# Pianificazione Non Lineare

- Pianificare -> prendere molte decisioni.
- Il pianificatore non conosce a priori l'effetto che ogni decisione avrà sul piano.
- Può essere necessario tornare indietro e prendere decisioni differenti
- Le decisioni su come scomporre un goal in sottogoal vengono prese in maniera esplicita e formalizzate dagli operatori.

**Decisioni da prendere durante la pianificazione:**

- La scelta di quale scomposizione usare per risolvere il problema
- La scelta dell'ordine nel quale risolvere I problemi irrisolti
- La scelta di quali oggetti del dominio instanziare per le azioni scelte per risolvere il problema;
- La scelta dell'ordine nel quale eseguire le azioni del piano



# Pianificazione Non Lineare

- Quando una decisione influenza un'altra decisione diciamo che le decisioni sono **dipendenti** o, nella situazione contraria, sono **indipendenti**.
- Esempio: *voglio bere del vino e non c'e' nessuna bottiglia aperta.Quindi devo decidere di aprire una bottiglia di vino.*
- **Least Commitment Planning**. L'idea è di non prendere decisioni finchè non e' assolutamente indispensabile.

  Se lo faccio prima che sia richiesto dal pianificatore quella potrebbe essere la decisione sbagliata e quindi devo ritirarla e prenderne un'altra. Se la decisione è ritardata finchè non si e' forzati a prenderla allora, presumibilmente, quella decisione dipende da altre già prese. Se la decisione non è forzata, allora è indipendente.
- Il **Least Commitment Planning** non garantisce una soluzione al problema ma può semplificarlo.

## Pianificazione – Ricerca nello spazio dei piani

- Alternativa: **cercare nello spazio dei *piani***, piuttosto che in quello degli stati.
- Si parte da un **piano parziale** che viene espanso e rifinito fino a che non si genera un piano completo che risolve il problema.
- **Operatori di raffinamento:** aggiungono vincoli al piano parziale e lo modificano.
- Operatori STRIPS-style:

  Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)
  Op(ACTION: RightSock, EFFECT: RightSockOn)
  Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)
  Op(ACTION: LeftSock, EFFECT: leftSockOn)

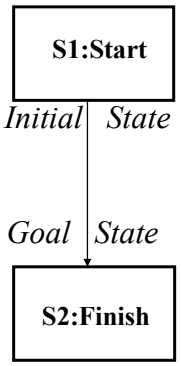


## Piano non Lineare

- Un piano non-lineare è costituito da:
  (1) Un insieme di **steps** $\{S_1, S_2, S_3, S_4 \ldots\}$
  Ogni step corrisponde alla descrizione di un operatore
  (2) Un insieme di **link causali** $\{ \ldots (S_i,C,S_j) \ldots\}$
  Significa che uno step $S_i$ soddisfa una precondizione C per lo step $S_j$
  (3) Un set di **vincoli di ordinamento** $\{ \ldots S_i<S_j \ldots \}$
  Se lo step $S_i$ deve venire, essere eseguito, prima di dello step $S_j$
- Un piano non lineare è **completo** sse
  – Ogni step in (2) e (3) è in (1)
  – Se $S_j$ ha una precondizione C, allora esiste un link causale in (2) della forma $(S_i,C,S_j)$ per qualche step $S_i$
  – Se $(S_i,C,S_j)$ è in (2) e uno step $S_k$ è in (1), e $S_k$ minaccia $(S_i,C,S_j)$ (fa diventare C falso), allora (3) contiene sia $S_k<S_i$ oppure $S_j>S_k$

# Il piano iniziale

Ogni piano inizia così:

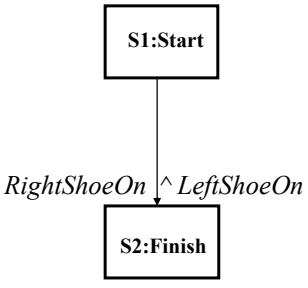





# Un esempio

**Operatori**:

Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)
Op(ACTION: RightSock, EFFECT: RightSockOn)
Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)
Op(ACTION: LeftSock, EFFECT: leftSockOn)

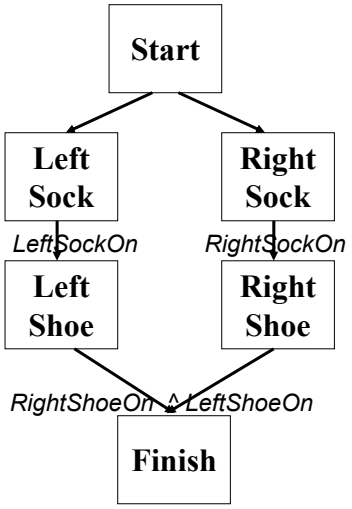



**Steps**: {S1:[Op(Action:Start)],
S2:[Op(Action:Finish,
Pre: RightShoeOn^LeftShoeOn)]}

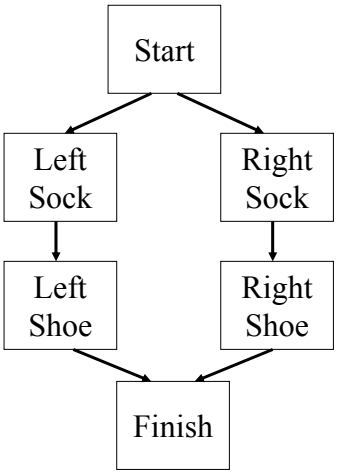
**Links**: {}

**Orderings**: {S1<S2}

# Soluzione


Start
Left Sock
Right Sock
LeftSockOn
RightSockOn
Left Shoe
Right Shoe
RightShoeOn ^ LeftShoeOn
Finish



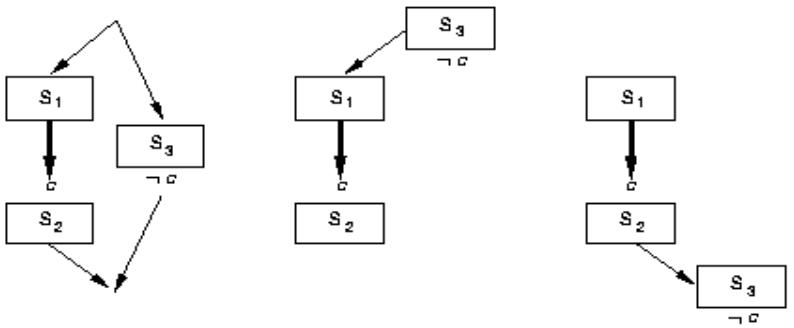

Start
Left Sock
Right Sock
Left Shoe
Right Shoe
Finish

# POP vincoli

- Aggiungere step che aggiungono precondizioni che sono correntemente non soddisfatte.
- Usare l'approccio least-commitment:
  - Non ordinare gli step a meno che non sia necessario ordinare
- Rispettare I link causali $S_1 \xrightarrow{c} S_2$ che **protegge** una condizione *c*:
  - Non aggingere uno step $S_3$ che viola *c*
  - Se una azione parallela **minaccia** *c* (i.e., ha l'effetto di negarlo), risolverlo aggiungendo link di ordinamento:
    - $S_3$ prima di $S_1$ (**demotion**)
    - $S_3$ dopo $S_2$ (**promotion**)



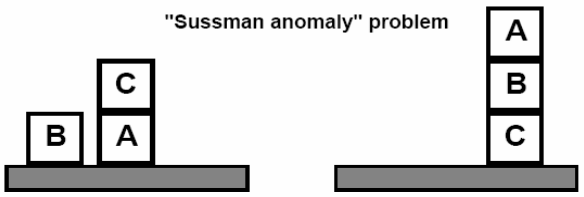
# Risolvere le "minacce"




(a) Threat (b) Demotion (c) Promotion

## Example: Blocks world

"Sussman anomaly" problem




Start State  Goal State

*Clear(x) On(x,z) Clear(y)*

PutOn(x,y)

*~On(x,z) ~Clear(y)*
*Clear(z) On(x,y)*

*Clear(x) On(x,z)*

PutOnTable(x)

*~On(x,z) Clear(z) On(x,Table)*

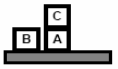
+ several inequality constraints



## Example contd.

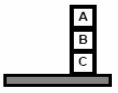
START

*On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)*

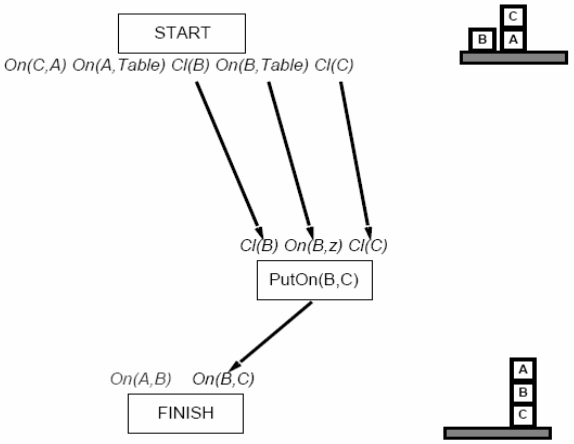



*On(A,B) On(B,C)*

FINISH

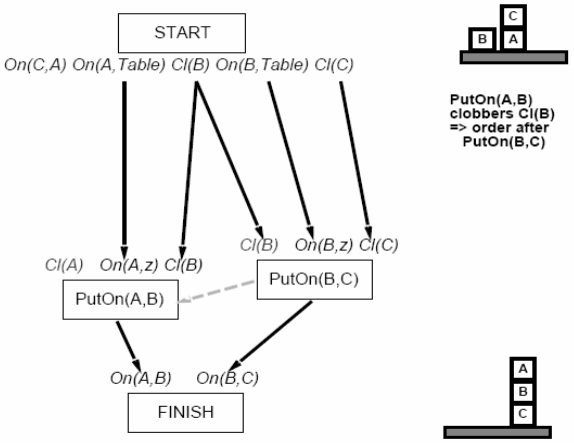


## Example contd.

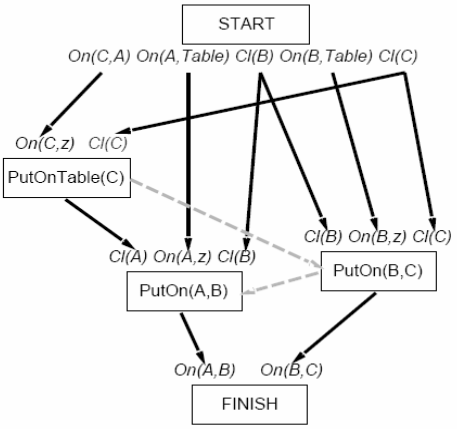


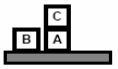



## Example contd.



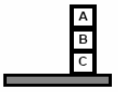
## Example contd.




PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)

PutOn(B,C)
clobbers Cl(C)
=> order after
PutOnTable(C)






# Algoritmo POP

**function** POP(*initial, goal, operators*) **returns** *plan*

*plan* ← MAKE-MINIMAL-PLAN(*initial, goal*)
**loop do**
  **if** SOLUTION?( *plan*) **then return** *plan*
  $S_{need}$, $c$ ← SELECT-SUBGOAL( *plan*)
  CHOOSE-OPERATOR( *plan, operators*, $S_{need}$, $c$)
  RESOLVE-THREATS( *plan*)
**end**

---

**function** SELECT-SUBGOAL( *plan*) **returns** $S_{need}$, $c$

pick a plan step $S_{need}$ from STEPS( *plan*)
  with a precondition $c$ that has not been achieved
**return** $S_{need}$, $c$

# Algoritmo POP

**procedure** CHOOSE-OPERATOR($plan$, $operators$, $S_{need}$, $c$)

  **choose** a step $S_{add}$ from $operators$ or STEPS($plan$) that has $c$ as an effect
  **if** there is no such step **then fail**
  add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS($plan$)
  add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS($plan$)
  **if** $S_{add}$ is a newly added step from $operators$ **then**
    add $S_{add}$ to STEPS($plan$)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS($plan$)

---

**procedure** RESOLVE-THREATS($plan$)

  **for each** $S_{threat}$ that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS($plan$) **do**
    **choose** either
      *Demotion:* Add $S_{threat} \prec S_i$ to ORDERINGS($plan$)
      *Promotion:* Add $S_j \prec S_{threat}$ to ORDERINGS($plan$)
    **if not** CONSISTENT($plan$) **then fail**
  **end**

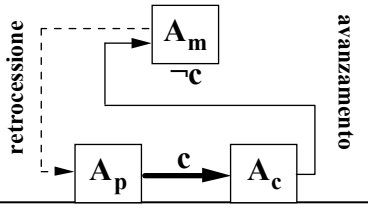


# Proprietà di POP

- Algoritmo nondeterministico: in caso di fallimento effettua backtracking sui punti di scelta (choice point)
  - scelta di $S_{add}$ per raggiungere $S_{need}$
  - scelta di demotion o promotion in caso di minaccia
  - la selezione di $S_{need}$ è irrevocabile
- POP è corretto, completo e sistematico (nessuna ripetizione)
- Esistono estensioni (es. azioni rappresentate con FOL)
- Efficiente se fornito di euristiche derivate dalla descrizione del problema

# Operazioni sui piani

- aggiunta di un legame causale da un'azione esistente ad una condizione aperta
- aggiunta di un'azione per soddisfare una condizione aperta
- ordinamento di un'azione rispetto ad un'altra per annullare una minaccia:
  - retrocessione ($A_m < A_p$)
  - avanzamento ($A_m > A_c$)

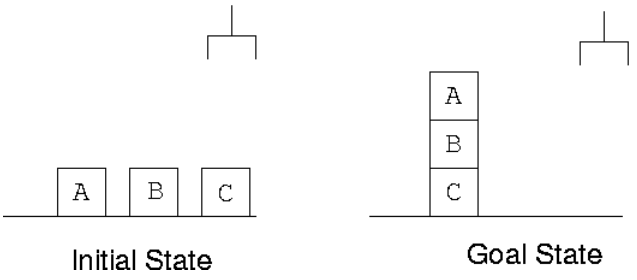





# Pianificare per livelli di astrazione

- **Gestire la complessità**
- Strutturare e distinguere fra:
  - Proprietà più o meno importanti
  - Operatori più o meno importanti
- Due modi di fare "**astrazione"** nella pianificazione:
- **Astrazione dello stato o situazione - abstrips (Abstraction-Based Strips )**
- Idea Principale: introdurre dei **pesi** per ogni termine e considerare solo quelli più importanti a quel livello di astrazione, raffinare considerando i termini con il secondo peso più alto.
- MONDO dei BLOCCHI: Pesi dei predicati:
  On 4 Clear 3 Holds 2 Ontable 2 Handempty 1
- Pesi più alti indicano proprietà più importanti.
- **Operatori del mondo dei blocchi**
  - PICKUP(x)
    **preconditions** Clear 3(x) Ontable 2(x) Handempty 1
    **delete list** Clear(x) Ontable(x) Handempty **add list** Holds(x)
  - PUTDOWN(x)
    **preconditions** Holds 2(x)
    **delete list** Holds(x) **add list** Clear(x) Ontable(x) Handempty
  - STACK(x,y)
    **preconditions** Holds 2(x) Clear 3(y)
    **delete list** Holds(x) Clear(y) **add list** Clear(x) On(x,y) Handempty
  - UNSTACK(x,y)
    **preconditions** Clear 3(x) On 4(x,y) Handempty 1
    **delete list** Clear(x) On(x,y) Handempty **add list** Holds(x) Clear(y)

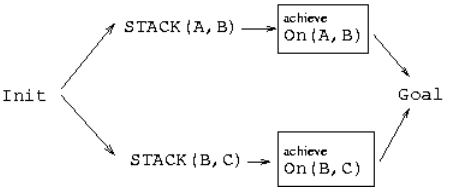
# Pianificare per livelli di astrazione



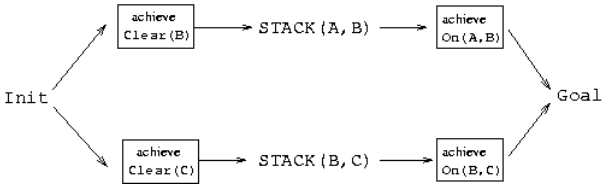



# Pianificare per livelli di astrazione

peso 4 (On):

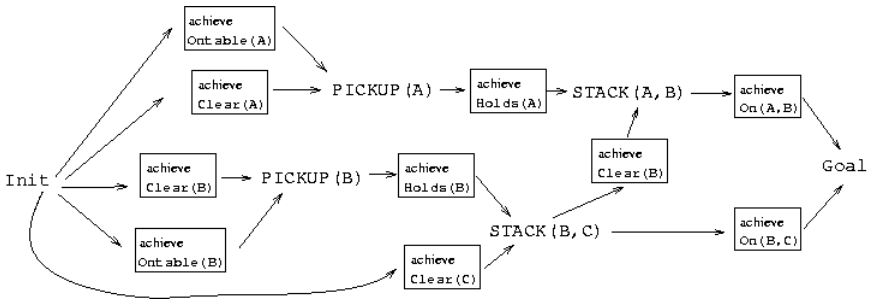



peso 3(On e Clear):



# Pianificare per livelli di astrazione

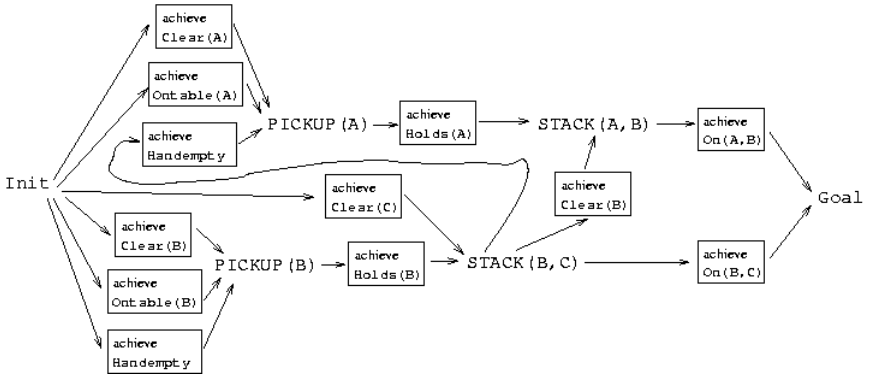
peso 2 (Holds e OnTable):






# Pianificare per livelli di astrazione

peso 1 (Handempty):



# Astrazione degli operatori

Supponiamo che
PICKUP, PUTDOWN, STACK, e UNSTACK siano operatori "astratti"
Supponiamo che una descrizione più dettagliata di PICKUP lo divida in tre parti:
•POSITION per posizionare la mano del robot
•CATCH per prendere il blocco
•LIFT per sollevarlo.
E quindi bisogna specificare questi operatori elementari
PICKUP può essere visto come una **astrazione di operatore** di POSITION, CATCH, e LIFT

# Esempio

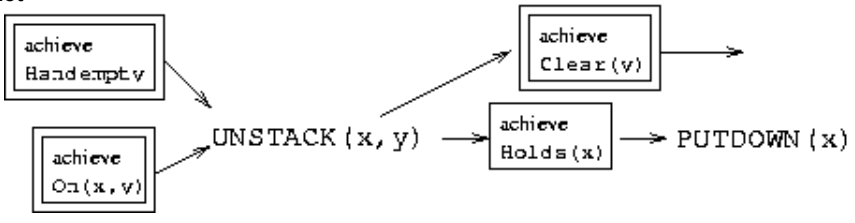
Definiamo l'operatore FREE come una astrazione di UNSTACK e PUTDOWN
FREE(x,y)
**preconditions** Handempty On(x,y)
**delete list** On(x,y) **add list** Clear(y)
**plot** (UNSTACK(x,y),PUTDOWN(x))
Differenza: dopo l'applicazione **non tutte le precondizioni vengono cancellate**.
Handempty è vera anche dopo l'esecuzione
**plot** -> scomposizione.



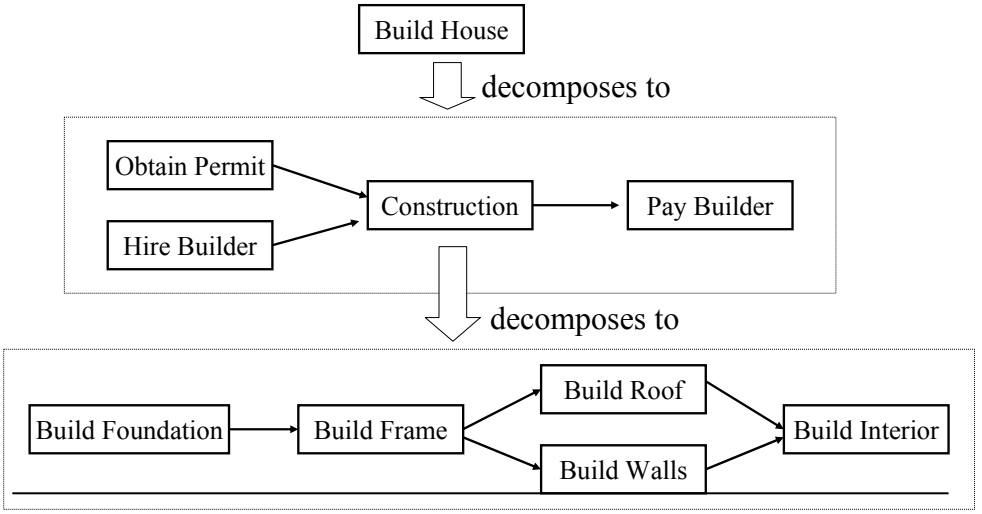
FREE(x,y)

| | |
|---|---|
| **preconditions** | Handempty |
| | On(x,y) |
| **delete list** | On(x,y) |
| **add list** | Clear(y) |

**Plot**



# Building a Frame House






# HD-POP

Operatori Primitivi Non-Primitivi:

Hammer(Nail), Build(House)

...

Un insieme di metodi di scomposizione:

Decompose(o,p): operatore n.p. -> o può essere scomposto in un piano p

*Decompose (Construction,*

*Plan(Passi: {S1: Build(Foundation), S2: Build(Frame),*

*S3: Build(Roof), S4: Build(Walls): S5: Build(Interior)}*

*Ordinamento: {S1<S2<S3<S5, S2<S4<S5}*

*Vincoli: {}*

*LINKS {S1→S2, S2 →S3, S2 →S4, S3 →S5, S4 →S5}))*

# HD-POP: Estensione dell'algoritmo POP

Function HD-POP(plan, operators, methods) returns plan

Inputs: plan: an abstract plan with start and goal steps ( and possibly other steps)

Loop do

If SOLUTION?(plan) then return plan

$S_{need}$,c ← SELECT-SUB-GOAL(plan)

CHOOSE-OP(plan, operators, $S_{need}$,c)

$S_{nonprim}$ ← SELECT-NONPRIM(plan)

CHOOSE-DECOMP(plan, methods, $S_{nonprim}$)

RESOLVE-THREATS(plan)

End



# HD-POP: Esempio