# Calcolo degli Eventi

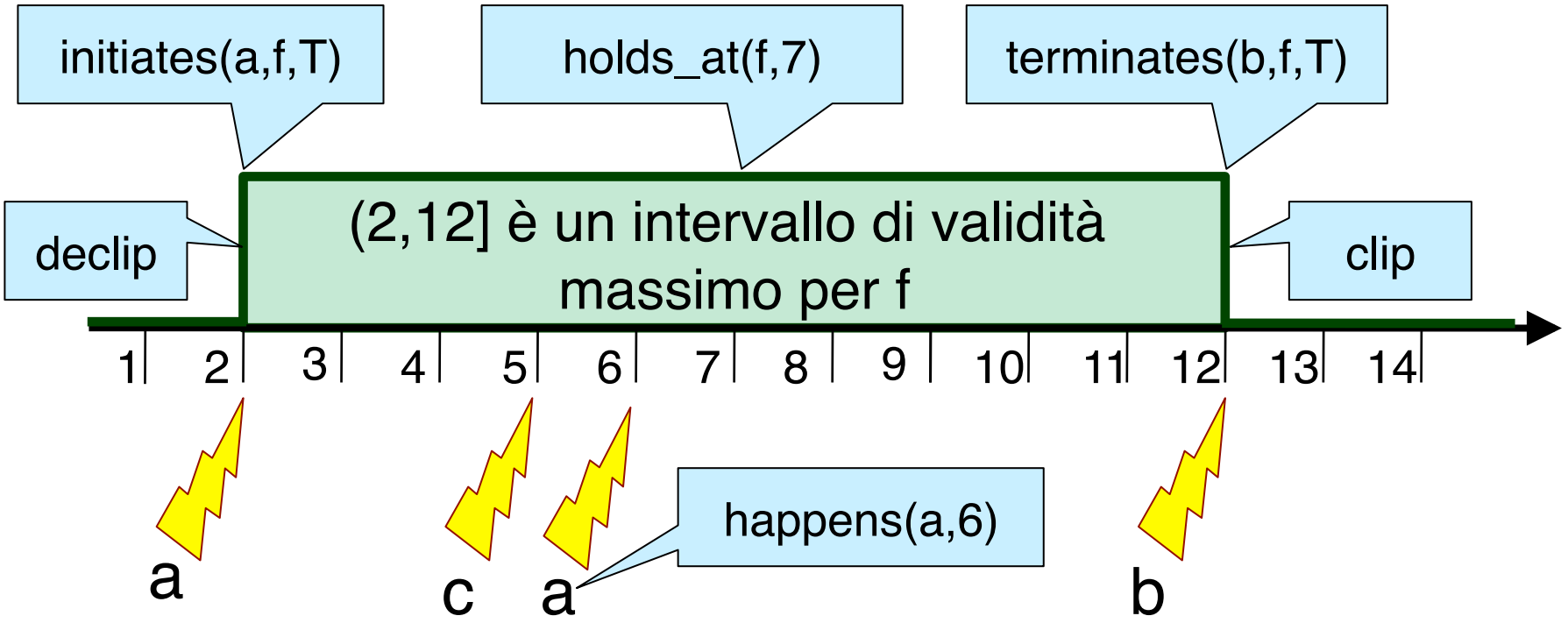
Fondamenti di Intelligenza Artificiale M

A.A. 2009/2010

# 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
- Gli eventi rappresentano l'esecuzione di azioni o il verificarsi di "qualcosa di rilevante" nell'ambiente
- Diversi dialetti del calcolo degli eventi
  - Eventi "non atomici"
  - Traiettorie
  - …
- Noi considereremo una delle forme più semplici del calcolo
  - Eventi puntuali, ovvero associati ad un singolo valore temporale

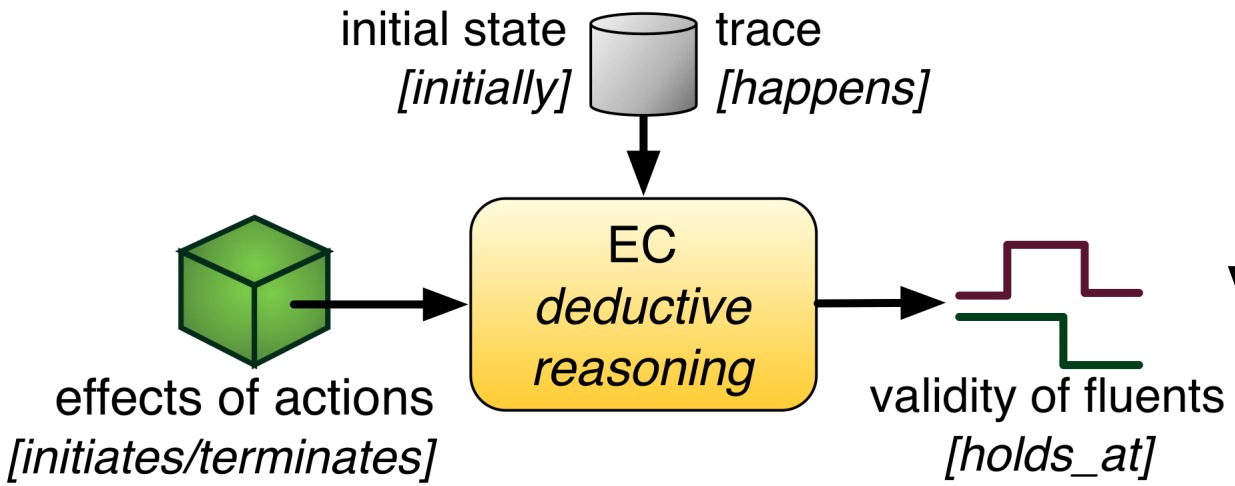
# Ontologia di EC 1/2



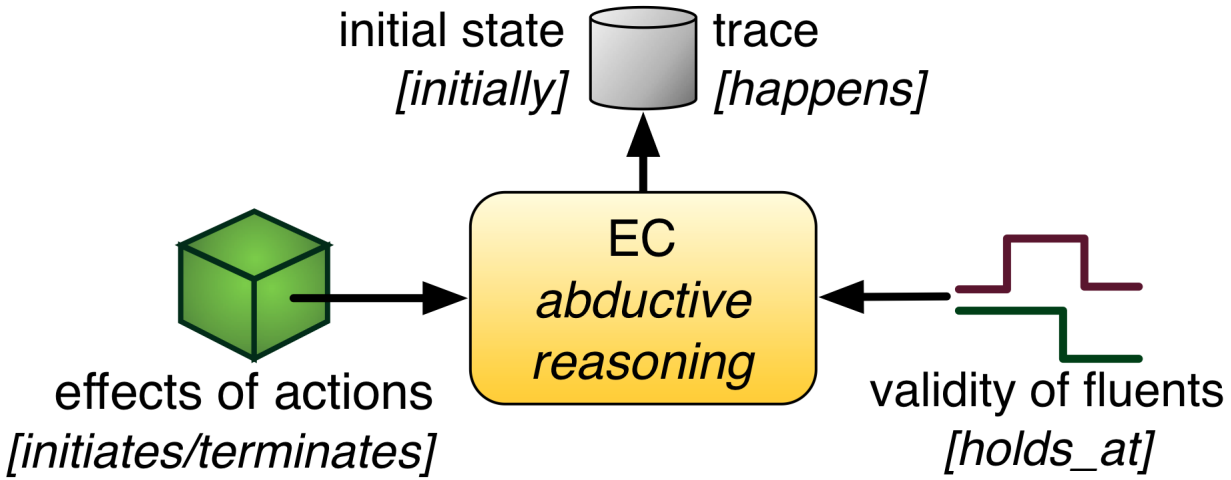
# Ontologia di EC 2/2

| 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<br>(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

# Event vs Situation Calculus

- Il calcolo degli eventi ragiona su intervalli temporali, non sulla transizione da una situazione all'altra
- 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, gestita separatamente dalle altre

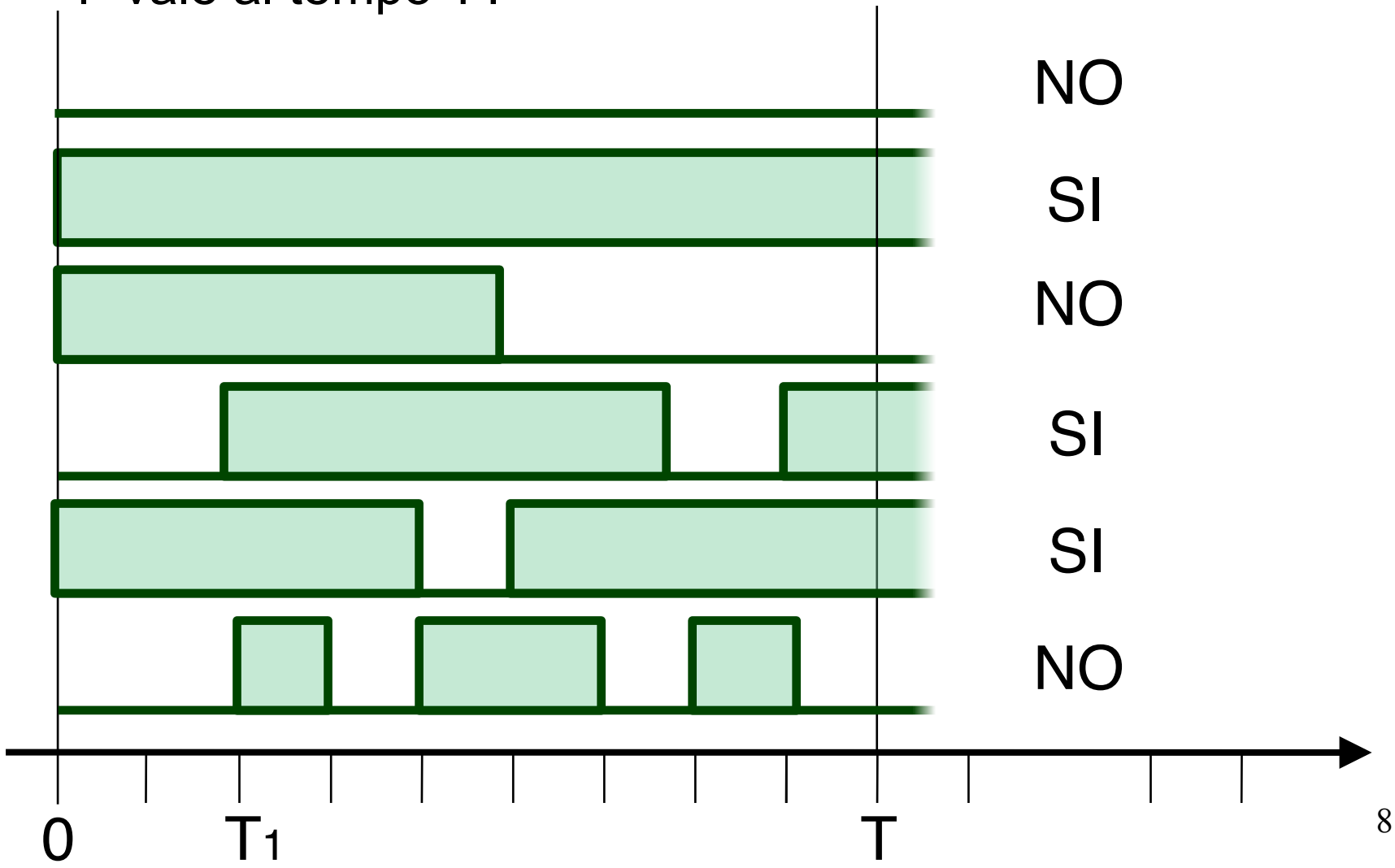
# Formalizzazione del Calcolo

- 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!

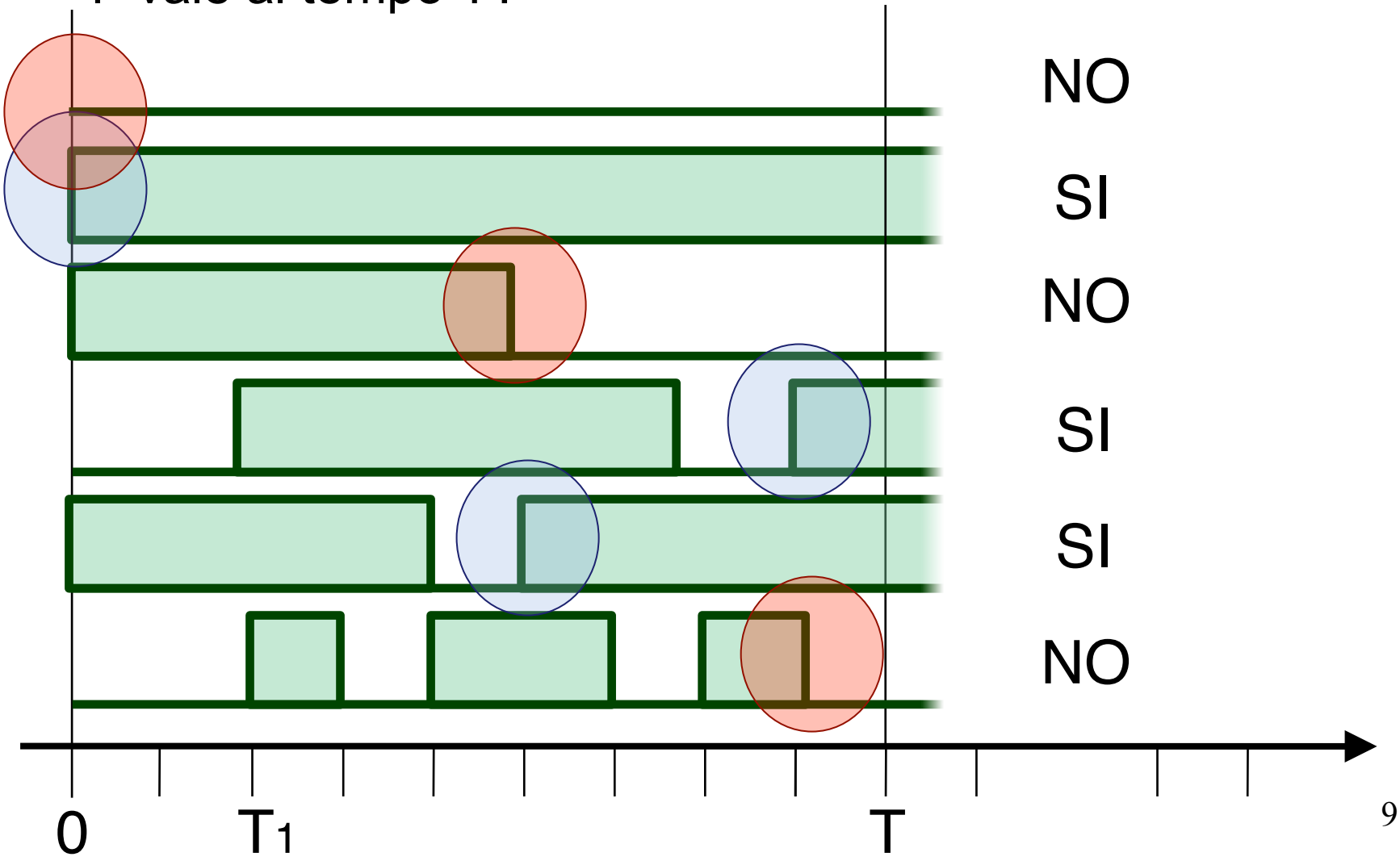
# Formalizzazione - Idea

- F vale al tempo T?



# Formalizzazione - Idea

- F vale al tempo T?



# 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

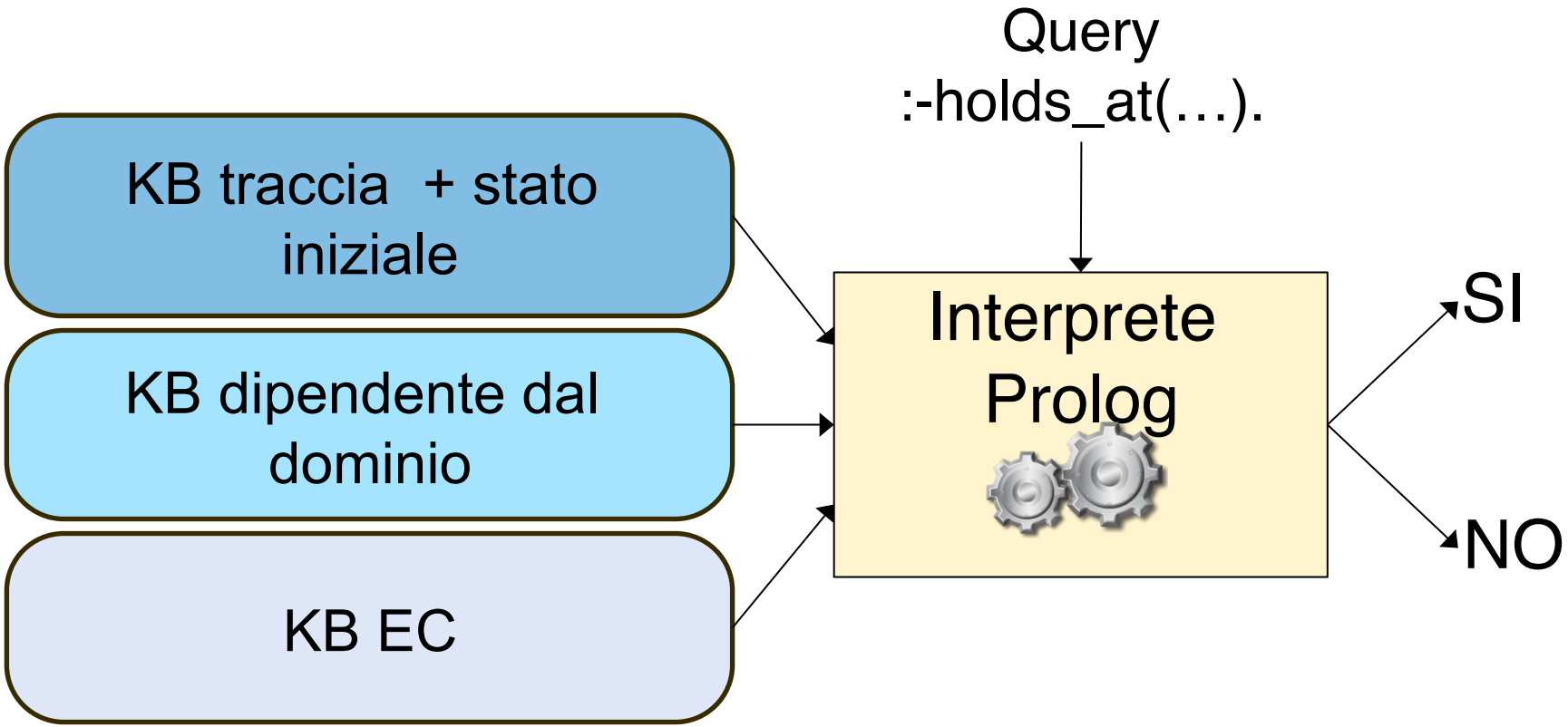
# Formalizzazione - Prolog

```
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 di successo:

```
:- \+ 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).
```

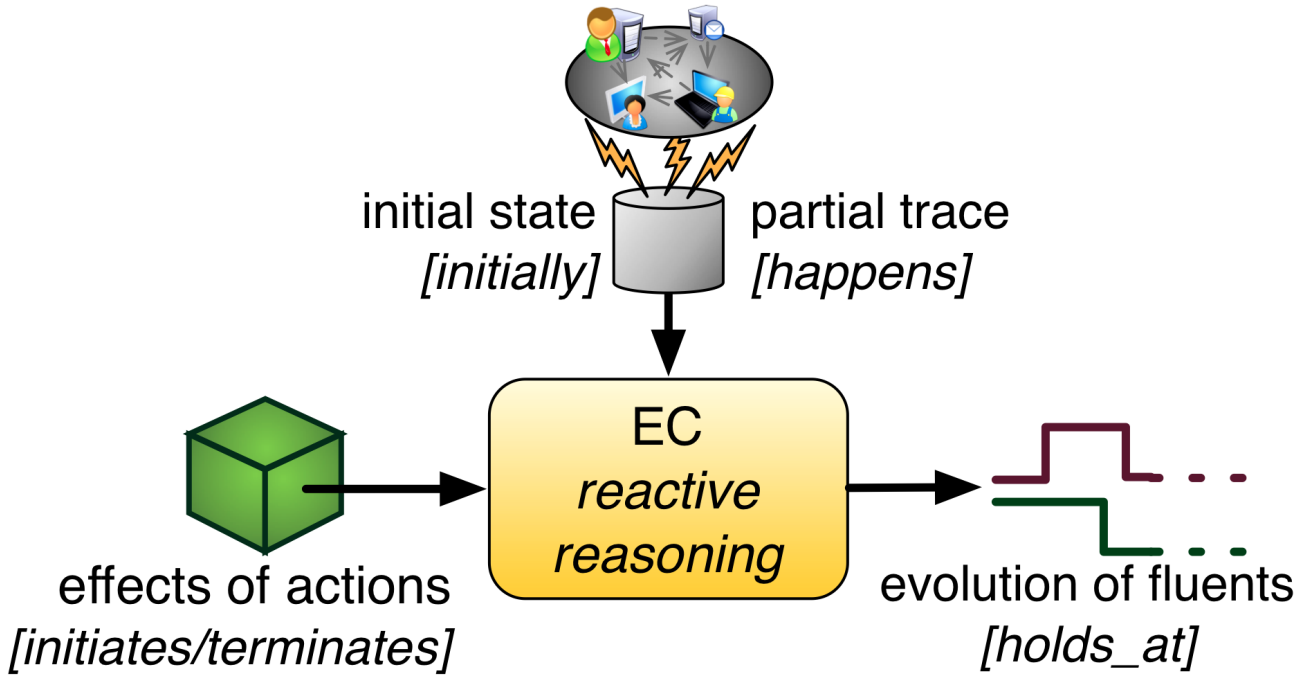
# Soluzione 1/2

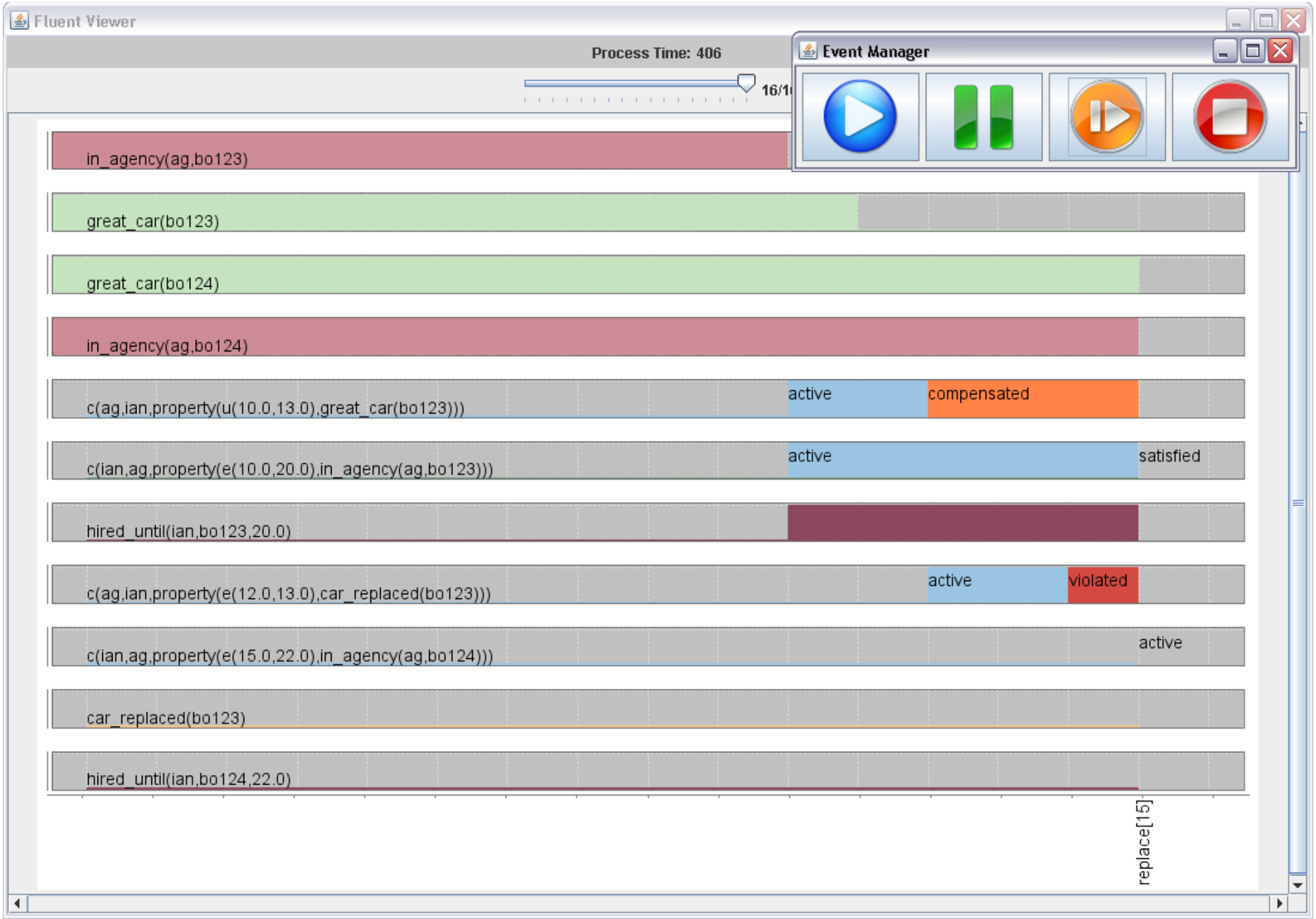
```
%quando la luce è accesa…
initiates(switch(L),off(L),T):-
  holds_at(on(L),T),
  \+ holds_at(broken(L),T).

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

# 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






*Credits: Alessandro Zorzi*