Pianificazione con ASP

Pianificazione con logica proposizionale o ASP

Un approccio alla pianificazione è basato sulla verifica di **soddisfacibilità** di una formula in logica proposizionale (**SATPLAN**).
La formula rappresenta un problema di pianificazione, e un **modello** della formula rappresenta una soluzione.

Un approccio analogo può essere usato per fare **pianificazione con ASP**.

La differenza principale è che SATPLAN formula il problema con la logica classica proposizionale, mentre la pianificazione con ASP modella il problema con **regole** della programmazione logica proposizionale, che consentono l'uso della **negazione per fallimento**.

Ambedue gli approcci hanno affinità con i **grafi di pianificazione** usati da GRAPHPLAN.

Formulare un problema di planning con ASP

Un piano è una sequenza S_0, A_0, S_1,A_1, …, S_n,A_n, S_{n+1}, dove gli S_i (**stati**) sono insiemi di letterali (**fluenti**) e gli A_i sono **insiemi di azioni**.

Fluenti e azioni hanno un argomento che rappresenta lo stato in cui valgono o, rispettivamente, sono eseguite.

on(b1, b2, 3): il blocco b1 è sul blocco b2 nello stato 3.

¬on(b1, b3, 3): il blocco b1 non è sul blocco b3 nello stato 3.

pickup(b, 2): esegui l'azione di prendere il blocco b nello stato 2.

Ruota di scorta

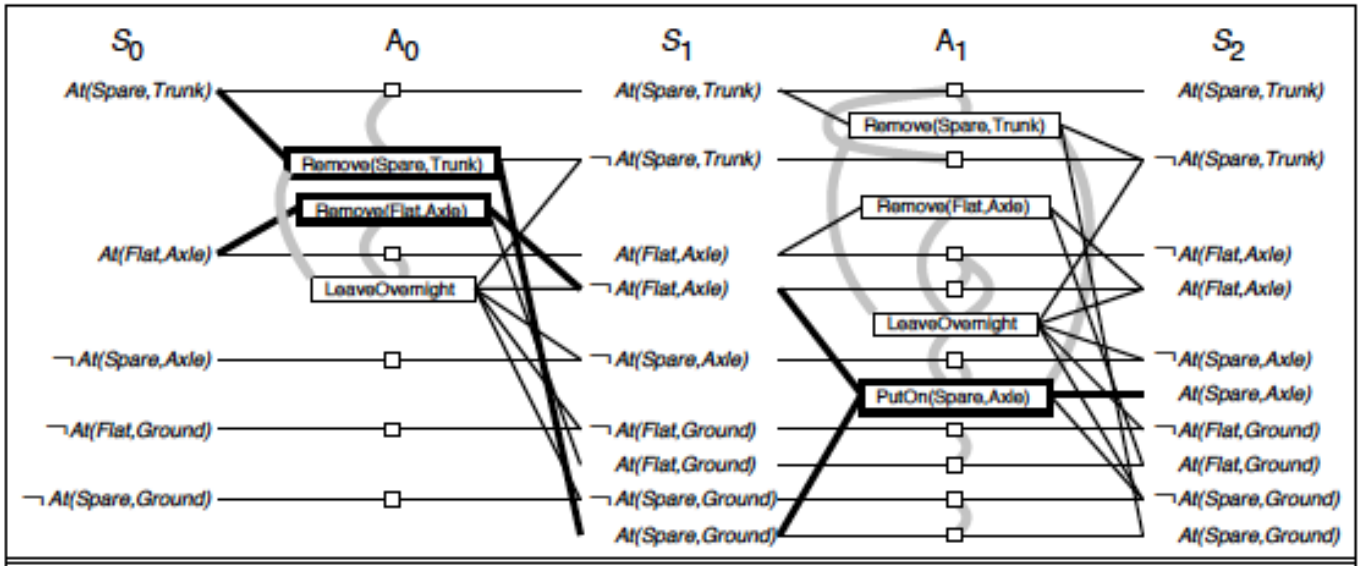

Figure 11.14 The planning graph for the spare tire problem after expansion to level S_2. Mutex links are shown as gray lines. Only some representative mutexes are shown, because the graph would be too cluttered if we showed them all. The solution is indicated by bold lines and outlines.

Ruota di scorta

Le **azioni** sono:

rimuovi(scorta,bagagliaio,S)
rimuovi(bucata,asse,S)
monta(scorta,asse,S)

I **fluenti** sono:

posizione(scorta,bagagliaio,S)
posizione(scorta,pavimento,S)
posizione(scorta,asse,S)
posizione(bucata,pavimento,S)

con S = 1 … n

I fluenti possono essere positivi o negativi.

Non ci sono azioni esplicite di persistenza.

Tralasciamo l'azione *abbandonaDiNotte*.

Ruota di scorta

Il file *ruota di scorta* contiene il programma clingo.

La regola:

```
1{rimuovi(scorta,bagagliaio,S), rimuovi(bucata,asse,S), monta(scorta,asse,S)} :-
      stato(S).
```

specifica che in ogni stato deve essere eseguita **almeno una azione.**

Gli **effetti** delle azioni sono formulati con regole:

```
posizione(scorta,pavimento,S+1):-
      rimuovi(scorta,bagagliaio,S), stato(S).
```

Le **precondizioni** sono espresse come vincoli:

```
:- rimuovi(scorta,bagagliaio,S), not posizione(scorta,bagagliaio,S).
```

Ruota di scorta

La **persistenza** è formulata con regole non monotone:

```
posizione(X,Y,S+1):-
        posizione(X,Y,S), stato(S),
        not -posizione(X,Y,S+1).
```

Il **goal** è formulato con un vincolo:

```
goal:- posizione(scorta,asse,lastlev+1).
:- not goal.
```

Ruota di scorta

In generale, se si permettono più azioni nello stesso livello, occorre introdurre vincoli **mutex**.

I vincoli dei *grafi di pianificazione* relativi a effetti o precondizioni inconsistenti non sono necessari, perché comunque non può esistere un answer set con fluenti inconsistenti.

Viceversa, se ci sono *interferenze* fra gli effetti di una azione a1 e le precondizioni di una azione a2, occorre introdurre un vincolo di mutua esclusione fra a1 e a2.
Nell'esempio questo non dovrebbe accadere.

Ruota di scorta 2

Nell'esempio gli stati sono **completi**, ossia contengono F o ¬F per ogni fluente F.

Lo stesso risultato si può ottenere mantenendo solo le regole con effetti positivi e introducendo una **regola causale**:

```
-posizione(R,P1,S):- ruota(R), pos(P1), stato(S), posizione(R,P,S), P!=P1.
```

che si applica ad ogni stato (v. file *ruota di scorta con regole causali*).

Formulazione di problemi di planning

In generale conviene definire dei predicati dipendenti dal dominio per specificare il tipo dei vari oggetti.
Es. blocco(B), aeroporto(A), merce(C), …

Si suggerisce anche di definire predicati generali come:

action(A): A è una azione
occurs(A, S): l'azione A è eseguita nello stato S

In questo modo si può definire il vincolo:

1{occurs(A,S) : action(A)} :- state(S).

Analogamente si possono definire i predicati:

fluent(F) : F è un fluente
holds(F,S) : il fluente F è vero nello stato S
-holds(F,S) : il fluente F è falso nello stato S

Con questi predicati si può dare un'unica regola per la persistenza valida per tutti i fluenti:

```
holds(F, S+1) :-
        fluent(F), state(S),
        holds(F, S), not  –holds(F, S+1).
-holds(F, S+1) :-
        fluent(F), state(S),
        -holds(F, S), not  holds(F, S+1).
```

Blocchi

Il file *blocchi* contiene una formulazione di un problema di planning per il mondo dei blocchi adattato dal manuale di clingo.

Esiste una unica azione move(B,L), dove B è un blocco e L è o un blocco o il tavolo.