Appunti Intelligenza Artificiale

Luca Seggiani
2024

1 Lezione del 23-09-24

1.1 Introduzione

Il corso si pone di fornire un'introduzione, almeno a livello culturale, alla storia dell'intelligenza artificiale dalle origini fino ad ora, e alcune specifiche sui modelli più usati oggi.

1.1.1 Cos'è l'intelligenza artificiale?

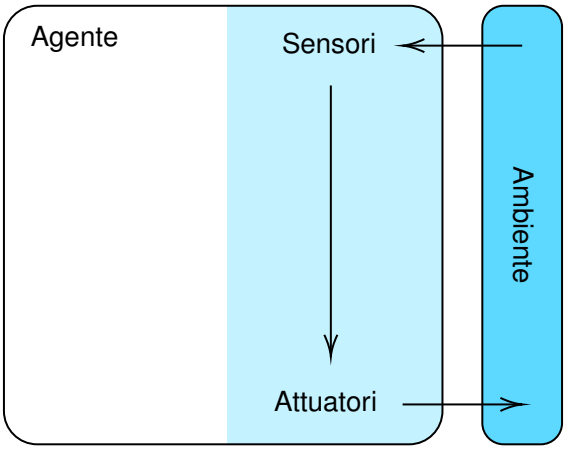
Con intelligenza artificiale ci riferiamo ad una serie di caratteristiche che vorremo ottenere da un particolare **agente**.

Definizione 1.1: Agente

Chiamiamo **agente** un'entità capace di:

- **Percepire**, attraverso *sensori*, informazioni dal suo ambiente esterno. Diremo sempre che una **percezione** segue dai rilevamenti dei sensori;
- **Agire**, attraverso *attuatori*, sull'ambiente esterno.

Graficamente, rendiamo l'agente attraverso lo schema:



Notiamo che nessun agente può esistere senza un'ambiente esterno, che sarà lo spazio dove i modelli che studieremo cercheranno di risolvere problemi.

Possiamo quindi dire che, da un agente **intelligente**, ci aspettiamo anche le proprietà di:

- **Imparare**, a partire da osservazioni, fino ad inferire leggi;
- **Ragionare** e **pianificare**, e quindi generare nuove informazioni;
- **Interagire**, e quindi scambiare informazioni, e modificare il suo ambiente.

I due modelli fondamentali che prenderemo in esempio sono il **cervello umano** e l'**agente razionale** (che potrebbe essere, volendo, anche più razionale della sua controparte umana).

Faremo poi una distinizione fra **pensiero** e **comportamento**: non è sempre rilevante farci domande sulla coscienza o meno del modello preso in analisi nel prendere le sue decisioni.

Possiamo quindi distribuire queste caratteristiche, fra di loro ortogonali, a formare quattro categorie:

Sistemi che pensano come umani	Sistemi che pensano razionalmente
Sistemi che si comportano come umani	Sistemi che si comportano razionalmente

Uno dei test più famosi per distinguere il comportamento di un programma *intelligente* è il cosiddetto **test di Turing**, il cui predicato è che per essere intelligente, un programma dovrebbe essere indistinguibile, a un osservatore esterno che comunica per via scritta, da un essere umano. Alcuni ricercatori inseriscono poi nel test di Turing altre facoltà, quali quelle di comunicazione in linguaggio naturale, interazione fisica con l'ambiente ecc... In ogni caso, per passare il test di Turing nella sua formulazione originale, un programma avrebbe bisogno di:

- Elaborare il **linguaggio naturale**, quindi comprendere espressioni e generarne di nuove;
- **Rappresentare l'informazione** che conosce, attraverso un qualche tipo di **knowledge base**;
- **Ragionare automaticamente** sulle informazioni ottenute, attraverso algoritmi di ricerca, algoritmi di inferenza, ecc...;
- **Apprendere automaticamente** nuove informazioni, attraverso varie rappresentazioni dell'ambiente esterno, tecniche di **machine learning**, ecc....

Questi, ad oggi, rappresentano i rami principali dell'IA, almeno a livello di agenti software. Altre aree di sviluppo sono rappresentate dalla **visione artificiale**, dalla **robotica** (ad esempio per creare robot intelligenti, capaci di muoversi nello spazio), ecc...

1.1.2 Pensare come umani

Per pensare come umani, dobbiamo prima capire come pensano gli umani. Alcune tecniche possono essere:

- **Introspezione**, cioè cercare di catturare i pensieri mentre ci passano per la testa;
- **Esperimenti** fisiologici o sociologici, che si concentrano invece sull'effetto che il pensiero ha sul mondo esterno, sotto forma di azioni;
- **Imaging** cerebrale (risonanza magnetica, ecc..), che mette in evidenza cosa accade fisicamente all'interno del cervello mentre si pensa.

Ammesso di poter estrapolare qualcosa di utile da questi strumenti, dobbiamo poi ricordare che gli umani possono essere irrazionali, commettere errori, e generalmente non sono agenti perfetti.

Inoltre, spesso nell'ingegneria ci *ispiriamo* alla natura, per poi ricavare sistemi migliori, più ottimizzati o più semplici. Un'ingegnere aerospaziale progetta aeromobili che possono volare grazie a propri principi specifici, e non macchine che si comportano come *uccelli*.

1.1.3 Agenti logici

Invece di pensare come umani, potremmo decidere di pensare in modo razionale. Questo richiederebbe scoprire **leggi di pensiero** definite a priori: quelle che definisce la logica. Chiamiamo **agente logico** un modello che applica queste leggi, o regole, per ottenere uno scopo prestabilito.

Modelli di questo tipo sono molto abili a formulare soluzioni di problemi matematici, e in generale a svolgere compiti dove l'ambiente esterno è le azioni concesse sono completamente conosciute. Il loro problema invece è che non sono in grado di rispondere a informazioni incerte: ogni ambiente reale invece, avrà inevitabilmente un certo grado di incertezza nella sua rappresentazione. Inoltre, cercare sempre la risposta ottima potrebbe richiedere troppe risorse dal punto di vista computazionale per essere effettivamente utile.

Si descrive così il classico diverbio fra *fare le cose in modo perfetto*, cioè applicando leggi matematiche potenzialmente poco pratiche alla risoluzione di problemi, e *fare le cose alla meglio*, cioè usando la statistica, varie euristiche e a volte addirittura la scelta casuale per ottenere soluzioni che non sono le ottime, ma le migliori possibili sotto determinati vincoli.

Paradossalmente, sembrerebbe che i nostri cervelli adottano la seconda filosofia. Infatti, in passato, le ricerche si sono concentrate a lungo sul creare agenti perfettamente razionali con risultati moderati, mentre solo di recente si ha avuto una spinta nel creare modelli più approssimativi, ma che sono risultati più promettenti nella simulazione di un comportamento, effettivamente, *umano*.

Un modello migliore potrebbe essere quello di un **agente razionale**, che si comporta in maniera tale da ottenere il miglior possibile risultato, ovvero *fa la cosa giusta*.

1.2 Agenti razionali

Abbiamo detto che un agente è un'entità che **percepisce** (attraverso sensori) e **agisce** (attraverso attuatori). L'agente razionale cerca di adottare le soluzioni più ottime possibili in ogni dato momento, basandosi su quello che può ragionevolmente assumere dall'ambiente esterno. L'agente razionale non è **perfetto**, cioè non conosce sempre la soluzione migliore, e non è **onniscente**, cioè non prevede il futuro.

Per effettuare i suoi compiti, diversi agenti adottano diverse strategie. L'approccio generale è quello di selezionare l'azione che massimizza la **utilità aspettata**, cioè l'opzione che appare, localmente, di portare un maggiore grado di *felicità* rispetto alle altre.

Un modello di questo tipo non usa sempre l'inferenza logica, ma a volte adotta linee di pensiero più intuitive: conviene togliere la mano dalla pentola bollente prima di pensare alle conseguenze. Inoltre, a volte non c'è una cosa migliore da fare, ma bisogna comunque fare una scelta.

Infine, l'ultima differenza fra questi modelli e i modelli logici è che decidiamo, di interessarci più a come il programma agisce, e non al modo in cui "pensa" ragionevolmente al modo di approcciare il problema.

1.2.1 Operazioni

Per permettere al programma di agire ragionevolmente, si stabilisce una **metrica di performance**, che valuta la sequenza di azioni operate dall'agente sull'ambiente. Possiamo quindi descrivere un operazione effettuata dall'agente attraverso:

- Una metrica di performance;
- L'ambiente su cui viene effettuata;
- Attuatori e sensori dell'agente che la effettua.

Chiamiamo questo schema **PEAS**, dall'inglese *Performance measure, Environment, Actuators and Sensors*.

Ad esempio, per un modello di guida automatica, la metrica sarà la velocità, la sicurezza, il comfort dei passeggeri, ecc... l'ambiente sarà la strada, gli attuatori saranno i comandi della vettura e i sensori saranno telecamere o radar posti in maniera tale da consentire la vista (*computer vision*) della strada, oppure ancora saranno l'odometro della vettura, eventuali accellerometri, ecc...

1.2.2 Ambiente

Abbiamo visto come l'ambiente è una componente fondamentale delle operazioni dell'agente. Questo può essere così caratterizzato:

- **Osservabile / parzialmente osservabile:** se i sensori dell'agente forniscono allo stesso una vista completa di tutte le variabili dell'ambiente in qualsiasi momento, abbiamo che quest'ultimo è completamente **osservabile**. Questi ambienti sono utili, in quanto permettono all'agente di non mantenere una rappresentazione interna dell'ambiente al suo interno. Nella stragrande maggioranza dei casì, però, i sensori saranno imprecisi, i problemi difficili da osservare se non semplicemente troppo grandi da essere completamente osservabili. In questo caso si dicono **parzialmente osservabili**, con diverse gradazioni di **osservabilità**. Il caso più radicale è quello di ambienti **non osservabili**, cioè dove l'agente non può usare sensori in primo luogo.
- **Deterministico / non deterministico / stocastico / strategico:** se il prossimo stato dell'ambiente è completamente determinato dal suo stato attuale e dalle azioni dell'agente, si ha che l'ambiente è **deterministico**. In teoria, un'ambiente osservabile e deterministico non ammette incertezze, mentre ambienti non osservabili potrebbero *sembrare* non deterministici, solo per la mancanza di informazioni sul loro conto. Si fa una nota fra i significati di non deterministico, **stocastico** e **strategico**:
  - **Non deterministico:** si dice di ambienti che possono presentare diverse e imprevedibili possibilità future;
  - **Stocastico:** si dice di ambienti che possono presentare diverse probabilità future, di cui si conoscono le probabilità;
  - **Strategico:** si riferisce a problemi, sopratutto multiagente, dove il modello deve utilizzare approcci strategici per comportarsi nel modo migliore possibile in previsione di risultanti imprevedibili, da parte dell'ambiente come degli altri agenti.

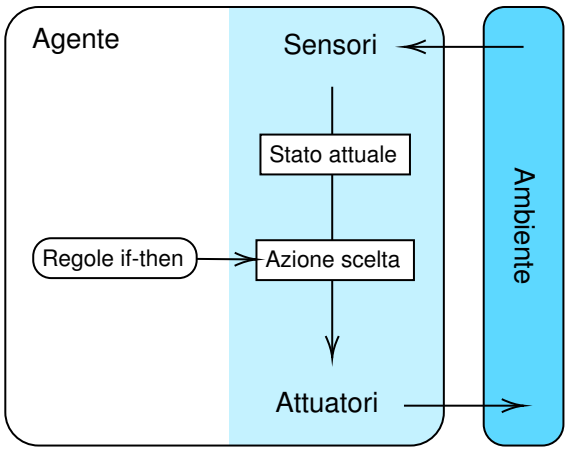
- **Episodico / sequenziale:** in un ambiente episodico, l'esperienza del modello è divisa in episodi separati. Le azioni intraprese in un episodio non hanno importanza negli episodi successivi. Al contrario, in un ambiente sequenziale ogni decisione corrente potrebbe influenzare il futuro (vedi sopra, ambiente strategico);
- **Statico / semidinamico / dinamico:** si riferisce alla temporizzazione dell'ambiente. Un ambiente statico non varia mentre l'agente sta deliberando, mentre un'ambiente dinamico cambia continuamente e richiede risposte tempestive del modello. Un ambiente semidinamico, in particolare, non varia di per sé, ma spinge comunque l'agente ad agire il più velocemente possibile (ad esempio, per rispettare un tempo limite);
- **Discreto / continuo:** si riferisce sempre alla temporizzazione dell'ambiente, e al modo in cui è disposto il suo spazio. Giochi come gli scacchi hanno divisioni distinte di tempo e spazio (mosse e caselle), mentre la maggior parte dei problemi reali si svolge su ambiente continuo (output in PWM ad attuatori, sensori che restituiscono distanze in virgola mobile, ecc...);
- **A singolo agente / multiagente:** un ambiente a singolo agente vede il modello agire da solo nella risoluzione del problema. Un modello multiagente invece prevede più agenti che **competono** o **collaborano** per un obiettivo.

Ad esempio, gli scacchi sono un'ambiente completamente osservabile, strategico, sequenziale, semistatico (diventa statico senza un orologio), discreto e multiagente. Il gioco del poker, invece, è solo parzialmente osservabile, strategico e stocastico, sequenziale, statico, discreto e multiagente. Infine, l'esempio del taxi di prima era parzialmente osservabile, stocastico, sequenziale, dinamico, continuo e multiagente.

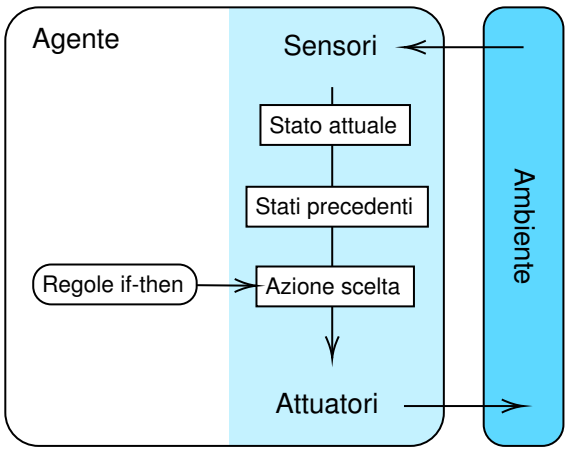
1.2.3 Tipi di agenti

Si può decidere su vari modi di strutturare un agente:

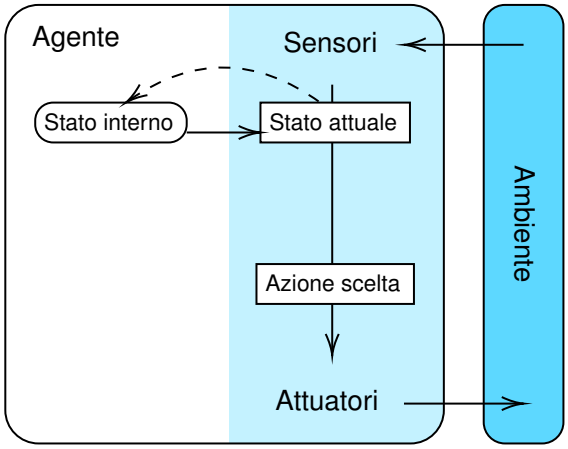
- **A riflesso semplice:** usano regole "if-then", applicate sulla loro percezione attuale dell'ambiente (ad esempio, il termostato: **se** la temperatura è sotto x, **allora** accendi il riscaldamento). Non dispongono di memoria, e quindi hanno una visibilità limitata del passato e del futuro.



- **A riflesso con stati:** simili al tipo precendente, ma con la caratteristica aggiunta che possono immagazzinare informazioni osservate precedentemente, e quindi ragionare su aspetti non più osservabili dello stato corrente (ad esempio, un robot aspirapolvere, che può ricordare la posizione degli ostacoli dopo averli incontrati la prima volta).

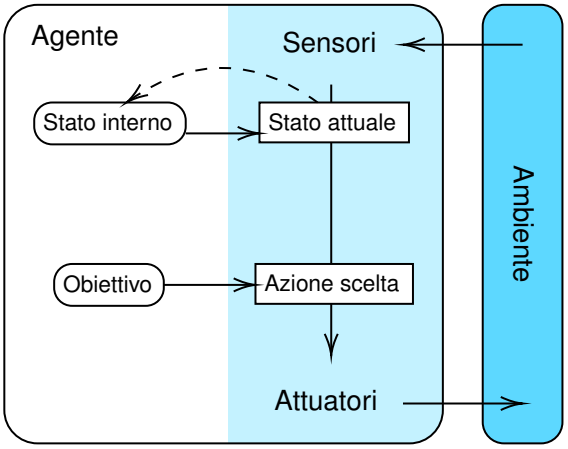


- **Basati su modelli:** anziché basarsi soltanto sugli input correnti dei sensori, questi modelli mantengono una rappresentazione interna dell'ambiente esterno, che aggiornano sulla base degli input dei sensori, e che usano per fare decisioni migliori:

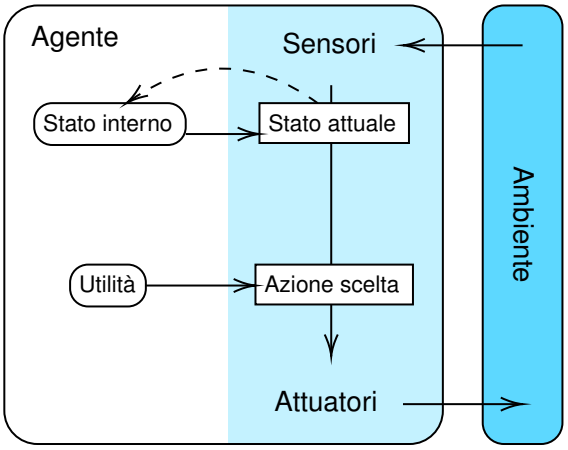


  Si dividono in:

  – **Basati su obiettivi:** dispongono di obiettivi che riflettono i loro "desideri". Per fare ciò, dispongono di informazioni riguardo a quello che le loro azioni fanno all'ambiente esterno, testano queste azioni sul loro modello interno dell'ambiente, e scelgono l'azione che li porta più vicino all'obiettivo (ad esempio, un'automobile a guida autonoma, che decide dove svoltare, quando frenare, ecc... con l'obiettivo di portare l'utente a destinazione).



  - **Basati sull'utilità:** dispongono di una funzione che valuta l'utilità, cioè il livello di *felicità* percepita, $f(\text{stato}) \to$ valore che cercano di massimizzare, confrontandosi con la loro rappresentazione interna in modo simile ai modelli basati su obiettivi (ad esempio, un modello decisionale che gioca in borsa: la funzione utilità rappresenterà il profitto, che verrà massimizzata tenendo conto di rischi e guadagni). Visto che non valutano soltanto se un azione li porterà o meno all'obiettivo sperato, ma fanno diverse considerazioni sul vantaggio relativo dato da più opzioni, sono solitamente migliori dei modelli basati su obiettivi.

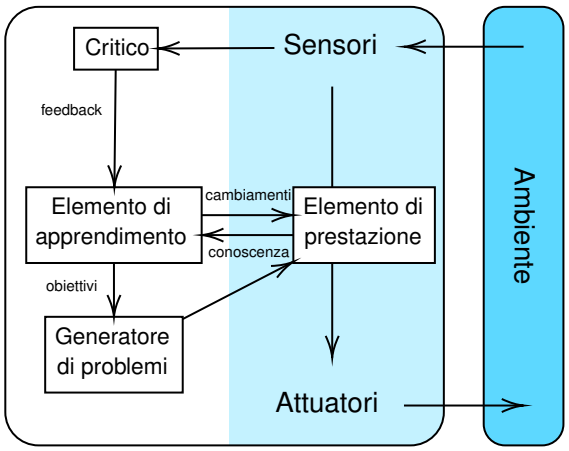


- **Capaci di apprendere:** implementano cicli di feedback interni che li spingono a rispondere in modo dinamico alle variazioni dell'ambiente e dell'utilità percepita: possono quindi adattare il loro comportamento a situazioni impreviste o in continua evoluzione (ad esempio, un assistente vocale che impara il modo di parlare, o le preferenze, ecc... del suo utente).

  Nel dettaglio, questi modelli sono formati da:

  - **Elemento di prestazione:** il componente che effettua le decisioni vere e proprie dell'agente, cioè il modello che vogliamo andare a migliorare;
  - **Critico:** un componente che si basa sui sensori e sulla metrica di performance usata per valutare il comportamento del modello;
  - **Elemento di apprendimento:** un modello che si esamina il **feedback** del critico, e usa i risultati trovati per migliorare l'elemento di prestazione.

– **Generatore di problemi:** un componente che si occupa di suggerire azioni che potrebbero aumentare la gamma di informazioni disponibile al modello.



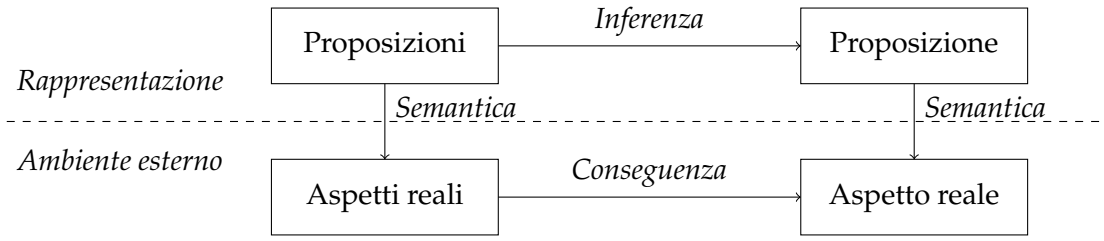
Oggi, la maggior parte dei modelli sul mercato è data da modelli basati su modelli e capaci di apprendere: la maggior parte della funzionalità del modello viene a svilupparsi proprio nella fase di apprendimento.

2 Lezione del 30-09-24

2.1 Agenti logici

Vediamo adesso l'implementazione di **agenti logici**, cioè agenti che si basano su una certa **rappresentazione**, detta **base di conoscenza** (KB, *Knowledge Base*) per immagazzinare **proposizioni** su ciò che hanno imparato riguardo all'ambiente esterno, e fare **inferenze**, sulla base di queste proposizioni, rispetto a informazioni non conosciute. Le proposizioni sono legate ad aspetti reali dell'ambiente esterno mediante una determinata **semantica**.

Possiamo schematizzare il funzionamento della KB, e la sua corrispondenza con l'ambiente esterno, come segue:



La conoscenza che l'agente ottiene può essere fornita manualmente, cioè in forma di **assiomi** o conoscenza di *background* (magari in un agente potrebbe essere "precaricata" della conoscenza riguardo allo stato iniziale del problema), estratta dai dati dei sensori, o ricavata dall'esperienza. Un agente logico dovrebbe essere in grado di fare inferenze in quanto spesso le informazioni che ha sull'ambiente esterno è **parziale** o **incompleta**.

Una KB deve poi rappresentare questa conoscenza attraverso un linguaggio, che dovrebbe essere sufficientemente **espressivo** da poter rappresentare la realtà dell'ambiente esterno, ma non troppo complesso da impedire di effettuare inferenze in modo efficiente.

Esistono due approcci all'implementazione di una KB:

- **Dichiarativo:** si concentra su una rappresentazione a **livello di conoscenza** dei fatti, cioè informazioni riguardo a *cosa* è vero. In sistemi di questo tipo, si usano primitive di scrittura (TELL) e query (ASK) sulla KB, partendo da un insieme di informazioni nullo o comunque limitato, fino ad arrivare ad avere una serie di conoscenze comprensive dell'ambiente esterno. I dettagli delle operazioni che poi l'agente andrà a svolgere, il cosiddetto **livello di implementazione**, sono mantenuti separati dalla KB.
- **Procedurale:** si concentra su una rappresentazione di *come* effettuare operazioni. Invece di implementare sistemi che possano immagazzinare proposizioni sull'ambiente esterno, si va a codificare l'informazione direttamente nel codice, attraverso procedure, algoritmi, o come avevamo visto nei modelli a riflesso, regole "if-then".

2.1.1 Basi di conoscenza

Come abbiamo detto, una base di conoscenza è formata da una serie di formule (formule *atomiche*, cioè proposizioni) contenenti informazioni riguardo all'ambiente esterno e codificate in un certo **linguaggio formale**. Si possono definire alcune primitive per l'interazione con la KB:

- **TELL**: aggiungi una nuova proposizione alla KB;
- **ASK**: richiedi informazioni dalla KB;
- **RETRACT**: elimina informazioni dalla KB.

La KB si basa sui fatti che già conosce per ricavare inferenze o **deduzioni logiche** α, della forma:

$$\text{KB} \models \alpha$$

L'agente logico si interfaccia con la KB attraverso le primitive sopra definita, e implementa effettivamente un ciclo TELL-ASK-TELL del tipo:

Algoritmo 1 Agente logico

Input: le percezioni correnti
Output: la prossima azione da eseguire
TELL(percezioni correnti)
prossima azione $\leftarrow$ ASK(KB)
TELL(prossima azione)
return prossima azione

Ovvero, l'agente invia le sue percezioni correnti alla KB, e richiede la prossima azione da eseguire. Invia poi l'azione scelta alla KB (così che possa diventare parte delle informazioni note), e la restituisce.

2.1.2 Differenza fra KB e DB

Una KB potrebbe sembrare simile ad un comune database: la differenza è che il database si occupa solo di ricavare fatti specifici, senza possibilità di deduzione di alcun tipo. La KB è invece progettata per mantenere una rappresentazione strutturata dei fatti, specifici o generali, come riferiti a oggetti reali, e permettere quindi inferenze su quei fatti.

2.2 Logica proposizionale

Un tipo di logica piuttosto limitato ma comunque utile è la **logica proposizionale** (detta anche *logica di ordine zero*).

- **Sintassi:** la logica proposizionale comprende di **proposizioni**, cioè simboli (vengono infatti detti anche **simboli di proposizione**) che possono assumere un valore vero o falso, e di **formule** o *espressioni*, formate dall'applicazione di operatori a altre formule, a proposizioni o a valori assoluti vero o falso, e cosi via ricorsivamente. Gli operatori previsti sono quelli classici, in ordine di precedenza da massima a minima:
  - **Negazione logica:** indicata con $\neg$, ad arietà unaria;
  - **Congiunzione logica:** indicata con $\wedge$, ad arietà binaria;
  - **Disgiunzione logica:** indicata con $\vee$, ad arietà binaria;
  - **Implicazione logica:** indicata con $\Rightarrow$, ad arietà binaria. Il membro sinistro si dice **antecedente**, e il membro destro **conseguente**;
  - **Coimplicazione logica:** indicata con $\Leftrightarrow$, ad arietà binaria.

  In particolare, chiamiamo **letterale** un simbolo di proposizione o la sua negazione.
- **Semantica:** una volta stabilita una sintassi, è necessario assegnare un significato **semantico** ai simboli che essa comprende: abbiamo detto che una proposizione rappresenta un dato sull'aspetto reale attraverso la sua semantica. In particolare, introduciamo il concetto di **modello** come possibile sostituzione di valori vero o falso a ogni simbolo di proposizione. Si ha quindi che ogni simbolo rappresenta semanticamente il valore di verità che assume in un dato modello m. Si possono quindi definire i significati semantici degli operatori:
  - **Negazione logica:** $\neg P$ è vera se P è falsa in m;
  - **Congiunzione logica:** $P \wedge Q$ è vera se P e Q sono entrambe vere in m, falsa altrimenti;
  - **Disgiunzione logica:** $P \vee Q$ è falsa se P e Q sono entrambe false in m, vera altrimenti;
  - **Implicazione logica:** $P \Rightarrow Q$ è falsa se P è vera e Q falsa in m, vera altrimenti;
  - **Coimplicazione logica:** $P \Leftrightarrow Q$ è vera se P e Q sono entrambe vere o entrambe false in m, falsa altrimenti.

  Notiamo che implicazione logica e coimplicazione logica non hanno molto significato a livello di singoli modelli, ma sono più utili a livello di **validità** (verità su tutti i modelli).

  Si nota inoltre l'esistenza dell'operatore di **disgiunzione esclusiva** indicato con $\oplus$ o *xor*. Non useremo esplicitamente questo operatore, e se lo faremo basterà la definizione come $P \oplus Q \equiv \neg(P \Leftrightarrow Q)$.

Quello che vogliamo fare una volta stabilita una logica formale è trovare i modi in cui si può estrarre **implicazione** a partire da formule logiche. Inoltre, vogliamo determinare se gli algoritmi che usiamo per svolgere questa operazione sono **validi**, cioè rispettano le regole della logica e ricavano implicazioni **vere** (tutti gli approcci che consideremo sono

validi), e **completi**, cioè capaci di derivare *tutte* le possibili implicazioni logiche di un insieme di formule (effettivamente una knowledge base), e quindi capaci di dimostrare, se implicabile, l'implicabilità di qualsiasi forma logica. Per quanto riguarda la logica proposizionale, abbiamo tre approcci principali a disposizione:

- **Verifica dei modelli**, in inglese *model checking*: si verifica il valore di verità di una formula su tutti modelli possibili (o, più intelligentemente, su un sottoinsieme di essi), in modo da arrivare ad una verifica per "forza bruta" della formula. Questi metodi si distinguono nello specifico in:
  - **Enumerazione totale:** si verifica la totalità dei modelli (è quindi un approccio **completi**), eventualmente impiegando algoritmi ricorsivi dotati di *backtracking*, cioè la capacità di tornare ad una soluzione precedente nel caso la regione esplorata finora si dimostri vuota di soluzioni valide. Un algoritmo di questo tipo e l'algoritmo di **Davis-Putnam**.
  - **Ricerca locale:** si verifica un sottoinsieme dei modelli, attraverso una qualche euristica, cercando di trovare modelli che soddisfano la formula data. Questi sistemi non saranno mai, ovviamente, completi (si può essere *quasi* sicuri che una formula sia vera o falsa, ma mai *completamente*), ma sono comunque utili nel campo dei modelli logici dove verificare con completezza ogni formula risulterebbe troppo dispendioso in risorse. L'euristica, in questo caso, viene data da una certa **funzione obiettivo** da massimizzare, che può essere ad esempio il numero di clausole rimaste insoddisfatte nella formula. Esempi di algoritmi di questo tipo sono quindi:
    * **Annealing simulato:** ispirandosi al comportamento di un metallo che viene portato a temperature elevate per portare la sua configurazione molecolare ad un punto di equilibrio stabile più basso, si modificano casualmente i valori di verità delle proposizioni nel modello, accettando le soluzioni che aumentano la funzione obiettivo e, entro un certo margine, che la diminuiscono (nella speranza che compromessi sul corto termine possano portare a guadagni sul lungo termine).
    * **WalkSAT:** è un algoritmo che alterna fra un comportamento casuale (modificare una variabile nel modello ad arbitrio) e probabilistico (modificare una variabile che, statisticamente, può migliorare la funzione obiettivo). Algoritmi di questo tipo si rivelano, nelle loro limitazioni, molto efficienti per la valutazione di formule logiche.
- **Deduzione automatica**, in inglese *theorem proving*: si applicano **regole di inferenza** alle formule, in modo di dimostrarne la **validità**. In particolare, questo torna utile in quanto l'implicazione può tradursi come:

  $$a \models b \Leftrightarrow a \Rightarrow b \text{ è valida}$$

  cioè, dimostrando che l'implicazione $a \rightarrow b$ è validà, cioè vera in tutti i modelli, allora effettivamente a implica b ($a \models b$) nell'ambiente esterno che volevamo modelizzare. Esistono diverse regole di inferenza usate nella dimostrazione dei teoremi, cioè **modus ponens**, **unificazione**, **instanziazione universale**, **reductio ad aburdum** ecc...
  - **Risoluzione:** una regola di inferenza che risulta utile nei sistemi di deduzione automatica è la **risoluzione**. Diciamo che due letterali sono **complementari** se

sono uno la negazione e uno l'affermazione così com'è di uno stesso simbolo di proposizione, cioè P è complementare a $\neg P$. Allora, si può dimostrare che:

$$\frac{l_1 \vee ... \vee l_k, \quad m_1 \vee ... \vee m_k}{l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{i-1} \vee m_{j+1} \vee ... \vee m_k}$$

Se esistono due letterali l_i e m_j fra di loro complementari. Chiaramente, questa regola si applica solo a disgiunzioni di letterali. Si introduce quindi il concetto di **CNF** (*conjunctive normal form*), ergo una congiunzione di disginuzioni di letterali:

$$d_1 \wedge ... \wedge d_k, \quad a_i = (l_1 \vee ... \vee l_n)$$

Esiste anche l'equivalente in disgiunzioni, la **DNF** (*disjunctive normal form*), che però non vediamo. In ogni caso, la conversione di una formula nell'equivalente CNF o DNF si fa attraverso un'algoritmo ricorsivo, applicando la proprietà distributiva alle disgiunzioni nel caso della CNF (congiunzioni ne caso della DNF), e applicando le leggi di De Morgan per portare le negazioni "in dentro" fino ai letterali. Una volta portata una formula in CNF, si può quindi applicare ricorsivamente la risoluzione alle clausole della CNF, che saranno, come volevamo, disgiunzioni di letterali.

– **Clausole di Horn:** approcci più efficienti, ma meno completi, sono dati dall'uso di clausole ristrette, fra cui ad esempio le **clausole di Horn**:

Definizione 2.1: Clausola di Horn

Si definisce clausola di Horn una disgiunzione di letterali di cui *al massimo uno* è positivo (non negato).

Le clausole di Horn sono un sovrainsieme delle cosiddette **clausole definite**, cioè disgiunzioni di letterali di cui *esattamente uno* è positivo. Le clausole di Horn permettono, a costo di minore espressività, di fare deduzioni in tempo lineare, attraverso due tipi di algoritmi:

* **Forward chaining:** si parte dai fatti che abbiamo a disposizione (le formule atomiche nella knowledge base) e si applicano le regole di inferenza (effettivamente il *modus ponens* sulle clausole di Horn) in modo da ricavare le possibili formule implicate dalla KB, fino a raggiungere la formula bersaglio, la cui validità vogliamo dimostrare. Per ottimizzare algoritmi di questo tipo si usano gli approcci di ricerca su alberi, in quanto quello che andiamo a costruire è un **albero and-or**.
* **Backward chaining:** si parte dalla formula bersaglio, e si lavora al contrario, cercando di soddisfarla trovando nella KB premesse valide. Un approccio di questo tipo, con backtracking nel caso di alberi non soddisfacenti, è quello usato dal linguaggio Prolog.

2.3 Logica del primo ordine

Una logica più potente ma più complessa della logica proposizionale è la **logica del primo ordine**. Nella logica del primo ordine prendiamo un *impegno ontologico* ed *epistemologico* più grande di quello che avevamo preso nella logica proposizionale: invece di **fatti** che possono essere veri o falsi, consideriamo **oggetti** in relazione fra di loro.

- **Sintassi:** la logica del primo ordine estende la logica proposizionale, introducendo i simboli di:
  - **Quantificatore universale:** si indica con $\forall x$, seguita dalle proprietà valide per x;
  - **Quantificatore esistenziale:** si indica con $\exists x$, seguita dalle proprietà valide per x.
- **Semantica:** Nella logica del primo ordine non parliamo di fatti, ma introduciamo le seguenti entità:
  - **Oggetti:** rappresentano le entità di cui parliamo nel nostro sistema formale;
  - **Relazioni:** predicati booleani su qualità unarie di un oggetto (è rosso, è grande, ecc...) o su relazioni n-arie fra oggetti (è fratello di, è supervisore di, ecc...);
  - **Funzioni:** relazioni che associano ad ogni elemento del dominio uno solo del codominio (la madre di, il migliore amico di, ecc...).

  Chiamiamo **predicati** le funzioni booleane, cioè le funzioni che verificano se una proprietà di uno o più oggetti è vera. I predicati saranno l'equivalente delle proposizioni che avevamo visto nella logica proposizionale.

  Una volta introdotta la nozione di *oggetti*, si ha che questi esistono all'interno di un **dominio**, e si possono quindi usare i cosiddetti **quantificatori**:
  - **Quantificatore universale:** significa che per ogni oggetto del dominio un dato predicato è vero;
  - **Quantificatore esistenziale:** significa che esiste almeno un oggetto del dominio per cui un dato predicato è vero. Quantificatori esistenziali più complicati come *esiste ed è unico* ($\exists!$), ecc... possono esprimersi comunque attraverso la logica del primo ordine (ad esempio $\exists x(P(x) \land \forall y(P(y) \Rightarrow y = x))$).

Come prima, siamo interessati a trovare modi per dimostrare l'implicazione logica di formule in logica del primo ordine a partire da una knowledge base di altre formule. Notiamo che in generale questo è più difficile di quanto lo era stato nella logica proposizionale, in quanto questa è solo **semidecidibile**: la possibilità di creare regole ricorsive rende l'implicazione decidibile, ma la non implicazione non sempre decidibile.

Si hanno quindi gli approcci:

- **Proposizionalizzazione:** l'approccio più immediato per la dimostrazione dell'implicazione nella logica del primo ordine è quello della conversione della formula in logica proposizionale. Questo è sempre possibile, instanziando le variabili nei quantificatori secondo le regole:
  - **Instanziazione universale:** si istanziano le variabili universalmente quantificati con **termini ground**, cioè che non contengono variabili (effettivamente riducendo la formula a una serie di termini che rappresentano fatti reali nella KB).
  - **Instanziazione esistenziale:** si istanziano le variabili esistenzialmente quantificate con un nuovo simbolo costante, detto **costante di Skolem** (motivo per cui il processo prende il nome di **skolemizzazione**).

Le instanziazioni sopra definite equivalgono effettivamente a **sostituzioni**, notate con la sintassi $\theta = \{x_1/k_1, ..., x_n/k_n\}$, di termini di ground o costanti di Skolem k_i alle variabili x_i di formule in logica del primo ordine.

- **Unificazione:** si può introdurre il *modus ponens generalizzato*: presa una formula logica data dall'implicazione attraverso n premesse (formule atomiche) di una conclusione q, cioè $p_1 \wedge ... \wedge p_n \Rightarrow q$, e una serie di altre formule atomiche $p'_1, ..., p'_n$, se si ha che se una sostituzione θ rende le p_i uguali alle p'_i, allora è implicata la sostituzione via θ di q:

$$\frac{p'_1, ..., p'_n, \quad (p_1 \wedge ... \wedge p_2)}{\text{subst}(\theta, q)}$$

  In altre parole, se sostituendo via θ una serie di proposizioni (quelle che avremo nella KB) si rendono uguali a quelle di ad un implicazione vera $p_1 \wedge ... \wedge p_n = q$, allora ciò che si ottiene dalla sostituzione θ in q è vero. Quest'operazione non è altro che il **lifting** del modus ponens alla logica del primo ordine: sostituendo termini di ground ad un implicazione di primo ordine si ottiene una forma del modus ponens che ci permette di mantenere la formula del primo ordine stesso (senza dover ricorrere alla proposizionalizzazione). Algoritmi di **unificazione** sfruttano questo principio per trovare il cosiddetto **MGU**, in inglese *Most General Unifier*, cioè la sostituzione θ più generale che rende $p_i = p'_i$.

- **Clausole definite:** possiamo sfruttare le clausole definite anche nella logica del primo ordine, e usare quindi algoritmi di **forward chaining** e **backward chaining**. Il processo è analogo a quanto avevamo fatto nella logica proposizionale:

  - **Forward chaining:** si trovano tutte le formule implicate attraverso sostituzioni θ, ergo tutte le formule implicate dagli antecedenti presi in considerazione. Come prima, questo significa muoversi potenzialmente verso direzioni non ottimali per la verifica di una formula particolare;
  - **Backward chaining:** si cercano di dimostrare le premesse della formula trovando sostituzioni θ efficaci nella KB.

- **Risoluzione:** si può estendere la risoluzione alla logica del primo ordine: attraverso la conversione in CNF che avevamo visto nella logica proposizionale, la skolemizzazione (o introduzione di *funzioni di skolem*), la rimozione dei quantificatori universali (le variabili universalmente quantificate saranno quelle che andremo a rimuovere per risoluzione), e un nuovo passo detto **standardizzazione delle variabili**, dove si cambia nome alle variabili quantificate omonime. A questo punto si ottiene una forma simile a quella che avevamo in logica proposizionale, sulla quale si possono eliminare letterali fra di loro complementari, in modo da ridurre fino dagli antecedenti fino al conseguente desiderato.