

Planificare

IA 2023/2024

Planificare

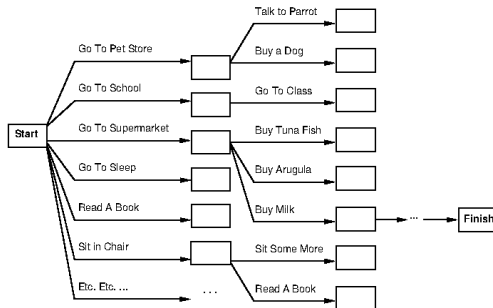
Reprezentarea problemelor de planificare

Algoritmi de căutare

Planificare cu ordine parțială

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 = 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

Planificare

Reprezentarea problemelor de planificare

Algoritmi de căutare

Planificare cu ordine parțială

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

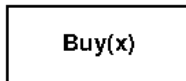
Exemplu:

ACȚIUNE: $Buy(x)$

PRECONDIȚIE: $At(p), Sells(p, x)$

EFFECT: $Have(x)$

$At(p) \quad Sells(p, x)$



$Have(x)$

Limbaaj restrâns \implies algoritmi eficienți

Reprezentarea acțiunilor

Exemplu: zborul unui avion

ACȚIUNE: $Fly(p, from, to)$,

PRECONDIȚIE: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

EFFECT: $\neg At(p, from) \wedge At(p, to)$

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

Include:

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

Reprezentarea problemelor

O acțiune este **aplicabilă** în orice stare care satisface condiția.

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 condiția acțiunii Fly cu substituția $\{p/P1, from/JFK, to/SFO\}$.

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

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.

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: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

EFFECT: $\neg At(p, from) \wedge At(p, to)$

vs.

Acțiune: $Fly(p : Plane, from : Airport, to : Airport)$,

PRECONDIȚIE: $At(p, from) \wedge (from \neq to)$

EFFECT: $\neg At(p, from) \wedge At(p, to)$

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 , $At(x, a)$ - x este în aeroportul a .

Acțiuni: $Load$, $Unload$, Fly .

```
Init( $At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$   
     $\wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$   
     $\wedge Airport(JFK) \wedge Airport(SFO)$ )  
Goal( $At(C_1, JFK) \wedge At(C_2, SFO)$ )  
Action( $Load(c, p, a)$ ,  
    PRECOND:  $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$   
    EFFECT:  $\neg At(c, a) \wedge In(c, p)$ )  
Action( $Unload(c, p, a)$ ,  
    PRECOND:  $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$   
    EFFECT:  $At(c, a) \wedge \neg In(c, p)$ )  
Action( $Fly(p, from, to)$ ,  
    PRECOND:  $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$   
    EFFECT:  $\neg At(p, from) \wedge At(p, to)$ )
```

Soluție: [$Load(C_1, P_1, SFO)$, $Fly(P_1, SFO, JFK)$,
 $Load(C_2, P_2, JFK)$, $Fly(P_2, JFK, SFO)$]

Problemă: un avion poate zbura spre și dinspre același aeroport

Exemplu ADL: schimbarea anvelopei

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

```
Init(At(Flat, Axle)  $\wedge$  At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(Remove(Spare, Trunk),
  PRECOND: At(Spare, Trunk)
  EFFECT:  $\neg$  At(Spare, Trunk)  $\wedge$  At(Spare, Ground))
Action(Remove(Flat, Axle),
  PRECOND: At(Flat, Axle)
  EFFECT:  $\neg$  At(Flat, Axle)  $\wedge$  At(Flat, Ground))
Action(PutOn(Spare, Axle),
  PRECOND: At(Spare, Ground)  $\wedge$   $\neg$  At(Flat, Axle)
  EFFECT:  $\neg$  At(Spare, Ground)  $\wedge$  At(Spare, Axle))
Action(LeaveOvernight,
  PRECOND:
  EFFECT:  $\neg$  At(Spare, Ground)  $\wedge$   $\neg$  At(Spare, Axle)  $\wedge$   $\neg$  At(Spare, Trunk)
          $\wedge$   $\neg$  At(Flat, Ground)  $\wedge$   $\neg$  At(Flat, Axle))
```

Obs: utilizează o condiție 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

```
Init(On(A, Table)  $\wedge$  On(B, Table)  $\wedge$  On(C, Table)  
     $\wedge$  Block(A)  $\wedge$  Block(B)  $\wedge$  Block(C)  
     $\wedge$  Clear(A)  $\wedge$  Clear(B)  $\wedge$  Clear(C))  
Goal(On(A, B)  $\wedge$  On(B, C))  
Action(Move(b, x, y),  
    PRECOND: On(b, x)  $\wedge$  Clear(b)  $\wedge$  Clear(y)  $\wedge$  Block(b)  $\wedge$   
            (b  $\neq$  x)  $\wedge$  (b  $\neq$  y)  $\wedge$  (x  $\neq$  y),  
    EFFECT: On(b, y)  $\wedge$  Clear(x)  $\wedge$   $\neg$  On(b, x)  $\wedge$   $\neg$  Clear(y))  
Action(MoveToTable(b, x),  
    PRECOND: On(b, x)  $\wedge$  Clear(b)  $\wedge$  Block(b)  $\wedge$  (b  $\neq$  x),  
    EFFECT: On(b, Table)  $\wedge$  Clear(x)  $\wedge$   $\neg$  On(b, x))
```

MoveToTable(*b*, *x*) mută blocul *b* de pe *x* pe tablă.

Soluție: [*Move*(*B*, *Table*, *C*), *Move*(*A*, *Table*, *B*)].

Editor PDDL (Planning Domain Definition Language)

<http://editor.planning.domains/>

Planificare

Reprezentarea problemelor de planificare

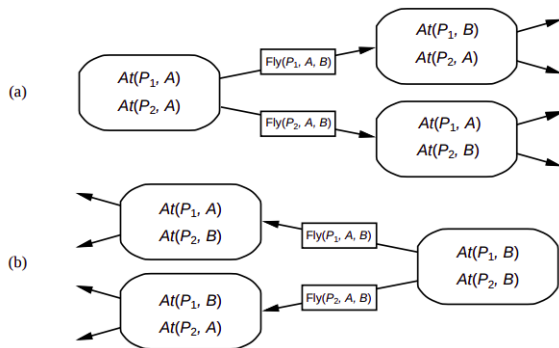
Algoritmi de căutare

Planificare cu ordine parțială

Algoritmi de căutare

Căutarea e posibilă

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



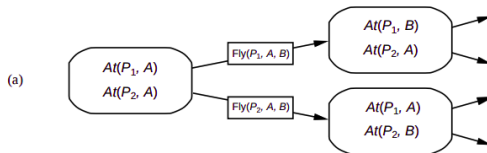
Căutare înainte (*progression*)

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 condițiile 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

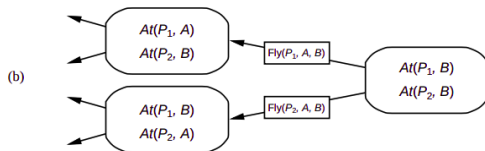
Exemplu: 10 aeroporturi, fiecare cu 5 avioane și 20 containere

Scop: mutarea containerelor de pe aeroportul A pe B

- ▶ factorul de ramificare: $10 \times 5 \times 20 \rightarrow b = 1000$
- ▶ 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 devine adevărat.

Exemplu: 20 de acțiuni din starea inițială în loc de 1000
Scopul $At(C_1, B) \wedge At(C_2, B) \wedge \dots \wedge At(C_{20}, B)$.

Pentru termenul $At(C_1, B)$, putem căuta acțiuni care au termenul ca efect; ex: $Unload(C_1, p, B)$.

- ▶ Căutarea înapoi are un factor de ramificare mai mic decât căutarea înainte datorită restricției pentru acțiuni relevante.

Căutare înapoi (*regression planning*)

- ▶ Regresia obiectivului prin acțiuni
Acțiunea relevantă $Unload(C_1, p, B)$ poate fi aplicată dacă condițiile sunt satisfăcute \rightarrow orice predecesor trebuie să includă condițiile $In(C_1, p) \wedge At(p, B)$.

Starea predecesor:

$$In(C_1, p) \wedge At(p, B) \wedge At(C_2, B) \wedge \dots \wedge At(C_{20}, B).$$

- ▶ Acțiune **consistentă**: nu modifică un literal.

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

- ▶ Construirea **predecesorilor**:
 - ▶ se elimină efectele pozitive ale acțiunii a care apar în g
 - ▶ se adăugă literalii precondiție ai acțiunii apentru scopul g , a o acțiune relevantă și consistentă.
- ▶ Terminarea algoritmului: când este generată o descriere a predecesorului care este satisfăcută de starea inițială.

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

- ▶ 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.

- ▶ *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

Planificare

Reprezentarea problemelor de planificare

Algoritmi de căutare

Planificare cu ordine parțială

Planificare cu ordine parțială

- ▶ Planificare cu ordine totală (progresie și regresie)
 - ▶ nu utilizează descompunerea problemei
- ▶ Este mai eficientă **rezolvarea independentă a subobiectivelor** și apoi combinarea subplanurilor. Putem lucra întâi cu deciziile importante/evidente.

Planificare cu ordine parțială

Planuri: secvențe parțial ordonate

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)

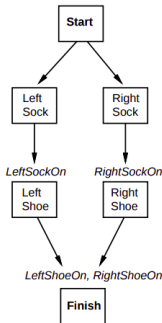
Action(LeftSock, EFFECT : LeftSockOn)

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

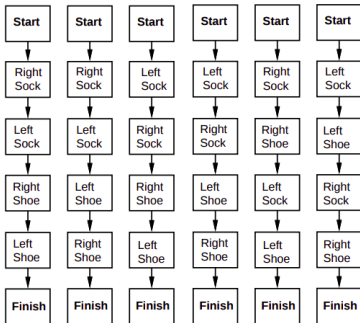
Planificare cu ordine parțială

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

Partial-Order Plan:



Total-Order Plans:



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 condiție a $RightShoe$;
 $RightSockOn$ trebuie să rămână adevărat 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:

$\{RightSock \xrightarrow{RightSockOn} RightShoe, LeftSock \xrightarrow{LeftSockOn} LeftShoe, RightShoe \xrightarrow{RightShoeOn} Finish, LeftShoe \xrightarrow{LeftShoeOn} Finish\}$

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 condițiile din *Finish* sunt deschise.
- ▶ Funcția succesor alege o condiție 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 $C \prec A$;
- ▶ Testarea obiectivului: verifică dacă un plan este soluție (nu există condiții deschise)

Exemplu

```
Init( $At(Flat, Axle) \wedge At(Spare, Trunk)$ )
Goal( $At(Spare, Axle)$ )
Action(Remove( $Spare, Trunk$ ),
  PRECOND:  $At(Spare, Trunk)$ 
  EFFECT:  $\neg At(Spare, Trunk) \wedge At(Spare, Ground)$ )
Action(Remove( $Flat, Axle$ ),
  PRECOND:  $At(Flat, Axle)$ 
  EFFECT:  $\neg At(Flat, Axle) \wedge At(Flat, Ground)$ )
Action(PutOn( $Spare, Axle$ ),
  PRECOND:  $At(Spare, Ground) \wedge \neg At(Flat, Axle)$ 
  EFFECT:  $\neg At(Spare, Ground) \wedge At(Spare, Axle)$ )
Action(LeaveOvernight,
  PRECOND:
  EFFECT:  $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$ 
          $\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle)$ )
```

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

Exemplu (I)

Secvența de evenimente:

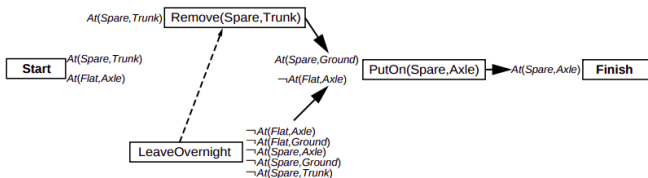
- ▶ Alege condiția deschisă $At(Spare, Axle)$. Alege acțiunea $PutOn(Spare, Axle)$
- ▶ Alege condiția $At(Spare, Ground)$ și acțiunea $Remove(Spare, Trunk)$



Exemplu (II)

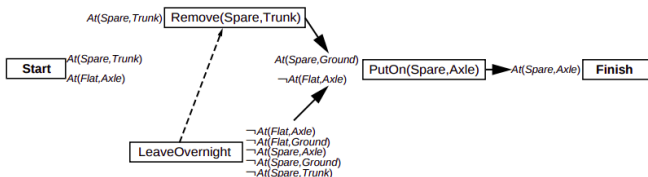
- Alege condiția $\neg \text{At}(\text{Flat}, \text{Axle})$.

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



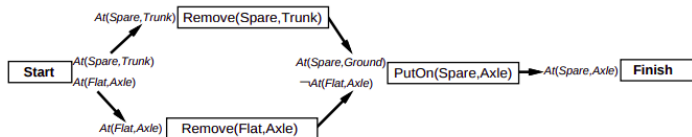
Exemplu (III)

- ▶ Singura condiție 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 preconditionia $\neg At(Flat, Axle)$. Alege $Remove(Flat, Axle)$.
- ▶ Alege preconditionia $At(Spare, Trunk)$ și alege $Start$ pentru a o îndeplini.
- ▶ Alege preconditionia $At(Flat, Axle)$ 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 preconditionii deschise
- ▶ selectează preconditionia deschisă care poate fi satisfăcută în cele mai puține moduri

- ▶ S. Russel, P. Norvig. *Artificial Intelligence: a modern approach*. Chapter 11 Automated Planning
<https://aima.cs.berkeley.edu/newchap11.pdf>
- ▶ D. Poole, A. Mackworth. *Artificial Intelligence. Foundations of Computational Agents*. Chapter 6. Deterministic Planning
<https://artint.info/3e/html/ArtInt3e.Ch6.html>