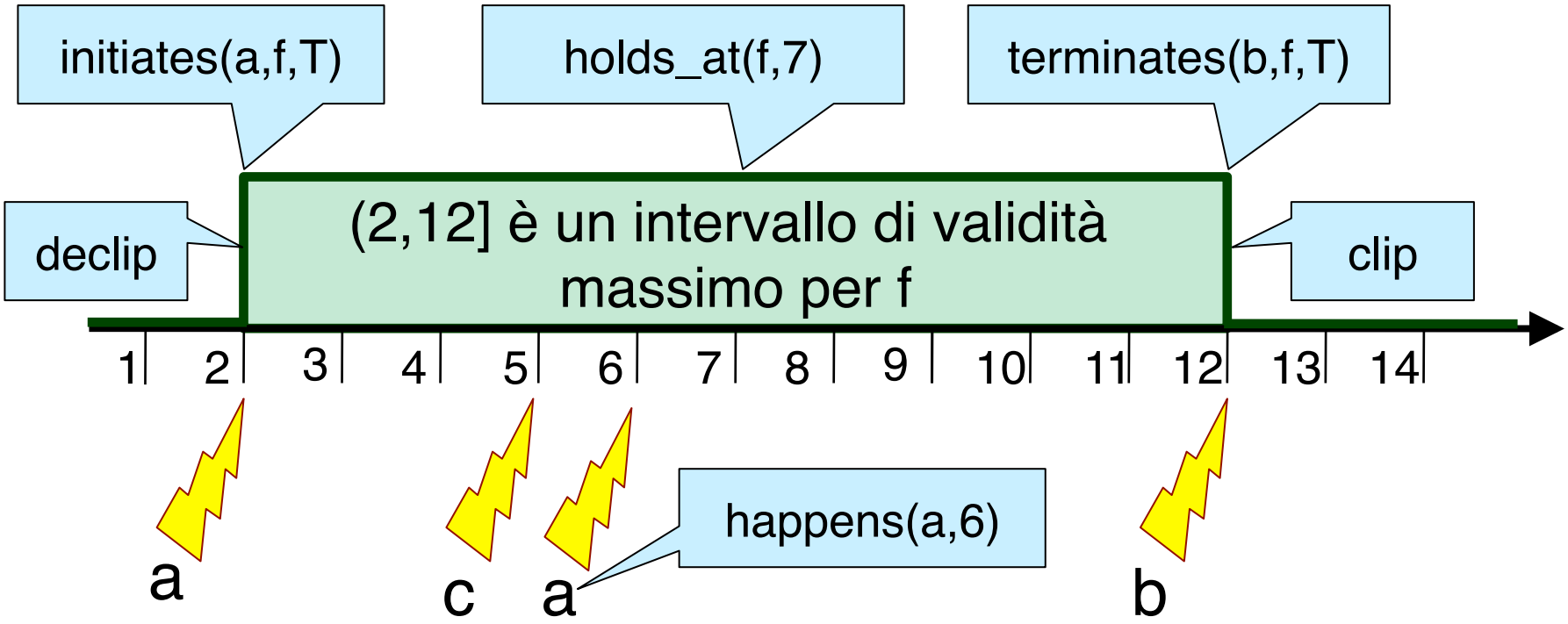
Calcolo degli Eventi (EC)

- Framework basato su FOL per ragionare su azioni, eventi, e tempo
- Permette di modellare e ragionare su proprietà la cui verità varia nel tempo (**fluenti**)
  - Quindi di rappresentare gli **effetti** delle azioni
- Diversi dialetti del calcolo degli eventi
  - Azioni "non atomiche"
  - Traiettorie
  - …
- Noi considereremo una delle forme più semplici del calcolo: ogni azione è atomica
  - L'esecuzione di un'azione è associata ad un evento puntuale (associato a un singolo valore temporale)

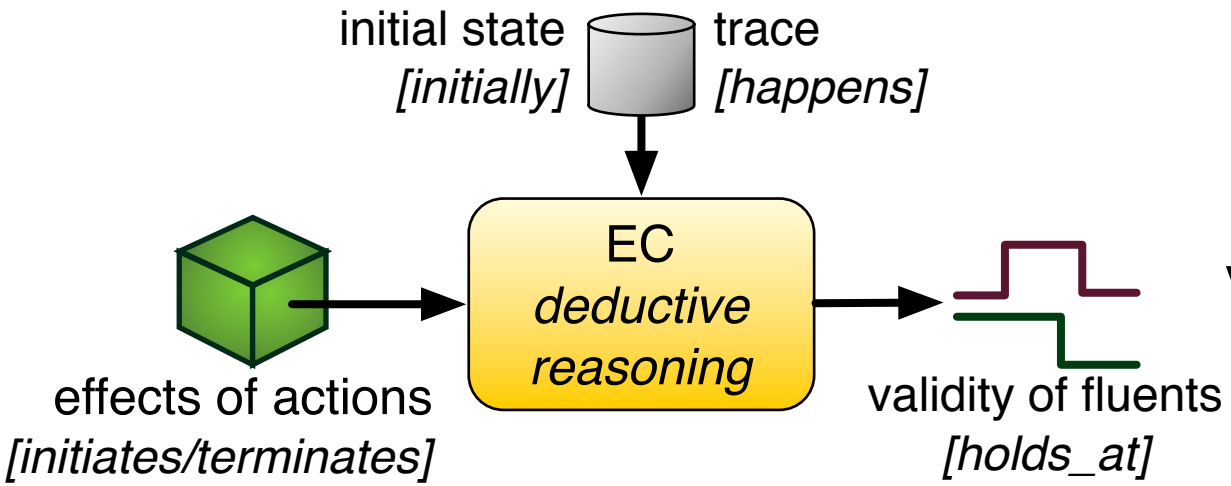
Ontologia di EC 1/2



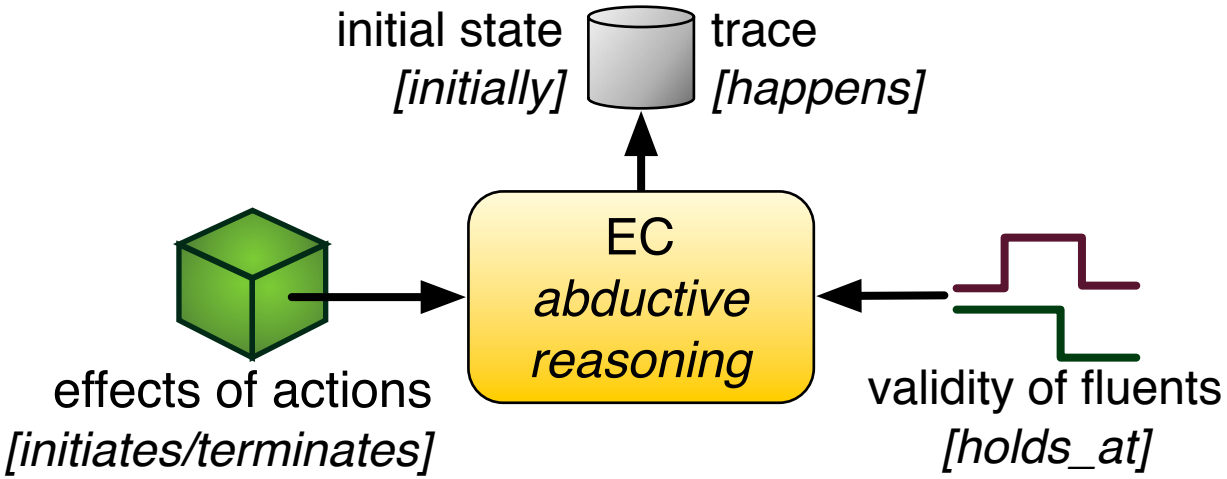
Ontologia di EC 2/2

Predicato	Significato
Descrizione del dominio	
initiates(Ev,F,T)	Ev è capace di iniziare F al tempo T
terminates(Ev,F,T)	Ev è capace di terminare F al tempo T
holds_at(F,T)	F vale al tempo T (nota: F non vale al tempo in cui viene iniziato, ma vale al tempo in cui viene terminato)
Descrizione di una traccia di esecuzione specifica	
initially(F)	Il fluente F vale nello stato iniziale
happens(Ev,T)	L'evento Ev accade al tempo T

Forme di ragionamento



Verifica di tracce



Planning

Differenze col Situation Calculus

- Il calcolo delle situazioni mira a rappresentare COMPLETAMENTE lo stato del sistema
- Nel calcolo degli eventi, l'andamento di ogni fluente rappresenta una parte dello stato
- Il calcolo degli eventi ragiona su intervalli temporali, non sulla transizione da una situazione all'altra
  - Non presenta il frame problem

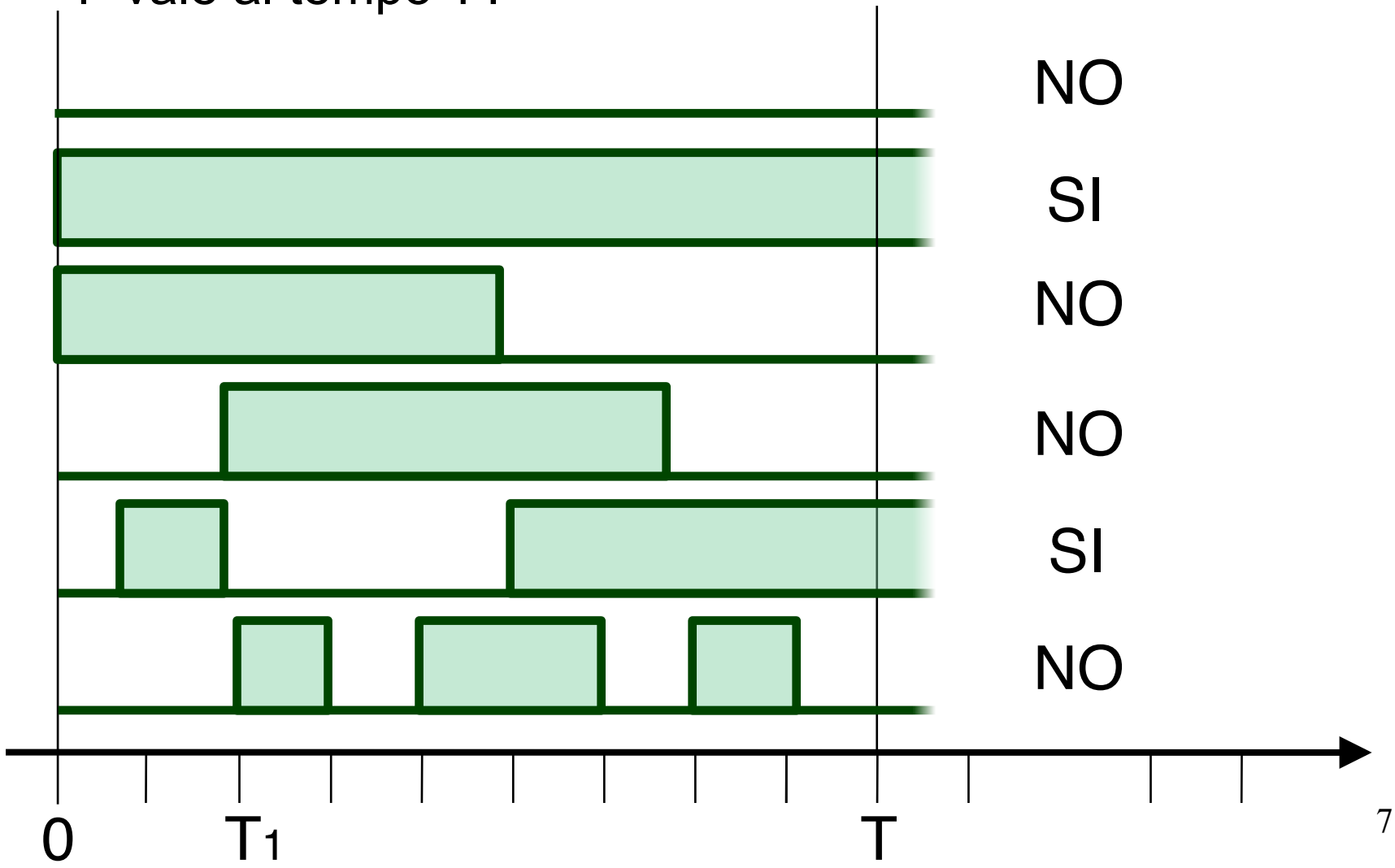
Formalizzazione del Calcolo

- Chi definisce il "significato" dei termini introdotti?
  - Una teoria logica generale
    - indipendente dal dominio
- EC può essere formalizzato mediante
  - Clausole di Horn
  - Negazione per fallimento

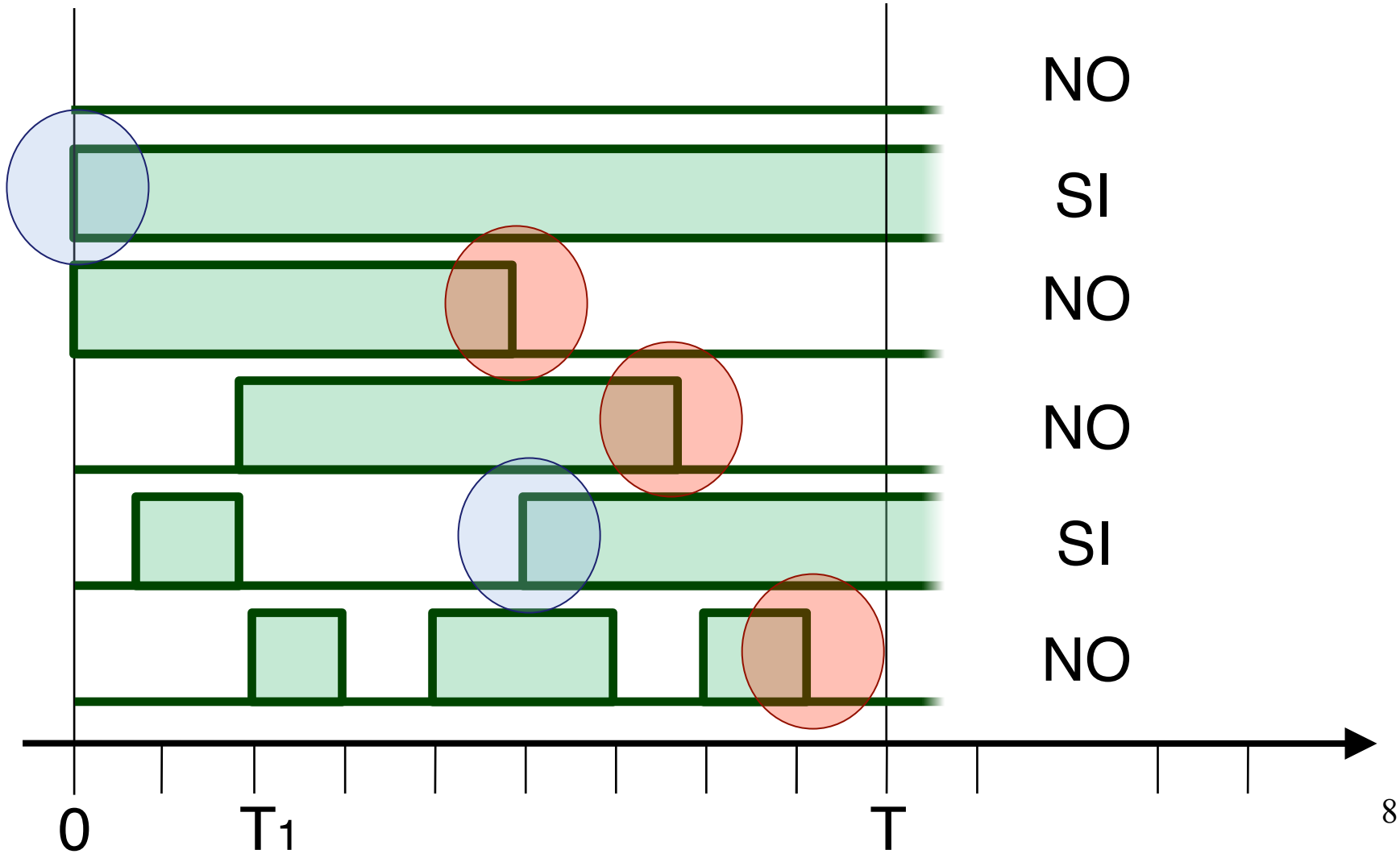
  ➔PROLOG per effettuare ragionamento deduttivo!

Formalizzazione - Idea

- F vale al tempo T?



Formalizzazione - Idea



Formalizzazione – linguaggio naturale

- F *vale* al tempo T se
  - Ad un tempo passato (Tstart) si è verificata una situazione che ha reso vero F
    - O F *vale nello stato iniziale* (Tstart = 0)
    - Oppure al tempo Tstart è *accaduto* un evento che ha *iniziato* F
  - Tra Tstart e T, F **non** è stato bloccato
    - Ovvero non è *accaduto* un evento che ha *terminato* F

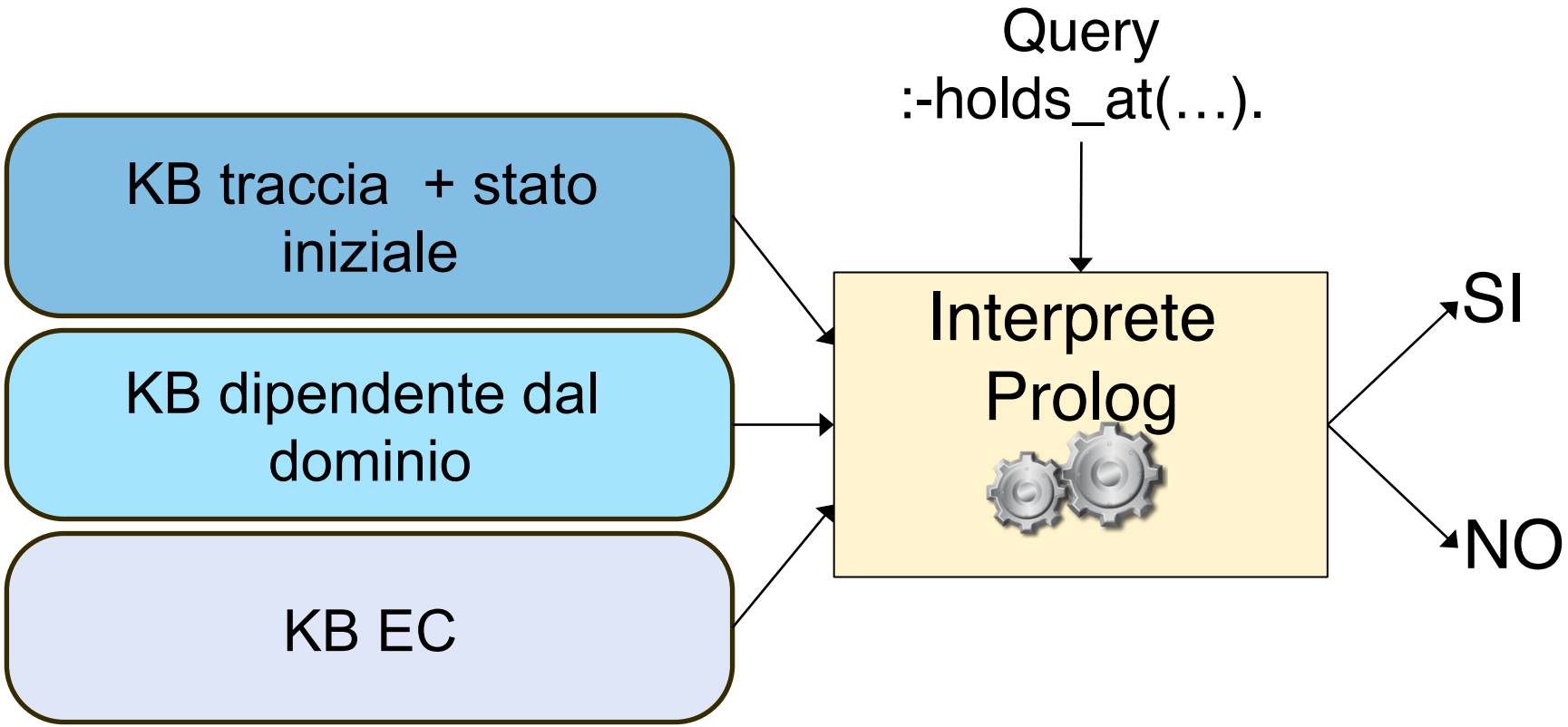
Formalizzazione - Prolog

```
holds_at(F, T):-
   initially(F),
   \+ (clipped(0, F, T)).

holds_at(F, T):-
   happens(Ev,Tstart),
   Tstart < T, Tstart >= 0,
   initiates(Ev, F, Tstart),
   \+ (clipped(Tstart, F, T)).

clipped(T1, F, T3):-
   happens(Ev,T2), T2 >= T1, T2 < T3,
   terminates(Ev, F, T2).
```

Ragionamento deduttivo in Prolog



Esempio – Carrello e azioni simultanee

- Un carrello può essere tirato o spinto
  - Se viene tirato senza essere spinto, si muove indietro (*backward*)
  - Se viene spinto senza essere tirato, si muove in avanti (*forward*)
  - Se viene spinto e tirato contemporaneamente, si mette a girare (*spinning*)
- Similmente per la terminazione degli effetti

Carrello – formalizzazione 1/2

```
initiates(push, forward, T):-
  \+ happens(pull,T).

initiates(pull,backward,T):-
  \+ happens(push,T).

initiates(pull,spinning,T):-
  happens(push,T).
```

Carrello – formalizzazione 2/2

```
terminates(push, backwards, T):-
   \+ happens(pull,T).

terminates(pull, forwards, T).

terminates(pull, backwards, T):-
   happens(push,T).

terminates(push, spinning, T):-
   \+ happens(pull,T).

terminates(pull, spinning, T):-
   \+ happens(push,T).
```

Carrello – Traccia e query

- Traccia di esecuzione

```
happens(push,0).
happens(pull,1).
happens(push,2).
happens(pull,2).
```

- Query con risposta positiva:

```
:- \+ holds_at(spinning, 1).
:- holds_at(backwards, 2).
:- \+ holds_at(backwards, 3).
:- \+ holds_at(forwards, 3).
:- holds_at(spinning, 3).
```

Esercizio – luce 1/2

- Modellare le seguenti azioni
  - **`switch(L)`** modifica lo stato della luce L
    - Se L è accesa, viene spenta
    - Se L è spenta, viene accesa
    - Tali effetti valgono solo se la luce non è rotta
  - **`touch(L)`** causa la rottura della luce, spegnendola

Esercizio – luce 2/2

- Data la seguente traccia:
  - Stato iniziale `initially(on(l)).`
  - Sequenza di eventi

```
happens(switch(l),2).
happens(switch(l),3).
happens(touch(l),4).
happens(switch(l),5).
happens(switch(l),7).
```

- Verificare le seguenti query:

```
:-holds_at(broken(l),5).
:-holds_at(on(l),2).
:-holds_at(on(l),3).
:-holds_at(off(l),3).
:-holds_at(off(l),4).
:-holds_at(off(l),5).
```

Soluzione 1/2

```
terminates(touch(L),on(L),T).
initiates(touch(L),broken(L),T).
initiates(touch(L),off(L),T).

%quando la luce è spenta…
initiates(switch(L),on(L),T):-
  holds_at(off(L),T),
  \+ holds_at(broken(L),T).

terminates(switch(L),off(L),T):-
  holds_at(off(L),T),
  \+ holds_at(broken(L),T).
```

Soluzione 1/2

```
%quando la luce è accesa…
initiates(switch(L),off(L),T):-
  holds_at(on(L),T),
  \+ holds_at(broken(L),T).

terminates(switch(L),on(L),T):-
  holds_at(on(L),T).
```

Esercizio – robot 1/2

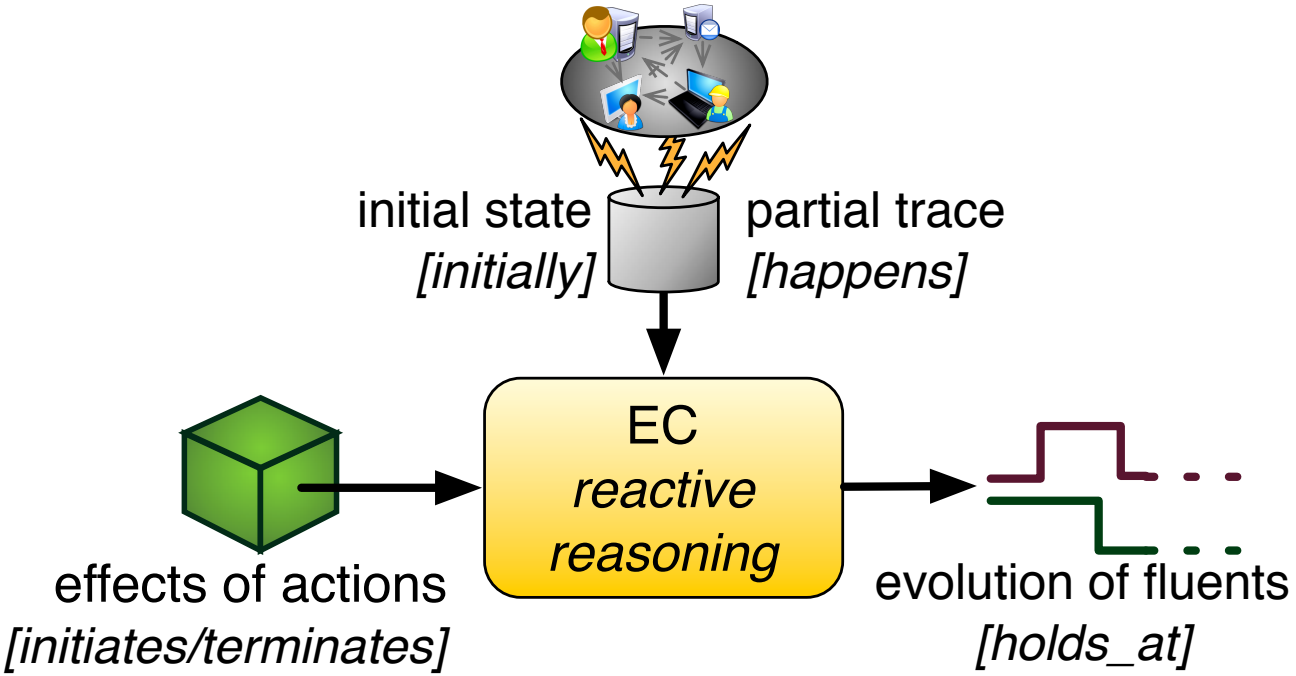
- Si supponga di descrivere un ambiente costituito da stanze connesse una all'altra come segue:
  - **`connected(stanza1,stanza2)`** indica che dalla stanza1 è possibile spostarsi nella stanza2
- Una serie di robot si possono muovere all'interno dell'ambiente
- La descrizione dello stato attuale dei robot si avvale dei seguenti fluenti
  - **`at(R,X)`** → il robot R è nella stanza X
  - **`status(battery(R),L)`** → la batteria del robot R ha livello L
  - Lo stato iniziale fissa la posizione iniziale e il livello di batteria iniziale dei vari robot (initially)

Esercizio – robot 2/2

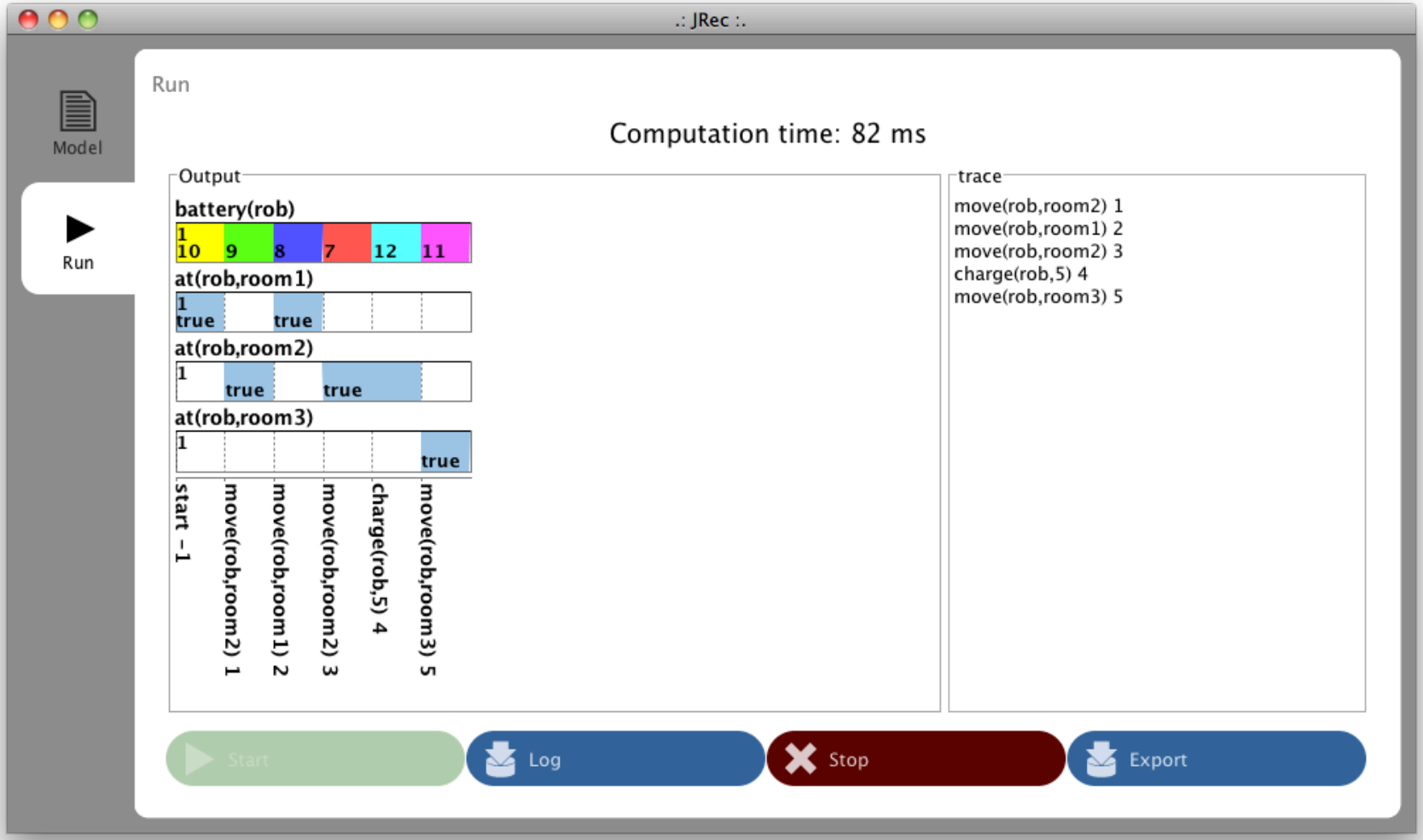
- Modellare le seguenti azioni:
  - **`move(R,Y)`** modifica la posizione del robot (e il livello di batteria)
    - Se si trova attualmente in una stanza connessa a Y
    - Se la batteria ha almeno una unità
    - Lo spostamento causa la perdita di un'unità della batteria
  - **`charge(R,V)`** carica di V unità la batteria di R

Calcolo degli Eventi Reattivo

- Sviluppato dal gruppo di Intelligenza Artificiale del DEIS
  - Permette di monitorare dinamicamente l'esecuzione del sistema, mostrando lo stato dei fluenti

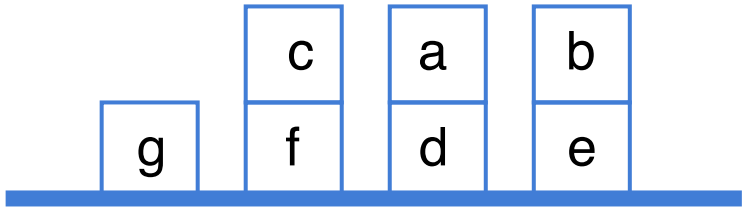


jREC


.: JRec :.
Model
Run
Run
Computation time: 82 ms
Output
battery(rob)
1
10
9
8
7
12
11
at(rob,room1)
1
true
true
at(rob,room2)
1
true
true
at(rob,room3)
1
true
start -1
move(rob,room2) 1
move(rob,room1) 2
move(rob,room2) 3
charge(rob,5) 4
move(rob,room3) 5
trace
move(rob,room2) 1
move(rob,room1) 2
move(rob,room2) 3
charge(rob,5) 4
move(rob,room3) 5
Start
Log
Stop
Export

PLANNING DEDUTTIVO

Prendiamo l'esempio visto a lezione e usiamo la formulazione di Kowalski:



Stato iniziale

holds(on(a,d),s0).

holds(on(b,e),s0).

holds(on(c,f),s0).

holds(clear(a),s0).

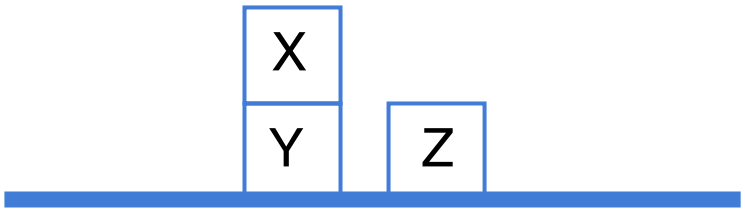
holds(clear(b),s0).

holds(clear(c),s0).

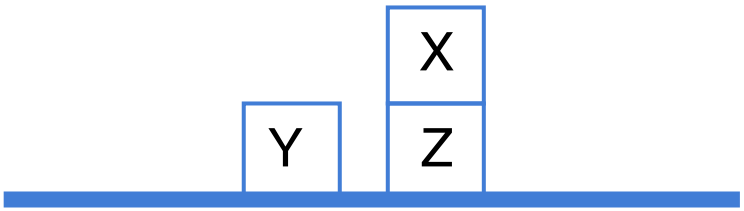
holds(clear(g),s0).

PLANNING DEDUTTIVO

Consideriamo l'unica azione move(X,Y,Z)



on(X,Y)
clear(X)
clear(Z)



clear(Y)
on(X,Z)

PLANNING DEDUTTIVO

%Effetti dell'azione move(X,Y,Z):

holds(clear(Y),do(move(X,Y,Z),S)).
holds(on(X,Z),do(move(X,Y,Z),S)).

%Clausola per esprimere le condizioni di frame:

```
holds(V,do(move(X,Y,Z),S)):-
    holds(V,S),
    V\=clear(Z),
    V\=on(X,Y).
```

PLANNING DEDUTTIVO

% Clausola che esprime le precondizioni dell'azione move(X,Y,Z):

```
pact(move(X,Y,Z),S):-
    holds(clear(X),S), holds(clear(Z),S),
    holds(on(X,Y),S), X\=Z.
```

%Clausola per esprimere la raggiungibilità di uno stato:

```
poss(s0).
poss(do(U,S)):-
    poss(S),
    pact(U,S).
```

PLANNING DEDUTTIVO

- NOTA: abbiamo una clausola che esprime condizioni di frame per ogni AZIONE

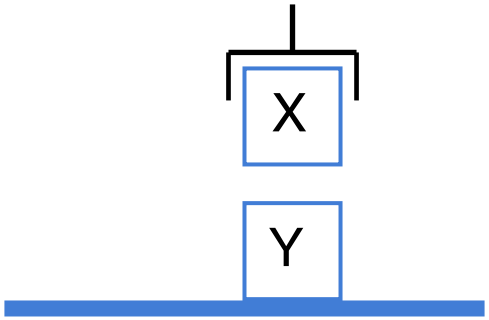
- Goal:

  :- poss(S), holds(on(a,b),S),holds(on(b,g),S).

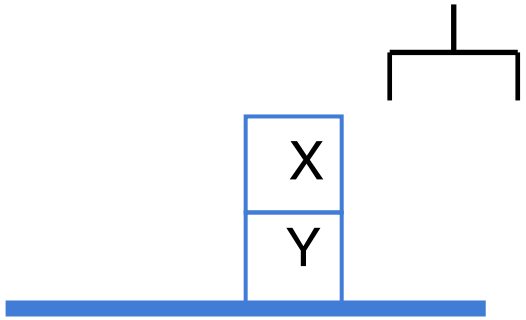
- Attivare il trace per monitorare la risoluzione
- Verificare la costruzione di altri piani con altre query

PLANNING DEDUTTIVO

- Secondo esercizio: usare il mondo a blocchi ma cambiare e modellare le azioni

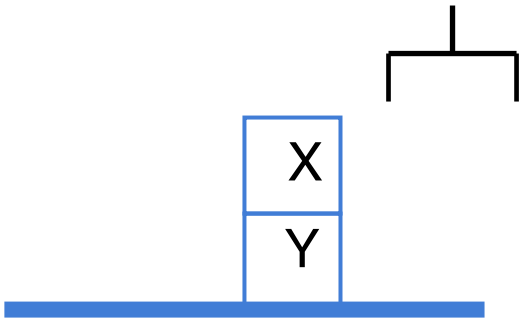


clear(Y), holding(X)

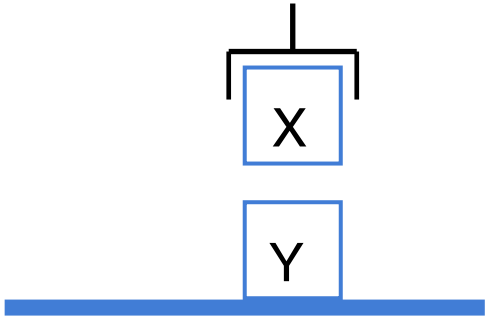


on(X,Y), handempty, clear(X)

STACK(X,Y)



on(X,Y), handempty, clear(X)

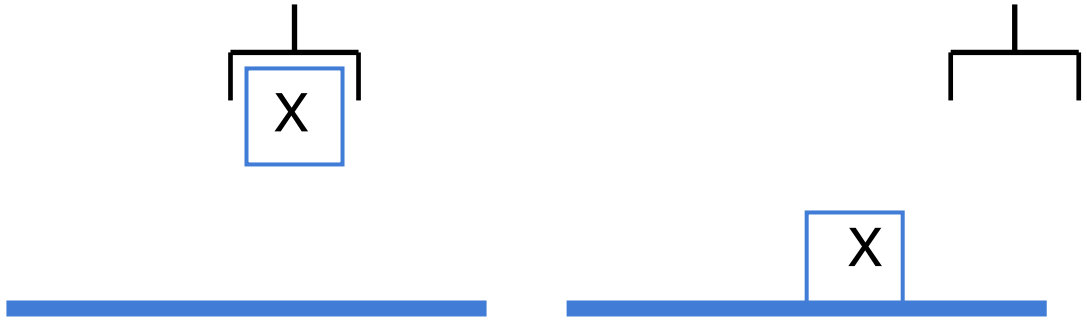


clear(Y), holding(X)

UNSTACK(X,Y)

PLANNING DEDUTTIVO

- Secondo esercizio: usare il mondo a blocchi ma cambiare e modellare le azioni



PUTDOWN(X)

holding(X)

ontable(X), handempty, clear(X)

PICKUP(X)

ontable(X), handempty, clear(X)

holding(X)

PLANNING DEDUTTIVO

Si modellino ora le seguenti azioni

Caricamento di un oggetto

```
load(Oggetto,Carrello,Location)
PREC:   at(Oggetto,Location), at(Carrello,Location)
ADD LIST: in(Oggetto,Carrello)
DELETE LIST: at(Oggetto,Location)
```

Trasporto

```
drive(Carrello,Location1,Location2)
PREC:at(Carrello,Location1), connected(Location1,Location2)
ADD LIST: at(Carrello,Location2)
DELETE LIST: at(Carrello,Location1)
```

Scaricamento di un oggetto

```
unload(Oggetto,Carrello,Location)
PREC:at(Carrello,Location), in(Oggetto,Carrello)
ADD LIST: at(Oggetto,Location)
DELETE LIST: in(Oggetto,Carrello)
```

PLANNING DEDUTTIVO

Con il seguente stato iniziale e goal

Stato iniziale:
`in(carico1,carrello1), at(carrello1,milano)`
`connected(milano,bologna), connected(bologna,roma)`

Stato goal: `at(carico1,roma)`