

# PLANNING

---

La pianificazione automatica (**planning**) rappresenta un'importante attività di problem solving che consiste nel sintetizzare una sequenza di azioni che eseguite da un agente a partire da uno stato "iniziale" del mondo provocano il raggiungimento di uno stato "desiderato".

# PLANNING

---

Dati:

- uno stato iniziale
- un insieme di azioni eseguibili
- un obiettivo da raggiungere (**goal**)

un problema di pianificazione consiste nel determinare un **piano**, ossia un insieme (parzialmente o totalmente) ordinato di azioni necessarie per raggiungere il goal.

Pianificare è:

- una applicazione di per se'
- un'attività comune a molte applicazioni quali
  - diagnosi: pianificazione di test o azioni per riparare (riconfigurare) un sistema
  - scheduling
  - robotica

# Esempio

---

Pianificare è una attività che tutti gli esseri umani svolgono nel loro quotidiano.

Supponiamo di avere come obiettivo (**goal**) di seguire una lezione di Intelligenza Artificiale e di essere attualmente a casa e di possedere una macchina (**stato iniziale**).

Al fine di raggiungere lo scopo prefisso dobbiamo fare una serie di **azioni** in una certa sequenza:

1. prendere materiale necessario per gli appunti,
2. prendere le chiavi della macchina,
3. uscire di casa,
4. prendere la macchina,
5. raggiungere la facoltà,
6. entrare in aula e così via.

Pianificare ci permette di fare le azioni giuste nella sequenza giusta: non possiamo pensare di invertire l'ordine delle azioni 2 e 3.

# Esempio

---

Per alcune di queste azioni non è necessario pianificare l'ordine (il piano può contenere un ordinamento parziale). Invertendo l'ordine delle azioni 1 e 2 si ottiene un piano corretto analogo al precedente.

Ci possono essere piani alternativi per raggiungere lo stesso obiettivo (posso pensare di andare in facoltà a piedi o in autobus).

Es di piano alternativo

1. prendere materiale necessario per gli appunti,
2. prendere biglietto autobus,
3. uscire di casa,
4. raggiungere la fermata,
5. salire sull'autobus,
6. raggiungere la facoltà,
7. entrare in aula e così via.

# La Pianificazione Automatica: alcuni concetti base

---

Un **pianificatore automatico** è un *agente intelligente* che opera in un certo dominio e che date:

1. una rappresentazione dello stato iniziale
2. una rappresentazione del goal
3. una descrizione formale delle azioni eseguibili

sintetizza dinamicamente il piano di azioni necessario per raggiungere il goal a partire dallo stato iniziale.

# Rappresentazione dello stato

---

Occorre fornire al pianificatore un modello del sistema su cui opera.

In genere lo stato è rappresentato in forma dichiarativa con una congiunzione di formule atomiche che esprime la situazione di partenza.

*Es:  $on(book, table) \wedge name(book, xyz) \wedge atHome(table)$*

Lo stato di un sistema spesso può essere osservato solo in modo parziale per una serie di motivi:

- perché alcuni aspetti non sono osservabili;
- perché il dominio è troppo vasto per essere rappresentato nella sua interezza (limitate capacità computazionali per la rappresentazione);
- perché le osservazioni sono soggette a rumore e quindi si hanno delle osservazioni parziali o imperfette;
- perché il dominio è troppo dinamico per consentire un aggiornamento continuo della rappresentazione.

# Rappresentazione del goal

---

Per rappresentare il goal si utilizza lo stesso linguaggio formale con cui si esprime lo stato iniziale. In questo caso la congiunzione rappresenta una descrizione parziale dello stato finale che si vuole raggiungere: descrive solo le condizioni che devono essere verificate affinché il goal sia soddisfatto.

# Rappresentazione delle azioni

---

È necessario fornire al pianificatore una descrizione formale delle azioni eseguibili detta ***Teoria del Dominio***.

Ciascuna azione è identificata da un nome e modellata in forma dichiarativa per mezzo di ***precondizioni*** e ***postcondizioni***.

Le precondizioni rappresentano le condizioni che devono essere verificate affinché l'azione possa essere eseguita; le postcondizioni rappresentano gli effetti dell'azione stessa sul mondo.

Spesso la Teoria del Dominio è costituita da operatori con variabili che definiscono **classi di azioni**. A diverse istanziazioni delle variabili corrispondono diverse azioni. 8

# Esempio: mondo dei blocchi

---

**Problema:** spostare blocchi su un tavolo con un braccio

**Azioni:**

**STACK(X,Y)**

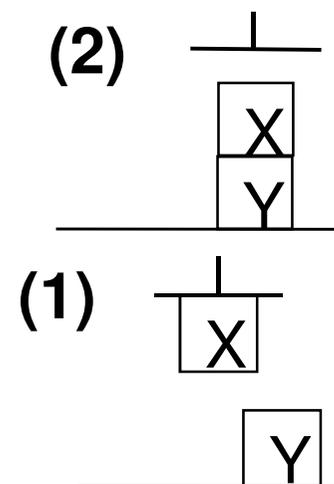
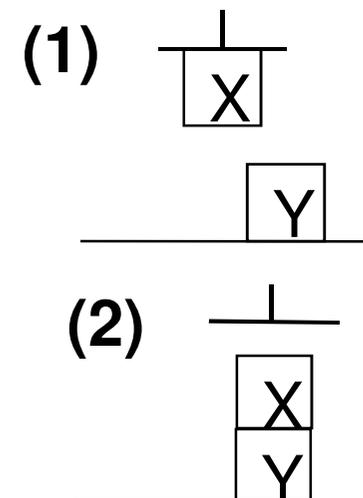
SE: holding(X) and clear(Y)

ALLORA: handempty and clear(X) and on(X,Y);

**UNSTACK(X,Y)**

SE: handempty and clear(X) and on(X,Y)

ALLORA: holding(X) and clear(Y);

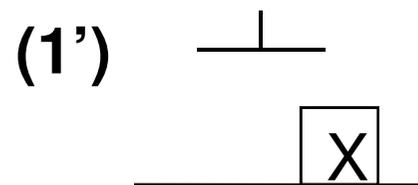


# Esempio: mondo dei blocchi

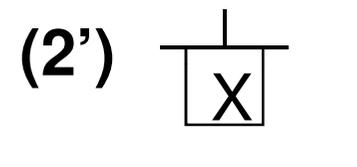
---

## PICKUP(X)

SE:  $\text{ontable}(X)$  and  $\text{clear}(X)$  and  $\text{handempty}$

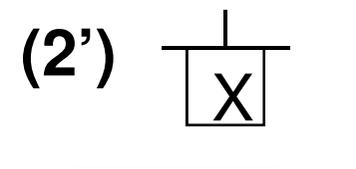


ALLORA:  $\text{holding}(X)$ ;

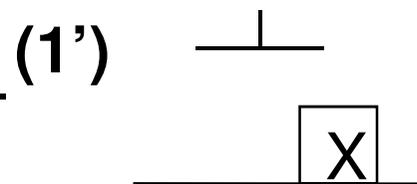


## PUTDOWN(X)

SE:  $\text{holding}(X)$



ALLORA:  $\text{ontable}(X)$  and  $\text{clear}(X)$  and  $\text{handempty}$ .



# Pianificazione

---

Processo per il calcolo dei diversi passi di una procedura di soluzione di un problema di pianificazione:

- *non decomponibile*

ci può essere interazione fra i sottoproblemi

- *reversibile*

le scelte fatte durante la **generazione** del piano sono revocabili (backtracking).

Un pianificatore è **completo** quando riesce sempre a trovare la soluzione se esiste.

Un pianificatore è **corretto** quando la soluzione trovata porta dallo stato iniziale al goal finale in modo consistente con eventuali vincoli.

# Esecuzione

---

Processo di applicazione della procedura di soluzione

➤ *irreversibile*

l'esecuzione delle azioni determina spesso un cambiamento di stato non reversibile,

➤ *non deterministica*

il piano può avere un effetto diverso quando applicato al mondo reale che è spesso imprevedibile. In questo caso è possibile rifare il piano solo parzialmente, oppure invalidarlo tutto a seconda del problema.

# Pianificazione classica

---

Tipo di pianificazione *off-line* che produce l'intero piano prima di eseguirlo lavorando su una rappresentazione istantanea (*snapshot*) dello stato corrente.

È basata su alcune assunzioni forti:

- tempo atomico di esecuzione delle azioni
- determinismo degli effetti
- stato iniziale completamente noto a priori
- esecuzione del piano unica causa di cambiamento del mondo

# Pianificazione reattiva

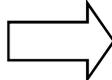
---

Metodo di pianificazione *on-line*

- considera l'ambiente non deterministico e dinamico
- è capace di osservare il mondo sia in fase di pianificazione sia in fase di esecuzione
- spesso alterna il processo di pianificazione a quello di esecuzione reagendo ai cambiamenti di stato

# Cosa vedremo:

---

- Tecniche di Pianificazione Classica
  - Planning Deduttivo
    - Situation Calculus
      - Formalizzazione di Green
      - Formalizzazione di Kowalsky
  - Planning mediante ricerca
    - Ricerca nello spazio degli stati  Planning Lineare
      - STRIPS
    - Ricerca nello spazio dei piani  Planning Non Lineare
      - Partial Order Planning (POP)
- Pianificazione Gerarchica Condizionale Reattiva - cenni
- Esercitazioni Graphplan e Prolog/Strips forward

# Planning Deduttivo

---

La tecnica di **pianificazione deduttiva** utilizza la logica per rappresentare stati, goal e azioni e genera il piano come dimostrazione di un teorema

# Situation Calculus (*fine anni '60*)

---

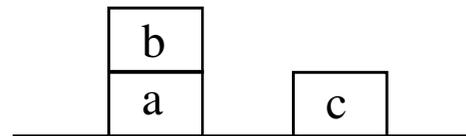
Formalizzazione del linguaggio (basato sulla logica dei predicati del primo ordine) in grado di rappresentare stati e azioni in funzione del tempo

➤ **Situation:** “fotografia” del mondo e delle proprietà (*fluent*) che valgono in un determinato istante/stato  $s$

– Esempio: mondo dei blocchi

$on(b,a,s)$

$ontable(c,s)$



➤ **Azioni:** definiscono quali fluent saranno veri come risultato di un'azione. Esempio

$on(X,Y,S)$  and  $clear(X,S) \rightarrow$

$(ontable(X,do(putOnTable(X),S)))$  and

$(clear(Y,do(putOnTable(X),S)))$

# Situation Calculus

---

- **Costruzione di un piano:** deduzione, dimostrazione di un goal
  - Esempio
    - $\text{:- ontable}(b,S)$ . Significa: esiste uno stato  $S$  in cui e' vero  $\text{ontable}(b)$
    - YES per  $S=\text{putOntable}(b,s0)$
- **Vantaggi:** elevata espressività, permette di descrivere problemi complessi
- **Problema:** frame problem

# Frame problem

---

Occorre specificare esplicitamente tutti i fluent che cambiano dopo una transizione di stato e anche quelli che NON cambiano (*assiomi di sfondo*: “Frame axioms”).

Al crescere della complessità del dominio il numero di tali assiomi cresce enormemente.

Il problema della rappresentazione della conoscenza diventa intrattabile

# Pianificazione come deduzione (GREEN)

---

Green usa il *situation calculus* per costruire un pianificatore basato sul metodo di risoluzione.



Si cerca la prova di una formula contenente una variabile di stato che alla fine della dimostrazione sarà istanziata al piano di azioni che permette di raggiungere l'obiettivo

# Esempio

---

- I seguenti assiomi descrivono tutte le relazioni vere nello stato iniziale  $s_0$

A.1  $on(a,d,s_0)$ .

A.2  $on(b,e,s_0)$ .

A.3  $on(c,f,s_0)$ .

A.4  $clear(a,s_0)$ .

A.5  $clear(b,s_0)$ .

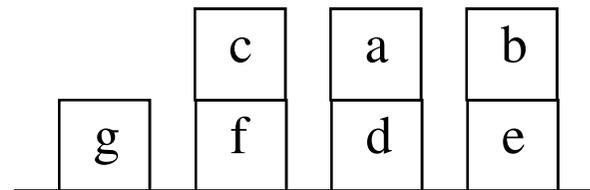
A.6  $clear(c,s_0)$ .

A.7  $clear(g,s_0)$ .

A.8  $diff(a,b)$

A.9  $diff(a,c)$

A.10  $diff(a,d)...$



# Esempio

---

- Le azioni si esprimono con assiomi nella forme a clausole

L'azione  $move(X, Y, Z)$

$clear(X, S)$  and  $clear(Z, S)$  and  $on(X, Y, S)$  and  $diff(X, Z)$

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

che sposta un blocco X da Y a Z, partendo dallo stato S e arriva allo stato  $do(move(X, Y, Z), S)$  si esprime con i seguenti assiomi (*effect axioms*)

A.11  $\sim clear(X, S)$  or  $\sim clear(Z, S)$  or  $\sim on(X, Y, S)$  or  $\sim diff(X, Z)$  or  $clear(Y, do(move(X, Y, Z), S))$ .

A.12  $\sim clear(X, S)$  or  $\sim clear(Z, S)$  or  $\sim on(X, Y, S)$  or  $\sim diff(X, Z)$  or  $on(X, Z, do(move(X, Y, Z), S))$ .

# Esempio

---

Dato un goal vediamo un esempio di come si riesce a trovare una soluzione usando il metodo di risoluzione:

*GOAL:- on(a,b,S1)*

*~on(a,b,S1)*

*(A.12) {X/a,Z/b,S1/do(move(a,Y,b),S)}*

*~clear(a,S) or ~clear(b,S) or ~on(a,Y,S) or ~diff(a,b)*

*(A.4)*

*{S/s0},*

*(A.5)*

*{S/s0},*

*(A.1)*

*{S/s0, Y/d}*

*(A.8)*

*true*

*~on(a,b,S1)* porta a una contraddizione quindi *on(a,b,S1)* risulta dimostrato con la sostituzione *S1/do(move(a,d,b),s0)*.

Supponiamo di voler risolvere un problema un po' più complesso

*Goal: on(a,b,S), on(b,g,S).*

*Soluzione: S/do(move(a,d,b),do(move(b,e,g),s0)).*

Per poterlo risolvere occorre una descrizione completa dello stato risultante dall'esecuzione di ciascuna azione.

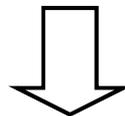
# Frame problem

---

Per descrivere un'azione oltre agli effect axioms occorre specificare tutti i fluent che NON sono invalidati dall'azione stessa (*frame axioms*). Nel nostro esempio occorrono i seguenti assiomi:

$$\text{on}(U, V, S) \text{ and } \text{diff}(U, X) \rightarrow \text{on}(U, V, \text{do}(\text{move}(X, Y, Z), S))$$
$$\text{clear}(U, S) \text{ and } \text{diff}(U, Z) \rightarrow \text{clear}(U, \text{do}(\text{move}(X, Y, Z), S))$$

Occorre esplicitare un frame axioms per ogni relazione NON modificata dall'azione.



Se il problema è complicato la complessità diventa inaccettabile

# Formulazione di Kowalsky

---

Rappresenta una formulazione più semplice:

- Viene utilizzato il predicato  $holds(rel, s/a)$  per indicare tutte le relazioni  $rel$  vere in un certo stato  $s$  o rese vere da una certa azione  $a$
- Il predicato  $poss(s)$  indica che uno stato  $s$  è possibile ovvero raggiungibile.
- Si utilizza il predicato  $pact(a,s)$  per affermare che è lecito compiere una determinata azione  $a$  in un particolare stato  $s$ , ovvero le precondizioni per svolgere l'azione sono soddisfatte in  $s$ .

Se uno stato  $S$  è possibile e se le precondizioni  $pact$  di un'azione  $U$  sono soddisfatte in quello stato, allora anche lo stato prodotto  $do(U,S)$  è possibile:

$$poss(S) \text{ and } pact(U,S) \rightarrow poss(do(U,S))$$

# Formulazione di Kowalsky

---

Quelli che nella formulazione di Green sono predicati qui diventano termini. In pratica, guadagnamo i vantaggi di una formulazione del II ordine mantenendoci in un sistema del I ordine.

In questo modo abbiamo bisogno di una singola frame assertion per ogni azione (vantaggio rispetto a Green)

Nell'esempio precedente l'unica frame assertion che deve essere esplicitata è:

$$\begin{aligned} & holds(V,S) \wedge diff(V,clear(Z)) \wedge diff(V,on(X,Y)) \quad \rightarrow \\ & holds(V,do(move(X,Y,Z),S)) \end{aligned}$$

che afferma che tutti i termini differenti da  $clear(Z)$  e  $on(X,Y)$  valgono ancora in tutti gli stati prodotti dall'esecuzione di  $move$ .

# Esempio

---

Goal

$\text{:- } \text{poss}(S), \text{holds}(\text{on}(a,b),S), \text{holds}(\text{on}(b,g),S).$

Usiamo PROLOG per risolverlo

Stato iniziale

$\text{poss}(s0).$

$\text{holds}(\text{on}(a,d),s0).$

$\text{holds}(\text{on}(b,e),s0).$

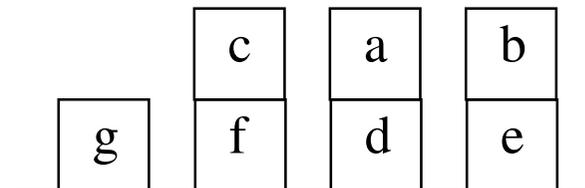
$\text{holds}(\text{on}(c,f),s0).$

$\text{holds}(\text{clear}(a),s0).$

$\text{holds}(\text{clear}(b),s0).$

$\text{holds}(\text{clear}(c),s0).$

$\text{holds}(\text{clear}(g),s0).$



# Esempio

---

Effetti dell'azione  $\text{move}(X,Y,Z)$ :

$\text{holds}(\text{clear}(Y), \text{do}(\text{move}(X,Y,Z), S)). \text{holds}(\text{on}(X,Z), \text{do}(\text{move}(X,Y,Z), S)).$

Clausola che esprime le precondizioni dell'azione  $\text{move}(X,Y,Z)$ :

$\text{pact}(\text{move}(X,Y,Z), S):-$

$\text{holds}(\text{clear}(X), S), \text{holds}(\text{clear}(Z), S), \text{holds}(\text{on}(X,Y), S), X \neq Z.$

Clausola per esprimere le condizioni di frame:

$\text{holds}(V, \text{do}(\text{move}(X,Y,Z), S)):- \text{holds}(V, S), V \neq \text{clear}(Z), V \neq \text{on}(X,Y).$

Clausola per esprimere la raggiungibilità di uno stato:

$\text{poss}(\text{do}(U, S)):- \text{poss}(S), \text{pact}(U, S).$

Soluzione

$:- \text{poss}(S), \text{holds}(\text{on}(b,g), S), \text{holds}(\text{on}(a,b), S).$

Yes per  $S = \text{do}(\text{move}(a, d, b), \text{do}(\text{move}(b, e, g), s0))$

# Pianificazione come ricerca

---

- *Quello che cambia è lo spazio di ricerca, definito da che cosa sono gli stati e gli operatori:*
  - *Pianificazione deduttiva come theorem proving: stati come insiemi di formule e operatori come regole di inferenza*
  - *Pianificazione nello spazio degli stati: stati come descrizioni di situazioni e operatori come modifiche dello stato*
  - *Pianificazione nello spazio dei piani: stati come piani parziali e operatori di raffinamento e completamento di piani*

# Planning classico non deduttivo

---

La pianificazione classica non deduttiva

- utilizza linguaggi specializzati per rappresentare stati, goal e azioni
- gestisce la generazione del piano come un problema di ricerca (*search*).

La ricerca può essere effettuata:

- nello **spazio degli stati** o situazioni  
*Nell'albero di ricerca ogni nodo rappresenta uno stato e ogni arco un'azione.*
- nello **spazio dei piani**  
*Nell'albero di ricerca ogni nodo rappresenta un piano parziale e ogni arco un'operazione di raffinamento del piano*

# Ricerca nello spazio degli stati: Planning Lineare

---

Un **pianificatore lineare** riformula il problema di pianificazione come problema di ricerca nello spazio degli stati e utilizza le strategie di ricerca classiche:

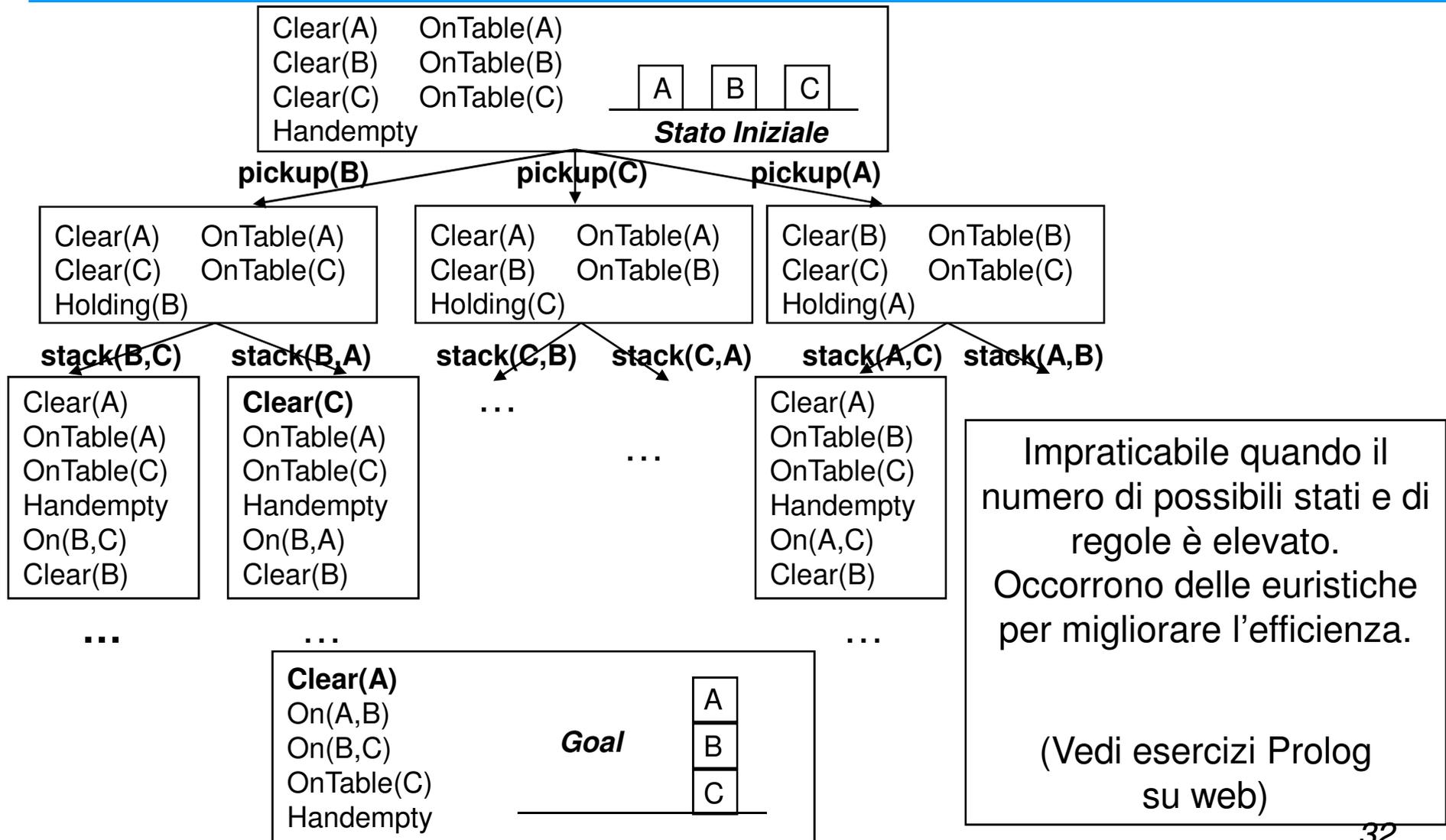
L'algoritmo di ricerca può essere:

- **Forward**: se la ricerca avviene in modo progressivo partendo dallo stato iniziale fino al raggiungimento di uno stato che soddisfa il goal.
- **Backward**: quando la ricerca è attuata in modo regressivo a partire dal goal fino a *ridurre* il goal in sottogoal soddisfatti dallo stato iniziale.



Problema: definire come si effettua la *riduzione di un goal in sottogoal* (GOAL REGRESSION) tenendo conto della forma delle regole

# Ricerca Forward: un esempio



# Ricerca backward: Goal Regression

---

La **regressione** è il meccanismo di base per ridurre un goal in sottogoal in un planner backward attraverso le regole

Dati un goal  $G$  e una regola  $R$

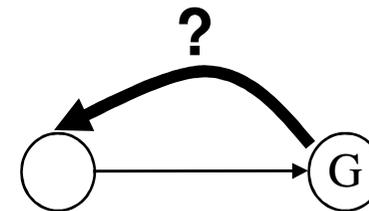
PRECOND:  $Plist$

DELETE:  $Dlist$

ADD:  $Alist$

la Regressione di  $G$  attraverso  $R$   **$Regr[G, R]$**  è:

- $Regr[G, R] = true$  se  $G \in Alist$
- $Regr[G, R] = false$  se  $G \in Dlist$
- $Regr[G, R] = G$  altrimenti



# Esempio

---

Data R1: *unstack*(X, Y)

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

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

ADD: holding(X), clear(Y)

si ha:

- $\text{Regr}[\text{holding}(b), R1] = \text{true}$
- $\text{Regr}[\text{handempty}, R1] = \text{false}$
- $\text{Regr}[\text{ontable}(c), R1] = \text{ontable}(c)$
- $\text{Regr}[\text{clear}(c), R1] = \begin{array}{l} \nearrow Y=c \\ \searrow \text{clear}(c) \end{array} = Y = c \vee \text{clear}(c)$   
 $X=c$  and  
**FALSE**

# Riduzione di goal in sottogoal

---

Dato il goal  $G:[G1, G2, \dots, Gn]$  e la regola  $R$

$R$ : PRECOND:  $Plist = P1, P2, \dots, Pm$

DELETE:  $Dlist = D1, D2, \dots, Dk$

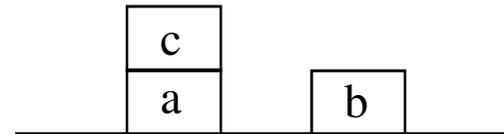
ADD:  $Alist = A1, A2, \dots, Al$

Il sottogoal che si ottiene da  $G$  attraverso  $R$  è dato dalla congiunzione:

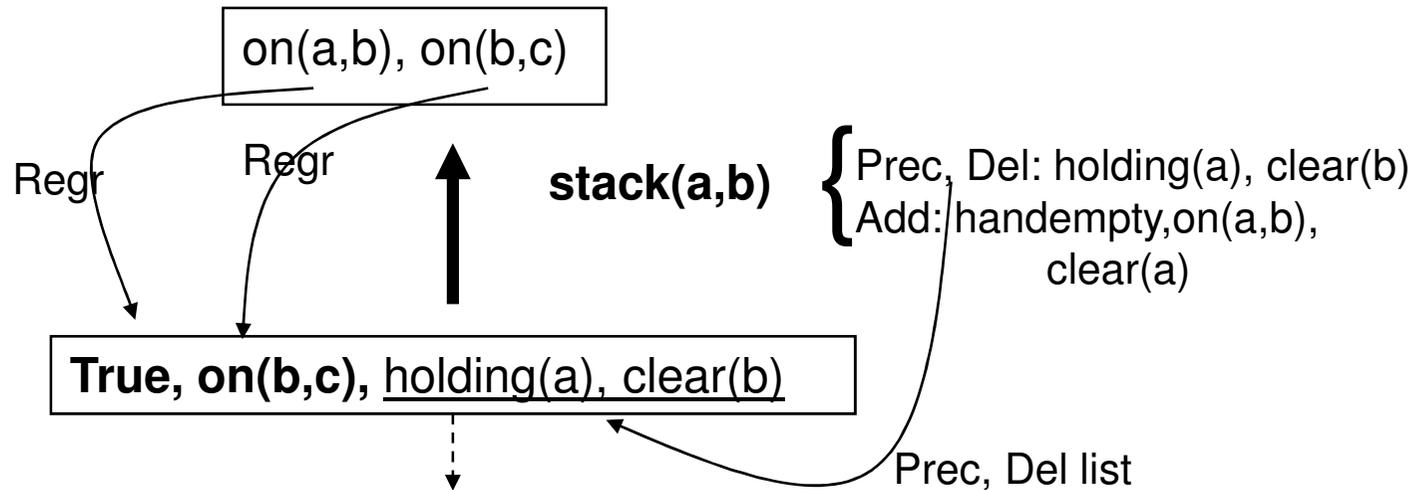
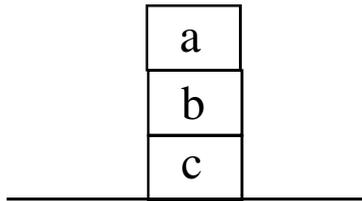
$Regr[G1, R], Regr[G2, R], \dots, Regr[Gn, R],$   
 $D1, D2, \dots, Dk, P1, P2, \dots, Pm$

# Esempio

Stato iniziale: clear(b), clear(c), on(c,a),  
handempty, ontable(a), ontable(b)



Goal: on(a,b), on(b,c)



Nuova lista di goal da soddisfare

# Algoritmo backward

---

- Data la lista  $G:[G_1,G_2,\dots,G_n]$  di tutti i goal e sottogoal del problema ancora da soddisfare e l'insieme delle regole  $R:[R_1,R_2,\dots,R_m]$ , applica la regression per ogni  $R_j$  tale che  $\exists G_i$  per cui  $G_i \in \text{Addlist}(R_j)$  fino a raggiungere un sottogoal soddisfatto dallo stato iniziale.
- Se  $\text{Regr}[G_i,R_j]=\text{false}$  allora taglia quel ramo

**Esempio precedente.** Date le azioni:

pickup(X)

PRECOND:  $\text{ontable}(X)$ ,  $\text{clear}(X)$ ,  $\text{handempty}$

DELETE:  $\text{ontable}(X)$ ,  $\text{clear}(X)$ ,  $\text{handempty}$

ADD:  $\text{holding}(X)$

putdown(X)

PRECOND:  $\text{holding}(X)$

DELETE:  $\text{holding}(X)$

ADD:  $\text{ontable}(X)$ ,  $\text{clear}(X)$ ,  $\text{handempty}$

# Algoritmo backward

---

stack(X,Y)

PRECOND: holding(X), clear(Y)

DELETE: holding(X), clear(Y)

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

unstack(X,Y)

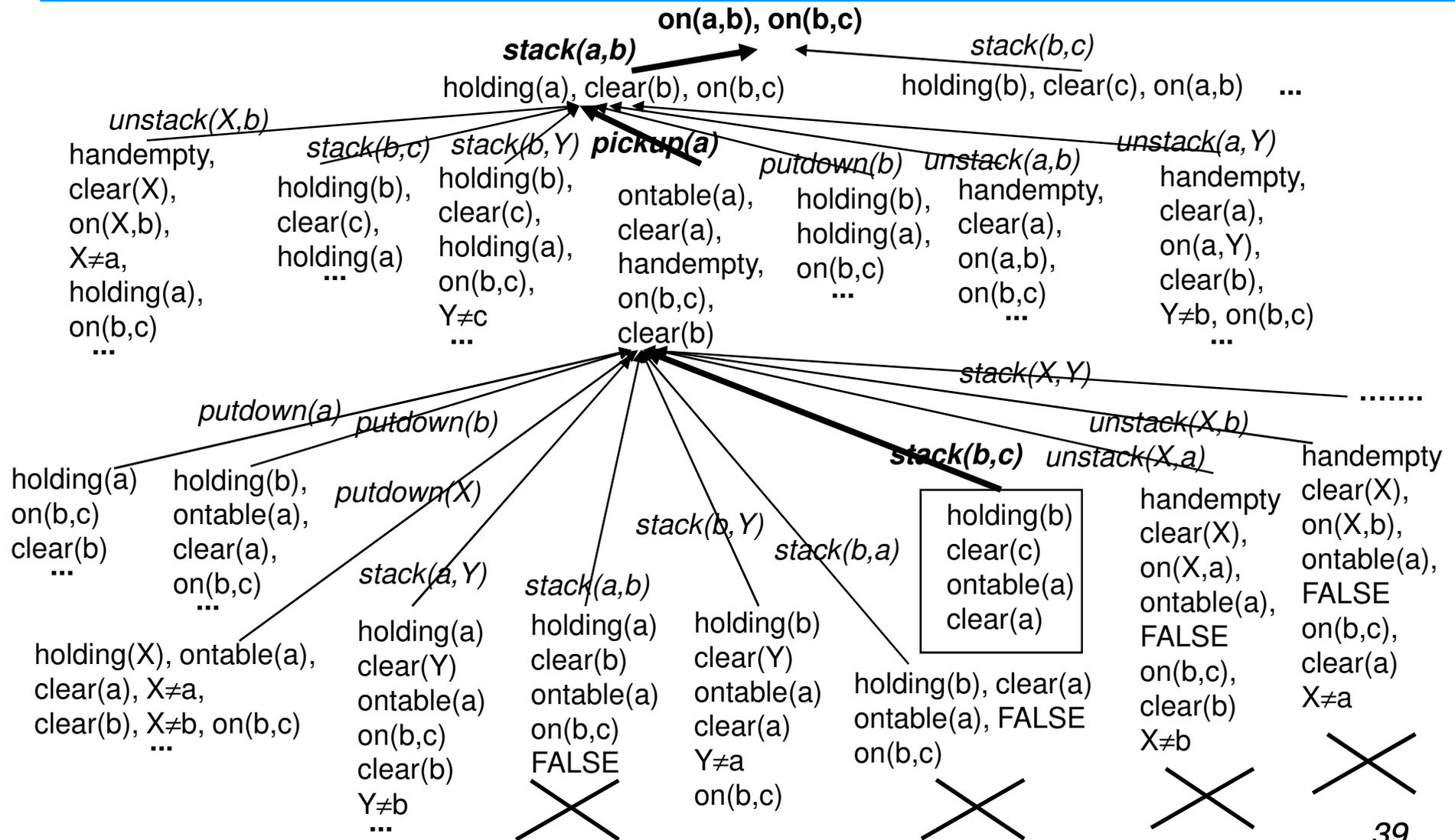
PRECOND: handempty, on(X,Y), clear(X)

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

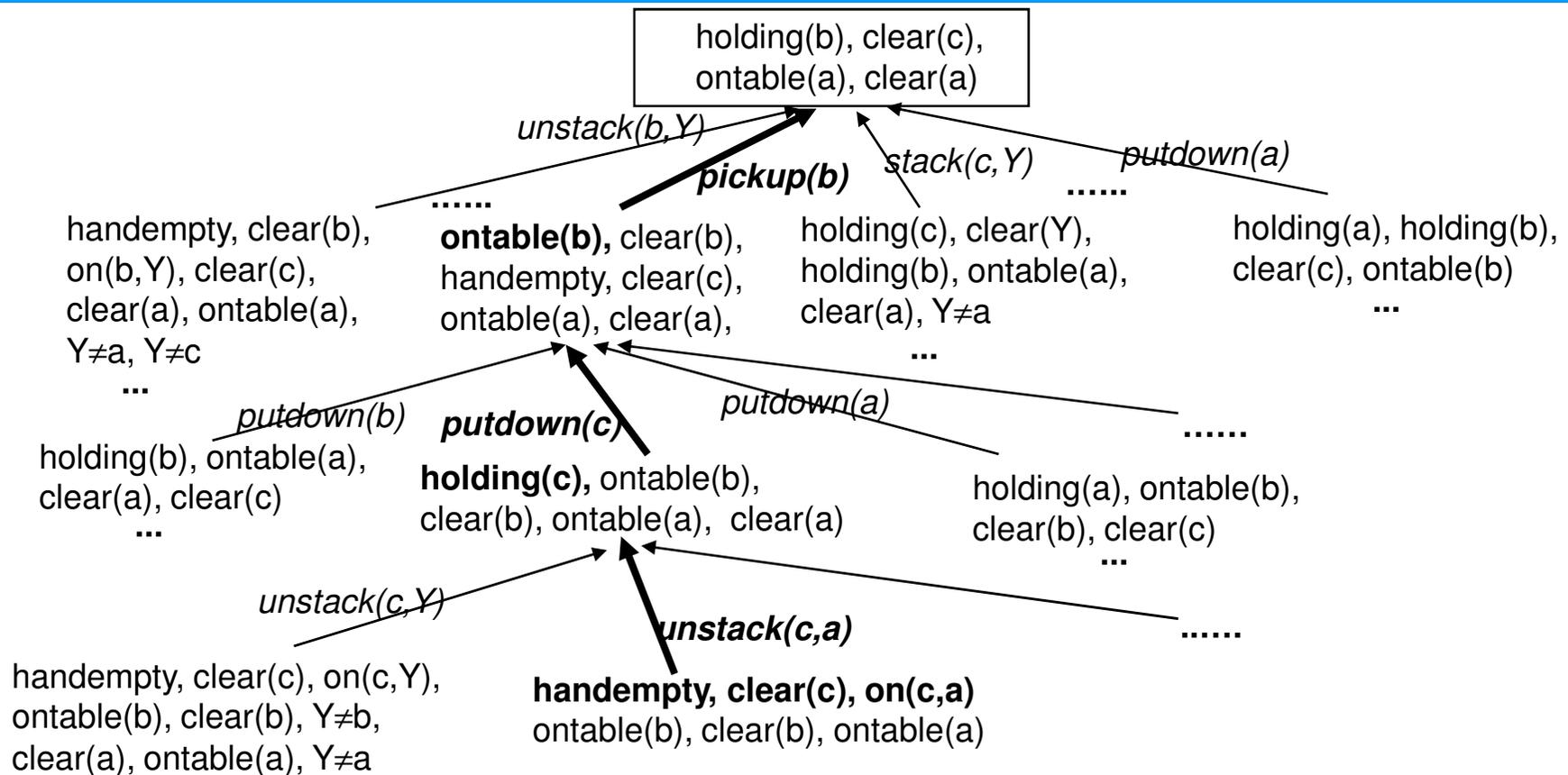
ADD: holding(X), clear(Y)

viene generato il grafo completo mostrato nelle prossime slide

# Esempio di grafo completo



# Esempio di grafo completo



**Questo coincide con  
lo stato iniziale**

# STRIPS

---

**STRIPS** - Stanford Research Institute Problem Solver

Antenato degli attuali sistemi di pianificazione. (primi anni '70)

- Linguaggio per la rappresentazione di azioni. Sintassi molto più semplice del Situation Calculus (meno espressività, più efficienza).
- Algoritmo per la costruzione di piani.

# Linguaggio Strips

---

- Rappresentazione dello stato
  - Insieme di fluent che valgono nello stato  
Esempio: *on(b,a)*, *clear(b)*, *clear(c)*, *ontable(c)*
- Rappresentazione del goal
  - Insieme di fluent (simile allo stato)
  - Si possono avere variabili  
Esempio: *on(X,a)*

# Linguaggio Strips

---

- Rappresentazione delle azioni (3 liste)
  - PRECONDIZIONI: fluent che devono essere veri per applicare l'azione
  - DELETE List: fluent che diventano falsi come risultato dell'azione
  - ADD List: fluent che diventano veri come risultato dell'azione

Esempio *Move(X, Y, Z)*

Precondizioni: *on(X, Y), clear(X), clear(Z)*

Delete List: *clear(Z), on(X, Y)*

Add list: *clear(Y), on(X, Z)*

A volte ADD e DELETE list sono rappresentate come **EFFECT** list con atomi positivi e negativi

Esempio *Move(X, Y, Z)*

Precondizioni: *on(X, Y), clear(X), clear(Z)*

Effect List: *¬clear(Z), ¬on(X, Y), clear(Y), on(X, Z)*

Frame problem risolto con la **Strips Assumption**:

*tutto ciò che non è specificato nella ADD e DELETE list resta immutato*

# *AZIONI in STRIPS (1)*

---

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

## *AZIONI IN STRIPS (2)*

---

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)

# Algoritmo STRIPS

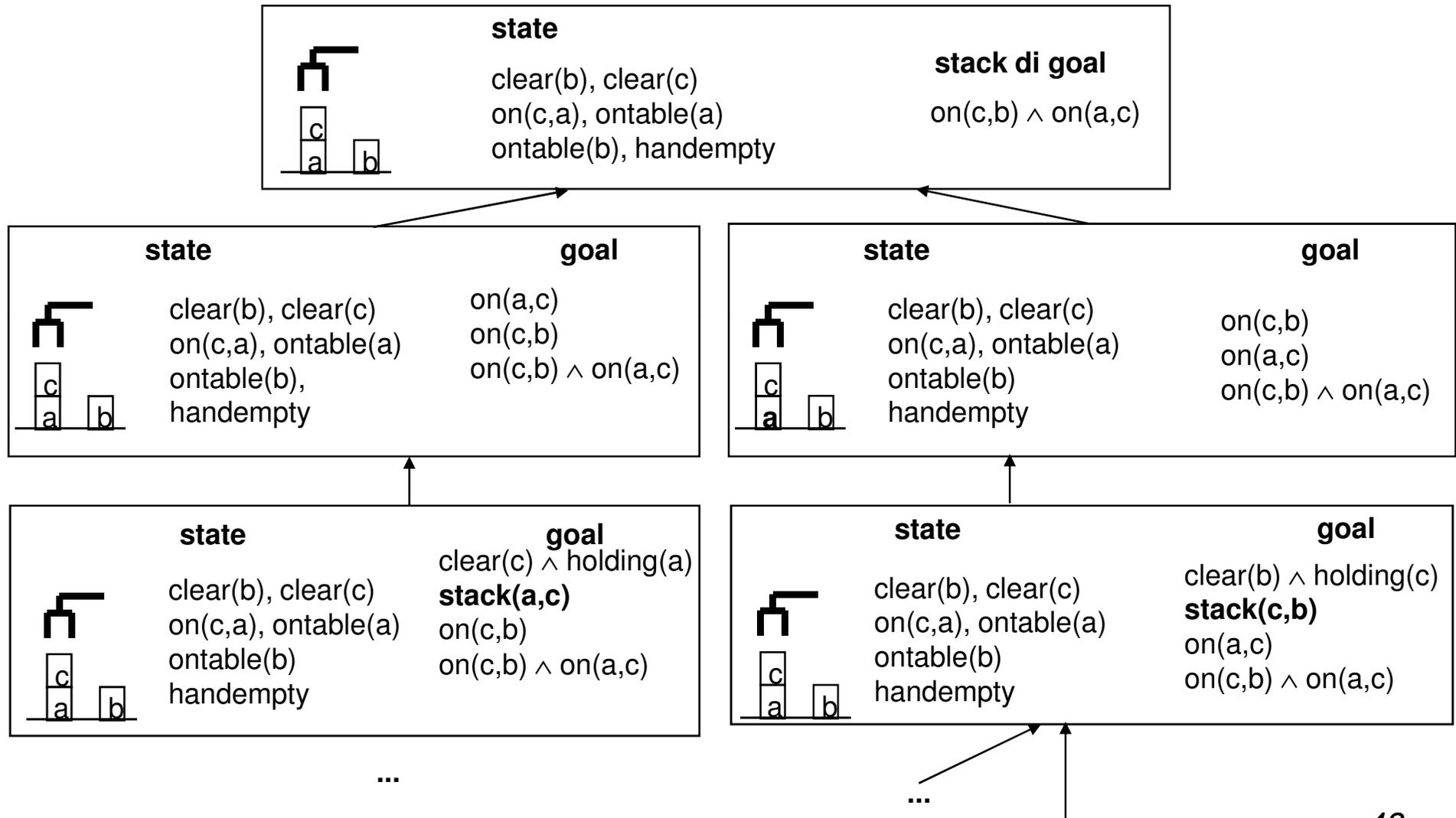
- Planner lineare basato su ricerca backward
- Assume che lo stato iniziale sia completamente noto (**Closed World Assumption**)
- Utilizza due strutture dati
  - stack di goal
  - descrizione S dello stato corrente
- Algoritmo
  - Inizializza stack con la congiunzione di goal finali
  - while stack non è vuoto do
    - if top(stack) = A and  $A \theta \subseteq S$  (si noti che A puo essere un **and** di goals o un atomo)
    - then pop(a) ed esegui sost  $\theta$  sullo stack
    - else if top(stack) = a
      - then
        - seleziona regola R con  $a \in \text{Addlist}(R)$ ,
        - pop(a), push(R), push(Precond(R));
      - else if top(stack) =  $a_1 \wedge a_2 \wedge \dots \wedge a_n$ 
        - (\*) then push( $a_1$ ), ..., push( $a_n$ )
        - else if top(stack) = R
    - then pop(R) e applica R trasformando S
- (\*) si noti che l'ordine con cui i sottogoal vengono inseriti nello stack rappresenta un punto di scelta non deterministica. La congiunzione rimane sullo stack e verrà riverificata dopo - interacting goals)

# Considerazioni sull'algoritmo

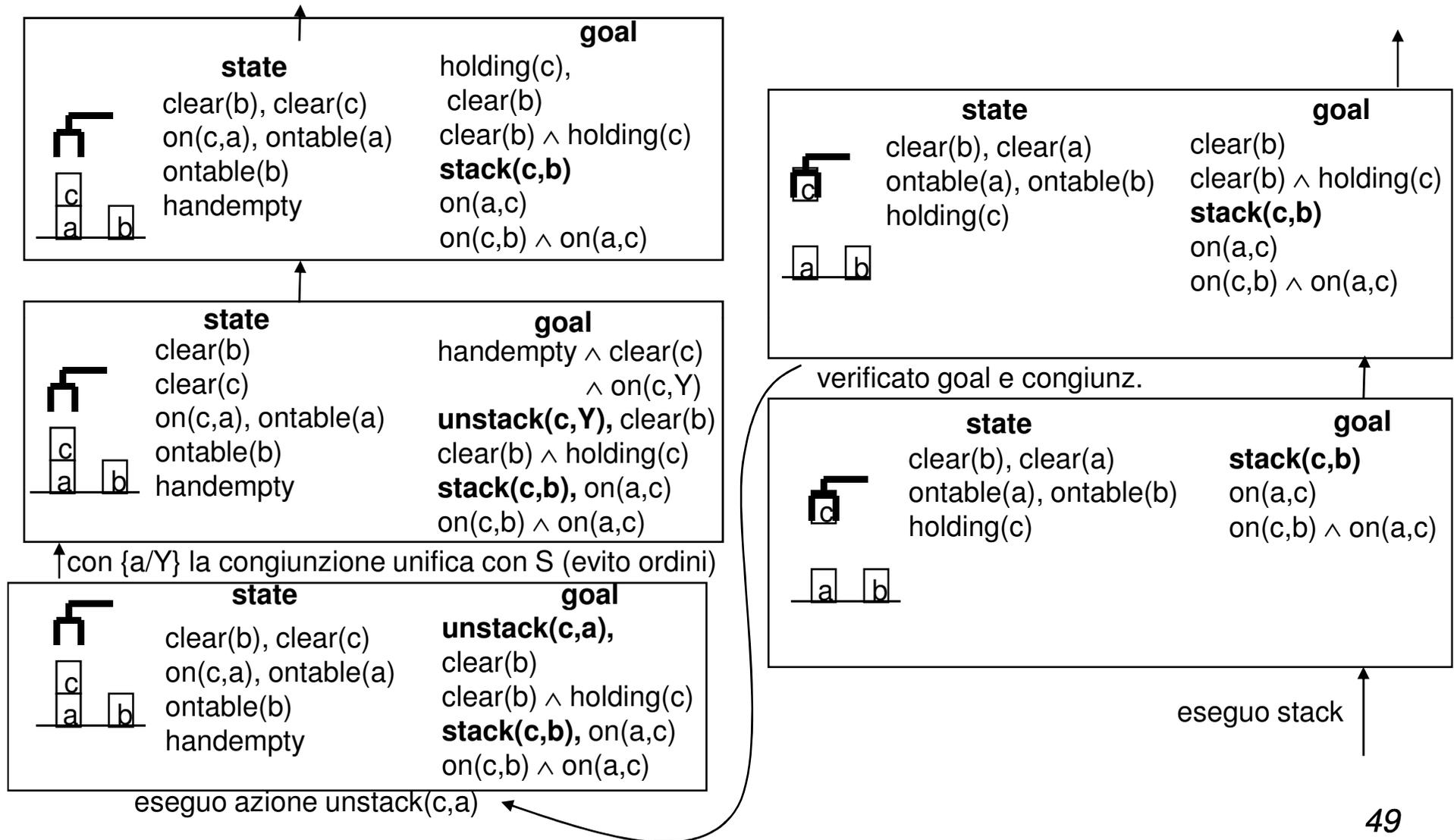
---

1. Il goal è la prima pila di obiettivi.
2. Suddividere il problema in sottoproblemi: ciascuno per un componente dell'obiettivo originale. Tali sottoproblemi possono interagire
3. Abbiamo tanti possibili ordini di soluzione.
4. Ad ogni passo del processo di risoluzione si cerca di risolvere il goal in cima alla pila.
5. Quando si ottiene una sequenza di operatori che lo soddisfa la si applica alla descrizione corrente dello stato ottenendo una nuova descrizione.
6. Si cerca poi di soddisfare l'obiettivo che è in cima alla pila partendo dalla situazione prodotta dal soddisfacimento del primo obiettivo.
7. Il procedimento continua fino allo svuotamento della pila.
8. Quando in cima alla pila si incontra una congiunzione si verifica che tutte le sue componenti siano effettivamente soddisfatte nello stato attuale. Se una componente non è soddisfatta (problema dell'interazione tra goal è spiegato più avanti) si reinserisce nella pila e si continua.

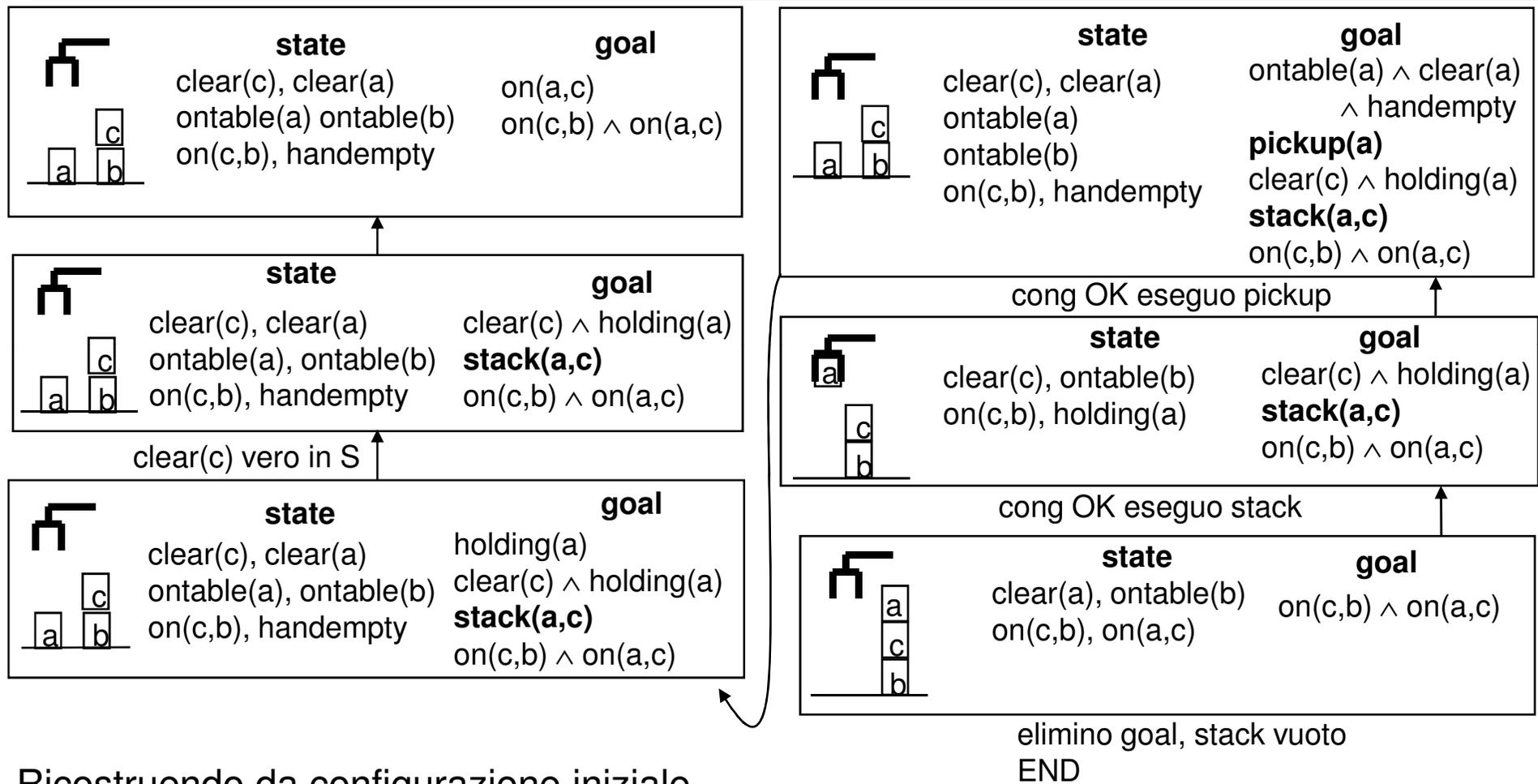
# Esempio



# Esempio



# Esempio



Ricostruendo da configurazione iniziale a finale ho una soluzione:

1. unstack(c,a)

2. stack(c,b)

3. pickup(a)

4. stack(a,c)

50

# Problemi (1)

---

**1. Grafo di ricerca molto vasto.** Nell'esempio abbiamo visto un cammino ma in realtà ci sono varie alternative

- scelte non deterministiche ordinamento dei goal
- più operatori applicabili per ridurre un goal

**Soluzione:** Strategie euristiche

- strategie di ricerca
- strategie per scegliere quale goal ridurre e quale operatore
  - **MEANS-ENDS ANALYSIS**
    - cercare la differenza più significativa tra stato e goal
    - ridurre quella differenza per prima

# Problemi (2)

---

**2. Problema dell'interazione tra goal.** Quando due (o più) goal inter-agiscono ci possono essere problemi di interazione tra le due soluzioni.

Goal G1, G2

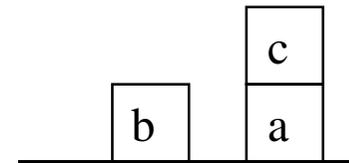
- pianifico azioni per G2
  - poi per risolvere G1 potrei dover smontare tutto, compreso lo stato che avevo prodotto con G2 risolto
- Soluzione completa:
- provare tutti gli ordinamenti dei goal e dei loro sottogoal.
- Soluzione pratica (Strips):
- provare a risolverli indipendentemente
  - verificare che la soluzione funzioni
  - se non funziona, provare gli ordinamenti possibili uno alla volta

# Esempio: Anomalia di Sussmann

---

Stato iniziale (come esempio precedente):

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

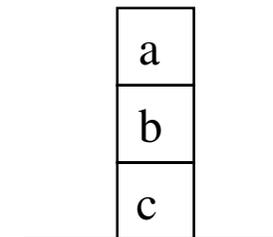


Goal:  $on(a,b), on(b,c)$

Possibili stack iniziali:

(1)  
 $on(a,b)$   
 $on(b,c)$   
 $on(a,b) \wedge on(b,c)$

(2)  
 $on(b,c)$   
 $on(a,b)$   
 $on(a,b) \wedge on(b,c)$



Decidiamo di scegliere (1)

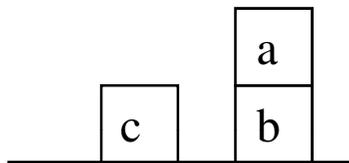
# Esempio: Anomalia di Sussmann

---

Applicando il procedimento di soluzione di STRIPS otteniamo:

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

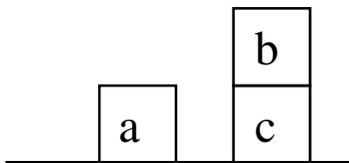
Stato attuale:



Adesso lavoriamo al soddisfacimento del secondo goal  $on(b,c)$ .

5. unstack(a,b)
6. putdown(a)
7. pickup(b)
8. stack(b,c)

Stato attuale:



# Esempio: Anomalia di Sussmann

---

La congiunzione  $\text{on}(a,b) \wedge \text{on}(b,c)$  non è valida.

Allora reinseriamo  $\text{on}(a,b)$  nello stack di goals (è la differenza rispetto allo stato attuale).

Otteniamo:

9. pickup(a)

10. stack(a,b)

Abbiamo ottenuto quello che volevamo, ma in modo scarsamente efficiente.

# Problemi (3)

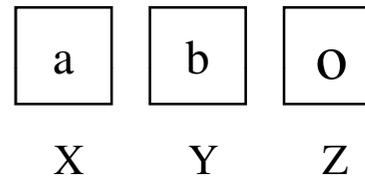
---

## 3. Perdita di informazioni:

Supponiamo di voler utilizzare STRIPS per generare un piano che attui lo scambio di valore fra due celle di memoria o registri X e Y.

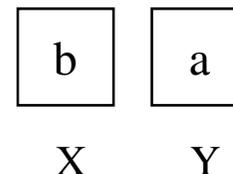
Stato iniziale:

cont(x,a) and cont(y,b) and cont(z,o)



Stato finale:

cont(x,b) and cont(y,a)



Operazione di assegnamento:

ASSIGN(X,Y,T,S)

Precondition: cont(Y,S) and cont(X,T)

Delete: cont(X,T)

Add: cont(X,S)

# Problemi (3)

---

Goal stack:

cont(x,b),  
cont(y,a),  
cont(x,b)  $\wedge$  cont(y,a)

Per ridurre il goal cont(x,b) applica ASSIGN con Y/y e T/a

cont(y,b)  $\wedge$  cont(x,a), **assign(x,y,a,b)**,

Dopo l'esecuzione lo stato diventa:

<div style="border: 1px solid black; display: inline-block; padding: 5px;">b</div>	<div style="border: 1px solid black; display: inline-block; padding: 5px;">b</div>	<div style="border: 1px solid black; display: inline-block; width: 40px; height: 40px;"></div>
X	Y	Z

cont(x,b), cont(y,b), cont(z,o)

Il goal cont(y,a) rimasto nello stack non potrà mai essere soddisfatto.

Proprietà distruttiva dello statement di assegnamento!

STRIPS è troppo "ansioso" di risolvere l'obiettivo e quindi distrugge inesorabilmente il contenuto di una cella di memoria!

# 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

---

- 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, con effetti una descrizione completa dello stato iniziale
  - **stop**: con precondizione il goal del planner
  - start < stop

# Ricerca nello spazio dei piani

## Pianificazione Non Lineare

---

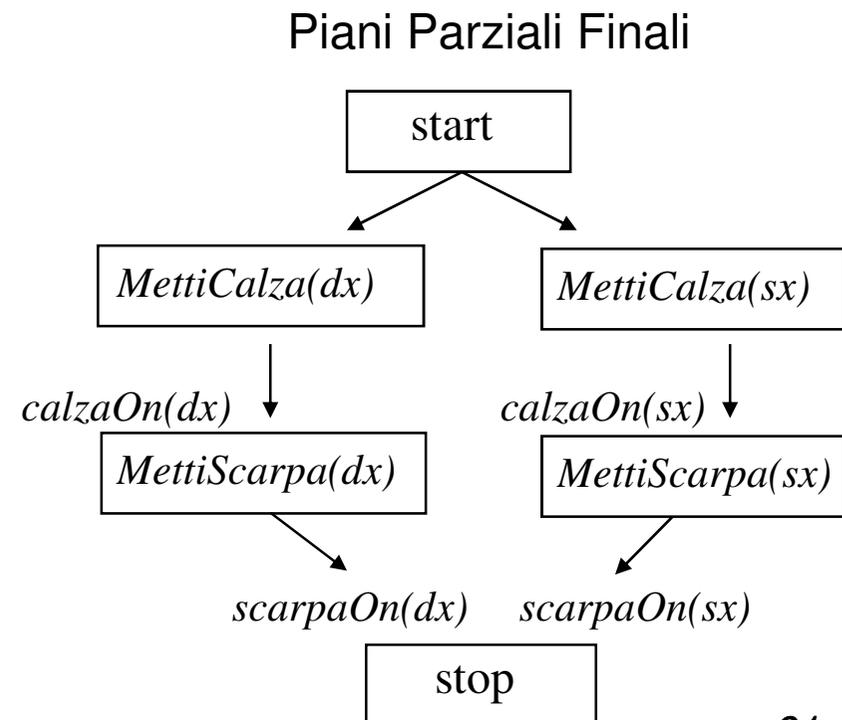
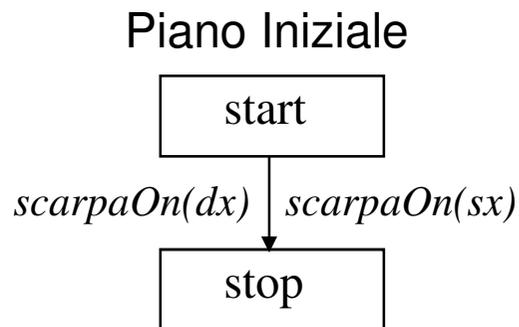
- Ad ogni passo si incrementano l'insieme degli operatori, degli ordinamenti parziali (non vengono imposti ordinamenti non richiesti) e dei causal link fino a che tutti i goal sono risolti.
- Una soluzione è un insieme di operatori parzialmente specificato e parzialmente ordinato.
- Per ottenere un piano 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:  $calzaOn(Piede)$ ,  $\neg scarpaOn(Piede)$

EFFECT:  $scarpaOn(Piede)$

- MettiCalza(Piede)  
PRECOND:  $\neg calzaOn(Piede)$   
EFFECT:  $calzaOn(Piede)$



# 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  $\langle S, SN, C \rangle$ ;
- risolvi eventuali violazioni a causal links

End

# Algoritmo di Partial Order Planning (intuitivo)

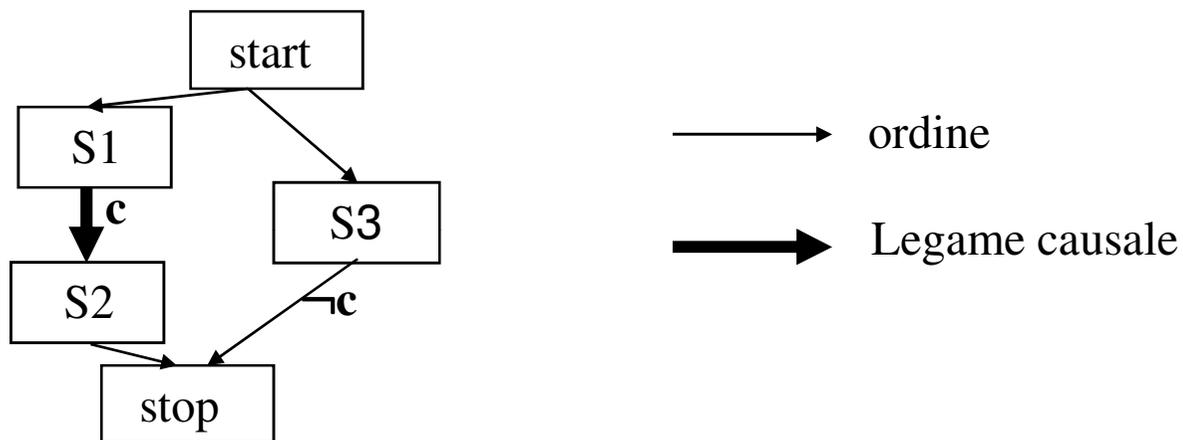
---

- 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  $S_i$ ,  $S_j$  e un sottogoal  $c$  tali che uno degli effetti di  $S_i$  soddisfa la preconditione  $c$  di  $S_j$ .

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*).

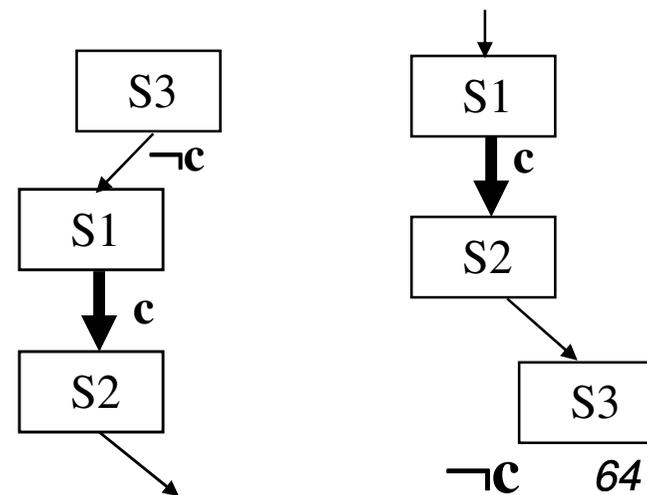
# Violazioni a vincoli causali (“minacce”)

Si dice che un'azione  $S3$  rappresenta una **minaccia** (*threat*) per un causal link  $\langle S1, S2, c \rangle$  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) x<> y*

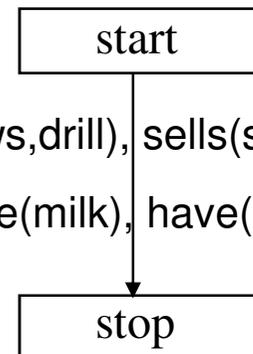
**buy(S,Y):**

PRECOND: *at(S), sells(S,Y)*

EFFECT: *have(Y)*

➤ Piano iniziale (detto *nullo*):

*at(home), sells(hws,drill), sells(sm,milk), sells(sm,banana)*  
*have(drill), have(milk), have(banana), at(home)*

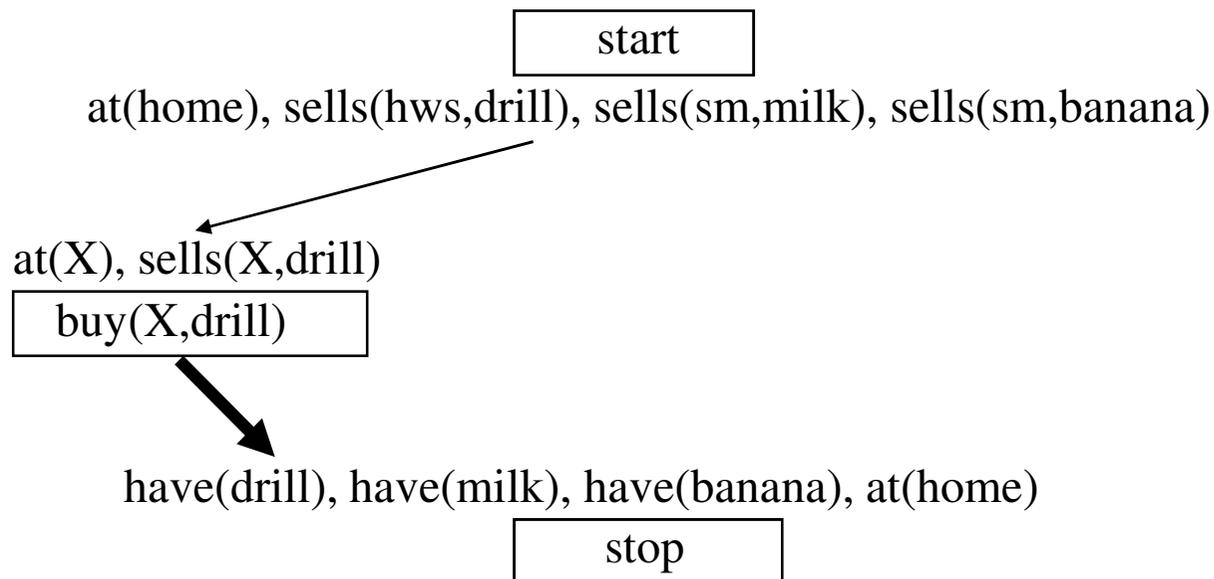


# Esempio: Pianificazione di Acquisti

---

➤ Primo passo:

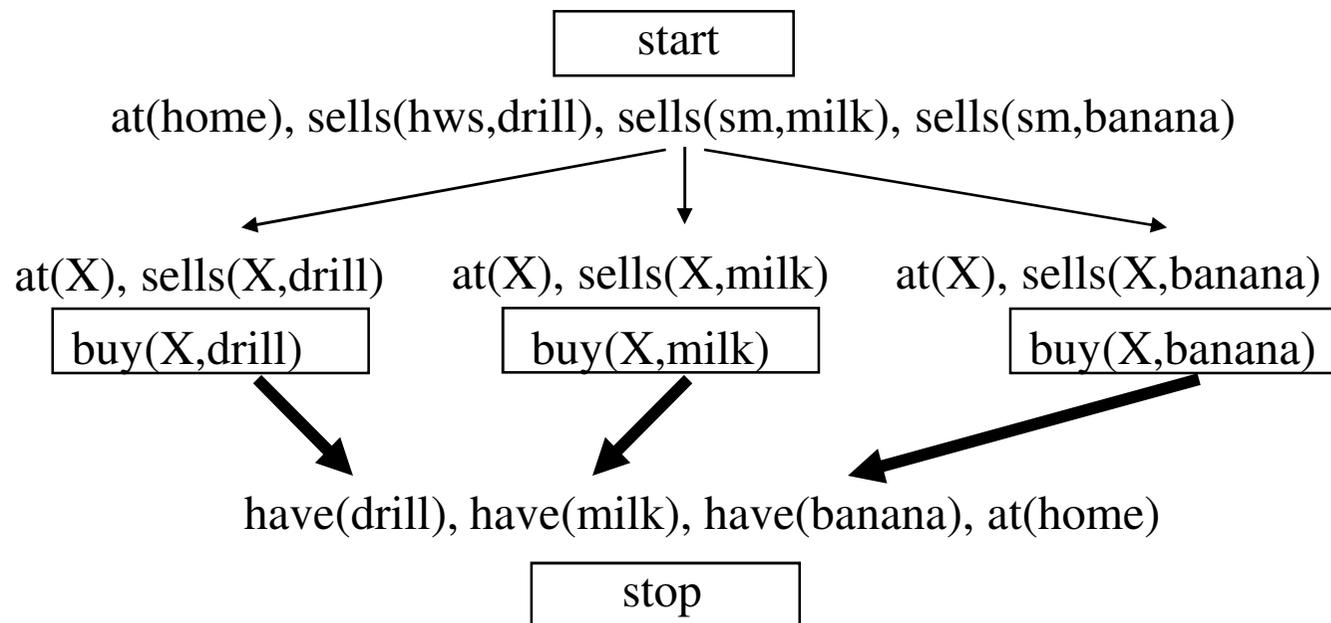
- seleziono una preconditione (goal) da soddisfare: *have(drill)*
- seleziono una azione che ha *have(drill)* come effetto: *buy(X, Y)*
- modifico il piano:
  - istanzio Y a *drill*
  - vincoli di ordinamento *Start < buy < Stop*
  - causal link *< buy(X, drill), stop, have(drill) >*



# Esempio: Pianificazione di Acquisti

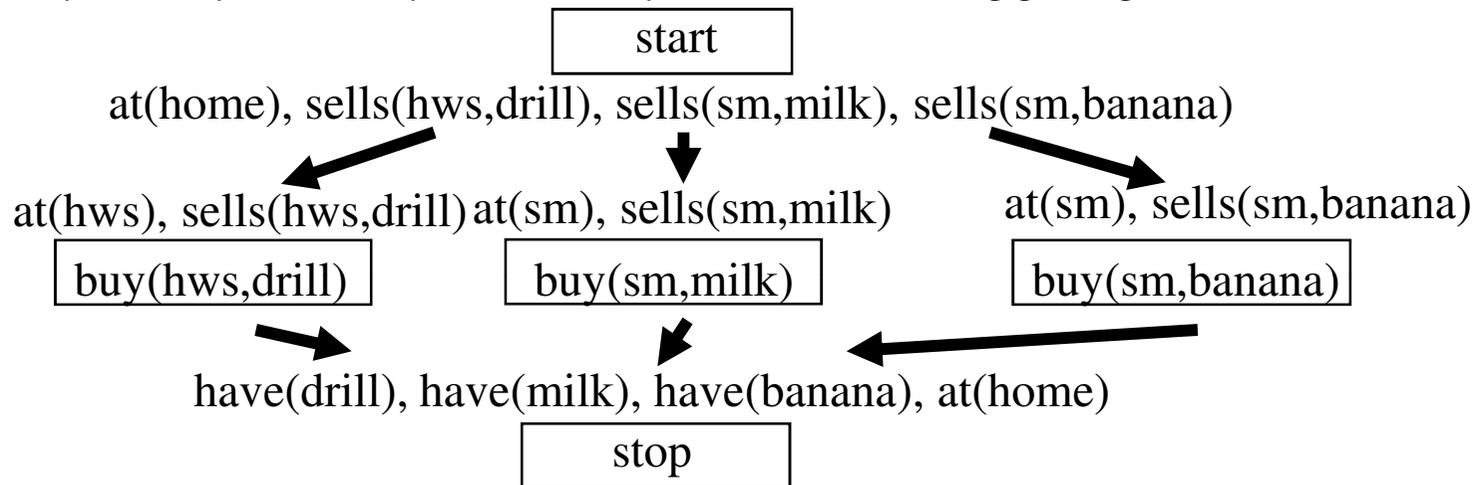
---

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



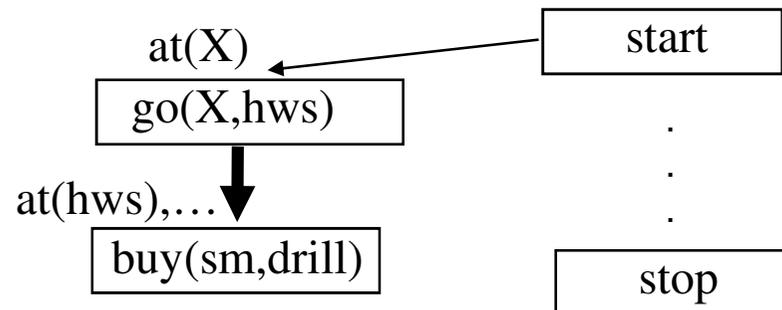
# Esempio: Pianificazione di Acquisti

- 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.



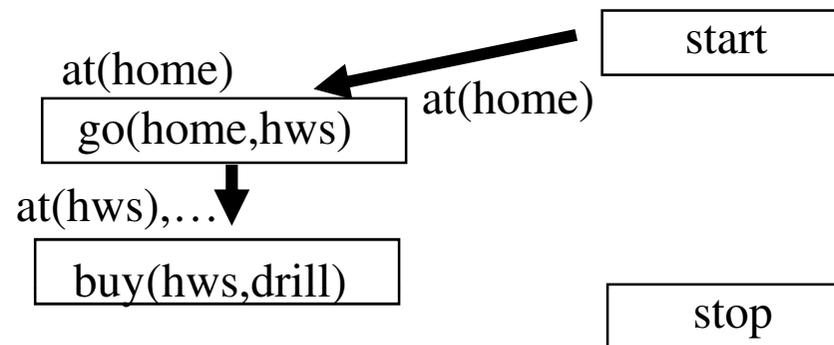
- Seleziono

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

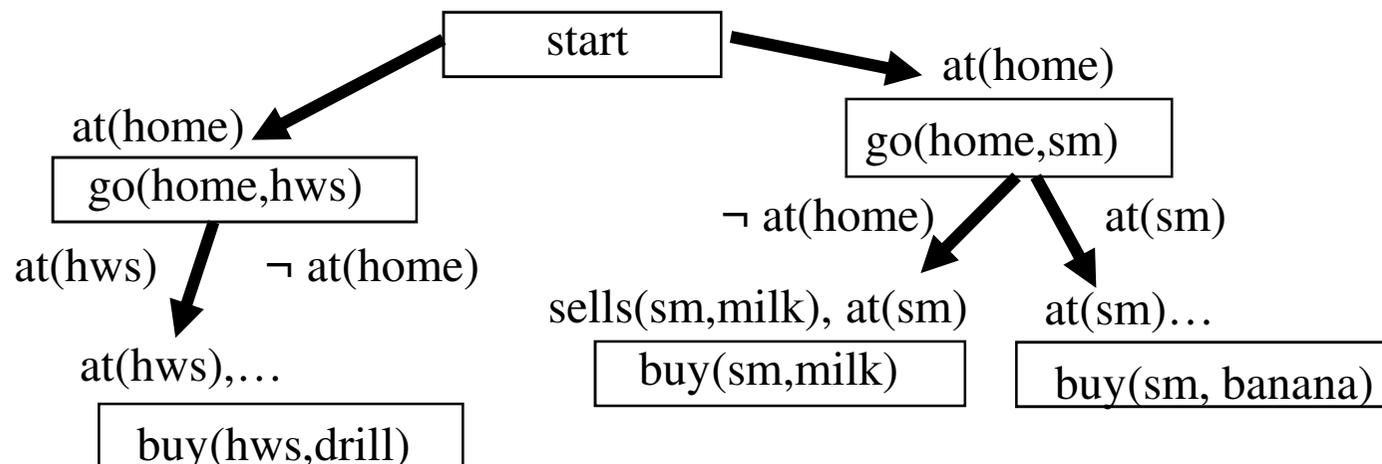


# Esempio: Pianificazione di Acquisti

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

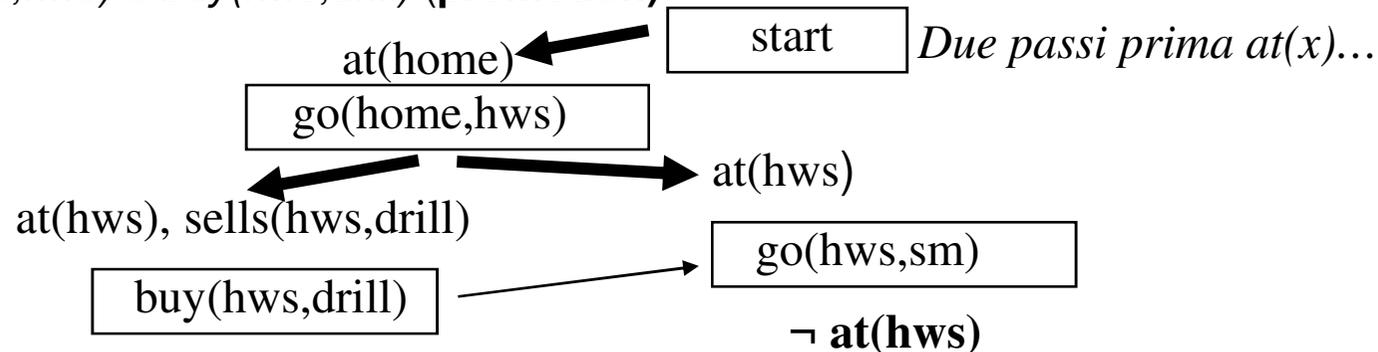


- Procedo in modo analogo per  $at(sm)$

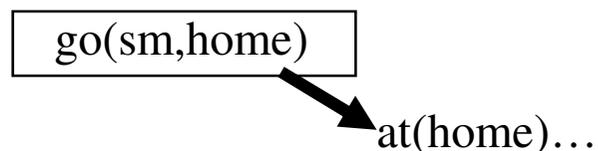


# Esempio: Pianificazione di Acquisti

- 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**)

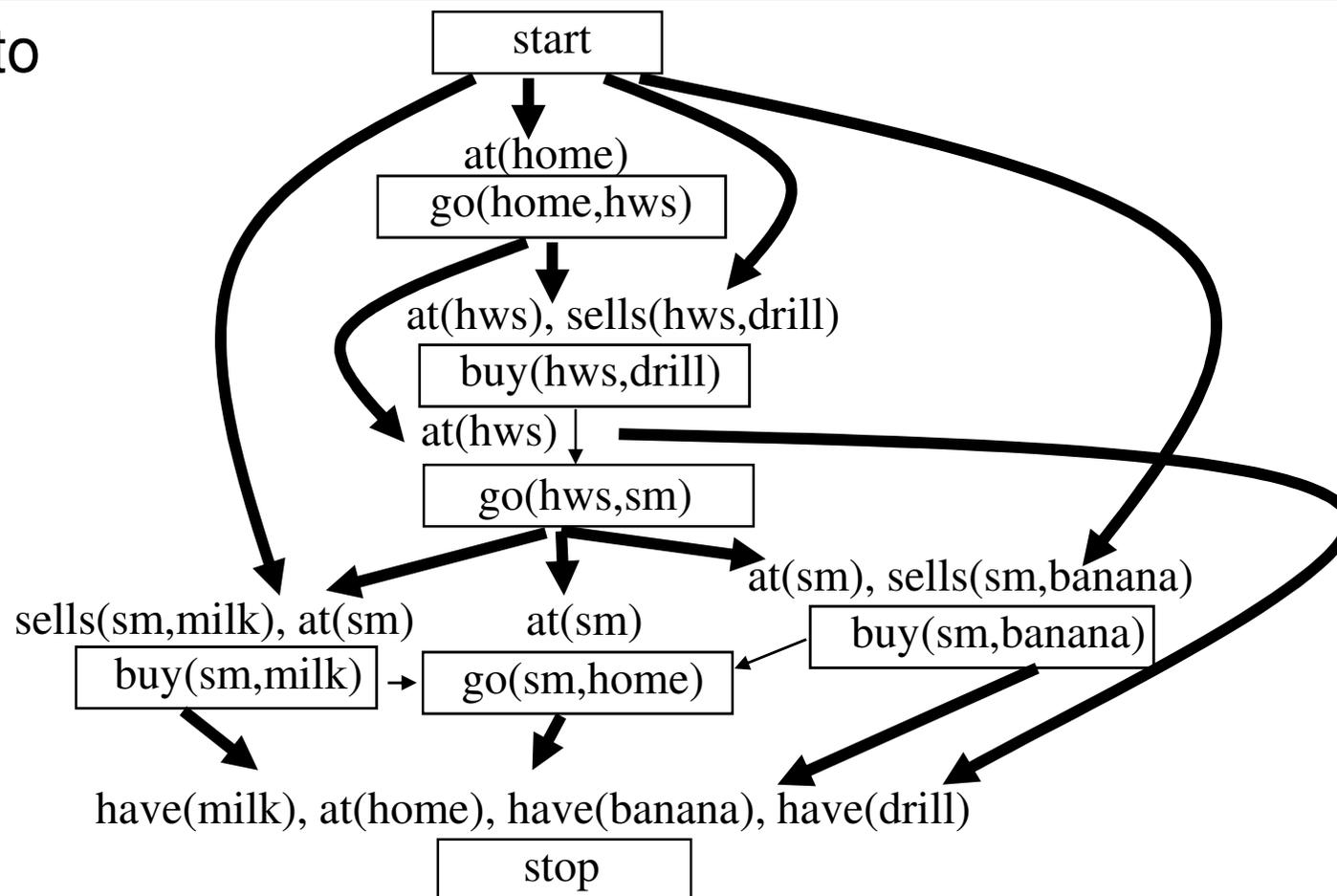


- Ultimo passo:
  - risolvere  $at(home)$  di  $stop$ : unico modo è mettere azione  $go(home)$  prima di  $stop$



# 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)*<sup>71</sup>

# 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  $\langle S, SN, C \rangle$

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  $S_i$  e  $S_j$ ,

choose either

demotion: aggiungi il vincolo  $S < S_i$

promotion: aggiungi il vincolo  $S_j < S$

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  $pickup(X) \xrightarrow{holding(X)} stack(X,b)$

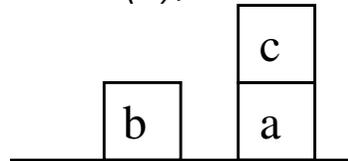
la minaccia rappresentata da un'eventuale azione  $stack(Y,c)$  può essere eliminata ponendo il vincolo  $X \neq Y$

# Esempio: Anomalia di Sussmann

---

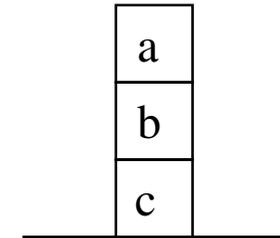
Stato iniziale ([*s*]):

*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,c)<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*

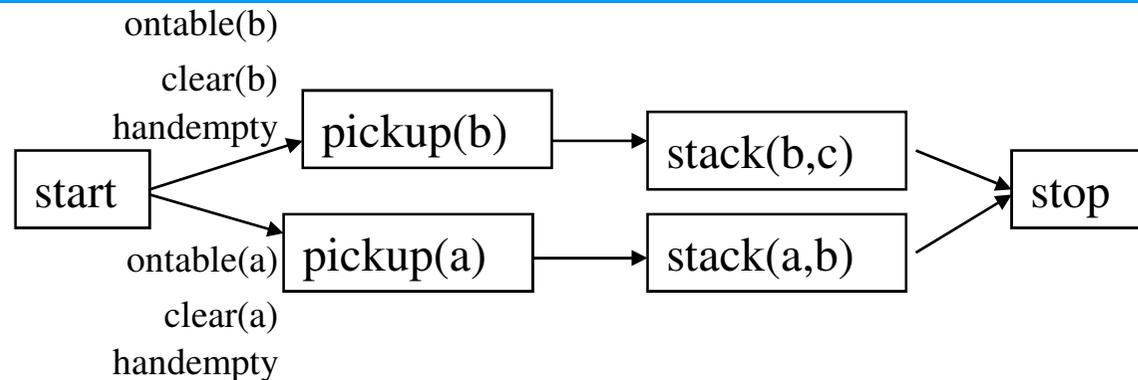
*Lista degli ordinamenti :* [..., pickup(a)<stack(a,b), pickup(b)<stack(b,c)]

*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] <sup>78</sup>

# 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 : [ $handempty$ , $clear(a)$ , $handempty$ ]

$stack(a,b)$  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  $\langle \text{start}, \text{pickup}(b), \text{handempty} \rangle$  è minacciato da  $\text{pickup}(a)$ :



Imponiamo (promotion)  $\text{pickup}(b) < \text{pickup}(a)$

La precondizione *handempty* di  $\text{pickup}(a)$  non può essere soddisfatta da *start* perché  $\text{pickup}(b)$  che precede  $\text{pickup}(a)$  la disfarebbe.



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

$\text{pickup}(b) < \text{stack}(b,c) < \text{pickup}(a)$

*Lista degli ordinamenti*: [...,  $\text{pickup}(b) < \text{stack}(a,b)$ ,  $\text{pickup}(b) < \text{pickup}(a)$ ,  
 $\text{stack}(b,c) < \text{pickup}(a)$ ]

*Lista caus. link*: [...,  $\langle \text{start}, \text{pickup}(b), \text{handempty} \rangle$ ,  $\langle \text{stack}(b,c), \text{pickup}(a), \text{handempty} \rangle$ ].

*Agenda di goal*: [ $\text{clear}(a)$ ]

# Esempio: Anomalia di Sussmann

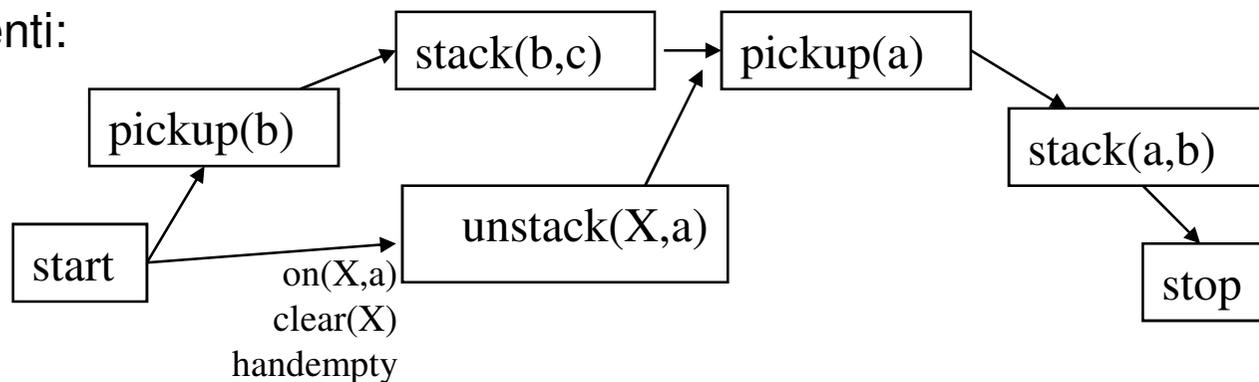
Per raggiungere  $clear(a)$  possiamo aggiungere l'azione:

**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  $unstack(X,a)$ .

# Esempio: Anomalia di Sussmann

---

$unstack(c,a)$  rappresenta una minaccia per il causal link  $\langle start, pickup(b), handempty \rangle$

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

---

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)  
<http://www.cs.washington.edu/research/projects/softbots/www/softbots.html>
  - Gestione di strumentazione spaziale  
<http://rax.arc.nasa.gov/>
  - 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*

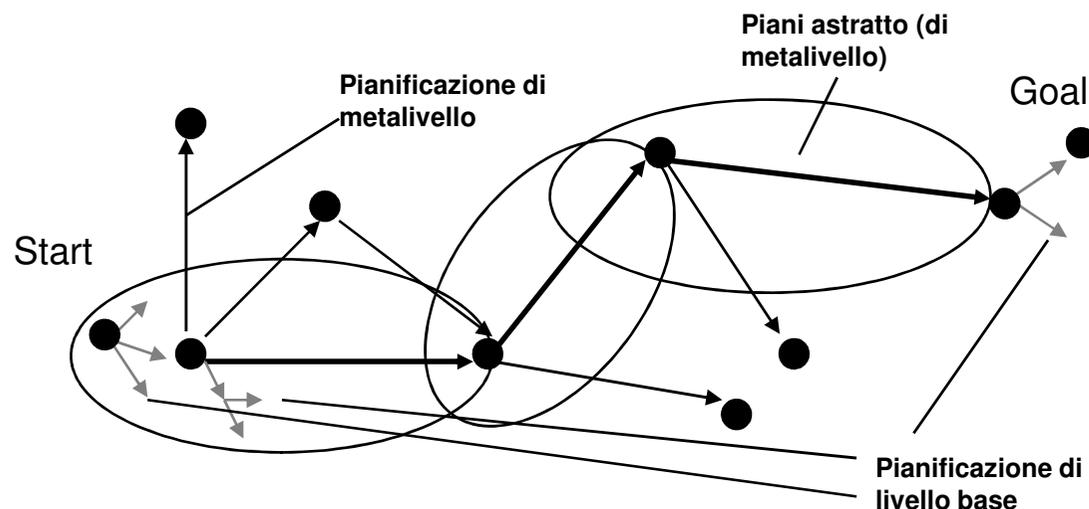
Dato un goal il pianificatore gerarchico effettua una ricerca di “meta-livello” per generare un piano anch’esso detto “di meta livello” che porta da uno stato molto vicino allo stato iniziale ad uno stato molto vicino al goal.

# 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 preconditione 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 preconditioni 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<sup>(\*)</sup> 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)*

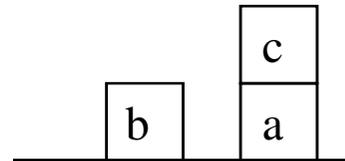
*clear(c)*

*on(c,a)*

*handempty*

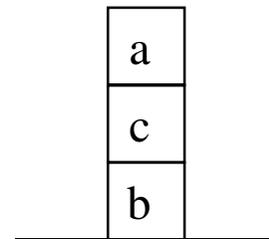
*ontable(a)*

*ontable(b)*



- Goal:

$on(c,b) \wedge on(a,c)$



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)  $\wedge$  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)	on(a,c)
clear(c)	on(c,b) $\wedge$ on(a,c)
ontable(a)	
ontable(b)	
<b>on(c,b)</b>	
handempty	
on(c,a)	

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.<sup>91</sup>

# 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)  $\wedge$  clear(b)

**stack(c,b)**

holding(a)

clear(c)

holding(a)  $\wedge$  clear(c)

**stack(a,c)**

on(c,b)  $\wedge$  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)  $\wedge$  clear(b)  
**stack(c,b)**  
holding(a)  
clear(c)  
holding(a)  $\wedge$  clear(c)  
**stack(a,c)**  
on(c,b)  $\wedge$  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(b)
clear(a)	holding(c) $\wedge$ clear(b)
ontable(a)	<b>stack(c,b)</b>
ontable(b)	holding(a)
handempty	clear(c)
holding(c)	holding(a) $\wedge$ clear(c)
	<b>stack(a,c)</b>
	on(c,b) $\wedge$ 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  $\wedge$  clear(c)  $\wedge$  on(c,a)

**unstack(c,a)**

holding(c)  $\wedge$  clear(b)

**stack(c,b)**

handempty  $\wedge$  clear(a)  $\wedge$  ontable(a)

**pickup(a)**

holding(a)  $\wedge$  clear(c)

**stack(a,c)**

on(c,b)  $\wedge$  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.

# 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 deterministici, imprevedibili

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.

# 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  $T_0$ , accendi il condizionatore;
- 2) Se la temperatura della stanza è  $K$  gradi sotto  $T_0$ , 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.