Planning in STRIPS

- STRIPS mantiene due strutture dati:
  - **Lista di Stati** – tutti i predicati correntemente veri.
  - **Pila di Goal** – una pila di goal da risolvere, con il goal corrente in testa alla pila.
- Se il goal corrente non è soddisfatto dallo stato presente, esamina gli effetti positivi degli operatori, e inserisce l'operatore e la lista delle precondizioni sulla pila. (Subgoal)
- Quando il goal corrente è soddisfatto, lo rimuove dalla pila.
- Quando un operatore è in testa alla pila, registra l'applicazione dell'operatore sulla sequenza del piano e usa gli effetti per aggiornare lo stato corrente.

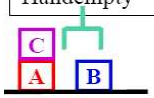
STRIPS Algorithm

- STRIPS (*initial-state, goals*)
  - *state* = *initial-state*; *plan* = []; *stack* = []
  - Push *goals* on *stack*
  - Repeat until *stack* is empty
    - If top of *stack* is **goal** that matches *state*, then pop *stack*
    - Else if top of stack is a **conjunctive goal** *g*, then
      - **Select** an ordering for the subgoals of *g*, and push them on *stack*
    - Else if top of *stack* is a **simple goal** *sg*, then
      - **Choose** an operator *o* whose add-list matches goal *sg*
      - Replace goal *sg* with operator *o*
      - Push the preconditions of *o* on the *stack*
    - Else if top of *stack* is an **operator** *o*, then
      - *state* = apply(*o*, *state*)
      - *plan* = [*plan*; *o*]

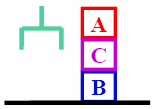
fonte: R. Simmons

Applicazione di STRIPS

Clear(B)
Clear(C)
On(C, A)
On(A, Table)
On(B, Table)
Handempty



Initial State



Goal

fonte: R. Simmons

Piani subottimi

- Consideriamo le seguenti azioni:

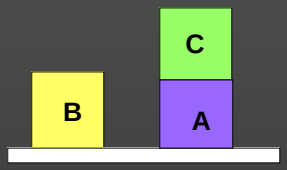
Op[Action: Load(obj,plane,loc),
Precondizioni: At(obj,loc) ∧ At(plane,loc),
Effetto: Inside(obj,plane) ∧ ¬ At(obj,loc)]

Op[Action: Unload(obj,plane,loc),
Precondizioni: Inside(obj,plane) ∧ At(plane,loc),
Effetto: At(obj,loc) ∧ ¬ Inside(obj,plane)]

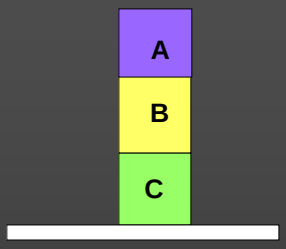
Op[Action: Fly(plane,from,to),
Precondizioni: At(plane,from),
Effetto: At(plane,to) ∧ ¬ At(plane,from)]

- Stato iniziale: **At(obj1,locA), At(obj2,locA), At(747,locA)**
- Goal: **At(obj1,locB), At(obj2,locB)**
- Piano: [**Load(obj1,747,locA); Fly(747,locA,locB); Unload(obj1,747,locB);**
Fly(747,locB,locA); Load(obj2,747,locA); Fly(747,locA,locB); Unload(obj2,747,locB)]

Esercizio



Stato iniziale



goal

Problemi non risolvibili

- Dovuti alla linearità e ad azioni irreversibili:

Op[Action: Load(obj,plane,loc),
Precondizioni: At(obj,loc) ∧ At(plane,loc),
Effetto: Inside(obj,plane) ∧ ¬ At(obj,loc)]

Op[Action: Unload(obj,plane,loc),
Precondizioni: Inside(obj,plane) ∧ At(plane,loc),
Effetto: At(obj,loc) ∧ ¬ Inside(obj,plane)]

Op[Action: Fly(plane,from,to),
Precondizioni: At(plane,from) ∧ **Have-fuel(plane)**,
Effetto: At(plane,to) ∧ ¬ At(plane,from)] ∧ ¬ **Have-fuel(plane)**

- Stato iniziale: **At(obj1,locA), At(obj2,locA), At(747,locA), Have-fuel(747)**
- Goal: **At(obj1,locB), At(obj2,locB)**

Problemi non risolvibili

- Tentiamo di risolvere prima il sottogoal **At(obj1,locB)**
  - [**Load(obj1,747,locA); Fly(747,locA,locB); Unload(obj1,747,locB)**]
  - ma non riusciamo a raggiungere **At(obj2,locB)** perché è finito il carburante!
- Tentiamo di risolvere prima il sottogoal **At(obj2,locB)**
  - [**Load(obj2,747,locA); Fly(747,locA,locB); Unload(obj2,747,locB)**]
  - ma non riusciamo a raggiungere **At(obj1,locB)** perché è finito il carburante!

In ogni caso STRIPS non è in grado di risolvere il problema !

Planning nello spazio degli stati: riassunto

- Spazio delle situazioni (localizzazione, possedimenti, etc.)
- Il piano è una soluzione trovata “cercando” tra le situazioni il goal
- Un **planner progressivo** cerca il goal in avanti (forward) a partire dallo stato iniziale
- Un **planner regressivo** cerca all’indietro (backward) a partire dal goal
- Attenzione: problema della Anomalia di Sussman

Planning nello spazio dei piani

- Una alternativa è la **ricerca attraverso lo spazio dei *piani***, piuttosto che delle situazioni.
- Si parte da un **piano parziale** che viene espanso e raffinato fino a raggiungere un piano completo che risolve il problema.
- **Operatori di raffinamento** aggiungono vincoli a piani parziali e operatori di modifica effettuano altri cambiamenti.
- Operatori alla STRIPS:

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

  possono risultare in un piano parziale del goal

  [RightShoe, LeftShoe]

Partial-order planning (POP)

- Un **planner lineare** costruisce un piano come una **sequenza totalmente ordinata** di passi
- Un **planner non-lineare (aka partial-order planner)** costruisce un piano come un insieme di passi con alcuni vincoli temporali
- Vincoli della forma S1<S2 se il passo S1 deve venire prima di S2.
- Si **raffina** un piano ordinato parzialmente (POP) per mezzo di:
  - **Aggiunta di un nuovo passo al piano**, o
  - **Aggiunta di un nuovo vincolo** ai passi già presenti nel piano.
- Un POP può essere **linearizzato** (convertito in un piano totalmente ordinato) attraverso un ordinamento topologico

Minimo Impegno

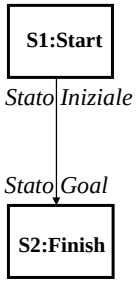
- I planner non-lineari incorporano il principio del **minimo impegno (least commitment)**
  - Si scelgono solo quelle azioni, ordinamenti, e assegnamenti di variabili che sono assolutamente necessari, lasciando le altre decisioni al futuro
  - Evita di prendere decisioni premature su aspetti che non contano
- Un planner lineare sceglie sempre di aggiungere un passo in un punto preciso della sequenza
- Un planner non-lineare sceglie di aggiungere un passo ed eventualmente qualche vincolo temporale fra passi

Piano Non-lineare

- Consiste di
  (1) Un insieme di **passi** $\{S_1, S_2, S_3, S_4 \ldots\}$
  Ogni passo ha la descrizione di un operatore, precondizioni e post-condizioni
  (2) Un insieme di **link causali** $\{ \ldots (S_i,C,S_j) \ldots\}$
  Che significa che uno dei propositi del passo S_i è di raggiungere la precondizione C del passo S_j
  (3) Un insieme di **vincoli di ordinamento** $\{ \ldots S_i<S_j \ldots \}$
  Nel caso in cui il passo S_i deve venire prima del passo S_j
- Un piano non-lineare è **completo** sse
  - Ogni passo menzionato in (2) e (3) è in (1)
  - Se S_j ha prerequisito C, allora esiste un link causale in (2) nella forma (S_i,C,S_j) per qualche S_i
  - Se (S_i,C,S_j) è in (2) e il passo S_k è in (1), e S_k "minaccia" (S_i,C,S_j) (rende C falso), allora (3) contiene $S_k<S_i$ o $S_k>S_j$

Il Piano Iniziale

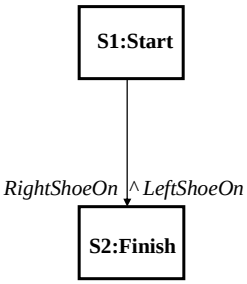
Ogni piano inizia nello stesso modo



Esempio Banale

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)

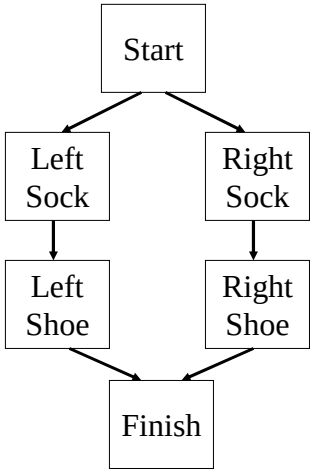


Passi: {S1:[Op(Action:Start)],
S2:[Op(Action:Finish,
Pre: RightShoeOn^LeftShoeOn)]}

Link: {}

Ordinamenti: {S1<S2}

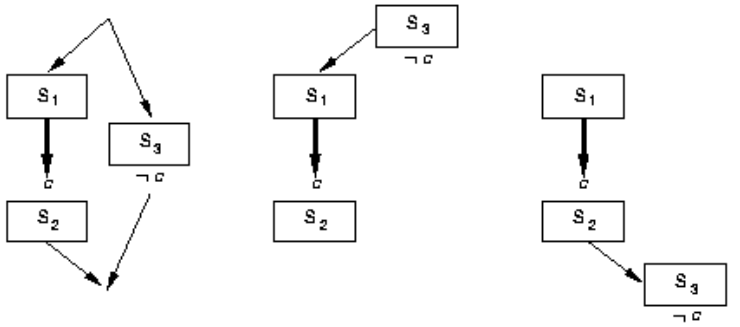
Soluzione



POP: vincoli ed euristiche

- Aggiungere solo passi che raggiungono una precondizione correntemente non raggiunta
- Usare un approccio a minimo-impegno:
  - Non ordinare passi a meno che non sia strettamente necessario
- Onorare link causali $S_1 \xrightarrow{c} S_2$ che **proteggono** una condizione *c*:
  - Non aggiungere mai un passo intermedio S_3 che viola *c*
  - Se una azione parallela minaccia (**threatens**) *c* (cioè, ha l'effetto di negare (in gergo, **clobbering**) *c*, risolvere la minaccia aggiungendo vincoli temporali:
    - Ordinare S_3 prima di S_1 (**demotion**), oppure
    - Ordinare S_3 dopo S_2 (**promotion**)

Risoluzione delle Minacce



(a) Minaccia (b) Demotion (c) Promotion