feed-forward già addestrata è un blocco che riceve vettori di dimensione data in ingresso e restituisce vettori in uscita, anche se nel nostro caso l'uscita consisterà di un solo valore, quindi in seguito verrà considerata uno scalare tranne se diversamente

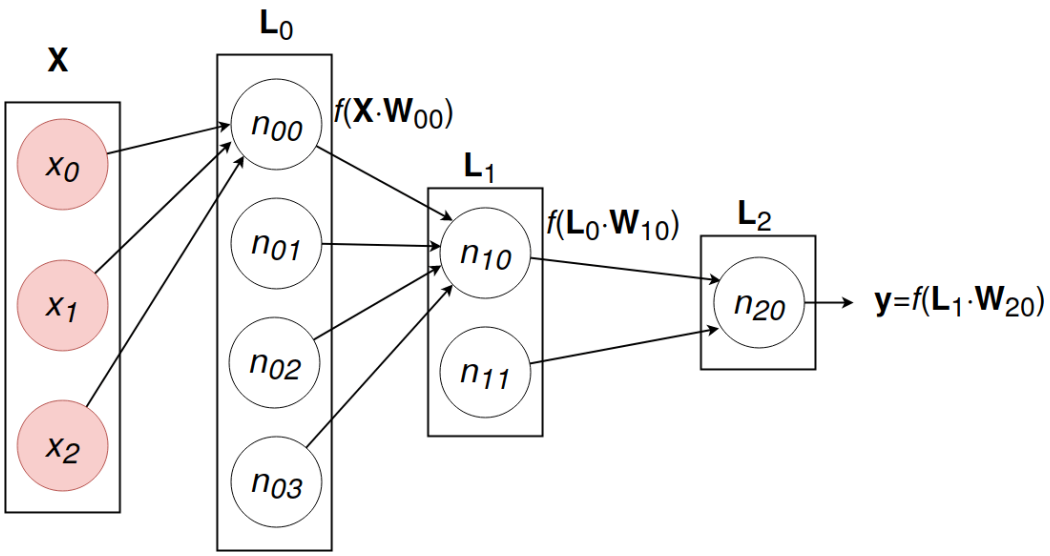

Figura 3.4: Rete feed-forward con 2 livelli nascosti rispettivamente di 4 e 2 neuroni. Solo alcuni pesi e funzioni di attivazione sono mostrati per una rappresentazione più chiara.

indicato. Come mostrato in figura 3.4, una rete feed-forward è costituita di un livello di ingresso, un insieme di livelli nascosti e uno di uscita.
Dato un vettore di ingresso X verranno una dopo l'altra calcolate le uscite di ogni livello applicando per ogni neurone del livello la formula vista. L'uscita di ogni livello farà quindi da input per quello successivo, idealmente andando a rappresentare *features* sempre di più alto livello, fino ad arrivare ad un valore o ad un insieme di valori direttamente interpretabili, dato che essi sono stati associati alle etichette prima della fase di addestramento. Nel caso in figura per esempio, dato l'unico valore in uscita, la rete può benissimo funzionare da classificatore binario indicando per ogni vettore X di ingresso se questo appartiene alla classe **0** (se **y** è minore di 0.5 ad esempio) o alla classe **1**. In alternativa reti feed-forward con più uscite $\boldsymbol{y_i} \in [0, 1]$ possono indicare per un maggior numero di classi la probabilità che X appartenga ad ogni classe, scegliendo poi la più probabile.

3.3.1 L'addestramento

Qualunque ANN prima di poter essere adoperata deve subire un processo di addestramento atto a modificarne accuratamente i pesi. Infatti una rete può essere vista come una funzione in generale non continua dei suoi ingressi $y = R(X)$. Modificare i pesi all'interno della rete significa ovviamente modificare la funzione.
L'addestramento, come già detto, sarà nel nostro caso supervisionato. Avremo quindi accesso ad un dataset composto da coppie campione-etichetta, ovvero istanze di X-y che consideriamo corrette (in generale un dataset non è esente da errori di varia natura). Durante l'addestramento quindi si modificheranno i pesi per riconoscere quante più coppie campione-etichetta del dataset.

Discesa lungo il gradiente

La modifica dei pesi di una rete sfrutta alcuni elementi di analisi matematica quali **gradiente** e **derivate parziali**. In questo contesto ne verrà data una definizione non

formale, utile solo a chiarire i concetti fondamentali e come questi elementi aiutino allo scopo dell'apprendimento della rete.

Definition 3.3.1. Gradiente di una funzione: Data una funzione $f : \mathbb{R}^n \to \mathbb{R}$ definiamo **gradiente** di f per $X \in \mathbb{R}^n$ il vettore $G \in \mathbb{R}^n, G = \nabla f(X)$ corrispondente alla direzione di massima crescita di f nel punto X.

Il gradiente di una funzione ci dice in pratica, data una posizione di partenza nello spazio di input, la direzione verso cui spostarci per massimizzare il valore della funzione. La direzione opposta rispetto al gradiente invece tenderà a minimizzare l'output. Per modificare i pesi al fine di riconoscere meglio i campioni del dataset definiremo una funzione di errore che dovrà essere minimizzata ogni epoca tramite discesa lungo il gradiente.

Definition 3.3.2. Funzione di errore: Data una rete feed-forward $R : \mathbb{R}^n \to \mathbb{R}$ i cui pesi sono ordinati nel vettore $W \in \mathbb{R}^m$, con n dimensione dell'ingresso di R e m numero di pesi, definiamo **funzione di errore** della rete relativa al singolo vettore di ingresso $X \in \mathbb{R}$ la funzione $E_X : \mathbb{R}^m \to \mathbb{R}$:

$$E_X = (y_{tag} - y_{pred})^2$$

$$con\ y_{pred} = R(X)\ e\ y_{tag}\ etichetta\ nota\ di\ X$$

La funzione di errore definita così considera un solo vettore di ingresso X.
È utile definirne una seconda versione che tenga in considerazione un insieme di l vettori di ingresso:

$$E_I = \sum_{i=1}^{l} (E_{X_i}) = \sum_{i=1}^{l} (y_{tag_i} - y_{pred_i})^2$$

La funzione di errore è quindi relativa ad una rete e ad un singolo vettore di ingresso o ad un insieme di essi. Inoltre vede gli ingressi della rete come fissi, e il vettore dei pesi della rete W come variabile indipendente.
Quale che sia la funzione di errore scelta f_{err} relativa alla rete R con pesi $W \in \mathbb{R}^m$, la legge di modifica dei pesi diventa:

$$G = \nabla f_{err}(W) \quad (1)$$

$$W = W - \alpha * G \quad (2)$$

dove in 1 viene calcolato il gradiente della funzione errore, e in 2 viene usato per aggiornare i pesi e minimizzare la funzione errore relativa ad X o ad I. Il parametro α è uno scalare chiamato *velocità di apprendimento*, che indica quanto la modifica debba incidere sui pesi, ovvero di quanto spostarsi nella direzione indicata dal gradiente. Si noti che in questa formula α deve essere positivo in modo che i pesi vengano modificati nella direzione opposta al gradiente.

Definition 3.3.3. Epoca: La legge di modifica dei pesi su cui si basa l'addestramento agisce su ogni campione del trainset, Nonostante questo, ciò non è sufficiente perché la rete impari a classificare gli elementi del trainset. Così come l'apprendimento umano richiede tempo e tentativi lo stesso vale per l'addestramento di una rete neurale artificiale. Chiamiamo allora **epoca** la singlola applicazione della legge di modifica dei pesi su tutti gli elementi del trainset. L'addestramento di una rete comprenderà generalmente più epoche.

Trainset e batch

Come visto precedentemente, quando si vanno a modificare i pesi della rete si può decidere di prendere in considerazione un solo vettore di ingresso per volta o un insieme di vettori di ingresso. Considerando una singola epoca e adottando il primo approccio e un trainset di 100 elementi, questo vuol dire che andremo a modificare i pesi della rete per 100 volte, una per ogni elemento del trainset. Questo comporta anche la presenza di 100 differenti funzioni errore. Ciò comporta alcuni svantaggi: innanzitutto l'ordinamento dei campioni all'interno del trainset influisce troppo sul risultato dell'addestramento il che è chiaramente non desiderabile. Basti pensare che sarà il primo campione e solo quello a determinare la prima modifica, il secondo a determinare la seconda e così via. Inoltre dato che le funzioni di errore sono molte e diverse tra loro, non è improbabile che alcune modifiche tendano a contrastarsi e quindi ad essere meno efficaci.
Si può adottare la seconda variante della funzione errore per evitare questi problemi, tenendo conto di tutti i campioni del trainset nella funzione errore e dunque modificando i pesi una sola volta per epoca. Anche questo approccio però non è esente da difetti, primo tra tutti la complessità computazionale. Infatti dovremo calcolare il gradiente di una funzione errore 100 volte più complessa invece di calcolare 100 gradienti di 100 funzioni errore "semplici". Dato che le ANN già tendono a soffrire della *maledizione della dimensionalità* è ovvio che questo approccio non è sempre possibile.
Inoltre l'allontanarsi del tutto da un approccio *a campione singolo* rende il risultato deterministico e per nulla dipendente dall'ordinamento dei campioni. Ciò comporta il rischiare che il gradiente porti a minimi locali, "valli" dell' unica funzione errore da cui non è possibile uscire come mostrato in figura 3.5. Il primo approccio non rischia di bloccarsi in un minimo locale perché anche se finissimo in un punto di minimo di una funzione errore, ne avremmo altre 99 per il quale questo non accade (a meno di essere estremamente sfortunati) e che quindi ci guideranno lontano da quella posizione.

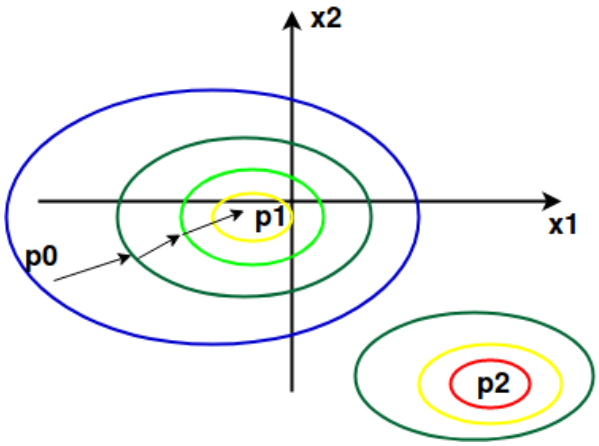

Figura 3.5: La funzione errore di una rete con soli 2 pesi (le 2 coordinate) è rappresentata in figura.
Il punto p0 di partenza si trova vicino al minimo locale p1, per cui l'usare durante epoche diverse la stessa funzione di errore porta verso p1 nonostante il punto p2 corrisponda ad un valore minore della funzione d'errore.

Quale è dunque l'approccio da seguire? La scelta spesso ricade su un approccio ibrido,

basato su **batch**. Un batch è molto semplicemente un sottoinsieme del trainset, un insieme di suoi campioni. Il trainset viene quindi diviso in un certo numero di batch e ad ogni epoca verrà effettuata la modifica dei pesi per ognuno di essi. Ad esempio nel caso avessimo 100 campioni potremmo organizzarli in 5 batch da 20 campioni l'uno. Ogni batch avrebbe la propria funzione errore relativa ai propri 20 campioni e ogni funzione errore verrebbe adoperata per una modifica dei pesi. Si hanno dunque i vantaggi di entrambi gli aspetti: da un lato un costo computazionale ridotto o quantomeno contenuto e il riuscire ad evitare minimi locali, dall'altro modifiche ai pesi meno mirate al singolo campione ma con una visione più *d'insieme*.

3.3.2 Reti feed-forward per la classificazione di sequenze

Le reti feed-forward sono un esempio di reti statiche. Durante l'addestramento i loro pesi variano, ma una volta terminato essi rimangono fissi. Ogni vettore di ingresso darà quindi sempre lo stesso risultato, indipendentemente da quelli precedenti. L'obiettivo di un oracolo comprende però la classificazione di stringhe di dimensione qualunque di simboli di un alfabeto finito, per cui questo tipo di reti non è la scelta ottimale e presenta vari problemi pratici. Ad esempio se immaginiamo di associare ogni simbolo ad un vettore di ingresso e di eseguire la classificazione di una parola inserendo in ordine nella rete i simboli che la compongono, avremmo che la classificazione dipenderebbe unicamente dall'ultimo simbolo inserito. In alternativa potremmo associare ogni simbolo ad uno scalare e di conseguenza ogni parola ad un vettore di numeri. Per l'alfabeto {a, b} potremmo decidere arbitrariamente "a" = 0.0 e "b" = 1.0, per cui "abaa" = [0.0, 1.0, 0.0, 0.0]. In questo caso dovremmo gestire la lunghezza fissa dell'ingresso della rete. Se ogni vettore di ingresso deve avere una data dimensione lo stesso vale per le parole che vogliamo classificare. Ciò significa non poter classificare parole più corte o più lunghe della dimensione dell'ingresso.
Una soluzione per poter usare parole più corte consiste nell'adoperare un valore di comodo che rappresenti un simbolo "nullo". Si potrebbe fare il padding di una stringa più corta del necessario, aggiungendo all'inizio un simbolo nullo. Ad esempio nel caso volessimo inserire la parola "aba" in una rete con ingresso di dimensione 5 potremmo mappare l'alfabeto nella seguente maniera:

$$\varepsilon \to -1$$

$$a \to 0$$

$$b \to 1$$

e quindi trasformare "aba" in "$\varepsilon\varepsilon$aba" usando il vettore [-1, -1, 0, 1, 0] come ingresso della rete. Questo però non risolve il problema delle parole più lunghe dell'ingresso, che dovrebbero per forza essere troncate perdendo importanti informazioni.
Inoltre ciò che rende questo approccio non adatto al nostro scopo è il non processare l'input come una sequenza, un simbolo alla volta. L'intero input viene preso e gestito come un blocco. Idealmente invece la nostra rete dovrebbe comportarsi come un Dfa, con uno "stato" della rete che cambi con ogni simbolo processato e un output (la classificazione fatta dalla rete) dipendente da stato e simbolo attuali. Per questo sono necessarie le RNN, per natura dinamiche rispetto all'input e che bene abbracciano questa filosofia.

3.4 RNN

Come riportato in [11] una rete neurale ricorrente è una rete ciclica, usata per gestire comportamenti temporali dinamici di problemi di apprendimento che potrebbero riguardare risultati precedentemente appresi. L'input adesso non è dato da un singolo vettore di ingresso come era per le reti feed-forward, ma da più vettori X_t che compongono una sequenza temporale. L'output dipenderà quindi da tutti gli X_t della sequenza. Ciò è possibile per via della presenza di sinapsi che portano il segnale a neuroni della rete in ritardo, in istanti successivi. In pratica ad alcuni neuroni arriveranno, oltre ai segnali dipendenti dall'ingresso corrente anche alcuni segnali derivati dalle uscite dei neuroni in istanti passati. Questi segnali che portano informazioni dal passato sono spesso considerati come una memoria della rete e possono essere visti come il suo stato interno. L'output della rete dipenderà quindi in ogni istante dall'input e dallo stato della rete.

3.4.1 Struttura di una RNN

Esistono moltissimi tipi diversi di RNN, dato che l'unico requisito è in pratica l'avere sinapsi che facciano da memoria. Dunque non esiste una struttura comune e rintracciabile in ogni RNN. Può essere mantenuta la gerarchia di livelli con cui le reti feed-forward organizzavano i neuroni ma deve essere abbandonata la rigorosità che non permetteva di connettere tramite sinapsi neuroni che non fossero di livelli adiacenti. Al contrario in questo caso è necessario che almeno una sinapsi metta in connessione l'output di un neurone n_i con l'input di un'altro neurone n_j , con n_i appartenente ad un livello non precedente al livello di n_j . La RNN raffigurata in 3.6.a consiste di un solo livello

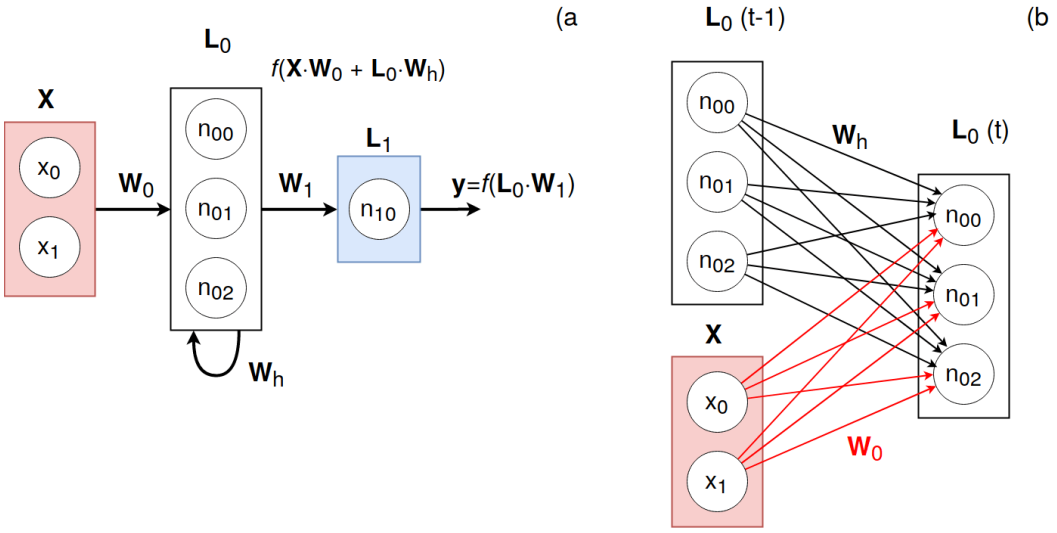

Figura 3.6: RNN con un singolo livelo nascosto, ricorsivamente connesso con se stesso.

nascosto. La differenza con una rete feed-forward è data dalla presenza delle sinapsi che connettono il livello con se stesso. Questo come mostrato in 3.6.b significa che all'istante t il livello nascosto L (t) avrà in input il vettore di ingresso X_t insieme al proprio output dell'istante precedente L_0 (*t-1*).
La rete ha quindi in ogni istante uno stato dato dai valori di output di n_{00} , n_{01} e n_{02} . L'ouput della rete sarà funzione anche di questi 3 valori oltre che dell'input.

3.4.2 RNN per l'apprendimento di linguaggi

Perché le RNN vengano usate da un oracolo di AL è necessario che siano in grado di apprendere o almeno di approssimare con una buona accuratezza i linguaggi regolari. Le reti ricorrenti sono uno strumento per la risoluzione di problemi di classificazione a tempo discreto, come mostrato ampiamente in letteratura.
In [19] è presentato un framework per la predizione di sequenze, basato sull'*evoluzione* di reti ricorrenti per la scoperta di buoni pesi relativi a livelli nascosti.
Gli autori di [6] presentano un approccio per il riconoscimento vocale basato su RNN, mentre in [4] sono valutati i risultati di più riconoscitori vocali al variare di architettura e unità delle RNN. In [5] invece le RNN sono adoperate per riconoscere la presenza di specifiche parole chiave in dialoghi non controllati.
In [7] e in [8] sono illustrate delle tecniche per il riconoscimento di testo rispettivamente *off-line* e *on-line*, la prima basata sull'immagine del testo da riconoscere e la seconda sui movimenti della penna.
In [9] è proposto un modello per la classificazione di azioni umane senza alcuna conoscenza anteriore.
Quindi è evidente che le RNN siano in grado di apprendere attività legate al comportamento umano, ma il loro metodo di apprendimento è di tipo statistico e dunque potrebbe non riuscire a catturare l'*esattezza* strutturale di un linguaggio regolare. In [20] una RNN viene addestrata per riconoscere il linguaggio $a^n b^n$, con $n \in \mathbb{N}$. Tale linguaggio è uno libero dal contesto, classe ben più complessa dei linguaggi regolari. I risultati sperimentali mostrano che la RNN adoperata, costruita con sole due unità nell'unico livello nascosto, non impari effettivamente il linguaggio ma una sua approssimazione che corrisponde ad un linguaggio regolare. In particolare la rete addestrata sulle parole $a^n b^n$ con $n = \{1, 2, ... 12\}$ finisce per assumere la forma di un oscillatore, arrivando a riconoscere il linguaggio $a^n b^n$ con $n = \{1, 2, ... 85\}$.
Dunque è in effetti possibile per un classificatore RNN imparare un linguaggio regolare ma data, a nostra conoscenza, l'assenza in letteratura di un metodo generale che lo porti sempre a questo risultato, non è irragionevole giungere alla conclusione che ciò dipenda fortemente sia dal linguaggio scelto che dalla struttura della RNN usata. Banalmente la RNN stessa può essere progettata a partire dal linguaggio target mantenedo al contempo la struttura abbastanza ridotta da poter verificare in maniera relativamente semplice se il linguaggio è stato appreso. Tutto ciò ovviamente a patto che il linguaggio non sia eccessivamente complesso. Inoltre anche se la rete potrebbe non imparare il linguaggio, a noi è sufficiente una approssimazione ragionevole di quest'ultimo. Avremmo però una approssimazione nel passare dal linguaggio target alla rete ed un'altra nel passare dalla rete al DFA se consideriamo che nel caso peggiore la rete riconoscerà un linguaggio non regolare. Questo potrebbe portare ad un peggioramento notevole della soluzione trovata. D'altro canto avendo coscienza di cosa la rete abbia appreso correttamente e di quali informazioni invece siano dubbie, si potrebbe creare una condizione di terminazione che eviti il sovra-adattamento del DFA ipotesi alla rete e che invece catturi la generalità del linguaggio target. Ad esempio si potrebbe considerare incerta la classificazione di qualunque parola più lunga del campione di lunghezza maggiore e dunque non restituire alcun controesempio se non dovesse venirne trovato uno con lunghezza adeguata. O in alternativa volendo una condizione ancora più restrittiva si potrebbe decidere di adoperare i soli campioni come possibili controesempi, adoperando la RNN solo per rispondere alle membership query.
È bene fare una distinzione a seconda di quale sia l'obiettivo che si vuole raggiungere adoperando l'approccio presentato in questa tesi:

1. la rappresentazione strutturale della rete passata per input, indipendentemente da quali siano i dati usati per l'addestramento;

2. l'automa relativo al linguaggio regolare più generico che riconosca correttamente tutti, o buona parte dei campioni che si hanno a disposizione adoperando una RNN addestrata su di essi.

Nel primo caso le considerazioni fatte prima non sono più rilevanti, perché l'interesse è di rappresentare la rete così come è, che il linguaggio appreso sia regolare o meno. L'eventuale non regolarità del linguaggio non è dunque un parametro che possa essere modificato. Quindi la rappresentazione esatta della rete tramite un DFA non potrà essere trovata tramite nessun algoritmo dato che questa non esiste. Si può però arrivare ad una sua approssimazione. Supponendo che quello imparato dalla rete non sia un linguaggio regolare, l'algoritmo di AL procederebbe all'infinito, dato che esisterebbe sempre un controesempio. Per impedire la crescita smisurata del DFA ipotesi e per permettere il termine dell'algoritmo però si potrebbe "falsare" una equivalence query mentendo al learner riguardo l'esistenza di controesempi, ponendo un limite ad esempio sul numero di stati dell'ipotesi o sul numero di equivalence query che possono essere effettuate. Gli algoritmi di AL producono sempre DFA ipotesi che non contraddicono quanto riportato dalle query fatte all'oracolo. Così anche l'ultima ipotesi pur non riconoscendo esattamente il linguaggio della rete ne darebbe una valida approssimazione, classificando correttamente un numero arbitrariamente grande di parole del linguaggio. Nel secondo caso partendo da un insieme di campioni noti del linguaggio target si vuole rappresentare la conoscenza contenuta in questi campioni tramite un DFA. Supponendo che questa conoscenza sia inerentemente regolare e dato che l'approccio scelto si basa su metodi di AL, è di vitale importanza che l'oracolo sia in grado di apprendere il linguaggio regolare. L'esito preferibile sarebbe infatti che il linguaggio fosse appreso esattamente dalla RNN e che il learner riuscisse a sua volta a rappresentare la RNN senza alcun errore. Questo permetterebbe di arrivare alla corretta rappresentazione della conoscenza espressa dai dati, tramite il DFA ipotesi.

Capitolo 4

Ricerca della conoscenza mediante reti ricorrenti

L'estrazione di automi da Recurrent Neural Network (RNN) al fine di ottenerne una rappresentazione più trasparente, è un campo di ricerca con una vasta panoramica contenente diversi approcci, la cui tassonomia è descritta da Jacobsson in [21]. Gli approcci che appartegono a questa categoria sono accomunati secondo Jacobsson dai seguenti punti, e lo stesso vale per quello proposto:

1. quantizzazione dello spazio degli stati della RNN in un insieme finito;
2. generazione di stati ed output a seguito dell'inserimento di sequenze di input;
3. rappresentazione della conoscenza a partire dall'insieme delle transizioni;
4. costruzione del modello più semplice permesso.

In particolare sono i primi due punti a riassumere il modo di procedere di questi algortmi, ovvero suddividere lo spazio delle RNN e legare gli insiemi risultanti tramite transizioni, *trasformando* la RNN in un Deterministic Finite Automaton (DFA). In [21] gli approcci sono inoltre classificati secondo una tassonomia che li distingue in base a:

1. metodo di quantizzazione dello spazio degli stati (partizionamento regolare, algoritmo *k-means*, clustering gerarchico, quantizzazione dei vettori e *self organising maps*);
2. procedura di generazione degli stati (per ricerca o per campionamento);
3. tipo e dominio della RNN.

L'approccio proposto suddivide lo spazio tramite un partizionamento basato su gerarchie di Support Vector Machine (SVM), procedendo alla generazione degli stati per ricerca (basata sull'algoritmo di AL). Il tipo di RNN è un classificatore binario e il suo dominio è dato da sequenze simboliche da accettare o rifiutare.

L'oracolo RNN

Nel capitolo precedente è stato illustrato come le Recurrent Neural Network (RNN) siano in grado di apprendere linguaggi regolari partendo dai loro campioni, come mostrato in [20], anche se ciò dipende da linguaggio obiettivo e dalla struttura della rete. Le RNN però non forniscono una rappresentazione della conoscenza perché quale che sia il loro utilizzo (classificatori, estrattori di features, riconoscitori di oggetti), esse

valutano un singolo campione per volta e non danno alcuna visione d'insieme relativa ai dati appresi. Nemmeno analizzare la struttura di una RNN è un'opzione in quanto questa è indipendente dai dati con i quali la rete è stata addestrata. Per questo, volendo rappresentare la conoscenza appresa dalla rete come un Deterministic Finite Automaton (DFA), si può fare uso un algoritmo di Active Learning (AL) che adoperi la RNN come un oracolo.

Dato un linguaggio regolare è possibile dividere tutte le infinite parole componibili a partire dal suo alfabeto, in appartenenti e non appartenenti al linguaggio. Quindi l'uso della RNN che apprenda il linguaggio è quello di un classificatore binario, che associ ad ogni parola l'etichetta **vero** o **falso**, **0** o **1**. Consideriamo adesso un classificatore binario che abbia effettivamente imparato il linguaggio target, o che almeno ne sia una buona approssimazione. Ci chiediamo se questo sia sufficiente a farne un oracolo valido. In particolare dovrà essere possibile a partire dal classificatore rispondere ai due tipi di query definite nel MAT framework: le membership query e le equivalence query. È evidente il primo tipo di query non ponga alcun problema: lo scopo di un classificatore è assegnare una etichetta ad ogni elemento passato come input, e ciò è esattamente quello che viene richiesto da una membership query.
Lo stesso non può essere detto per le equivalence query. Come già visto queste richiedono o che si indichi la correttezza dell'ipotesi fornita portando a termine l'algoritmo di active learning, o che si restituisca un controesempio che spieghi perché l'ipotesi era errata. Dipende quindi tutto dal controesempio e da come si decide di cercarlo. Un metodo diretto ma poco efficiente consiste nell'approssimare l'equivalence query effettuando più membership query, tramite una ricerca tra le parole dell'alfabeto. Lo spazio di ricerca evidentemente cresce in maniera esponenziale al crescere della lunghezza delle parole, tanto più in fretta tanto più grande è l'alfabeto. Stabilita una lunghezza massima si chiede alla RNN e al DFA ipotesi di classificare ogni parola. La prima incongruenza viene quindi restituita come controesempio. Ovviamente non si può procedere all'infinito, per cui bisognerà fissare un punto in cui interrompere la ricerca. Il non trovare un controesempio non assicura quindi di essere giunti alla soluzione, ma solo che tutte le parole nello spazio di ricerca sono classificate correttamente.
Un approccio migliore potrebbe prevedere di estrarre un automa direttamente dalla rete e usare quello per trovare un controesempio. Un metodo di estrazione di informazioni strutturali da reti ricorrenti è descritto in [22], dove delle reti ricorrenti vengono addestrate a partire dai linguaggi *tomita*. Da esse vengono quindi estratti dei DFA che dovrebbero riportare ai linguaggi di partenza. I risultati mostrano che, anche se ciò che viene trovato non è il linguaggio desiderato, esso gli si avvicina molto, arrivando a punteggi di accuratezza che superano il 90%.
Dato il DFA estratto è possibile tramite l'algoritmo del *table filling* trovare un controesempio. Purtroppo tale controesempio dimostrerà solo la non equivalenza di DFA ipotesi e DFA estratto. Di conseguenza potrebbe verificarsi che il controesempio sia classificato male dal DFA estratto e non da quello ipotesi. Questo è però meno probabile dell'alternativa se il DFA estratto rappresenta bene la rete. Questo approccio presenta un vantaggio rispetto al precedente non solo in termini di efficienza, ma anche di accuratezza poichè il confronto tra due DFA tramite il table filling corrisponde a valutare qualunque parola dell'alfabeto, trovando sempre un controesempio se questo esiste. Il metodo presentato in [22] ha però il difetto di essere *statico* poiché esso procede clusterizzando gli stati della RNN e costruendo il DFA a partire da un numero finito di questi insiemi. Ciò significa che la struttura ottenuta dipende unicamente da alcuni parametri che regolano la cluserizzazione, non legati al linguaggio obiettivo. Sarebbe invece auspicabile che il DFA venisse modificato, adattandosi ogni qual volta si

trovi una parola classificata incorrettamente, facendo crescere così durante l'algoritmo il DFA estratto assieme a quello ipotesi, a seconda di quale dei due sia in errore. Per questo motivo l'oracolo mostrato in questa tesi conterrà sia la RNN che una struttura che permetta di aggiornare il DFA dinamicamente.

4.1 L'oracolo

L'oracolo sviluppato in questa tesi si basa sul lavoro svolto in [23]. L'idea di base consiste nell'usare la RNN per rispondere alle membership query e mantenere un DFA *oracolo* per rispondere alle equivalence query. Per ogni equivalence query verrà effettuato un confronto tra il DFA ipotesi dell'algoritmo di AL e il DFA oracolo, al fine di trovare una parola riconosciuta da un DFA ma non dall'altro. Questa parola però non può essere restituita subito come controesempio, perchè nonostante il DFA oracolo idealmente rappresenti la rete, nella pratica questo avviene solo al termine dell'algoritmo e potrebbe comunque corrispondere ad una sua approssimazione fedele ma non perfetta. Dunque potrebbe essere il DFA oracolo a non riconoscere correttamente tale parola che non rappresenterebbe un controesempio valido per il DFA ipotesi. Per il possibile controesempio verrà effettuata una membership query al fine di individuare quale DFA abbia commesso l'errore. Nel caso ad essere in errore fosse il DFA ipotesi, la parola effettimente rappresenterebbe un controesempio e verrebbe restituita all'algoritmo di AL. Altrimenti la parola verrà adoperata per *raffinare* il DFA oracolo aggiungendo un nuovo stato. Oltre alla RNN e al DFA, l'oracolo si compone di un terzo elemento, il cui scopo è di *legare* i primi due facendo sì che la struttura del DFA oracolo *ricalchi* i dati appresi dalla RNN. Questo elemento è l'*albero di classificazione*, un albero binario che di permette di suddividere lo *spazio degli stati* della RNN in un numero finito di insiemi da associare agli stati del DFA.

Lo spazio degli stati

I DFA e le RNN sono modelli estremamente differenti, relativi rispettivamente ad approcci *euristici* e *cognitivi* che hanno filosofie diametralmente opposte. Ciò che accomuna i due modelli è il poter ricevere in ingresso sequenze di elementi e produrre una uscita che dipenda dall'intera sequenza e non solo dall'ultimo elemento inserito. In pratica essi hanno uno *stato* interno che influisce sull'output tanto quanto l'input e che per le RNN corrisponde alle attivazioni dei neuroni dei livelli nascosti. Da questo punto di vista lo stato di una RNN non è diverso da quello di un DFA, ma mentre gli stati di un DFA sono in numero finito, lo stato di una RNN può variare nel continuo in uno spazio a più dimensioni, le cui coordinate corrispondono alle attivazioni delle unità nascoste. Per costruire il DFA oracolo sulla base della conoscenza appresa dalla rete, seguendo il lavoro in [23], in questa tesi si procede clusterizzando lo spazio degli stati della rete in un numero finito di insiemi associati ai singoli stati del DFA oracolo. Tale classificazione viene fatta tramite un albero binario il quale permette la suddivisione dinamica dello spazio degli stati.
Una differenza sostanziale rispetto ai lavori in [22] e [23] è data dalla scelta di non adoperare ogni unità nascosta della rete nella rappresentazione dei suoi stati, poiché sebbene lo stato di una RNN in effetti comprenda tutte le sue unità, a seconda dell'applicazione scelta il numero di neuroni potrebbe portare la complessità ad esplodere. Lavorando con un sottoinsieme dei neuroni si ottiene una proiezione del reale spazio degli stati e dunque una riduzione della dimensionalità di quest'ultimo. Inoltre l'interdipendenza dei neuroni fa sì che le loro attivazioni non siano tutte necessarie ma

che presentino informazioni ridondanti. Questo risulta particolarmente evidente se la RNN presenta dei livelli *feed-forward*, nel qual caso ogni livello dipende interamente dal precedente. Quindi una possibilità è selezionare uno dei livelli feed-forward come stato della rete, considerando però che livelli troppo vicini all'uscita del classificatore potrebbero non contenere informazioni sufficienti e portare ad uno spazio degli stati la cui dimensione è tale da non permettere di discriminare correttamente le parole del linguaggio. Al contrario un livello vicino all'ingresso della rete può contenere informazioni ridondanti e portare ad uno spazio con dimensione inutilmente grande.

4.1.1 L'albero di classificazione

L'**albero di classificazione** è un albero binario ai cui nodi interni sono associate delle SVM e le cui foglie invece corrispondono a stati del DFA dell'oracolo. Esso clusterizza lo spazio degli stati della rete in modo da classificare ogni vettore stato della RNN in stati del DFA. La sua funzione è analoga a quella del *discrimination-tree* per l'Observation Pack (non vi è alcuna distinzione tra i nodi interni come invece accade per i nodi *finali* e *temporanei* del TTT). Come per il discrimination-tree esso effettua la classificazione di elementi in stati del DFA partendo dal suo nodo radice e scendendo fino a raggiungere un nodo foglia a seconda dei risultati dei nodi interni. Ma mentre il discrimination-tree è un classificatore basato sui suffissi per la classificazione delle parole, l'albero di classificazione dell'oracolo è basato sulla suddivisione dello spazio degli stati della RNN da parte delle SVM relative ai propri nodi interni, e non classifica parole ma vettori appartenenti allo spazio degli stati. Ogni SVM dell'albero partiziona lo spazio degli stati in due metà. Per classificare uno stato della RNN, si verifica a partire dalla SVM del nodo radice e poi per tutti i nodi interni incontrati successivamente, a quale metà dello spazio lo stato appartenga, procedendo quindi verso il sottoalbero destro o sinistro a seconda del risultato, ripetendo il processo fino a raggiungere un nodo foglia.

4.1.2 Il DFA

Adoperando l'albero di classificazione è possibile associare cluster di stati della RNN a stati del DFA, ma se questi stati non sono connessi da transizioni allora non ne è stata ricavata alcuna nuova informazione strutturale. A tale scopo viene mantenuto un insieme di *stringhe di accesso* prefix-closed, ognuna delle quali è associata ad uno stato del DFA. Così ad esempio, volendo chiudere la transizione c che parte dallo stato relativo al prefisso ab si procede concatenando prefisso e transizione ottenendo abc, verificando quale sia lo stato cui giunge la RNN a seguito di questa sequenza e infine adoperando l'albero di classificazione per trovare lo stato del DFA corrispondente allo stato della RNN raggiunto. L'uso dell'insieme prefix-closed associato agli stati del DFA dà una maggiore connotazione strutturale all'approccio, non presente in [23] e permette di non adoperare l'albero di classificazione nel caso in cui l'informazione sia nota. Riprendendo l'esempio precedente, se la parola abc dovesse essere il prefisso di uno stato, la transizione verrebbe fatta puntare direttamente a tale stato senza ricorrere all'albero di classificazione.

L'equivalence query

L'equivalence query segue esattamente gli stessi passi illustrati in [23]. Se i due DFA sono equivalenti allora l'algoritmo è arrivato al termine e il DFA ipotesi è quello corretto (anche se il DFA oracolo è equivalente esso potrebbe però non essere *canonico*). Alrimenti deve esistere necessariamente una coppia di parole w_1 e w_2, corrispondenti

a stringhe d'accesso per stati differenti del DFA ipotesi e a stringhe d'accesso per lo stesso stato del DFA oracolo. Le parole w_1 e w_2 sono necessariamente non Nerode congruenti per il DFA ipotesi perché quest'ultimo è canonico e dunque non esistono due suoi stati che siano equivalenti. Deve esistere quindi un suffisso s individuabile con una variazione dell'algoritmo del *table filling*, che le *distingua* per il DFA ipotesi. Al contrario non esiste alcun suffisso che le possa separare per il DFA oracolo, dato che le due parole portano allo stesso stato. Quindi i due DFA riconoscono diversamente o la parola $w_1 \cdot s$ o la parola $w_2 \cdot s$. Nel caso in cui l'errore sia del DFA ipotesi la parola va restituita come controesempio. Supponendo che invece sia il DFA oracolo a non riconoscere correttamente la parola $w \cdot s$ (con w corrispondente a w_1 o w_2), ciò significa che w e p devono essere separate (dove p è il prefisso associato allo stato del DFA oracolo attualmente puntato da w). Ovvero, dato che si è trovato il *separatore* s delle due parole w e p, queste non devono più portare allo stesso stato. Si procede quindi dividendo il cluster relativo a tale stato in due metà, sostituendo al nodo foglia corrispondente una SVM addestrata per distinguere gli stati (della RNN) relativi alle due parole. Questo corrisponde all'aggiunzione di un nuovo stato al DFA oracolo.

4.1.3 Ulteriori considerazioni

L'approccio proposto soffre della stessa problematica di quello descritto in [23], ovvero che i due DFA potrebbero convergere immediatamente verso un risultato molto semplice ma errato, nel caso in cui la prima costruzione del DFA oracolo dipendesse da un numero insufficiente di cluster. In [23] per risolvere questo problema vengono adoperati i seguenti aggiustamenti:

1. effettuare una prima clusterizzazione *forte*, per evitare il fenomeno alla base;
2. indicare all'oracolo una lista di *controesempi pronti*, parole di una certa lunghezza da valutare ad ogni equivalence query e da usare come controesempio se il DFA ipotesi non dovesse riconoscerle correttamente.

Nonostante le condizioni siano entrambe stringenti in un approccio pratico, data la necessità di inserire manualmente informazioni aggiuntive relative ai dati, la seconda appare meno problematica. Infatti essa richiede di selezionare un insieme di campioni di una certa lunghezza dal dataset, che verranno necessariamente appresi dal DFA. Al contrario il numero iniziale di stati del DFA oracolo è molto più vincolante, essendo una informazione strettamente legata alla struttura del linguaggio dei dati, l'estrazione della quale è l'obiettivo dell'intero approccio. Per questi motivi, nello sviluppo di questa tesi, si è deciso di adottare solo la seconda soluzione, creando il DFA oracolo iniziale solo a partire solo da alcune parole, in particolare la stringa vuota e una relativa ad ogni simbolo dell'alfabeto.

Capitolo 5

Risultati sperimentali

L'approccio presentato in questa tesi è stato testato su alcuni dei linguaggi tomita, punto di riferimento di precedenti lavori relativi ad estrazione di automi. L'obiettivo è stato quello di valutare i risultati di questo approccio nel caso in cui l'intero dataset sia riconosciuto dalla rete, e poi la distanza tra automa ottenuto e reale automa del linguaggio, per valori di accuratezza della rete rispetto al dataset via via decrescenti. A tale scopo si è adoperata una misura di similarità strutturale tra Deterministic Finite Automaton (DFA) denominata *neighbour matching*, la quale cerca il miglior modo di associare stati del primo DFA con stati del secondo e calcola il punteggio di somiglianza verificando per ogni coppia di stati, quante transizioni provengono da/portano a stati simili. Inoltre sono stati valutati i risultati ottenuti al variare dell'algoritmo di Active Learning (AL) adoperato. In particolare sono stati generati i dataset relativi ai DFA di tali linguaggi, contenenti parole di a e b di lunghezza pari o inferiore a 14, per un massimo di 32.767 campioni per dataset. Questi dataset sono stati adoperati per addestrare una RNN con diversi valori di accuratezza desiderati. Si è scelto di adoperare la stessa rete per tutti i linguaggi per mostrare come la sua struttura abbia un certo grado di indipendenza da quella del linguaggio da apprendere.

La rete

La rete adoperata è quella di figura 5.1, composta di un livello di ingresso, un livello *One Hot Encoder*, due livelli nascosti e una sola uscita.
Il livello One Hot Encoder è necessario ad evitare un ordinamento nei simboli dell'alfabeto, che verrebbe inevitabilmente generato se venisse usata una *codifica per interi* e che potrebbe portare a scarsi risultati. In particolare il One Hot Encoder genera i tre vettori di ingresso [1, 0, 0], [0, 1, 0] e [0, 0, 1] relativi rispettivamente a ε (il simbolo nullo), a e b.
Il primo dei livelli nascosti è composto di dieci unità Long Short Term Memory (LSTM), un tipo di RNN molto adoperato per risolvere i problemi relativi a dipendenze a lungo termine noti come *vanishing* e *exploding gradient*.
Il secondo livello nascosto è un livello *feed-forward*, adoperato oltre che per arrivare alla classificazione finale della parola, anche per "riassumere" le informazioni del livello LSTM in uno spazio di dimensione ridotta, portando la sua dimensione da quindici a cinque. È questo livello infatti a corrispondere allo stato della RNN.

I linguaggi

I linguaggi target sono quattro dei linguaggi tomita (in particolare tomita2, tomita3, tomita4 e tomita13) tutti basati su un alfabeto di 2 simboli, mostrati in figura 5.2. Si è

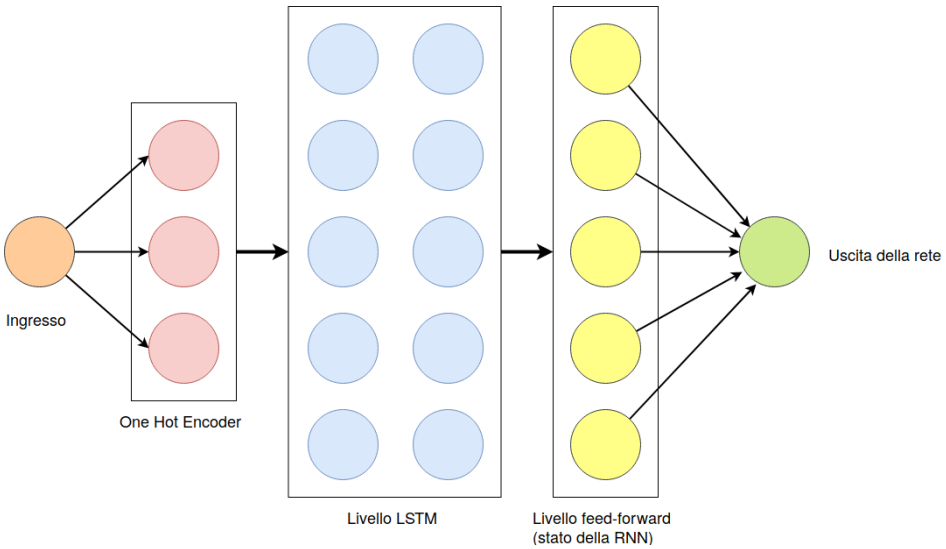

Figura 5.1: RNN adoperata

scelto di adoperare questi DFA in particolare, data la loro presenza in letteratura e la possibilità di poter descrivere ognuno di essi in linguaggio naturale (numero dispari di simboli, presenza di una data sottostringa...) che permette più facilmente di valutare i risultati ottenuti. I DFA con alfabeti di 3 simboli sono stati scartati perchè in quantità

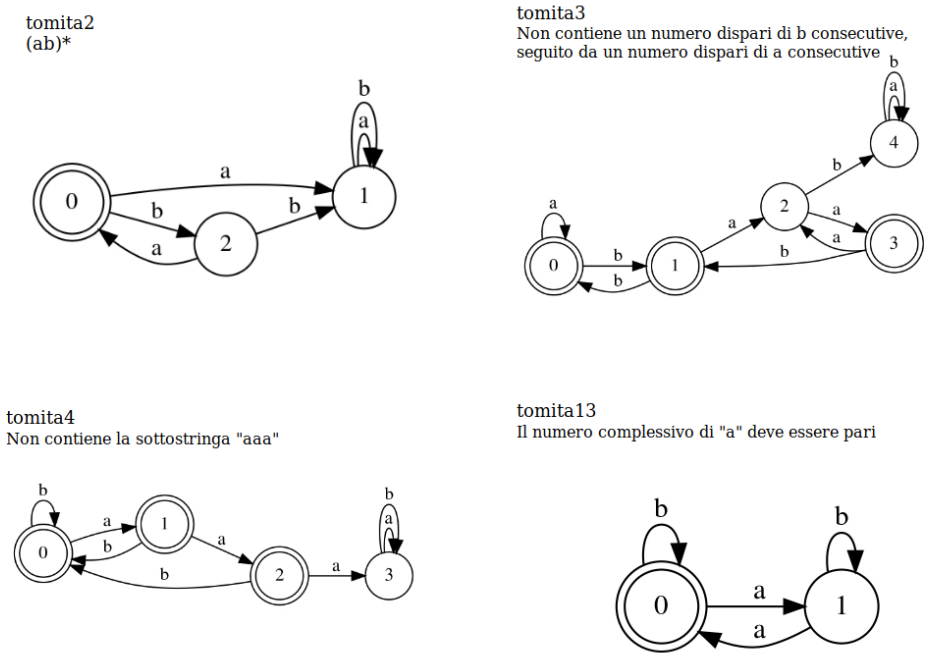

Figura 5.2: Linguaggi da cui sono stati generati i dataset

minore, in modo da poter usare la stessa rete il cui livello di One Hot Encoding dipende dalla dimensione dell'alfabeto, per tutti i linguaggi target. Dei DFA restanti alcuni sono stati scartati per via della assoluta disparità tra quantità di parole riconosciute e quantità di parole rifiutate dai loro linguaggi, il che ha fatto sì che la rete arrivasse

Costi: (Eq) Equivalence query; (Mem) Membership query; (Sim) costo in simboli;
S: punteggio di similarità tra DFA estratto e DFA target.

	tomita2			tomita3		
	L^*	*Ob.Pack*	*TTT*	$L*$	*Ob.Pack*	*TTT*
Eq	2	2	2	2	3	3
Mem	14	10	10	35	14	15
Sim	33	31	19	187	60	36
S	1.0	1.0	1.0	1.0	0.6788	0.6788
	tomita4			tomita13		
	L^*	*Ob.Pack*	*TTT*	$L*$	*Ob.Pack*	*TTT*
Eq	2	3	2	1	1	1
Mem	39	25	15	5	5	5
Sim	228	129	41	6	6	6
S	1.0	0.55	1.0	1.0	1.0	1.0

Tabella 5.1: Costi e punteggi di similarità relativi ai tre algoritmi e ai linguaggi tomita appresi.

a riconoscere non il linguaggio tomita ma quello che ammette/rifiuta qualsiasi parola. Per quanto questo risultato possa sembrare banale, è comunque interessante in quanto rende evidente la *collaborazione* tra Active Learning e approccio *euristico* dell'oracolo, anche se in questo caso corrisponde alla semplice eliminazione da parte della rete di una quantità ridotta di *outliers* che deviano da un unico grande cluster omogeneo, per il *learner*. Per i DFA restanti sono stati valutati i tre algoritmi di AL sulla base di costi di equivalence query, membership query e costo relativo a simboli, e sulla somiglianza di DFA risultato e target. I risultati sono riassunti nella tabella 5.1.

I risultati mostrano che nella maggior parte dei casi il linguaggio viene appreso esattamente (similarità pari ad 1) e quando ciò non avviene la similarità strutturale tra linguaggio ottenuto e linguaggio target è superiore al 50%. Sebbene il secondo risultato potrebbe non sembrare molto positivo è importante tenere in considerazione il fatto che variazioni ridotte nella struttura del DFA (differenza di uno o due stati, modifica di poche transizioni), portano a grosse variazioni del punteggio di similarità. Si noti inoltre che L* è arrivato sempre alla corretta rappresentazione del linguaggio target, il che dipende dalla sua tendenza ad *indagare* maggiormente per mezzo di membership query. In questo caso, dato che i dataset sono stati appresi senza alcun errore, ogni membership query ha restituito il valore corretto. D'altro canto se la rete avesse avuto una percentuale di campioni classificati erroneamente, una maggior quantità di equivalence query avrebbe aumentato la probabilità che un dato errato modificasse la struttura del DFA risultato. Quindi l'algoritmo L* dà risultati migliori nel caso di totale o quasi totale assenza di errori nei dati. Observation Pack e in particolare TTT invece, dipendendo molto di più da equivalence query (maggiormente controllabili dall'oracolo dato che è lui a scegliere il controesempio) sono più adatti a gestire dataset con una maggior percentuale d'errore, purchè l'oracolo scelga attentamente i controesempi.

Tali considerazioni sono avvalorate dai risultati mostrati in figura 5.3, ottenuti ad-

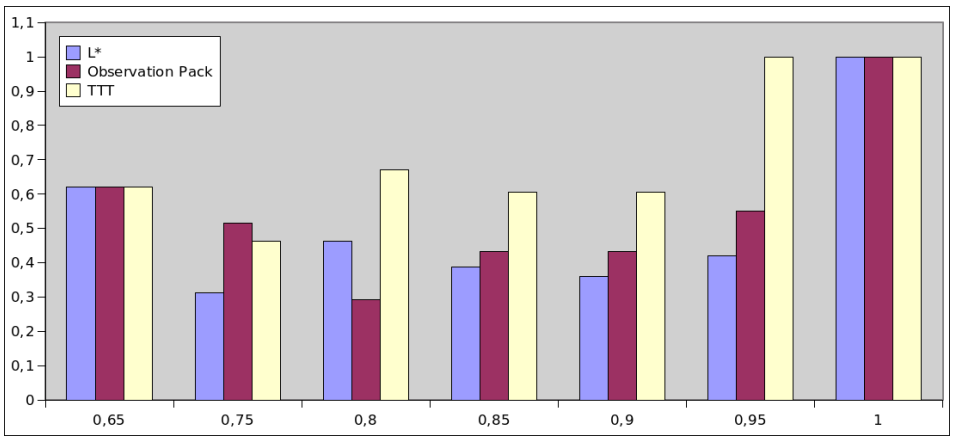

Figura 5.3: Correttezza del risultato al variare dell'accuratezza della rete

destrando la rete con diverse percentuali di accuratezza sul dataset di 32.767 campioni relativi al linguaggio tomita4. Si può notare come, nonostante la presenza di oscillazioni, il TTT risponda meglio in presenza di una accuratezza ridotta da parte della rete, per via del ricorrere meno a membership query.

Infine è bene sottolineare il fatto che nel corso degli esperimenti si è deciso di limitare il numero massimo di aggiornamenti del DFA oracolo a 35, al fine di evitare che l'algoritmo procedesse indefinitivamente. Tale decisione non ha però inciso sugli esperimenti, in quanto è stato verificato che DFA oracolo e DFA ipotesi hanno sempre converso prima.

Capitolo 6

Conclusioni e sviluppi futuri

In questa tesi è stato proposto un approccio alla knowledge discovery, il cui scopo è l'estrazione di un modello strutturale (il DFA) dai dati o da modelli *black-box* quali sono le Recurrent Neural Network (RNN). Questo approccio si differenzia da altri già presenti in letteratura per via della sua natura *dinamica*, che fa sì che il risultato venga raffinato con il procedere dell'algoritmo permettendo di stabilire un compromesso tra generalità del DFA e riscontro del modello nei dati. Inoltre la separazione dai dati fornita dall'*oracolo* evita che l'approccio sia eccessivamente *data-driven* come può accadere in approcci come il Passive Learning (PL). L'intuizione che rende questo approccio particolarmente interessante è che i dati relativi all'attività umana presentino una regolarità che può essere catturata con altri approcci, come le RNN, che però non sono in grado di rappresentarla. Tale regolarità viene accostata a quella sintattica dei *linguaggi regolari*, il che motiva maggiormente l'uso di un automa a stati finiti per la rappresentazione della conoscenza, essendo questo il più semplice riconoscitore di linguaggi regolari.
Gli esperimenti hanno mostrato che l'approccio è promettente, riuscendo ad estrarre la conoscenza se questa viene riflessa dai dati e se questi non contengono una percentuale d'errore eccessiva. I dataset adoperati però sono stati generati artificialmente a partire da linguaggi regolari e dunque il metodo deve essere ancora valutato sulla base di dati reali. Questo potrebbe essere un problema in quanto i DFA tendono a presentare strutture molto differenti anche quando i linguaggi appresi differiscono per un numero ridotto di parole. Considerando ad esempio il linguaggio *vuoto* e il linguaggio composto da una singola parola di lunghezza l, essi differiscono per una sola parola dunque possono essere considerati *linguisticamente* simili, ma i relativi DFA avranno rispettivamente 1 ed $l + 1$ stati e sono dunque strutturalmente molto diversi.
Sono stati altresì eseguiti esperimenti volti ad evidenziare come i tre diversi algoritmi di Active Learning (AL) presi in esame, influissero sul risultato. Si è osservato che tra i tre, solo L^* abbia sempre restituito il linguaggio target senza errori e benché ciò lo faccia apparire come l'algoritmo migliore, dipende sia da un maggior numero di membership query effettuate da L^* rispetto agli altri due algoritmi, che dalla assulata mancanza di errori della rete nel classificare i dati. Di norma è lecito aspettarsi errori da parte della rete, per cui l'eccessivo *zelo* di L^* nell'effettuare membership query sarebbe al contrario uno svantaggio, aumentando notevolmete la possibilità di inglobare nel DFA parole classificate non correttamente. L'approccio apparso migliore è il TTT, che pur arrivando in un caso al risultato sbagliato, lo ha fatto *generalizzando* il linguaggio con un DFA di dimensione minore, e effettuando quasi sempre il minor numero di membership query anche se al costo di un maggior numero di equivalence query. Questo è un grosso vantaggio dato che le membership query sono scelte dall'algoritmo di AL,

e potrebbero dunque riguardare parole non apprese correttamente dall'oracolo mentre essendo la scelta di un controesempio gestita interamente dall'oracolo, i controesempi possono essere scelti tra le parole che l'oracolo ha appreso correttamente.
Tra gli sviluppi futuri va considerata la possibilità di sfruttare questa caratteristica del TTT per lo sviluppo di un oracolo il quale mantenga autonomamente campioni con una bassa probabilità di errore da adoperare come controesempi, ad esempio scegliendo tra i campioni del dataset quelli riconosciuti dalla rete con un minor margine di errore (con valori maggiori di 0.85 per i campioni positivi e minori di 0.15 per quelli negativi ad esempio).
Bisognerebbe inoltre valutare i risultati ottenuti scegliendo un ambito applicativo e adoperando dataset reali, al fine di individuare quali campi siano adatti a questo approccio e di evitare risultati eccessivamente ottimistici.
Come già osservato, tutti gli esperimenti effettuati sono stati fatti a partire dalla stessa RNN addestrata sui dati diversi, al fine di mostrare che conoscenza e struttura di questo particolare modello siano indipendenti. Una controprova interessante consisterebbe nel fare l'opposto, cioè fare apprendere ad RNN strutturalmente molto diverse gli stessi dati e verificare la presenza di sostanziali differenze nei DFA risultanti.
Infine si potrebbe ampliare il metodo, usando altri classificatori binari come nucleo dell'oracolo, per i quali sia stata verificata la bontà in ambiti applicativi d'interesse, ma inadatti a rappresentare la conoscenza. Questo sarebbe di gran lunga lo sviluppo più ambizioso richiedendo molto lavoro ed una profonda conoscenza del tipo di classificatore scelto.

Bibliografia

[1] J. Naisbitt. *Megatrends: Ten New Directions Transforming Our Lives*. Megatrends: Ten New Directions Transforming Our Lives No. 158. Warner Books, 1982.

[2] N. Chomsky. "Three models for the description of language". Inglese. In: *IRE Transactions on Information Theory* 2 (1956), pp. 113–124.

[3] D. Angluin. "Learning Regular Sets from Queries and Counterexamples". In: *Inf. Comput.* 75.2 (nov. 1987), pp. 87–106.

[4] A. Graves e J. Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *Neural networks : the official journal of the International Neural Network Society* 18 (lug. 2005), pp. 602–10.

[5] S. Fernández, A. Graves e J. Schmidhuber. "An Application of Recurrent Neural Networks to Discriminative Keyword Spotting". In: *Proceedings of the 17th International Conference on Artificial Neural Networks*. ICANN'07. Porto, Portugal: Springer-Verlag, 2007, pp. 220–229.

[6] A. Graves, A. Mohamed e G. Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. Mag. 2013, pp. 6645–6649.

[7] A. Graves e J. Schmidhuber. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 21*. A cura di D. Koller, D. Schuurmans, Y. Bengio e L. Bottou. Curran Associates, Inc., 2009, pp. 545–552.

[8] A. Graves, M. Liwicki, H. Bunke, J. Schmidhuber e S. Fernández. "Unconstrained On-line Handwriting Recognition with Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 20*. A cura di J. C. Platt, D. Koller, Y. Singer e S. T. Roweis. Curran Associates, Inc., 2008, pp. 577–584.

[9] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia e A. Baskurt. "Sequential Deep Learning for Human Action Recognition". In: nov. 2011.

[10] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain [J]". In: *Psychol. Review* 65 (dic. 1958), pp. 386–408.

[11] Y. Wang *et al.* "A Survey and Formal Analyses on Sequence Learning Methodologies and Deep Neural Networks". In: *2018 IEEE 17th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*. Lug. 2018, pp. 6–15.

[12] M. Isberner. "Foundations of active automata learning: an algorithmic perspective". In: 2015.

[13] F. Howar. "Active Learning of Interface Programs. PhD thesis, TU Dortmund University". In: 2012.

[14] J. L. Balcázar, J. Díaz e R. Gavaldà. "Algorithms for Learning Finite Automata from Queries: A Unified View". In: gen. 1997, pp. 53–72.

[15] M. Merten, F. Howar, B. Steffen e T. Margaria. "Automata Learning with On-the-Fly Direct Hypothesis Construction". In: *Leveraging Applications of Formal Methods, Verification, and Validation*. A cura di R. Hähnle, J. Knoop, T. Margaria, D. Schreiner e B. Steffen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 248–260.

[16] M. Merten. "Active automata learning for real life applications". In: 2013.

[17] R. De La Briandais. "File Searching Using Variable Length Keys". In: *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*. IRE-AIEE-ACM '59 (Western). San Francisco, California: ACM, 1959, pp. 295–298.

[18] Y. Wang. “On Cognitive Foundations and Mathematical Theories of Knowledge Science”. In: *International Journal of Cognitive Informatics and Natural Intelligence* 10 (giu. 2016).

[19] J. Schmidhuber, D. Wierstra e F. Gomez. “Evolino: Hybrid Neuroevolution/Optimal Linear Search for Sequence Learning.” In: gen. 2005, pp. 853–858.

[20] J. Wiles e J. Elman. “Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks”. In: (1995).

[21] H. Jacobsson. “Rule Extraction from Recurrent Neural Networks: A Taxonomy and Review”. In: *Neural Computation* 17.6 (2005), pp. 1223–1263.

[22] Q. Wang *et al.* “An Empirical Evaluation of Rule Extraction from Recurrent Neural Networks”. In: *Neural Computation* 30.9 (2018). PMID: 30021081, pp. 2568–2591.

[23] G. Weiss, Y. Goldberg e E. Yahav. “Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples”. In: *Proceedings of the 35th International Conference on Machine Learning*. A cura di J. Dy e A. Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, ott. 2018, pp. 5247–5256.