Implementazione di un Sistema di Supporto Mnemonico

Giorgio Ravera

A.A 2007/2008

Indice

Obiettivo

In campo psico-fisiologico recenti studi correlano lo stress con la difficolta' a trasferire informazione dalla memoria a breve termine a quella a medio termine. Questa difficolta' spesso ingenera una riduzione dell'efficienza personale poiche' le cose da fare vengono in mente in modo disordinato e non pianificato. La forma estrema e' Activity Thrashing cioè l'incapacità di concludere alcunchè in quanto qualcosa di diverso da fare continua a venire in mente. E' nostra opinione che questo fenomeno potrebbe essere tenuto sotto controllo fornendo ai soggetti puntuali informazioni relativamente a quali attivita' possono essere efficientemente svolte nel suo attuale contesto.

Questo progetto si propone di studiare e sviluppare un sistema che analizzi il comportamento dell'utente per mezzo di canali quali parlato, gestualita' e movimenti oculari e traduca questo comportamento in attivita' prioritarizzate. Queste attivita' verranno poi tradotte in interrogazioni ad un sistema informativo geografico per fornire all'utente indicazioni sulle attività che possono essere efficacemente svolte nel suo contesto.

Parte I

Introduzione all'Intelligenza Artificiale

Capitolo 1

Agenti Intelligenti

1.1 Definizione di Agente

Un **agente** è un entità che agisce, cioé svolge un azione. Deve essere in grado di percepire l'ambiente che lo circonda attraverso dei **sensori** ed eseguire delle azioni attraverso degli **attuatori**.

Un **agente intelligente** o **razionale** è un agente che agisce in modo da ottenere il migliore risultato o, in caso di incertezza, il migliore risultato atteso.

Consideriamo un esempio comune di agente: una persona umana. Quando un soggetto si trova a dover risolvere un problema per prima cosa analizza l'ambiente esterno, attraverso i sensori (occhi e orecchie). In secodo luogo prova ad elaborare, nella sua mente, un idea del problema e prova ad elaborare una soluzione ottimale correlando tra loro le informazioni in suo possesso (pregresse e acquisite). Infine, attraverso gli attuatori (braccia, piedi o, più in generale, muscoli) la applica.

A questa descrizione molto elementare possiamo ricondurre ogni singola attività, dalla soluzione di un problema matematico, alla scelta di quale canale televisivo guardare oppure semplicemente a quale strada percorrere quando si è davanti a un bivio.
Un agente è composto da due elementi fondamentali ed indivisibili:

- **architettura**: il sistema fisico (corpo umano, hardware di un calcolatore) che compone l'agente e gli consente di eseguire i calcoli.
- **programma**: un insieme di istruzioni che guidano l'agente nella risoluzione dei problemi e consentano di interpretare i segnali provenienti dal mondo esterno e ricodificare i risultati per controllare gli attuatori.

Esistono diverse tipologie di agenti che si differenziano in base alla struttura che implementano. Per ogni tipologia vi è un approccio teorico specifico.

Nei paragrafi successivi verranno approfonditi aspetti legati all'ambiente esterno e alla struttura di un agente.

1.2 Ambienti

Un agente interagisce continuamente con l'ambiente che lo circonda:

- preleva informazioni attraverso i sensori
- compie azioni attraverso gli attuatori

Con il termine **percezione** si indicano gli input che l'agente ottiene, tramite i sensori, dall'ambiente esterno in un dato istante. La **sequenza percettiva** è la storia completa di tutto ciò che l'agente ha percepito nella sua esistenza.

In generale la scelta dell'azione di un agente in un qualsiasi istante può dipendere dall'intera sequenza percettiva osservata fino a quel momento. Se possiamo specificare l'azione prescelta dall'agente per ogni possibile sequenza percettiva, allora abbiamo descritto l'agente in modo completo. Ciò implicherebbe avere una conoscenza completa dell'ambiente esterno e questo non è sempre possibile, come vedremo in seguito.

Si indentifica con **funzione agente** l'insieme di azioni che descrivono il comportamento dell'agente in risposta a ciascun elemento della sequenza percettiva. La si può vedere come una tabella che mette in corrispondenza di ogni percezione una delle possibili azioni. Nel caso non si conoscesse tale funzione è possibile ricavarla (interamente o solo parte di essa) provando tutte le possibili sequenze percettive e registrando il comportamento dell'agente.

La tabella appena descritta ha una validità *esterna* all'agente. Al suo interno ci sarà un **programma agente** che implementerà tale funzione. E' importante sottolinearne la differenza: la funzione è una descrizione matematica astrazione, il programma è una sua implementazione concreta in esecuzione sull'architettura dell'agente.

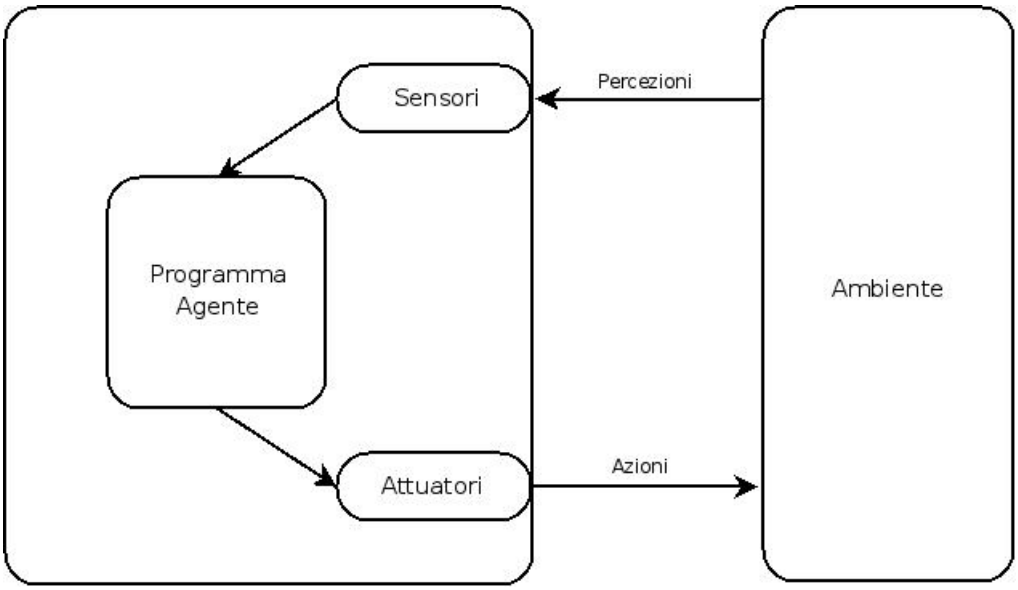


Figura 1.1: Schema di un generico Agente

La figura 1.1 illustra visivamente quanto detto fin ora: come si può vedere viene separato dall'ambiente esterno l'agente che è composto da sensori, attuatori e programma agente. Non è presentato un dettaglio di quest'ultimo per mantenere il più generica possibile la presentazione. Nel paragrafo 1.3 ne verranno mostrate le diverse implementazioni.

1.2.1 Tipologie di Ambienti

Per poter realizzare un agente con un dato obiettivo, è necessario definire l'ambiente che lo circonda. Da esso, infatti, l'agente trarrà tutte le informazioni necessarie per poter effettuare le decisioni e gli elementi per poterle attuare.

E', quindi, necessario identificare delle proprietà in base a cui suddividerli in categorie. Queste proprietà determinano in gran parte la progettazione di agenti appropriati e l'applicabilità delle principali tecniche alla loro implementazione.
Le principali proprietà sono le seguenti:

- **osservabilità completa/parziale**: caratterizza il grado d'accesso dei sensori al mondo esterno. Un ambiente parzialmente osservabile può essere dovuto a sensori inaccurati o imprecisi, alla presenza di rumore oppure all'impossibilità di ottenere alcune informazioni.

- **deterministico/stocastico**: nel caso di determinismo l'agente riesce a prevedere lo stato successivo dell'ambiente sulla base della misura fatta e dell'azione. Qualora ciò non fosse possibile si parla di ambiente stocastico. Nel caso in cui l'ambiente è deterministico ad eccezione delle azioni che altri agenti potrebbero compiere si parla di ambiente **strategico**.

- **episodico/sequenziale**: un ambiente si dice episodico se ogni azione generata non dipende dalle precedenti. Sequenziale nel caso in cui più percezioni influenzano la scelta dell'azione da svolgere.

- **statico/dinamico**: un ambiente è statico nel caso in cui non può cambiare nel periodo in cui l'agente ragiona al fine di prendere la decisione ottima. E' dinamico quando può variare durante la fase di ragionamento. Ciò comporta grossi problemi perché l'azione generata potrebbe non essere più quella corretta. Se l'ambiente non cambia con il tempo ma la valutazione dell'agente si allora si dice **semidinamico**

- **discreto/continuo**: tale classificazione è applicata allo *stato* dell'ambiente e a come viene rappresentato il *tempo* alle percezioni e azioni dell'agente. Può essere discreto, campionato ad intervalli regolari, oppure continuo, analizzato con continuità senza salti.

- **agente singolo/multiagente**: si parla di ambiente a singolo agente nel caso in cui eventuali altri elementi autonomi nell'ambiente non influenzano le scelte effettuate. Si parla di multiagente quando altri agenti possono partecipare attivamente al raggiungimento dell'obiettivo.

Si può capire facilmente che il caso più complesso è dato dalla combinazione di ambiente: *parzialmente osservabile, stocastico, sequenziale, dinamico, continuo e multiagente.*

1.3 Struttura di un agente

Come già visto in precedenza vale la seguente legge:

$$agente = architettura + programma \tag{1.1}$$

Con **architettura** si identifica la componente fisica, materiale dell'agente, ovvero l'hardware, i circuiti e le periferiche che lo compongono. E' composta prevalentemente da quattro differenti elementi:

- **strumenti di calcolo**: componenti che si occupano di elaborare i dati per relazionarli ed ottenere gli strumenti per poter effettuare le decisioni. In genere sono rappresentati dal/dai processore/i.
- **sensori**: analizzano l'ambiente esterno e codificano le informazioni raccolte in dati analizzabili dagli apparati di calcolo. Sono in genere microfoni, tastiere, videocamere o misuratori di temperatura.
- **attuatori**: componenti che ricevono istruzioni dagli apparati di calcolo e li traducono in azioni, come ad esempio il movimento di un braccio o la produzione di un segnale visivo/uditivo. Alcuni esempi possono essere display o casse acustiche.
- **memoria**: in alcuni agenti è presente la possibilità di memorizzare informazioni per poter avere più dati da tenere in considerazione nell'elaborazione di decisioni. Possono essere realizzati mediante database memorizzati su dischi o nastri.

Il **programma agente** è un software che deve essere in grado di:

- acquisire correttamente i dati provenienti dai sensori
- codificare tali dati per ottimizzare la funzione di decisione/ragionamento
- inviare agli attuatori azioni da svolgere compatibili con la loro progettazione

Come si può osservare dalla sua definizione, tale programma deve essere progettato per una specifica architettura e, quindi, risultare compatibile con i sensori e gli attuatori scelti.

E' importante sottolineare la differenza tra il programma agente che prende come input solamente la percezione corrente e la funzione agente il cui input è costituito dall'intera storia delle percezioni. Il programma agente si basa sulla sola percezione corrente perché l'ambiente non può fornirgli nulla di più. Se le sue azioni dipendono dalla sequenza percettiva precedente è necessario utilizzare un sistema di memorizzazione.
Il seguente pseudocodice dovrebbe aiutare a capire la struttura base di un programma:

```
class agente-semplice
{
        map<perception, action> m;

        action calcola(perceptions p)
        {
                return m.lookup(p);
        }
}
```

Esistono sei tipologie diverse di agenti associate a una opportuna tipologia di programma agente che rappresentano i principi alla base di quasi tutti i sistemi intelligenti:

- agenti reattivi semplici
- agenti basati su modello
- agenti basati su obiettivi
- agenti basati sull'utilità
- agenti che apprendono
- agenti basati sulla conoscenza

1.3.1 Agenti Reattivi semplici

Un *agente reattivo semplice* è un agente che sceglie le azioni sulla base della sola percezione corrente, ignorando completamente quelle passate. In altre parole non ha memoria.

Un tale agente è, appunto, molto semplice da realizzare ma ha una intelligenza molto limitata. Si può schematizzare il suo operato attraverso il seguente pseudocodice:

```
class agente-reattivo-semplice
{
        map<state, list<rule>> rules;
        map<rule, action> actions;

        action calcola(perceptions p)
        {
                state s = code_percption(p);
                rule r1 = (rules.lookup(s, p)).getFirst();
                return actions.lookup(r1);
        }
}
```

Per prima cosa, viene codificata in forma astratta la percezione attraverso una sua elaborazione (funzione code_perception). Da qui si potrà ottenere uno stato dell'ambiente esterno. A questo punto è necessario interpretare lo stato e capire che **regola condizione-azione** attribuire a ciò che si è osservato. Nel caso in cui ci fossero più regole per lo stesso stato verrà considerata sempre e solo la prima. A questo punto si procede cercando l'azione da compiere.

Interpretiamo con un esempio quanto appena descritto. Consideriamo un sistema di guida autonomo: ci troviamo dietro ad un'automobile e dobbiamo capire quando questa frena per fare la stessa cosa. I sensori sono una serie di telecamere montate sul parabrezza, gli attuatori sono i freni. Ad intervalli regolari viene catturata un immagine: dall'immagine (percezione) si studiano le luci di segnalazione e in particolare se sono accese o spente (stato). In base al valore appena elaborato si cerca una corrispondenza tra questo e l'azione da svolgere. Si estrarrà, in caso di luci accese una regola condizione azione del tipo:

if (**light_on**) then **start_to_break**

Con questa regola diventa semplice capire che azione svolgere, nel caso corrente frenare.

Anche gli esseri umani fanno uso di connessioni analoghe, alcune basate sull'apprendimento, altre sui riflessi. Questi aspetti verranno approfonditi nei capitoli successivi.

Un tale agente funziona solo e unicamente nel caso in cui l'ambiente sia **completamente osservabile**. Anche una minima parte di inosservabilità potrebbe causare errori molto gravi. Nel caso in esempio la struttura potrebbe non prevedere la possibilità che le luci di segnalazione della macchina davanti non funzionino, oppure non differenziarli dalle luci di posizione. Limitandosi, quindi, alla sola analisi delle luci il sistema non funziona.

Infine un tale agente tende ad eseguire **cicli infiniti** se l'azione da compiere è sempre la stessa per ogni circostanza e lo stato non può mutare per peculiarità proprie dell'azione stessa. Supponiamo di avere un robot che deve muoversi si un terreno in relazione al colore delle piastrelle: se le piastrelle sono bianche si muove a destra o in basso, se sono nere a sinistra o in alto. A un tratto arriva in una piastrella nera con alla sua sinistra un muro. L'azione che dovrebbe essere eseguita sarà il movimento verso sinistra (prima regola) ma non sarà possibile e alla percezione successiva lo stato sarà analogo al precedente.

Per ovviare a questo problema è possibile scegliere in maniera casuale le azioni da svolgere. Ciò dovrebbe evitare cicli infiniti.

1.3.2 Agenti basati su Modello

Il modo più efficace di gestire l'osservabilità parziale, per un agente, è tener traccia della parte del mondo che non può vedere nell'istante corrente. Questo significa che l'agente deve memorizzare una sorta di stato interno che dipende dalla storia delle percezioni e che quindi riflette almeno una parte degli aspetti non osservabili dello stato corrente.

Ritornando al problema della frenata, lo stato interno non è troppo complesso: basta mantenere in memoria il fotogramma precedente scattato dalla telecamera, permettendo cosi all'agente di accorgersi quando due luci rosse, alle estremità laterali del veicolo, si accendono o spengono contemporaneamente.

Aggiornare l' informazione di stato al passaggio del tempo richiede che il programma agente possieda due tipi di conoscenza:

- informazione sull'evoluzione del mondo indipendente dalle sue azioni
- informazioni sull'effetto che hanno sul mondo le azioni dell'agente stesso

Questa conoscenza sul funzionamento del mondo, viene chiamata **modello del mondo**. Un agente che si appoggia a un simile modello prende appunto il nome di **agente basato su modello**.
Si può schematizzare il suo operato attraverso lo pseudocodice sottostante:

```
class agente-reattivo-con-stato
{
        stete s;
        actions last_action;
        map<state, list<rule>> rules;
        map<rule, action> actions;

        action calcola(perceptions p)
        {
                s.update-state(p,last_action);
                rule r1 = (rules.lookup(s, p)).getFirst();
                last_action = actions.lookup(r1);
                return last_action;
        }
}
```

La parte interessante è la funzione update-state(), responsabile della creazione del nuovo stato interno. Oltre a interpretare la nuova percezione alla luce della conoscenza preesistente dello stato, la funzione utilizza le informazioni sull'evoluzione del mondo per tener traccia delle parti di esso che non sono presentemente visibili. La funzione deve anche conoscere l'effetto delle azioni dell'agente sullo stato del mondo.

1.3.3 Agenti basati su Obiettivi

Conoscere lo stato corrente dell'ambiente non sempre basta a decidere che cosa fare. Oltre che della descrizione corrente dello stato l'agente ha bisogno di qualche tipo di informazione riguardante il suo obiettivo. Il programma agente può unire quest'informazione a quella che riguarda i risultati delle possibili azioni (la stessa usata anche per aggiornare lo stato interno in un agente reattivo) per scegliere quelle che portano al soddisfacimento dell'obiettivo.

Talvolta scegliere un'azione in base a un obiettivo è molto semplice, quando questo può essere raggiunto in un solo passo. Altre volte è più difficile, quando l'agente deve considerare lunghe sequenze di azioni alternative per trovare il cammino che porta al risultato desiderato. La ricerca e la pianificazione sono dei sottocampi dell'Iintelligenza

Artificiale dedicati proprio a identificare le sequenze di azioni che permettono a un agente di raggiungere i propri obiettivi.

Notate che questo tipo di decisioni non ha nulla a che vedere con le regole condizione-azione che abbiamo descritto prima, perchè ora dobbiamo prendere in considerazione il futuro sotto due aspetti:

- cosa accadrà se si esegue l'azione prescelta?
- il risultato dell'azione sarà soddisfacente?

Nella progettazione degli agenti reattivi quest'informazione non viene rappresentata esplicitamente, perchè le regole interne mettono direttamente in corrispondenza percezioni e azioni. L'agente reattivo frena quando vede le luci di frenata. Un agente bassato su obiettivi, in via di principio, potrebbe ragionare che se la macchina di fronte ha le luci di frenata accese starà rallentando. Dato il modo in cui evolve normalmente il mondo, l'unica azione che permetterebbe di raggiungere l'obiettivo di non tamponare altre macchine sarà allora quella di frenare.

Benchè un agente basato su obiettivi sembri meno efficiente, d'altra parte è più flessibile, perchè la conoscenza che motiva le sue decisioni è rappresentata esplicitamente e può essere modificata. Se comincia a piovere, l'agente può aggiornare la propria conoscenza circa l'efficienza dei propri freni. Nel caso dell'agente reattivo semplice sarà necessario riscrivere molte regole condizione-azione.

1.3.4 Agenti basati sull'Utilità

Nella maggior parte degli ambienti gli obiettivi, da soli, non bastano a generare un comportamento di alta qualità. Ci sono molte sequenze di azioni che porteranno al raggiungimento dell'obiettivo ma alcune potrebbero risultare migliori, in termini di costo, sicurezza o affidabilità, rispetto ad altre.

Gli obiettivi forniscono solamente una distinzione binaria tra stati soddisfacenti o meno, Occorre una misura di prestazione più generale permettendo di confrontare stati del mondo differenti e misurare precisamente il grado di contentezza che si otterrebbe se l'agente riuscisse a raggiungerli.

Una funzione di utilità assegna a uno stato (o a una sequenza di stati) un numero reale che quantifica il grado di preferibilità ad esso associato. Una specifica completa della funzione di utilità permette decisioni razionali in due categorie di casi in cui non bastano gli obiettivi:

- **conflitti**: quando ci sono più obiettivi da raggiungere che non si possono ottenere insieme. La funzione di utilità specifica quali privilegiare.
- **incertezza**: quando ci sono più obiettivi raggiungibili ma nessuno può essere ottenuto con certezza. Il concetto di utilità fornisce un mezzo per confrontare le probabilità di successo e l'importanza degli obiettivi.

Ogni agente razionale deve comportarsi come se possedesse una funzione di utilità di cui cerca di massimizzare il valore atteso. Un agente che possiede una funzione di utilità esplicita può quindi prendere decisioni razionali, e lo può fare seguendo un algoritmo generale che non dipende dalla specifica funzione di utilità che si desidera massimizzare. In questo la definizione globale di razionalità, che definisce razionali quelle funzioni agente che hanno le prestazioni migliori, viene trasformata in un veicolo locale in progetti di agenti razionali che possono esseri espressi in un semplice programma.

1.3.5 Agenti che Apprendono

Secondo Turing, in un articolo pubblicato nel 1950, è possibile costruire macchine capaci di apprendere e poi addestrarle. L'apprendimento presenta un enorme vantaggio: permette agli agenti di operare inizialmente in ambienti sconosciuti, diventando col tempo più competenti di quanto fossero all'inizio, allorchè si basavano sulla sola conoscenza iniziale. Un agente capace di apprendere può essere diviso in quattro componenti astratti:

- elemento di apprendimento
- elemento esecutivo
- elemento critico
- generatore di problemi

La distinzione più importante è tra: **elemento di apprendimento** (learning element), responsabile del miglioramento interno, e **l'elemento esecutivo** (performance element) che si occupa della selezione delle azioni esterne. Quest'ultimo è ciò che abbiamo considerato fin qui come se costituisse l'intero agente: prende in input le percezioni e decide le azioni. L'elemento di apprendimento utilizza informazione proveniente dall'elemento critico riguardo le prestazioni correnti dell'agente e determina se e come modificare l'elemento esecutivo affinchè in futuro si comportino meglio. Il progetto dell'elemento di apprendimento dipende molto da quello dell'elemento esecutivo. Quando si cerca di progettare un agente che impara a svolgere una certa attività, la prima domanda da porsi non è come faccio a fargli imparare questa cosa? ma quale tipo di elemento esecutivo permetterà al mio agente di fare questa cosa, una volta l'avrà imparata?. Qualsiasi progetto di agente può essere migliorato in ogni sua parte dall'aggiunta di meccanismi di apprendimento.

L'**elemento critico** dice a quello di apprendimento come si sta comportando l'agente rispetto a uno standart di prestazione fissato. Quest'elemento è necessario perchè le percezioni, in sé, non forniscono alcuna indicazione del successo dell'agente. Un programma di scacchi potrebbe ricevere una percezione che indica che ha dato scacco matto all'avversario, ma ha bisogno di uno standart di prestazione per sapere che questa è una cosa buona; la percezione da sola non basta. E importante che lo standart sia prefissato: concettualmente lo si può pensare come un'entità del tutto separata dall'agente, dato che quest'ultimo non lo deve modificare per adattarlo al suo comportamento.

L'ultimo componente di un agente capace di apprendere è il **generatore di problemi**, il cui scopo è suggerire azioni che portino a esperienze nuove e significative. L'idea è che se si lasciasse mano libera all'elemento esecutivo, esso continuerebbe a ripetere le azioni che ritiene migliori date le conoscenze attuali. Ma se l'agente è disposto a esplorare qualche altra possibilità, e magari eseguire nel breve termine qualche azione subottima, potrebbe scoprire l'esistenza di azioni molto superiori a lungo termine. Scopo del generatore di problemi è di suggerire tali azioni esplorative.

Applicando quanto detto al caso del taxi possiamo sostenere che: l'elemento esecutivo corrisponde alla collezione di conoscenze ed procedure usate dal taxi per selezionare le possibili azioni di guida. l'elemento critico osserva il mondo e passa delle informazioni a quello di apprendimento. Da esse l'elemento di apprendimento può formulare una regola che viene aggiunta all'insieme. A questo punto il generatore di problemi potrebbe identificare certe aree di comportamento che necessitano di qualche miglioria e suggerire esperimenti da effettuare.

Tale tipologia di agente è applicabile a tutti quelli analizzati in precedenza. L'elemento di apprendimento, infatti, può modificare uno qualsiasi dei comportamenti degli agenti appena descritti.

1.3.6 Agenti basati sulla Conoscenza

Gli agenti visti fin ora appartengono all'insieme degli agenti reattivi che operano attraverso il calcolo delle conseguenze delle azioni. Questo porta ad ottenere una conoscenza limitata non deduttiva della realtà. Consideriamo gli scacchi: un agente reattivo sa calcolare le mosse di ogni singola pedina e le conseguenze di ogni azione ma, in senso generale, non sa giocare a scacchi.

Gli agenti basati sulla conoscenza possono trarre beneficio da conoscenze espresse in forma generale, combinando e ricombinando le informazioni per adattarle a una varietà di scopi. Spesso, tale procedimento può non avere un legame diretto con le necessità immediate. La conoscenza la si può suddividere in due elementi fondamentali:

- **regole apprese**: procedure, informazioni e, in generale, conoscenza acquisita da uno studio o trasmessa da agenti o persone
- **schemi di associazione**: collegamenti tra le informazioni

Il componente più importante degli agenti basati sulla conoscenza è appunto la **base di conoscenza**, o KB (dall'inglese knowledge base). Informalmente, la base di conoscenza è costituita da un insieme di formule, espresse mediante un linguaggio di rappresentazione della conoscenza. Ogni formula rappresenta un'asserzione sul mondo.

La base di conoscenza deve prevedere meccanismi per raggiungere nuove formule e per le interrogazioni. I nomi standard per queste due azioni sono rispettivamente TELL (asserisci) e ASK (chiedi): entrambe possono comportare un processo di inferenza, ovvero la derivazione di nuove formule a partire da quelle conosciute. La risposta a ogni richiesta (ASK), posta alla base di conoscenza, sia una conseguenza di quello che le è stato detto

(attraverso TELL) in precedenza.
Si può schematizzare quanto detto attraverso lo pseudocodice sottostante:

```
class KB-agent
{
        database knowledge;

        action calcola(perceptions p)
        {
                knowledge.tell(p);
                action a = knowledge.ask(p);
                knowledge.tell(a);
                return a;
        }
}
```

L'agente mantiene in memoria una base di conoscenza, KB, che può contenere conoscenza iniziale (background knowledge). Ogni volta che viene invocato, il programma agente fa due cose

- comunica le sue percezioni alla base di conoscenza tramite la funzione TELL.
- chiede quale azione eseguire tramite la funzione ASK

Rispondere a questa domanda può comportare un esteso processo di ragionamento sullo stato corrente del mondo, le conseguenze delle possibili azioni e cosi via. Una volta che è stata scelta una azione l'agente la registra con TELL prima di eseguirla. Il secondo TELL è necessario per dire alla base di conoscenza che l'ipotetica azione è stata effettivamente eseguita.

I dettagli del linguaggio di rappresentazione sono racchiusi nelle tre finzioni che implementano l'interfaccia tra i sensori e attuatori da una parte e il sistema interno di rappresentazione e ragionamento dall'altra. Un agente siffatto si presta bene a essere descritto a livello di conoscenza, in cui per fissare il comportamento basta specificare solamente ciò che l'agente conosce e i suoi obiettivi. Ad esempio, un taxi automatico potrebbe avere l'obiettivo di portare un passeggero in Via Opera Pia 13, e potrebbe sapere di trovarsi a Genova. Allora ci aspetteremo che il taxi identifichi la via migliore per raggiungere la destinazione, perchè sa che cosi facendo raggiungerà il suo obiettivo.

Notate che quest'analisi è del tutto indipendente dal funzionamento del taxi a livello di implementazione: non importa se la sua conoscenza della geografia è realizzata con liste dinamiche o mappe di pixel, o se per ragionare manipola stringhe di simboli memorizzate in registri o propaga segnali in una rete neurale.

Capitolo 2

Elaborazione del Linguaggio Naturale

2.1 Comunicare informazioni

La **comunicazione** consiste nello scambio intenzionale di informazioni attraverso la produzione e percezione di **segni** appartenenti a un sistema convenzionale e condiviso. E' uno strumento attraverso il quale è possibile condividere informazioni.

Il **linguaggio** è il complesso sistema di messaggi strutturati che permettono a due o più entità di comunicare con chiarezza una parte della loro conoscenza. Solitamente un linguaggio prevede *regole sintattiche* e *semantiche* che consentono a due o più interlocutori di capirsi. Appare ovvio, da questa definizione che due o più agenti che vogliano comunicare tra loro devono condividere il linguaggio utilizzato.

Un agente, per produrre un linguaggio, compie un azione che prende il nome di **atto linguistico**. Con questo termine non si intende solamente la produzione di linguaggio parlato ma, più in generale, tutto ciò che può costituire una comunicazione. Ciò che viene prodotto è un insieme di segni comunicativi chiamati **parole**.

Un agente può intraprendere differenti tipologie di atti linguistici:

- interrogare altri agenti su aspetti dell'ambiente esterno
- informare altri agenti riguardo aspetti dell'ambiente esterno
- richiedere ad altri agenti di eseguire azioni
- acconsentire alle richieste
- promettere o impegnarsi di svolgere azioni
- dichiarare situazioni o proprietà del mondo esterno

Il riconoscimento della tipologia dipende dalla frase stessa. E' simile a qualsiasi altro **problema di comprensione**, come il riconoscimento di immagini o la diagnosi medica. L'agente riceve un insieme di input ambigui e deve procedere all'indietro per decifrarne il significato.

2.2 Fondamenti del linguaggio

Un **linguaggio formale** è definto da un insieme di **stringhe**, ciascuna costituita da **simboli terminali** chiamati **parole**. Hanno una sintassi molto rigorosa e sono facilmente interpretabili.

Un **linguaggio naturale**, invece, è definto da un insieme di stringhe costituite da **simboli terminali** ma non presentano una sintassi rigorosa. Sono i linguaggi utilizzati dalle persone per esprimersi.

Una **grammatica** è un insieme finito di regole che specifica un linguaggio. I linguaggi formali possiedono una grammatica ufficiale e ben definita mentre per i linguaggi naturali non è così.

Si possono approssimare linguaggi naturali a linguaggi formali imponendo una sintassi rigorosa basata su uno studio delle forme di comunicazione più comuni gestendo le eccezzioni attraverso aggiornamenti alla grammatica stessa.

Ad ogni stringa valida, ovvero che appartiene al linguaggio, è associato un significato, detto **semantica**. Nei linguaggi naturali è importante anche comprendere la **pragmatica** di una stringa, ovvero il suo significato in relazione al contesto (spazio-tempo) in cui viene prodotta.

I formalismi su cui si fondano le principali grammatiche sono basati sull'idea di **struttura sintagmatica**, secondo cui le stringhe sono composte da sottostringhe, chiamate **sintagmi** e suddivise in categorie. Tale suddivisione è utile per diverse ragioni. Innanzitutto i sintagmi corrispondono ad elementi semantici naturali da cui si può ricostruire il significato di un enunciato. In secondo luogo, una tale categorizzazione aiuta a descrivere quali sono le stringhe accettabili nel linguaggio.
Consideriamo ad esempio una frase:

Il re *è caduto*

Si può evidenziare chiaramente che la parte sottolineata corrisponde a un sintagma nominale, la parte in corsivo a un sintagma verbale.
Da ciò si può generalizzare che una frase è così ottenuta:

```
< frase > = <  sintagma nominale> < sintagma verbale >
```

Questa, di fatto, rappresenta una **regola di produzione** di una grammatica. I simboli in essa presenti sono detti **simboli non terminali**. Si possono intendere come dei segnaposto che dovranno essere sostituiti con altri simboli. A partire da un **assioma**, ovvero un simbolo non terminale, e quindi astratto, si iniziano ad applicare regole di produzione per ottenere, una frase sintatticamente corretta appartenente al linguaggio. Tale processo prende il nome di **derivazione**.

Si tenga presente che non tutte le frase sintatticamente corrette hanno un senso. In altre parle la correttezza sintattica non implica la correttezza semantica.

Le regole delle grammatiche possono essere utilizzate sia per l'**analisi** (accettare/rifutare stringhe, associare alberi che descrivono la composizione di stringhe) sia per la **generazione** di stringhe appartenenti al linguaggio.

2.3 Fasi della comunicazione

La comunicazione tra due agenti è suddivisa in diverse fasi. Ciascuna si occupa di realizzare una specifica funzione, superando i problemi che si possono generare. L'obiettivo è quello di inviare un informazione o una richiesta ad un altro agente.
Consideriamo due agenti A e B. A decide di intraprendere un atto linguistico con B per comunicare la frase:

il re è caduto

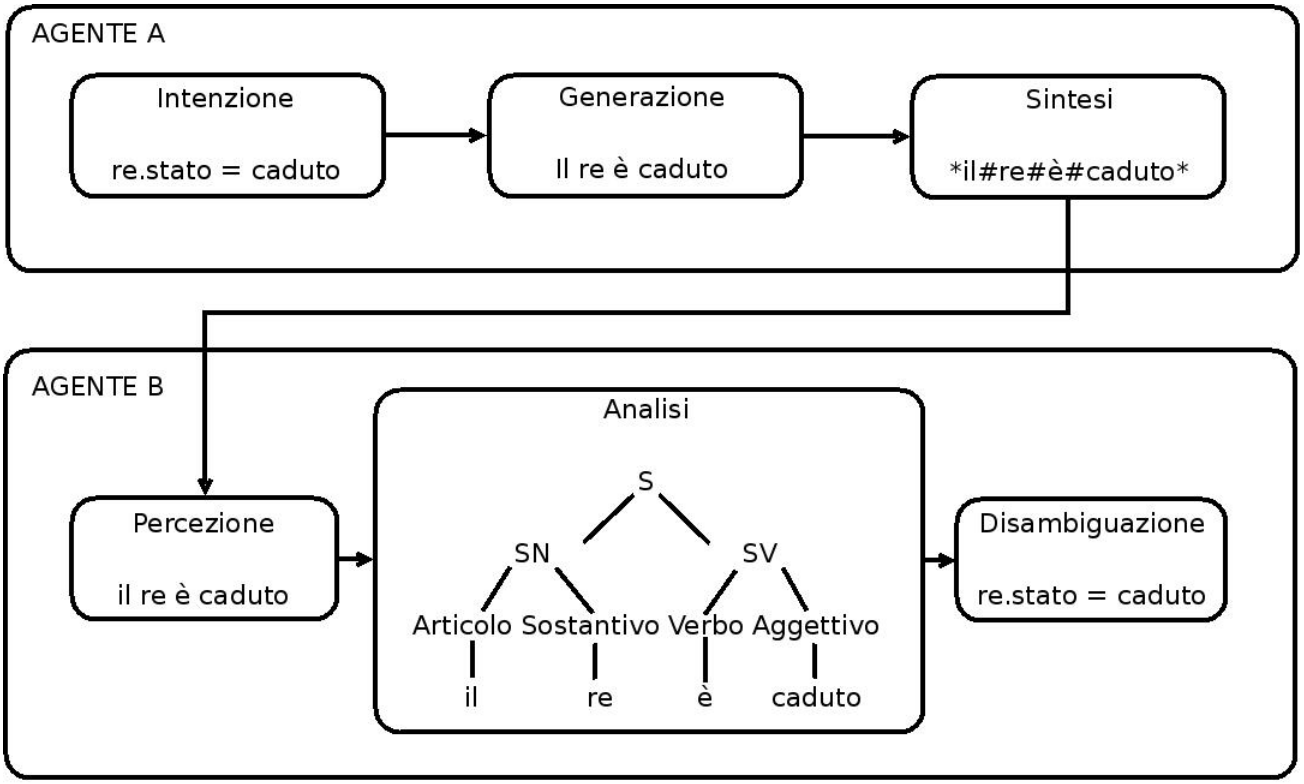


Figura 2.1: Schema di una comunicazione tra agenti

La figura 2.1 mostra schematicamente le fasi che si susseguono al fine di realizzare una comunicazione completa tra i due agenti:

1. **Intenzione**: l'agente A decide di comunicare con B.
2. **Generazione**: l'agente A trasforma l'informazione, codificata nella sua base dati in un enunciato che, una volta percepito dall'ascoltatore nella situazione corrente, con buona probabilità gli farà dedurre il significato.

3. **Sintesi**: il parlante produce la realizzazione fisica delle parole. Questo può essere fatto in svariati modi a seconda di ciò che gli attuatori consentono di fare. Ad esempio inchiostro su carta, vibrazioni nell'aria (suono), variazioni di luminosità o di tensione. Nell'esempio si aggiugono i simboli # e per segnalare il delimitarsi di una parola o di una frase.

4. **Percezione**: l'agente B, tramite i sensori, percepisce la realizzazione fisica delle parole e le decodifica. Se la comunicazione è vocale, tale passo prende il nome di **riconoscimento del parlato**, se è scritto **riconoscimento ottico**.

5. **Analisi**: l'agente B applica alla frase ottenuta la grammatica del linguaggio e procede alla costruzione dell'**albero sintattico** di cui parleremo nel paragrafo 2.4.

6. **Disambiguazione**: può capitare che alla stessa frase corrispondano più allberi sintattici diversi e quindi differenti possibili significati. In questi casi è necessario procedere a considerare gli aspetti pragmatici citati in precedenza. La scelta è guidata dalla probabilità con cui i diversi alberi possono corrispondere alla frase letta.

7. **Assimilazione**: una volta interpretato il significato di una frase, l'agente B può decidere se considerabile valida o meno la comunicazione, vale a dire aggiungere o meno l'informazione alla base di conoscenza oppure eseguire o meno l'azione richiesta.

Per completezza riportiamo di seguito una grammatica tale da riconoscere la frase del linguaggio vista in precedenza:

```
< frase > = < SN > < SV > |
                    < SV > < SN > |
                    < S > <Congiunzione > < S >
< congiunzione > = e | oppure | ma ...
< SN > = < Pronome > |
                 < NomeProprio> |
                 < Sostantivo > |
                 < Articolo > |
                 < Cifra > |
                 < SN > < PP > |
                 < SN > < PropCond >
< SV > = < Verbo > |
                 < SV > < SN > |
                 < SV > < Aggettivo > |
                 < SV > < PP > |
                 < SV > < Avverbio >
< PP > = < Preposizione > < SN >
< Preposizione > = di | a | da | in | con | su | per | tra | fra
< PropCond > = che < SV >
< Pronome > = io | me | tu | te | lui | lei | egli | noi | voi | essi | loro
```

```
< Articolo > = il | lo | la | i | gli | le
< Cifra > = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
< NomeProprio > = ... | re | ...
< Aggettivo > = ... | caduto | ...
```

2.4 Analisi Sintattica

L'**analisi sintattica** (o **parsing**) è il processo atto ad analizzare uno stream continuo in input (letto per esempio da un file o una tastiera) in modo da determinare la sua struttura grammaticale.

Si può pensare al parser come a un agente software che ha per input un file di testo e produce in output un albero sintattico. La costruzione di tale albero dipende dalla grammatica che si utilizza per il riconoscimento del testo.

Il *parsing* può essere considerato un processo di ricerca di un albero sintattico. Ad ogni passo, vengono, infatti, esplorate le regole di produzione fornite dalla grammatica al fine di scegliere quella più opportuna da utilizzare. Esistono due metodologie nella scelta delle regole da utilizzare ad ogni passo di derivazione:

- **derivazione canonica sinistra**: si sceglie sempre di sviluppare il simbolo non terminale più a sinistra. Tecnica più usata perché riflette il metodo di lettura più diffuso (da destra verso sinistra).
- **derivazione canonica destra**: si sceglie sempre di sviluppare il simbolo non terminale più a destra.

Scegliere una delle due tecniche è fondamentale perché evita la creazione di alberi sintattici errati e migliora l'efficienza del parsing stesso.

Le grammatiche che si utilizzano per la rappresentazione della sintassi sono **grammatiche non contestuali** (CFG) che si differenziano dalle altre per la tipologia di regole di derivazione. Queste ultime dovranno avere la seguente caratteristica: *la parte sinistra della regola deve essere un unico simbolo non terminale e la parte destra una sequenza qualsiasi di simboli terminali e non senza vincoli sulla loro disposizione.*

Esistono due diversi approcci nella realizzazione di un parser che dipendono dal modo con cui viene costruito, durante l'analisi dell'input, l'albero sintattico:

- **top-down**: l'albero viene costruito a partire da un simbolo non terminale, assioma, e si aggiungono nodi in relazione alla parola dell'input puntata. Per ogni simbolo terminale che si andrà a trovare nelle regole di produzione si sposterà il puntatore di lettura sulla parola successiva.
- **botton-up**: l'albero viene costruito leggendo un simbolo terminale per scegliere quale regola di produzione utilizzare per creare i nodi dal basso verso l'alto fino ad arrivare al nodo radice, l'assioma.

La realizzazione del parser è una semplice applicazione di sintesi di codice. Il suo funzionamento è intrinsecamente legato alla definizione della grammatica stessa. Esistono molto tool in grado di generare un parser dalle specifiche della grammatica.
I più comuni sono:

- antlr
- javacc
- bison
- flex
- yacc
- lex

2.4.1 Parsing Top-Down

Un parser **top-down** è un parser che costruisce l'albero sintattico a partire dalla radice. Lo può pensare come un software di ricerca che opera nel seguente modo:

1. **stato iniziale**: si parte da un albero avente un unico nodo, l'assioma della grammatica. Lo stato è rappresentato da un albero sintattico che si evolve nel tempo.
2. **funzione successore**: sceglie nell'albero il nodo più a sinistra da sviluppare (ovvero un codo corrispondente a un simbolo non terminale senza figli), quindi cerca le regole nella grammatica che hanno come parte sinistra tale nodo. Per ogni regola trovata la funzione vengono creati tanti nodi figli quanti sono gli elementi nella parte destra della regola. La scelta della regola è un proglema molto importante e dipende da differenti fattori.
3. **test obiettivo**: verifica se le foglie dell'albero sintattico corrispondono esattamente alla stringa in input.

2.4.2 Parsing Botton-Up

Un parser **botton-up** è un parser che costruisce l'albero sintattico a partire dalle foglie per arrivare al nodo radice. Lo può pensare come un software di ricerca che opera nel seguente modo:

1. **stato iniziale**: si parte dalla lista di parole che compongono la stringa di input. Lo stato, in questo caso, è rappresentato da una lista di alberi sintattici che dovranno convergere tutti al nodo radice.

2. **funzione successore**: considera la posizione i-esima nella lista di alberi e la parte destra di ogni regola di produzione. Se la sottosequenza della lista di alberi che comincia con i corrisponde alla parte destra, viene sostituita da un nuovo albero la cui categoria è la parte sinistra della regola e i cui figli sono la sequenza stessa.

3. **test obiettivo**: verifica se le foglie dell'albero sintattico corrispondono esattamente alla stringa in input.

2.4.3 Problematiche del parsing

Esistono alcune problematiche riguardati l'analisi sintattica che possono portare a seri problemi:

- inefficienza
- ambiguità
- ricorsione a sinistra

Efficienza del parsing

Entrambe le tipologie di parser analizzate possono risultare inefficienti a casua di molteplici modi in cui si possono combinare più analisi sintattiche di sintagmi differenti. E' possibile perdere tempo cercando in porzioni non rilevanti dello spazio di ricerca. Il parsing top-down può generare noti intermedi che non potranno mai legare con nessun simbolo terminale nella stringa di input, mentre il botton-up può generare sottoalberi che non si potranno unire tra loro per arrivare all'assioma.

Questo aspetto appare evidente se si pensa al **backtrace**. Nel caso in cui venga provata una derivazione e si arrivasse a un simbolo terminale differente da quello puntato in input, è necessario tornare indietro e provare un altra regola di produzione.

Esiste una tecnica di parsing che si ispira alla **programmazione dinamica** che può migliorare l'efficienza dell'analisi. Il concetto è il seguente: *ogni volta che si analizza una sottostringa, si memorizzano i risultati in modo da non doverla rianalizzare in seguito.* Questa tecnica prende il nome di **chart parser**.

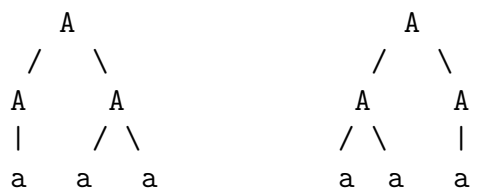
Ambiguità

Si genera un **ambiguità** quando alla stessa stringa di input corrispondono più alberi sintattici e quindi più interpretazioni semantiche diverse. Il problema non dipende dal parser ma dalla grammatica che viene utilizzata. Una grammatica è ambigua se possiede più derivazioni canoniche (destre o sinistre) per la stessa frase.

L'ambiguità di una grammatica è una **proprietà indecidibile**, pertanto non è possibile costruire un algoritmo che identifichi automaticamente se una grammatica è o no ambigua e l'eliminazione delle ambiguità dalla grammatica è un procedimento necessariamente manuale. Consideriamo il seguente esempio:

```
input: aaa

< A > = < A > < A > | a
```



Tale grammatica è ambigua. Dallo stesso input (aaa) si possono estrarre due alberi sintattici (destra e sinistra). Si può riscrivere la grammatica per renderla non ambigua e ottenere la seguente:

```
< A > = < A > < B > | < B >
< B > = a
```

oppure

```
< A > = < B > < A > | < B >
< B > = a
```

Ci sono casi in cui non si possono eliminare le ambiguità. Si utilizzano **grammatiche non contestuali probabilistiche** (PCFG) in cui si pesano con le probabilità le regole di produzione. La scelta dell'albero sintattico, quindi, è un applicazione di un problema probabilistico. E', infine, possibile modificare le probabilità attraverso tecniche di apprendimento.

Ricorsione a sinistra

Un problema tipico dei parser top-down è la ricorsione a sinistra. Questo può portare a loop infiniti dal quale il parser non potrà più uscire.
Consideriamo le seguenti regole:

1. `< A > = < A > x | y`
2. `< B > = < C >`
3. `< C > = < B > | y`

Consideriamo la prima regola: se si applica la procedura di derivazione canonica a sinistra il processo continuerà a sostituire la regola con se stessa senza uscire da questo ciclo infinito. Questa ricorsione prende il nome di **ricorsione diretta**. Le regole 2 e 3, invece, analizzate da sole non generano ricorsione infinita ma, quando la 2 viene applicata, dovrà essere applicata la 3 successivamente, che a sua volta richiamerà la 2. Questa ricorsione prende

il nome di **ricorsione indiretta**.
Rimuovere le ricorsioni è facile e si può applicare un algoritmo che va a modificare le regole di produzione come segue:

```
< A > = < A > x | y

si trasforma in:

< A > = y < B >
< B > = x < B > | < NULL >
```

2.5 Analisi Semantica

L'analisi semantica ha il compito di ha il compito di attribuire un significato all'albero sintattico ottenuto. In altre parole deve trovare un significato degli enunciati.

I significati, ovvero i sensi, spesso sono rappresentati tramite collezioni di sinonimi o synset. Un synset definisce un concetto noto all'uomo e classificato in una gerarchia di concetti. Un concetto può essere espresso tramite una o piu parole, note come lessicalizzazioni.

Attribuendo ai vari synset i codici univoci, si può classificare i concetti in una struttura reticolare con le relazioni, detta ontologia, ed arrivare a traduzione automatica che permette di passare da una lessicalizzazione ad altra.

La semantica è direttamente collegata con la rappresentazione della conoscenza, di cui parleremo nel capitolo 3. La si può vedere come una fase di traduzione volta a convertire un albero sintattico nel formalismo con cui vengono rappresentate le informazioni, obiettivi e azioni nell'agente stesso.

Spesso, l'analisi semantica contribuisce alla generazione di linguaggi attraverso l'uso di **grammatiche aumentate** o grammatica a clausole definite (DCG), che si occupano di etichettare le regole per poterle scegliere correttamente al fine di generare una frase che, oltre ad essere corretta sintatticamente, lo è anche semanticamente. Se ad esempio si volesse generare:

Io voglio il budino

si applicherebbe una struttura grammaticale simile alla seguente:

```
< pronome > < verbo > < nome >
```

Budino è chiaramente un nome e ne indica l'oggetto desiderato. Il verbo è espresso alla prima persona ed implica una volontà, quindi si sceglie voglio ma per il pronome? Qual'è la differenza tra Io e Me? Il primo è utilizzato in caso di soggetto, il secondo in caso di oggetto. Si potrebbe scegliere di utilizzare regole aggiuntive ma ciò renderebbe molto complicata la grammatica. L'alternativa è quella di effettuare annotazioni sulle regole stesse. Riprendendo la grammatica vista in precedenza, si modificherebbe come segue:

```
< frase > = < SN(Soggetto) > < SV > | ...
< SN(Type) > = < Pronome(Type) > |
                  ...
                  < SN > < PP(Type) >
< SV > = < Verbo > |
                  < SV > < SN(Oggetto) > |
                  ...
                  < SV > < PP(Oggetto) >
< PP > = < Preposizione > < SN(Oggetto) >
< Pronome(Soggetto) > = io | tu ...
< Pronome(Oggetto) > = me | te ...
```

Un metodo alternativo è rappresentato dagli **schemi di traduzione** che rappresentano una notazione per collegare particolari azioni, sotto forma di istruzioni, in un qualche linguaggio di programmazione, alle regole di produzione di una grammatica. Quando viene scelta una regola durante l'analisi della frase, viene svolta l'azione ad essa associata. Il risultato della generazione è, quindi, il risultato dell'esecuzione di tutte le azioni combinate fra di loro nell'ordine indotto dai costrutti della grammatica.

2.6 Induzione di grammatiche

L'**induzione di grammatiche** è il processo di apprendimento di una grammatica partendo dai dati. E' un'attività che viene spontaneo tentare, visto che costruire grammatiche a mano è molto faticoso mentre sulla rete sono disponibili miliardi di enunciati esemplificativi gratuiti. Il compito è difficile poiché lo spazio delle possibili grammatiche è infinito e verificare che una data grammatica generi un particolare insieme di frasi è computazionalmente molto costoso.

Un modello interessante è quello del sistema **SEQUITUR** (Nevill-Manning e Witten, 1997). Non è richiesto alcun input tranne un singolo testo (che non dev'essere precedentemente suddiviso in frasi). Il sistema produce una grammatica in forma molto specializzata, e precisamente quella che genera una singola stringa: il testo originale. Si può dire che SEQUITUR apprende tanta grammatica quanta è strettamente necessaria ad eseguire l'analisi sintattica del testo di input.

SEQUITUR è basato sull'idea che una grammatica è buona quando è compatta. In particolare sono sempre rispettati i seguenti due vincoli:

1. Nessuna coppia di simboli adiacenti deve comparire più d' una volta nella grammatica. Se la coppia A B compare nella parte destra di più regole, la si deve sostituire con un nuovo simbolo non terminale C e aggiungere la regola C AB.

2. Ogni regola dev'essere usata almeno due volte. Se un simbolo non terminale C compare una sola volta in tutta la grammatica, si deve eliminare la regola in questione e sostituire il suo singolo uso con la parte destra.

Questi due vincoli sono applicati all'interno di una ricerca golosa che esamina l'input da sinistra a destra, costruendo incrementalmente la grammatica e imponendo il rispetto dei vincoli il più presto possibile.

Capitolo 3

Rappresentazione della Conoscenza

3.1 Ingegneria della conoscenza

Il processo generale di costruzione di una base di conoscenza prende il nome di **ingegneria della conoscenza** (knowledge engineering). Un ingegnere della conoscenza investiga un particolare dominio, impara quali concetti sono importanti e scrive una rappresentazione formale degli oggetti al suo interno e delle relazioni tra essi.

Tra i diversi progetti di ingegneria della conoscenza ci sono grandi differenze di contenuto, dimensione e difficoltà, ma tutti includono i seguenti passi.

1. **Identificare il compito della base di conoscenza**. L'ingegnere della conoscenza deve delineare la gamma di domande a cui la KB dovrà rispondere e le categorie di fatti disponibili per ogni specifica istanza di problema.

2. **Raccogliere la conoscenza rilevante**. L'ingegnere della conoscenza potrebbe già essere un esperto del dominio, o potrebbe lavorare insieme ad altri esperti per estrarre quello che sanno: quest'ultimo processo prende il nome di *acquisizione della conoscenza*. In questa fase non viene utilizzata una rappresentazione formale; lo scopo è comprendere l'entità della base di conoscenza, determinata dal suo compito, e capire come funziona effettivamente il dominio.

3. **Definire un vocabolario di predicati, funzioni e costanti**. A questo punto occorre tradurre i concetti importanti del dominio in nomi di simboli logici. Nel far questo sorgono molti problemi di stile: come lo stile di programmazione, anche quello di ingegneria della conoscenza può avere un impatto significativo sul successo di un progetto. Una volta fatte queste scelte, il risultato sarà un vocabolario che prende il nome di ontologia del dominio. Con la parola ontologia si intende una particolare teoria riguardante la natura dell'esistenza. Un'ontologia definisce le categorie di oggetti esistenti, ma non le loro specifiche caratteristiche e le relazioni tra esse.

4. **Codificare la conoscenza generale riguardante il dominio**. L'ingegnere della conoscenza scrive gli assiomi per tutti gli elementi del vocabolario. Questo definisce

precisamente (nei limiti del possibile) il significato di ogni termine permettendo all'esperto del dominio di verificare il contenuto. Spesso questo passo rivela malintesi o lacune nel vocabolario, che dovranno essere colmate tornando al passo precedente e ripetendo il processo iterativamente.

5. **Codificare una descrizione della specifica istanza del problema**. Se l'ontologia è ben definita, questo passo dovrebbe risultare semplice: si tratta di scrivere semplici formule atomiche che riguardano istanze di concetti che fanno già parte dell'ontologia. Per un agente logico le istanze dei problemi sono fornite dai sensori, e la base di conoscenza iniziale è riempita di formule aggiuntive allo stesso modo con cui i programmi tradizionali ricevono dati di input.

6. **Interrogare la procedura di inferenza e ottenere da essa risposte**. A questo punto si possono raccogliere i frutti del proprio lavoro: la procedura di inferenza, partendo dagli assiomi e dai frutti specifici del problema, sarà in grado di derivare i fatti che ci interessa conoscere.

7. **Fare il debugging della base di conoscenza**. Purtroppo le risposte saranno raramente corrette al primo tentativo. Per essere più precisi le risposte saranno quelle giuste per la base di conoscenza così com'è scritta.

AGGIUNGERE QUALCOSA

Capitolo 4

Apprendimento

Bibliografia

[1] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prantice Hall, 2003.

[2] A. Tacchella *Appunti del corso di Linguaggi e Traduttori 1*. 2007.

[3] Alan M. Turing. *Computing Machinery and Intelligence*. Mind, 1950.

Indice analitico