Homework 3 Intelligenza artificiale

In questo Homework è stato implementato un sistema in grado di riconoscere comandi in linguaggio naturale tramite un interprete, successivamente i comandi interpretati verranno eseguiti. L'implementazione è stata realizzata totalmente con il linguaggio Prolog, l'operato è stato testato e fatto girare tramite l'interprete SWI-Prolog. Di seguito si riporta la struttura dei comandi eseguibili sul sistema.

Tipo	Frase	Descrizione
Asserzione	Is b on the table?	Restituisce Sì o No, a seconda che b sia o meno sul tavolo (cioè Sì, se Knowledge contiene *on (b, tabella)*).
	Is b on c?	Come prima, ma con un blocco (ad esempio, Sì se Knowledge contiene *on (b, c)*).
Comando	Grasp block b	Prende il blocco b con il braccio. È possibile solo se il braccio non sta afferrando nulla.
	Put it onto the table	È creato per essere usato in combinazione con la presa (Grasp block b). Permette di mettere il blocco afferrato sul tavolo.
	Put it onto c	È creato per essere usato in combinazione con la presa (Grasp block b). Permette di mettere il blocco afferrato su un altro (nel nostro caso sul blocco c).
	Print state	Stampa uno stato (ossia una descrizione) del mondo a blocchi.
	Put a onto b	Questo è il comando che richiede Planner.pl. Restituisce il piano per modificare lo stato del mondo a blocchi (in questo caso, l'elenco delle azioni necessarie per mettere il blocco a sul blocco b).
Query	Which blocks are on the table?	Stampa a schermo la lista di blocchi che si trovano sul tavolo in quel momento.
	What color is b?	Stampa a schermo il colore del blocco b.

I File del progetto

Questo homework realizza un Sistema di elaborazione del linguaggio naturale per il mondo a blocchi che accetta asserzioni, query e comandi. Il progetto contiene i seguenti file:

1. **Main.pl** – Questo file importa tutti i file di lavoro del progetto, e istanzia le due funzioni che il sistema permette di utilizzare tramite terminale. Per iniziare a dare istruzioni al sistema è necessario invocare la funzione (start.) in questo modo verrà richiesto all'utente di inserire un comando in linguaggio naturale, di seguito vediamo l'esempio:

```
?- start.

Dammi un istruzione :
```

   Il sistema come si è visto all'inizio di questa relazione permette l'inserimento in linguaggio naturale solo di determinate istruzioni, per ottenere l'elenco dei comandi disponibili basta digitare il comando (help.). Di seguito vediamo l'output prodotto da questo comando:

```
?- help.
Assertion:
 1) 'Is b on the table?'
 2) 'Is b on c?'

Command:
 1) 'Grasp block b'
 2) 'Put it onto the table'
 3) 'Put it onto c'
 4) 'Print state'
 5) 'Put a onto b'

Query:
 1) 'Which blocks are on the table?'
 2) 'What color is b?'
true.
```

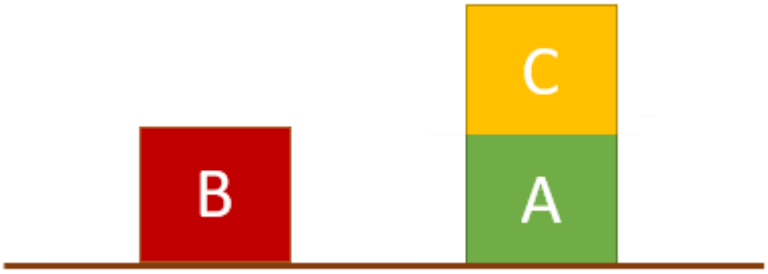
2. **Planner.pl** – Contiene tutte le informazioni di partenza, ad esempio dove sono I blocchi (ossia se sono l'uno sull'altro o sopra il tavolo, ecc.). Inoltre contiene tutte quelle istruzioni necessarie per pianificare lo spostamento automatico dei blocchi in modo intelligente.

3. **Parser.pl** – Implementa un gestore che trasforma la frase data in input in una lista di token, successivamente questa lista viene passata ad un parser che restituisce l'albero di analisi della frase, dopo di che viene passato tutto ad un interprete che elabora le asserzioni per provvedere a rispondere all'utente

Come funziona il programma

Si noti che, per rendere questo report più semplice e comprensibile, si è creato questo paragrafo concentrandoci sulle asserzioni *"b on c?"*, ma il comportamento del programma è molto simile per qualsiasi frase supportata e ammissibile. L'elenco completo delle query supportate, delle asserzioni e dei comandi è mostrato sopra (nell' Introduzione).

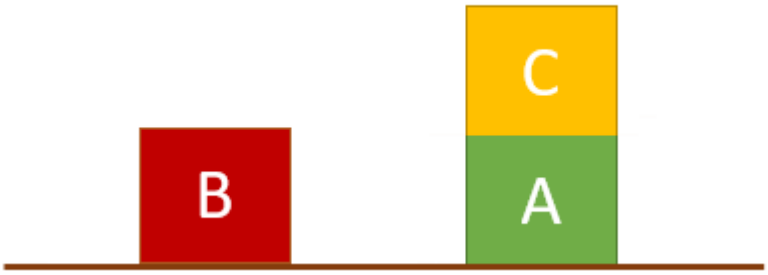
Lo stato iniziale del mondo a blocchi è il seguente:



Il programma funziona in questo modo, digitando il comando **help.** è possibile scoprire tutte le possibili istruzioni eseguibili su questa implementazione, fatta assunzione di tutte le richieste che è possibile fare digitando ***start***. ti verrà chiesto di inserire qualcosa messo nella forma *'is b on c?'*. Questa domanda deve essere trasformata in una lista di termini di forma [is, b, on, c,?]. Una volta che la domanda viene espressa sotto forma di token come appena illustrato, viene passata all'analisi dei predicati (Tokens, Tree). Viene restituito il suo albero di analisi. In realtà, questo albero di analisi è una query Prolog. Questo vuol dire che una parte dell'interpretazione è già presente nel parser. L'albero di analisi è formato da comandi come: *on (b, c)*. Per eseguire il predicato, *on (b, c).* viene passato al modulo interpreter, qui, gli argomenti (b e c) vengono estratti e attivati (Object, RelativeObject). Il comando poi può essere eseguito, restituendo Sì o No in quanto la risposta alla domanda iniziale "*è b on c?*". Questo compito è realizzato da **Interpreter.pl**.

Nel caso in cui un comando di forma *Put b onto c* o *Put b onto table* sia dato come input, il componente **Intepreter** richiederà un servizio al componente **Planner** (vale a dire, risolvere (...)). Ciò restituisce l'insieme di azioni da eseguire per eseguire quell'azione.

Esempio di OutPut



Tenendo in considerazione la situazione di partenza mostrato qui sopra vediamo come il sistema risponde ad alcuni comandi:

```
?- start.

Dammi un istruzione : 'is c on the table?'.
No.
false.
```

Il sistema risponde No alla domanda c'è sul tavolo!! Questa affermazione è giusta in quanto il blocco C è adagiato sul blocco A

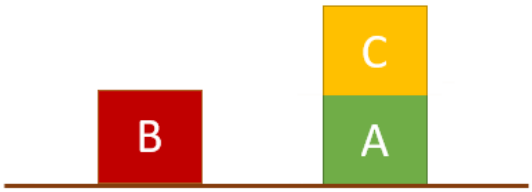
```
?- start.

Dammi un istruzione : 'What color is c?'.
The color is: yellow
true .
```

Vediamo come il sistema alla domanda di che colore è il blocco C risponde Yellow !!

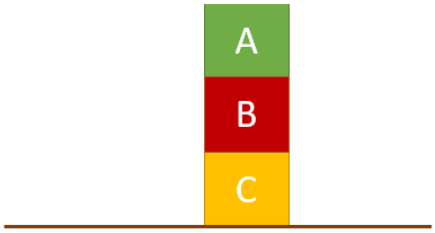
```
?- start.

Dammi un istruzione : 'Print state'.
blocks: [a,b,c]
colors: [b,red], [a,green], [c,yellow],
clear: table, c, b,
on relations: [c,a], [a,table], [b,table],
holding: nothing
true .
```

Per concludere vediamo come l'istruzione print state ci restituisce i blocchi esistenti, il loro colore e la relazione esistente tra i blocchi

Stato di Partenza



Stato di Arrivo



In questo esempio vogliamo arrivare allo stato finale che vede C adagiato sul tavolo, B posto sul blocco C e per concludere il blocco A posto sul blocco B. Di seguito si riportano le azioni necessaria per realizzare quanto detto:

```
?- start.

Dammi un istruzione : 'grasp block c'.
0K, holding c
true .
```

Per prima cosa afferriamo il blocco C per rimuoverlo dalla cima del blocco A.

```
?- start.

Dammi un istruzione : 'put it onto the table'.
0K, put c on table
true .
```

Dopo aver afferrato il blocco possiamo depositarlo sul tavolo.

```
?- start.

Dammi un istruzione : 'grasp b'.
0K, holding b
true .
```

Afferriamo il blocco B cosi che possiamo iniziare a costruire la nostra pila.

```
?- start.

Dammi un istruzione : 'put it onto c'.
0K, put b on c
true .
```

Depositiamo il blocco B sulla cima del blocco C.

```
?- start.

Dammi un istruzione : 'grasp a'.
0K, holding a
true .
```

Afferriamo il blocco A.

```
?- start.

Dammi un istruzione : 'put it onto b'.
0K, put a on b
true .
```

Depositiamo il blocco A sulla cima del blocco B

```
?- start.

Dammi un istruzione : 'print state'.
blocks: [a,b,c]
colors: [b,red], [a,green], [c,yellow],
clear: table, table, table, table, table, a,
on relations: [c,table], [b,c], [a,b],
holding: nothing
true .
```

Per concludere stampiamo lo stato, e vediamo subito nella relazione come il blocco C sia depositato sul tavolo, il blocco B sia depositato sul blocco C e per finire il blocco A è al top della pila.

Riferimenti

Per comprendere meglio l'algoritmo e per scrivere una corretta implementazione del sistema realizzato, si è fatto uso della seguente documentazione

1) URL - http://lia.disi.unibo.it/Courses/AI/fundamentalsAI2004-05/lucidi/mondoblocchi.pdf
2) Artificial Intelligence: A Modern Approach, 3rd Edition

L'elaborato è stato sviluppato in collaborazione con Ciavarro Cristina