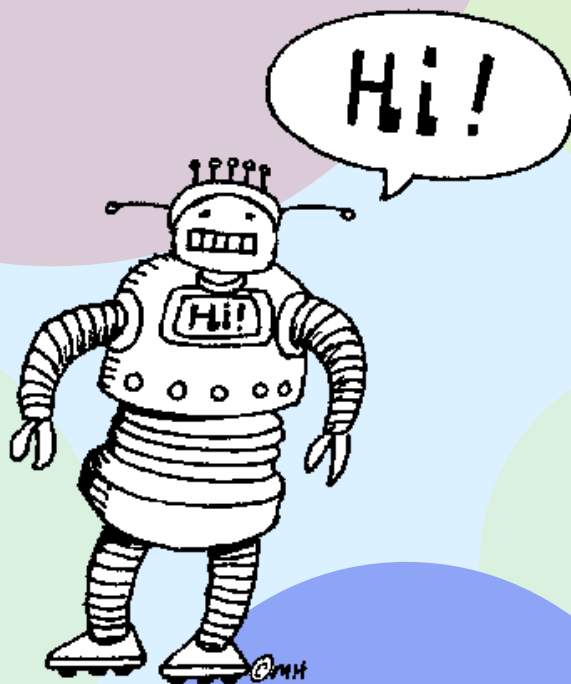


Shakey



di *Cavenaghi Mattia*
matricola 640926

Indice

Introduzione

- ❖ L'architettura di Shakey pag. 2
- ❖ Il linguaggio visuale pag. 2
- ❖ Il linguaggio PROLOG pag. 4
- ❖ Le decisioni progettuali pag. 4

Il mondo di Shakey

- ❖ Gli obiettivi di Shakey pag. 5

Il ragionamento visuale

- ❖ Problema a: "Il facchino" pag. 6
- ❖ Problema b: "La torre di Pisa" pag. 7
- ❖ Problema c: "L'elettricista" pag. 9

Il ragionamento con PROLOG

- ❖ La base di conoscenza pag. 12
- ❖ Problema a: "Il facchino" pag. 13
- ❖ Problema b: "La torre di Pisa" pag. 15
- ❖ Problema c: "L'elettricista" pag. 15

Introduzione

Come tema per il nostro progetto, abbiamo scelto di sviluppare il problema sulla gestione, e pianificazione delle azioni del robottino Shakey.

Come prima cosa, faremo una breve panoramica sull'architettura di Shakey e sul linguaggio utilizzato per scrivere i programmi in linguaggio formale.

Successivamente, introdurremo il mondo in cui Shakey interagisce, lo discuteremo con un linguaggio visuale, e con dei piccoli programmi in PROLOG.

L'architettura di Shakey

Shakey, fu il primo robot a "camminare" e ragionare, realizzato tra il 1966 ed il 1972, dal dipartimento di Intelligenza Artificiale, dello Stanford Research Center (SRI), ed influenzò notevolmente, il mondo della robotica, fino ai nostri giorni.

Equipaggiato con limitate abilità di percezione, modellazione e pianificazione, era però in grado di pianificare dei tragitti ed interagire con piccoli oggetti.

Shakey, programmato in Fortran a LISP, era equipaggiato con:

- Una telecamera;
- Un rilevatore di distanza;
- Un sensore sensibile agli urti;
- Un computer DEC PDP-10 e PDP-15
- Un carrello motorizzato.

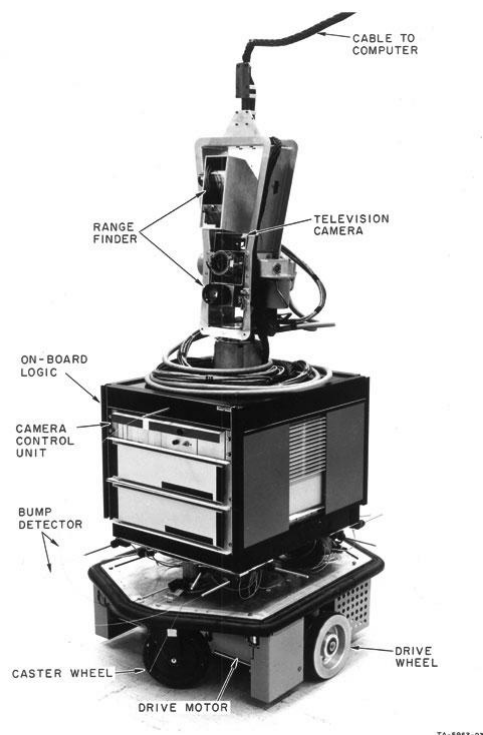


Figura 1 - Il robot Shakey

Oggi, Shakey, è custodito nel Museo sulla storia dei computer a Mountain View in California.

Il linguaggio visuale

In questa sezione, introduciamo il linguaggio che abbiamo ideato, il quale ci permetterà di comprendere meglio il comportamento di Shakey, senza l'ausilio di termini o concetti troppo tecnici. Tale linguaggio, infatti, si limita a descrivere le azioni che Shakey compie nel suo mondo, e di come interagisce con gli oggetti al suo interno.

Di seguito, abbiamo riassunto le 6 azioni che Shakey può compiere: *vai avanti*, *prendi/lascia scatola*, *spingi*, *accendi/spegni luce*, *sali/scendi scatola* e *gira a destra/sinistra di 90°*. I cui simboli e colori sono riportati nella figura 2.

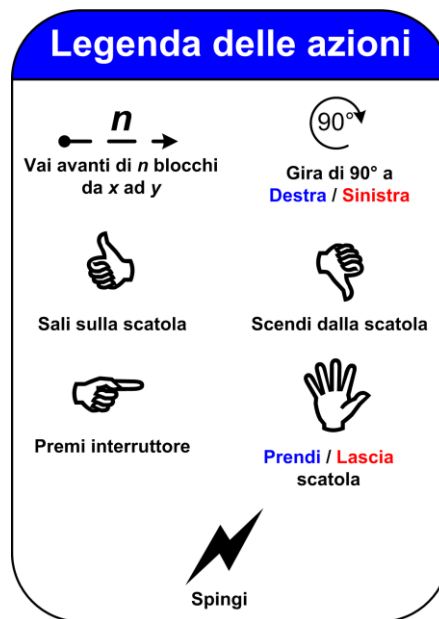


Figura 2 - Legenda delle azioni con cui Shakey interagisce col mondo esterno

Inoltre, come mondo esterno, intendiamo le stanze dell'SRI, e gli oggetti in esso contenuti, riportati in figura 3.

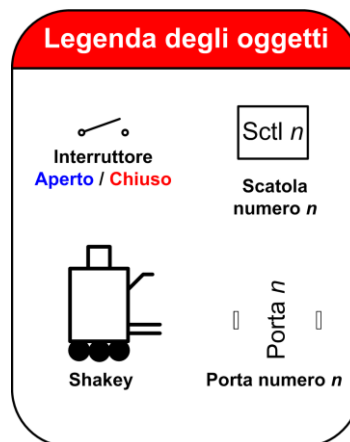








Figura 3 - Legenda degli oggetti con cui Shakey interagisce

Le precedenti due legende, si riferiscono alla parte grafica del linguaggio, ossia quella che viene utilizzata nelle piantine; per la parte logica, vengono utilizzati i simboli riportati nella seguente tabella, i quali vanno specificati di volta in volta, con il numero o nome della stanza in cui vengono compiute le azioni:

Legenda delle azioni (parte logica)	
 Vai avanti di n blocchi	 Gira di 90° a Destra / Sinistra
 Sali sulla scatola	 Scendi dalla scatola

 Accendi/Spigni Luce	 Prendi/Lascia scatola
 Spingi	^ Connettivo logico <i>and</i>

Il linguaggio PROLOG

Per simulare il comportamento del calcolatore, inserita all'interno di Shakey, abbiamo deciso di utilizzare il linguaggio logico PROLOG.

Il PROLOG (PROgramming in LOGic) è un linguaggio di programmazione logica basato sulle clausole di Horn.

Un programma logico consiste in un insieme di procedure espresse in clausole di Horn ed attivate da una asserzione iniziale d'obiettivo.

Considerando le clausole espresse, secondo le due categorie di fatti e regole, affermiamo che un programma PROLOG è costituito da:

- Un insieme di fatti che dichiarano un certo stato di cose;
- Un insieme di regole che definiscono relazioni fra stati di cose;
- Obiettivi (o domande) a cui rispondere.

La procedura utilizzata dal PROLOG è il "Principio di Risoluzione", cioè si tenta di dimostrare un teorema, o meglio, si tenta di derivare la clausola vuota a partire dalle ipotesi (i fatti e le regole).

Il PROLOG, fornisce una risposta se c'è reputazione, oppure, risponde "no" se non c'è.

Essendo questo, un progetto che usa il PROLOG, e non una relazione su PROLOG, non ci addentreremo ulteriormente nell'esposizione delle regole di semantica e sintassi di tale linguaggio.

Le decisioni progettuali

Progredendo nella realizzazione del progetto, si è ritenuto necessario, apportare delle modifiche fisiche all'architettura di Shakey; in particolare abbiamo innestato un piccolo carrello elevatore, un braccio eiettabile e delle ruote cingolate tipo carro-armato.

Inoltre, abbiamo imposto delle regole comportamentali al robot:

- Shakey si muove ed occupa, un solo quadrante per volta (di 1 m²), inoltre si sposta solo in direzione orizzontale o verticale, mai in diagonale, e mai in retromarcia;
- Ogni obiettivo, si presuppone che venga implementato, partendo dalla situazione base, già rappresentata in figura 4;
- Shakey, può ruotare a sinistra o a destra di 90°, se deve compiere rotazioni più ampie, ripete l'azione "ruota di 90°", più volte;
- Il robottino, ha in memoria, la cartina dell'ambiente (figura 4), con la sua posizione iniziale, quella delle porte, delle scatole, e degli interruttori.
- La stanza ha dimensione di 11 x 12 = 132 quadrati, aventi ognuna una coordinata del tipo rappresentato in tabella:

1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10	1.11
2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10	2.11
3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.10	3.11
...

Si può notare quindi, che ogni spostamento è dato da una coordinata iniziale x , una coordinata finale y , ed uno spostamento n

Il mondo di Shakey

Gli obiettivi di Shakey

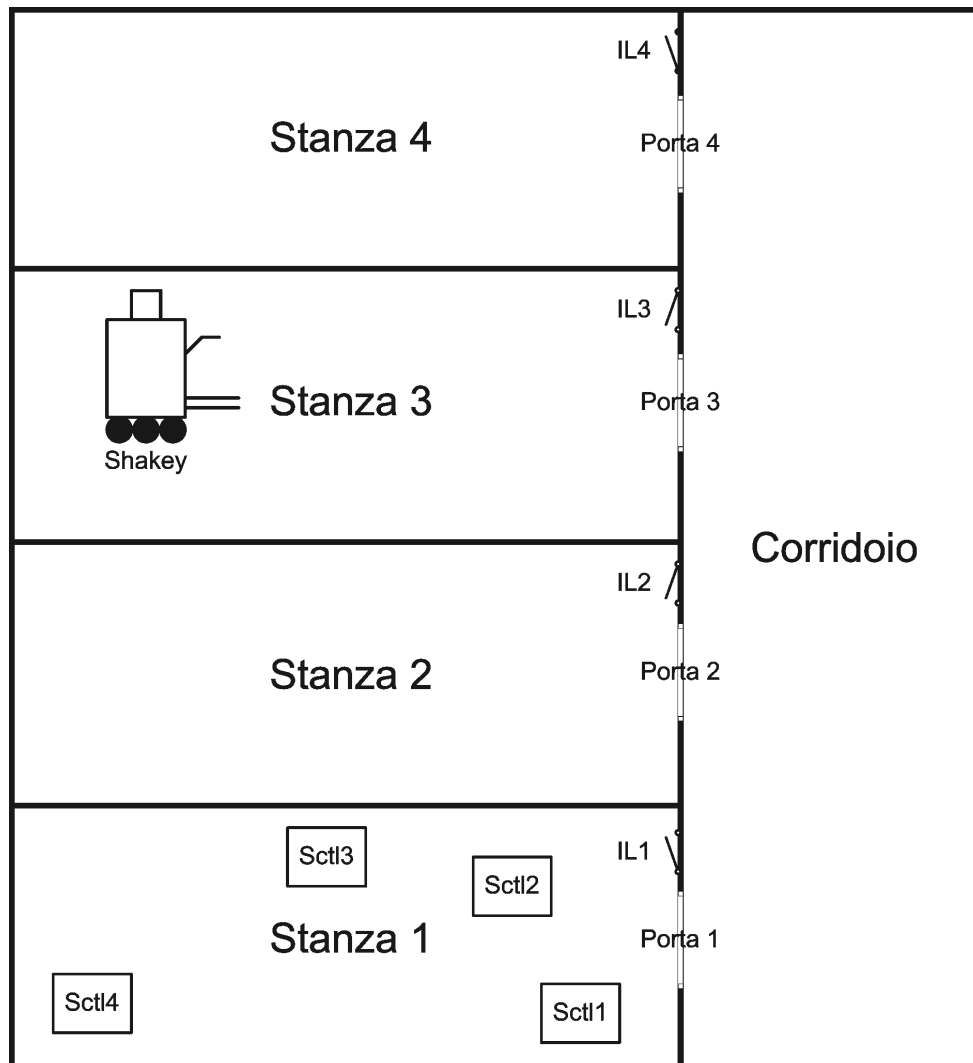


Figura 4 - Il mondo di Shakey

La figura 4, illustra le stanze dell'SRI; si può notare, che è priva di ostacoli, ad esclusione delle sole scatole, con cui il robot Shakey, interagisce.

Prima di presentare gli stati obiettivo, che devono essere implementati, dobbiamo fare alcune precisazioni, ritenute necessarie:

- ❖ Le porte sono tutte chiuse e numerate, inoltre sono "a spinta", simili a quelle presenti nelle cucine dei ristoranti;
- ❖ Tutte le scatole sono nella stanza numero 1;
- ❖ Nel corridoio, e nella stanza numero 3 c'è la luce accesa, mentre, nelle altre stanze è spenta.

Introduciamo ora, gli stati obiettivo, che tratteremo nel nostro progetto. Pur essendo all'apparenza semplici, rivelano una certa complessità d'azione, che rendono più interessante la loro progettazione, e sono:

- a) Shakey, deve spostare la scatola numero 2, nella stanza numero 2;
- b) Shakey, deve prendere tutte le scatole, impilarle e lasciarle nella stanza numero 1;
- c) Shakey, deve accendere la luce in ogni stanza, ad esclusione della stanza numero 3 e del corridoio.

Il ragionamento visuale

Problema a: "Il facchino"

Proprio come un facchino, Shakey deve andare a prendere una scatola (la numero 2), e recapitarla in una stanza (la numero 2).

Per rendere più semplice la discussione del problema, suddivideremo il compito di Shakey in due parti: l'andata ed il ritorno (figura 5 e 6)

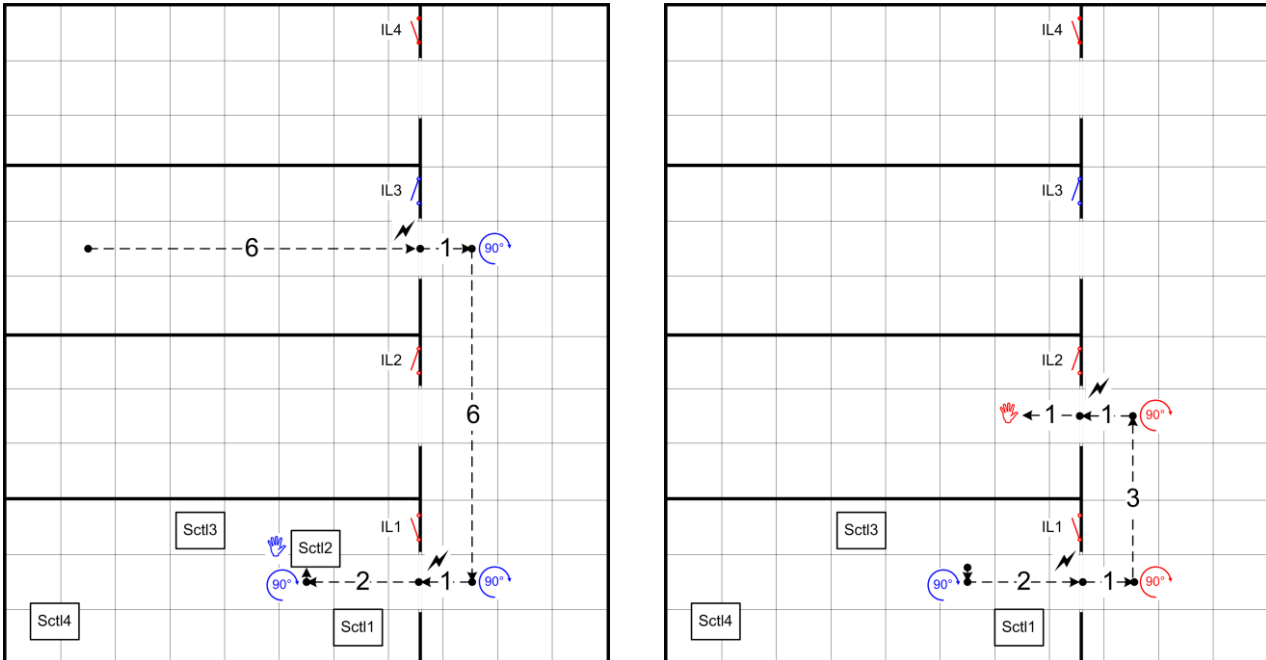


Figura 5 e 6 – Il problema del facchino

Shakey, parte dalla stanza numero 3 e percorrendo 6 quadranti sbatte contro la porta, a questo punto, i sensori ne rilevano la presenza, e il robot compie l'azione spingi ed uno spostamento di un quadrante.

Ritrovandosi nel corridoio, gira a destra e percorrere altri sei passi, gira di nuovo a destra e, camminando per un altro settore Shakey, come prima, apre la porta ed entra nella stanza numero 1.

Qui va avanti di due quadranti e girando subito di 90° a destra afferra la scatola.

Il codice per questa parte di problema è:

```

Stanza 3: →6 ^ ⚡
Corridoio: →1 ^ ↻ ^ →6 ^ ↻ ^ →1 ^ ⚡
Stanza 1: →2 ^ ↻ ^ 🖐
    
```

Passiamo ora alla seconda parte del nostro problema: afferrata la scatola, Shakey può ora portarla nella stanza numero 2.

Facendo esattamente il percorso inverso, e giunto al corridoio, Shakey deve percorrere solo tre quadranti; aprire la porta, entrare nella stanza numero 2, fare un passo e depositare la scatola per terra.

Riportiamo infine, il codice relativo a quest'ultima parte di problema:

```

Stanza 1: ↻ ^ →2 ^ ⚡
Corridoio: →1 ^ ↻ ^ →3 ^ ↻ ^ →1 ^ ⚡
Stanza 2: →1 ^ 🖐
    
```

Problema b: "La torre di Pisa"

In questo problema, Shakey deve prendere ogni scatola presente nella stanza numero 1 ed impilarle una sopra l'altra.

L'abbiamo chiamato "torre di Pisa", poiché, sulle nostre piantine, la pila di scatole, appare storta dando l'illusione di una torre pendente.

Complessivamente la risoluzione è troppo lunga, ma non complessa, l'abbiamo quindi suddivisa in quattro fasi distinte.

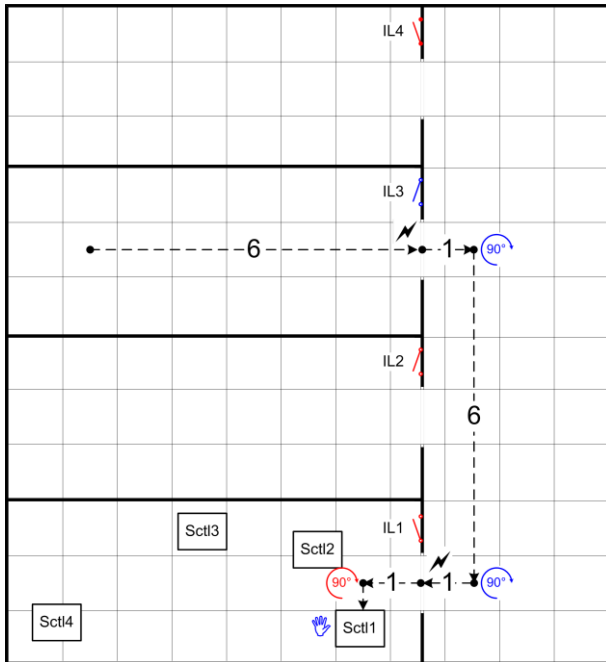


Figura 7 - La torre di Pisa

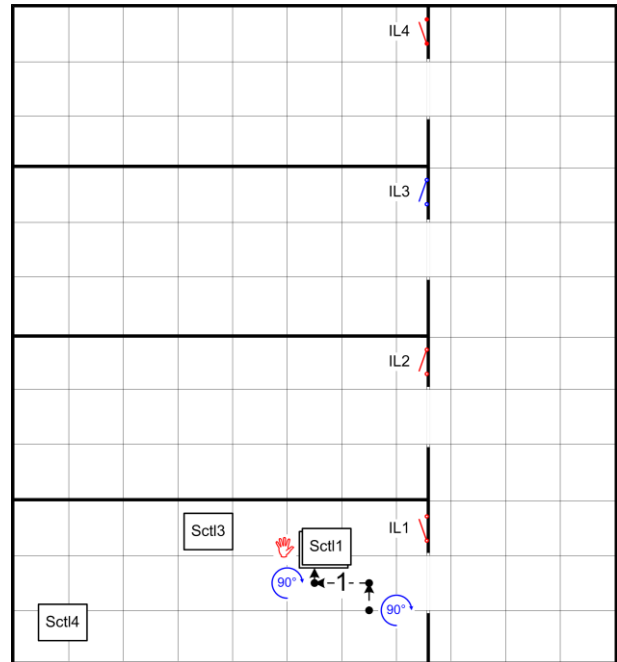


Figura 8 - La torre di Pisa

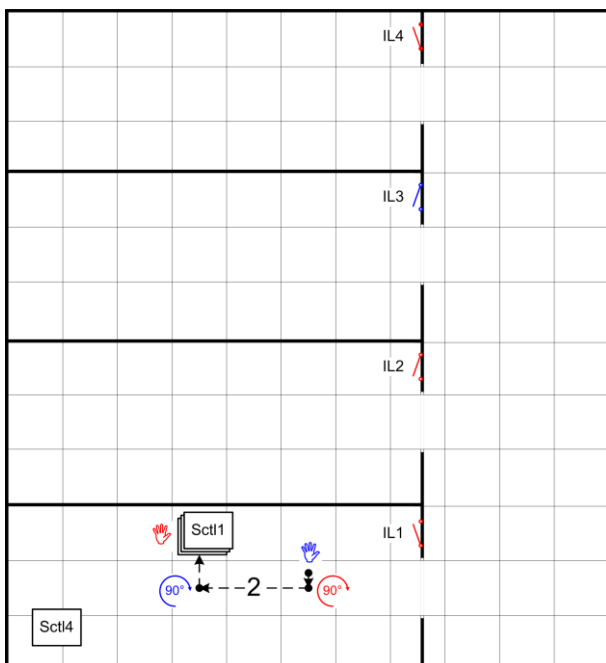


Figura 9 - La torre di Pisa

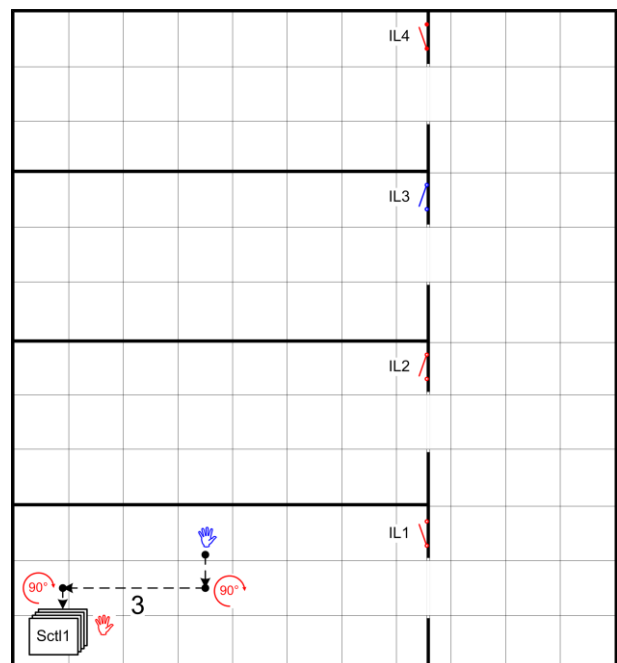


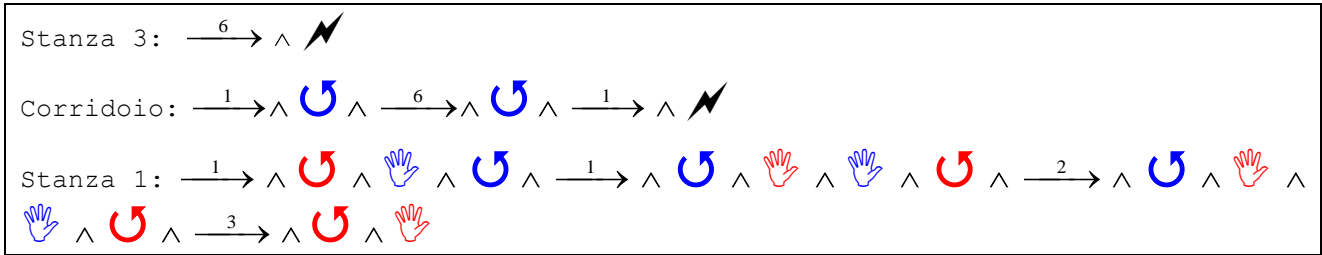
Figura 10 - La torre di Pisa

Shakey, partendo dalla stanza numero 3, apre la porta (o meglio vi sbatte contro), gira a destra, compie un tragitto di 6 quadranti, entra nella stanza numero 1 e raccoglie la scatola, tramite il carrello elevatore, di cui è dotato (figura 7).

Si può notare, che il percorso dalla stanza numero 3 alla stanza numero 1 è simile a quello già programmato nel problema del "facchino" (confronta figura 7 con figura 5).

Giunto a questo punto, il robot, non deve fare altro che raccogliere tutte le scatole ed impilarle, lasciando la scatola 4 sul fondo e la scatola 1 in testa (figura 8, 9, 10).

Passiamo ora, al codice visuale programmato:



Problema c: "L'elettricista"

Veniamo ora all'ultimo problema che Shakey deve risolvere; una volta arrivato nella stanza numero 1, il robot deve prendere una scatola e, con l'ausilio di quest'ultima accendere le luci nelle varie stanze.

Come già accennato nel capitolo "Il mondo di Shakey", la stanza 3 ed il corridoio non necessitano dell'intervento di Shakey, poiché la luce è già accesa.

Passiamo ora alla rappresentazione grafica del nostro problema:

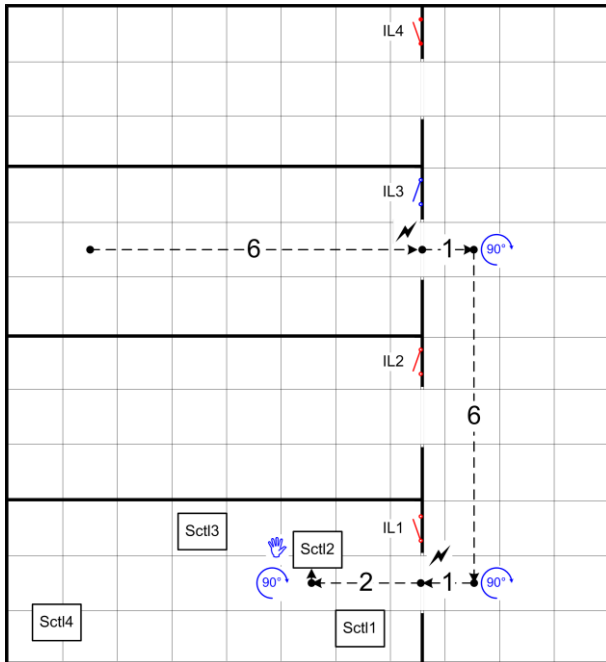


Figura 11 - L'elettricista

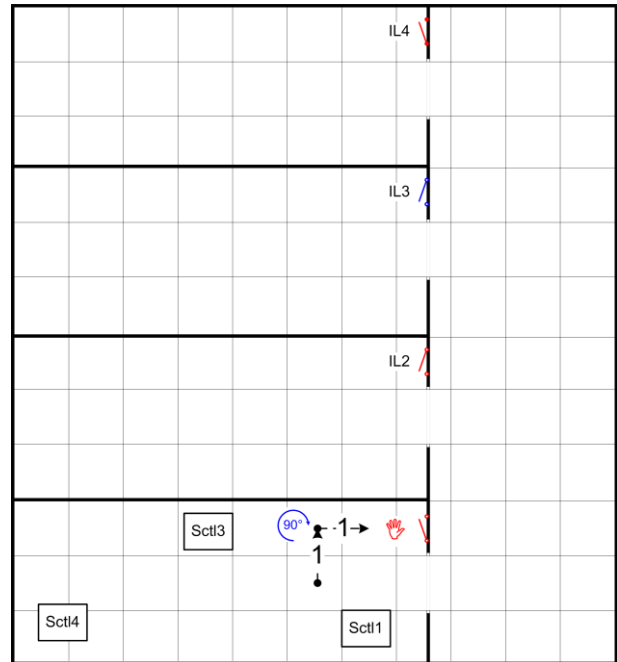


Figura 12 - L'elettricista

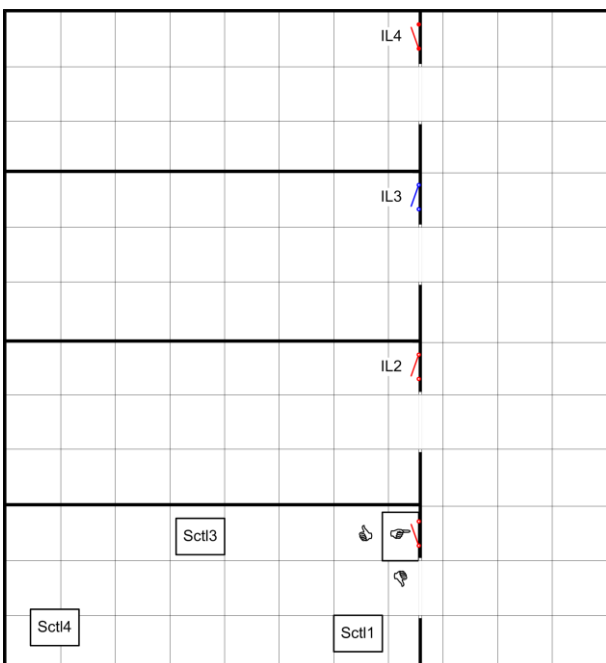


Figura 13 - L'elettricista

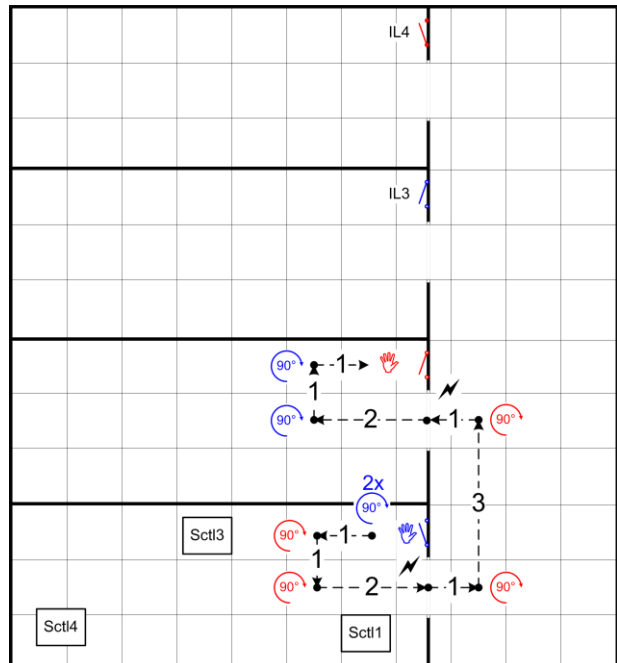


Figura 14 - L'elettricista

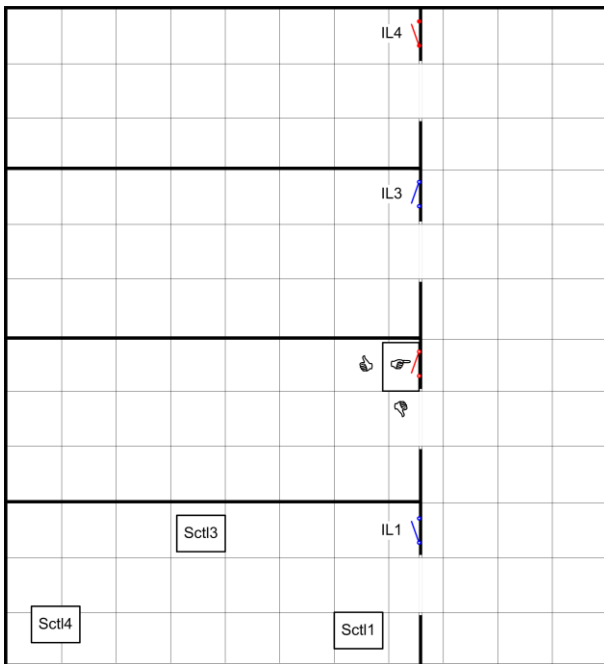


Figura 15 - L'elettricista

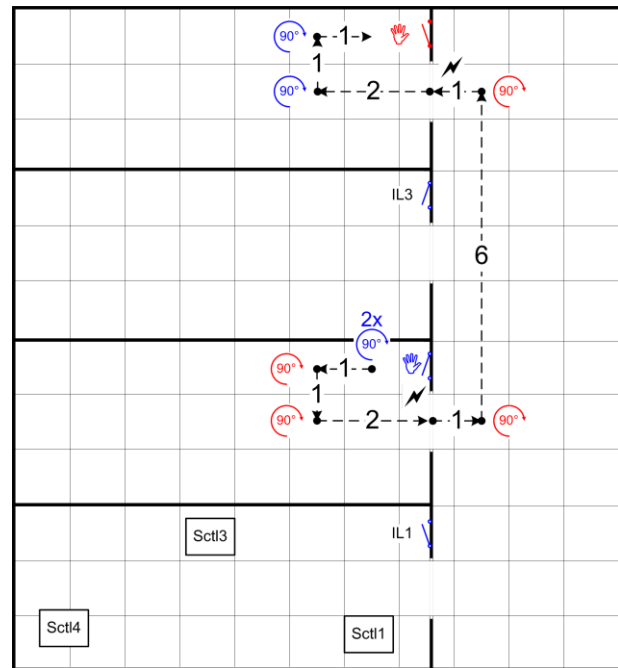


Figura 16 - L'elettricista

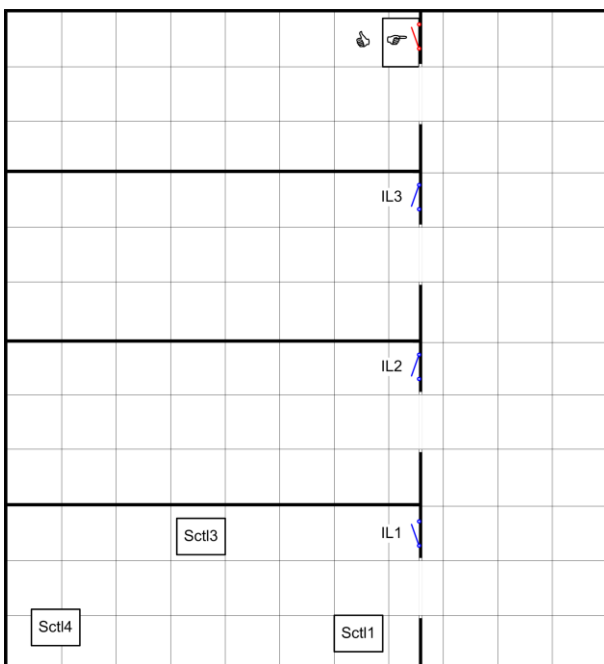


Figura 17 - L'elettricista

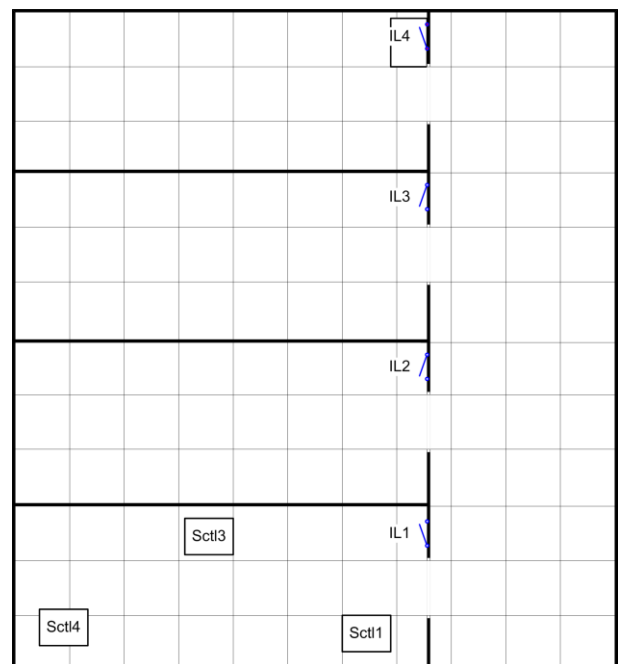


Figura 18 - L'elettricista

La figura 11 mostra come Shakey arriva alla stanza numero 1; tale procedimento è già stato discusso nei precedenti problemi.

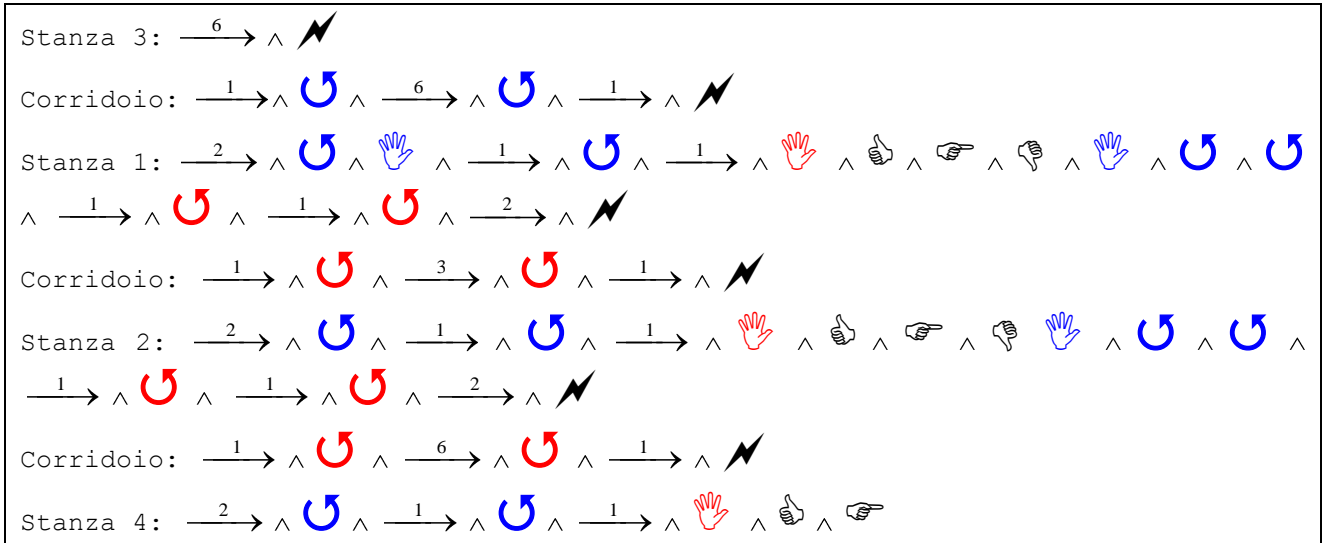
Successivamente, il robottino, sposta la scatola di fronte all'interruttore della luce, vi sale sopra ed accende la luce (figura 12).

Infine Shakey, scende dalla scatola (figura 13) e va nelle altre stanze, ripetendo lo stesso procedimento appena descritto (figura 14, 15, 16, 17, 18).

È interessante notare, (figura 14 e 16) che Shakey, per uscire deve compiere una rotazione a destra di 180°. Tale azione viene implementata compiendo due volte l'operazione "gira di 90°".

Quando il robot, ha acceso tutte le luci, ossia, il problema è stato risolto, la sua posizione finale è come rappresentato in figura 18: sulla scatola nella stanza numero 4.

Possiamo quindi scrivere il codice di quest'ultimo problema:



Il ragionamento con PROLOG

La base di conoscenza

Prima di analizzare i ragionamenti realizzati con il PROLOG, introdurremo la base di conoscenza implementata, riguardante il mondo che circonda Shakey gli oggetti che vi sono contenuti, e le azioni che il robot può compiere.

Tale base di conoscenza, l'abbiamo ripartita in due parti:

- La descrizione del mondo di Shakey:

```
in(scatola1, stanza1).
in(scatola2, stanza1).
in(scatola3, stanza1).
in(scatola4, stanza1).

in(porta1, stanza1).
in(porta2, stanza2).
in(porta3, stanza3).
in(porta4, stanza4).

in(il1, stanza1).
in(il2, stanza2).
in(il3, stanza3).
in(il4, stanza4).

in(porta1, corridoio).
in(porta2, corridoio).
in(porta3, corridoio).
in(porta4, corridoio).

su(scatola1, pavimento).
su(scatola2, pavimento).
su(scatola3, pavimento).
su(scatola4, pavimento).

su(shakey, pavimento).
```

Questa sezione di codice, descrive dove sono posizionati gli oggetti nelle varie stanze, inoltre specifica che le scatole, e Shakey sono inizialmente posizionate sul pavimento.

Quest'ultimo dato ci sarà molto utile nella risoluzione del "problema dell'elettricista" ed "il problema della torre di Pisa".

- Le azioni che Shakey compie per interagire con il mondo esterno, le commenteremo una ad una, al fine di comprenderle meglio:

`in`(shakey, **Stanza**).

Serve ad indicare che Shakey si trova nella stanza "**Stanza**" (generalizzazione);

`spinge`(shakey, **Porta**).

Con questa linea di codice, generalizziamo la condizioni con cui Shakey deve aprire una porta "**Porta**";

`ha`(shakey, **Scatola**).

Shakey, possiede la scatola "**Scatola**" (generalizzazione);

`su`(shakey, **Scatola**).

Shakey è sulla scatola "**Scatola**" (generalizzazione);

`preme`(shakey, **Interruttore**).

Con tale istruzione, Shakey, preme l'interruttore "**Interruttore**" (generalizzazione);

`su`(**Scatola1**, **Scatola2**).

Quest'ultima linea di codice, serve a generalizzare la condizione che la scatola "**Scatola1**" è sulla scatola "**Scatola2**".

Definita la nostra base di conoscenza, possiamo passare ora alle regole di.

Tali regole ci serviranno per risolvere i nostri problemi di logica.

Problema a: "Il facchino"

Il primo problema, il più semplice dei tre, necessita della specifica di quattro regole principali, le quali, ci torneranno utili anche per la risoluzione degli altri due problemi:

```

vai(Partenza, Destinazione) :- in(shakey, Partenza), in(Portal, Partenza),
spinge(shakey, Portal), in(shakey, corridoio), in(Portal, corridoio), in(Porta2,
Destinazione), in(Porta2, corridoio), spinge(shakey, Porta2), in(shakey,
Destinazione).
prendi(Scatola) :- vai(stanza3, stanza1), su(Scatola, pavimento),
ha(shakey, Scatola).
sposta(Scatola, Destinazione2) :- prendi(Scatola), vai(stanza1, Destinazione2).
lascia(Scatola, Destinazione2) :- sposta(Scatola, Destinazione2), su(Scatola,
pavimento).

```

La regola **vai**, serve a far muovere il robottino dalla stanza definita come **Partenza**, alla stanza **Destinazione**.

Guardando le clausole che compongono tale regola, possiamo notare che, Shakey per uscire da una stanza ed entrare in un'altra, spinge le varie porte che incontra.

È interessante notare inoltre, come si "deduce" la stanza di destinazione; il programma sostanzialmente compie una refutazione con le clausole della base di conoscenza, con lo scopo di rintracciare in quali stanze, si trova la porta definita con **Porta2**. Una volta trovata la relativa definizione porta-stanza nella base di conoscenza, Shakey entra nella stanza **Destinazione**.

Con la regola **prendi**, il robot viene in possesso della scatola, il cui numero deve essere scritto al momento dell'immissione della query.

sposta, utilizza le medesime le clausole della regola **prendi**, aggiungendo una clausola **vai** per trasportare la scatola nella stanza **Destinazione2**.

Infine, con la regola **lascia**, Shakey abbandona la scatola in suo possesso, in **Destinazione2**.

Per comprendere meglio quanto esposto, ed il meccanismo con cui opera il PROLOG, illustriamo, a titolo di esempio, il funzionamento della regola **vai**.

Supponiamo quindi di voler andare dalla stanza 1 alla stanza 4; sulla riga di comando del compilatore, inseriremo:

```
?- trace.
```

E successivamente:

```
?- vai(stanza1, stanza4).
```

Il comando **trace**, è molto utile per monitorare il processo di refutazione, il quale restituirà, per la nostra query immessa precedentemente:

```

Call: (7) vai(stanza1, stanza4) ? creep
Call: (8) in(shakey, stanza1) ? creep
Exit: (8) in(shakey, stanza1) ? creep
Call: (8) in(_L187, stanza1) ? creep
Exit: (8) in(scatola1, stanza1) ? creep
Call: (8) spinge(shakey, scatola1) ? creep
Exit: (8) spinge(shakey, scatola1) ? creep
Call: (8) in(shakey, corridoio) ? creep
Exit: (8) in(shakey, corridoio) ? creep
Call: (8) in(scatola1, corridoio) ? creep
Fail: (8) in(scatola1, corridoio) ? creep
Redo: (8) in(_L187, stanza1) ? creep
Exit: (8) in(scatola2, stanza1) ? creep
Call: (8) spinge(shakey, scatola2) ? creep
Exit: (8) spinge(shakey, scatola2) ? creep
Call: (8) in(shakey, corridoio) ? creep
Exit: (8) in(shakey, corridoio) ? creep
Call: (8) in(scatola2, corridoio) ? creep

```

```

Fail: (8) in(scatola2, corridoio) ? creep
Redo: (8) in(_L187, stanza1) ? creep
Exit: (8) in(scatola3, stanza1) ? creep
Call: (8) spinge(shakey, scatola3) ? creep
Exit: (8) spinge(shakey, scatola3) ? creep
Call: (8) in(shakey, corridoio) ? creep
Exit: (8) in(shakey, corridoio) ? creep
Call: (8) in(scatola3, corridoio) ? creep
Fail: (8) in(scatola3, corridoio) ? creep
Redo: (8) in(_L187, stanza1) ? creep
Exit: (8) in(scatola4, stanza1) ? creep
Call: (8) spinge(shakey, scatola4) ? creep
Exit: (8) spinge(shakey, scatola4) ? creep
Call: (8) in(shakey, corridoio) ? creep
Exit: (8) in(shakey, corridoio) ? creep
Call: (8) in(scatola4, corridoio) ? creep
Fail: (8) in(scatola4, corridoio) ? creep
Redo: (8) in(_L187, stanza1) ? creep
Exit: (8) in(portal, stanza1) ? creep
Call: (8) spinge(shakey, portal) ? creep
Exit: (8) spinge(shakey, portal) ? creep
Call: (8) in(shakey, corridoio) ? creep
Exit: (8) in(shakey, corridoio) ? creep
Call: (8) in(portal, corridoio) ? creep
Exit: (8) in(portal, corridoio) ? creep
Call: (8) in(_L188, stanza4) ? creep
Exit: (8) in(porta4, stanza4) ? creep
Call: (8) in(porta4, corridoio) ? creep
Exit: (8) in(porta4, corridoio) ? creep
Call: (8) spinge(shakey, porta4) ? creep
Exit: (8) spinge(shakey, porta4) ? creep
Call: (8) in(shakey, stanza4) ? creep
Exit: (8) in(shakey, stanza4) ? creep
Exit: (7) vai(stanza1, stanza4) ? creep

```

Senza addentrarci nei dettagli tecnici del linguaggio, ci limitiamo ad alcune piccole osservazioni.

Per ogni regola, PROLOG, fa una chiamata alla base di conoscenza, la confronta con le varie clausole in essa contenute, e se trova una refutazione, esce dalla "chiamata" proseguendo con la clausola successiva, altrimenti genera un fallimento, e tramite l'algoritmo di backtracking, implementato nel linguaggio, continua la refutazione con le clausole successive.

Si possono notare infine, dei termini del tipo L187 ed L188, esse indicano l'utilizzo delle variabili, come ad esempio **Porta**.

Problema b: "La torre di Pisa"

Questo problema, necessita dell'ausilio, oltre delle regole precedentemente illustrate, anche di:

```
metti_su(Scatola1, Scatola2) :- prendi(Scatola1), su(Scatola1, Scatola2).
impila([Scatola1, Scatola2, Scatola3, Scatola4]) :- mettis_u(Scatola1,
Scatola2), mettis_u(Scatola2, Scatola3), mettis_u(Scatola3, Scatola4).
```

`impila`, è il risultato della congiunzione logica di più regole `metti_su`; in particolare, quest'ultima regola, fa sì che il robot prenda la scatola `Scatola1` e successivamente la impili su un'altra scatola `Scatola2`.

È da notare che tale operazione di impilare le scatole una sull'altra, viene fatta nella stanza numero 1, e l'agente vede quest'operazione, come la creazione di una lista formata da più termini `Scatola`.

Problema c: "L'elettricista"

Veniamo ora al nostro ultimo problema; come illustrato nel capitolo sul linguaggio visuale, Shakey, deve prendere una scatola dalla stanza 1, portarla nella stanza desiderata, e con l'ausilio di tale scatola, premere l'interruttore della luce.

Per semplicità, nella nostra base di conoscenza, non abbiamo inserito una clausola riguardante lo stato degli interruttori, quindi, con la seguente regola di interrogazione, ci limiteremo a preme far l'interruttore al nostro agente.

```
accendi(Interruttore, Scatola) :- in(Interruttore, Destinazione2),
lascia(Scatola, Destinazione2), su(shakey, Scatola), preme(shakey,
Interruttore).
```

La regola `accendi`, prende come input due parametri, ossia il nome dell'interruttore e la scatola da utilizzare.

Successivamente, il motore di inferenza, cerca la stanza in cui l'interruttore è contenuto, fa in modo che Shakey, prenda la scatola designata, e la lasci nella stanza `Destinazione2`.

Infine, il robot sale sulla scatole e preme l'interruttore desiderato.