

Pianificazione

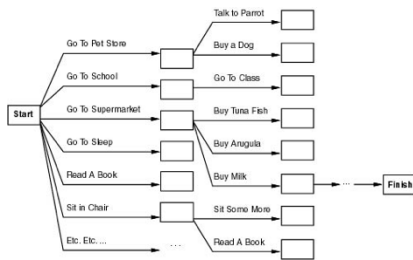
Maria Simi, 2011-2012
AIMA – Cap 10

Il filo conduttore [AIMA]

- *Agente per il problem solving*: agente che decide la sequenza di azioni prima di agire (ambiente accessibile)
- *Agente basato su conoscenza*: mantiene una rappresentazione esplicita degli stati e dell'effetto delle azioni (ambiente complesso e parzialmente accessibile)
- *Agente pianificatore*: è un agente che pianifica utilizzando conoscenza esplicita delle azioni e del loro effetto e quindi è anche un agente KB

Un esempio

Goal: Possedere del latte, delle banane e un trapano



Pianificazione vs problem solving

- Un *PS agent*
 - *Azioni*: capacità di generare i successori di uno stato
 - *Obiettivi*: goal test (la rappresentazione del goal è implicita)
 - *Pianificazione*: ottenere, mediante un processo di ricerca euristica, una sequenza di azioni che portano dallo stato iniziale allo stato goal
- Un *agente pianificatore*
 - ha una rappresentazione esplicita dell'obiettivo, delle azioni e loro effetto
 - può decomporre il goal in sotto-goal indipendenti
 - ha libertà nella costruzione del piano
- Un *agente pianificatore* può essere più efficiente

Di cosa parleremo

- Pianificazione come dimostrazione di teoremi
 - Il calcolo di situazioni
 - Il *frame problem*
 - Problemi di rappresentazione collegati e inefficienza
- Pianificazione con algoritmi specializzati
 - Rappresentazione delle azioni in PDDL (Planning Domain Definition Language)
 - Pianificazione come ricerca in uno spazio di stati
 - Pianificazione a progressione e regressione
 - Grafi di pianificazione
 - Pianificazione come ricerca in uno spazio di piani

Il caso proposizionale

- Problemi di rappresentazione nel mondo del Wumpus
 - Proprietà immutabili (atemporali)
$$B_{1,1} \Leftrightarrow P_{1,2} \vee P_{2,1}$$
 - Proprietà dipendenti dal tempo (fluenti)
$$L^0_{1,1} \wedge \text{HaFreccia}^0 \wedge \text{RivoltoEst}^0 \wedge \text{Brezza}^2 \wedge \text{Avanti}^0$$
 - Descrivere come cambia il mondo (transizioni):
$$L^0_{1,1} \wedge \text{RivoltoEst}^0 \wedge \text{Avanti}^0 \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,1})$$

Non basta ...

- Non solo gli effetti ma anche in non effetti!!!
- Bisogna assicurarsi anche di assiomatizzare ...
 - il fatto che l'agente non può essere in due locazioni diverse o abbia due orientamenti diversi allo stesso tempo
 - criteri affinché le azioni siano possibili
 - mutua esclusione di azioni: per evitare che vengano svolte due azioni allo stesso tempo (almeno quando esse sono incompatibili) $\neg A_i \vee \neg A_j$

Pianificazione in Prop

- Soluzione ibrida
 - La formalizzazione logica usata per inferire lo stato nel mondo in un certo istante (es. se locazione OK)
 - La KB viene aggiornata ad ogni azione
- Impostare come problema SAT
 - Init^0 tutto ciò che è vero al tempo 0
 - tutte le transizioni: *transizioneⁱ per $i \leq t$*
 - l'asserzione goal: $\text{HaOro}^t \wedge \text{Uscito}^t$
 - se trova un modello abbiamo il piano ... o quasi

SATPLAN

- Non sappiamo quanto è lungo il piano
- SATPLAN prova con valori crescenti di t
- Garantisce di trovare il piano di lunghezza minima

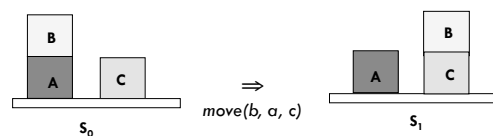
Il calcolo di situazioni in FOL

- Una particolare ontologia che tratta di azioni e cambiamento:
 - Situazioni: fotografie del mondo in un certo istante, stato risultante da una azione (vs istanti di tempo)
 - Proprietà dipendenti dalla situazione (fluenti)
 - Azioni
 - Cambiamento per effetto di azioni
- Il calcolo di situazioni è una formalizzazione in FOL di questa ontologia [McCarthy, 69]

Calcolo di situazioni nel mondo dei blocchi

- Situazioni: $s, s_0, s_1, s_2 \dots$ e funzioni che denotano situazioni
- Proprietà o funzioni che variano da una situazione all'altra (fluenti): *On, Table, Clear ... Hat*
 - $\text{On}(a, b)$ diventa $\text{On}(a, b, s)$
 - $\text{Hat}(a)$ diventa $\text{Hat}(a, s)$
 - Le proprietà immutabili vengono rappresentate come prima
- Azioni: sono modellate come funzioni (termini)
 - $\text{move}(a, b, c)$ è una funzione che denota l'azione di spostare il blocco A da B a C (istanza dell'operatore generico *move*)
- Sequenze di azioni

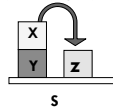
Situazioni risultanti dalle azioni



- Effetto delle azioni: funzione Risultato: $A \times S \rightarrow S$
 - $\text{Risultato}(\text{move}(a, b, c), s_0)$ denota la situazione risultante dall'azione $\text{move}(a, b, c)$ compiuta in s_0
- Effetto di sequenze di azioni: RisultatoSeq: $[A^*] \times S \rightarrow S$
 - $\text{RisultatoSeq}([], s) = s$
 - $\text{RisultatoSeq}([a | \text{seq}], s) = \text{RisultatoSeq}(\text{seq}, \text{Risultato}(a, s))$

Assiomi per l'effetto delle azioni

- Sono del tipo: *precondizioni* \Rightarrow *effetto*
 $\forall x \forall y \forall z \forall s \text{On}(x, y, s) \wedge \text{Clear}(x, s) \wedge \text{Clear}(z, s) \wedge x \neq z \Rightarrow$
 $\text{On}(x, z, \text{Risultato}(\text{Move}(x, y, z), s)) \wedge \text{Clear}(y, \text{Risultato}(\text{Move}(x, y, z), s))$
- Altri effetti collaterali, ad esempio
 $\neg \text{Clear}(z, \text{Risultato}(\text{Move}(x, y, z), s))$
 sono ricavabili da vincoli di stato del tipo
 $\forall x \forall s \text{Clear}(x, s) \Leftrightarrow \neg \exists y \text{On}(y, x, s)$
 "Un blocco è libero sse non c'è niente sopra"
- Ma non basta! Nel nuovo stato ...
 y è o no sul tavolo? x è o no libero?



C'è un problema ...

- Nello stato risultante non si sa niente delle cose che *non* cambiano, che erano vere (o false) prima dell'azione e che rimangono vere (o false) perché non influenzate.
 E queste sono la maggioranza!
- Esempi
 - Se $\text{Table}(a, s)$ non si deriva $\text{Table}(a, \text{Risultato}(\text{move}(b, a, c), s))$ eppure si sposta solo b!
 - Se l'agente si sposta nel mondo del Wumpus la sua locazione cambia (effetto primario) ma nel nuovo stato non so se il Wumpus è ancora vivo, se l'agente ha ancora la sua freccia ...

Il frame problem

- Il problema del contorno (*frame problem*): uno dei problemi più classici dell'AI [McCarthy-Hayes, 1969]
- Analogia col mondo dell'animazione: il *frame problem* è il problema di distinguere il *background* (che rimane fisso) dal *foreground* (ciò che cambia).
- Dobbiamo introdurre tutta una serie di assiomi solo per dire ciò che non cambia

Assiomi frame

- Clear* rispetto a *move*:
 $\text{Clear}(x, s) \wedge x \neq w \Rightarrow \text{Clear}(x, \text{Risultato}(\text{Move}(y, z, w), s))$
Un blocco rimane libero se l'azione non ci sposta qualcosa sopra
 $\neg \text{Clear}(x, s) \wedge x \neq z \Rightarrow \neg \text{Clear}(x, \text{Risultato}(\text{Move}(y, z, w), s))$
Un blocco rimane occupato a meno che non sia liberato dall'azione
- In maniera analoga per ogni coppia fluente-azione
 F fluenti e A azioni $\Rightarrow O(AF)$ assiomi
 Troppi!!!
- Problema di rappresentazione del *frame*

Assioma di stato successore

- Si possono anche scrivere in maniera più compatta in un modo che combina *effetto* e *frame axiom*, tramite un assioma di *stato-successore*:
Precondizioni \Rightarrow *Azione possibile*
Azione possibile \Rightarrow
 $[\text{Fluente vero dopo} \Leftrightarrow [\text{un'azione possibile l'ha reso vero oppure era vero e nessun azione possibile l'ha reso falso}]]$
- Esempio:
 $\text{Clear}(x, \text{Risultato}(a, s)) \Leftrightarrow$
 $[(a = \text{Move}(y, x, z)) \vee (a = \text{Unstack}(y, x)) \vee (a = \text{Noop})] \vee$
 $[\text{Clear}(x, s) \wedge (a \neq \text{Move}(z, w, x)) \wedge (a \neq \text{Stack}(z, x))]$
- Migliori: due assiomi per ogni fluente ... ma un problema più sostanziale è la complessità computazionale

Approccio non-monotono al frame problem

- L'idea: implementare una nozione di *persistenza*: in assenza di informazione in contrario (per *default*) le cose rimangono come erano.
- Un unico *frame axiom* con una rappresentazione reificata (di *meta-livello*):
 $\forall p, a, s \text{True}(p, s) \wedge \neg \text{Ab}(p, a, s) \Rightarrow \text{True}(p, \text{Do}(a, s))$
 - Ab sta per 'Abnormal'
 - $\neg \text{Ab}(p, a, s)$ vale a meno che $\text{Ab}(p, a, s)$ non sia derivabile
- La teoria dovrà avere gli assiomi necessari per derivare tutte le anomalie, ad esempio:
 $\text{Ab}(\text{Clear}(x), \text{Do}(\text{Stack}(y, x), s))$

Problemi correlati

- Il problema della *qualificazione*: in casi reali non è semplice elencare tutte le precondizioni rilevanti:

$$\text{Clear}(x) \wedge \text{Clear}(y) \wedge \text{Clear}(z) \wedge y \neq z \wedge \neg \text{Pesante}(x) \wedge \neg \text{Incollato}(x) \wedge \neg \text{Bollente}(x) \wedge \dots \Rightarrow \text{possibile move}(x, y, z)$$
- Il problema della *ramificazione*: quali delle proprietà derivate persistono e quali no?
 - Sappiamo che gli oggetti appoggiati su un tavolo sono nella stanza in cui si trova il tavolo
 - Se si sposta il tavolo da una stanza all'altra anche gli oggetti sopra il tavolo devono cambiare locazione ma i *frame axiom* potrebbero far persistere la vecchia locazione

Pianificazione come dimostrazione di teoremi

- Per generare una sequenza di azioni p per raggiungere l'obiettivo G , si cerca di dimostrare che

$$\exists p G(\text{RisultatoSeq}(p, s_0))$$
- Il pianificatore di Green usa un dimostratore di teoremi basato sulla refutazione
- Complesso per varie sorgenti di non determinismo:
 - lunghezza incognita della sequenza delle azioni
 - frame axiom* che deducono molte cose irrilevanti
 - strategie *ad hoc* necessarie, ma fanno perdere la completezza
- Un theorem prover generale è *inefficiente e semidecidibile*
- Nessuna garanzia sull'efficienza del piano generato

Pianificazione "classica"

- Un linguaggio specializzato per la pianificazione (PDDL)
- Rappresentazione specializzata per gli stati:
 - un insieme di formule atomiche *ground* positive (*fluents*), prive di funzioni
 - tutto ciò che non è detto viene considerato falso (assunzione di mondo chiuso) e costanti diverse corrispondono a oggetti diversi (assunzione di nome unico)
 - Es. $\text{On}(a, b), \text{Clear}(a), \text{Table}(b)$
 NOTA: niente variabili, niente negazioni, niente vincoli di stato
- Rappresentazione del goal
 - congiunzione di formule atomiche positive, che possono anche contenere variabili
 - Es. $\text{On}(b, a) \wedge \text{On}(c, b)$
- Non proprio logica proposizionale ...

Rappresentazione per le azioni

- Azioni: espresse mediante schemi di azione con variabili:

Stack(x, y):	
Precondizioni:	$\text{Clear}(x), \text{Table}(x), \text{Clear}(y)$
Effetti:	$\text{On}(x, y), \neg \text{Table}(x), \neg \text{Clear}(y)$
[Add-list:	$\text{On}(x, y)$ <i>effetto positivo</i>
Delete-list:	$\text{Table}(x), \text{Clear}(y)$ <i>effetto negativo</i>
Unstack(x, y):	
Precondizioni:	$\text{Clear}(x), \text{On}(x, y)$
Effetti:	$\text{Table}(x), \text{Clear}(y), \neg \text{On}(x, y)$
[Add-list:	$\text{Table}(x), \text{Clear}(y)$ <i>effetto positivo</i>
Delete-list:	$\text{On}(x, y)$ <i>effetto negativo</i>

Rappresentazione per le azioni

- Tutte le variabili nella precondizione e nell'effetto devono apparire come parametri dell'azione
- Le precondizioni possono contenere letterali positivi e negativi, soddisfatti se i letterali non sono nello stato
- Nota:** i letterali non menzionati nell'effetto si intendono non modificati
 - Conseguenza 1: il *frame problem* non è più un problema
 - Conseguenza 2: vanno menzionati esplicitamente tutti i cambiamenti; anche quelli che sarebbero "derivabili" in un linguaggio che consentisse di esprimere vincoli di stato

Pianificazione con PDDL

- Decidibilità: garantita in pianificazione classica (con PDDL); il numero di stati è finito se non ci sono le funzioni.
- Complessità del problema di decidere se esiste un piano (PlanSAT): NP. Senza precondizioni negative è nella classe P.
- Complessità di trovare un piano di lunghezza k o inferiore (bounded PlanSAT): NP-completo
- Ma esistono euristiche efficaci.

Pianificazione come ricerca

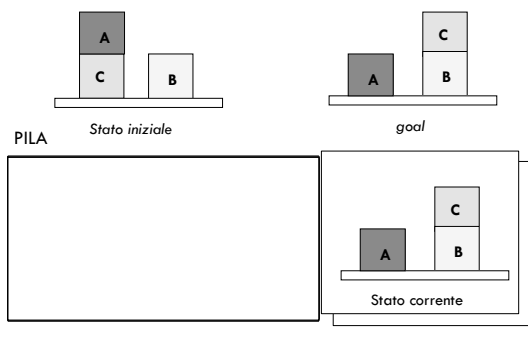
- Lo spazio di ricerca è definito da che cosa sono gli stati e gli operatori:
 - Pianificazione come *theorem proving*: stati come insiemi di formule e operatori come regole di inferenza
 - Pianificazione nello spazio degli stati: stati come descrizioni di situazioni e azioni come modifiche dello stato
 - In avanti (a progressione)
 - All'indietro (a regressione)
 - Pianificazione nello spazio dei piani: stati come piani parziali e operatori di raffinamento e completamento di piani

Pianificazione all'indietro

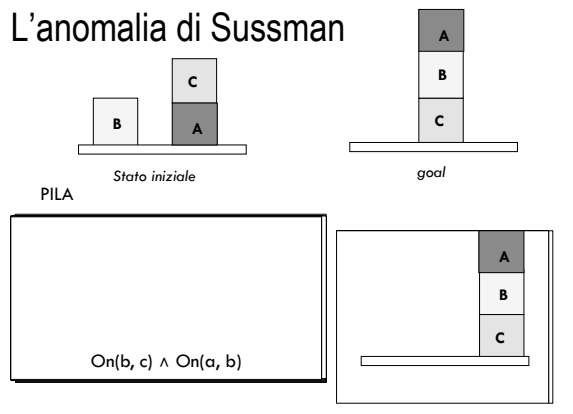
- Si cerca un'azione a tale che uno dei suoi effetti contribuisca positivamente all'obiettivo g . Lo stato precedente è:

$$g' = (g - \text{ADD}(a)) \cup \text{Precondizioni}(a)$$
 anche parzialmente istanziate (con variabili)
- Un esempio storico è STRIPS: un pianificatore a regressione nello spazio degli stati
 - Utilizza una pila in cui all'inizio viene messo il goal; ha una nozione di stato corrente, all'inizio lo stato iniziale
 - Ad ogni passo, per risolvere un goal, cerca di utilizzare un operatore che rende vero il goal e di renderne vere le precondizioni (che diventano dei sotto-goal)

Esempio di funzionamento di STRIPS



L'anomalia di Sussman



L'anomalia di Sussman

- Il piano generato è:
 - [Unstack(c), Stack(a, b), Unstack(a), Stack(b, c), Stack(a, b)]
- Un altro piano generabile è:
 - [Stack(b, c), Unstack(b), Unstack(c), Stack(a, b), Unstack(a), Stack(b, c), Stack(a, b)]
- Il piano ideale non ottenibile con pianificazione "lineare" (con ipotesi di indipendenza)
 - [Unstack(c), Stack(b, c), Stack(a, b)]

Ricerca in avanti

- Apparentemente più inefficiente
 - si tende ad esplorare stati irrilevanti per la soluzione
 - le azioni da esplorare possono essere molte
- È importante usare euristiche efficienti:
 - dipendenti dal problema
 - di utilità generale, sfruttando la rappresentazione fattorizzata

Euristiche per la pianificazione

- Euristiche ammissibili: cercare di stimare quante azioni servono a soddisfare i goal in problemi *rilassati* (meno vincolati). In questo siamo aiutati dalla rappresentazione *fattorizzata*
- Ignorando tutte le precondizioni abbiamo un problema rilassato in cui tutte le azioni sono applicabili
- possiamo cercare di contare il numero di azioni richieste per soddisfare l'obiettivo e usare questo come stima di distanza dalla soluzione
- Non coincide con contare i letterali nel goal
 1. Pessimista: ci sono azioni che possono risolvere più obiettivi
 2. Ottimista: trascura interazioni negative

Euristiche per la pianificazione

- **Strategia 1**
 - Trascurare (oltre alle precondizioni) gli effetti delle azioni che non sono nel goal
 - Contare le azioni necessarie a "coprire" il goal
- **Strategia 2**
 - Ignorare le liste di eliminazioni, gli effetti negativi
 - fa sì che si possa procedere in maniera monotona: nessuna azione annullerà mai gli effetti
 - si può usare hill-climbing per essere più efficienti

Euristiche per la pianificazione

- Lavorare con astrazioni degli stati (ignorando alcuni fluenti)
- Assumere sotto-obiettivi indipendenti e costruire piani separati:
 - max costi è un'euristica ammissibile
 - somma dei costi non è ammissibile in generale ma lo è se i sotto-obiettivi sono davvero indipendenti

Grafi di pianificazione

Possono essere utilizzati (1) per trovare una euristica e (2) sono alla base dell'algoritmo GRAPHPLAN

Init: Have(Cake)

Obiettivo: Have(Cake), Eaten(Cake)

Azione: Eat(Cake)

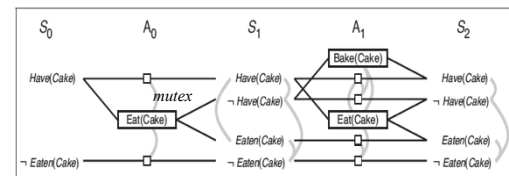
Azione: Cook(Cake)

Precond: Have(Cake)

Precond: ¬Have(Cake)

Effetti: ¬Have(Cake), Eaten(Cake)

Effetti: Have(Cake)



Ricavare un'euristica dal grafo

- Se il goal g non compare in nessun livello possiamo essere sicuri che è irraggiungibile. Altrimenti il grafo può dare una stima di quanti passi servono
- Costo di livello di g_i : il primo livello in cui appare g_i , considerando però grafi serializzati: grafi in cui si esegue una sola azione alla volta (tutte le coppie di azioni sono *mutex*)
- Stima di una congiunzione di goal:
 - max dei costi di livello, poco accurata
 - somma dei livelli, accurata per problemi indipendenti o quasi
 - livello di insieme: livello in cui tutti i letterali del goal congiunto appaiono non in *mutex*

GraphPlan

- Costruisce incrementalmente il grafo di pianificazione espandendo per livelli fino a quando tutti i sotto-goal congiunti compaiono non in *mutex*, poi esegue *Estrai soluzione*.
- Se *Estrai soluzione* fallisce continua con l'espansione.
- *Estrai soluzione* può essere risolto come:
 - Un CSP booleano in cui le variabili sono le azioni del piano (*true, false*) e i vincoli sono i *mutex*, i goal e le precondizioni
 - Un problema di ricerca all'indietro.

Pianificazione nello spazio dei piani

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

Partial Order Planning (POP)

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

Azioni Start e Finish



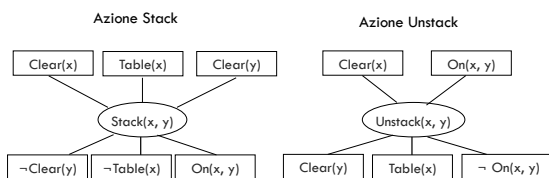
Rappresentazioni per i piani

I piani sono:

- Un insieme di azioni, tra cui Start e Finish.
- Un insieme di precondizioni "aperte"
- Con vincoli di due tipi tra le azioni
 - Relazioni di ordinamento: $S_1 < S_2$ (S_1 prima di S_2)
 - Link causali: $S_1 \text{ cond} \rightarrow S_2$ (S_1 realizza cond per S_2)
- Nota: Se $S_1 \text{ cond} \rightarrow S_2$ allora $S_1 < S_2$ ma non viceversa
- Esempio:
 - {Unstack(a, b), Unstack(c, d), Stack(b, a), Stack(d, c), Start, Finish}
 - Unstack(a, b) < Stack(b, a) Unstack(a, b) Clear(b) \rightarrow Stack(b, a)
 - Unstack(c, d) < Stack(d, c) Unstack(c, d) Clear(d) \rightarrow Stack(d, c)

Rappresentazione per le azioni

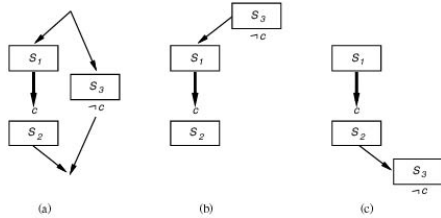
- Azioni come in STRIPS (qui visualizzate a grafo)



POP funzionamento

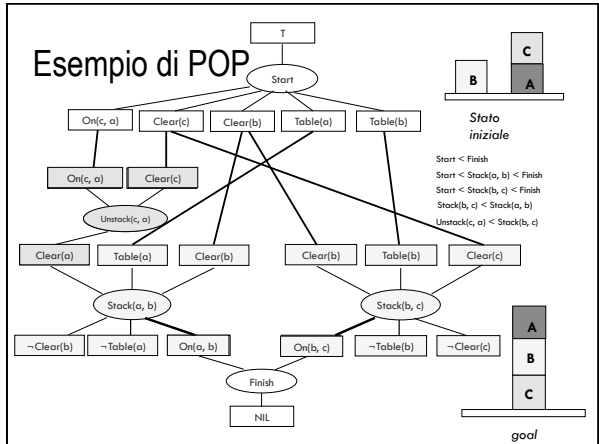
- Si parte dal piano vuoto con Start e Finish.
- Ad ogni passo:
 - Si sceglie arbitrariamente un passo B e una sua precondizione aperta p e si genera un piano successore per ogni azione A (vecchia o nuova) che ha tra gli effetti p
 - Scelta A si ristabilisce la consistenza come segue:
 - Si aggiunge al piano $A < B$ e $A \text{ P} \rightarrow B$
 - Eventuali azioni C che hanno come effetto $\neg p$, sono potenziali *conflitti* (o minacce) e vanno anticipati o ritardati imponendo $C < A$ o $B < C$. Questo passo può anche fallire.
- Ci si ferma quando la lista delle precondizioni aperte è vuota.

Rimozione delle minacce



- (a) S_3 minaccia la preconditione c di S_2 , attivata da S_1
 (b) Risoluzione della minaccia per *demozione*
 (c) Risoluzione della minaccia per *promozione*

Esempio di POP



POP: conclusione

- Abbiamo ottenuto un piano completo e consistente
- Ogni linearizzazione è una soluzione
- In questo caso una sola:
 $[\text{unstack}(c, a), \text{stack}(b, c), \text{stack}(a, b)]$
- POP è corretto e completo (ogni piano calcolato è una soluzione e se un piano esiste viene trovato)

Possibili euristiche per POP

- Tra le precondizioni aperte scegliere quella che ha meno modi per essere soddisfatta
- Analogo alla variabile più vincolata nei CSP
- Si riconoscono prima i fallimenti.

Pianificazione in ambienti reali

- Gli ambienti reali sono:
 - Complessi
 - Non accessibili (incompletezza)
 - Non deterministici (non correttezza)
- Evoluzioni delle tecniche di pianificazione:
 - Pianificazione *gerarchica* per dominare la complessità
 - Pianificazione *condizionale* (o *con contingenza*)
 - Monitoraggio dell'esecuzione di un piano
 - Pianificazione "situata" (integrazione di pianificazione ed esecuzione)