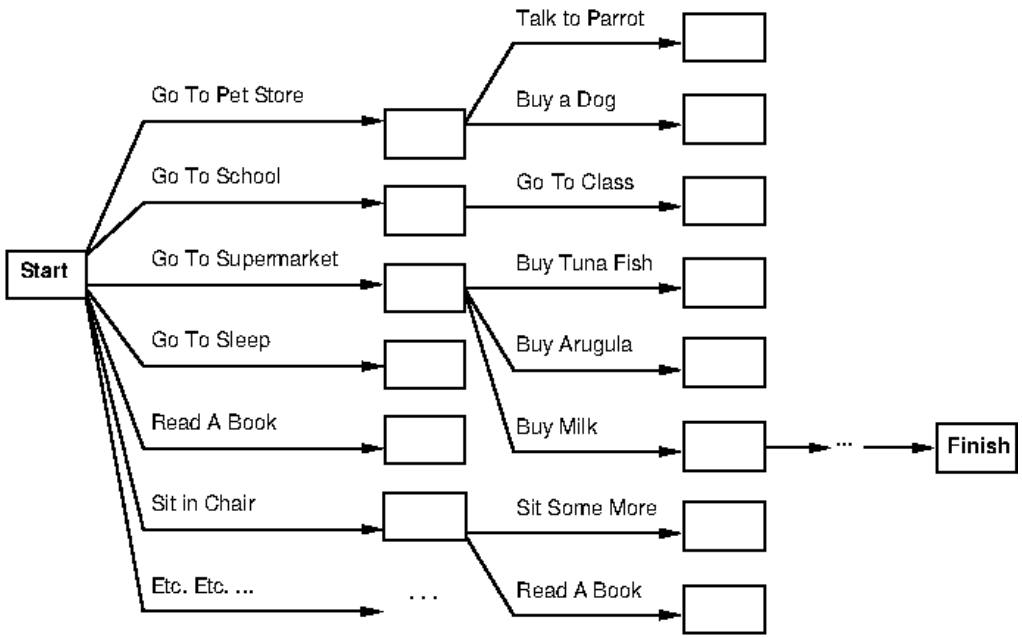
Planificare

IA 2022/2023

Conținut

Planificare

- Planificarea reprezintă alegerea unei succesiuni de acțiuni pentru atingerea unui obiectiv
- Similară cu rezolvarea problemelor de căutare
  Algoritmii de căutare eșuează pe probleme de tipul *get milk, bananas, and a cordless drill* (foarte multe acțiuni irelevante)



Planificare

- Planificare = Reprezentare logică + Rezolvarea problemelor
- Căutare vs. Planificare
  Similare: construim planuri pentru atingerea obiectivelor și apoi le executăm

  Diferite datorită:
  - reprezentării stărilor, acțiunilor, obiectivului
  - utilizarea reprezentărilor logice
  - modalității de construcție a soluțiilor

Conținut

Reprezentarea problemelor de planificare

- Algoritmii de planificare utilizează structura logică a problemei
- Limbaje STRIPS (*STanford Research Institute Problem Solver*)
  - stare: conjuncție de literali pozitivi, propoziționali și de ordin I
    Exemple: $Poor \wedge Unknown$
    $At(Plane_1, Melbourne) \wedge At(Plane_2, Sydney)$

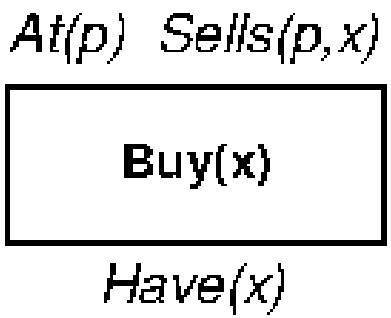
    Presupunerea lumii închise: condițiile care nu sunt menționate într-o stare sunt considerate false.

Reprezentarea problemelor de planificare

- obiectiv: o conjuncție de literali pozitivi
  Exemple: $Rich \wedge Famous$
  $At(P_2, Tahiti)$
  - o stare s **satisface** obiectivul g dacă s conține toți atomii din g
    Exemplu: $Rich \wedge Famous \wedge Miserable$ satisface scopul
    $Rich \wedge Famous$
- acțiuni
  - **precondiții** care trebuie satisfacute înainte de a fi executate
  - **efectele** care rezultă atunci când acțiunile sunt executate

STRIPS

Exemplu:
Acțiune: $Buy(x)$
Precondiție: $At(p), Sells(p, x)$
Efect: $Have(x)$



Limbaj restrâns $\Longrightarrow$ algoritmi eficienți

Reprezentarea acțiunilor

Exemplu: zborul unui avion
ACȚIUNE: $Fly(p, from, to)$,
PRECONDIȚIE: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFECT: $\neg At(p, from) \wedge At(p, to)$

Schema de acțiune reprezintă un număr de acțiuni diferite care pot fi derivate instanțiind variabilele p, $from$, to;
Include:

- numele acțiunii și o listă de parametri
- **precondiție**: conjuncție de literali pozitivi
- **efect**: conjuncție de literali
  - lista de adăugare (literali pozitivi), lista de ștergere (literali negativi)

Reprezentarea problemelor

O acțiune este aplicabilă în orice stare care satisface precondiția.

Rezultatul execuției unei acțiuni a în s este starea s', identică cu s, cu excepția: literalii pozitivi P din efect sunt adăugați la s' și cei negativi $\neg P$ sunt eliminați din s'.

Exemplu:
Starea curentă: $At(P1, JFK) \wedge At(P2, SFO) \wedge Plane(P1) \wedge Plane(P2) \wedge Airport(JFK) \wedge Airport(SFO)$

Aceasta satisface precondiția acțiunii Fly cu substituția $\{p/P_1, from/JFK, to/SFO\}$.

Aplicând $Fly(P1, JFK, SFO)$, starea curentă devine $At(P1, SFO) \wedge At(P2, SFO) \wedge Plane(P1) \wedge Plane(P2) \wedge Airport(JFK) \wedge Airport(SFO)$.

Reprezentarea problemelor

Presupunerea STRIPS: literalii care nu sunt menționați în efect rămân neschimbați.

Soluție: o **secvență de acțiuni** care executate din starea inițială rezultă într-o stare care satisface obiectivul.

Limbaje pentru reprezentare

STRIPS vs. *Action Description Language (ADL)*

STRIPS **Language**	ADL **Language**
Only positive literals in states: $Poor \wedge Unknown$	Positive and negative literals in states: $\neg Rich \wedge \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \wedge \neg Q$ means add P and delete Q.	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \wedge Famous$	Quantified variables in goals: $\exists x At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
Effects are conjunctions.	Conditional effects allowed: **when** P: E means E is an effect only if P is satisfied.
No support for equality.	Equality predicate ($x = y$) is built in.
No support for types.	Variables can have types, as in ($p : Plane$).

STRIPS vs. ADL

Acțiune: $Fly(p, from, to)$,
PRECONDIȚIE: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFECT: $\neg At(p, from) \wedge At(p, to)$

vs.

ACȚIUNE: $Fly(p : Plane, from : Airport, to : Airport)$,
PRECONDIȚIE: $At(p, from) \wedge (from \neq to)$
EFECT: $\neg At(p, from) \wedge At(p, to)$

Notația $p : Plane$ este o abreviere pentru $Plane(p)$; precondiția $(from \neq to)$

Exemplu STRIPS: transportul mărfurilor cu avionul

Predicate: $In(c, p)$ - marfa c este în avionul p, $At(x, a)$ - obiectul x este în aeroportul a.
Acțiuni: $Load$, $Unload$, Fly.

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Soluție: $[Load(C1, P1, SFO), Fly(P1, SFO, JFK),$
$Load(C2, P2, JFK), Fly(P2, JFK, SFO)]$
Problemă: un avion poate zbura spre și dinspre același aeroport

Exemplu ADL: schimbarea anvelopei

Acțiuni: scoate rezerva, scoate anvelopa, așează anvelopa de rezervă, lasă mașina nesupravegheată peste noapte.

$$
\begin{aligned}
&Init(At(Flat, Axle) \wedge At(Spare, Trunk))\\
&Goal(At(Spare, Axle))\\
&Action(Remove(Spare, Trunk),\\
&\quad \text{PRECOND: } At(Spare, Trunk)\\
&\quad \text{EFFECT: } \neg At(Spare, Trunk) \wedge At(Spare, Ground))\\
&Action(Remove(Flat, Axle),\\
&\quad \text{PRECOND: } At(Flat, Axle)\\
&\quad \text{EFFECT: } \neg At(Flat, Axle) \wedge At(Flat, Ground))\\
&Action(PutOn(Spare, Axle),\\
&\quad \text{PRECOND: } At(Spare, Ground) \wedge \neg At(Flat, Axle)\\
&\quad \text{EFFECT: } \neg At(Spare, Ground) \wedge At(Spare, Axle))\\
&Action(LeaveOvernight,\\
&\quad \text{PRECOND:}\\
&\quad \text{EFFECT: } \neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)\\
&\qquad \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))
\end{aligned}
$$

Utilizează o precondiție negată $\neg At(Flat, Axle)$

Exemplu ADL: lumea blocurilor

O mulțime de blocuri (cuburi) așezate pe masă; un braț al robotului poate ridica și muta un bloc; scopul: a construi una sau mai multe stive de blocuri

$$
\begin{aligned}
&Init(On(A, Table) \land On(B, Table) \land On(C, Table) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \\
&\quad \land Clear(A) \land Clear(B) \land Clear(C)) \\
&Goal(On(A, B) \land On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \textsc{Precond}: On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land \\
&\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
&\quad \textsc{Effect}: On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \textsc{Precond}: On(b, x) \land Clear(b) \land Block(b) \land (b \neq x), \\
&\quad \textsc{Effect}: On(b, Table) \land Clear(x) \land \neg On(b, x))
\end{aligned}
$$

$MoveToTable(b, x)$ mută blocul b de pe x pe tablă.
Soluție: $[Move(B, Table, C), Move(A, Table, B)]$.

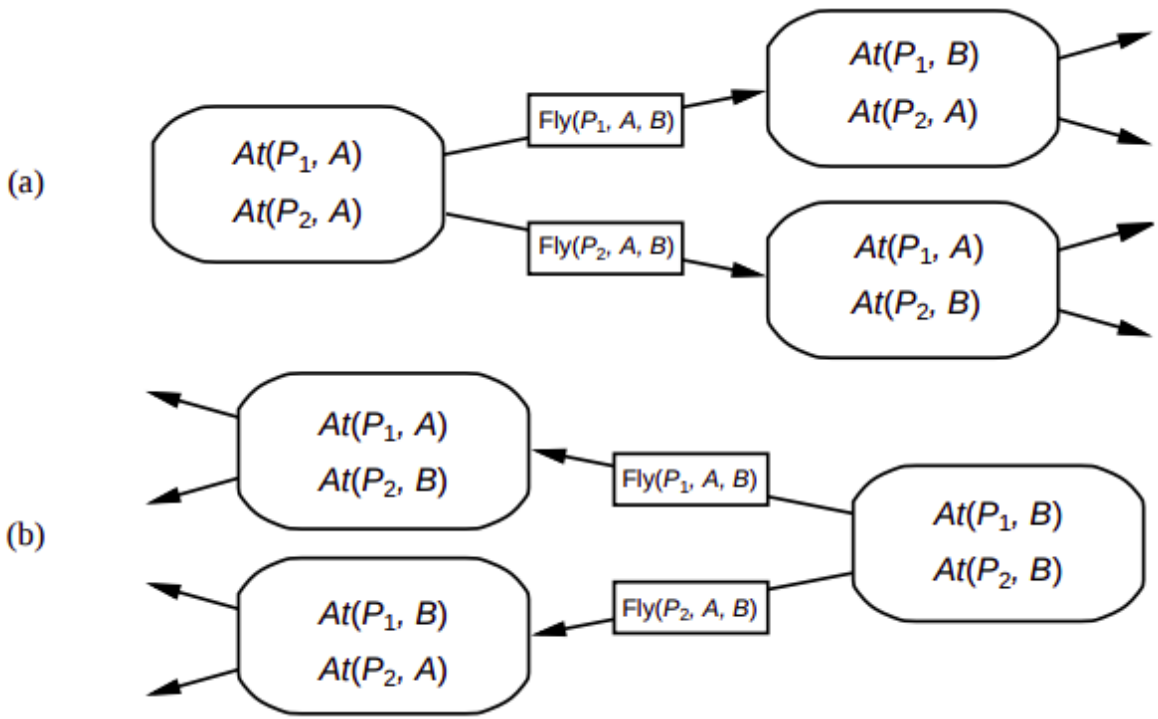
Editor PDDL `http://editor.planning.domains/`

Conținut

Algoritmi de căutare

Căutarea e posibilă

- fie înainte (*progression*), de la starea inițială, utilizând acțiunile
- fie înapoi (*regression*), de la obiectiv.



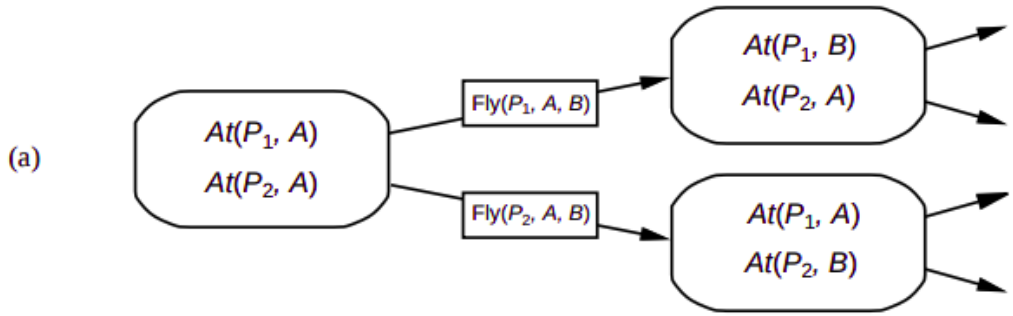
Căutare înainte (*progression*)

Formularea problemelor de planificare ca probleme de căutare:

- starea inițială
  O stare este o mulțime de literali pozitivi (cei care nu apar sunt falși).
- acțiunile aplicabile unei stări: cele pentru care precondițiile sunt satisfăcute;
  starea succesoare este generată adăugând literalii cu efect pozitiv și ștergându-i pe cei cu efect negativ;
- testarea obiectivului: starea satisface obiectivul
- costul unui pas este 1

În absența simbolurilor funcționale, spațiul stărilor este finit. Se poate utiliza orice algoritm de căutare pe grafuri, care este complet (exemplu: A*).

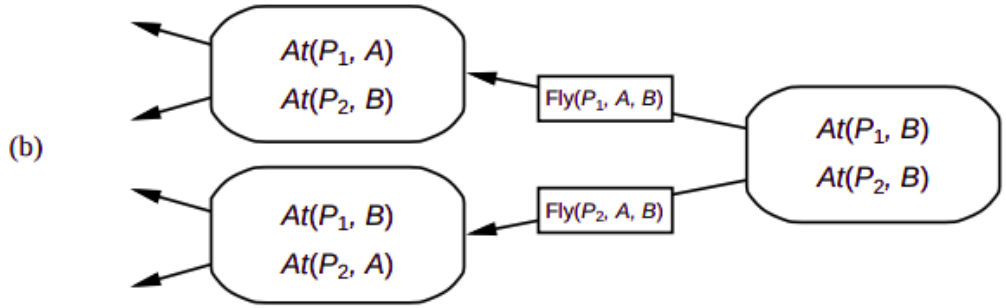
Căutare înainte



- sunt considerate toate acțiunile aplicabile din fiecare stare; acțiuni irelevante
  Exemplu: 10 aeroporturi, fiecare cu 5 avioane și 20 containere
  Scop: mutarea containerelor de pe aeroportul A pe B
  - factorul de ramificare: 10 x 5 x 20 $\rightarrow$ $b = 1000$
  - adâncimea soluției: 20 încărcări + 1 zbor + 20 descărcări $\rightarrow$ $d = 41$
  - $1000^{41} (= 10^{123})$ noduri

- e nevoie de euristici bune

Căutare înapoi (*regression planning*)



Avantaj: permite considerarea doar a acțiunilor relevante.

- O acțiune este relevantă pentru un scop (conjuncție) dacă unul din termenii conjuncției este adevărat.

  Exemplu: 20 de acțiuni din starea inițială în loc de 1000
  Scopul $At(C_1, B) \wedge At(C_2, B) \wedge ... \wedge At(C_{20}, B)$.
  Pentru termenul $At(C_1, B)$, putem căuta acțiuni care au termenul ca efect; ex: $Unload(C_1, p, B)$.

- Restricția pentru acțiuni relevante: căutarea înapoi are un factor de ramificare mai mic decât căutarea înainte.

Căutare înapoi

Care sunt stările care satisfac scopul, aplicând o anumită acțiune?

- Regresia obiectivului prin acțiuni
  Acțiunea relevantă $Unload(C_1, p, B)$ poate fi aplicată dacă precondițiile sunt satisfăcute $\rightarrow$ orice predecesor trebuie să includă precondițiile $In(C_1, p) \wedge At(p, B)$.
  Starea predecesor:
  $In(C_1, p) \wedge At(p, B) \wedge At(C_2, B) \wedge ... \wedge At(C_{20}, B)$.

- Acțiune consistentă: nu modifică un literal.

  Exemplu: acțiunea $Load(C_2, p, B)$ nu este consistentă cu scopul deoarece neagă literalul $At(C_2, B)$.

Căutare înapoi

- Construirea predecesorilor
  Dată o descriere a scopului g, a o acțiune relevantă și consistentă
  - sunt eliminate efectele pozitive ale acțiunii a care apar în g
  - sunt adăugați literalii precondiție ai acțiunii a

- Terminarea algoritmului: când este generată o descriere a predecesorului care este satisfăcută de starea inițială.

Euristici

Euristica estimează distanța de la stare la scop. Distanța: numărul de acțiuni necesare.

1. derivăm problema relaxată, soluția acesteia fiind o euristică admisibilă

2. *divide et impera*
   Ipoteza independenței subobiectivelor: costul rezolvării unei conjuncții de subobiective este aproximativ egal cu suma costurilor rezolvării *independente* a subobiectivelor

Euristici

- relaxăm problema prin **eliminarea precondițiilor** din acțiuni; euristica: # de scopuri nesatisfăcute

- relaxăm problema prin **eliminarea efectelor negative** euristica: # min de acțiuni necesare a.i. reuniunea efectelor pozitive ale acțiunilor satisface scopul

  Exemplu: $Goal(A \wedge B \wedge C)$
  $Action(X, EFFECT : A \wedge P)$
  $Action(Y, EFFECT : B \wedge C \wedge Q)$
  $Action(Z, EFFECT : B \wedge P \wedge Q)$
  Mulțimea minimală de acoperire a scopului $\{A, B, C\}$ este dată de acțiunile $\{X, Y\}$, deci euristica returnează costul 2.

Euristici

- *empty-delete-list*: relaxăm problema prin **eliminarea efectelor negative, fără a elimina precondițiile**
  (dacă o acțiune are efectul $A \wedge \neg B$, va avea efectul A și în problema relaxată)
  - euristica presupune execuția unui algoritm (simplu) de planificare; în practică, căutarea în problema relaxată este rapidă

Algoritmul Fast-Forward

Algoritmul FF (Fast-Forward) = căutare înainte + euristici

- euristica *empty-delete-list* ca estimare
- folosește o strategie *Local Search*, în stările intermediare folosește căutarea în lățime (BFS) pentru a determina un succesor mai bun
- dacă nu găsește o soluție, aplică A* din starea inițială

Conținut

Planificare cu ordine parțială

- Planificare cu ordine totală (progresie și regresie)
  - nu utilizează descompunerea problemei
  - identifică simultan secvențele de acțiuni pentru subprobleme

- Este mai eficientă rezolvarea independentă a subobiectivelor și apoi combinarea subplanurilor. Putem lucra întâi cu deciziile importante/evidente.

Planificare cu ordine parțială

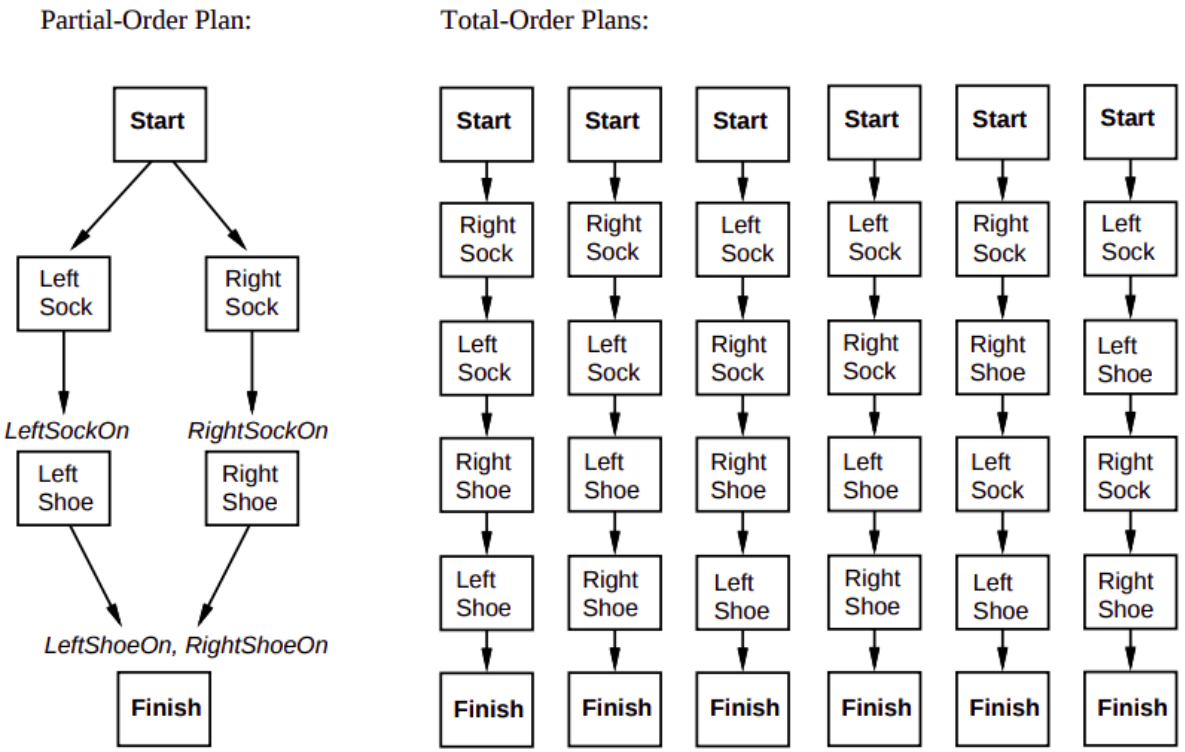
Strategia *Least commitment*: întârzierea unei alegeri

Exemplu: încălțarea unei perechi de pantofi
$Goal(RightShoeOn \wedge LeftShoeOn)$
$Init()$
$Action(RightShoe, PRECOND : RightSockOn, EFFECT : RightShoeOn)$
$Action(RightSock, EFFECT : RightSockOn)$
$Action(LeftShoe, PRECOND : LeftSockOn, EFFECT : LeftShoeOn)$
$Action(LeftSock, EFFECT : LeftSockOn)$

Pentru a obține primul termen din conjuncție: secvența de acțiuni $[RightSock, RightShoe]$; similar pentru cel de-al doilea termen.

Planificare cu ordine parțială

Planificator cu ordine parțială: un algoritm de planificare care poate plasa două acțiuni într-un plan, *fără a specifica ordinea*



Soluția este reprezentată ca un graf de acțiuni.
Soluția corespunde celor șase planuri posibile cu ordine totală (**liniarizarea** planului cu ordine parțială).

Planificare cu ordine parțială

Căutarea în spațiul planurilor cu ordine parțială

- un plan vid
- rafinarea planului: adăugarea unui pas planului, ordonarea acțiunilor

Componentele unui plan

- acțiuni (pașii planului); un plan vid conține doar acțiunile *Start* și *Finish*

- constrângeri de ordonare $A \prec B$ (A înaintea lui B)
  - Un ciclu $A \prec B$ și $B \prec A$ reprezintă o contradicție.

- legături cauzale $A \rightarrow^{p} B$ (A realizează p pentru B; A, B acțiuni)
  Exemplu: $RightSock \rightarrow^{RightSockOn} RightShoe$
  RightSockOn este un efect al acțiunii *RightSock* și o precondiție a *RightShoe*;
  RightSockOn trebuie să rămână adevarat între cele două acțiuni (planul nu poate fi extins adăugând o nouă acțiune C, conflictuală cu legătura cauzală).

- precondiții deschise: nu sunt îndeplinite de nici o acțiune

Componentele unui plan: exemplu

Acțiuni: $\{RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish\}$
Ordonări: $\{RightSock \prec RightShoe, LeftSock \prec LeftShoe\}$
Legături:
$\{RightSock \rightarrow^{RightSockOn} RightShoe, LeftSock \rightarrow^{LeftSockOn} LeftShoe, RightShoe \rightarrow^{RightShoeOn} Finish, LeftShoe \rightarrow^{LeftShoeOn} Finish\}$
Precondiții deschise: $\{\}$

Planificare cu ordine parțială

- Într-un plan consistent nu există cicluri în constrângerile de ordonare și nici conflicte în legăturile cauzale.
- Soluție: un plan consistent fără precondiții deschise.
- Un plan cu ordine parțială este executat alegând *oricare* din acțiunile posibile următoare.

Problema de căutare

- Planul inițial conține *Start* și *Finish*, constrângerea de ordine $Start \prec Finish$, nu conține legături cauzale și toate precondițiile din *Finish* sunt deschise.

- Funcția succesor alege o precondiție deschisă p pentru acțiunea B și generează un plan pentru fiecare modalitate consistentă de alegere a acțiunii A care obține p
  - adaugă legătura $A \rightarrow^p B$, $A \prec B$; dacă A este o acțiune nouă, adaugă $Start \prec A$, $A \prec Finish$
  - rezolvăm conflictele între legătura cauzală și acțiunile existente
    - conflictul între $A \rightarrow^p B$ și C este rezolvat dacă C apare în afara intervalului: adăugăm $B \prec C$ sau $C \prec A$; adăugăm stări succesoare

- Testarea obiectivului: verifică dacă un plan este soluție (nu există precondiții deschise)

Exemplu

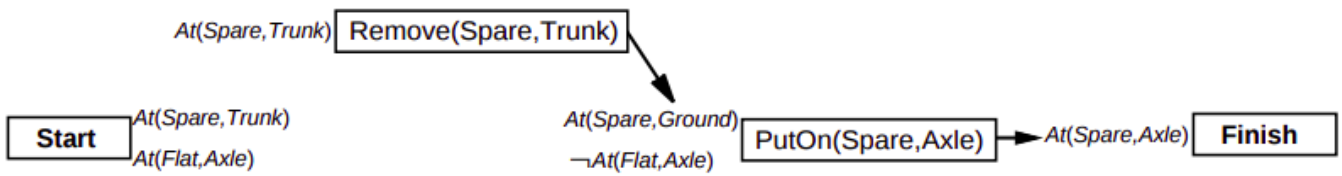
$$
\begin{aligned}
&Init(At(Flat, Axle) \wedge At(Spare, Trunk))\\
&Goal(At(Spare, Axle))\\
&Action(Remove(Spare, Trunk),\\
&\quad \text{PRECOND: } At(Spare, Trunk)\\
&\quad \text{EFFECT: } \neg At(Spare, Trunk) \wedge At(Spare, Ground))\\
&Action(Remove(Flat, Axle),\\
&\quad \text{PRECOND: } At(Flat, Axle)\\
&\quad \text{EFFECT: } \neg At(Flat, Axle) \wedge At(Flat, Ground))\\
&Action(PutOn(Spare, Axle),\\
&\quad \text{PRECOND: } At(Spare, Ground) \wedge \neg At(Flat, Axle)\\
&\quad \text{EFFECT: } \neg At(Spare, Ground) \wedge At(Spare, Axle))\\
&Action(LeaveOvernight,\\
&\quad \text{PRECOND:}\\
&\quad \text{EFFECT: } \neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)\\
&\qquad \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))
\end{aligned}
$$

Planul inițial conține acțiunea *Start* cu efectul $At(Flat, Axle) \wedge At(Spare, Trunk)$ și *Finish* cu precondiția $At(Spare, Axle)$.

Exemplu (I)

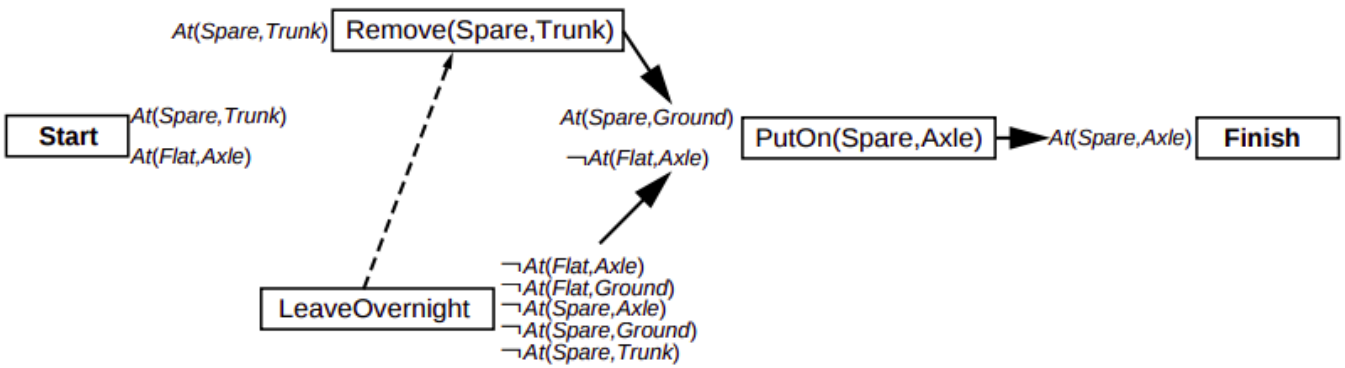
Secvența de evenimente:

- Alege precondiția deschisă $At(Spare, Axle)$. Alege acțiunea $PutOn(Spare, Axle)$
- Alege precondiția $At(Spare, Ground)$ și acțiunea $Remove(Spare, Trunk)$



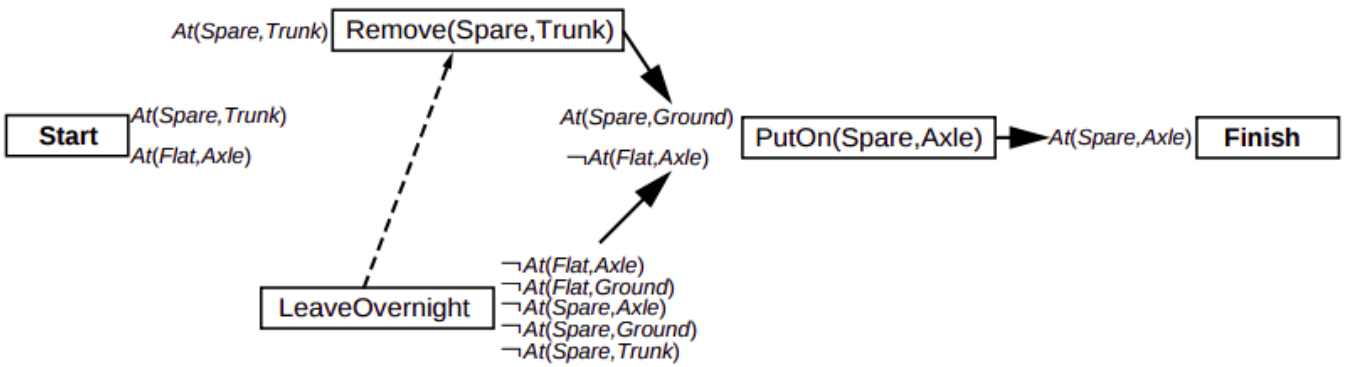
Exemplu (II)

- Alege precondiția $\neg At(Flat, Axle)$.
  Alege acțiunea *LeaveOvernight* care are și efectul $\neg At(Spare, Ground)$ ce intră în conflict cu legătura cauzală $Remove(Spare, Trunk) \rightarrow^{At(Spare,Ground)} PutOn(Spare, Axle)$.
  Pentru a rezolva conflictul, adăugăm o constrângere de ordine: $LeaveOvernight \prec Remove(Spare, Trunk)$



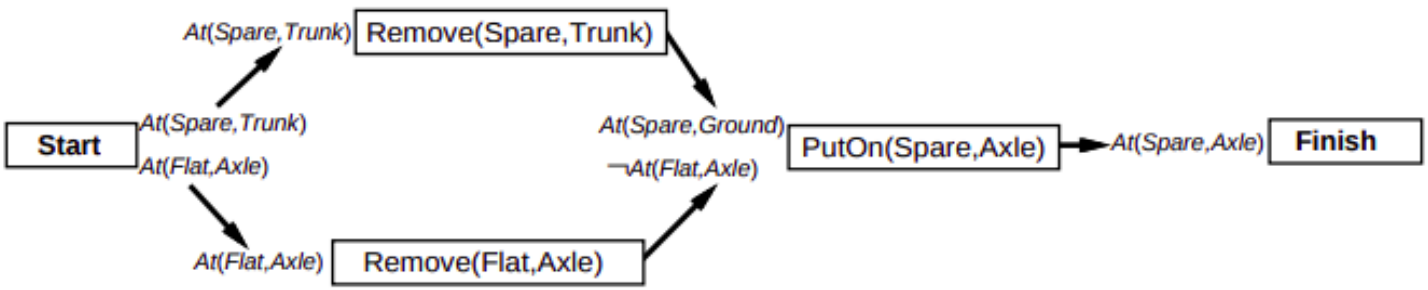
Exemplu (III)

- Singura precondiție deschisă este $At(Spare, Trunk)$. Singura acțiune care o obține este *Start*, însă legătura cauzală de la *Start* la $Remove(Spare, Trunk)$ este în conflict cu efectul $\neg At(Spare, Trunk)$.
  - nu putem rezolva conflictul cu *LeaveOvernight*
  - trebuie să ne întoarcem și să ștergem acțiunea *LeaveOvernight*



Exemplu (IV)

- Alege precondiția $\neg At(Flat, Axle)$. Alege $Remove(Flat, Axle)$.
- Alege precondiția $At(Spare, Trunk)$ și alege *Start* pentru a o îndeplini.
- Alege precondiția $At(Flat, Axle)$ și alege *Start* pentru a o îndeplini.



Planificare cu ordine parțială

Avantaje

- algoritmul este corect, complet
- legăturile cauzale conduc la eliminarea unor porțiuni a spațiului de căutare

Dezavantaje

- e dificil de estimat cât de departe suntem de atingerea obiectivului

Euristici

- # de precondiții deschise
- selectează precondiția deschisă care poate fi satisfăcută în cele mai puține moduri