Planning

Il Situation Calculus

Il Situation Calculus è un formalismo logico progettato per rappresentare e ragionare su domini dinamici. Tale formalismo logico rappresenta il cambiamento di scenario come un set di formule F.O.L. (First Order Logic). Gli elementi base del Calculus sono le Azioni, i Fluent (descrivono lo stato del mondo) e la Situazione. Un dominio è formalizzato da un numero di formule chiamate:

1. Action Precondition Axioms, una per ogni azione.
2. Successor State Axioms, uno per ogni Fluent.
3. Axioms che descrivono l'ambiente nelle varie situazioni.
4. Foundational Axioms del Situation Calculus.

Un certo numero di oggetti sono anche tipicamente coinvolti nella descrizione del mondo. Il Situation Calculus si basa su un dominio ordinato in tre tipi: azioni, situazioni e oggetti, dove gli oggetti includono tutto ciò che non è un'azione o una situazione. Le variabili di ogni tipo possono essere usate. Mentre le azioni, situazioni e oggetti sono elementi del dominio, i Fluent sono modellati come predicati o funzioni.
La Situazione nel Situation Calculus rappresenta la storia delle azioni eseguite. La nuova Situazione risultate dall'esecuzione di una azione è espressa tramite la funzione do().
Un Fluent sono predicati che prendono una Situazione come loro finale argomento e ritornano una Situazione che dipende dall'azione eseguita.
The Foundational Axioms formalizzano l'idea che le Situazioni sono storie avendo: do(a,s)=do(a',s') iff a=a' and s=s'.
Un problema che è legato in generale a tutti i linguaggi che hanno a che fare con una variazione di stato è il problema del Frame. Per gli oggetti che sono coinvolti nell'azione si sa cosa è vero o non lo è più negli stati successivi, ma cosa succede agli oggetti che non sono coinvolti nell'azione (si trovano tra virgolette sullo "sfondo o frame" rispetto all'azione)? Non si ha informazione negli stati successivi per questi oggetti.
Bisogna ricordarsi degli oggetti che non sono coinvolti nell'azione che fa cambiare lo stato. Un modo per risolvere questo problema è aggiungere anche gli assiomi del frame (positivo e negativo). Si ha un assioma del frame positivo e negativo per ogni possibile coppia azione fluente, perché le informazioni sullo stato sono nei fluenti e chi fa cambiare lo stato sono le azioni, quindi ogni volta che si compie un'azione bisogna dire che cosa succede al fluente, per tutte le azioni. Un'altro problema è quello della Ramificazione il quale si occupa delle conseguenze indirette di un azione.
Il Domain nel Situation Calculus viene descritto con la "Theory of Actions":

1. Unique Name Assumptions (UNA), ogni azione ha un nome unico.
2. Descrizione del Domain nello stato iniziale.
3. Precondiction Axioms: descrivono le condizioni che rendono un azione eseguibile, un Axioms per ogni Azione
4. Successors State Axioms: descrivono le leggi causali, un Axioms per ogni fluent.

L'approccio deduttivo al Planning garantisce correttezza e completezza:
1. Se un esiste un Plan per il goal, allora il Plan esiste.
2. Se una sequenza di azioni è intuita, allora è un Plan.

Correttezza e completezza non eliminano la possibilità di dedurre i Plans che contengono i cicli o che sono al contrario non ottimali.

Assunzioni del Classical Planning

1. L'Ambiente è deterministico
2. L'Ambiente è osservabile
3. L'Ambiente è statico (risponde solo alle azioni dell'Agente)

Differenze tra Search e Planning

Per Planning si intende il processo di calcolare diversi step di una procedura di risoluzione di un problema prima di eseguirli. Lo stesso problema può essere risolto con il metodo Search. La principale differenza tra Search e Planning è come vengono rappresentati gli stati. Nel Search gli stati vengono rappresentati come entità singole (atomic), nel Planning invece gli stati hanno una rappresentazione strutturata (costrutti logici). Le azioni nel Search sono fatte con il codice mentre nel Planning sono un insieme di precondizioni ed effetti.
Il goal è sempre codice nel Search mentre nel Planning sono ancora costrutti logici. Il Plan nel Search è una sequenza di azione dallo stato iniziale mentre nel Planning sono vincoli sulle azioni.

STRIPS (Standford Research Institute Problem Solver)

STRIPS è un sistema per la risoluzione dei problemi nello spazio basato sulla logica proposizionale con l'aggiunta di predicati. Ogni stato del sistema, compreso quello attuale e quello finale, è rappresentato da una congiunzione di simboli nella logica STRIPS. Lo stato del sistema è descritto da una serie di predicati che possono essere veri o falsi come previsto dalla logica proposizionale ma è aggiunta la possibilità di aggiungere oggetti e funzioni che diano un risultato booleano in base agli oggetti ricevuti in input.
Dunque, STRIPS è un metodo per semplificare la formalizzazione dei problemi.
L'Agente è in grado di compiere un certo numero di azioni che manipolano lo stato del sistema: ognuna di queste operazioni ha dei prerequisiti e degli effetti sul sistema che sono noti a priori e codificati nel linguaggio STRIPS.
Il risolutore STRIPS esplora lo stato degli spazi possibili esplorando quali configurazioni si otterrebbero applicando un certo stato tutte le azioni possibili per quello stato applicandole al modello e ripetendo in modo ricorsivo le azioni per tutti gli stati provati finché uno di essi non corrisponde allo stato finale. In questo modo STRIPS calcola la sequenza di azioni da fare per raggiungere il goal dallo stato iniziale passando se necessario dagli stati intermedi; altrimenti dimostra che il goal è irraggiungibile.
Il calcolo è effettuato con complessità polinomiale per lo spazio ma complessità esponenziale per il tempo.
Il problema di decidere se esiste un piano per un istanza STRIPS è PSPACE(Polinomial-Space), ovvero tutti quei problemi che possono essere risolti da un algoritmo che utilizzi

uno spazio di memoria la cui dimensione sia al più funzione polinomiale della dimensione dell'input.

STRIPS è efficace in tutti quei casi in cui il problema e l'ambiente siano noti a priori con certezza ma ha alcune limitazioni:

1. Impossibilità di gestire differenze tra modello e ambiente.
2. Le modifiche al sistema avvengono sempre per effetto del robot, quindi è inadatto alla cooperazione.
3. Aumentando la complessità del modello, la quantità di tempo necessario a risolvere il problema cresce esponenzialmente.

Differenze tra State-Space e Plan-Space

Gli algoritmi State-Space Planning cercano attraverso lo spazio degli stati raggiungibili del mondo un Path che risolve il problema. Questi algoritmi possono essere basati su Progression: un forward cerca dallo stato iniziale allo stato finale, oppure, possono essere basati su Regression: un backward cerca dallo stato finale verso lo stato iniziale.
STRIPS è un algoritmo incompleto basato su Regression.
I Planners del Plan-space cercano attraverso lo spazio di Plans parziali, i quali sono insiemi di azioni che potrebbero non essere totalmente ordinate.
La Regression e la Progression sono entrambi insoddisfacenti, infatti è consigliabile utilizzare una buona euristica che si basa su una buona stima della distanza dalla meta.
Si ha buona euristica quando:

1. Il problema viene rilassato, cioè si aumenta il numero di archi nello state space. Si hanno euristiche ammissibili come problemi semplificati rimuovendo le precondizioni e gli effetti negati.
2. Si assume l'indipendenza del subgoal, cioè astraendo insiemi di stati, il costo di risolvere un goal è la somma dei costi per risolvere ciascuno dei subgoal (quando pessimistica non è ammissibile).

Planning Graph

GraphPlan

Graphplan è un algoritmo per l'Automated Planning che prende in input un Planning problem espresso in STRIPS e produce, se il problema è risolvibile, una sequenza di operazioni per il raggiungimento dello stato finale.
Il nome Graphplan è dovuto all'uso di un novel planning graph per ridurre la quantità di ricerca necessaria per trovare la soluzione attraverso l'esplorazione semplice dello state space graph. Nello state space graph, i nodi sono raggiungibili, ed i bordi indicano la raggiungibilità attraverso una certa azione.
Al contrario, nello state space graph di Graphplan: i nodi sono azioni e fatti atomici, disposti in livelli alternati, ed i bordi sono di due tipi: il primo tipo permette di andare da un fatto atomico verso le azioni per le quali è una condizione oppure da un'azione ai fatti atomici che li rende TRUE o FALSE. Il primo livello contiene fatti atomici TRUE che identificano lo stato iniziale. Liste di fatti incompatibili che non possono essere vere contemporaneamente

e azioni incompatibili che non possono essere eseguite insieme sono comunque mantenute. L'algoritmo poi estende in modo iterativo il planning graph, dimostrando che non esistono soluzioni di lunghezza L-1 prima di cercare per i piani di lunghezza L attraverso un concatenamento all'indietro: supponendo che gli obiettivi sono TRUE, Graphplan cerca le azioni e gli stati precedenti da cui gli obiettivi possono essere raggiunti, escludendo il numero maggiore di azioni incompatibili.
I Mutex links possono essere di due tipi:

1. I conflitti tra le azioni che altro volta producono: inconsistenza (effetti di un azione sono in conflitto con gli effetti di un altra azione), interferenze (effetti di un azione sono in conflitto con le precondizioni di un altra azione), Competing needs(la precondizione di un'azione è mutua esclusione con la precondizione di un altra azione).
2. I conflitti tra letterali che si verificano quando abbiamo letterali negativi e inconsistent support (due letterali si escludono a vicenda se ogni coppia di azioni che essi possono raggiungere è mutualmente esclusiva).

SatPlan

Un approccio strettamente legato al Planning è il Planning as Satisfiability (Satplan) il quale riduce il problema di automated planning. Esso converte una Planning problem instance in una istance del Boolean Satisfiability Problem, che viene poi risolto utilizzando un metodo per stabilire la soddisfacibilità come il DPLL o WalkSAT. Dato un problem instance nella progettazione, con un dato stato iniziale, un dato insieme di azioni, un obiettivo, e la lunghezza del goal, una formula viene generata in modo tale che la formula è soddisfacibile se e solo se esiste un Plan con la data lunghezza goal. Questo è simile alla simulazione della macchina di Turing con il problema della soddisfacibilità nella dimostrazione del teorema di Cook. Un Plan può essere trovato testando la soddisfacibilità delle formule per le diverse lunghezze del goal. Il modo più semplice per farlo è passare attraverso goal di lunghezza sequenzialmente, 0, 1, 2, e così via.

Partial Order Planning

Partial Order Planing è un approccio alla pianificazione che lascia le decisioni circa l'ordine degli interventi il più aperto possibile. È in contrasto con la Total Order Planning, che produce un ordinamento esatto di azioni. Dato un problema in cui è richiesta una sequenza di azioni per raggiungere un obiettivo, un POP specifica tutte le azioni che devono essere prese, ma specifica un ordinamento delle azioni solo dove necessario (Principio di Least Commitment).
Un POP è costituito da quattro componenti:

1. Un set di azioni
2. Un partial order per le azioni. Esso specifica le condizioni sull'ordine di alcune azioni.
3. Un set di Causal links, specifica quali azioni incontrano quali precondizioni di altre azioni. In alternativa, un insieme di associazioni tra le variabili nelle azioni.
4. Un set di precondizioni aperte, specifica quali condizioni non sono soddisfatte da qualsiasi azione nel POP.

HTN

In intelligenza artificiale, HTN, è un approccio all'automated planning in cui la dipendenza tra le azioni può essere data sotto forma di networks. L'idea base di HTN è creare una decomposizione gerarchica tale che gli operatori del classical planning sono preservati, e viene introdotto un concetto più astratto di azione: Hight Level Action. Il planning procede decomponendo tasks non primitivi ricorsivamente in tasks sempre più piccoli, finché non vengono ottenuti task primitivi. Con questo tipo di approccio però, nei real problem, la ricerca al livello delle azioni primitive può essere molto inefficiente.
In HTN abbiamo due tipi di azioni:

1. Azioni primitive che, come nel classical planning, possono essere eseguite dall'agente.
2. HLA: non possono essere eseguite direttamente dall'agente, ma possono avere uno o più refinements nelle sequenza di azioni.

Queste sequenze di azioni possono contenere sia HLA che primitive. I refinements di un HLA contengono solo primitive e sono chiamati implementazioni di HLA. Un implementazione di un Hight Level Plan è la concatenazione dell'implementazione di ogni HLA in sequenza. Date le definizioni di precondizione-effetto di ogni primitiva, si può determinare se ogni data implementazione di HLP raggiunge il goal. Un HLP raggiunge il goal se almeno una delle sue implementazioni può raggiungere il goal.
Nel caso base, se un HLA ha esattamente una sola implementazione, allora possiamo definire le precondizioni-effetti dell'HLA esattamente com se fosse una primitiva. Se invece ho differenti implementazioni, il Plan cerca tra le varie implementazioni (primitive) e cerca un Plan ragionando sulle HLA.
La ricerca gerarchica perfeziona le HLA dalla sua definizione astratta alla sua sequenza di primitive per determinare se un Plan funzionante.
Ripetutamente viene scelta un HLA nell Plan corrente e lo sostituisce con una delle sue implementazioni finché non viene raggiunto il goal. Il controllo sul raggiungimento del goal può essere effettuato quando il Plan è ridefinito come una sequenza di azioni primitive.
Per modellare gli effetti di HLA con diverse implementazioni, si utilizza un approccio conservativo: si includono solo gli effetti positivi che sono raggiunti da ogni implementazione dell'HLA e gli effetti negativi di qualsiasi implementazione.

HTN Reachable Sets

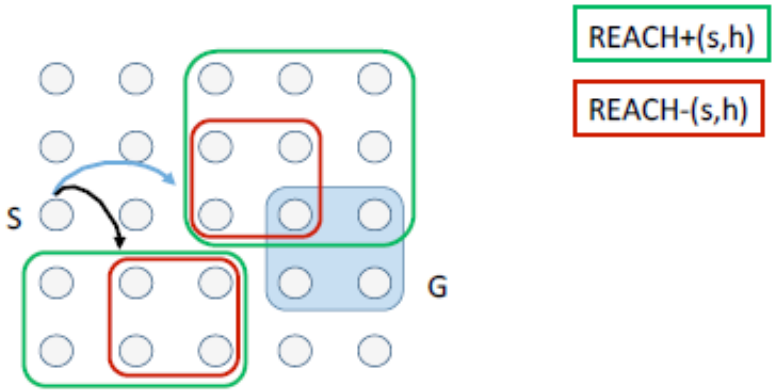
REACH(s,h) di un HLA h da uno stato s, è una funzione che rappresenta il set di stati raggiungibili da ogni implementazione dell'HLA. Il set di stati raggiungibili di una sequenza di HLA è l'unione di tutti i set raggiungibili ottenuti applicando l'n-HLA in ogni stato appartenente al set raggiungibile dell'(n-1)HLA.
Data una sequenza di HLA, si raggiunge il goal se il set di stati raggiungibili da questa sequenza interseca il set dei possibili goal state. (no intersezione = no Plan)
Come raggiunge HLA il goal?

1. Cerca attraverso HLP.
2. Se uno degli stati raggiungibili interseca il goal, memorizzalo all'interno dell'abstract Plan e return YES.
3. Altrimenti ridefinisce il piano ulteriormente. Se questa operazione return YES, allora tutto ok, altrimenti return NO.

Le azioni primitive possono aggiungere, cancellare, lasciare invariate alcune variabili. Un HLA può controllare il valore di una variabile tramite i simboli: +, -, ~, ±.
Reach(s,h) può essere ridefinito con effetti approssimativi:

1. In una descrizione ottimistica sopravvaluta il set di stati raggiungibili(REACH+(s,h)), gli effetti possibili come defunti.
2. In una descrizione pessimistica sottovaluta il set (REACH-(s,h)), gli effetti possibili non avvengono.

Se l'optimistic reachable set non interseca il goal allora il Plan non funziona.
Se il pessimistic reachable set interseca il goal allora il Plan funziona.



Questa analisi non è sempre vera come mostrata nella figura seguente.