Partial Order Planning (POP)

Pianificazione (non lineare) nello spazio dei piani

Ricerca nello spazio dei piani
Pianificazione Non Lineare

- I pianificatori non lineari sono algoritmi di ricerca che gestiscono la generazione di un piano come un problema di ricerca nello spazio dei piani e non più degli stati:
  - L'algoritmo non genera più il piano come una successione lineare (completamente ordinata) di azioni per raggiungere i vari obiettivi.
  - Nell'albero di ricerca ogni nodo rappresenta un piano parziale e ogni arco un'operazione di raffinamento del piano.
  - Un pianificatore non lineare generativo assume che lo stato iniziale sia completamente noto (**Closed World Assumption**: tutto ciò che non è dichiarato nella rappresentazione dello stato iniziale è falso)
  - Viene adottata generalmente la tecnica del **Least Commitment** sugli ordinamenti: non imporre mai più vincoli di quelli strettamente necessari. Non fare scelte quando non sono imposte evita molti backtracking.

Ricerca nello spazio dei piani
Pianificazione Non Lineare

- Piani parzialmente ordinati (POP)
  - Principio del minimo impegno: non ordinare i passi se non necessario
  - Pianificatore con ordine parziale: i passi sono parzialmente ordinati.
  - Linearizzazione di un piano: imporre un ordine totale a un piano parzialmente ordinato
- Piani parzialmente istanziati
  - Principio del minimo impegno: lasciare le variabili non istanziate finché non è necessario istanziarle
  - Un piano senza variabili si dice totalmente istanziato

Partial Order Planning (POP)

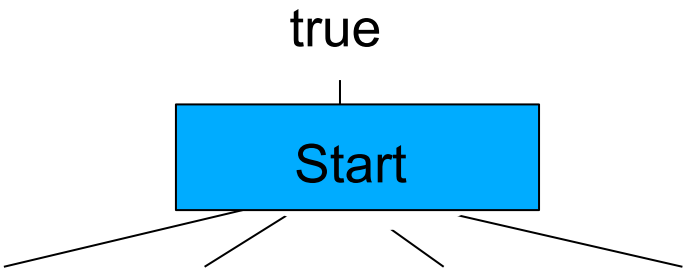
- Si parte da un piano generico
- Ad ogni passo si utilizzano operatori
  - di aggiunta di passi per soddisfare precondizioni
  - di istanziazione di variabili
  - di ordinamento tra passi
- Fino ad arrivare a un piano completo e consistente (tutte le precondizioni soddisfatte e vincoli non violati)
- Ogni sua linearizzazione è una soluzione

Ricerca nello spazio dei piani
Pianificazione Non Lineare

- Un piano non lineare è rappresentato come
  - un insieme di **azioni** (istanze di operatori)
  - un insieme (non completo) di **ordinamenti** fra le varie azioni
  - un insieme di "**causal link**" (descritti dopo)

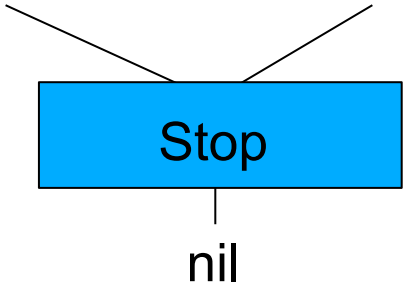
- Piano iniziale: due azioni fittizie
  - **start**: senza precondizioni (true), con effetti una descrizione completa dello stato iniziale
  - **stop**: con precondizione il goal del planner, senza effetti (nil)
  - start < stop

Azioni Start e Stop



Fatti veri nello stato iniziale

Fatti che devono essere veri nello stato goal



Rappresentazioni per i piani

I piani sono:

1. Un insiemi di passi con precondizioni "aperte"
2. Vincoli di due tipi tra i passi
   - Relazioni di ordinamento: $S_1 < S_2$ (S_1 prima di S_2)
   - Link causali: $S_1 \xrightarrow{cond} S_2$ (S_1 realizza **cond** per S_2)

   Nota: Se $S_1 \xrightarrow{cond} S_2$ allora $S_1 < S_2$, ma non viceversa

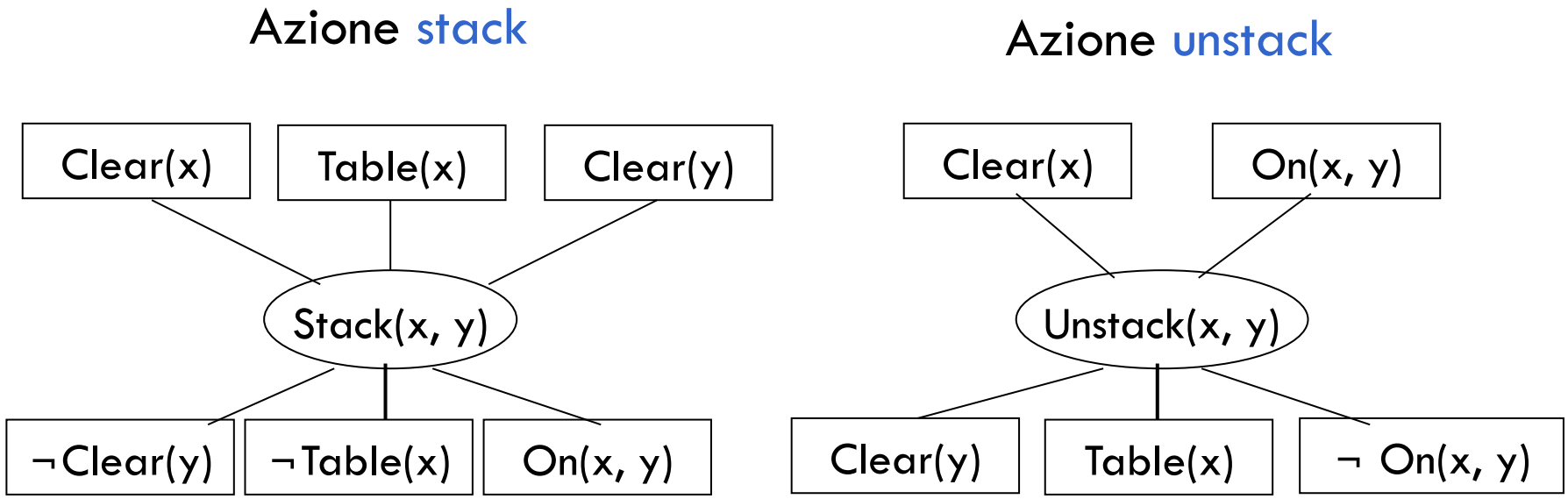
Esempio:

{Unstack(a, b), Unstack(c, d), Stack(b, a), Stack(d, c)}

Unstack(a, b) < Stack(b, a) Unstack(a, b) $\xrightarrow{Clear(b)}$ Stack(b, a)
Unstack(c, d) < Stack(d, c) Unstack(c, d) $\xrightarrow{Clear(d)}$ Stack(d, c)

Rappresentazione per le azioni

- Azioni come in STRIPS (qui visualizzate a grafo)



Ricerca nello spazio dei piani
Pianificazione Non Lineare

- Ad ogni passo si incrementano l'insieme degli operatori, degli ordinamenti parziali (non vengono imposti ordinamenti non richiesti) e dei causal link fino a che tutti i goal sono risolti.

- Una soluzione è un insieme di operatori parzialmente specificato e parzialmente ordinato.

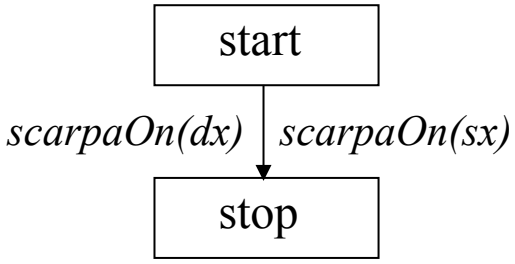
- Per ottenere un piano effettivo si converte l'ordine parziale in uno tra i diversi ordini totali possibili (*operazione di Linearizzazione*)

Esempio: piano per mettersi le scarpe

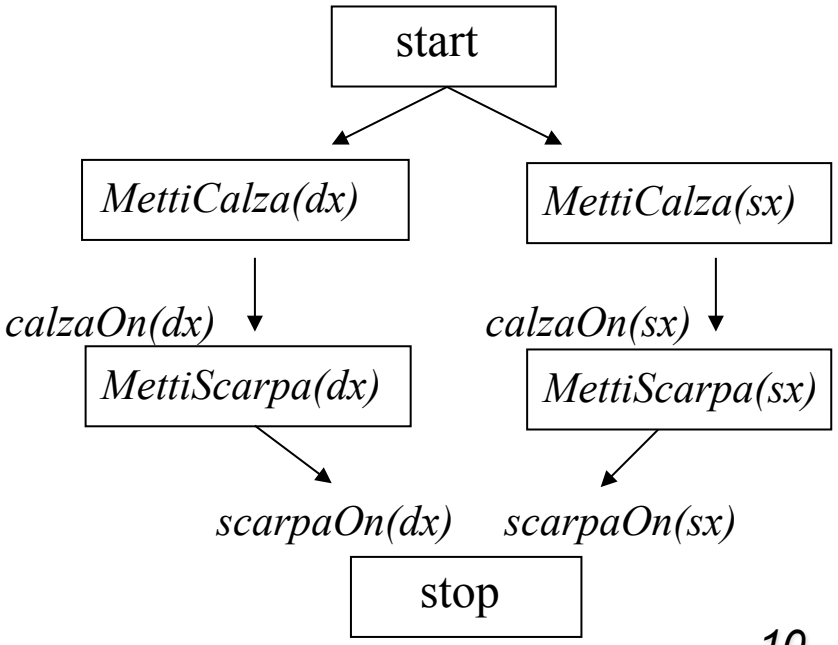
- Goal: scarpaOn(dX), scarpaOn(sX)

- Azioni
  - MettiScarpa(Piede)
    PRECOND: calzaOn(Piede)
    EFFECT: scarpaOn(Piede)
  - MettiCalza(Piede)
    PRECOND: ¬ calzaOn(Piede)
    EFFECT: calzaOn(Piede)

Piano Iniziale



Piani Parziali Finali



Algoritmo di Partial Order Planning (intuitivo)

While (piano non terminato) do

- seleziona una azione SN del piano che ha una precondizione C non soddisfatta;
- seleziona una azione S (nuova o già nel piano) che abbia C tra i suoi effetti;
- aggiungi il vincolo di ordine S<SN;
- se S è nuova aggiungi il vincolo d'ordine Start<S<Stop;
- aggiungi il causal link <S, SN, C>;
- risolvi eventuali violazioni a causal links

End

Algoritmo di Partial Order Planning (intuitivo)

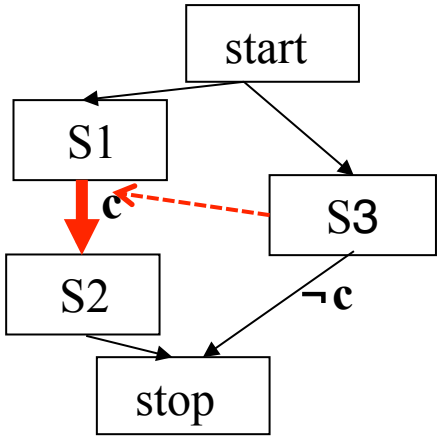
- ognuno dei passi di selezione è non deterministico, in caso di fallimento si può avere backtracking su questi passi.

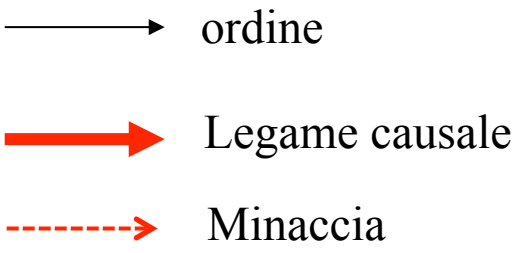
- Un **causal link** $S_i \xrightarrow{c} S_j$ è una terna costituita da due operatori Si, Sj e un sottogoal c tali che uno degli effetti di Si soddisfa la precondizione c di Sj.

Un causal link serve a memorizzare perché un certo operatore è stato introdotto nel piano così da affrontare in modo efficiente il problema delle interazioni fra goal (*interacting goals*).

Violazioni a vincoli causali ("minacce")

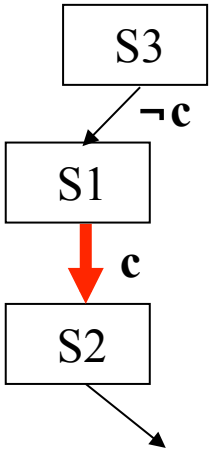
Si dice che un'azione *S3* rappresenta una **minaccia** ('*threat*') per un causal link <*S1,S2,c*> quando contiene un effetto che nega *c* e non c'è nessun vincolo di ordinamento che impedisce a *S3* di essere eseguita dopo di *S1* e prima di *S2*.

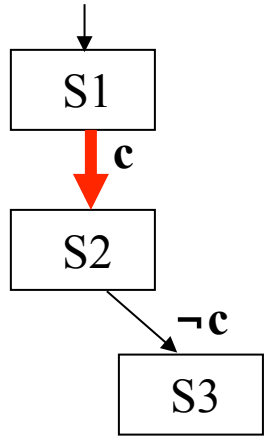




Possibili soluzioni

- Demotion: si impone il vincolo di ordine *S3<S1*
- Promotion: si impone il vincolo di ordine *S2<S3*

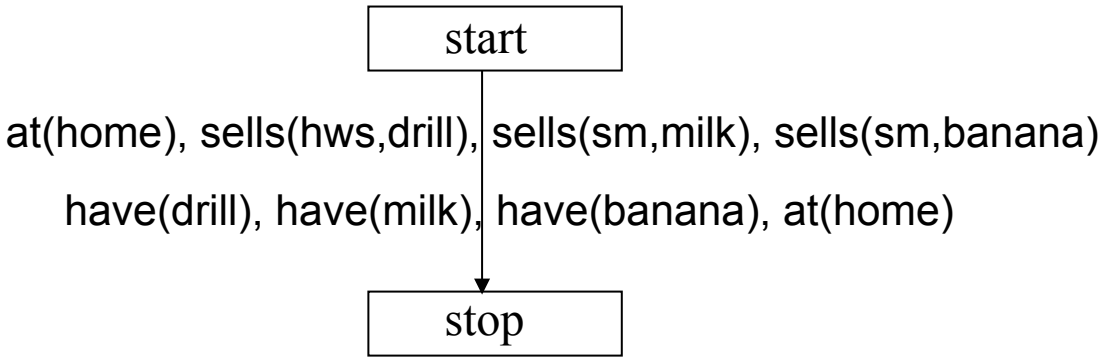




Esempio: Pianificazione di Acquisti

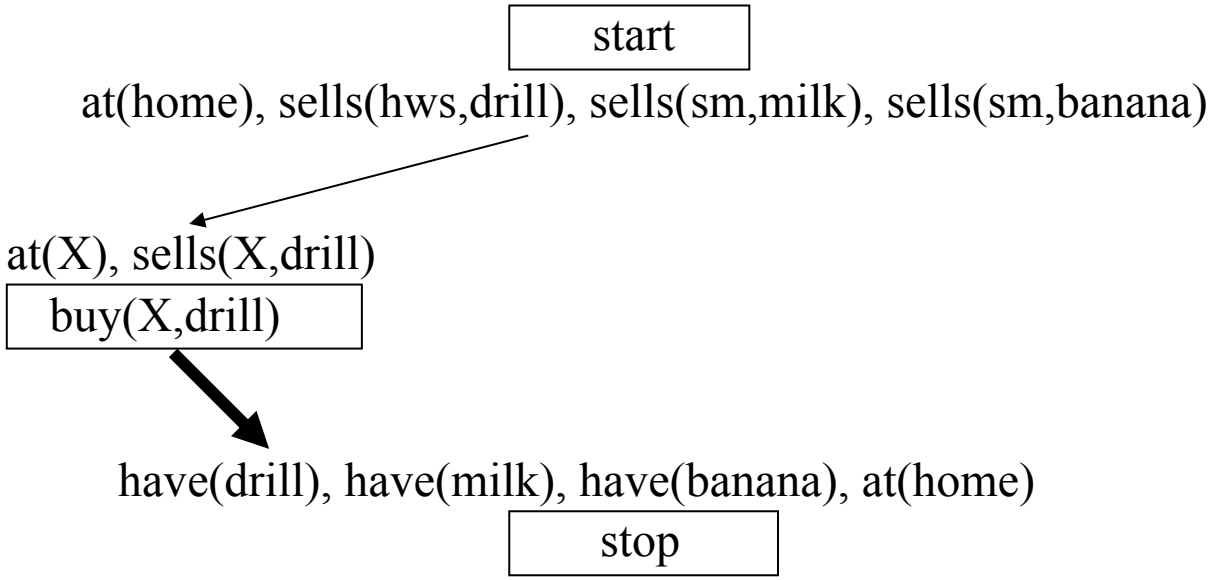
- Stato Iniziale:
  at(home), sells(hws,drill), sells(sm,milk), sells(sm,banana)
- Goal:
  at(home), have(drill), have(milk), have(banana)
- Azioni:
  Go(X,Y):
  PRECOND: at(X)
  EFFECT: at(Y), ¬ at(X)
  buy(S,Y):
  PRECOND: at(S), sells(S,Y)
  EFFECT: have(Y)

- Piano iniziale (detto *nullo*):



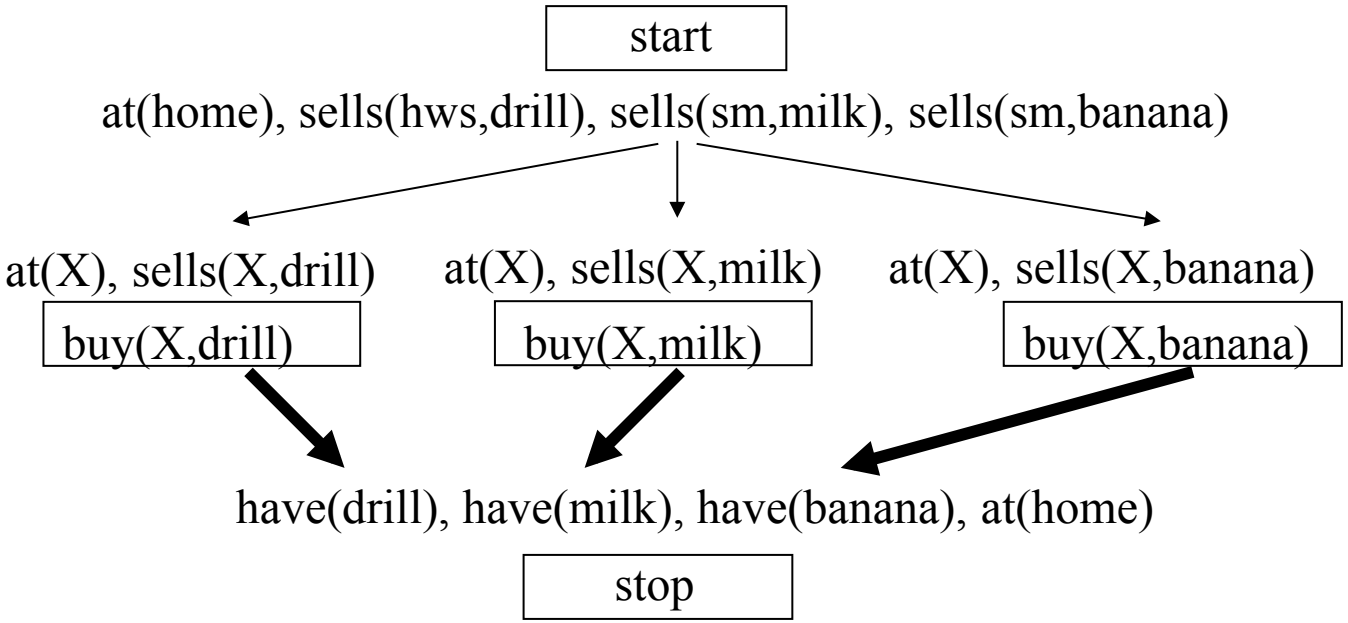
Esempio: Pianificazione di Acquisti

- Primo passo:
  - seleziono una precondizione (goal) da soddisfare: *have(drill)*
  - seleziono una azione che ha *have(drill)* come effetto: *buy(X,Y)*
  - modifico il piano:
    - istanzio Y a *drill*
    - vincoli di ordinamento *Start<buy<Stop*
    - causal link *<buy(X,drill),stop,have(drill)>*



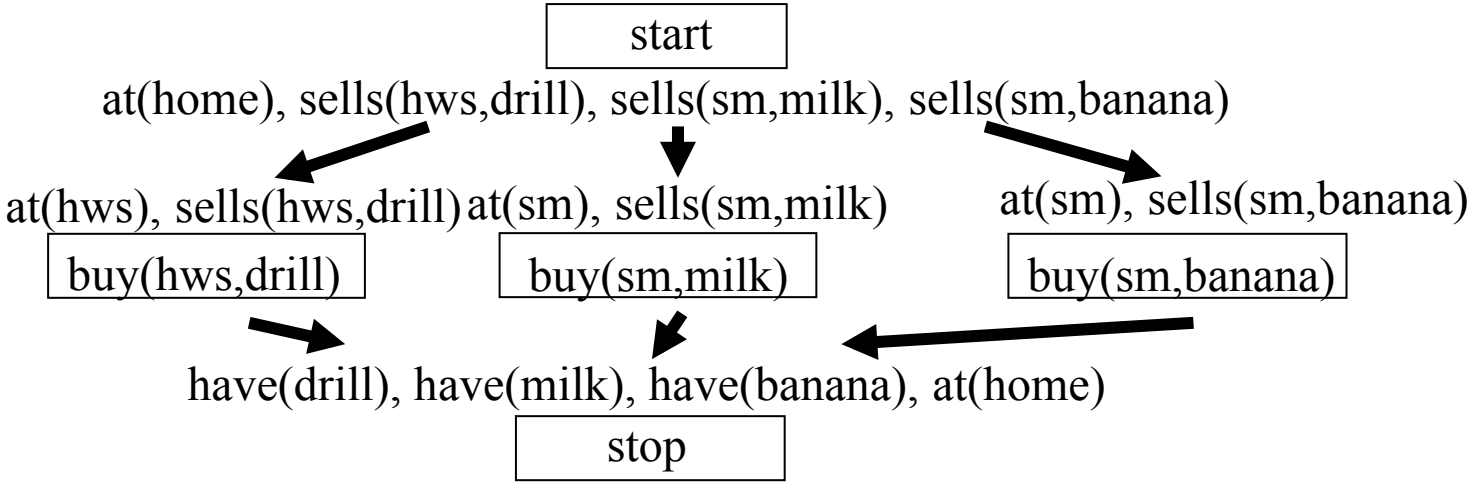
Esempio: Pianificazione di Acquisti

- Procedo in modo analogo per
  - *have(milk)*
  - *have(banana)*

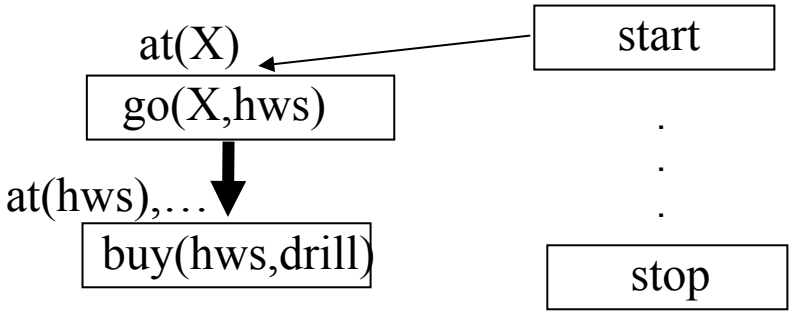


Esempio: Pianificazione di Acquisti

- Seleziono *sells(X,drill),* vera nello stato iniziale con *X=hws*. Stessa cosa per *sells(X,milk)* e *sells(X,banana)* con *X=sm*. Aggiungo i causal link.

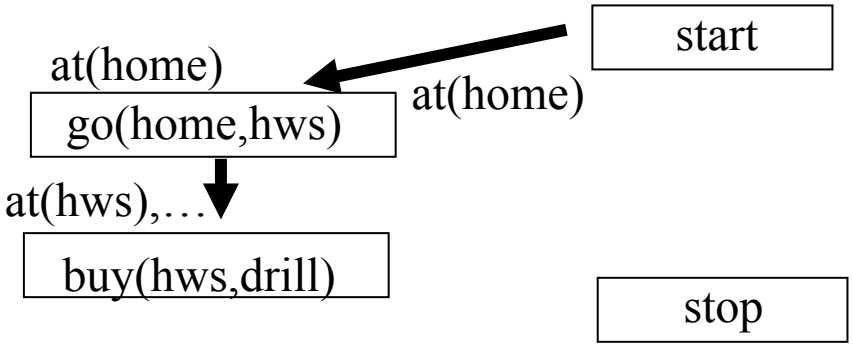


- Seleziono
  - *at(hws)* precondizione di *buy(hws,drill)*
  - aggiungo azione *go(X,hws)* al piano con relativi vincoli di ordinamento e causal link.

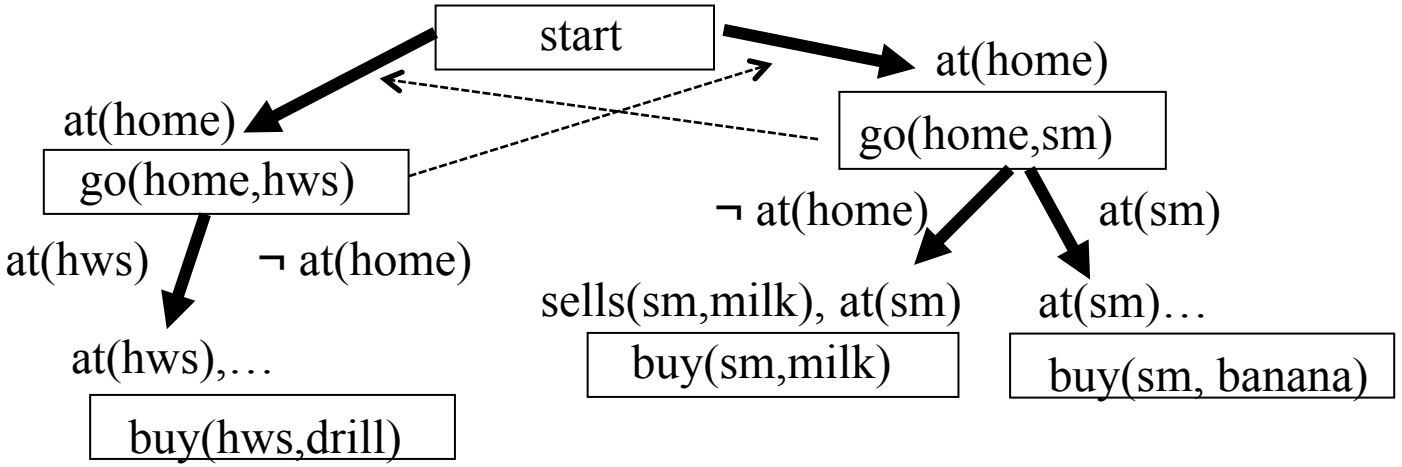


Esempio: Pianificazione di Acquisti

- Seleziono *at(X)* precondizione di *go(X,hws),* soddisfatta da *Start* con *X=home;* proteggo il causal link.

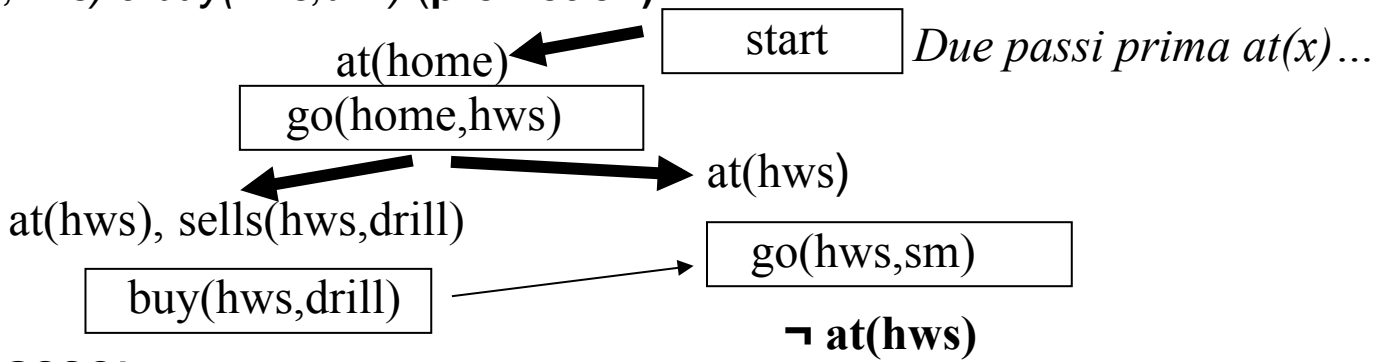


- Procedo in modo analogo per *at(sm)*

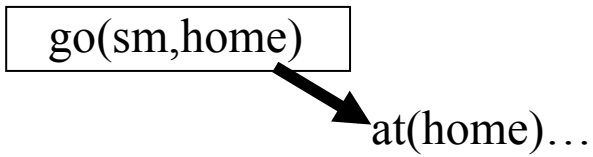


Esempio: Pianificazione di Acquisti

- Occorre risolvere la minaccia fra le azioni *go(home,hws)* e *go(home,sm)*
  - se l'agente esegue *go(home,hws)* non può essere *at(home)* per eseguire azione *go(home,sm)* e viceversa
  - imporre vincoli d'ordine tra le azioni non funziona
  - backtracking sul passo di risoluzione di *at(X)* (precondizione di *go(X,sm))*
  - si soddisfa *at(X)* con l'effetto di *go(home,hws)* (*X=hws*) invece che con *Start* con *X=home*
  - si impone *buy(hws,drill)<go(hws,sm)* in quanto *at(hws)* è protetto dal causal link fra *go(home,hws) e buy(hws,drill)* (**promotion)**

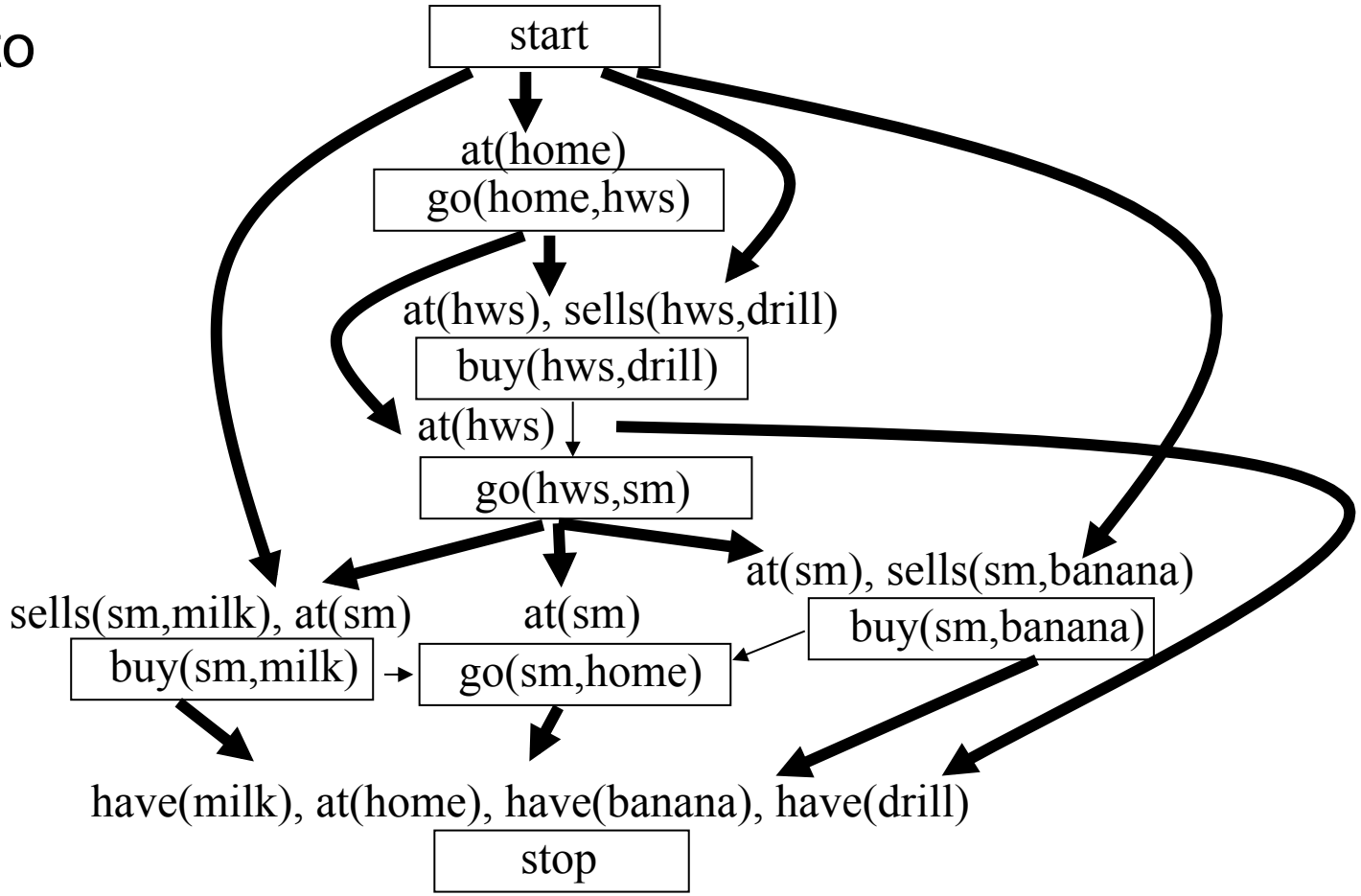


- Ultimo passo:
  - risolvere *at(home)* di *stop*: unico modo è mettere azione *go(home)* prima di *stop*



Esempio: Pianificazione di Acquisti

- Piano completo



Un piano finale completamente ordinato:

1) go(home,hws) *2) buy(hws,drill)* *3) go(hws,sm)*
4) buy(sm,milk) *5) buy(sm,banana)* *6) go(sm,home)*

Algoritmo Partial Order Planning (POP)

```
function POP(initialGoal, operators) returns plan
    plan := INITIAL_PLAN(start, stop, initialGoal)
    loop
        if  SOLUTION(plan) then return plan;
        SN, C := SELECT_SUBGOAL(plan);
        CHOOSE_OPERATOR(plan, operators, SN, C);
        RESOLVE_THREATS(plan)
    end

function SELECT_SUBGOAL(plan)
    seleziona SN da STEPS(plan) con precondizione C non risolta;
    return SN, C
```

Algoritmo Partial Order Planning (POP)

procedure CHOOSE_OPERATOR(plan, ops, SN, C)
    selez *S* da *ops* o da *STEPS(plan)* con effetto *C*;
    if non esiste *S* con tali proprietà then fail;
    aggiungi il link causale <*S,SN,C*>
    aggiungi il vincolo di ordinamento *S* < *SN*
    if *S* è un nuovo operatore aggiunto al piano
    then aggiungi *S* a *STEPS(plan)*
                aggiungi il vincolo *Start* < *S* < *Stop*
procedure RESOLVE_THREAT(plan)
    for each azione *S* che viola causal link fra *Si* e *Sj*,
    choose either
        demotion: aggiungi il vincolo *S* < *Si*
        promotion: aggiungi il vincolo *Sj* < *S*
        if *NOT_CONSISTENT(plan)* then fail

Modal Truth Criterion (MTC)

I due metodi *promotion* e *demotion* da soli non bastano a garantire la soluzione di un qualunque problema risolubile di pianificazione non lineare (completezza del pianificatore).

Il *Modal Truth Criterion* rappresenta un procedimento di costruzione del piano che garantisce la completezza del pianificatore.

Un algoritmo di Partial Order Planning alterna passi di soddisfacimento di precondizioni con passi di risoluzione di minacce.

MTC fornisce 5 metodi di correzione del piano (1 per il soddisfacimento delle precondizioni e 4 per la risoluzione delle minacce) sufficienti a garantire la completezza del pianificatore.

Modal Truth Criterion (MTC)

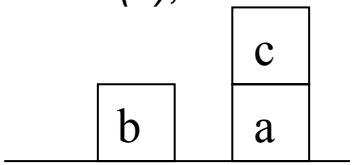
1. ***establishment***, cioè soddisfacimento di una precondizione per mezzo di: (1) inserimento di una nuova azione, (2) di un vincolo di ordinamento con un'azione già nel piano o semplicemente (3) di un assegnamento di variabili.
2. ***promotion***, cioè vincolare una mossa a precederne un'altra nel piano finale;
3. ***demotion***, cioè vincolare una mossa a seguirne un'altra nel piano finale;
4. ***scopertura***, cioè inserire un operatore S2 nuovo o già nel piano (detto *white knight*) fra due mosse vecchie S1 ed S3 tale che i suoi effetti contengano la precondizione di S3 minacciata da S1.
5. ***separazione***, cioè inserire vincoli di *non codesignazione* fra le variabili dell'effetto negativo e della precondizione minacciata in modo da evitare la possibile unificazione.
   Possibile quando queste variabili non sono ancora state istanziate.

Es. Dato il causal link $pickup(X) \xrightarrow{holding(X)} stack(X,b)$

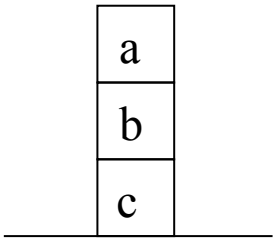
la minaccia rappresentata da un'eventuale azione stack(Y,c) può essere eliminata ponendo il vincolo X\=Y

Esempio: Anomalia di Sussmann

Stato iniziale ([*si*]):
clear(b),clear(c),on(c,a),ontable(a),
ontable(b),handempty.



Goal ([*g*]):
on(a.b),on(b,c).



Azioni

pickup(X)
PRECOND: *ontable(X), clear(X), handempty*
POSTCOND: *holding(X) not ontable(X), not clear(X), not handempty*

putdown(X)
PRECOND: *holding(X)*
POSTCOND: *ontable(X), clear(X), handempty*

stack(X,Y)
PRECOND: *holding(X), clear(Y)*
POSTCOND: *not holding(X), not clear(Y), handempty, on(X,Y), clear(X)*

unstack(X,Y)
PRECOND: *handempty, on(X,Y), clear(X)*
POSTCOND: *holding(X), clear(Y),.not handempty, not on(X,Y), not clear(X),*

Esempio: Anomalia di Sussmann

Piano iniziale: start *[si]* *[g]* stop

Lista degli ordinamenti: [start<stop]
Lista dei causal link: [].
Agenda di goal:[on(a,b), on(b,c)]
Scegliamo le azioni (senza ordinamenti) che soddisfano i goal (**establishment**):

stack(a,b)
PRECOND: *holding(a), clear(b)*
POSTCOND: *handempty, on(a,b), not clear(b), not holding(a)*

stack(b,c)
PRECOND: *holding(b), clear(c)*
POSTCOND: *handempty, on(b,c), not clear(c), not holding(b)*

Lista degli ordinamenti : [start<stop, start<stack(a,b), start<stack(b,c), stack(b,c)<stop, stack(a,b)<stop]
Lista dei causal link : [<stack(a,b),stop,on(a,b)>, <stack(b,c),stop,on(b,c)>].
Agenda di goal : [holding(a),clear(b),holding(b),clear(c)]

Le mosse non sono ordinate fra di loro, ma sappiamo che dovranno essere eseguite entrambe.

Esempio: Anomalia di Sussmann

Alcune delle precondizioni in agenda (*clear(b) e clear(c)*) sono già soddisfatte nello stato iniziale: basta aggiungere dei causal link.

Lista degli ordinamenti: [vd. sopra]
Lista dei causal link: […, <start,stack(a,b),clear(b)>, <start,stack(b,c),clear(c)>].
Agenda di goal:[holding(a),holding(b)]

Dobbiamo aggiungere al piano nuove mosse per raggiungere le precondizioni rimaste in agenda *holding(a)* e *holding(b)*.

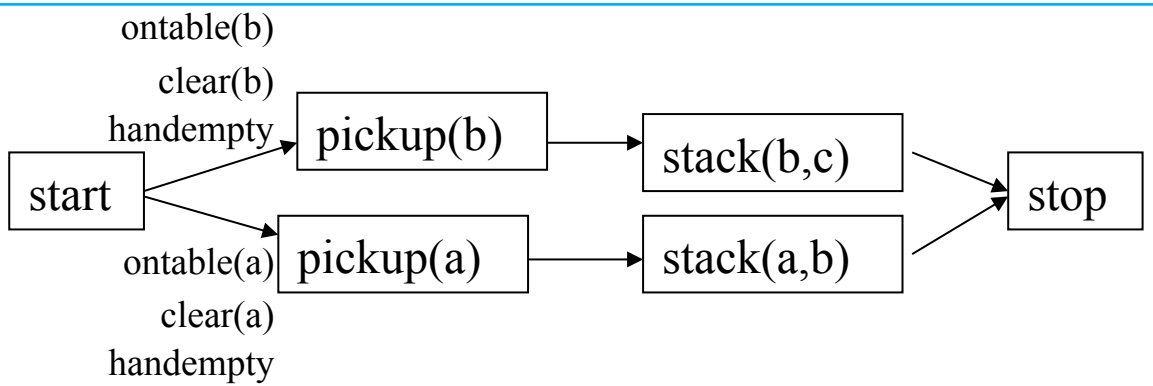
pickup(a)
PRECOND: *ontable(a), clear(a), handempty*
POSTCOND: *not ontable(a), not clear(a), holding(a), not handempty,*

pickup(b)
PRECOND: *ontable(b), clear(b), handempty*
POSTCOND: *not ontable(b), not clear(b), holding(b), not handempty*

Lista degli ordinamenti : […, pickup(a)<stack(a,b), pickup(b)<stack(b,c)]
Lista cas. link:[…,<pickup(a),stack(a,b),holding(a)>,<pickup(b),stack(b,c),holding(b)>]
Agenda di goal: [ontable(a), clear(a), handempty ontable(b), clear(b), handempty]

Esempio: Anomalia di Sussmann

Piani Parziali correnti:



ontable(b), ontable(a) clear(b) sono soddisfatte dallo stato iniziale.

Lista degli ordinamenti : [vd. sopra]
Lista dei causal link : […,<start,pickup(a),ontable(a)> , <start,pickup(b),ontable(b)> ,
<start,pickup(b),clear(b)>]
Agenda di goal : *[handempty,clear(a),handempty,clear(b)]*

stack(a,b) minaccia *<start,pickup(b),clear(b)>* perché nei suoi effetti contiene *not clear(b)* e nessuno degli ordinamenti le impedisce di precedere *pickup(b)* e invalidare la sua precondizione *clear(b)*.

↓

Imponiamo (**promotion**) *pickup(b)* < *stack(a,b)*

Esempio: Anomalia di Sussmann

Anche una delle due precondizioni *handempty* può essere soddisfatta dallo stato iniziale.

Il nuovo causal link <*start,pickup(b),handempty*> è minacciato da *pickup(a)*:

↓

Imponiamo (promotion) *pickup(b)* < *pickup(a)*

La precondizione *handempty* di *pickup(a)* non può essere soddisfatta da start perche' *pickup(b)* che precede *pickup(a)* la disfarrebbe.

↓

Applichiamo la **scopertura** usando l'azione del piano *stack(b,c)* (la mossa *pickup(b)* copre la condizione *handempty*, mentre *stack(b,c)* la scopre):

pickup(b) < *stack(b,c)* < *pickup(a)*

Lista degli ordinamenti: […, pickup(b) < stack(a,b), pickup(b) < pickup(a), stack(b,c) < pickup(a)]

Lista caus. link: [… , <start,pickup(b),handempty> , <stack(b,c),pickup(a),handempty>].

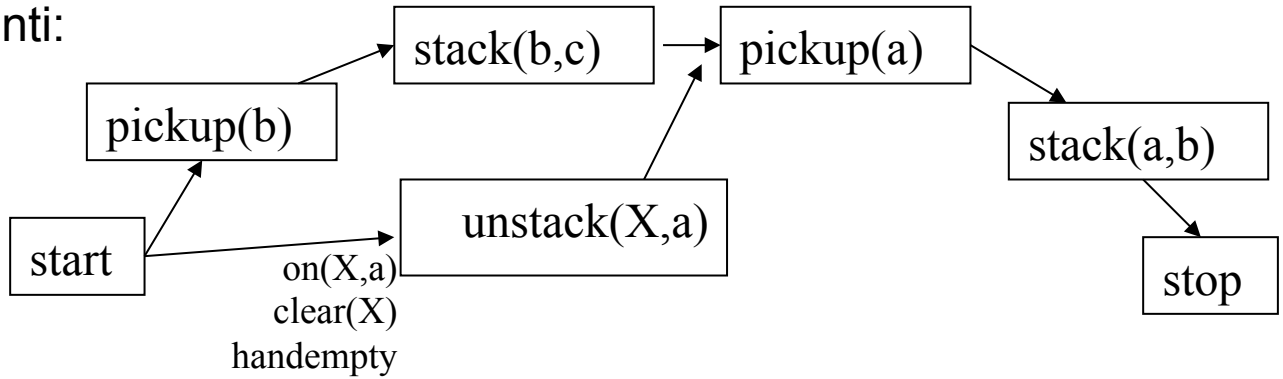
Agenda di goal: [clear(a)]

Esempio: Anomalia di Sussmann

Per raggiungere *clear(a)* possiamo aggiungere l'azione:

unstack(X,a)
PRECOND: *handempty, on(X,a), clear(X)*
POSTCOND: *not handempty, clear(a),holding(X) not on(X,a)*

Piani parziali correnti:



Lista ordinamenti: […, start<unstack(X,a), unstack(X,a)<stop, unstack(X,a)<pickup(a)]
Lista dei causal link: […,<unstack(X,a),pickup(a),clear(a)>
Agenda di goal: [on(X,a),clear(X),handempty]

Le tre precondizioni in agenda sono soddisfatte nello stato iniziale imponendo *X=c* nella mossa *unstack(X,a) (establishment)*

Esempio: Anomalia di Sussmann

unstack(c,a) rappresenta una minaccia per il causal link <*start,pickup(b),handempty*>

Aggiungiamo al piano una nuova mossa di **scopertura**:

putdown(c)
PRECOND: *holding(c)*
POSTCOND: *ontable(c), clear(c), handempty, not holding(c)*

ordinando così:
unstack(c,a) < putdown(c)< pickup(b)

Abbiamo quindi usato due tipi di scopertura, uno usando una mossa già nel piano, uno introducendo una nuova mossa.

Adesso tutte le precondizioni sono soddisfatte per cui si può linearizzare il piano (aggiungere i vincoli di ordinamento) ed eventualmente istanziare le variabili non ancora istanziate, per ottenere un piano concreto.

1) unstack(c,a) **2) putdown(c)** **3) pickup(b)**
4) stack(b,c) **5) pickup(a)** **6) stack(a,b)**

Esempio: Anomalia di Sussmann

unstack(c,a) rappresenta una minaccia per il causal link <*start,pickup(b),handempty*>

Aggiungiamo al piano una nuova mossa di **scopertura**:

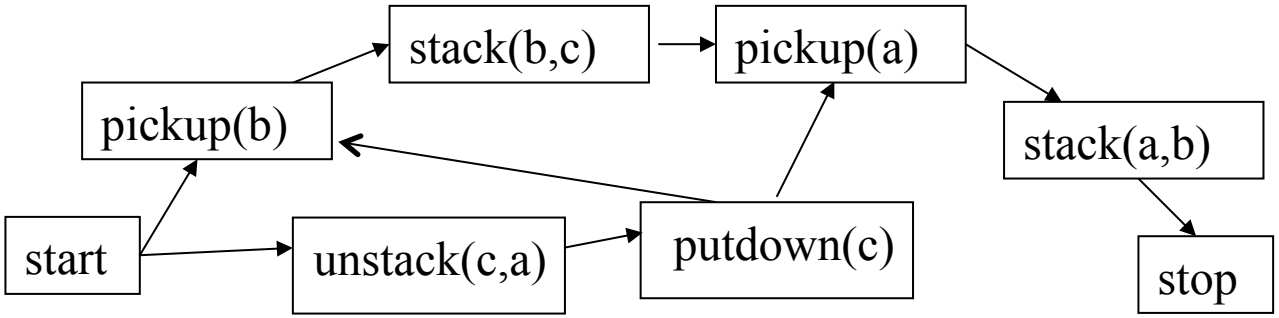
putdown(c)

PRECOND: *holding(c)*

POSTCOND: *ontable(c), clear(c), handempty, not holding(c)*

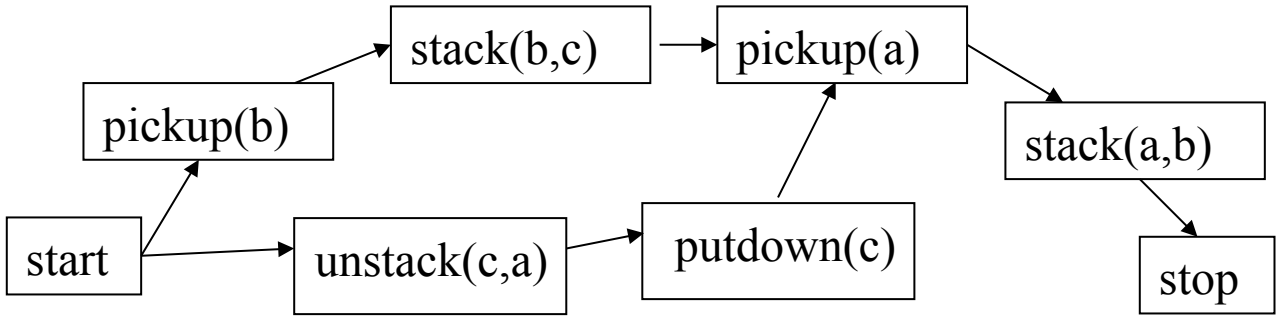
ordinando così:

unstack(c,a) < putdown(c)< pickup(b)



Abbiamo quindi usato due tipi di scopertura, uno usando una mossa già nel piano, uno introducendo una nuova mossa.

Esempio: Anomalia di Sussmann



Adesso tutte le precondizioni sono soddisfatte per cui si può linearizzare il piano (aggiungere i vincoli di ordinamento) ed eventualmente istanziare le variabili non ancora istanziate, per ottenere un piano concreto.

Ad esempio:

1. **unstack(c,a)**
2. **putdown(c)**
3. **pickup(b)**
4. **stack(b,c)**
5. **pickup(a)**
6. **stack(a,b)**

Esempio: Anomalia di Sussmann

È preferibile applicare la promozione prima della scopertura (con aggiunta di nuove azioni). Purtroppo tutti questi pianificatori possono generare piani molto inefficienti, anche se corretti.

La pianificazione è semi-decidibile: se esiste un piano che risolve un problema il pianificatore lo trova, ma se non esiste, il pianificatore può lavorare indefinitivamente.

All'aumento della complessità diventa difficile pensare ad un pianificatore corretto e completo, efficiente ed indipendente dal dominio: occorrono metodi ad hoc.

Planning in pratica

- Molte applicazioni in domini complessi:
  - Pianificatori per Internet (ricerca di informazioni, sintesi di comandi Unix)
  - Gestione di strumentazione spaziale
  - Robotica

  http://www.robocup.org/
  - Graphplan sviluppato dalla Carnegie Mellon University

  http://www.cs.cmu.edu/~avrim/graphplan.html
  - Piani di produzione industriale
  - Logistica

- Algoritmi visti presentano problemi di efficienza in caso di domini con molti operatori

- Tecniche per rendere più efficiente il processo di pianificazione

PLANNING GERARCHICO

Pianificazione Gerarchica

I pianificatori gerarchici sono algoritmi di ricerca che gestiscono la creazione di piani complessi a diversi livelli di astrazione, considerando i dettagli più semplici solo dopo aver trovato una soluzione per i più difficili.

- Linguaggio che permette di definire operatori a diversi livelli di astrazione:
  - *Valori di criticità assegnati alle precondizioni*
  - *Operatori ‘‘atomici’’/Macro operatori*

Tutti gli operatori sono definiti ancora con PRECONDIZIONI ed EFFETTI.

- Algoritmo di pianificazione gerarchica più diffusi:
  - *STRIPS-Like*
  - *Partial-Order*

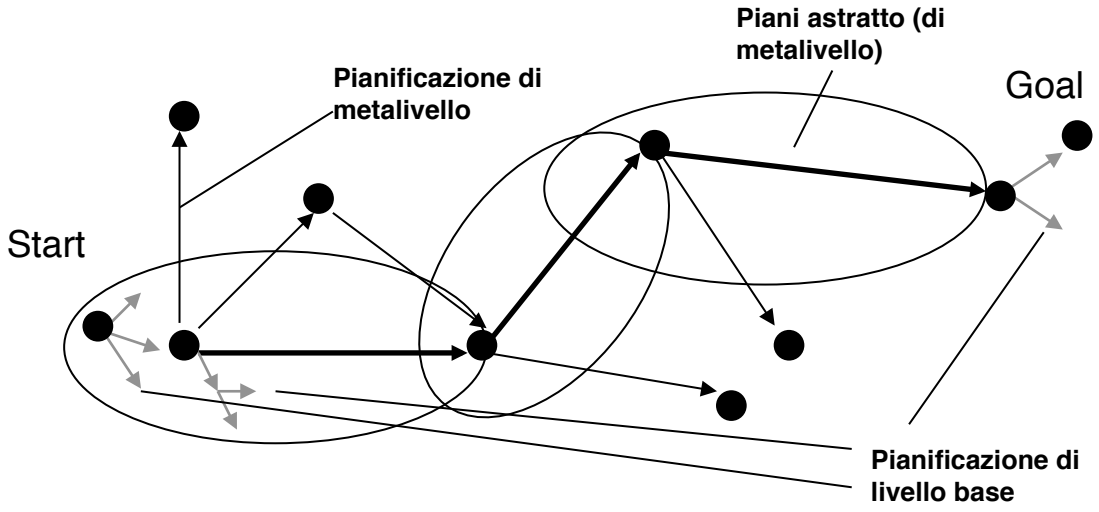
Dato un goal il pianificatore gerarchico effettua una ricerca di “meta-livello” per generare un piano anch’esso detto “di meta livello” che porta da uno stato molto vicino allo stato iniziale ad uno stato molto vicino al goal.

Pianificazione Gerarchica

Questo piano viene poi rifinito con una ricerca di più basso livello che tiene conto dei dettagli fin qui tralasciati.

Quindi un algoritmo gerarchico deve essere in grado di:

- pianificare a livello alto (*meta-livello*)
- espandere piani astratti in piani concreti
  - pianificando parti astratte in termini di azioni più specifiche (pianificazione di *livello base*)
  - espandendo piani già precostruiti



ABSTRIPS

Pianificatore Gerarchico che usa la definizione delle azioni di Strips dove in più a ciascuna precondizione è associato un **valore di criticità** che consiste nella sua difficoltà di raggiungimento.

La pianificazione procede a livelli in una gerarchia di spazi di astrazione in ciascuno dei quali vengono ignorate le precondizioni di livelli di difficoltà inferiore.

ABSTRIPS esplora interamente lo spazio di un determinato livello di astrazione prima di passare ad un livello più dettagliato: **ricerca in lunghezza**.

Ad ogni livello di astrazione viene generato un piano completo

Esempi applicativi:

- costruzione di un palazzo,
- organizzazione di un viaggio,
- progetto di un programma top-down.

ABSTRIPS: Metodologia di soluzione

1. Viene fissato un valore di soglia.
2. Si considerano vere tutte le precondizioni il cui valore di criticità è minore del valore di soglia.
3. Si procede come STRIPS(*) per la ricerca di un piano che soddisfi tutte le precondizioni con valore di criticità superiore o uguale al valore di soglia.
4. Si usa poi lo schema di piano completo così ottenuto come guida e si abbassa il valore di soglia di 1.
5. Si estende il piano con gli operatori che soddisfano le precondizioni. di livello di criticità maggiore o uguale al nuovo valore di soglia.
6. Si abbassa il valore di soglia fino a che si sono considerate tutte le precondizioni delle regole originarie.

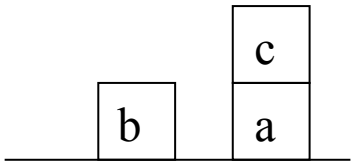
È importante assegnare bene i valori di criticità delle precondizioni!!!

(*) Ad ogni livello possiamo utilizzare un planner diverso, non necessariamente STRIPS.

Esempio (Algoritmo Strips-Like)

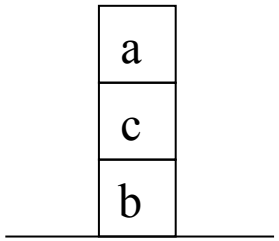
- Stato iniziale:

clear(b)
clear(c)
on(c,a)
handempty
ontable(a)
ontable(b)



- Goal:

on(c,b) ∧ *on(a,c)*



Specifichiamo una gerarchia di goal e precondizioni assegnando dei valori di criticità (che riflettono il grado di difficoltà nel soddisfarli):

on **(3)**
ontable, clear, holding **(2)**
handempty **(1)**

Esempio (Algoritmo Strips-Like)

Utilizziamo le regole STRIPS:

pickup(X)
PRECOND: *ontable(X), clear(X), handempty*
DELETE: *ontable(X), clear(X), handempty*
ADD: *holding(X)*

putdown(X)
PRECOND: *holding(X)*
DELETE: *holding(X)*
ADD: *ontable(X), clear(X), handempty*

stack(X,Y)
PRECOND: *holding(X), clear(Y)*
DELETE: *holding(X), clear(Y)*
ADD: *handempty, on(X,Y), clear(X)*

unstack(X,Y)
PRECOND: *handempty, on(X,Y), clear(X)*
DELETE: *handempty, on(X,Y), clear(X)*
ADD: *holding(X), clear(Y)*

Esempio (Algoritmo Strips-Like)

- Al **primo livello** di astrazione si considerano solo le precondizioni con valore di criticità 3 e si ottiene il seguente goal stack:

on(c,b)
on(a,c)
on(c.b) ∧ on(a,c)

- L'azione *stack(c,b)* viene aggiunta al piano per soddisfare *on(c,b)*.
- Visto che le sue precondizioni hanno tutte valore di criticità minore di 3 vengono considerate soddisfatte.

Per cui viene generato una nuova descrizione di stato simulando l'azione *stack(c,b)*.

state description	**goal stack**
clear(b) clear(c) ontable(a) ontable(b) **on(c,b)** handempty on(c,a)	on(a,c) on(c,b) ∧ on(a,c)

Nota: nella descrizione dello stato vengono aggiunti o cancellati solo i letterali delle ADD e DELETE list con valore di criticità maggiore o uguale a 3. Ci possono quindi essere contraddizioni nella descrizione dello stato ma questo non causa problemi.

Esempio (Algoritmo Strips-Like)

- Si aggiunge al piano l'azione *stack(a,c)* con cui si effettua lo stesso procedimento visto per *stack(c,b)*. Si ottiene il piano completo di livello 1:

1. stack(a,c)
2. stack(c,b)

- Passiamo la soluzione al **livello 2**: si riconsidera la descrizione di stato iniziale e il goal stack di partenza diventa:

holding(c)
clear(b)
holding(c) ∧ clear(b)
stack(c,b)
holding(a)
clear(c)
holding(a) ∧ clear(c)
stack(a,c)
on(c,b) ∧ on(a,c)

Esempio (Algoritmo Strips-Like)

- la condizione *holding(c)* è soddisfacibile con un'azione *unstack(c,X)* per cui lo stack diventa

clear(c)
on(c,X)
clear(c) and on(c,X)
unstack(c,X)
clear(b)
holding(c) ∧ clear(b)
stack(c,b)
holding(a)
clear(c)
holding(a) ∧ clear(c)
stack(a,c)
on(c,b) ∧ on(a,c)

- le precondizioni di *unstack(c,X)* da considerare a livello 2 (*clear(c)* e *on(c,X)*) sono tutte soddisfatte nello stato iniziale con la sostituzione *X/a*.

Esempio (Algoritmo Strips-Like)

Simulando l'azione *unstack(c,a)* si ottiene:

state description	**goal stack**
clear(b) clear(a) ontable(a) ontable(b) handempty holding(c)	clear(b) holding(c) ∧ clear(b) **stack(c,b)** holding(a) clear(c) holding(a) ∧ clear(c) **stack(a,c)** on(c,b) ∧ on(a,c)

E così via fino ad ottenere il piano completo:

1. **unstack(c,a)**
2. **stack(c,b)**
3. **pickup(a)**
4. **stack(a,c)**

Esempio (Algoritmo Strips-Like)

- A **livello 3** abbiamo il seguente goal stack:

  handempty ∧ clear(c) ∧ on(c,a)
  unstack(c,a)
  holding(c) ∧ clear(b)
  stack(c,b)
  handempty ∧ clear(a) ∧ ontable(a)
  pickup(a)
  holding(a) ∧ clear(c)
  stack(a,c)
  on(c,b) ∧ on(a,c)

Tutte le precondizioni sono già state soddisfatte ai livelli di astrazione precedenti a parte *handempty*.

La ricerca di una soluzione a livello 3 in questo caso risulta semplicemente una verifica della soluzione trovata a livello 2 che risulta essere corretta anche a livello 3.

Nota: A livello 1 sarebbe stato altrettanto valido il piano

1. **stack(a,c)**
2. **stack(c,b)**

che si sarebbe ottenuto considerando i due goal in ordine inverso. Ma ai livelli più bassi questo piano sarebbe fallito causando backtracking.

Esempio (Algoritmo Strips-Like)

- A livello 2 il goal stack iniziale

holding(a) ∧ clear(c)
stack(a,c)
holding(c) ∧ clear(b)
stack(c,b)
on(c,b) ∧ on(a,c)

avrebbe portato ad una soluzione tipo:

unstack(c,a)
pickup(a)
putdown(c)
stack(a,c)
unstack(a,c)
pickup(c)
stack(c,b)

dove pero' la descrizione di stato finale derivante dalla simulazione delle azioni del piano non soddisfa la congiunzione di goal *on(c,b) ∧ on(a,c)*.
Occorre cercare una soluzione diversa a livello 1.

Pianificazione Gerarchica con Operatori Macro

Due tipologie di operatori:

- operatori *atomici*
- operatori *macro*.

- Gli operatori **atomici** rappresentano azioni elementari tipicamente definite come regole STRIPS.

- Gli operatori **macro** a loro volta rappresentano una sequenza di azioni elementari: sono decomponibili in operatori "atomici".

La loro decomposizione può essere *precompilata* o da *pianificare*.

Pianificazione Gerarchica con Operatori Macro

1) Nel primo caso la descrizione dei macro operatori contiene anche la DECOMPOSITION che rappresenta la sequenza di operatori base in cui viene espanso l'operatore macro in questione durante la sua esecuzione.

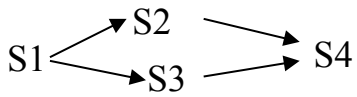
ACTION: ***Cook(X)***

PRECOND: *have(X), have(pot), have(salt), in(water,pot)*

EFFECT: *cooked*(X)

DECOMPOSITION: *S1:boil(water),S2:put(salt,water),S3:put(X,pot), S4:boilinWater(X)*

ORDER:S1 < S2, S1 < S3, S2 < S4, S3 < S4



2) Nel caso in cui manchi la DECOMPOSITION nella definizione delle azioni occorre che il pianificatore effettui una ricerca di basso livello per sintetizzare un piano di azioni elementari che "implementino" l'azione macro.

Esempio della spesa:

Le azioni elementari di spostamento di un robot portano a dover pianificare più in dettaglio le azioni (macro) "go" e "buy"

Pianificazione Gerarchica con Operatori Macro

L'algoritmo di ricerca può essere sia lineare sia non lineare.

Un algoritmo gerarchico non lineare tipico è lo stesso algoritmo POP già visto, dove ad ogni passo si può scegliere tra:

– soddisfare un sottogoal con un operatore (compresi operatori macro)

– espandere un macro step del piano (il metodo di decomposizione può essere precompilato o da pianificare).

```
function HD_POP(initialGoal, methods, operators) returns plan
  plan := INITIAL_PLAN(start, stop, initialGoal)
  loop
    if  SOLUTION(plan) then return plan;
    choose between
      • SN, C := SELECT_SUBGOAL(plan);
        CHOOSE_OPERATOR(plan, operators, SN, C);
      • SnonPrim := SELECT_MACRO_STEP(plan)
        CHOOSE_DECOMPOSITION(SnonPrim, methods, plan)
    RESOLVE_THREATS(plan)
  end
```

Esempio (Algoritmo Partial Order)

Stato iniziale

have(pot), have(pan), have(oil), have(onion),
have(pasta), have(tomato), have(salt), have(water)

Goal

made(pasta,tomato)

AZIONI ATOMICHE

ACTION ***makePasta(X)***
PRECOND: *have(pasta), cooked(pasta), sauce(X)*
EFFECT: *made(pasta,X)*

ACTION ***boil(X)***
PRECOND: *have(X),have(pot),in(X,pot),plain(X)*
EFFECT: *boiled(X)*

ACTION ***boilinWater(X)***
PRECOND: *have(X),have(pot),in(water,pot), boiled(water),in(salt,water),in(X,water)*
EFFECT: *cooked(X)*

ACTION ***cookSauce(X)***
PRECOND: *have(X),have(pan),in(onion,pan), fried(onion),in(X,oil)*
EFFECT: *sauce(X)*

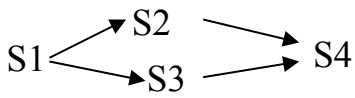
Esempio (Algoritmo Partial Order)

ACTION ***fry(X)***
PRECOND: *have(X),have(pan),have(oil), in(oil,pan),in(X,pan),plain(oil)*
EFFECT: *fried(X)*

ACTION: ***put(X,Y)***
PRECOND: *have(X),have(Y)*
EFFECT: *in(X,Y),¬ plain(Y)*

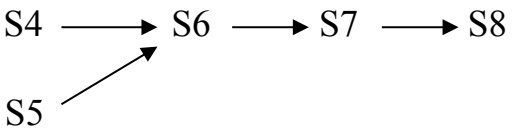
AZIONI MACRO

ACTION: ***Cook(X)***
PRECOND: *have(X), have(pot), have(salt), in(water,pot)*
EFFECT: *cooked*(X)



DECOMPOSITION: *S1:boil(water),S2:put(salt,water),S3:put(X,pot), S4:boilinWater(X)*
ORDER:S1 < S2, S1 < S3, S2 < S4, S3 < S4

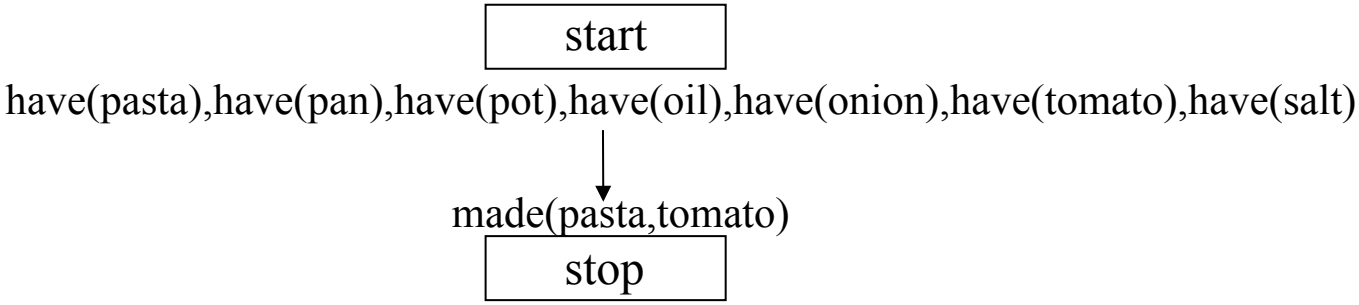
ACTION: ***MakeSauce(X)***
PRECOND: *have(X), have(oil), have(onion), have(pan)*
EFFECT: *sauce(X)*



DECOMP.: *S4: put(oil,pan), S5:put(onion,pan), S6: fry(onion), S7: put(X,oil), S8: cookSauce(X)*
ORDER:S4 < S6,S5 < S6,S6 < S7,S7 < S8

Esempio (Algoritmo Partial Order)

- Piano iniziale (detto *nullo*):
-



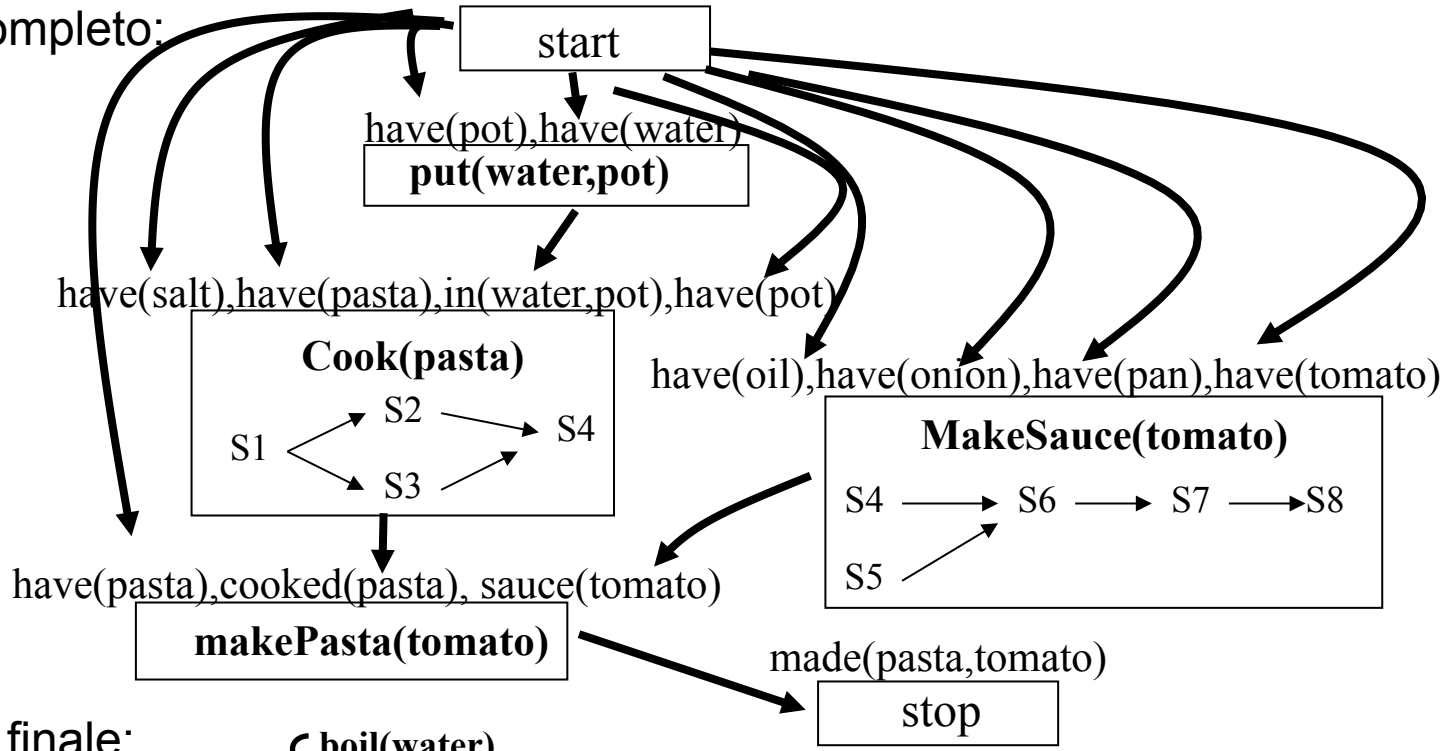
- Ad ogni passo si può scegliere fra

- selezionare una precondizione da soddisfare e cercare un'azione (macro o atomica) che la soddisfi (vedi pianificazione partial order classica)

- decomporre un macro operatore del piano (nel nostro esempio la decomposizione è precompilata ma potrebbe essere a sua volta un piano da costruire)

Esempio (Algoritmo Partial Order)

Schema di piano completo:



Un possibile piano finale:

put(water,pot)
Cook(pasta)
MakeSauce(tomato)
makePasta(tomato)

- **boil(water)**
- **put(salt, pot)**
- **put(pasta, pot)**
- **boilinWater(pasta)**
- **put(oil,pan)**
- **put(onion,pan)**
- **fry(onion)**
- **put(tomato,pan)**
- **cookSauce(tomato)**

Condizioni su Planning Gerarchico

Affinché il planning gerarchico funzioni, devono essere garantite alcune proprietà:

Vincoli sulla decomposizione

- Se l'azione macro A ha come effetto X e viene espansa con il piano P
  - X deve essere effetto di almeno una delle azioni in cui A viene decomposta e deve essere protetto fino alla fine del piano P
  - ogni precondizione delle azioni in P deve essere garantita dai passi di P precedenti oppure deve essere una precondizione di A
  - le azioni di P non devono violare vincoli causali quando P viene sostituito ad A nel piano che si sta costruendo

Solo a queste condizioni si può sostituire la azione macro A con il piano P

Condizioni su Planning Gerarchico

Sostituzione di A con P

- Si devono mettere a posto sia le relazioni d'ordine sia i link causali

  – **Relazioni d'ordine s**

    - per ogni B tale per cui B < A si impone B < first(P) (prima azione di P)
    - per ogni B tale per cui A < B si impone last(P) < B (ultima azione di P)

  – **Link causali**

    - se <S,A,C> era un causal link nel piano, allora si deve sostituire con una serie di link <S,Si,C> dove Si sono le azioni di P che hanno C come precondizione e nessun altro passo di A prima di Si ha C come precondizione
    - se <A,S,C> era un causal link nel piano, allora si deve sostituire con una serie di link <Si,S,C> dove Si sono le azioni di P che hanno C come effetto e nessun altro passo di P dopo Si ha C come effetto

Esecuzione

I pianificatori visti finora permettono di costruire piani che vengono poi eseguiti da un agente "**esecutore**".

Possibili problemi in esecuzione:

- Esecuzione di un'azione in condizioni diverse da quelle previste dalle sue precondizioni
  - conoscenza incompleta o non corretta
  - condizioni inaspettate
  - trasformazioni del mondo per cause esterne al piano
- Effetti delle azioni diversi da quelli previsti
  - errori dell'esecutore
  - effetti non deteministici, impredicibili

Occorre che l'esecutore sia in grado di *percepire* i cambiamenti e *agire* di conseguenza

Esecuzione

Alcuni pianificatori fanno l'ipotesi di *mondo aperto* (**Open World Assumption**) ossia considerano l'informazione non presente nella rappresentazione dello stato come *non nota* e *non falsa* diversamente dai pianificatori che lavorano nell'ipotesi di mondo chiuso (*Closed World Assumption*)

Alcune informazioni non note possono essere cercate tramite azioni di "raccolta di informazioni" (**azioni di sensing**) aggiunte al piano.

Le ***azioni di sensing*** sono modellate come le azioni causali.

Le precondizioni rappresentano le condizioni che devono essere vere affinché una certa osservazione possa essere effettuata, le postcondizioni rappresentano il risultato dell'osservazione.

Due possibili approcci:

- Planning Condizionale
- Integrazione fra Pianificazione ed Esecuzione

Planning Condizionale

Un **pianificatore condizionale** è un algoritmo di ricerca che genera diversi piani alternativi per ciascuna fonte di incertezza del piano.

Un **piano condizionale** è quindi costituito da:

- Azioni causali,
- Azioni di sensing per le verifiche
- Diversi piani parziali alternativi di cui uno solo verrà eseguito a seconda dei risultati delle verifiche.

Esempio: sostituzione ruote in un'auto

Azioni:

remove(X,Y)
PRECOND: *on(X,Y),* ***¬ intact(X)***
EFFECT: *¬ on(X,Y), off(X), clearHub(Y)*

putOn(X,Y)
PRECOND: *off(X), clearHub(Y)*
EFFECT: *on(X,Y), ¬ off(X), ¬ clearHub(Y)*

inflate(X)
PRECOND: ***intact(X)****, flat(X)*
EFFECT: *inflated(X), ¬ flat(X)*

Azione di sensing*:*

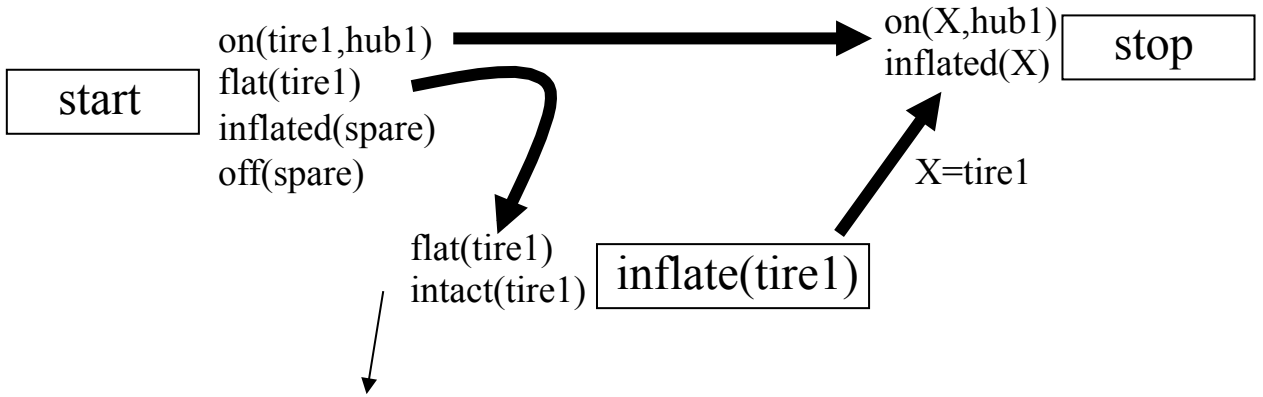
checkTire(X)
PRECOND: tire(X)
EFFECT: knowsWhether(intact(X))

Stato iniziale*: on(tire1,hub1), flat(tire1), inflated(spare), off(spare)*

Goal*: on(X,hub1), inflated(X)*

Esempio: sostituzione ruote in un'auto

Un pianificatore tradizionale (senza azioni di sensing) produrrebbe il seguente piano:

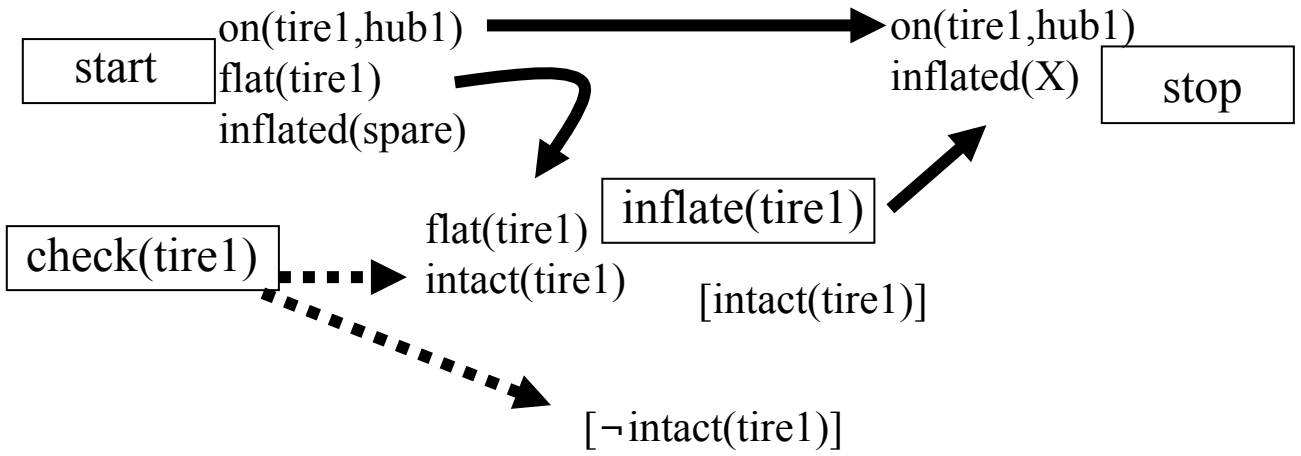


La condizione *intact(tire1)* è un'informazione che non compare nella descrizione dello stato iniziale e non può essere ottenuta con nessuna azione: **fallimento** del piano.

In backtracking fallisce anche il piano che si ottiene con *X=spare*: *remove(tire1,hub1)* - *putOn(spare,hub1)* visto che la precondizione ¬ *intact(tire1)* non può essere verificata in nessun modo.

Esempio: sostituzione ruote in un'auto

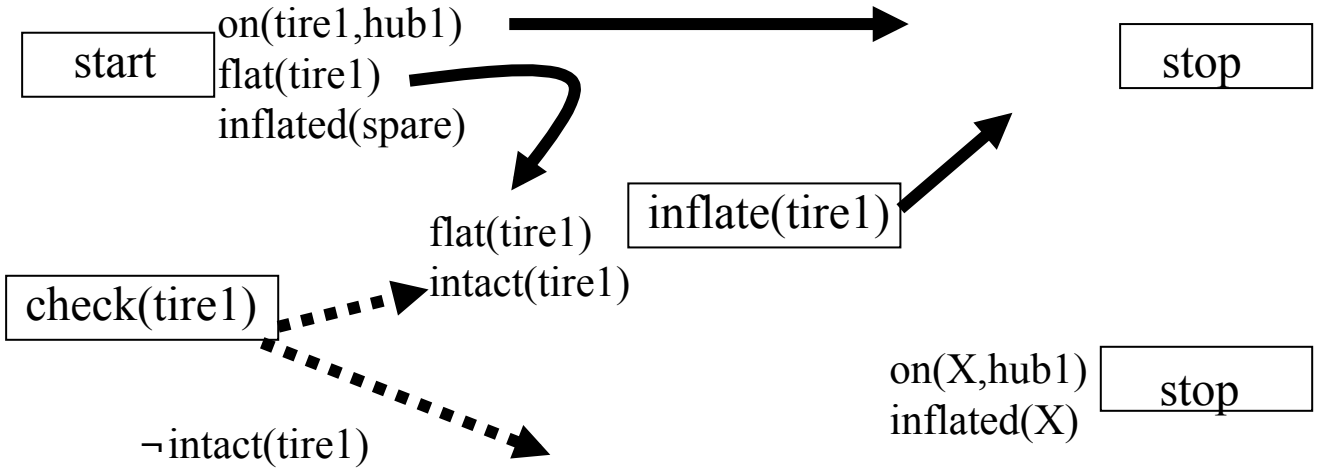
Usando le azioni di sensing si puo costruire un **piano condizionale**:



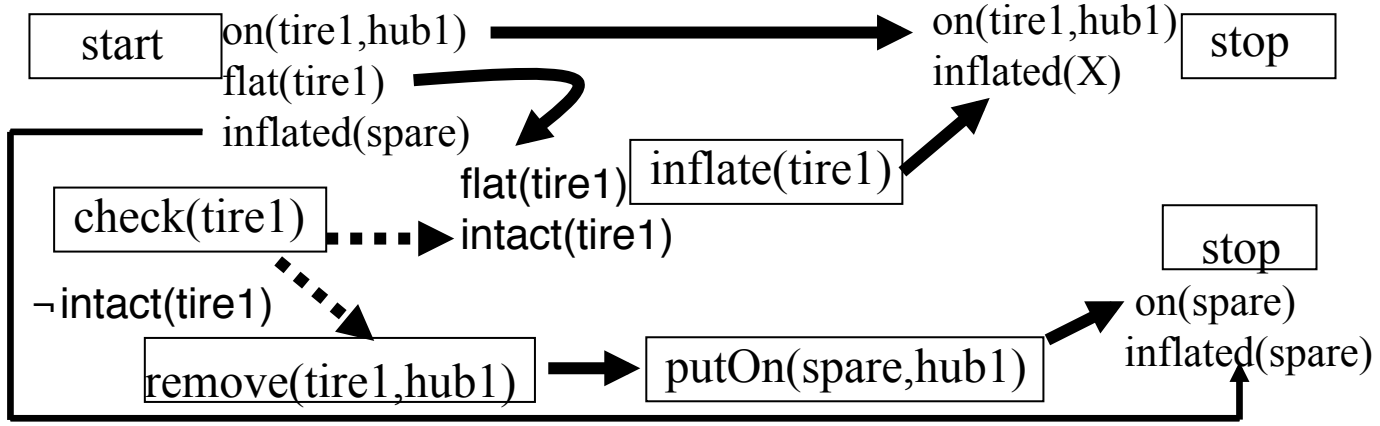
- *inflate(tire1)* rappresenta il piano corretto solo in una delle possibili risposte alla verifica: in un **contesto** (quello in cui *intact(tire1)* è verificata nello stato iniziale)

- Occorre generare una copia del goal per ogni altro possibile contesto e generare un piano per ognuno di essi. In questo modo si avrà un piano condizionale che può funzionare in tutte le possibili risposte alla verifica.

Esempio: sostituzione ruote in un'auto



Si costruisce quindi il piano **condizionale**



Problemi dei pianificatori condizionali

Esplosione computazionale dell'albero di ricerca nel caso di problemi con un numero di contesti alternativi elevato.

Un piano completo che tenga conto di ogni possibile contingenza potrebbe richiedere molta memoria.

Non sempre è possibile conoscere a priori tutti i contesti alternativi

Spesso si combina l'approccio condizionale con l'**approccio probabilistico** dove si assegnano dei valori di probabilità alle varie alternative e si pianifica solo per quelle più probabili.

Planning Reattivo (Brooks - 1986)

Abbiamo descritto fino qui un processo di pianificazione deliberativo nel quale prima di eseguire una qualunque azione viene costruito l'intero piano.

I pianificatori **reattivi** sono algoritmi di pianificazione on-line, capaci di interagire con il sistema in modo da affrontare il problema della dinamicità e del non determinismo dell'ambiente:

- osservano il mondo in fase di pianificazione per l'acquisizione di informazione non nota
- monitorano l'esecuzione delle azioni e ne verificano gli effetti
- spesso alternano il processo di pianificazione a quello di esecuzione reagendo ai cambiamenti di stato

Discendono dai *sistemi reattivi "puri"* che evitano del tutto la pianificazione ed utilizzano semplicemente la situazione osservabile come uno stimolo per reagire.

Sistemi reattivi puri

Hanno accesso ad una base di conoscenza che descrive quali azioni devono essere eseguite ed in quali circostanze. Scelgono le azioni una alla volta, senza anticipare e selezionare un'intera sequenza di azioni prima di cominciare.

Esempio -Termostato utilizza le semplici regole:
1) Se la temperatura T della stanza è K gradi sopra la soglia T0, accendi il condizionatore;
2) Se la temperatura della stanza è K gradi sotto T0, spegni il condizionatore.

Vantaggi:

- Sono capaci di interagire con il sistema reale. Essi operano in modo robusto in domini per i quali è difficile fornire modelli completi ed accurati.
- Non usano modelli, ma solo l'immediata percezione del mondo e per questo sono anche estremamente veloci nella risposta.

Svantaggio:

Il loro comportamento in domini che richiedono di ragionare e deliberare in modo significativo è deludente (es. scacchi) in quanto non sono in grado di generare autonomamente piani.

Pianificatori ibridi

I moderni pianificatori reattivi detti **ibridi** integrano **approccio generativo** e **approccio reattivo** al fine di sfruttare le capacità computazionali del primo e la capacità di interagire con il sistema del secondo affrontando così il problema dell'esecuzione.

Un pianificatore **ibrido:**

- genera un piano per raggiungere il goal
- verifica le precondizioni dell'azione che sta per eseguire e gli effetti dell'azione appena eseguita
- smonta gli effetti di un'azione (importanza della reversibilità delle azioni) e ripianifica in caso di fallimenti
- corregge i piani se avvengono azioni esterne impreviste.