Classic Planning

AI

School of Electronic and Computer Engineering
Peking University

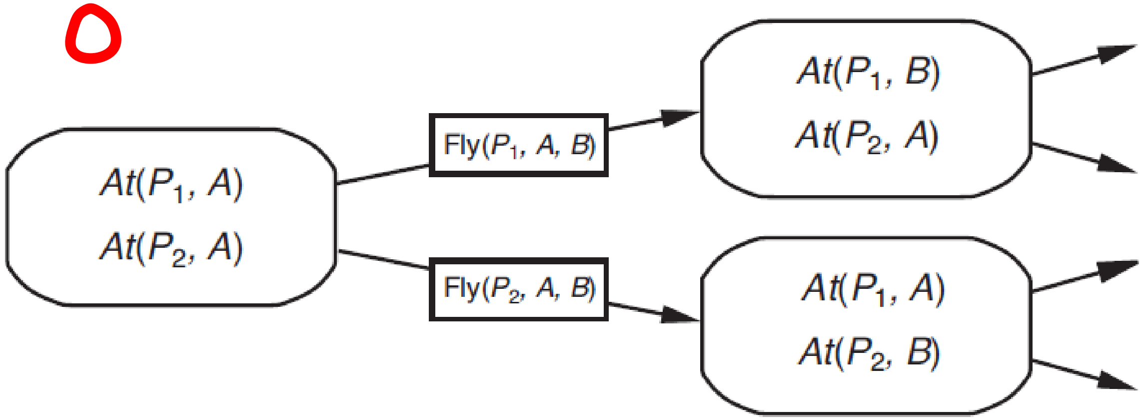
Wang Wenmin

Contents

Two approaches to searching for a plan 搜索计划的两种方式

1) Forward state-space search 前向状态空间搜索

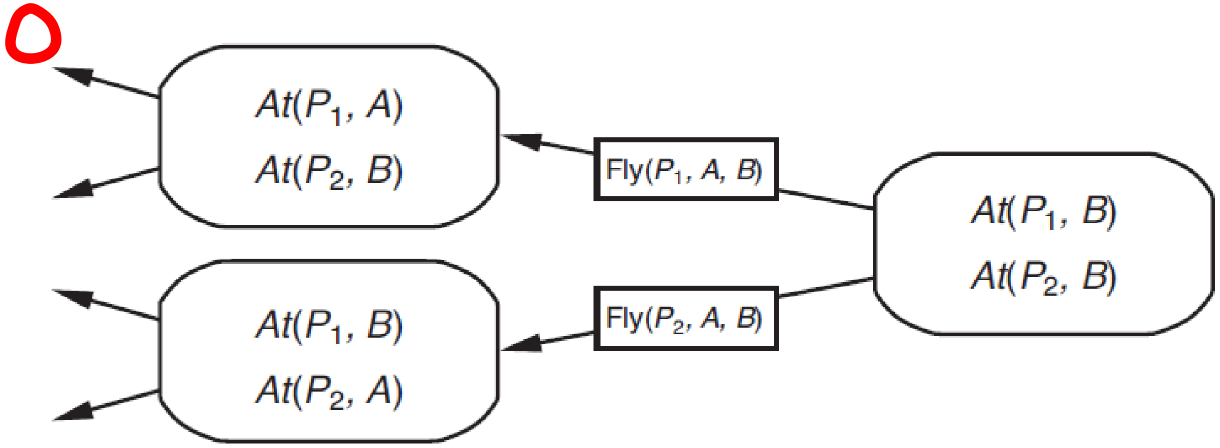
- starting in the initial state,
  从初始状态开始，
- using the problem's actions,
  运用该问题的动作，
- search forward for a member of the goal states.
  朝着一个目标状态向前搜索。



Two approaches to searching for a plan 搜索计划的两种方式

□ 2) Backward relevant-states search 后向状态空间搜索

- starting at the set of states representing the goal,
  从表示该目标的状态集开始，
- using the inverse of the actions,
  运用反向的动作，
- search backward for the initial state.
  朝着初始状态向后搜索。



Heuristics for planning 规划的启发法

- Think of a search problem as a graph 将搜索问题视为一个图
  - where the nodes are states and the edges are actions, to find a path connecting the initial state to a goal state.
    其中节点表示状态、边为动作，寻找一条连接初始状态至某个目标状态的路径。
- Two ways to make this problem easier 该问题简化的两种方式
  - adding edges 增加边
    add more edges to the graph, making it easier to find a path.
    在图上增加更多的边，使之容易找到一条路径。
  - state abstraction 状态抽象
    group multiple nodes together, form an abstraction of the state space that has fewer states, thus is easier to search.
    将多个节点组织在一起，形成具有较少状态的一个状态空间抽象，从而容易搜索。

Two heuristics by adding edges to the graph 图中添加边的两种启发法

☐ 1) Ignore-preconditions heuristic 忽略前提启发法

- Drop all preconditions from actions.
  放弃动作中所有的前提条件。
- Every action becomes applicable in every state, and any single goal fluent can be achieved in one step.
  每个动作变成可作用于每个状态，并且任一目标变数可以用一个步骤实现。

Example: 8-puzzle as a planning problem 8数码难题作为规划问题

$Action(Slide(t, s_1, s_2),$
PRECOND: $On(t, s_1) \land Tile(t) \cancel{\land Blank(s_2) \land Adjacent(s_1, s_2)})$
EFFECT: $On(t, s_2) \land Blank(s_1) \land \neg On(t, s_1) \land \neg Blank(s_2))$

Removing the two preconditions, any tile can move in one action to any space, and get the number-of-misplaced-tiles heuristic.

去掉两个前提条件后，任何棋子可以用一个动作移动到任意空间，从而得到错放棋子个数的启发法。

Two heuristics by adding edges to the graph 图中添加边的两种启发法

- 2) Ignore-delete-lists heuristic 忽略删除表启发法
  - Remove the delete lists from all actions,
    从所有动作中移除删除表，
    i.e., removing all negative literals from effects.
    即，从作用中删除所有的否定文字。
  - That makes it possible to make monotonic progress towards goal:
    这样就使其可以朝向目标单调进展：
    no action will ever undo progress made by another action.
    任何动作都不会取消另一个动作的进展。

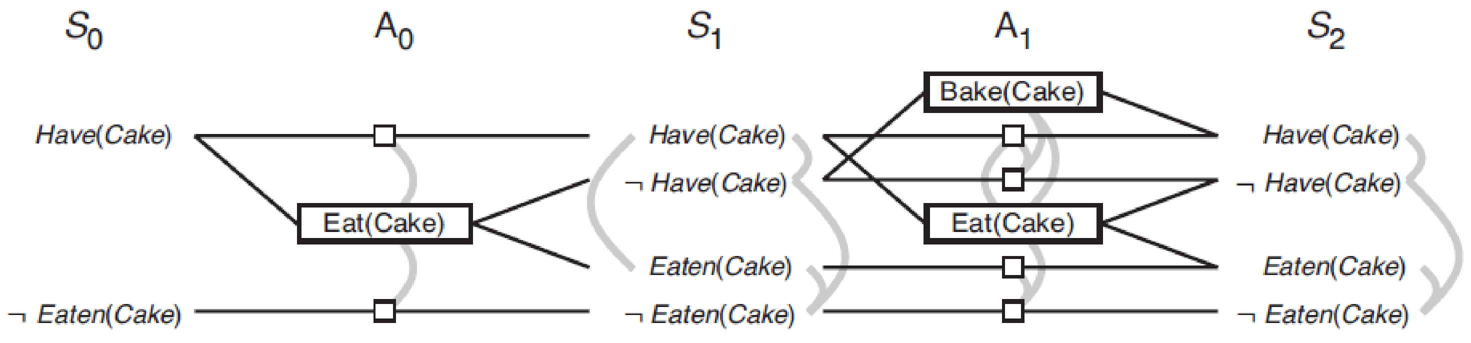
What is a planning graph 什么是规划图

- A directed graph organized into *levels*: 组成层次的有向图：
  - first, a level S_0 for initial state, consisting of nodes representing each fluent;
    首先，初始状态的层次 S_0，包含 表示每个变数的节点；
  - then, a level A_0 consisting of nodes for each action may be applicable in S_0;
    然后，层次 A_0，包含可能适用于 S_0 的每个动作的节点；
  - then, alternating levels S_i followed by A_i;
    然后，交替进入层次 S_i，接着是 A_i；
  - until we reach a termination condition.
    直到到达一个结束条件。
- Work only for propositional planning problems 仅适用于命题规划问题
  - ones with no variables.
    无变量项。

Example 1: Have cake and eat cake too 有蛋糕和吃蛋糕

Init(*Have*(*Cake*))
Goal(*Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Eat*(*Cake*)
  PRECOND: *Have*(*Cake*)
  EFFECT: ¬ *Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Bake*(*Cake*)
  PRECOND: ¬ *Have*(*Cake*)
  EFFECT: *Have*(*Cake*))

The "have cake and eat cake too" problem.
"有蛋糕和吃蛋糕"问题



The "have cake and eat cake too" planning graph.
"有蛋糕和吃蛋糕"规划图

GRAPH-PLAN algorithm GRAPH-PLAN算法

function GRAPH-PLAN(*problem*) **returns** solution or failure
  graph ← INITIAL-PLAN-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*)

It calls EXPAND-GRAPH to add a level, until either a solution is found by EXTRACT-SOLUTION, or no solution is possible.

调用EXPAND-GRAPH来增加一层，直到通过调用EXTRACT-SOLUTION找到一个解，或者没有可能存在的解。

Example 2: Spare tire problem 备用轮胎问题

Init(*Tire*(*Flat*) ∧ *Tire*(*Spare*) ∧ *At*(*Flat*, *Axle*) ∧ *At*(Spare, Trunk))

Goal(*At*(*Spare*, *Axle*))

Action(*Remove*(*obj*, *loc*),
PRECOND: *At*(*obj*, *loc*)
EFFECT: ¬ *At*(*obj*, *loc*) ∧ At(*obj* , *Ground*))

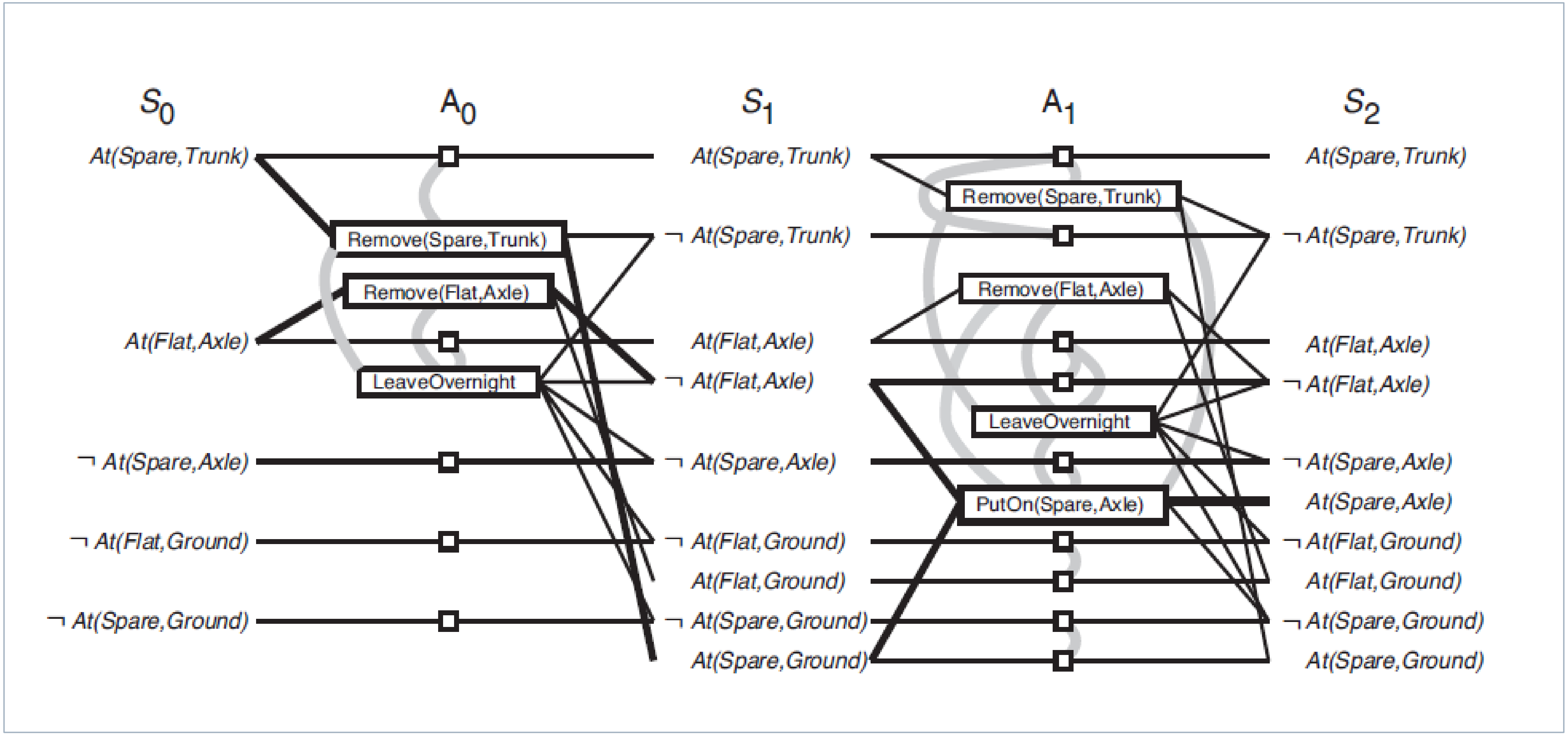
Action(*PutOn*(*t*, *Axle*),
PRECOND: *Tire*(*t*) ∧ *At*(*t*, *Ground*) ∧ ¬ *At*(*Flat*, *Axle*)
EFFECT: ¬ *At*(*t*, *Ground*) ∧ *At*(*t*, *Axle*))

Action(*LeaveOvernight*,
PRECOND:
EFFECT: ¬ *At*(*Spare*, *Ground*) ∧ ¬ *At*(*Spare*, *Axle*) ∧ ¬ *At*(*Spare*, *Trunk*) ∧
¬ *At*(*Flat*, *Ground*) ∧ ¬ *At*(*Flat*, *Axle*) ∧ ¬ *At*(*Flat*, *Trunk*))

The initial state has a flat tire on the axle and a good spare tire in the trunk, and the goal is to have the spare tire properly mounted onto the car’s axle.

初始状态是车轴上有一个瘪的轮胎并且后备箱里有一个好的备胎，而目标是将这个备胎正确地装在车轴上。

Example 2: Planning graph for spare tire problem 备用轮胎问题的规划图



Other Approaches of Classical Planning 其它经典规划方法

- Four other influential approaches:
  其它四种有影响力的方法：
  - 1) planning as Boolean satisfiability,
    化作布尔可满足性的规划
  - 2) planning as first-order logical deduction,
    化作一阶逻辑推理的规划
  - 3) planning as constraint satisfaction,
    化作约束满足的规划
  - 4) planning as plan refinement.
    化作规划精进的规划

1) Planning as Boolean satisfiability 化作布尔可满足性的规划

- Boolean Satisfiability (SAT) 布尔可满足性 (SAT)

  It is the problem of determining if there exists an interpretation that satisfies a given Boolean formula.

  这是确定是否存在满足给定布尔表达式的解释的问题。

  - Satisfiable formula 可满足表达式

    if the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE which make the formula evaluates to TRUE.

    如果给定布尔表达式的变量可一直被TRUE和FALSE值替换，使得表达式的结果为TRUE。

  - Unsatisfiable formula 不可满足表达式

    if no such assignment exists, the function expressed by the formula is identically FALSE for all possible variable assignments.

    如果没有这样的赋值存在，即对所有可能的变量赋值，该布尔表达式的结果始终FALSE。

Example: Planning as Boolean satisfiability 化作布尔可满足性的规划

- Satisfiable formula 可满足表达式

  the formula "*a* AND NOT *b*" is satisfiable, because one can find values

  表达式 "*a* AND NOT *b*" 是可满足的，因为人们可以找到值

  $$a = \text{TRUE}, \quad \text{and} \quad b = \text{FALSE}$$

  which make "*a* AND NOT *b*" to be TRUE.

  使得表达式 "*a* AND NOT *b*"为TRUE。

- Unsatisfiable formula 不可满足表达式

  the formula "*a* AND NOT *a*" is unsatisfiable.

  表达式 "*a* AND NOT *b*" 是不可满足的。

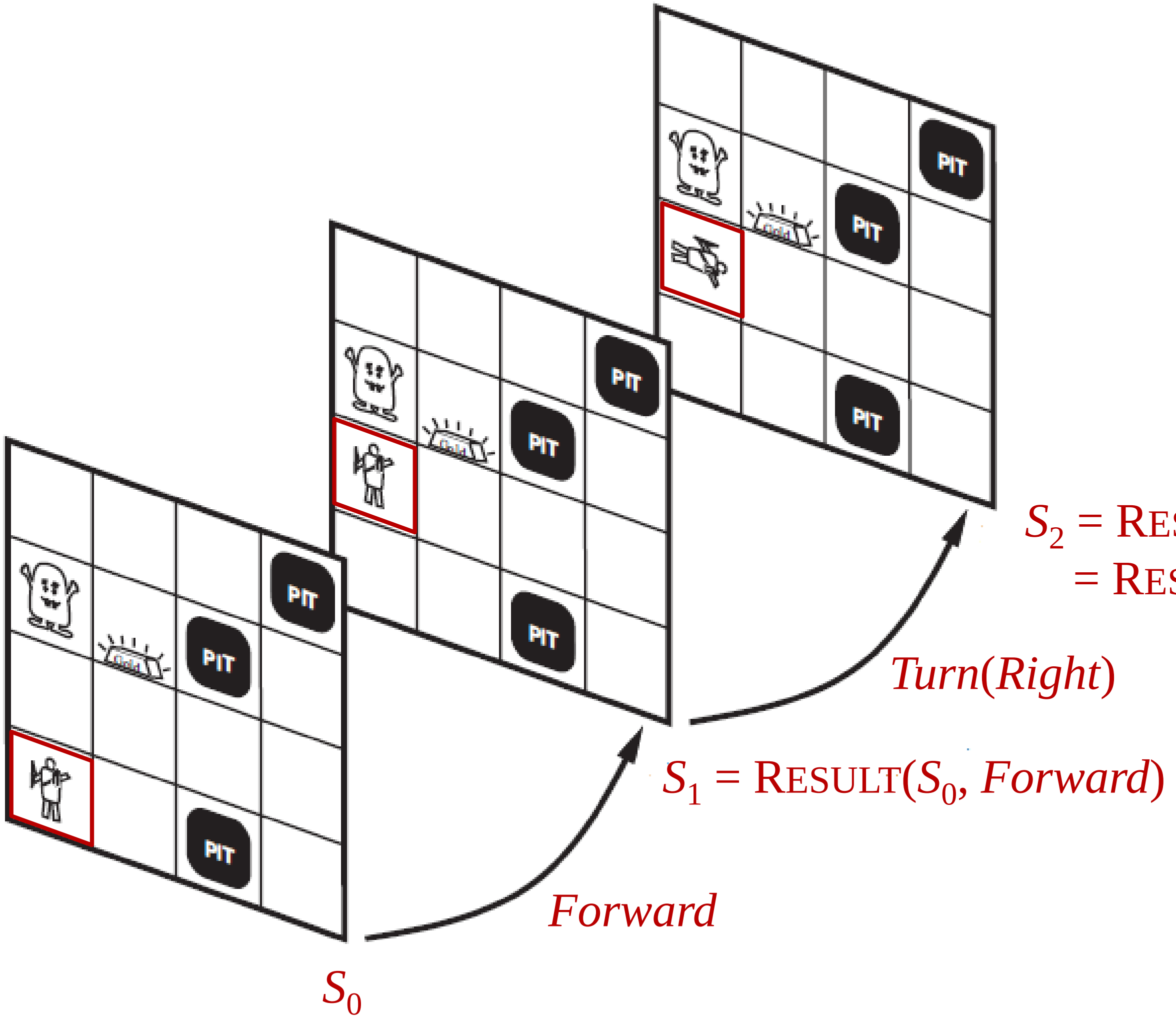
2) Planning as first-order logical deduction 化作一阶逻辑推理的规划

- PDDL is difficult to express some planning problems:
  PDDL难以表达某些规划问题：
  - e.g. can't express the goal: "move all the cargo from A to B regardless of how many pieces of cargo there are".
    例如无法表示如下目标，"把所有的货物从A移到B，不管有多少件货物"。
- Propositional logic also has limitations for some planning problems:
  命题逻辑对某些规划问题也有局限性：
  - e.g. no way to say: "the agent would be facing south at time 2 if it executed a right turn at time 1; otherwise it would be facing east."
    例如无法表达："智能体若在时间1执行了一个右转则将在时间2时朝南；否则将朝东。
- First-order logic lets us get around those limitations.
  一阶逻辑则让我们摆脱这些局限性。

Situation calculus in first-order logic 一阶逻辑中的情景演算

- It is a logic formalism designed for representing and reasoning about dynamical domains. Its main elements are actions, fluents and situations.
  是设计用于动态域的表示和推理的一种逻辑形式论。其主要元素是动作、变数和情景。
- Situation calculus in first-order logic: 一阶逻辑中的情景演算：
  - Initial state is called a *situation*. A solution is a situation that satisfies the goal.
    初始状态称为一个情景。一个解是满足目标的动作序列。
  - A function or relation that can vary from one situation to the next is a *fluent*.
    可将一个情景转变到下一个的函数或关系是变数。
  - Each *action*'s preconditions are described with a *possibility axiom.*
    每个动作的前提用一个可能性公理来描述。
  - Each fluent is described with a *successor-state axiom.*
    每个变数用一个后记状态公理来描述。
  - Need *unique action axioms* so that the agent can deduce that.
    需要唯一动作公理以便智能体能够对其进行推理。

Situations as actions in Wumpus world 魔兽世界中情景为动作



If s is a situation and a is an action, then $\textsc{Result}(s, a)$ is also a situation. Thus, a situation corresponds to a sequence of actions.

如果s是一个情景且a是一个动作，则$\textsc{Result}(s, a)$也是一个情景。因此，一个情景对应于一个动作序列。

3) Planning as constraint satisfaction 化作约束满足的规划

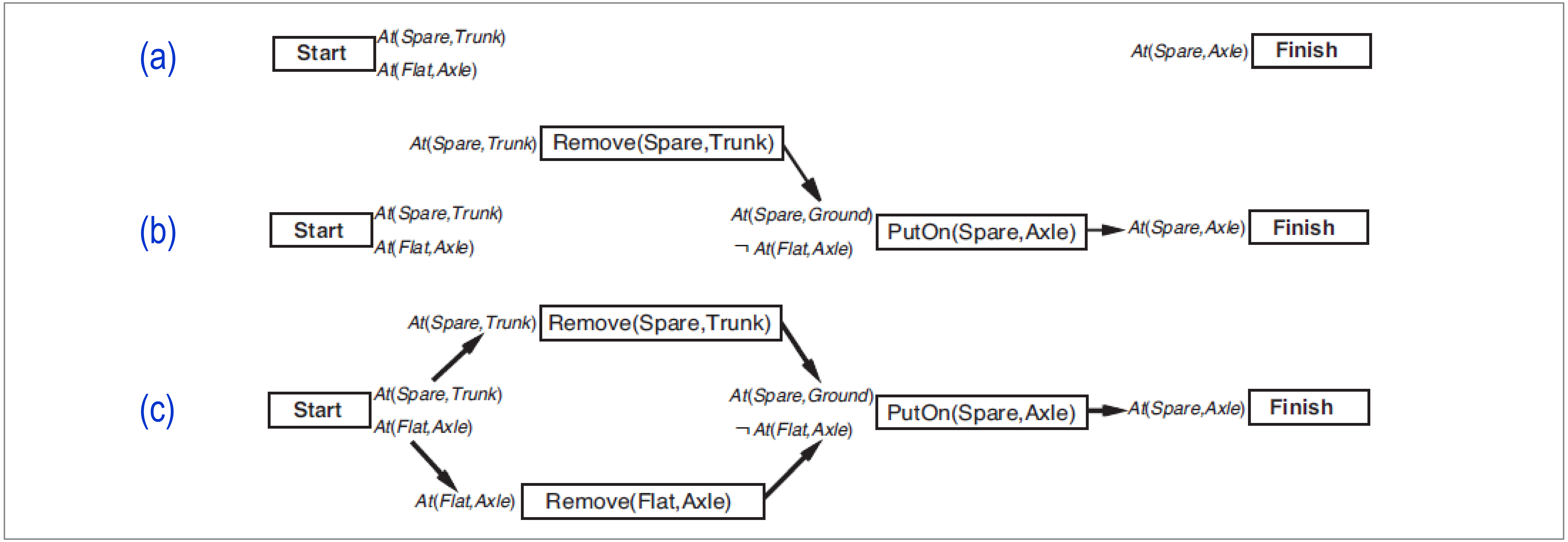
- We have seen 我们已经知道
  - Constraint satisfaction has a lot in common with Boolean satisfiability.
    约束满足与布尔可满足性有许多共性，
  - CSP (constraint satisfaction problem) techniques are effective for scheduling problems.
    CSP（约束满足问题）技术对调度问题很有效。
- So we can 因此我们可以
  - encode a *bounded planning problem* as a CSP, i.e., the problem of finding a plan of length k;
    将有界规划问题进行编码为CSP，例如，寻找一个长度为k的规划的问题；
  - also encode a planning graph into a CSP.
    还可以将规划图编码为CSP。

4) Planning as plan refinement 化作规划精进的规划

- Totally ordered plan 全序规划
  - The totally ordered plan is constructed by all the approaches we have seen so far, consisting of a strictly linear sequence of actions.
    全序规划是由迄今为止我们学到的所有方法所构建的，由严格的线性动作序列组成。
  - This representation ignores the fact that many sub-problems are independent.
    这种表示忽视了许多子问题是独立的这个事实。
- Partially ordered plan 偏序规划
  - An alternative is to represent plans as *partially ordered* structures.
    替代方式是将规划表示为偏序结构。
  - This representation is a set of actions and a set of constraints of the form $Before(a_i, a_j)$, saying that one action occurs before another.
    这种表示是一组动作和一组形式为$Before(a_i, a_j)$的约束，表示一个动作在另一个之前发生。

Example: spare tire problem 备用轮胎问题

Boxes represent actions, arrows indicate orders. 方框表示动作，箭头指出顺序



(a) the tire problem expressed as an empty plan. 将轮胎问题表示为一个空的规划
(b) an incomplete partially ordered plan for the tire problem. 轮胎问题的一个不完全偏序规划
(c) a complete partially-ordered solution. 一个完整的偏序解决方案

Thank you for your attention!

AI