Robotica cognitiva

Combinare l'esperienza ed il buon senso con la pianificazione di azioni complesse: questo è l'obiettivo di questa branca dell'intelligenza artificiale, il cui scopo è gestire in maniera corretta il non determinismo e l'incompletezza che caratterizzano il mondo reale

di Marco Pirrone e Andrea Pompili

Marco Pirrone *è laureato in Ingegneria Informatica all'università "La Sapienza" di Roma. Attualmente svolge il dottorato di ricerca in Intelligenza Artificiale presso il Dipartimento di "Informatica e Sistemistica" della Sapienza ed in particolare si occupa di Visione Artificiale e Navigazione Robotica Autonoma. Può essere contattato per e-mail all'indirizzo mpirrone@infomedia.it*

Definire il concetto di "intelligenza" non è molto semplice, soprattutto se la definizione deve essere lo spunto per la realizzazione di un sistema autonomo complesso. L'intelligenza artificiale ha tentato (e ancora tenta) di dare una risposta a questa domanda formalizzando concetti e linguaggi anche piuttosto complessi.

In particolare, la tendenza di oggi è di concentrare l'intelligenza su problematiche astratte in modo da poter gestire l'indeterminatezza del mondo circostante e pianificare una lista di azioni complesse anche in situazioni di conoscenza non completa.

Daremo quindi un'introduzione alle finalità e alle problematiche sollevate dalla robotica cognitiva (etichetta sotto la quale si identificano tutti gli studi orientati a rispondere alle domande precedenti) e vedremo come in pratica questi temi potrebbero essere affrontati. Vista la vastità dell'argomento, daremo solo una presentazione formale di questo linguaggio, lasciando al lettore la scelta di approfondire il tema su come può essere poi formalizzato un agente che agisca in un mondo non deterministico.

AGENTI INTELLIGENTI E ROBOTICA COGNITIVA

La realizzazione di un sistema autonomo che compia determinate scelte in base all'ambiente circostante è uno degli obiettivi principali dell'intelligenza artificiale. Un'entità di questo tipo viene chiamata in gergo "agente" ed è descritta come un insieme di componenti specializzati che le consentono di interagire in maniera più o meno intelligente con il mondo esterno. La prima classe di elementi è quella degli attuatori, ossia quei sistemi elettromeccanici che consentono all'agente di manipolare il mondo esterno o modificare la propria posizione (ad esempio, bracci meccanici, ruote motrici, trivelle, ecc.). Tali elementi sono attivi, ossia modificano lo stato del mondo e possono essere sfruttati per risolvere situazioni critiche o raggiungere obiettivi prefissati.

La seconda categoria è quella dei sensori, ossia elementi passivi il cui scopo è quello di fornire informazioni sullo stato del mondo circostante (ad esempio, telecamere, sensori di posizione o tattili, sonar, ecc.). Un agente può avere poi più sensori ed attuatori secondo il grado di affidabilità voluto o la complessità delle attività da compiere.

Quello che manca è "l'intelligenza" o, più propriamente, il motore deduttivo che consenta all'agente di relazionare tra loro le informazioni ricevute dai sensori con quelle già in possesso ed attivare di conseguenza gli attuatori per raggiungere uno scopo autonomamente prefissato.

Una prima soluzione al problema è quella del *simple reflex agent*, ossia un agente che cortocircuita sensori ed attuatori mediante una logica decisionale scarna e semplificata. Tale approccio è simile a quello adottato dagli insetti in cui determinate azioni vengono compiute solo nel caso in cui si presentino delle condizioni particolari. Ad esempio è possibile istruire l'agente ad andare indietro con lo sterzo a destra ogni volta che un sensore di distanza indichi un ostacolo prossimo (questo comportamento è simile a quello di alcuni giocattoli tuttora in commercio). Ovviamente una soluzione di questo tipo è esageratamente semplificata ed è praticabile solo per agenti dedicati a compiti molto semplici e lineari.

Ben diversa è la situazione di un agente che abbia al suo interno una serie di affermazioni e regole che descrivano la realtà esterna e tutti i suoi possibili cambiamenti in relazione agli eventi manifestabili (base di conoscenza). In questo caso l'agente può usare i sensori per allargare o verificare la propria conoscenza e decidere le azioni da compiere basandosi su di essa; ad esempio può auto-imporsi un nuovo obiettivo o pianificare off-line una serie di azioni per raggiungerne uno già formulato, può agire direttamente in base a determinati eventi manifestati o attendere nuovi input dall'esterno, ecc.

Il più semplice esempio di agenti di questo tipo sono quelli basati sugli algoritmi di ricerca per la risoluzione automatica di problemi (ad esempio, *A**, *depth first*, ecc.). In questo caso l'agente mantiene una descrizione dello stato corrente del mondo esterno (più o meno completo) e trova mediante ricerca cieca o euristica una lista di azioni plausibile che raggiunga un determinato obiettivo (*goal*).

Il passo in più che trasforma un agente programmato in un sistema che agisce secondo una logica è la presenza nell'apparato decisionale di un motore di ragionamento adeguatamente affinato basato sulla deduzione.

In questo ambito si colloca la *robotica cognitiva*, ossia studiare formalismi e soluzioni che consentano ad un agente di ragionare ad alto livello su ciò

Andrea Pompili *è laureato in Ingegneria Informatica. Ha svolto attività di analisi e sviluppo per la realizzazione di portali di tipo enterprise su piattaforma Java. Attualmente lavora presso WIND Telecomunicazioni come referente per la sicurezza informatica. Può essere contattato per e-mail all'indirizzo apompili@infomedia.it*

Listato 1 Formalizzazione in Prolog del dominio di esempio utilizzando il calcolo delle situazioni come metodologia di rappresentazione

```
/* ------------------------- */
/* Modello del mondo dei blocchi            */
/* formalizzato nel Calcolo delle  Situazioni */
/* ------------------------- */

/* ------------------------- */
/* ASSIOMI STATO-SUCCESSORE                 */
/* ------------------------- */

/* Fluenti per determinare la posizione di un blocco */
on(X,Y,do(A,S)) :- A=put(X,Y);
                   not(A=put(X,Z)),
                   not(A=putOnTable(X)),
                   on(X,S).

onTable(X,do(A,S)) :- A=putOnTable(X);
                      not(A=put(X,Y)), onTable(X,S).

clear(X,do(A,S)) :- A=put(X,Z);
                    A=put(Y,Z), on(Y,X,S);
                    A=putOnTable(Y), on(Y,X,S);
                    not(A=put(Y,X)), clear(X,S).

/* ------------------------- */
/* ASSIOMI DI PRECONDIZIONE                 */
/* ------------------------- */
poss(put(X,Y),S) :- not(X = Y),
                    not(on(X,Y,S)),
                    clear(X,S),
                    clear(Y,S).

poss(putOnTable(X),S) :- not(onTable(X,S)),
                         clear(X,S).

/* ------------------------- */
/* SITUAZIONE INIZIALE E DEFINIZIONI        */
/* ------------------------- */

/* |abc     posizione iniziale dei blocchi   */
/* |d                                        */
/* |ef                                       */

on(b,a,s0).
on(c,b,s0).
on(f,e,s0).
onTable(a,s0).
onTable(d,s0).
onTable(e,s0).
clear(c,s0).
clear(d,s0).
clear(f,s0).

blue(e).
blue(b).
red(a).
red(c).
green(d).
green(f).
```

che è noto al fine di decidere autonomamente le proprie necessità (effettuare ulteriori *sensing* sul mondo utilizzando particolari sensori, oppure compiere delle azioni di test al fine di ottenere informazioni dai risultati ottenuti, ecc.).

ASTRAZIONE E ARCHITETTURE MULTI-LIVELLO

L'obiettivo principale della robotica cognitiva diventa quindi quello di formalizzare un livello di astrazione tale da consentire all'agente di non ragionare più sui movimenti elettrici o meccanici dei propri attuatori, ma su azioni complesse che consentano il raggiungimento di obiettivi più descrittivi. L'agente può eseguire azioni del tipo: posa l'oggetto *X* sopra l'oggetto *Y*, vai all'aeroporto, ecc. senza doversi preoccupare di cosa praticamente bisogna fare per compierle (ad esempio, muoversi avanti di 5 metri, prendere l'oggetto con la pinza *A*, alzare l'oggetto, ecc.). L'astrazione dall'ambiente fisico semplifica notevolmente la descrizione della realtà e consente all'agente di concentrarsi su obiettivi di alto livello e pianificare adeguatamente quelli che vengono chiamati *task* primari, ossia dei compiti astratti il cui soddisfacimento permette il raggiungimento di un obiettivo. Come poi vengano svolti con gli attuatori questi task non è importante a questo livello e verrà demandato ai sistemi di controllo sottostanti.

Una prima differenza fondamentale rispetto agli altri approcci è appunto quella di non definire il motore intelligente come un blocco monolitico che ragiona su tutti i piani della conoscenza, ma avere un'architettura a più livelli ciascuno con una propria base di conoscenza affinata in base ai compiti assegnati.

Ad esempio potremmo avere un robot mobile che ragioni su due livelli di astrazione: uno dedicato ai task fondamentali (vado prima nella stanza *X* o nella *Y*? passo per il corridoio *Z*?), che utilizza una rappresentazione del mondo del tipo "la stanza *A* è collegata con la *B* mediante un corridoio *C*" ed uno dedicato alle operazioni elementari che si occupi di raggiungere gli obiettivi definiti dai task generati dal primo livello (ad esempio, per andare nella stanza *X* devo girarmi a destra di 30° e percorrere 10 metri).

PERCEZIONE DEL MONDO

Fin qui la differenza con la risoluzione automatica di problemi è molto sottile e riguarda semplicemente il fatto di dover ragionare su azioni e stati più astratti. La vera differenza tra i due approcci si manifesta quando si viola la condizione di agire in un "mondo chiuso", ossia un ambiente complesso in cui ogni evento è noto e predicibile.

Il caso più semplice è quello dell'incompletezza dell'informazione, dovuta per esempio all'impossibilità di determinare all'inizio lo stato di determinati oggetti o la misura di determinate variabili. A questo si può aggiungere l'indeterminatezza degli effetti (ad esempio, l'azione "avanza di 10 metri" non è detto che faccia avanzare il robot realmente di 10 metri a causa di attriti o macchie d'olio non previste), la possibilità di azioni esogene (ad esempio, qualcuno sposta i cubi mentre il robot li sta

Figura 1 Rhino, il robot realizzato dall'università di Bonn. La sua intelligenza è basata sul GOLOG per la pianificazione di alto livello e su alcuni moduli dedicati al collision-avoidance e alla realizzazione di una mappatura dell'ambiente circostante per quanto riguarda i movimenti di basso livello

impilando) o addirittura l'eventualità di sensori non affidabili (ad esempio, una telecamera che definisca un oggetto blu quando invece è verde).

In questo caso la risoluzione automatica mostra tutte le proprie lacune poiché non gestisce né la probabilità, né rappresentazioni di stati incompleti. Al contrario l'intento della robotica cognitiva è quello di dotare l'agente di una "percezione" del mondo, ossia una specie di "base di conoscenza virtuale" che includa tutto ciò che l'agente può supporre in relazione all'esperienza maturata.

Queste informazioni sono effetto dell'incertezza che può caratterizzare una realtà complessa e sono ben distinte da quella che è la rappresentazione "reale" del mondo esterno. Per esempio, se il robot prende l'oggetto *X* e lo poggia sull'oggetto *Y*, è certa un'affermazione che dica "*X è sopra Y*"; al contrario non è possibile affermare che *Y* è sopra *Z* a meno che non esista un'azione precedente che giustifichi tale sentenza. In quest'ultimo caso l'agente può "supporre" che *Y* sia sopra *Z* poiché è l'evento più probabile in relazione alle informazioni in suo possesso.

Questa dualità percezione-realtà può portare anche a contraddizioni forti che possono essere risolte modificando l'esperienza dell'agente al fine di portarlo a supporre in futuro delle conclusioni più realistiche. Ovviamente nel caso di un mondo chiuso i due livelli tendono a coincidere e non è più necessario differenziare ciò che è deducibile da ciò che è percepito.

In alcuni casi l'agente può anche decidere di introdurre delle azioni di sensing (calcola la distanza dell'oggetto *X*, dammi la forma dell'oggetto *W*, ecc.) che solo all'atto dell'esecuzione finale del piano potranno provare la veridicità di una supposizione. Se si verificano delle incongruenze il sistema può ritrattare il piano e aggiornare la propria esperienza in modo da migliorare le proprie conclusioni.

Per impedire questa ripianificazione dei task è anche possibile creare a valle di ciascuna azione di sensing più rami distinti contenenti diverse sequenze di azioni (piano condizionale) che possono essere legati ad uno o più valori restituiti dall'azione. Nel caso di possibilità infinite è anche lecito sfoltire le diramazioni sempre basandosi sull'esperienza ed il buon senso maturato.

In definitiva si può dire che la robotica cognitiva si basa su una rappresentazione astratta del mondo che viene interpretata e modificata in base agli effetti delle azioni pianificate e a ciò che l'agente può supporre in relazione alla propria percezione del mondo. A questo si aggiunge un sistema basato sull'esperienza che guida l'agente nelle proprie supposizioni anche in relazione agli effetti positivi o negativi che queste hanno prodotto.

Un agente può avere più sensori o attuatori secondo il grado di affidabilità voluto o la complessità delle attività da compiere

BASE DI CONOSCENZA E SITUATION CALCULUS

Come già accennato, un agente ha bisogno di una rappresentazione del mondo legata al livello di astrazione voluto. Uno dei formalismi più noti che possono aiutare a descrivere un dominio dinamico è sicuramente il calcolo delle situazioni.

In breve, il calcolo delle situazioni si basa sull'idea che tutti i possibili cambiamenti nel mondo sono dovuti esclusivamente ad azioni elementari, quindi lo stato corrente è l'effetto della sequenza di azioni eseguite su di esso a partire da uno stato iniziale.

Formalmente il calcolo delle situazioni ha bisogno delle seguenti definizioni:

- Una situazione iniziale identificata con il simbolo *s0*.
- Una o più azioni rappresentate da simboli di funzione come, ad esempio, *go(P)*.
- Un simbolo di funzione logico *do(A,S)=S1*, dove *A* denota sempre un'azione elementare e *S* e *S1* le situazioni di partenza e d'arrivo.
- Uno o più fluenti relazionali o funzionali che rappresentano le proprietà che possono variare in relazione alle azioni svolte, ad esempio *Posizione(A,P,S)*.
- Per ciascun fluente un assioma stato-successore espresso nella seguente forma:

```
F(X1,... Xn,do(A,S)) :-
A=action1,X1=... Xn=... ;
A=action2,X1=...  Xn=... ;
...
...not(A=action1),not(A=action2),...
                              F(X1,... Xn,S).
```

dove *X1…Xn* denotano le variabili che caratterizzano il fluente, *A* rappresenta l'azione corrente e la regola può essere letta come "se *A* è un'azione che modifica lo stato del fluente, dai i nuovi valori, altrimenti dai al fluente il valore della situazione precedente".

- Una serie di condizioni iniziali per ciascun fluente del tipo *F(X1,…,Xn,s0)*.
- Un insieme di assiomi di precondizione per specificare le condizioni preliminari che devono essere verificate affinché un'azione possa essere applicata, espressi mediante il predicato *poss(A,S)* che indica la possibilità o meno di compiere l'azione *A* nella situazione *S*.

L'astrazione semplifica notevolmente la descrizione della realtà e consente all'agente di concentrarsi su obiettivi di alto livello

Una volta definiti gli elementi elencati si ottiene una assiomatizzazione che consente di descrivere lo stato corrente del mondo e le sue eventuali evoluzioni.

Una nota particolare meritano le azioni di sensing, ossia quelle operazioni che riguardano l'attivazione di uno o più sensori. Queste azioni non apportano alcuna variazione alla base di conoscenza poiché non modificano lo stato dei fluenti, però possono confermare o contraddire determinate affermazioni oppure dipanare determinate situazioni di incertezza dovute ad una non completa descrizione dello stato iniziale o degli effetti a fronte di un'azione. Il **Listato 1** presenta un dominio di esempio in Prolog per il mondo dei blocchi. Il mondo consiste in un tavolo ed una serie di blocchi colorati che possono essere messi uno sopra l'altro o sul tavolo. Le uniche azioni eseguibili sono quella di mettere un blocco sopra l'altro o di posare un blocco sul tavolo. Ovviamente non è possibile muovere un blocco se è all'interno di una torre e non può essere messo sul tavolo o su un altro blocco se già è sistemato in quel punto. Questo dominio servirà da esempio per tutte le successive implementazioni.

RAPPRESENTAZIONE DI AZIONI COMPLESSE

La limitazione principale del calcolo delle situazioni è di non poter definire azioni complesse o procedure al fine di definire un piano d'azione logico da seguire. Levesque e Reiter dell'università di Toronto hanno fornito un approccio per raggiungere tale obiettivo, introducendo un nuovo linguaggio di programmazione logica chiamato *GOLOG* (alGOl in LOGic).

Il GOLOG è un linguaggio di programmazione logica molto utile per implementare domini dinamici in applicazioni quali: robotica, controllo dei processi, agenti software intelligenti, ecc.

Esso si basa su una teoria formale per le azioni specificata in una versione estesa del calcolo delle situazioni. In generale un programma GOLOG è una macro che viene espansa, durante la valutazione, in formule in cui vengono menzionate solamente le azioni primitive ed i fluenti definiti dall'utente.

La valutazione viene realizzata attraverso un *theorem prover* il cui compito è quello di verificare tali formule relativamente ad una assiomatizzazione di base, comprendente gli assiomi fondamentali del calcolo delle situazioni, gli assiomi di precondizione per le azioni primitive, gli assiomi di stato successore per i fluenti e gli assiomi che descrivono la situazione iniziale.

L'esecuzione di un programma GOLOG consiste nell'individuare un termine *s* per cui è possibile rendere vera la funzione *Do(programma, s0, s)* partendo da una base di conoscenza già formalizzata.

La valutazione di questa funzione fornisce come side effect sulla variabile *s* un termine del tipo $do(a_n, \ldots, do(a_2, do(a_1, s_0)) \ldots)$.

La funzione *Do(…)* utilizza un theorem prover (un motore

Tabella 1 Definizione delle principali azioni-macro all'interno del GOLOG. Ogni azione complessa viene tradotta ricorsivamente in una espressione da valutare in base alle regole di questa tabella

Azione Primitiva	`Do(a, s, s') → Poss(a,s) ∧ s' = do(a[s],s)` Definisce l'esecuzione di un'azione elementare. Con *a[s]* si intende il termine che si ottiene ripristinando l'argomento *S* a tutti i fluenti citati nel termine *a*. Ad esempio se *a* è *drop(blue(X))*, *a[s]* è *drop(blue(X,S))*.
Test su un'espressione	`Do(?(P), s, s') → P[s] ∧ s = s'` Definisce un test sull'espressione specificata nell'argomento. L'esecuzione prosegue se l'esito è positivo, altrimenti bisogna regredire su *s'*. *P* è un'espressione che individua una formula del calcolo delle situazioni in cui tutti i termini in *S* sono stati soppressi. *P[s]* è invece la formula che si ottiene da *P* ripristinando il termine *S* in tutti i fluenti citati in essa. Ad esempio se *P* è *onTable(X)*, *P[S]* è *onTable(X,S)*.
Sequenza	`Do(E1 : E2, s, s') → ∃s".Do(E1,s,s") ∧ Do(E2,s",s')` Identifica l'esecuzione sequenziale di due azioni.
Scelta non deterministica tra due azioni	`Do(E1 # E2, s, s') → Do(E1,s,s') ∨ Do(E2,s,s')` Identifica la scelta non deterministica (randomica) tra due possibili azioni.
Scelta non deterministica dell'argomento di una azione	`Do((px)E, s, s') → ∃x.Do(E(x),s,s')` Consente la scelta non deterministica di un valore per l'argomento x che viene impostato all'interno di tutte le ricorrenze nell'azione *E*.
Iterazione non deterministica	`Do(E*, s, s') → s = s' # Do(E: E*, s, s')` Definisce l'esecuzione ciclica di un'azione per un numero di iterazioni non noto a priori (non determinismo nel numero di esecuzioni successive).

deduttivo che dimostra l'effettiva valutabilità della funzione in relazione agli assiomi che descrivono la base di conoscenza) che deve essere lanciato in un processo off-line. Il risultato di questa elaborazione è un piano d'azione composto da una sequenza più o meno lineare di operazioni che potrà successivamente essere passata ad un execution module che si occuperà dell'esecuzione delle azioni primitive contenute.

In pratica si definisce induttivamente un termine del tipo *Do(A, S, S')* che viene espanso dal sistema diversamente in base alla struttura del suo primo argomento. Il termine viene quindi espanso ricorsivamente e fino a raggiungere termini atomici che vengono valutati dal motore e scartati o meno se risultano veri o falsi.

Ad esempio, se il primo argomento corrisponde ad un'azione primitiva *action*, questa viene trasformata in un termine logico del tipo:

Poss(action[S], S) ∧ S' = do(action[s], s)

dove con *action[s]* si intende il termine che si ottiene ripristinando l'argomento *s* su tutti i fluenti citati nel termine action che identifica l'azione primitiva (ad esempio, *go(position(A))* che identifica un'azione di spostamento verso la posizione di un oggetto X diventa *go(position(A, S))*, mentre *drop(A)* che rappresenta l'azione di gettare un oggetto non subirà alcuna variazione).
Se l'esito dei questa formula logica è vero il termine "action" viene accettato ed inserito come side-effect in *S'*.

Nella **Tabella 1** sono riportate tutte le regole di espansione definite per questo linguaggio.

Alla stessa maniera è possibile definire le espressioni per i costrutti del tipo *while* o *if-then-else* usando i precedenti nel modo seguente:

if P then E1 else E2 endif	→ ?(P): E1 # ¬(?P): E2
while P do E endwhile	→ [?(P): E]*: ¬(?P)

Infine è possibile aggregare più istruzioni all'interno di procedure che possono essere anche invocate ricorsivamente. Il **Listato 2** mostra una implementazione in Eclipse Prolog dell'interprete GOLOG realizzato presso l'università di Toronto da Levesque e Reiter, in cui sono state fatte due importanti assunzioni:

- Non ci siano fluenti funzionali, ma solamente un numero finito di fluenti proposizionali e simboli di funzione per le azioni.
- Nel definire la base di conoscenza iniziale si ipotizza di essere in un mondo chiuso.

Listato 2 L'interprete GOLOG realizzato dall'università di Toronto (Reiter-Levesque)

```
:- set_flag(print_depth,100).
:- nodbgcomp.
:- dynamic(proc/2).            % Direttive del compilatore
:- set_flag(all_dynamic, on).

:- op(800, xfy, [&]).    % congiunzione (and)
:- op(850, xfy, [v]).    % disgiunzione (or)
:- op(870, xfy, [=>]).   % implicazione
:- op(880,xfy, [<=>]).   % equivalenza
:- op(950, xfy, [:]).    % sequenza di azioni
:- op(960, xfy, [#]).    % scelta non deterministica di un'azione

do(E1 : E2,S,S1) :- do(E1,S,S2), do(E2,S2,S1).
do(?(P),S,S) :- holds(P,S).
do(E1 # E2,S,S1) :- do(E1,S,S1) ; do(E2,S,S1).
do(if(P,E1,E2),S,S1) :- do((?(P) : E1) # (?(-P) : E2),S,S1).
do(star(E),S,S1) :- S1 = S ; do(E : star(E),S,S1).
do(while(P,E),S,S1):- do(star(?(P) : E) : ?(-P),S,S1).
do(pi(V,E),S,S1) :- sub(V,_,E,E1), do(E1,S,S1).
do(E,S,S1) :- proc(E,E1), do(E1,S,S1).
do(E,S,do(E,S)) :- primitive_action(E), poss(E,S).

% sub(Name,New,Term1,Term2): Term2 Ë Term1 con ciascun termine Name
                                                    sostituito da New.

sub(X1,X2,T1,T2) :- var(T1), T2 = T1.
sub(X1,X2,T1,T2) :- not var(T1), T1 = X1, T2 = X2.
sub(X1,X2,T1,T2) :- not T1 = X1, T1 =..[F|L1], sub_list(X1,X2,L1,L2),
                    T2 =..[F|L2].
sub_list(X1,X2,[],[]).
sub_list(X1,X2,[T1|L1],[T2|L2]) :- sub(X1,X2,T1,T2), sub_list(X1,X2,L1,L2).

/* Il predicato holds implementa le trasformazioni di Lloyd-Topor
   per le condizioni di test.  */

holds(P & Q,S) :- holds(P,S), holds(Q,S).
holds(P v Q,S) :- holds(P,S); holds(Q,S).
holds(P => Q,S) :- holds(-P v Q,S).
holds(P <=> Q,S) :- holds((P => Q) & (Q => P),S).
holds(-(-P),S) :- holds(P,S).
holds(-(P & Q),S) :- holds(-P v -Q,S).
holds(-(P v Q),S) :- holds(-P & -Q,S).
holds(-(P => Q),S) :- holds(-(-P v Q),S).
holds(-(P <=> Q),S) :- holds(-((P => Q) & (Q => P)),S).
holds(-all(V,P),S) :- holds(some(V,-P),S).
holds(-some(V,P),S) :- not holds(some(V,P),S).  % Negazione
holds(-P,S) :- isAtom(P), not holds(P,S).       % per fallimento
holds(all(V,P),S) :- holds(-some(V,-P),S).
holds(some(V,P),S) :- sub(V,_,P,P1), holds(P1,S).

/* La clausola seguente riguarda il predicato holds per gli elementi non
                                                                  fluenti,
   inclusi i predicati di sistema del Prolog. Per funzionare correttamente,
   il programmatore GOLOG deve includere per ciascun fluente una clausola
   restoreSitArg che restituisca l'argomento situazione per tutti i termini
   che non la hanno, per esempio:
         restoreSitArg(ontable(X),S,ontable(X,S)).              */

holds(A,S) :- restoreSitArg(A,S,F), F ;
              not restoreSitArg(A,S,F), isAtom(A), A.

isAtom(A) :- not (A = -W ; A = (W1 & W2) ; A = (W1 => W2) ;
             A = (W1 <=> W2) ; A = (W1 v W2) ; A = some(X,W) ; A =
                                                          all(X,W)).

restoreSitArg(poss(A),S,poss(A,S)).
```

La robotica cognitiva tende a formalizzare un processo deduttivo efficace in caso di indeterminatezza o incompletezza dell'informazione

Il GOLOG è un linguaggio di programmazione logica utile per implementare domini dinamici in applicazioni quali: robotica, controllo dei processi, agenti software intelligenti, ecc.

Listato 3 Realizzazione di una procedura GOLOG il cui scopo è posare un cubo blu sul tavolo a costo di disfare tutte le torri presenti

```
:- include(golog).
:- include(dominio).

/* Lancia la valutazione dell'interprete e produce
   come side-effect una situazione S contenente le
   azioni elementari che soddisfano il task descritto
   dalla procedura putBlueBlockOnTable */

demo(S) :- do(putBlueBlockOnTable,s0,S).


/* ------------------------- */
/* PROCEDURA putBlueBlockOnTable */
/* ------------------------- */
/* sposta un blocco blu sul tavolo.
   Nel caso non esista un blocco blu libero
   disfa le torri finchÈ non ne trova uno (chiamata
   ricorsiva della stessa funzione).
*/

proc(putBlueBlockOnTable,
     pi(X, ?(blue(X) & -onTable(X)) : putOnTable(X)) #
     pi(X, putOnTable(X) : putBlueBlockOnTable)).


/* ------------------------- */
/* Definizioni base per il GOLOG */
/* ------------------------- */

primitive_action(put(X,Y)).
primitive_action(putOnTable(X)).

restoreSitArg(on(X,Y),S,on(X,Y,S)).
restoreSitArg(onTable(X),S,onTable(X,S)).
```

In questa implementazione le condizioni di test P possono essere o formula atomica o una espressione nella forma: P_1 & P_2, P_1 v P_2, -P, P_1 => P_2, P_1 <=> P_2, *all(V, P)*, *some(V, P)*, dove V è un atomo e P è una condizione che usa V.

Nel valutare tali condizioni l'interprete utilizza la negazione per fallimento per –P (not P) ed il predicato *holds* per determinare quali fluenti sono veri.

I termini *some* ed *all* corrispondono ai quantificatori ∃ e ∀. Alcune azioni complesse sono state formalizzate nel seguente modo:

- Scelta non deterministica di un argomento: *pi(X,E)*
- Iterazione non deterministica: *star(E)*
- if-then-else: *if(P,E1,E2)*
- ciclo while: *while(P,E)*

Infine, va notato che la versione dell'interprete mostrata presuppone l'esistenza di una teoria di base per le azioni opportunamente definita attraverso clausole dedicate.

In particolare una applicazione in GOLOG dovrà includere le seguenti parti:

- Un insieme di clausole nella forma *primitive_action (action)* per ciascuna azione eseguibile.
- Un insieme di clausole nella forma *proc(nome, istruzioni)* per dichiarare le procedure.
- Un insieme di clausole che definiscano le precondizioni per ciascuna azione.
- Una clausola nella forma *restoreSitArg(fluente, S, fluente[S])* per ciascun fluente che verrà utilizzata dal predicato *holds* per valutare espressioni condizionali del GOLOG.

Il **Listato 3** presenta un'implementazione in GOLOG di una procedura per il mondo dei blocchi precedentemente formalizzato che consente all'agente di posare sul tavolo uno dei blocchi blu.

CONCLUSIONI

Nel GOLOG le azioni complesse sono definite per mezzo di macro che vengono espanse durante la valutazione in formule del calcolo delle situazioni. Molte varianti sono state apportate a questo linguaggio introducendo il concetto di tempo (possibilità di definire la durata delle azioni e il loro tempo iniziale e finale di esecuzione) o la possibilità di avere più azioni concorrenti (la variante si chiama CONGOLOG e la sua implementazione di riferimento non è in Prolog, ma in C).

Infine è possibile introdurre azioni di sensing mirate a risolvere determinate situazioni di incertezza o formalizzare la presenza di eventi esogeni, fino a definire funzioni di "affidabilità" per ciascun sensore finalizzate a dare più o meno credito alle informazioni ottenute.

In realtà il campo è ancora aperto e la maggior parte delle problematiche sono ancora in studio presso i principali atenei di tutto il mondo. Questo implica che ancora non esiste un'applicazione commerciale che risolva insieme tutte le nozioni presentate (tranne qualche raro caso come il progetto Rhino dell'università di Bonn). La speranza, quindi, è che questi studi trovino sbocco verso una soluzione interessante che possa garantire anche ottimi tempi di risposta in ambienti particolarmente complessi e che sia in grado di rispondere alla domanda iniziale posta in questo articolo: come è possibile implementare l'intelligenza?

BIBLIOGRAFIA

[1] Stuart J. Russel, Peter Norvig, "*Artificial Intelligence, a modern approach*", Prentice-Hall International Inc., 1996.

[2] R. Reiter, "*Knowledge in Action: logical foundations for describing and implementing dynamical systems*", MITPress.

[3] R. Reiter, "*The Frame Problem in the Situation Calculus: a simple solution*", Academic Press, San Diego (CA), 1991.

[4] F. Pirri, R. Reiter, "*Some contributions to the Metatheory of the Situation Calculus*", Journal of ACM.

[5] A. Pompili, "*Risoluzione automatica dei problemi*", DEV nr. 94, Gruppo Editoriale Infomedia