ANNO ACCADEMICO 2024/2025

Intelligenza Artificiale e Laboratorio

Teoria - Micalizio

Altair's Notes

UNIVERSITÀ DI TORINO

DIPARTIMENTO DI INFORMATICA

Premessa

Licenza

Formato utilizzato

Box di "Concetto sbagliato":

Concetto sbagliato 0.1: Testo del concetto sbagliato

Testo contente il concetto giusto.

Box di "Corollario":

Corollario 0.0.1 Nome del corollario

Testo del corollario. Per corollario si intende una definizione minore, legata a un'altra definizione.

Box di "Definizione":

Definizione 0.0.1: Nome delle definizione

Testo della definizione.

Box di "Domanda":

Domanda 0.1

Testo della domanda. Le domande sono spesso utilizzate per far riflettere sulle definizioni o sui concetti.

Box di "Esempio":

Esempio 0.0.1 (Nome dell'esempio)

Testo dell'esempio. Gli esempi sono tratti dalle slides del corso.

Box di "Note":

Note:-

Testo della nota. Le note sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive.

Box di "Osservazioni":

Osservazioni 0.0.1

Testo delle osservazioni. Le osservazioni sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive. A differenza delle note le osservazioni sono più specifiche.

1 Planning

Seconda parte del corso:

- Comprendere i problemi fondamentali alla base degli algoritmi di pianificazione.
- Studiare alcuni degli approcci classici alla pianificazione:
  - Assunzioni del planning classico.
  - Algoritmi di ricerca nello spazio degli stati (progression, regression, STRIPS).
  - Algoritmi di ricerca nello spazio dei piani (least-commitment planning).
  - Algoritmi di ricerca basati su grafi (grafo di pianificazione, GRAPHLAN).
  - Euristiche domain-independent per il planning.
  - Altri approcci al planning (HTN).
- Sistemi a regole:
  - Paradigma dei sistemi esperti basati su regole di produzione.
  - Sperimentare il paradigma a regole in CLIPS.
- Incertezza:
  - Modellare l'incertezza con le probabilità.
  - Meccanismi di inferenze probabilistiche.

1.1 Che Cos'è il Planning?

Definizione 1.1.1: Planning (secondo Haslum)

Il *planning* è l'arte e la pratica di pensare prima di agire.

Definizione 1.1.2: Automated Planning (Hoffman)

Selezionare un goal che motivi delle azioni basate su una descrizione ad alto livello del mondo.

In altre parole:

- Il planning è un processo deliberativo che sceglie ed organizza le azioni in base all'effetto che ci si aspetta queste producano.
- AI planning è lo studio della calcolabilità di questo processo deliberativo.

1.1.1 Modello Concettuale per il Planning

State Transition System $\Sigma = (S, A, E, \gamma)$

- Dove:
  - $S = \{s_1, s_2, \ldots\}$ insieme finito, ricorsivamente enumerabile di stati
  - $A = \{a_1, a_2, \ldots\}$ insieme finito, ricorsivamente enumerabile di azioni
  - $E = \{e_1, e_2, \ldots\}$ insieme finito, ricorsivamente enumerabile di eventi
  - $\gamma : S \times (A \cup E) \longrightarrow 2^S$ è una relazione di transizione di stato
- Se $a \in A$ e $\gamma(s, a) \neq \emptyset$, allora a è *applicabile* in s
- Applicare a in s causerà una transizione di stato del sistema da s a s', dove $s' \in \gamma(s, a)$

Osservazioni 1.1.1

Un STS $\Sigma = (S, A, E, \gamma)$ può essere rappresentato come un grafo diretto $G = (N_G, E_G)$ dove:

- $N_G = S$ è l'insieme dei nodi del grafo coincidente con l'insieme degli stati di Σ
- E_G è l'insieme degli archi del grafo tale che esiste un arco $s \xrightarrow{u} s'$ (anche rappresentato come $\langle s, u, s' \rangle$) da s a s' etichettato con $u \in A \cup E$, *se e solo se*:
  - $s, s' \in S$ e
  - $s' = \gamma(s, u)$

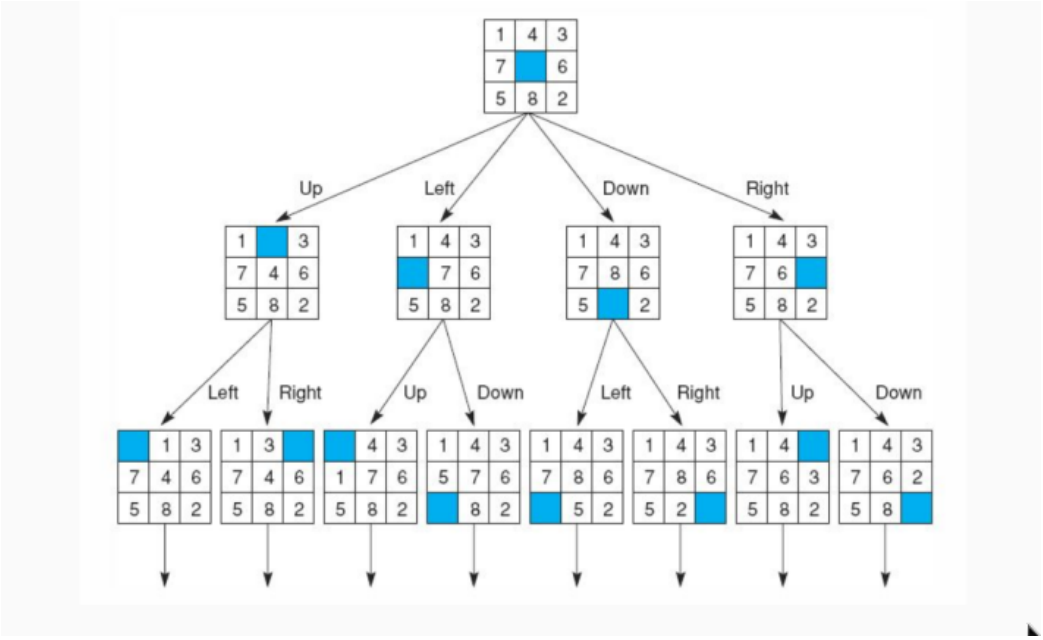

Figure 1.1: STS visto come grafo.

Ingredienti del planning:

- *STS*:
  - Descrive tutte le possibili evoluzioni del sistema.
- *Piano*:
  - Struttura che traccia le azioni necessarie per raggiungere un certo obiettivo G dato uno stato iniziale I.
  - È un cammino da I a G nello spazio degli stati tracciato da STS.
- *Goals*:
  - Un goal state s_g o un sottoinsieme di possibili goal state S_g.
  - Soddisfacimento di condizioni in tutta la sequenza di stati prodotta dalle azioni.
  - Ottimizzazione di funzioni di utilità.
  - Vincoli sulle azioni che possono essere eseguite.

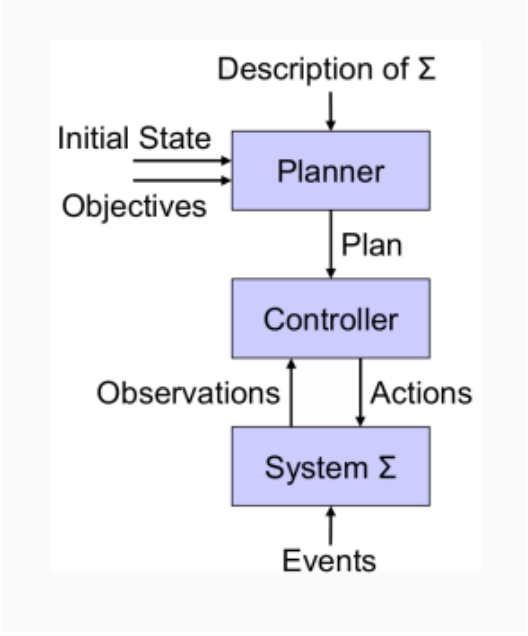

Figure 1.2: Si assume che gli eventi non interferiscano con il controller.

- *Planner:*
  - Data la descrizione di un STS Σ, lo stato iniziale, e il goal.
  - Genera un piano che raggiunge il goal dallo stato iniziale.
- *Controller:*
  - Dato un piano e lo stato corrente (funzione di osservabilità $\eta : S \rightarrow O$).
  - Seleziona ed esegue un'azione del piano.
- *STS Σ:*
  - Evolve in funzione delle azioni eseguite e degli eventi che possono accadere.

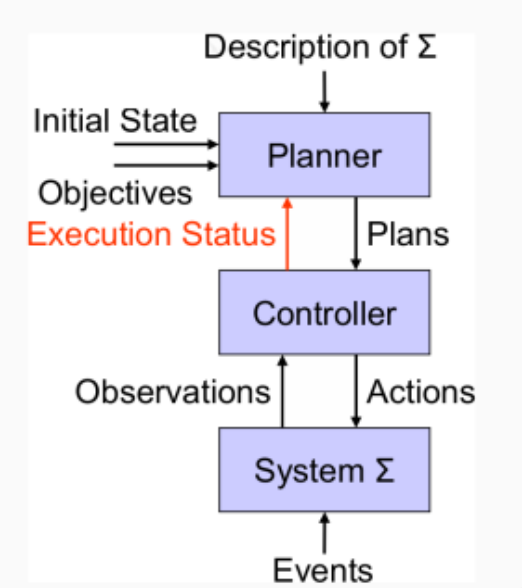

Figure 1.3: Gli eventi possono interferire.

- *Challenge:*
  - Il mondo reale può essere diverso da come è descritto nel modello.
- *Continual planning:*
  - Plan supervision.
  - Plan revision.
  - Re-planning.
- Continual planning consente un loop chiuso di feedback tra planner e controller.

1.1.2 Pianificazione Classica

Note:-

La pianificazione classica è la pianificazione che avviene sotto alcune assunzioni.

Definizione 1.1.3: Dominio Finito

Σ contiene un numero finito di stati.

Corollario 1.1.1 Rilassare un Dominio Finito (A0)

Per:

- Descrivere azioni che producono nuovi oggetti nel mondo.
- Trattare fluenti numerici.

Problemi:

- Decidibilità e terminazione del pianificatore.

Definizione 1.1.4: Dominio Completamente Osservabile

La funzione $\eta : S \rightarrow O$ è la funzione identità.

Corollario 1.1.2 Rilassare un Dominio Completamente Osservabile (A1)

Per

- Trattare state in cui non tutto è osservabile o può essere conosciuto.

Problemi:

- In genere si osserva solo un sottoinsieme della realtà, può accadere che $\eta(s) = \eta(s') = o$ con $s \neq s'$.
- Le osservazioni sono ambigue perché consistenti con più stati possibili.
- Determinare lo stato successore può essere problematico.

- Conformant planning (pianificazione in assenza di osservazioni): pianificare a prescindere dal reale stato del mondo.

Definizione 1.1.5: Dominio Deterministico

Σ è deterministico, cioè per ogni $s \in S, u \in A \cup E$ si ha $|\gamma(s,u)| \leqslant 1$.

Corollario 1.1.3 Rilassare un Dominio Deterministico (A2)

Per:

- Pianificare con azioni che possono avere risultati alternativi.

Problemi:

- Il controller deve osservare il risultato reale di ogni azione.
- Il piano soluzione potrebbe contenere dei branch condizionali o iterativi.

Definizione 1.1.6: Dominio Statico

Σ è statico, ovvero $E = \emptyset$ e STS può essere ridotto a $\Sigma = (S, A, \gamma)$.

Corollario 1.1.4 Rilassare un Dominio Statico (A3)

Per:

- Modellare domini in cui eventi al di là del controllo dell'esecutore sono possibili.

Problemi:

- Il mondo diventa non deterministico dal punto di vista del pianificare.

Definizione 1.1.7: Dominio con Goal Semplici

Consistono in uno stato s_g da raggiungere o un insieme di stati S_g (è sufficiente che il piano porti a uno di essi).

Note:-

Gli stati possono essere descrizioni parziali di situazioni desiderate.

Corollario 1.1.5 Rilassare un Dominio con Goal Semplici (A4)

Per:

- Trattare vincoli su stati e piani, funzioni di utilità/costo, ottimalità.

Problemi:

- Esprimere e ragionare su vincoli ulteriori nella specifica del goal rende il planning computazionalmente costoso.

Definizione 1.1.8: Dominio con Piani Sequenziali

Un piano soluzione è una sequenza finita di azioni linearmente ordinate.

Note:-

Una sola azione per volta è possibile.

Corollario 1.1.6 Rilassare un Dominio con Piani Sequenziali (A5)

Per:

- Sfruttare le capacità degli esecutori nel caso potessero eseguire più azioni.
- Non introdurre vincoli che non sono parte del dominio.

Problemi:

- Ragionare su e gestire strutture dati più complesse.

Definizione 1.1.9: Dominio con Tempo Implicito

Le azioni e gli eventi non hanno durata (oppure hanno durata istantanea).

Corollario 1.1.7 Rilassare un Dominio con Tempo Implicito (A6)

Per:

- Trattare azioni durative, problemi di concorrenza e deadline.

Problemi:

- Rappresentare e ragionare sul tempo.
- Gli effetti delle azioni si sviluppano nel tempo.

Definizione 1.1.10: Dominio con Single Agent

Un solo pianificatore e un solo controller (esecutore).

Corollario 1.1.8 Rilassare un Dominio con Single Agent (A7)

Per:

- Sfruttare meglio le risorse disponibili.
- Trattare situazioni in cui più esecutori sono presenti ma non sono sotto il controller di un unico pianificatore.

Problemi:

- Multi-agent planning: necessità di trattare le interazioni, coordinazione, competizione, negoziazione e planning della teoria dei giochi.

1.1.3 Problemi di Pianificazione Classica

Un problema di pianificazione classica è $P = (\Sigma, s_0, S_g)$:

- $\Sigma = (S, A, \gamma)$ è il modello del dominio espresso come STS.
- $s_0 \in S$ è lo stato iniziale.
- $S_g \subset S$ è l'insieme degli stati goal.

Una soluzione π a un problema P:

- Una sequenza totalmente ordinata di azioni istanziate (ground). $\pi = \langle a_1, a_2, \ldots, a_n \rangle$.
- Danno origine a una sequenza di transazioni di stato $\langle s_0, s_1, \ldots, a_n \rangle$ tale che:
  - $s_1 = \gamma(s_0, a_1)$.
  - $\forall k : 2..n s_k = \gamma(s_{k-1}, a_k)$.
  - $s_n \in S_g$.

Domanda 1.1

Quali sono le sfide dell'approccio classico al planning?

- Come rappresentare stati e azioni in modo da non dover esplicitamente enumerare S, A, e γ?
- Come ricercare una soluzione in modo efficiente? Quali algoritmi? Quali euristiche?
- Come generalizzare le soluzioni? Classical Planning troppo semplice per essere utile nei casi pratici, ma può essere la base per soluzioni in contesti più complessi (rilassando alcune assunzioni).

Domanda 1.2

Perché la pianificazione è difficile?

- È dimostrabile che il planning è un task computazionalmente costoso:
  - *PlanSAT*: esiste un piano che risolve un problema di pianificazione?
  - *Bounded PlanSAT*: esiste un piano di lunghezza k?
- Per la pianificazione classica entrambi i problemi sono decidibili (la ricerca avviene in spazio finito).
- Se si estende a uno spazio infinito:
  - PlanSAT diventa semi-decidibile: esiste un algoritmo che termina quando la soluzione non esiste, potrebbe non terminare quando la soluzione non esiste.
  - Bounded PlanSAT rimane decidibile.

Osservazioni 1.1.2

- Bounded PlanSAT è NP completo mentre PlanSAT è P.
- Trovare una soluzione è meno costoso che trovare una soluzione ottima.
- La complessità del planning giustifica la ricerca di euristiche, possibilmente domain-independent, che guidino il pianificatore nella sintesi di una soluzione.

Proprietà di un buon algoritmo di pianificazione:

- *Soundness* (correttezza): un pianificatore è corretto se tutte le soluzioni che trova sono piani corretti, ovvero realmente eseguibili dal controller:
  - Tutti i goals sono soddisfatti.
  - Nessuna precondizione di azione è open (mancante).
  - Nessun vincolo ulteriore è violato (vincoli temporali, istanziazione di variabili, etc.).
- *Completeness* (completezza):
  - Un pianificatore è completo se trova una soluzione quando il problema è risolubile.

  - Un pianificatore è strettamente completo se tutte le soluzioni sono mantenute nello spazio di ricerca (eventuali *pruning* dello spazio non scartano soluzioni).

- *Ottimalità*: un pianificatore è ottimo se l'ordine con cui le soluzioni sono trovate è coerente con una qualche misura di qualità dei piani (lunghezza, costo complessivo, etc.).

Note:-
Molti algoritmi, per favorire la velocità, rinunciano all'ottimalità.

1.2 Algoritmi di Pianificazione

1.2.1 Ricerca in Avanti e Ricerca all'Indietro

Definizione 1.2.1: Progression

Calcolo dello stato successore s' di uno stato s rispetto all'applicazione di un operatore o:

$$s' = \gamma(s, o)$$

Pianificatori basati su progression applicano delle ricerche in avanti tipicamente nello spazio degli stati:

- La ricerca comincia da uno stato iniziale.
- Iterativamente viene applicato un operatore o per generare un nuovo stato s'.
- La soluzione è trovata quando lo stato s' appena generato soddisfa il goal ($s' \vDash s_g$ e $s_g \in S_g$).
- Ha il vantaggio di essere molto intuitivo e facile da implementare.

```
function fwdSearch(O, s_0, s_g)
  state ← s_0
  π ← ⟨⟩
  loop
      if state.satisfies(s_g) then return π
      applicables ← {istanze ground da O applicabili in state}
      if applicables.isEmpty() then return failure
      action ← applicables.chooseOne()
      state ← γ(state, action)
      π ← π ∘ ⟨action⟩
  return failure
```

Figure 1.4: applicables.chooseOne() rappresenta una scelta non deterministica, ma esaustiva, di un'azione.

Note:-
chooseOne() avrà varie implementazioni a seconda dell'algoritmo scelto.

Corollario 1.2.1 Soundness di Progression

fwdSearch è corretto: se la funzione termina con un piano come soluzione, allora questo piano è effettivamente una soluzione al problema iniziale.

Corollario 1.2.2 Completeness di Progression

fwdSearch è completo: se esiste una soluzione allora esiste una traccia d'esecuzione che restituirà quella

soluzione come piano.

Osservazioni 1.2.1

- Il numero di azioni applicabili in un dato stato è in genere molto grande.
- Anche il branching factor tende a essere grande.
- La ricerca in avanti corre il rischio di non essere praticabile dopo pochi passi.

Ricerca in avanti vs. Ricerca all'indietro:

- La ricerca in avanti comincia da un singolo stato iniziale mentre la ricerca all'indietro comincia da un insieme di stati.
- Quando si applica in avanti un operatore o a uno stato s si genera un unico stato successore s', all'indietro possono esserci molteplici stati predecessori.
- Nella ricerca in avanti lo spazio di ricerca coincide con lo spazio degli stati, all'indietro ogni stato dello spazio di ricerca corrisponde a un insieme di stati del dominio.

Definizione 1.2.2: Regression

Il calcolo del regresso, ovvero il sottogoal predecessore di un goal dato, avviene nel seguente modo:

- Dato un goal g.
- Sia a azione ground tale che $g \in effects^+(a)$.
- $g' = \gamma^{-1}(g, a) = (g\ effects^+(a)) \cup pre(a)$.

g' è il regresso di g attraverso la relazione di transizione γ e l'azione a.

Nella ricerca all'indietro:

- Si comincia dall'insieme di stati goal.
- Iterativamente si seleziona un sottogoal generato precedentemente e si regredisce attraverso un operatore generando un nuovo sottogoal.
- La soluzione è trovata quando il nuovo sottogoal è soddisfatto dallo stato iniziale.
- Ha il vantaggio di poter gestire più stati contemporaneamente.
- È più costoso e difficile.

```
function bwdSearch(O, s0, g)
  subgoal ← g
  π ← ⟨⟩
  loop
      if s0.satisfies(subgoal) then return π
      relevants ← {ground instances from O relevant for subgoal}
      if relevants.isEmpty() then return failure
      action ← relevants.chooseOne()
      subgoal ← γ⁻¹(subgoal, action)
      π ← ⟨action⟩ ∘ π
```

Osservazioni 1.2.2

- La ricerca all'indietro si basa su azioni rilevanti, cioè quelle che contribuiscono attivamente al goal:
  - Producono almeno uno degli atomi che compaiono nel goal.
  - Non hanno effetti negativi sul goal stesso (non negano uno degli atomi del goal).
- La ricerca all'indietro mantiene un fattore di ramificazione più basso rispetto alla ricerca in avanti, ma il fatto di dover mantenere un belief state può complicare le strutture dati usate dal planner e la definizione di euristiche.
- Nonostante il vantaggio teorico la ricerca in avanti è preferita alla ricerca all'indietro.

1.2.2 STRIPS

Definizione 1.2.3: STRIPS

STanford Research Institute Problem Solver:

- Introduce una rappresentazione esplicita degli operatori di pianificazione.
- Fornisce una operalizzazione delle nozioni di:
  - Differenza tra stati.
  - Subgoal.
  - Applicazione di un operatore.
- Gestisce il Frame Problem.
- Si basa su due idee fondamentali:
  - Linear planning.
  - Means-End Analysis.

Osservazioni 1.2.3 Linear Planning

- Idea di base:
  - Risolvere un goal per volta, passare al successivo solo quando il precedente è stato raggiunto.
- L'algoritmo di planning mantiene uno *stack dei goal*:
  - Risolvere un goal può comportare la risoluzione di sottogoal.
- Conseguenze:
  - Non c'è interleaving nel conseguimento dei goal.
  - La ricerca è efficiente se i goal non interferiscono troppo tra loro.
  - Ma è soggetto alla *Sussmann's Anomaly*.

Osservazioni 1.2.4 Means-End Analysis

- Idea di base:
  - Considera solamente gli aspetti rilevanti al problema (backward).
  - Quali mezzi (means, operatori) sono disponibili e necessari per raggiungere il goal (end).

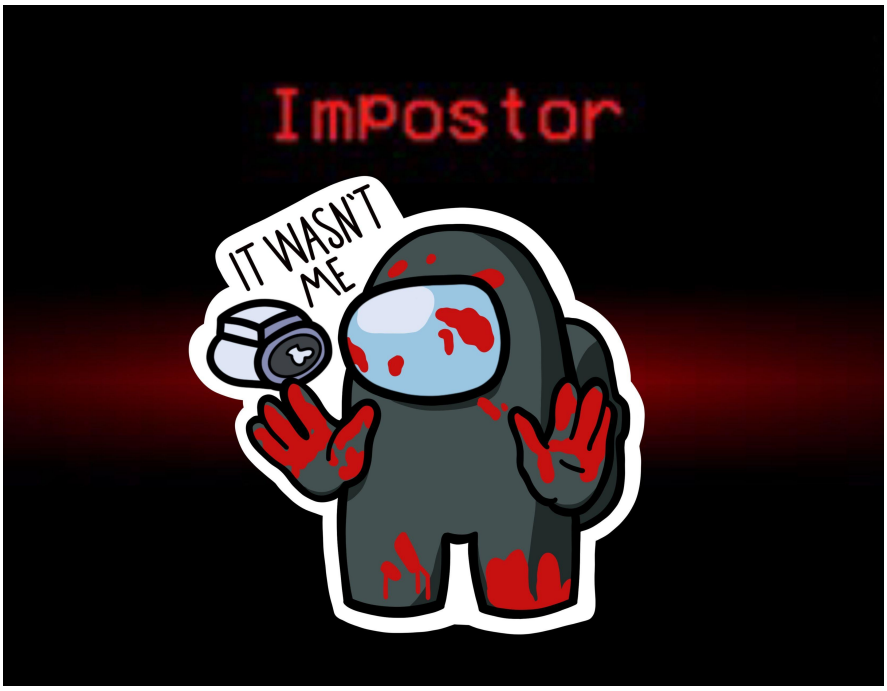

Figure 1.5: Sussmann's Anomaly caught in 4k.

- Occorre stabilire quali differenze ci sono tra lo stato corrente e il goal.
- Trovare un operatore che riduce tale differenza.
- Ripetere l'analisi sui sottogoal ottenuti per regressione attraverso l'operatore selezionato.

Caratteristiche di STRIPS:

- È un sottoinsieme della logica del primordine:
  - Numero finito di simboli di predicati, simboli di costanti e senza simboli di funzione né quantificatori.
- Uno stato in STRIPS è una congiunzione di *atomi ground* (privi di simboli di funzione).
- Vige l'*assunzione di mondo chiuso*: quello che non è descritto è considerato falso.
- Semantica degli insiemi:
  - Un atomo ground p vale in uno stato s se e solo se $p \in s$.
- Semantica logic-oriented:
  - Uno stato s soddisfa una congiunzione di letterali g, denotato come $s \vDash g$ se ogni letterale positivo in g occorre in s e se ogni letterale negativo in g non occorre in s.

Relazioni Fluenti:

- Predicati che rappresentano relazioni il cui valore di verità può cambiare da uno stato al successivo.

Relazioni Persistenti:

- Predicati il cui valore di verità non può cambiare.

Definizione 1.2.4: Plan Operators

Un operatore di pianificazione in STRIPS è una tripla: $o : (name(0), precond(o), effects(o))$ dove:

- $name(o)$ è una rappresentazione sintattica del tipo $n(x_1, \ldots, x_k)$ dove n è il nome dell'operatore e $x_1, \ldots, x_k$ è una lista di variabili che compaiono in o.
- $precond(o)$ è l'insieme di letterali che rappresentano le precondizioni dell'azione.
- $effect(o)$ è l'insieme di letterali che rappresentano gli effetti dell'azione.

Note:-

Una variabile può comparire negli effetti solo se è anche menzionata nelle precondizioni.

Definizione 1.2.5: Calcolo dello Stato Successore (progression)

Sia s uno stato del mondo ($s \in S$ per un dato dominio Σ). Sia a un'azione. Diremo che a è *applicabile* in s se e solo se:

- $precond^+(a) \subseteq s$.
- $precond^-(a) \cap s = \emptyset$.

La funzione di transizione di stato $\gamma(s, a)$ è definita:

- Se a è applicabile in s:
  - $\gamma(s, a) = (s\ effects^-(a)) \cup effects^+(a)$.
  - Altrimenti $\gamma(s, a)$ è indefinita.

Note:-

STRIPS usa la progression per modificare lo stato corrente a cui si giunge eseguendo le azioni fin qui selezionate nel piano in costruzione.

Definizione 1.2.6: Calcolo del Sottogoal Predecessore (regression)

Il calcolo del regresso avviene:

- Dato un goal g.
- Sia a azione ground tale che $g \in effects^+$(a).
- $g' = \gamma^{-1}(g, a) = (g\ effects^+(a)) \cup pre(a)$

Note:-

STRIPS usa la regression per ridurre la distanza tra il goal che si vuole raggiungere e lo stato corrente.

Il planner di STRIPS:

✓ Lo spazio di ricerca è ridotto perché i goal sono considerati uno per volta.

✓ Ideale quando i goal sono indipendenti tra loro.

✓ È sound.

✗ Può produrre piani subotttimi (Sussmann's Anomaly).

STRIPS (*initState*, *goals*)

`state` = *initState*; `plan` = []; `stack` = []

Push *goals* on `stack`

Repeat until `stack` is empty

- **If** top of `stack` is a goal g satisfied in *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 $effects^+$ matches goal sg
  Replace goal sg with operator o
  Push the preconditions of o on `stack`
- **Else if** top of `stack` is an operator o, then
  state = **apply**(o, *state*)
  `plan` = [`plan`; o]

✗ È incompleto (perché alcune azioni in alcuni domini non sono reversibili).

Definizione 1.2.7: Sussmann's Anomaly

È necessario disfare parte dei goal già raggiunti per poter risolvere l'intero problema.

Note:-

È una conseguenza di:

- Interdipendenza tra i sottogoal.
- Ordinamento sfavorevole dei goal.

Definizione 1.2.8: Planning Domain Definition Language

PDDL è uno standard per definire problemi e domini. Nella sua versione base è una sistematizzazione di STRIPS.

Estensioni di PDDL:

- *Conditional effects*: stabilire che alcuni effetti sono raggiunti solo se sono vere certe condizioni.
- *Durative actions*: un plan operator si divide in tre parti:
  - Un evento iniziale.
  - Un evento finale.
  - Una condizione invariante che deve permanere per tutta la durata dell'azione.
- *Numeric fluent*: consente di tracciare il consumo di risorse nel tempo.
- *Quantificators*.

1.3 Pianificazione nello Spazio dei Piani

Domanda 1.3

Come costruire un modello espresso da variabili di stato?

- Dato un ambiente E che si vuole modellare.
- Dato l'insieme degli oggetti *rilevanti* B.
- B deve astrarre i dettagli insignificanti (non indispensabili per la risoluzione del problema).

Gli oggetti hanno due tipi di proprietà:

- *Rigid*:
  - Proprietà invarianti, persistono in ogni possibile stato del sistema.
  - Nella rappresentazione classica corrispondono agli atomi grounf.
  - In una rappresentazione state-variable questi atomi ground corrispondono a costanti booleane.
- *Varying*:
  - Possono cambiare nelle transizioni di stato.
  - Nella rappresentazione classica solo le relazioni fluenti che vengono aggiunte/tolte per effetto delle azioni.
  - Sono modellate come variabili a cui possiamo assegnare un valore.
  - Uno stato è rappresentato su un insieme e ogni variabile di stato ha un dominio.

Logica del primordine vs. state-variable:

- La logica del primordine ha origini storiche, il planning viene visto come meccanismo di theorem proving.
- La logica del primordine è *equivalente* a state-variable:
  - Stesso potere espressivo.
  - Riconducibili l'uno all'altro.
- State-variable è più sintetica e uniforme.
- State-variable si presta meglio a estensioni verso il non classical planning e all'uso di euristiche domain-independent.

1.3.1 Least-Commitment Planning

Definizione 1.3.1: Principio Least-Commitment

Si fanno scelte solo quando sono indispensabili per risolvere una parte del problema e durante la ricerca non si pongono più vincoli dello stretto necessario.

Decisioni che è possibile ritardare:

- *Ordinamenti*: non ordinare le azioni a meno che non sia strettamente necessario.
- *Bindings*: non ordinare le azioni a meno che non sia indispensabile unificarle con costanti al fine di conseguire i goal.

Idea:

- La ricerca avviene nello spazio dei piani.
- Ogni nodo della ricerca è un piano parzialmente ordinato con *flaws* (difetti).
- A ogni passo si rimuove un difetto per mezzo di *raffinamenti incrementali* del piano parziale corrente.
- Se l'algoritmo termina con successo il piano risultante è completamente istanziato ma solo parzialmente ordinato.

Note:-
Spazio di ricerca = insieme dei piani parziali.

Un piano è una tupla $\langle A, O, B, L\rangle$ dove:

- A è l'insieme di azioni, anche parzialmente istanziate.
- O è l'insieme dei *vincoli di ordinamento* della forma $a_i < a_j$, è una rekazione d'ordine parzialmente definita.
- B è l'insieme dei *bindings* (vincoli) della forma:
  - $v_i = C$, a v_i è assegnata la costante C.
  - $v_i \neq C$, v_i può assumere qualsiasi valore tranne C.
  - $v_i = v_j$, v_i e v_j assumono lo stesso valore.
  - $v_i \neq v_j$, v_i e v_j devono essere distinte.
- L è l'insieme di *causal links* della forma $a_i \rightarrow^c a_j$, in cui c è un effetto dell'azione a_i che è necessario per l'azione a_j.

Corollario 1.3.1 Flaws - Open Goals

Sia π il piano del nodo corrente:

- Una precondizione p per un'azione b è un open goal se non c'è un link causale a supporto di p.
- Si può risolvere il difetto aggiungendo il link causale mancante:
  1. Si trova un'azione a tale che:
     - p può appartenere a $effect^+(a)$.
     - a può precedere b in π.
  2. Si istanzia l'azione a in modo che asserisca p.
  3. Si aggiunge un vincolo di precedenza tra a e b in O.
  4. Si aggiunge un causal link $a \rightarrow^p b$ in L.

Corollario 1.3.2 Flaws - Threats

Sia π il piano del nodo corrente:

- Dato un link causale $l : a \rightarrow^p b$.
- Un'azione c minaccia il link l se:
  - c può modificare il valore di verità di p e può posizionarsi tra a e b.
  - c può produrre p, il link l impone che sia a a produrre p e non un'altra azione c.
- c viene detta *clobber*.

Risolvere le minacce:

- *Promotion*: imporre il vincolo che c precede a.
- *Demotion*: imporre il vincolo che b precede a.
- *Separation*: imporre un vincolo di non-codesignation in modo tale che l'effetto di c non unifichi con p.

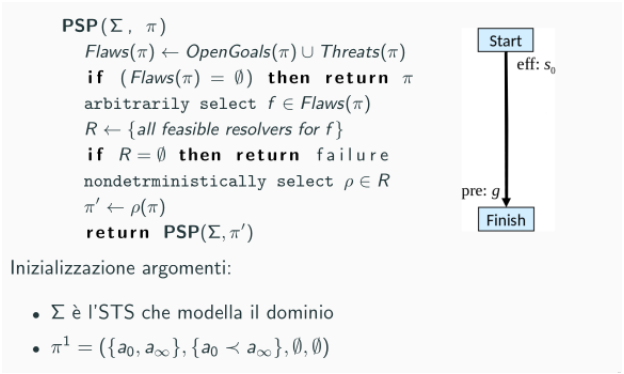

Figure 1.6: Algoritmo di Plan-Space Planning (PSP).

Osservazioni 1.3.1

PSP è corretto e completo:

- Ritorna un piano parzialmente ordinato π tale che qualsiasi ordinamento totale delle sue azioni raggiunge il goal.
- Dove il dominio lo consenta, le azioni non strettamente sequenziali possono essere eseguite in parallelo.

1.4 Graphplan

1.4.1 Introduzione

Definizione 1.4.1: Grafo di Pianificazione

Struttura dati speciale per:

- Definire euristiche domain-independent.
- Generare un piano.

Note:-

Il grafo astrae lo spazio di ricerca e può essere costruito con un algoritmo polinomiale.

In dettaglio:

- Grafo orientato aciclico organizzato a livelli:
  - Livello iniziale S_0 che contiene tutti i letterali che valgono nello stato iniziale (un nodo per ogni fluente).
  - Livello di azioni A_0 che contiene le azioni ground che possono essere applicate a S_0.
  - In generale S_i, A_i si alternano fino a che non si raggiunge una condizione di terminazione:
    - S_i contiene tutti i letterali che *potrebbero* valere al tempo i.
    - A_i contiene tutte le azioni che *potrebbero* avere le precondizioni soddisfatte al tempo i.

Osservazioni 1.4.1

- Graphplan (GP) traccia solo un sottoinsieme delle possibili interazioni negative (un letterale che appare in uno stato S_i per la prima volta) potrebbe di fatto essere producibile solo in uno stato successivo a S_i. Tuttavia i è una buona stima per il letterale.
- GP è costruito a partire da azioni ground.
- Ogni S_i è un *belief state*: codifica più stati possibili e alternative.

Quando si costruisce il grafo bisogna tenere a mente che:

- Un letterale al tempo i può essere inteso sia come precondizione di un'azione in A_i sia come atomo persistente. In questo caso la persistenza è realizzata da un'azione speciale `no-op`.
- In ogni livello S_i possono essere presenti link di *mutua esclusione* tra letterali.
- In ogni livello A_i possono essere presenti link di *mutua esclusione* tra azioni.
- GP cresce monotonicamente, quindi prima o poi si livella: due stati consecutivi S_i e S_{i+1} sono identici.

Mutua esclusione tra due azioni in un dato livello A_i:

- *Effetti inconsistenti:* un'azione nega gli effetti dell'altra.
- *Interferenza:* uno degli effetti di un'azione è la negazione di una precondizione dell'altra.
- *Competizione delle precondizioni:* una delle precondizioni di un'azione è mutuamente esclusiva con le precondizioni dell'altra.

Mutua esclusione tra due letterali in un dato livello S_i:

- *Complementarity:* se uno è la negazione dell'altro.
- *Inconsistent Support:* se ogni possibile coppia di azioni al livello A_{i-1} che producono i due letterali sono mutuamente esclusive.

Domanda 1.4

Perché la costruzione di un GP è polinomiale?

- I: letterali.
- a: azioni.
- Ogni livello S_i ha non più di I nodi e I^2 mutex tra letterali.
- Ogni livello A_i non ha più di $I + a$ nodi e $(a + I)^2$ link di mutex.
- Un grafo con n livelli ha dimensione $O(n(a + I)^2)$ e la stessa complessità per costruirlo.

Osservazioni 1.4.2

- Se un letterale del goal non compare in nessun livello allora il goal non è raggiungibile.
- Se anche tutti i letterali del goal compaiono in un qualche livello del GP non vuol dire che esista una soluzione.

Euristiche per stimare il costo di un singolo letterale:

- Profondità del livello tra quelli in cui un letterale del goal compare per la prima volta.
- Lunghezza del piano serializzabile estratto dal GP.

Euristiche per stimare il costo di una congiunzione di letterali:

- *Max-level:* prendere il massimo livello tra quelli in cui un letterale del goal compare per la prima volta.
- *Somma dei livelli:* assume indipendenza dei letterali del goal, non è ammissibile.
- *Livello di insieme:* profondità del livello in cui tutti i letterali del goal compaiono senza che alcuna coppia di essi sia in mutua esclusione:
  - È ammissibile, funziona bene quando i letterali sono indipendenti tra di loro.
  - Ignora le dipendenze tra tre o più letterali.

Teorema 1.4.1

Se esiste un piano valido allora questo è un sottografo del grafo di pianificazione.

Intuitivamente:

1. Si espande il grafo un livello per volta fino a quando tutti gli atomi del goal non compaiono all'interno dell'i-esimo stato.
2. Si invoca `extract-solution` per cercare un piano all'interno del grafo:
   - Se trova una soluzione termina con successo.
   - Se la soluzione non esiste *certamente* termina con insuccesso.
   - In tutti gli altri casi va al punto 3.
3. Si espande il grafo di un livello e si torna al passo 2.

function GRAPHPLAN(*problem*) **returns** solution or failure

graph ← INITIAL-PLANNING-GRAPH(*problem*)
goals ← CONJUNCTS(*problem*.GOAL)
nogoods ← an empty hash table
for *tl* = 0 **to** ∞ **do**
  if *goals* all non-mutex in S_t of *graph* **then**
    solution ← EXTRACT-SOLUTION(*graph*, *goals*, NUMLEVELS(*graph*), *nogoods*)
    if *solution* ≠ *failure* **then return** *solution*
  if *graph* and *nogoods* have both leveled off **then return** *failure*
  graph ← EXPAND-GRAPH(*graph*, *problem*)

Figure 10.9 The GRAPHPLAN algorithm. GRAPHPLAN calls EXPAND-GRAPH to add a level until either a solution is found by EXTRACT-SOLUTION, or no solution is possible.

Figure 1.7: Algoritmo di Graphplan.

Definizione 1.4.2: Extract Solution

L'estrazione di un piano da un grafo di pianificazione corrisponde a una ricerca backward in un sottografo AND/OR del GP:

- OR: gli archi che dalle aziono a un livello A_{i-1} producono un letterale p in un goal g al livello S_i.
- AND: sono gli archi che dagli atomi a un livello S_i rappresentano le precondizioni per un'azione al livello A_i.

Osservazioni 1.4.3

- Si sfrutta una funzione di supporto `GP-Search`. Sono *mutuamente ricorsive*.
- `extract-solution` invoca `GP-Search` su un livello di azioni e su un sottogoal.
- `GP-Search` pianifica per il solo livello su cui è invocato e se ha sucesso invoca `extract-solution` sul livello di stato precedente.

```
EXTRACT-SOLUTION(G, g, i)
  if i = 0 then return ⟨⟩
  if g ∈ no-good(i) then return failure
  π ← GP-Search(G, g, ∅, i − 1)
  if π ≠ failure then return π
  no-good(i) ← no-good(i) ∪ {g}
  return failure
```

Figure 1.8: Extract-solution.

- G è il grafo di pianificazione.
- g è il sottogoal su cui la funzione è invocata.
- i è la profondità di un livello di stato S_i.

```
GP-SEARCH(G, g, π_i, i)
  if g = ∅ then do
    Π ← EXTRACT-SOLUTION(G, ⋃{precond(a) | ∀a ∈ π_i}, i)
    if Π = failure then return failure
    return Π ∪ π_i
  else
    select any p ∈ g
    resolvers ← {a ∈ A_i | p ∈ effects⁺(a) and ∀b ∈ π_i : (a, b) ∉ μA_i}
    if resolvers = ∅ return failure
    nondetrministically choose a resolver a for p
    return GP-SEARCH(G, g − effects⁺(a), π_i ∪ {a}, i)
```

Figure 1.9: GP-Search.

- G è il grafo di pianificazione.
- g è l'insieme di sottogoal ancora da risolvere al livello i.
- π_i è il piano in costruzione al livello i.

Considerazioni su `extract-solution`:

-

1.4.2 Euristiche

2
CLIPS