

計画立案

Chapter 11

計画立案

- 計画立案の問題
- 状態空間探索での計画立案
- 半順序での計画立案
- 計画立案グラフ
- 命題論理での計画立案
- 計画立案方法の分析

計画立案とはなにか

- 作業の遂行と目的の達成のために一連の動作を生成する
 - 状態、動作、ゴール
- 計画の抽象的な空間で解を探索する
- 実用的な応用のなかで人間を助ける
 - 設計と製造
 - 軍事演習
 - ゲーム
 - 宇宙探索

現実世界の問題の難しさ

- ある探索方法を用いて問題を解いてくれるエージェントでは
 - どの動作が関係あるか
 - 全数探索か後方探索か
 - よい発見的方法はどれか
 - 状態の価値を評価できるか
 - 問題に従属か独立か
 - どのように問題を分割できるか
 - 現実世界の問題のほとんどは分割可能に近い

計画立案の言語

- いい言語とは何か
 - 幅広い多様な問題を記述するのに十分な表現能力を有する
 - それを操作するための効率のよいアルゴリズムを可能にするために十分に制限的である
 - 計画立案アルゴリズムは問題の論理的な構造を利用できないといけない
- STRIPSとADL

汎用言語の特徴

- 状態の表現
 - 論理的な状況で世界を分割し、肯定のリテラルの連言として状態を表現する
 - 命題論理でのリテラル: $Door \wedge Unknown$
 - 一階述語論理でのリテラル(grounded and function-free):
 $At(Plane1, Melbourne) \wedge At(Plane2, Sydney)$
 - 閉じた世界を仮定
- ゴールの記述
 - 部分的に示された状態で、肯定の基底リテラルで表されたもの
 - 状態がゴールのすべてのリテラルを含んでいればゴールは満足されている

汎用言語の特徴

- 動作の記述

- 動作 = 前提 + 効果

- 動作(Ation): $(Fly(p, from, to),$

- 前提条件(PRECOND): $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

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

- = 動作のスキーム (p, from, toはインスタンス化のため必要)

- 動作の名前とパラメータのリスト
 - 前提条件 (関数を用いないリテラルの連言)
 - 動作 (関数を用いないリテラルの連言とPは真とnot Pは偽)

- 効果は追加のリストと消去のリスト

言語の意味

- 動作は状態にどのように影響するか
 - 動作は、前提条件を満たしているいかなる状態にも適用可能
 - 一階述語論理の動作スキームでは、条件で変数に対する置換 θ を適用する

$At(P1, JFK) \wedge At(P2, SFO) \wedge Plane(P1) \wedge Plane(P2) \wedge$
 $Airport(JFK) \wedge Airport(SFO)$

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

With $\theta = \{p/P1, from/JFK, to/SFO\}$

Thus the action is applicable.

言語の意味

- 状態sで動作aを実行した結果は状態s'
 - s'は次を除いてsと同じ
 - aの効果での全ての肯定のリテラルPはs'に含まれる
 - 全ての否定のリテラル¬Pはs'から除かれる

$At(P1, SFO) \wedge At(P2, SFO) \wedge Plane(P1) \wedge Plane(P2) \wedge$
 $Airport(JFK) \wedge Airport(SFO)$

- STRIPSの仮定: (表現上のフレーム問題を避けるために)
効果の中ですべてのリテラルNOTは変化しないで残る

表現力と拡張

- STRIPSは簡易化されている
 - 重要な制限: 関数を用いないリテラル
 - 命題論理での表現となる
- 関数シンボルは無限に多数の状態と動作へ導く
- 最近の拡張: Action Description language (ADL)

Action(Fly(p:Plane, from: Airport, to: Airport),

PRECOND: At(p,from) \wedge (from \neq to)

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

標準化 : *planning domain definition language (PDDL)*

Example: air cargo transport

Init(*At*(C1, SFO) \wedge *At*(C2, JFK) \wedge *At*(P1, SFO) \wedge *At*(P2, JFK) \wedge *Cargo*(C1) \wedge
Cargo(C2) \wedge *Plane*(P1) \wedge *Plane*(P2) \wedge *Airport*(JFK) \wedge *Airport*(SFO))

Goal(*At*(C1, JFK) \wedge *At*(C2, 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))

[*Load*(C1, P1, SFO), *Fly*(P1, SFO, JFK), *Load*(C2, P2, JFK), *Fly*(P2, JFK, SFO)]

Example: Spare tire problem

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: *At*(*Spare*, *Axle*) \wedge \neg *Ar*(*Spare*, *Ground*)

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*))

この例題はSTRIPSの枠外: 条件に否定のリテラル(ADL記述)

Example: Blocks world

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: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x)$

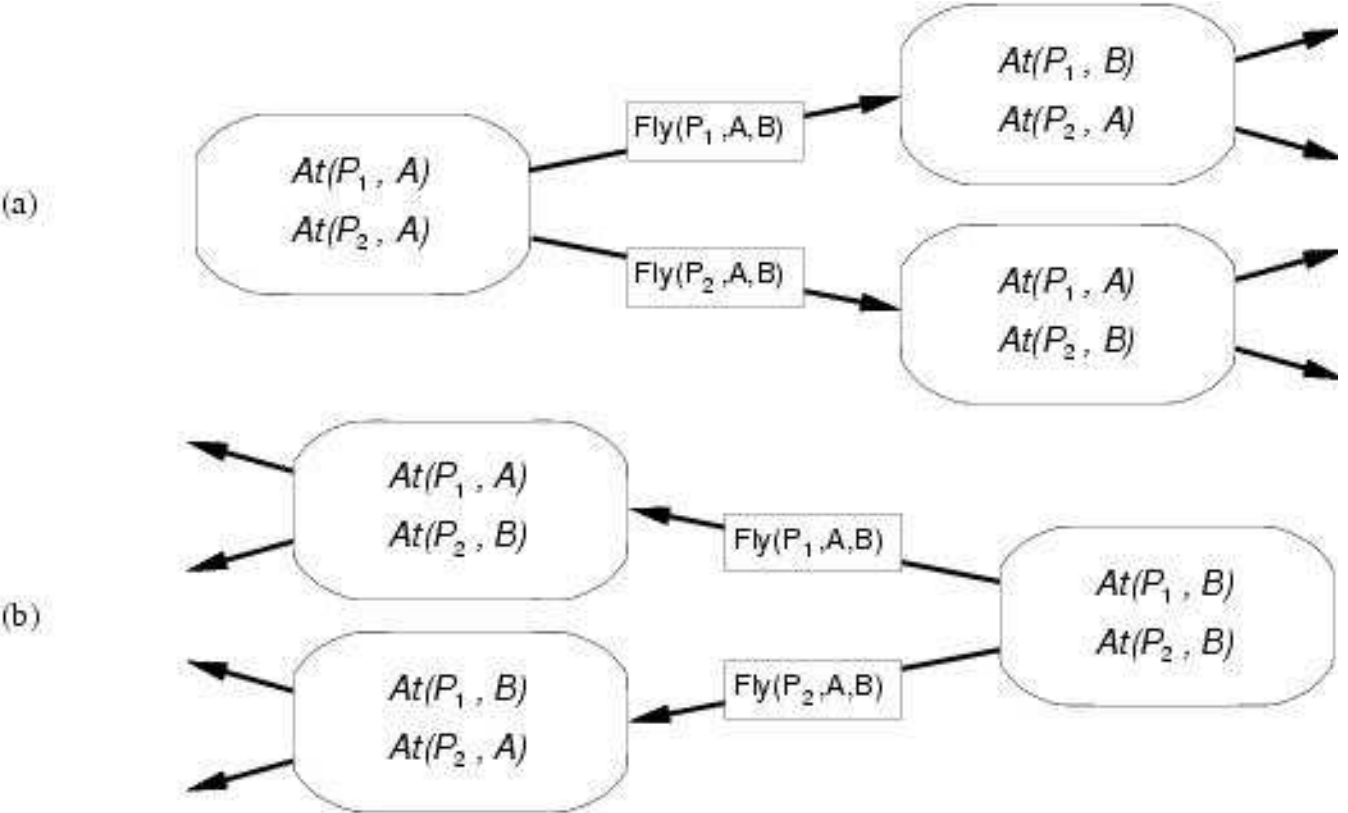
EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x)$

誤った動作が可能: *Move*(B, C, C)

状態空間探索を伴った計画立案

- 前方と後方の両探索が可能
- 前進プランナー
 - 前方状態空間探索
 - 与えられた状態で、可能なすべての動作がもたらす効果を考える
- 後退プランナー
 - 後方状態空間探索
 - ゴールを達成するために、前の状態で何が真でなければならないか

前進と後退



前進アルゴリズム

- 状態空間探索問題としての定式化:
 - 初期状態 = 計画立案問題の初期条件
 - 現れていないリテラルは偽
 - 動作 = 前提条件が満たされている動作
 - 肯定の効果を加え、否定の効果を消去
 - ゴールテスト = この状態はゴールを満たすか
 - ステップコスト = すべての動作はコストが1である
- 関数なし ... 完備であるすべてのグラフ探索は完備な計画立案アルゴリズムである
- 非効率: (1) 無関係な動作問題を指摘できない (2) よい発見的手法なしには効率的な探索を行えない

後退アルゴリズム

- いかにかに先行する状態を見つけるか
 - どの状態に、与えられた動作を適用すれば、ゴールに導くか
 - ゴールの状態 = $At(C1, B) \wedge At(C2, B) \wedge \dots \wedge At(C20, B)$
 - 連言の最初のリテラルに関係のある動作: $Unload(C1, p, B)$
 - 前提条件が満たされたときのみ機能する
 - 前の状態 = $In(C1, p) \wedge At(p, B) \wedge At(C2, B) \wedge \dots \wedge At(C20, B)$
 - サブゴール $At(C1, B)$ はこの状態では現れてはいけない
- 動作は望まれるリテラルをやり直してはいけない (一貫性)
- 主な利点: 関係ある動作のみ考えればよい
 - 前方探索よりは多くの場合少ない分岐

後退アルゴリズム

- 先行を構成するための一般的なプロセス
 - ゴール記述Gを与える
 - Aを関係があり一貫性のある動作とする
 - 先行は次のものである:
 - Gに現れるAの肯定の効果を消去する
 - Aの前提条件はすでに現れていない限り付け加える
- いかなる標準的な探索アルゴリズムも探索するために加えらる
- 先行が初期条件を満たしたとき終了する
 - 一階述語論理のときは、条件の満足は置換を必要とする

状態空間探索のための発見的手法

- 前進も後退アルゴリズムもよい発見的手法なしにはとても効率的ではない
 - ゴールを達成するためにどれだけの動作が必要か
 - 正確な解を求めることはNP困難。よい見積もりを見つけること
- 許容できる発見的手法を見つけるための二つのアプローチ:
 - 弛緩法での最良の解
 - 動作から全ての前提条件を取り除く
 - 独立なサブゴールとしての仮定:
 - サブゴールの連言を解くコストはそれらのサブゴールを独立に解いたときの和である

半順序計画立案

- 前進と後退の計画立案は完全に全順序計画検索方法
 - これらの方法は問題を分割したときの利点が得られない
 - 全てのサブゴールについて動作がどのように順序付けられるかを判断しなければならない
- 最小決定戦略:
 - 探索の間選択を遅らせる

Shoe example

Goal(RightShoeOn \wedge LeftShoeOn)

Init()

Action(RightShoe, PRECOND: RightSockOn
 EFFECT: RightShoeOn)

Action(RightSock, PRECOND:
 EFFECT: RightSockOn)

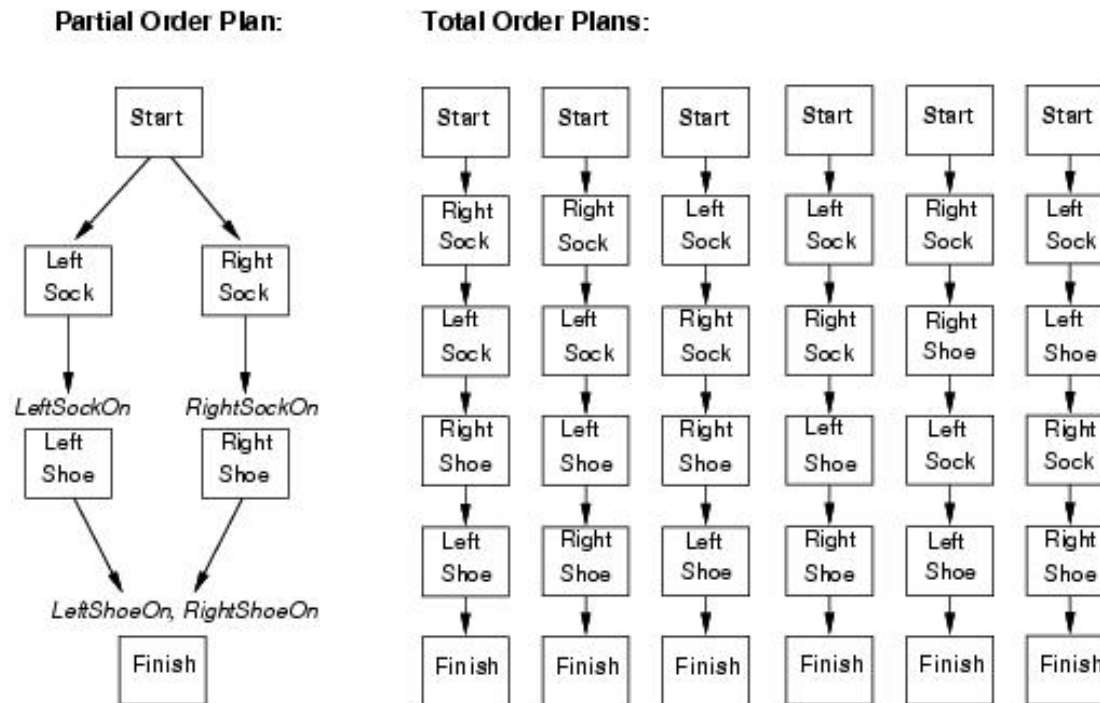
Action(LeftShoe, PRECOND: LeftSockOn
 EFFECT: LeftShoeOn)

Action(LeftSock, PRECOND:
 EFFECT: LeftSockOn)

Planner: combine two action sequences
(1)leftsock, leftshoe (2)rightsock, rightshoe

半順序計画立案

- どちらの動作が先に来るかを決めていない計画立案・アルゴリズムはいかなるものでも半順序計画立案である



探索問題としての半順序計画立案

- 状態は(多くの場合終わっていない)計画である
 - 空の計画は最初と最後の動作のみを含んでいる
- 全ての計画は4つの要素:
 - 動作の集合 (計画のステップ)
 - 順序付けの制約の集合 $A : A < B$
 - サイクルは矛盾となる
 - 因果リンクの集合 $A \xrightarrow{p} B$
 - 因果リンクと矛盾する新しい動作を付加することで、計画を拡張してはならない (Cの効果は $\neg p$ 、CはAの後でそしてBの前がこれにあたる)
 - 未解決な前提条件の集合
 - 前提条件が計画の中での動作によって成し遂げられないもの

探索問題としての半順序計画立案

- 順序付けされた制約にサイクルがなく、因果リンクに矛盾がないとき、計画は一貫性があるという
- 未解決な前提条件がない一貫性のある計画は解である
- 半順序計画立案は可能な次の動作を繰り返し選択することで実行される
 - 排他的にしか使えない環境ではここでの柔軟性は有効である(12章)

半順序計画立案を解く

- 命題論理での 計画立案を仮定:
 - 初期の計画は *Start* と *Finish* を含んでいて、順序づけられた制約は $Start < Finish$ で。因果リンクはなく、*Finish* での前提条件は未解決
 - 継続関数：
 - 動作 *B* から未解決な前提条件 *p* を取り出す
 - 全ての可能な一貫性のある方法で、*p* を成し遂げる動作 *A* を選択し継続の計画を生成する
 - ゴールをテストする

一貫性を強化するために

- 継続計画を生成するとき:
 - 因果リンク $A \rightarrow B$ と順序付けされた制約 $A < B$ を計画に加える
 - A が新しいときは $start < A$ と $A < B$ を加える
 - 新しい因果リンクとこれまでの動作との矛盾を解決する
 - A が新しいければこれとこれまでの因果リンクとの矛盾を解決する

プロセスのまとめ

- 部分計画での操作
 - 存在している計画から未解決な前提条件へリンクを加える
 - 未解決な前提条件を満たすステップを加える
 - 可能な矛盾を避けるために他のステップとの間でこのステップの順序付けをする
- 不完全でぼんやりした計画を完全に明確な計画に徐々に変えていく
- オープンな前提条件を達成できないときあるいは矛盾を解決できないときは後戻りする

Example: Spare tire problem

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*)

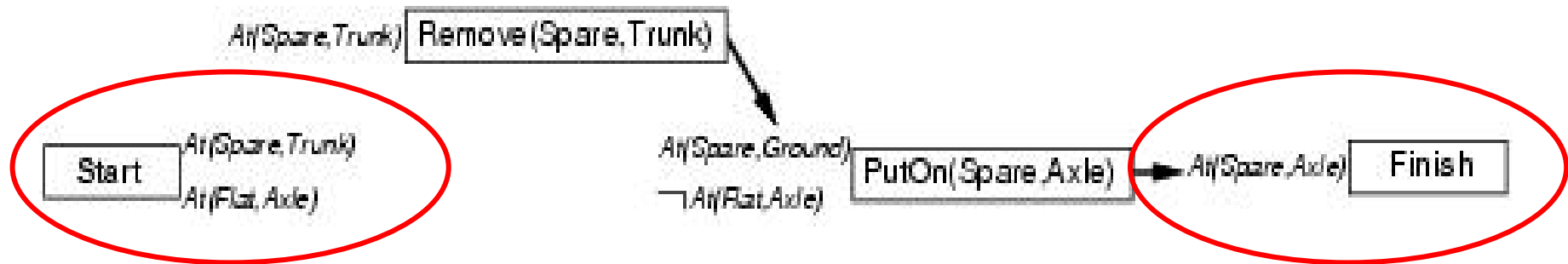
EFFECT: *At*(*Spare*, *Axle*) \wedge \neg *Ar*(*Spare*, *Ground*)

Action(*LeaveOvernight*

PRECOND:

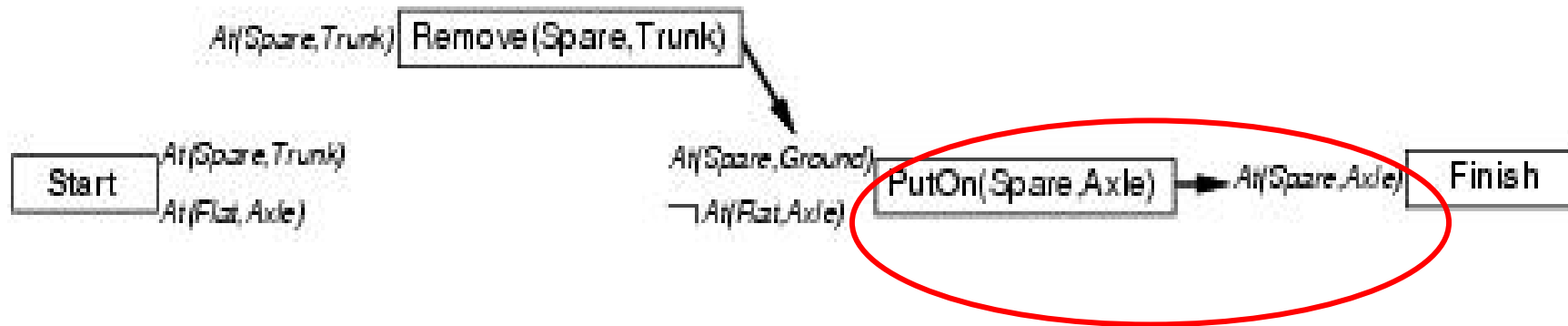
EFFECT: \neg *At*(*Spare*, *Ground*) \wedge \neg *At*(*Spare*, *Axle*) \wedge \neg *At*(*Spare*, *trunk*) \wedge \neg *At*(*Flat*, *Ground*) \wedge \neg *At*(*Flat*, *Axle*))

Solving the problem



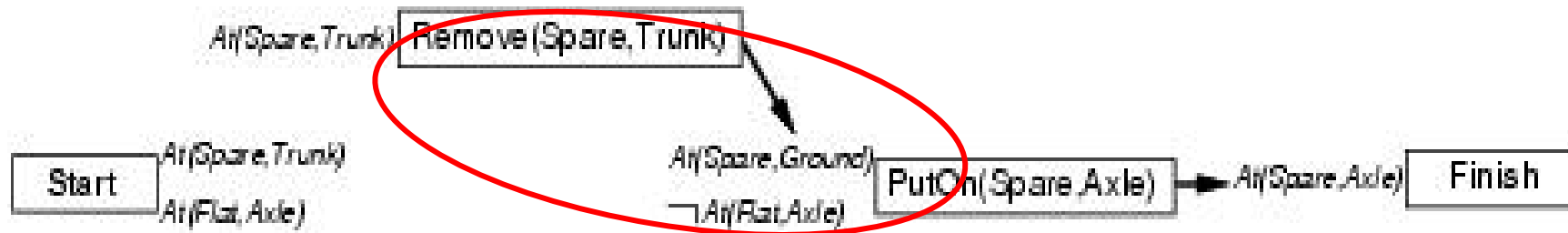
- Initial plan : Start with EFFECTS and Finish with PRECOND.

Solving the problem



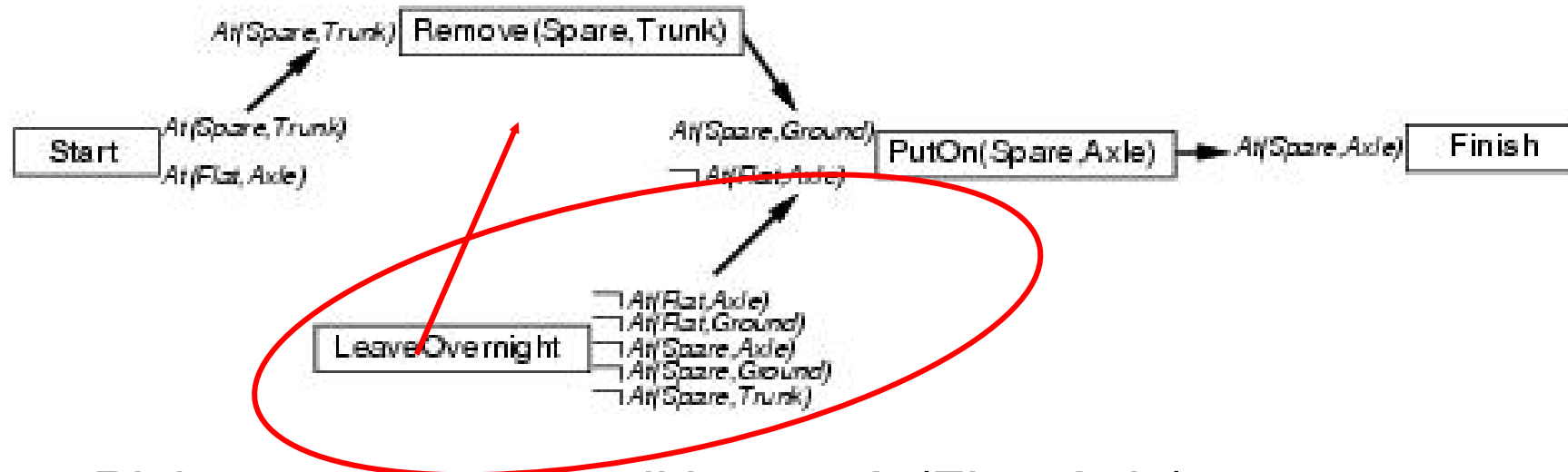
- Initial plan: Start with EFFECTS and Finish with PRECOND.
- Pick an open precondition: $At(Spare, Axle)$
- Only $PutOn(Spare, Axle)$ is applicable
- Add causal link: $PutOn(Spare, Axle) \xrightarrow{At(Spare, Axle)} Finish$
- Add constraint : $PutOn(Spare, Axle) < Finish$

Solving the problem



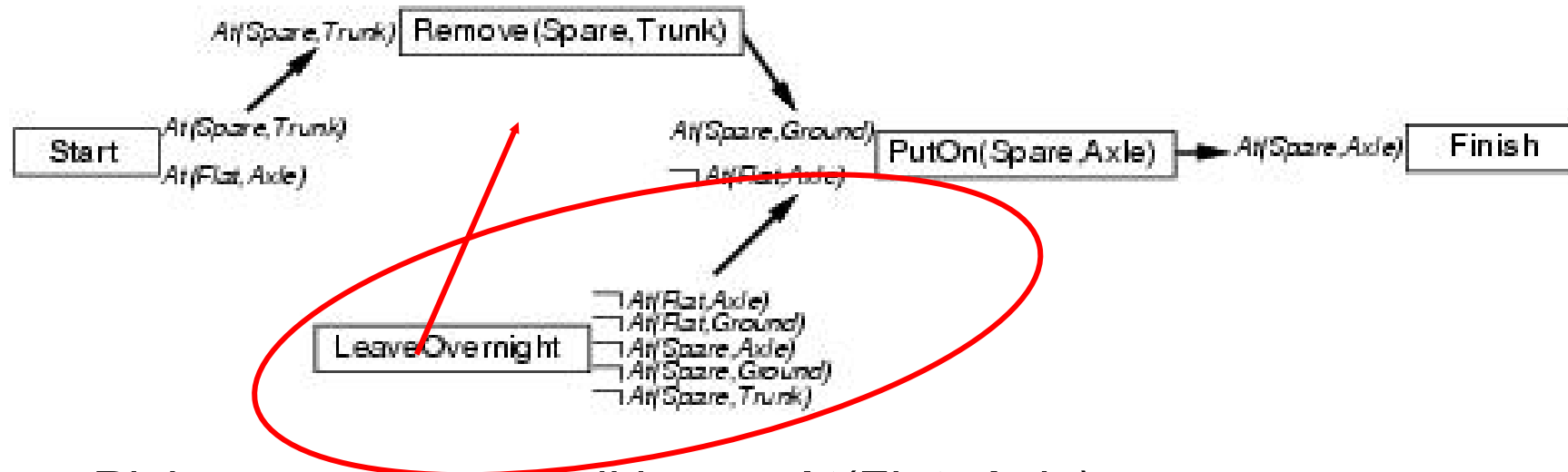
- Pick an open precondition: $At(Spare, Ground)$
- Only $Remove(Spare, Trunk)$ is applicable
- Add causal link: $Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$
- Add constraint : $Remove(Spare, Trunk) < PutOn(Spare, Axle)$

Solving the problem



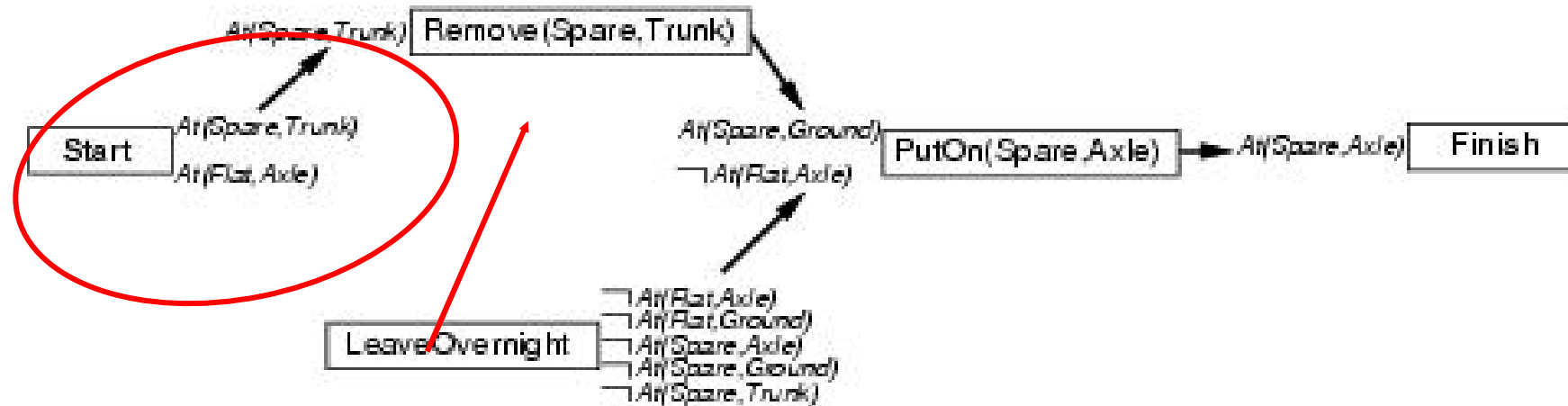
- Pick an open precondition: $\neg At(Flat, Axle)$
- *LeaveOverNight* is applicable
- conflict: $Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$
- To resolve, add constraint : $LeaveOverNight < Remove(Spare, Trunk)$

Solving the problem



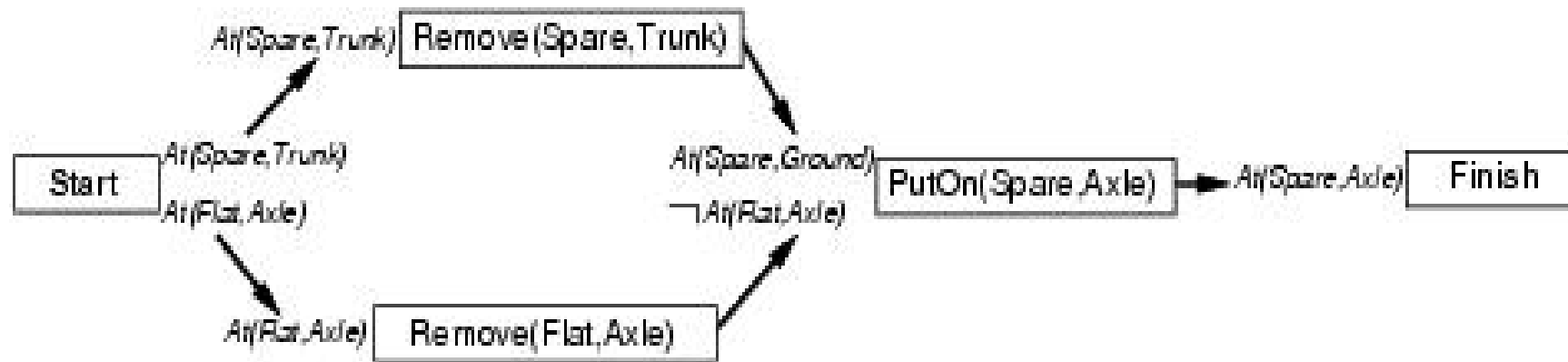
- Pick an open precondition: $\neg At(Flat, Axle)$
- *LeaveOverNight* is applicable
- conflict: $Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$
- To resolve, add constraint : $LeaveOverNight < Remove(Spare, Trunk)$
- Add causal link: $LeaveOverNight \xrightarrow{\neg At(Spare, Ground)} PutOn(Spare, Axle)$

Solving the problem



- Pick an open precondition: $At(Spare, Trunk)$
- Only *Start* is applicable
- Add causal link: $Start \xrightarrow{At(Spare, Trunk)} Remove(Spare, Trunk)$
- Conflict: of causal link with effect $At(Spare, Trunk)$ in *LeaveOverNight*
 - No re-ordering solution possible.
- backtrack

Solving the problem



- Remove *LeaveOverNight*, *Remove(Spare, Trunk)* and causal links
- Repeat step with *Remove(Spare, Trunk)*
- Add also *Remove(Flat, Axle)* and finish

詳細

- 変数が含まれる一階述語論理での表現が使われるとき何が起こるか
 - 矛盾の発見と解決の過程を複雑にする
 - 不一致の制約を導入することで解決される
- 計画立案のアルゴリズムが 前提条件を選択できるようにするため、制約充足問題の最も制約された変数の制約を使うことができる

計画立案グラフ

- よりよい発見的見積もりをするために使われる
 - GRAPHPLANを使って解が直接導かれる
- 計画がタイムステップに関連するレベルの列で構成される
 - レベル0は初期状態
 - 各レベルはリテラルの集合と動作の集合からなっている
 - リテラル = 先行するタイムステップでの動作を実行したときに得られる真となる全てのリテラル
 - 動作 = 先行するタイムステップでの動作を実行したときに得られる真となるリテラルについて実行できる動作

計画立案グラフ

- これは命題論理の問題だけに機能する
- 例:

Init(Have(Cake))

Goal(Have(Cake) \wedge Eaten(Cake))

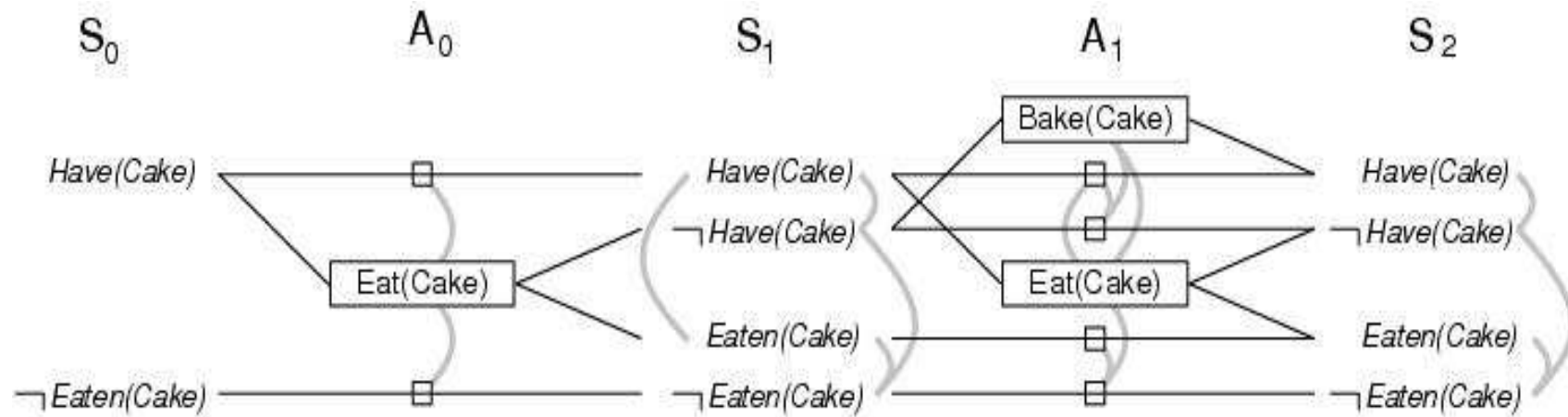
Action(Eat(Cake), PRECOND: Have(Cake)

 EFFECT: \neg Have(Cake) \wedge Eaten(Cake))

Action(Bake(Cake), PRECOND: \neg Have(Cake)

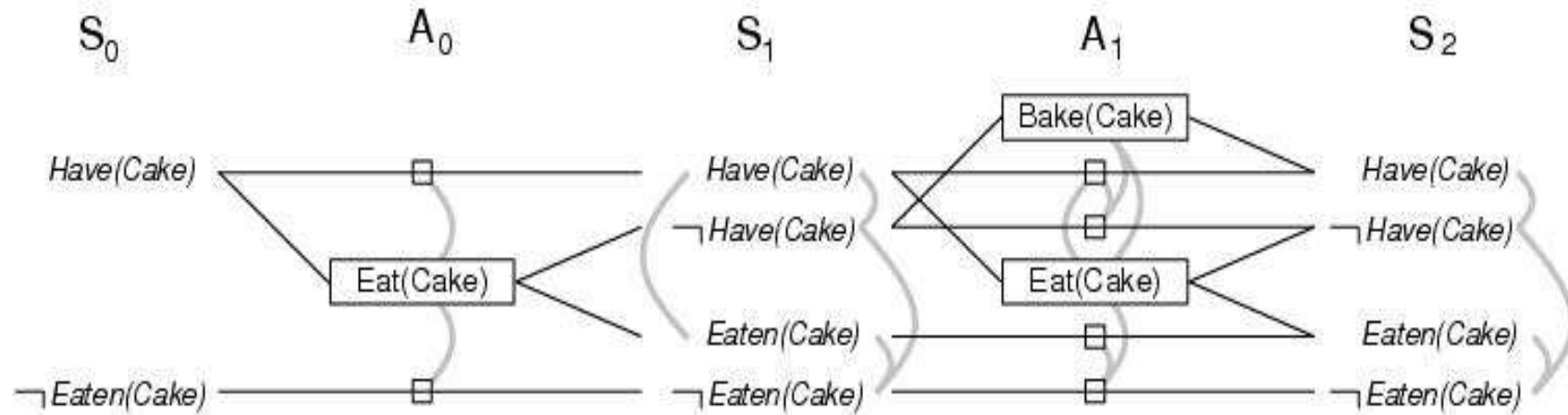
 EFFECT: Have(Cake))

Cake example



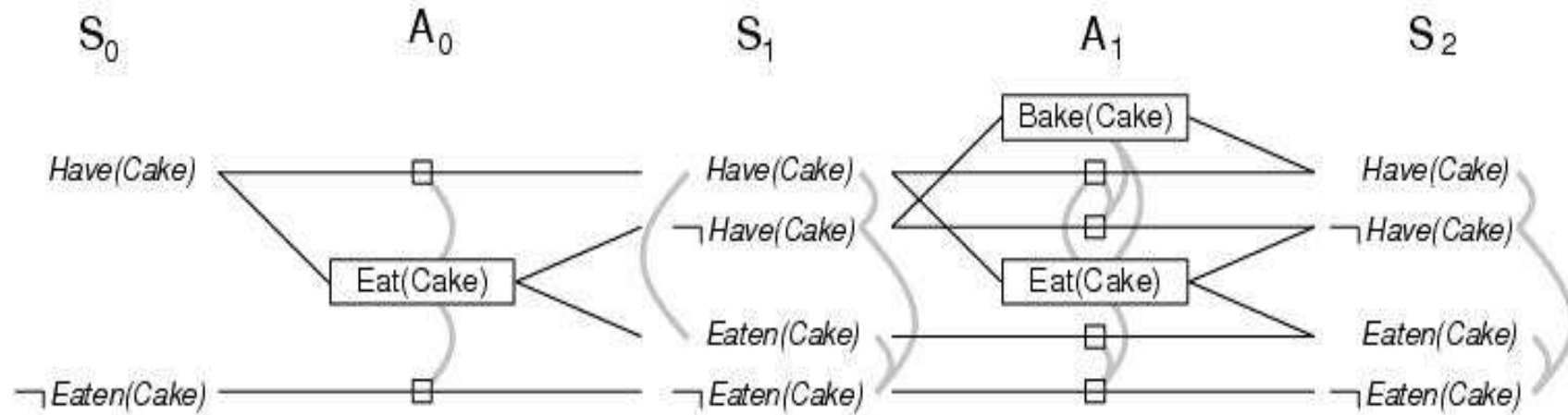
- レベル S_0 から開始。動作としてのレベル A_0 と次のレベル S_1 を決定
 - $A_0 \gg$ 前のレベルで前提条件が満たされている全ての動作
 - 前提条件と動作の効果 $S_0 \rightarrow S_1$ を結びつける
 - 無活動は持続動作として表す
- レベル A_0 は起きえる動作を含む
 - 動作の間での衝突は相互排除リンクで表す

Cake example



- レベルS1はA0での動作のサブセットを取り出して得られるリテラルを全て含んでいる
 - 同時に起こることのないリテラルの間の衝突は相互矛盾リンクで表される
 - S1は多重の状態を表し、相互排除リンクは状態の集合を定義する制約である。
 - 二つの連続するレベルが同一になるまで続ける: *到達*
 - あるいは同じ数のリテラルを含むまで (説明は後で)

Cake example



- 相互矛盾関係は二つの動作のあいだで次のようになっているとき真である：
 - 矛盾する効果: 一方の動作が他方の動作の効果を否定する
 - 干渉: 一方の動作の効果の一つが他方の動作の前提の否定である
 - 競争する必要: 一方の動作の前提条件の一つが他方の動作の前提条件と相互に排他的である
- 相互矛盾関係は二つのリテラルのあいだで次のようになっているとき真である (矛盾する支持)
 - 一方が他方の否定であるか
 - これらのリテラルを達成できる可能な動作の対が相互に矛盾している

計画立案グラフと発見的見積もり

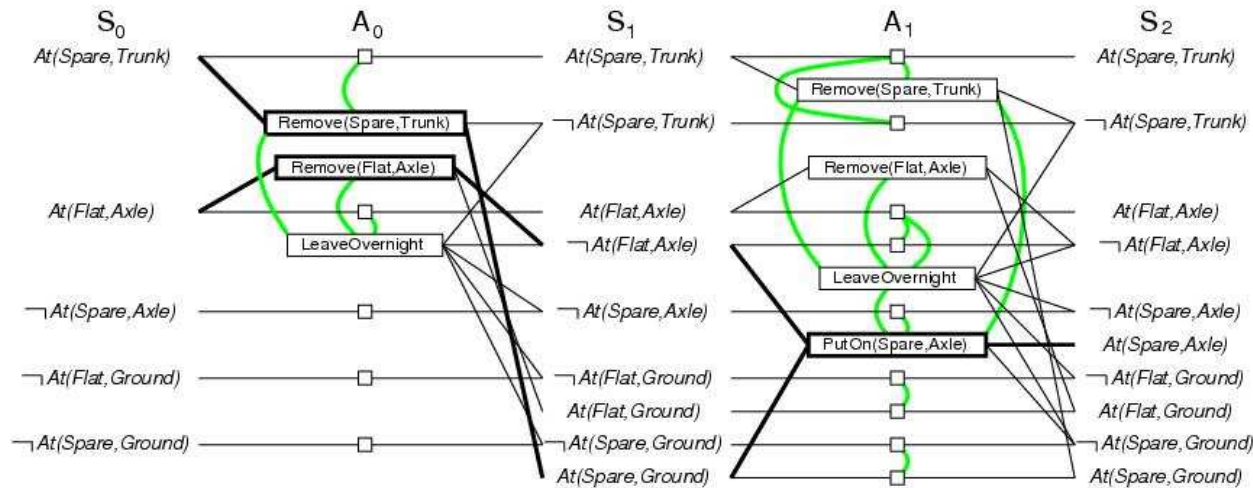
- 計画立案グラフは問題に関する情報を提供する
 - グラフの最終レベルで現れないリテラルはいかなる計画でも達成できない
 - 後方探索に有効である ($\text{cost} = \text{inf}$).
 - レベルがいくつあるかによってゴールのリテラルを成し遂げるのに必要なコストを見積もることができる
 - 小さな問題: 複数の動作が一緒に生じる
 - 直列な計画立案グラフを使うことで一つの動作に限定する (対となっている動作については相互排除リンクを追加する)
 - Max-level, sum-level, set-level heuristicsを使う
- 計画立案グラフは弛緩問題

GRAPHPLANアルゴリズム

- 計画立案グラフから直接解を導き出す方法

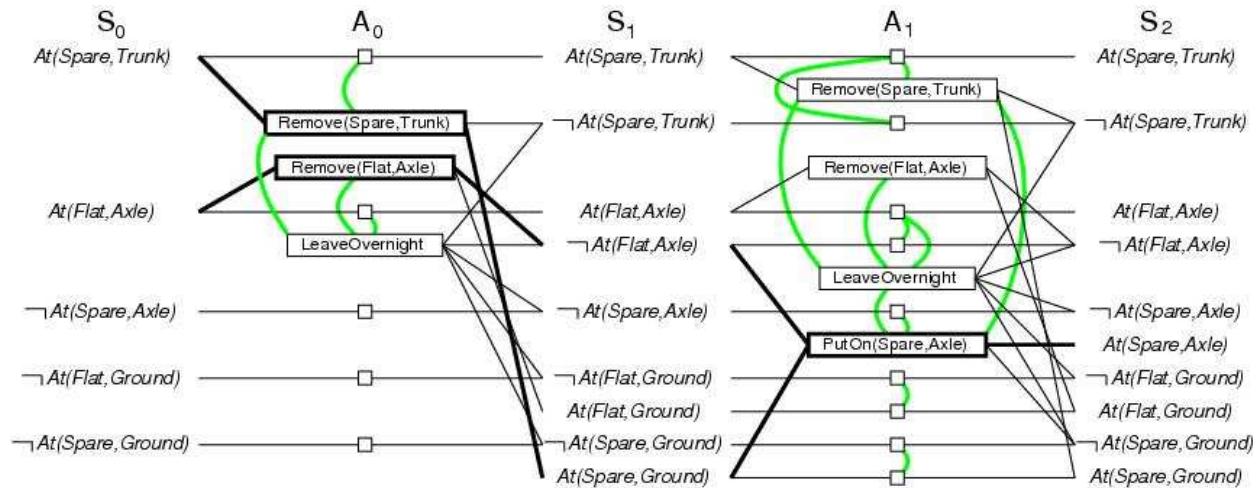
```
function GRAPHPLAN(problem) return solution or failure
  graph ← INITIAL-PLANNING-GRAPH(problem)
  goals ← GOALS[problem]
  loop do
    if goals all non-mutex in last level of graph then do
      solution ← EXTRACT-SOLUTION(graph, goals,
        LENGTH(graph))
      if solution ≠ failure then return solution
      else if NO-SOLUTION-POSSIBLE(graph) then return
failure
  graph ← EXPAND-GRAPH(graph, problem)
```

GRAPHPLAN example



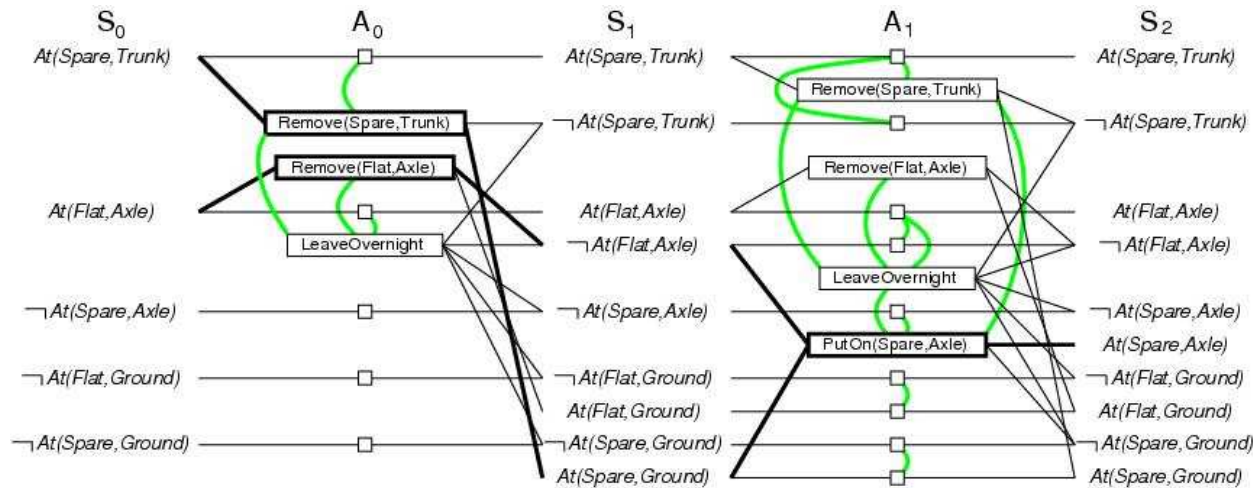
- 最初、計画は初期状態と閉じた世界という仮定から5つのリテラルで構成されている
- 前提条件がEXPAND-GRAPH (A₀)で満足される動作を付け加える
- 持続動作と相互矛盾関係を加える
- レベルS₁での効果を付け加える
- ゴールがレベルS_iになるまで続ける

GRAPHPLAN example



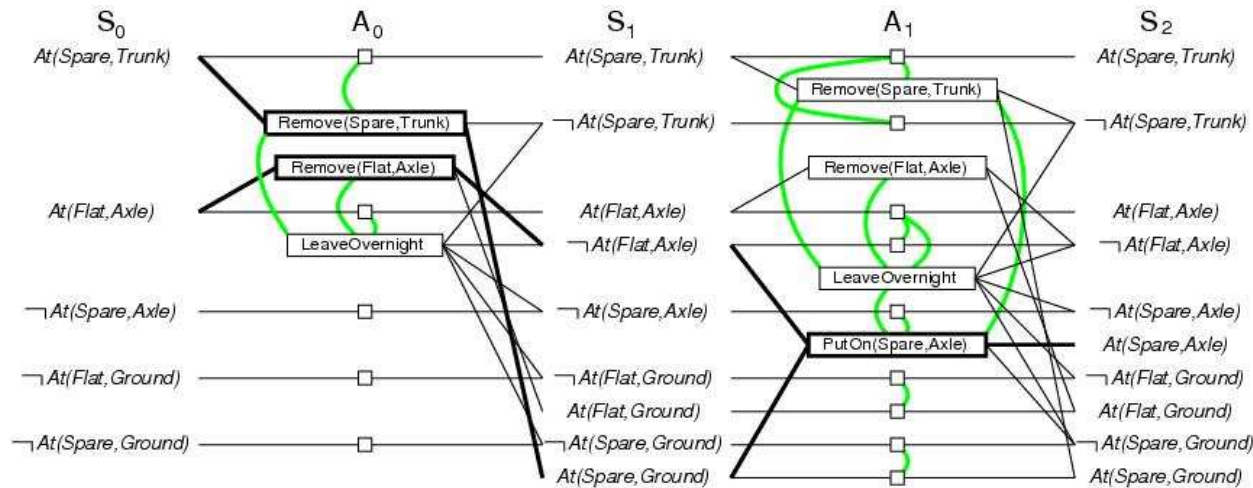
- EXPAND-GRAPHは相互排除の関係を探す
 - 不一致の効果
 - E.g. Remove(Spare, Trunk) and LeaveOverNight
 - 干渉
 - E.g. Remove(Flat, Axle) and LeaveOverNight
 - 競争する必要
 - E.g. PutOn(Spare, Axle) and Remove(Flat, Axle)
 - 矛盾する支持
 - E.g. in S2, At(Spare, Axle) and At(Flat, Axle)

GRAPHPLAN example



- S_2 にはゴールリテラルが存在し、他のリテラルとは相互排除ではない
 - 解が存在する可能性があり、EXTRACT-SOLUTIONがそれを探しを試みをする
- EXTRACT-SOLUTIONは問題あるいは探索プロセスを解くためにブール代数での制約充足問題を使うことができる：
 - 初期状態 = 計画立案グラフの最後のレベルと計画立案問題のゴール
 - 動作 = 状態の中でゴールをカバーする衝突しない動作のある集合を選ぶ
 - ゴール = 全てのゴールが見たされるようにレベル S_0 に到達したとき
 - コスト = 各動作に対して1

GRAPHPLAN example



- 終了したか YES
- 計画立案グラフは単調に増加かあるいは減少:
 - リテラルは単調に増大
 - 動作は単調に増大
 - 相互排除は単調に減少
- この性質と有限の動作とリテラルしか存在しないことにより、全ての計画立案グラフはゴールに到達する

命題論理での計画立案

- 計画立案はsituation calculusでの定理を提供することで実行できる
- 次の論理の文が満足されるかテストする:

initial state \wedge *all possible action descriptions* \wedge *goal*

- 論理の文は動作が起こったときの全ての命題を含んでいる
 - モデルは正しい計画の部分であるときその動作を真にする。そうでないときは偽とする
 - 正しくない計画に対する真偽の割り当てはモデルではない。それはゴールが真でなくてはならないという表明に反する
 - 計画立案が論理の文を解くことができなければ不満足である

SATPLANアルゴリズム

function SATPLAN(*problem*, T_{max}) **return** *solution* or failure

inputs: *problem*, a planning problem

T_{max} , an upper limit to the plan length

for $T = 0$ **to** T_{max} **do**

cnf, *mapping* \leftarrow TRANSLATE-TO_SAT(*problem*, T)

assignment \leftarrow SAT-SOLVER(*cnf*)

if *assignment* is not null **then**

return EXTRACT-SOLUTION(*assignment*, *mapping*)

return failure

$cnf, mapping \leftarrow \text{TRANSLATE-TO_SAT}(problem, T)$

- タイムステップの表明に対する明確な命題
 - 肩につけられた番号はタイムスタンプを表す $At(P1, SFO)^0 \wedge At(P2, JFK)^0$
 - 命題論理は閉じた世界という仮定を持たないので真でない命題も明らかにしなければならない $\neg At(P1, JFK)^0 \wedge \neg At(P2, SFO)^0$
 - 未知の命題は明確にしないままにする
- ゴールは特定のタイムステップである
 - しかしどれかは分からない

$cnf, mapping \leftarrow \text{TRANSLATE-TO_SAT}(problem, T)$

- ゴールに達成したときのタイムステップはどのように決めるか
 - Start at $T=0$
 - Assert $At(P1, SFO)^0 \wedge At(P2, JFK)^0$
 - Failure .. Try $T=1$
 - Assert $At(P1, SFO)^1 \wedge At(P2, JFK)^1$
 - ...
 - 最小の経路が発見されるまでこれを繰り返す
 - 終了は T_{max} で保証されている

$cnf, mapping \leftarrow \text{TRANSLATE-TO_SAT}(problem, T)$

- 動作を命題論理でいかに表すか
 - 後継公理の命題論理版
$$At(P1, JFK)^1 \Leftrightarrow (At(P1, JFK)^0 \wedge \neg(Fly(P1, JFK, SFO)^0 \wedge At(P1, JFK)^0)) \vee (Fly(P1, SFO, JFK)^0 \wedge At(P1, SFO)^0)$$
 - このような公理は飛行機、空港、タイムステップに対して必要
 - 空港が増えれば、右辺はさらに長くなる
- これらの公理がそろったとき、満足なアルゴリズムが計画を探し始める

$assignment \leftarrow SAT-SOLVER(cnf)$

- 複数のモデルが見つけられる
- 以下のものは満足ではない: (for $T=1$)
 $Fly(P1, SFO, JFK)^0 \wedge Fly(P1, JFK, SFO)^0 \wedge Fly(P2, JFK, SFO)^0$
2番目の動作は現実的ではないが、この計画は次の論理の文のモデルである
 $initial\ state \wedge all\ possible\ action\ descriptions \wedge goal^1$
- 不正な動作を避ける: 前提条件の公理
 $Fly(P1, SFO, JFK)^0 \Rightarrow At(P1, JFK)^0$
- ゴールが $T=1$ で達成されるなら一つのモデルだけが全ての公理を満たす

$assignment \leftarrow SAT-SOLVER(cnf)$

- 飛行機は一度に2つの目的地に飛ぶことができる
- これらは満足ではない: (for $T=1$)

$$Fly(P1, SFO, JFK)^0 \wedge Fly(P2, JFK, SFO)^0 \wedge Fly(P2, JFK, LAX)^0$$

2番目の動作は現実的でないが肩にタイムスタンプをつけるのを許している

- 肩での問題を避けるために: 動作排除の公理

$$\neg(Fly(P2, JFK, SFO)^0 \wedge Fly(P2, JFK, LAX)^0)$$

これは同時動作を防ぐ

- 計画が完全に順序付けられていることによる柔軟性の欠如: いかなる動作も一度に起こることは許さない
 - 前提条件への排除を制限する

計画立案アプローチの分析

- 計画立案はAI分野で大きな関心を呼んでいる領域
 - 解の探索
 - 解の存在の構成的証明
- 最大の問題は状態の組み合わせ的爆発
- 効率的な手法は研究中
 - 例: 分断攻略 (divide-and-conquer)