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

- Utilizza linguaggi specializzati per rappresentare stati, goal e azioni (STRIPS, ADL, etc)
- 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. ⟹ ***Pianificazione lineare***

- nello **spazio dei piani**

  Nell'albero di ricerca ogni nodo rappresenta un piano parziale e ogni arco un'operazione di raffinamento del piano

  ⟹ ***POP***

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)*

Il frame problem è risolto con la **STRIPS Assumption**: *tutto ciò che non è specificato nella ADD e DELETE list resta immutato*

Planning Lineare

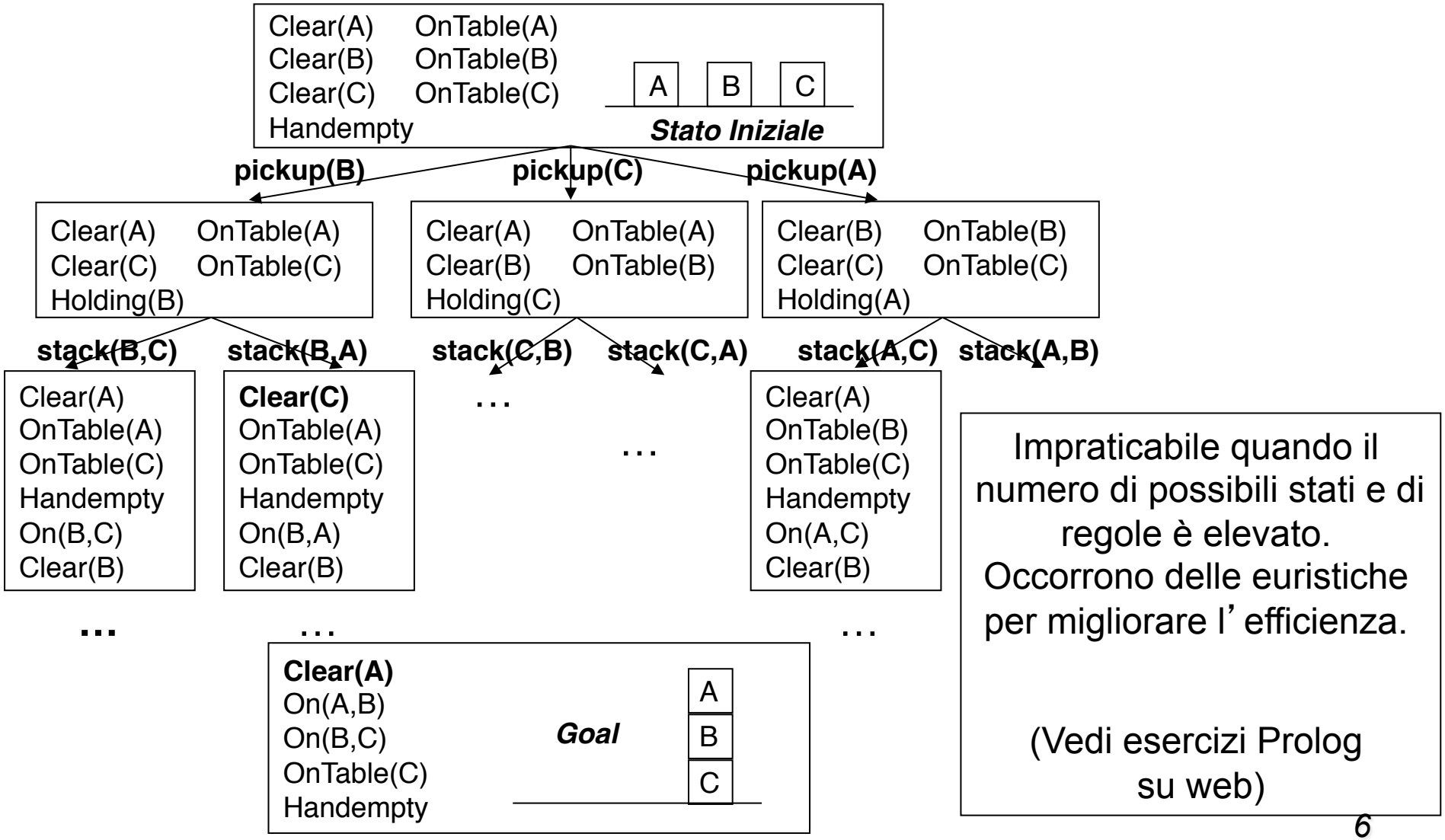
- Un **pianificatore lineare** riformula il problema di pianificazione come problema di ricerca nello spazio degli stati e utilizza le strategie di ricerca classiche (*già viste in Fondamenti di IA*)

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

Ricerca Forward: un esempio



Impraticabile quando il numero di possibili stati e di regole è elevato. Occorrono delle euristiche per migliorare l'efficienza.

(Vedi esercizi Prolog su web)

Planning come ricerca forward (esercitazione Prolog)

- Rappresentazione dello stato (iniziale, finale, ...)
- Si parte dal nodo (stato) iniziale e l'albero (o grafo) viene espanso applicando, ad ogni nodo, gli operatori le cui precondizioni sono soddisfatte nello stato che il nodo stesso rappresenta
- La ricerca di uno stato finale nel grafo/ albero può avvenire in modi differenti:
  - ricerca in **profondità**;
  - ricerca in **ampiezza**;
  - ricerca **euristica**.

Planning Lineare Backward

- Problema: occorre definire come si effettua la *riduzione di un goal in sottogoal*, tenendo conto della forma delle regole (azioni)

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

Goal Regression

Dati un goal *G (atomico)* e una regola *R*

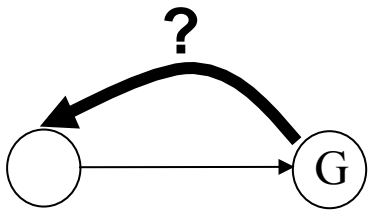
PRECOND: *Plist*

DELETE: *Dlist*

ADD: *Alist*

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

- Regr[G,R] = true se G∈Alist
- Regr[G,R] = false se G∈Dlist
- Regr[G,R] = G altrimenti



Esempio

Data R1:*unstack(X,Y)*

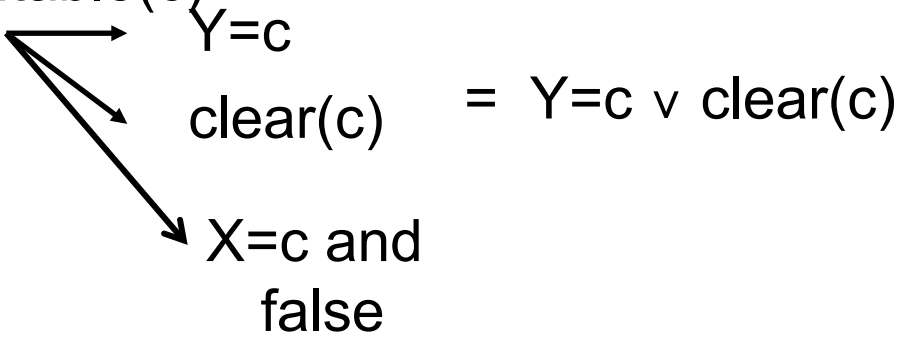
PRECOND: handempty, on(X,Y), clear(X)

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

ADD: holding(X), clear(Y)

si ha:

- Regr[holding(b), R1] = true X=b
- Regr[handempty,R1]=false
- Regr[ontable(c),R1]=ontable(c)
- Regr[clear(c), R1] =



Riduzione di goal in sottogoal

Dato il goal *G:[G1, G2, …, Gn]* e la regola *R*

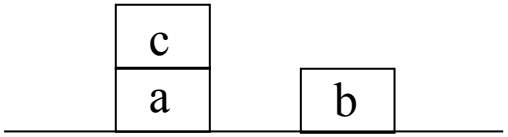
R: PRECOND: *Plist* = *P1, P2, …Pm*
DELETE: *Dlist* = *D1, D2, …, Dk*
ADD: *Alist* = *A1, A2, … Al*

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

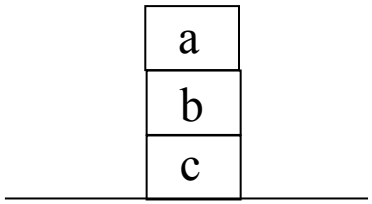
Regr[G1, R], Regr[G2, R], …, Regr[Gn,R],
D1, D2, …,Dk, P1, P2, … Pm

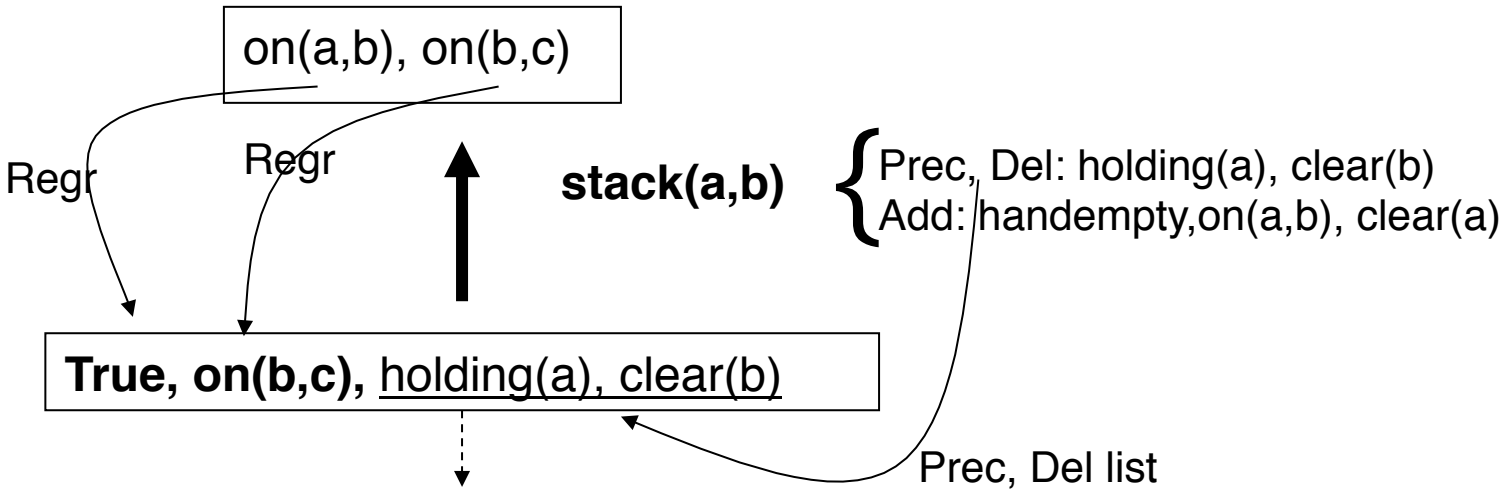
Esempio

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

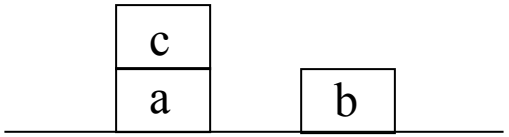
- Data la lista G:[G1,G2,…,Gn] di tutti i goal e sottogoal del problema ancora da soddisfare e l'insieme delle regole R: [R1,R2,…,Rm], applica la regression per ogni Rj tale che ∃ Gi per cui Gi∈Addlist(Rj) fino a raggiungere un sottogoal soddisfatto dallo stato iniziale

- Se Regr[Gi,Rj]=false allora taglia quel ramo

Esempio (*il precedente*)

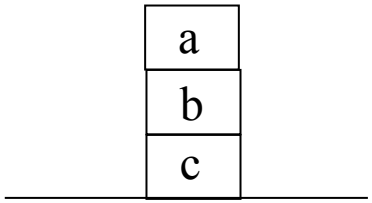
Stato iniziale:

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



Goal:

on(a,b), on(b,c)



Esempio (*il precedente*)

- Date le azioni:

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

Esempio (*il precedente*)

stack(X,Y)

PRECOND: holding(X), clear(Y)

DELETE: holding(X), clear(Y)

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

unstack(X,Y)

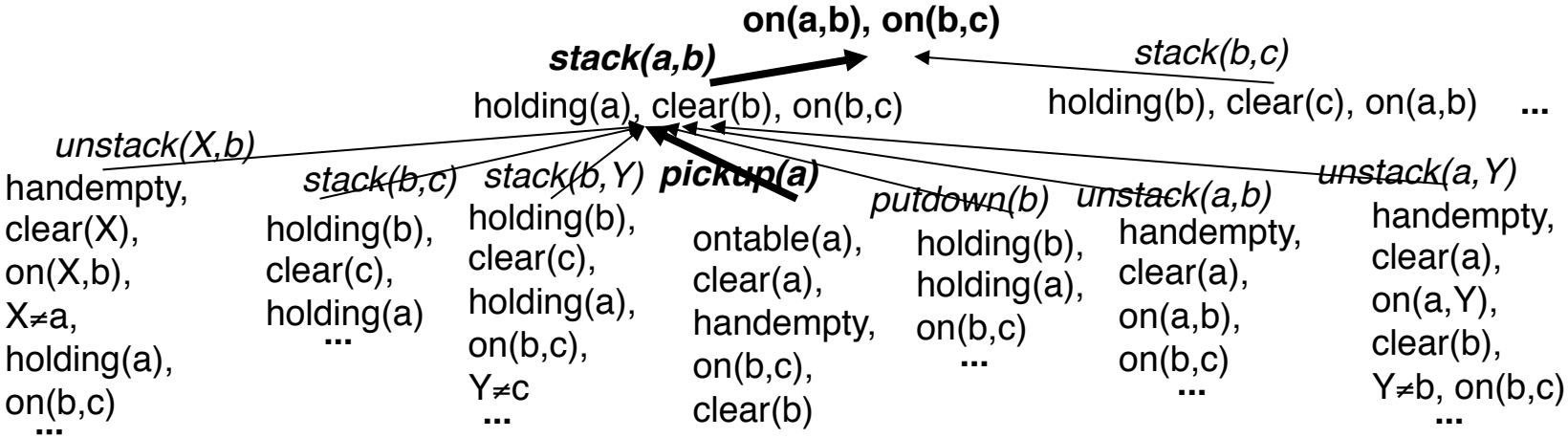
PRECOND: handempty, on(X,Y), clear(X)

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

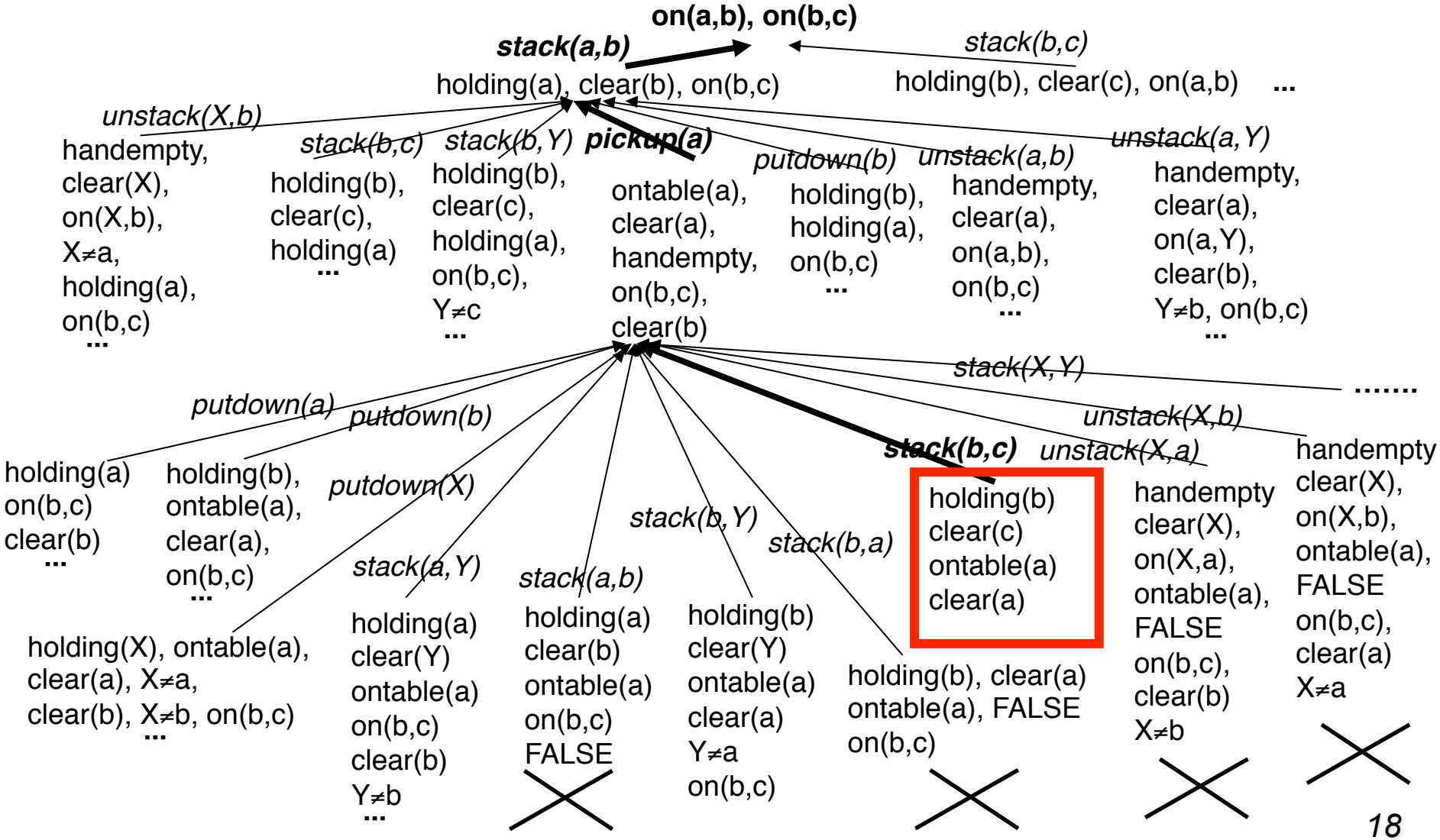
ADD: holding(X), clear(Y)

- Con l'algoritmo backward viene generato il grafo completo mostrato nelle prossime slide

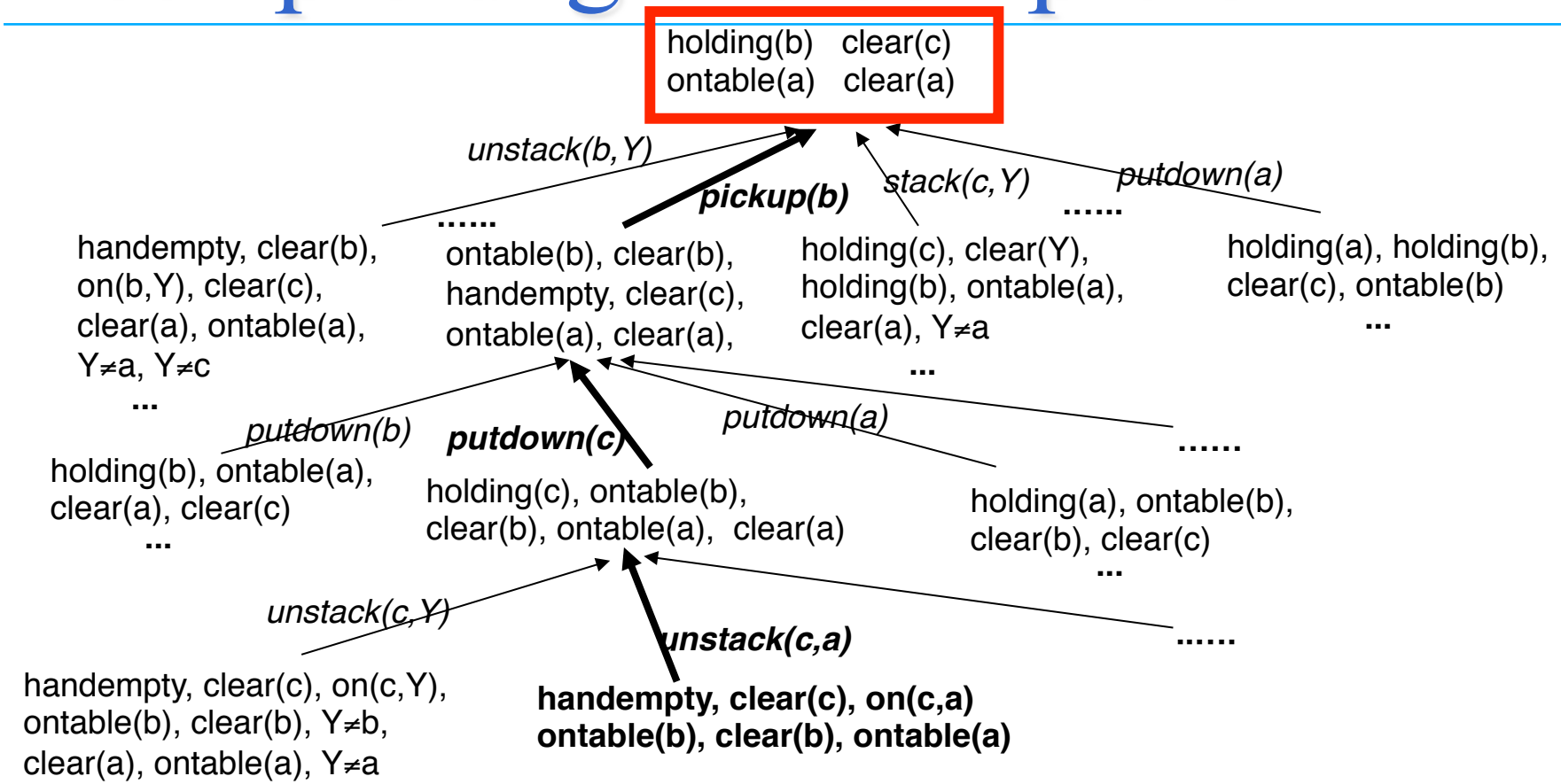
Esempio di grafo completo



Esempio di grafo completo



Esempio di grafo completo



Questo stato coincide con lo stato iniziale

Pianificatori lineari backward

- Occorre una modalità di esplorazione dello spazio di ricerca

- Ad esempio:
  - ricerca in **profondità**;
  - ricerca in **ampiezza**;
  - ricerca **euristica**.

STRIPS

- Acronimo per Stanford Research Institute Problem Solver (STRIPS)
- Antenato degli attuali sistemi di pianificazione (primi anni ’70)
- Ha un proprio linguaggio per la rappresentazione di azioni, con una sintassi molto più semplice del Situation Calculus (meno espressività, ma più efficienza).
- **Planner lineare** basato su ricerca **backward** per la costruzione di piani
- Assume che lo stato iniziale sia completamente noto (**Closed World Assumption**)

Linguaggio STRIPS

- Rappresentazione dello stato
  - Insieme di fluent che valgono nello stato

    Esempio: *on(b,a), clear(b), clear(c), ontable(c)*

    (Non compare lo stato come argomento)
- 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

STRIPS Assumption: tutto ciò che non è specificato nella ADD e DELETE list resta immutato

Non devo scrivere alcun Frame axiom

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)

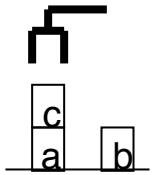
PRECOND: handempty, on(X,Y), clear(X)

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

ADD: holding(X), clear(Y)

Algoritmo STRIPS

- Utilizza due strutture dati:
  - stack di goal / azioni
  - descrizione S dello stato corrente



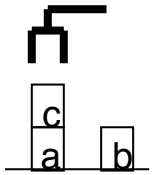
state

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

stack di goal

on(c,b) ∧ on(a,c)

- Funziona backward (goal-oriented)
- Partendo dallo stato obiettivo (goal), lo separa in singoli congiunti se si tratta di un AND, ad esempio:



state

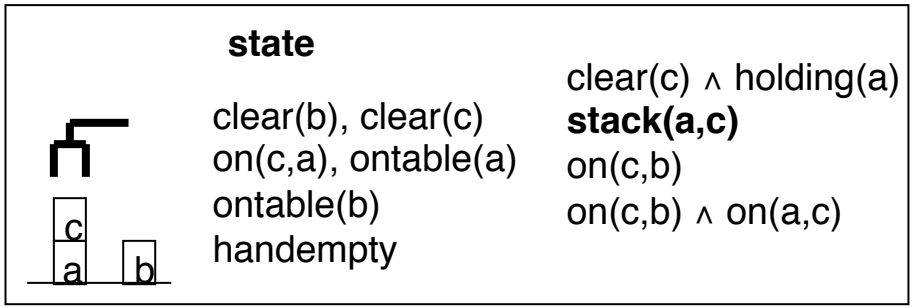
clear(b), clear(c)
on(c,a), ontable(a)
ontable(b), handempty

stack di goal

on(a,c)
on(c,b)
on(c,b) ∧ on(a,c)

Algoritmo STRIPS

- I goal soddisfatti nello stato corrente sono tolti (pop) dallo stack, altrimenti …
- Seleziona una regola che menziona il top goal atomico nella sua ADD list e la inserisce nello stack (con eventuali sostituzioni), insieme alle sue precondizioni da soddisfare:



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

- Quando ***l'elemento affiorante dello stack è un'azione*** (eventualmente istanziata) significa che le sue precondizioni sono state soddisfatte e quindi ***la esegue***, modificando lo stato corrente (*simulando la modifica*)

Algoritmo STRIPS

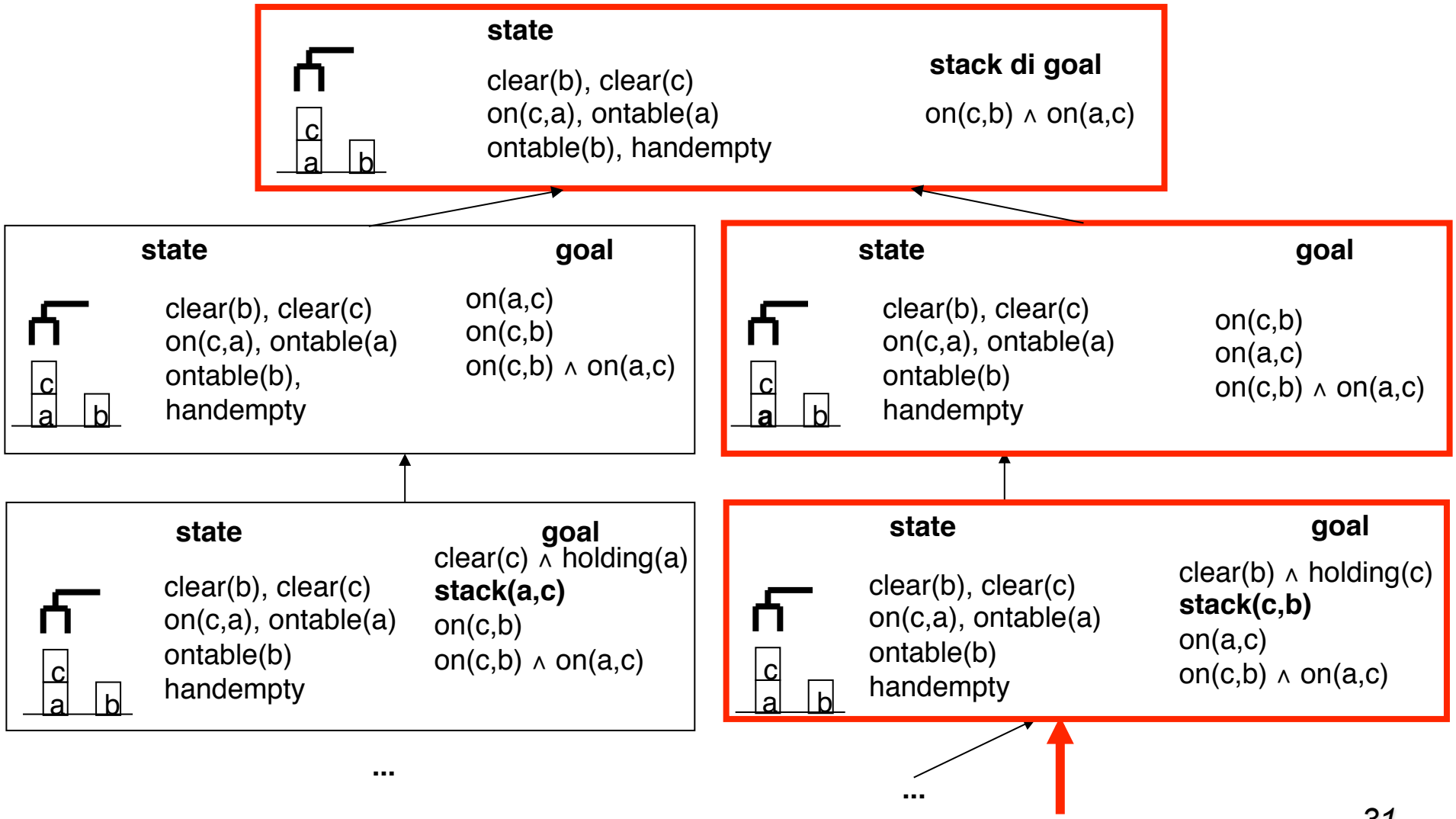
Inizializza **stack** con la congiunzione di goal finali
while **stack** non è vuoto do
    if top(**stack**) = A and Aθ⊆S (si noti che A puo essere un and di goal o un atomo)
    then pop(A) e applica la sostituzione θ sullo **stack**
    else if top(**stack**) = a *% atomico*
        then
            - seleziona regola R con a ∈ Addlist(R),
            - pop(a), push(R), push(Precond(R));
        else if top(**stack**) = a1 ∧ a2 ∧ … ∧ an *% congiunzione*
            (*) then push(a1),…, push(an)
            else if top(**stack**) = R *% azione*
                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 ri-verificata dopo - interacting goals)

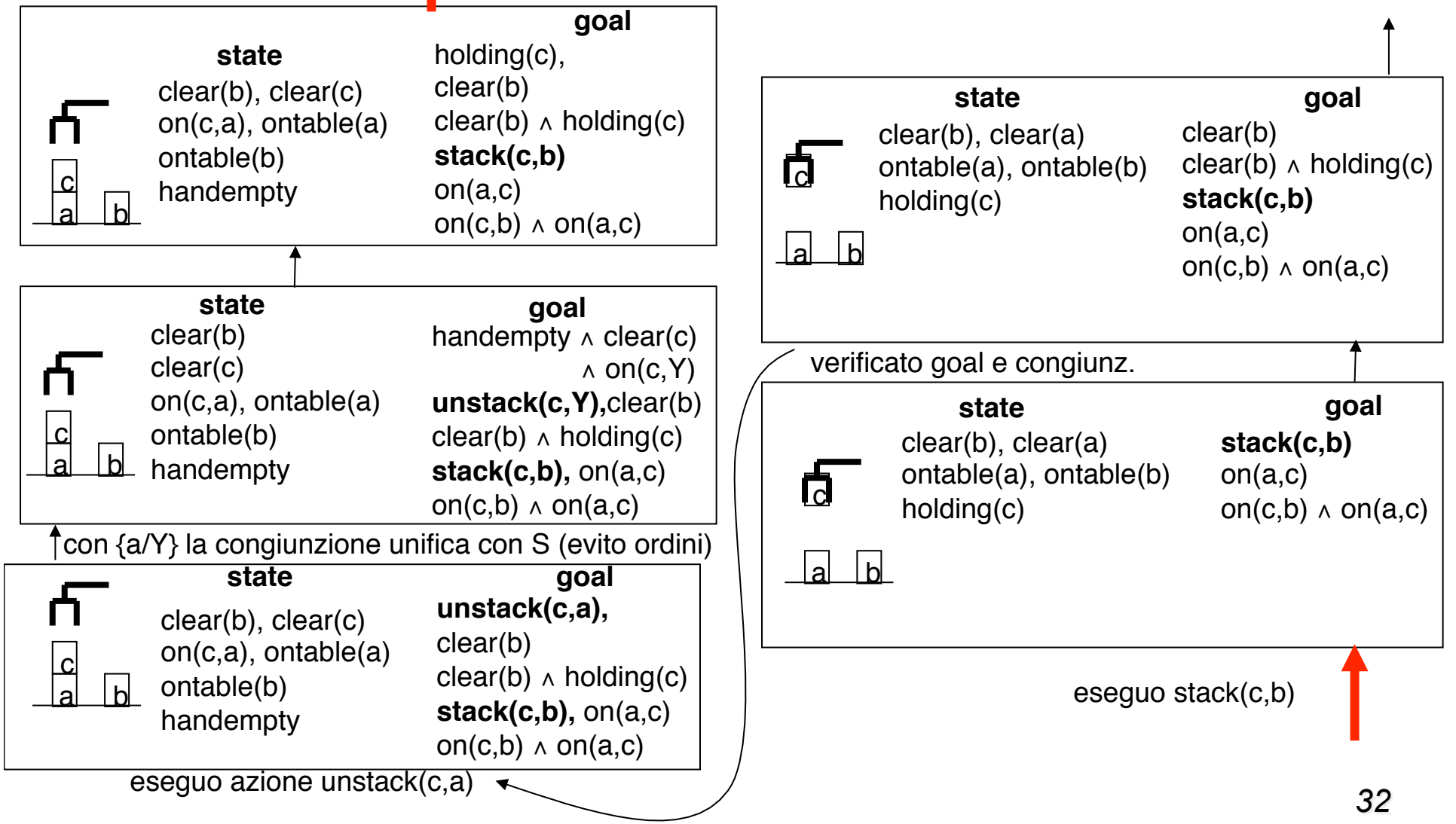
Considerazioni sull'algoritmo

1. Il goal è il contenuto iniziale dello stack (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.

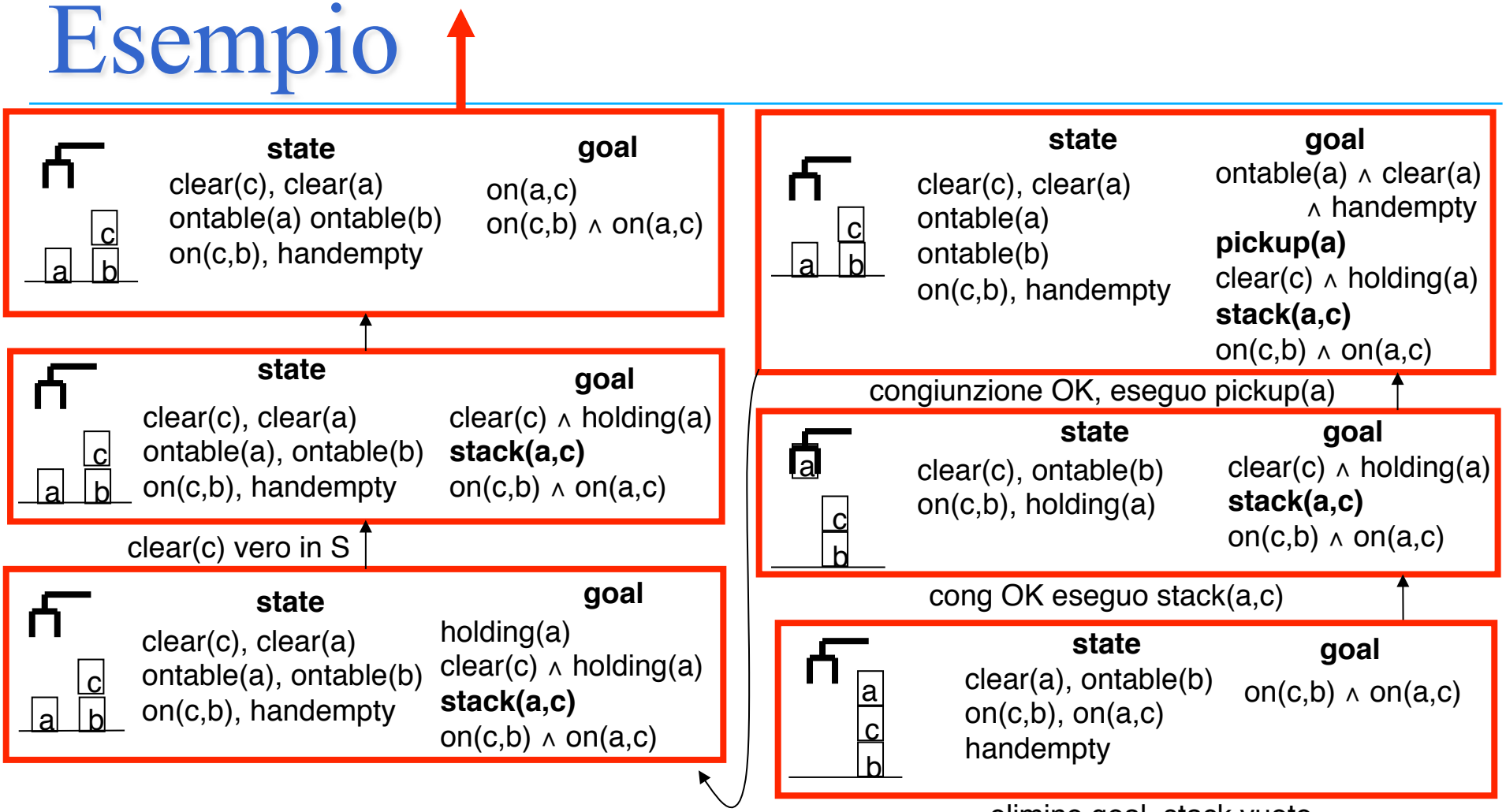
Esempio


state
clear(b), clear(c)
on(c,a), ontable(a)
ontable(b), handempty
stack di goal
on(c,b) ∧ on(a,c)
state
clear(b), clear(c)
on(c,a), ontable(a)
ontable(b),
handempty
goal
on(a,c)
on(c,b)
on(c,b) ∧ on(a,c)
state
clear(b), clear(c)
on(c,a), ontable(a)
ontable(b)
handempty
goal
on(c,b)
on(a,c)
on(c,b) ∧ on(a,c)
state
clear(b), clear(c)
on(c,a), ontable(a)
ontable(b)
handempty
goal
clear(c) ∧ holding(a)
stack(a,c)
on(c,b)
on(c,b) ∧ on(a,c)
state
clear(b), clear(c)
on(c,a), ontable(a)
ontable(b)
handempty
goal
clear(b) ∧ holding(c)
stack(c,b)
on(a,c)
on(c,b) ∧ on(a,c)
...
...

Esempio


state
clear(b), clear(c)
on(c,a), ontable(a)
ontable(b)
handempty
goal
holding(c),
clear(b)
clear(b) ∧ holding(c)
stack(c,b)
on(a,c)
on(c,b) ∧ on(a,c)
state
clear(b)
clear(c)
on(c,a), ontable(a)
ontable(b)
handempty
goal
handempty ∧ clear(c)
∧ on(c,Y)
unstack(c,Y),clear(b)
clear(b) ∧ holding(c)
stack(c,b), on(a,c)
on(c,b) ∧ on(a,c)
con {a/Y} la congiunzione unifica con S (evito ordini)
state
clear(b), clear(c)
on(c,a), ontable(a)
ontable(b)
handempty
goal
unstack(c,a),
clear(b)
clear(b) ∧ holding(c)
stack(c,b), on(a,c)
on(c,b) ∧ on(a,c)
eseguo azione unstack(c,a)
state
clear(b), clear(a)
ontable(a), ontable(b)
holding(c)
goal
stack(c,b)
on(a,c)
on(c,b) ∧ on(a,c)
eseguo stack(c,b)
state
clear(b), clear(a)
ontable(a), ontable(b)
holding(c)
goal
clear(b)
clear(b) ∧ holding(c)
stack(c,b)
on(a,c)
on(c,b) ∧ on(a,c)
verificato goal e congiunz.

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)

Problemi (1)

1. **Grafo di ricerca molto vasto.** Nell'esempio abbiamo visto un cammino, ma in realtà ci sono varie alternative
   - scelte non deterministiche nell'ordinamento dei (sotto)goal
   - più operatori applicabili per ridurre un goal atomico

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 interagiscono 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

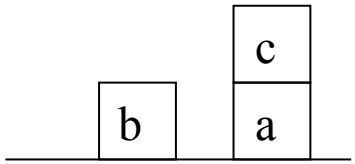
- Soluzione completa:
  - provare tutti gli ordinamenti dei goal e dei loro sottogoal.

- 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 l'esempio precedente):

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

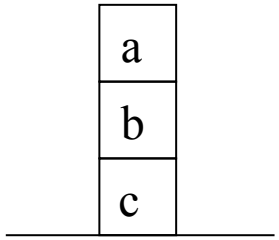


Goal: on(a,b),on(b,c)

Possibili stack iniziali:

(1)
on(a,b)
on(b,c)
on(a,b) ∧ on(b,c)

(2)
on(b,c)
on(a,b)
on(a,b) ∧ on(b,c)

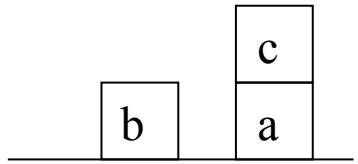


Decidiamo di scegliere (1)

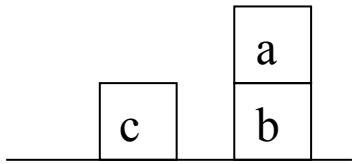
Esempio: Anomalia di Sussmann

Applicando il procedimento di soluzione di STRIPS otteniamo:

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



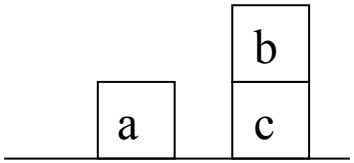
Stato attuale:



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 on(a,b) ∧ on(b,c) non è valida.

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

Otteniamo:

9. pickup(a)
10. stack(a,b)

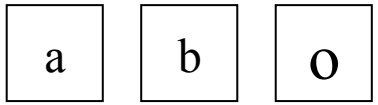
Abbiamo ottenuto quello che volevamo, ma in modo scarsamente efficiente.

Problemi (3)

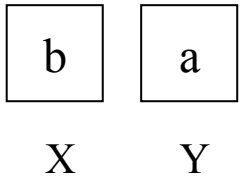
3. Perdita di informazioni:

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)

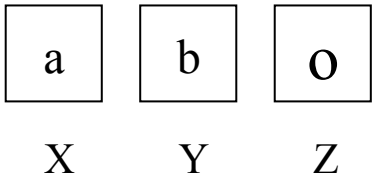
Problemi (3)

Goal stack:

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

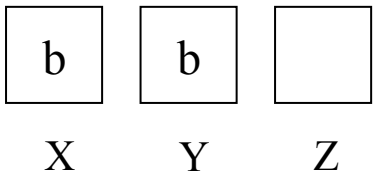
Per ridurre il goal cont(x,b) applica ASSIGN con Y/y e T/a, il goal stack diventa:

cont(y,b) ∧ cont(x,a)
assign(x,y,a,b),
cont(y,a),
cont(x,b) ∧ cont(y,a)



Dopo l'esecuzione di **assign(x,y,a,b)**
lo stato diventa:

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" (eager) di risolvere l'obiettivo e quindi distrugge inesorabilmente il contenuto di una cella di memoria!

Soluzione: 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
  - Si inseriscono vincoli di precedenza tra le azioni solo se strettamente necessari
  - POP (Partial Order Planning)

ADL (Action Description Language)

- Molti linguaggi per il planning … ad esempio, ADL che è più espressivo di STRIPS
- Stati: letterali positivi e negativi negli stati (OWA)
- Azioni
  - Variabili con tipo. es. x: Blocco
  - Predicato di uguaglianza predefinito
  - Effetto "P e ¬Q" significa "aggiungi P e ¬Q" e cancella gli eventuali ¬P e Q
  - Sono permessi effetti condizionali. Cond: effetto
- Obiettivi: ammettono esistenziali, disgiunzioni

  (On(x, a) ∧ Clear(x)) ∨ Clear(a) esiste un x tale che ...

- Linguaggio PDDL come tentativo di stardardizzazione

Planning Domain Definition Language (PDDL)

- E’ un tentativo di standardizzare I linguaggi di descrizione del dominio e delle azioni per la pianificazione automatica

- (include STRIPS, ADL, etc)