人工智能原理与算法

第11章-经典规划

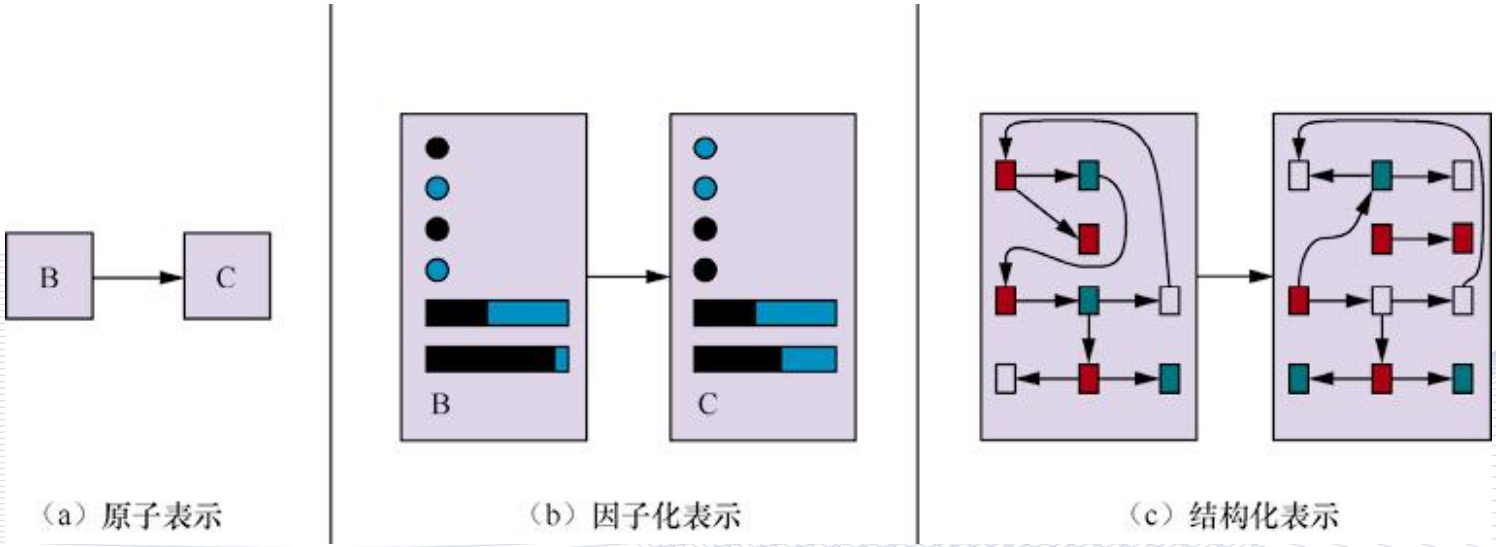
中科院自动化研究所　朱翔昱

前言

- 本章节介绍智能体如何根据问题的结构高效地构建复杂的动作计划：
  - 规划一系列**动作**是智能体的关键需求
  - 正确表示的动作和状态、以及正确的算法可以使规划变得更加容易。
  - 本章节的核心是一种因子化的表示语言

前言

- **原子表示**：世界的每一个状态都是不可分割的，它没有内部结构。两个状态只有“相同”或“不同“
- **因子化表示**：将每个状态拆分为一组固定的变量或属性，每个变量或属性都可以有一个值。
- **结构化表示**：结构化表示状态包括对象，每个对象可能有自己的属性以及与其他对象的关系



- 经典规划的定义
- 状态空间搜索规划算法
- 规划图
- 其他经典规划方法

11.1 经典规划的定义

- 经典规划定义为在一个离散的、确定性的、静态的、完全可观测的环境中，找到完成目标的一系列动作的任务。
  - 离散的：环境的状态、处理时间的方式以及智能体的感知和动作。
  - 静态的：智能体思考时环境不会发生变化。
  - 确定性的：环境的下一个状态完全由当前状态和智能体执行的动作决定。
  - 完全可观测：如果智能体的传感器能让它在每个时间点都能访问环境的完整状态。

11.1 经典规划的定义

- 已介绍两种完成该任务的方法：
  - 问题求解智能体：能够找到导致目标状态的动作序列，其中状态使用原子表示。需要很好的领域相关启发知识以较好地执行
  - 混合命题逻辑智能体：没有领域相关启发知识就能找到规划，使用了基于问题的逻辑结构的启发知识，依赖于基元命题推理
- 限制：
  - 每个**新领域**都需要特定的启发式方法
  - 都需要明确地表示**指数级**的状态空间
    - Wumpus逻辑模型中，向前一步的公理在所有4个朝向、T个时间步、n^2个当前位置重复

11.1 经典规划的定义

- **因子化表示**：用一组变量表示世界的一个状态的方法。可使用一种称为**PDDL** (planning domain definition language，规划领域定义语言)
  - 基本的PDDL可以处理经典规划领域
  - 扩展的PDDL可以处理连续的、部分可观测的、并发的和多智能体的非经典领域
- PDDL处理一个搜索问题时，会关心：**1）初始状态、2）在一个状态可用的动作、3）应用一个动作后的结果**以及目标测试

11.1 经典规划的定义

- **状态**：
  - 每个状态表示为基本原子流（状态变量）的合取，**这些流是基本的（不含变量、只有一个谓词，有参数则必定常量）、无函数的**：

$$Poor \wedge Unknown$$

$$At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$$

  - 在一个状态中，以下的流是不允许的：

    At(x, y)（因为有变量），¬Poor（因为它是否定式）以及
    At(Father(Fred), Sydney)（因为使用了函数符号）

  - 封闭世界假设：任何没有提到的流都是假的。
  - 唯一名称假设：$Truck_1$和$Truck_2$是不同的。

11.1 经典规划的定义

- **动作模式**：
  - 包含：动作名称、动作中的所有变量的列表、**前提（precondition）**和**效果（effect）**组成

$$Action(Fly(p, from, to),$$
$$\text{PRECOND}: At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$
$$\text{EFFECT}: \neg At(p, from) \wedge At(p, to))$$

  - 通过选择常量来实例化变量，产生一个基本动作：

$$Action(Fly(P_1, SFO, JFK),$$
$$\text{PRECOND}: At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$$
$$\text{EFFECT}: \neg At(P_1, SFO) \wedge At(P_1, JFK))$$

11.1 经典规划的定义

- **动作**：
  - 动作定义了问题求解所需要的行为和结果，**即作为动作的结果，什么发生变化，什么保持不变。**
  - 经典规划集中于那些**多数动作使多数事物保持不变的问题**，PDDL根据什么发生了变化来描述一个动作结果；**不提及所有保持不变的东西**

$$Action(Fly(p, from, to),$$
$$\text{PRECOND}: At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$
$$\text{EFFECT}: \neg At(p, from) \wedge At(p, to))$$

11.1 经典规划的定义

- **动作两大要素**：前提和效果
  - 一个动作的前提和效果都是文字（正的或负的原子语句）的合取
  - 前提定义了动作能被执行的状态
  - 效果定义了执行这个动作的结果

11.1 经典规划的定义

- **动作的前提**：如果状态s蕴涵了基本动作a的前提，即前提中的每一个正文字都在s中，每一个负文字都不在s中，则动作a适用于状态s

$$Action(Fly(P_1, SFO, JFK),$$
$$\text{PRECOND: } At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$$
$$\text{EFFECT: } \neg At(P_1, SFO) \wedge At(P_1, JFK))$$

状态S: $\text{At}(P_1, SFO)\ \Lambda\ Plane(P_1)\ \Lambda\ Airport(SFO) \Lambda\ Airport(JFK) \Lambda\ New(P_1)$

前提中的所有正文字都在状态中，前提中没有负文字，
所以可以使用该动作

11.1 经典规划的定义

- **动作的效果**：
  - 在状态s执行动作a的结果定义为s’，它由一组流表示，**这组流由s开始，去掉在动作效果中以负文字出现的流（DEL(a)），并增加在动作效果中以正文字出现的流（ADD(a)）**：

$$\text{RESULTS}(s,a) = (s) - \text{DEL}(a)) \cup \text{ADD}(a)$$

  - 在PDDL中，动作模式里的时间和状态是隐式的：前提总是指时间t，而效果总是指时间t+1

$$Action(Fly(P_1, SFO, JFK),$$
$$\text{PRECOND}: At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$$
$$\text{EFFECT}: \neg At(P_1, SFO) \wedge At(P_1, JFK))$$

经过这个动作后，我们将在状态中删除流$\text{At}(P_1, SFO)$并添加流$\text{At}(P_1, JFK)$

11.1 经典规划的定义

- **规划领域是由一组动作模式定义的**，该域中的一个**特定问题**用一个**初始状态**加一个**目标**来定义：
  - 初始状态是基于原子的合取。
  - **目标是**可以**含有变量**的文字（正文字或负文字）的合取。任何**变量默认是存在量词量化**的，当我们能找到一个动作序列，达成了蕴涵了目标的状态s，问题就得到了解决。

$$At(C_1, SFO) \wedge \neg At(C_2, SFO) \wedge At(p, SFO)$$

C_1在SFO, C_2不在SFO, **有一架**飞机在SFO

  - **规划就被定义成了一个搜索问题：初始状态、ACTIONS函数、Result函数、目标测试。**

11.1 例1：航空货物运输

- 涉及装载和卸载货物，从一个地点飞到另一个地点，能够用三个动作定义：
  - **Load、Unload以及Fly**
- 这些动作影响两个谓词：
  - In(c,p)表示货物c在飞机p里
  - At(x,a)表示对象x在机场a

11.1 例1：航空货物运输

- 问题描述：

初始状态：货物C1在SFO，货物C2在JFK，飞机P1在SFO，飞机P2在JFK

目标：货物C1在JFK, 货物C2在SFO。

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$

动作模式1：Load

$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$

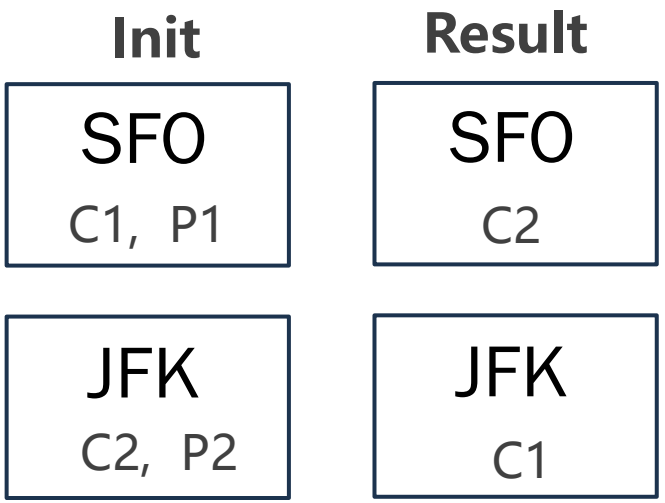
动作模式2：Unload

$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$

动作模式3：Fly

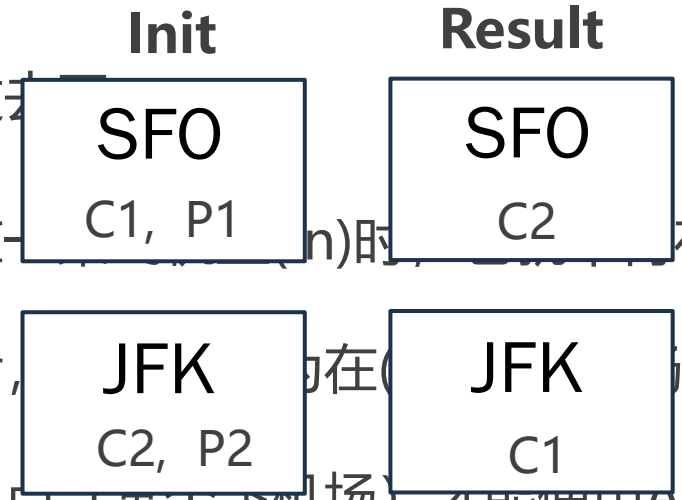
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

图 11-1　航空货物运输规划问题的 PDDL 描述



11.1 例1：航空货物运输

- **注意：**
  - 当一架飞机从一个机场飞到另一个机场时，飞机里的所有货物也跟着飞过去
  - 当一件货物在一架飞机里(In)时，它就不在(At)任何地方；当它被卸载后，它就在(At)机场。因此At意味着“在给定的地点（某个飞机场）才能使用At。”



  - 下面的规划是该问题的一个解

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

11.1 例2：备用轮胎问题

- 考虑更换漏气轮胎的问题
  - **目标**：是把一个好的备用轮胎合适地装配到汽车轮轴上
  - **初始状态**：是有一个漏气的轮胎在轮轴上和一个好的备用轮胎在后备箱内
  - **动作（四种）**：
    - 从后备箱(Trunk)取出备用轮胎(Spare)
    - 从轮轴(Axle)卸下漏气的轮胎(Flat)
    - 将备用轮胎装在轮轴上(PutOn)
    - 彻夜将汽车留下(LeaveOvernight)无人照看
  - 假设汽车停放的环境特别差，这样将它彻夜留下的效果是轮胎消失。

11.1 例3：备用轮胎问题

- 问题描述：

初始状态：漏气轮胎Flat在轮轴上，备用轮胎在后备箱里

目标：备用轮胎在轮轴上

$Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))$

$Goal(At(Spare, Axle))$

$Action(Remove(obj, loc),$

PRECOND: $At(obj, loc)$

EFFECT: $\neg At(obj, loc) \land At(obj, Ground))$

动作模式1：Remove

$Action(PutOn(t, Axle),$

PRECOND: $Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle) \land \neg At(Spare, Axle)$

EFFECT: $\neg At(t, Ground) \land At(t, Axle))$

动作模式2：Puton

$Action(LeaveOvernight,$

PRECOND:

EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk) \land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))$

动作模式3：LeaveOvernight

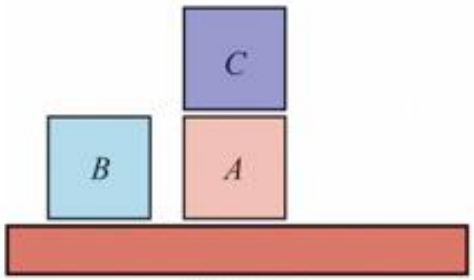
(没有任何条件，Spare将消失)

图 11-2　简单的备用轮胎问题

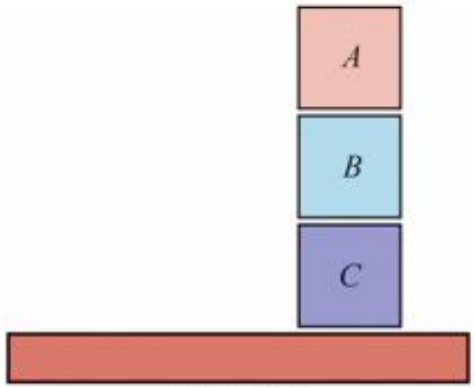
解：$[Remove(Flat, Axle), Remove(Spare, Trunk), PutOn(Spare, Axle)]$

11.1 例3：积木世界

- 最著名的规划问题之一：积木世界(blocks world)
  - 这个问域由一组放在桌子上的立方体形状的积木组成，积木能够被叠放，但是只有一块积木能够直接放在另一块的上面
  - 一个机器臂能够拿起一块积木并把它移到别的位置，可以在桌子上或在另一块积木上。
  - 机械臂每次只能拿起一块积木，所以它不能拿起一块上面有其他积木的积木。目标总是建造一堆或多堆积木，根据哪些积木在其他积木的上面进行指定



开始状态



目标状体

11.1 例3：积木世界

- 动作模式：
  - 定义谓词：On(b, x)表示积木b在x上，其中x可以是另一块积木或是桌子。
  - 定义动作：Move(b, x, y)表示将积木b从x的上面移到y的上面。
    - **移动b的一个前提是没有其他积木在它的上面**，在一阶逻辑中，这可以是¬∃x On(x,b)，或者用∀x ¬On(x,b)来表示
    - 基本的PDDL不允许量词，因此我们引入谓词Clear(x)，当x上没有东西时该谓词为真。

11.1 例3：积木世界

- 问题描述：

初始状态：A在桌子上，B在桌子上，C在A上，B、C、桌子上是干净的。

目标：A在B上，B在C上

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table))$
$Goal(On(A, B) \land On(B,C))$

动作模式1：Move

$Action(Move(b, x, y),$
PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$(b \neq x) \land (b \neq y) \land (x \neq y)$
EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$

(x可以为桌子，但y必须为积木)

动作模式2：MovetoTable

$Action(MoveToTable(b, x),$
PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land Block(x),$
EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$

(b必须在某个积木x上面)

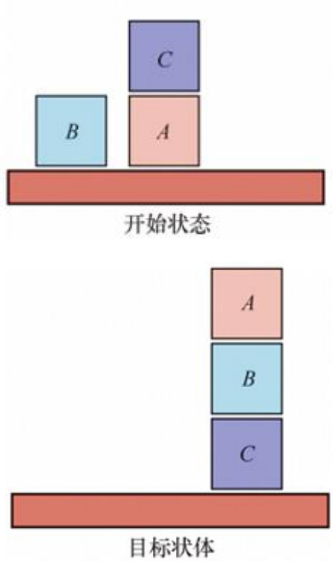


图 11-4　积木世界中的一个规划问题：建造一个 3 块积木构成的塔。一个解是序列[$MoveToTable(C, A)$, $Move(B, Table, C)$, $Move(A, Table, B)$]

解：[MoveToTable(C,A), Move(B, Table, C), Move(A, Table, B)]

- 经典规划的定义
- 状态空间搜索规划算法
- 规划图
- 其他经典规划方法

11.2 状态空间搜索规划算法

- 规划问题定义了一个搜索问题：
  - 从初始状态开始搜索状态空间，寻找一个目标
  - 也可以从目标开始后向搜索初始状态

1. 前向（前进）状态空间搜索
2. 后向（后退）相关状态搜索

11.2 状态空间搜索规划算法

- 将一个规划问题映射到一个搜索问题之后，可以使用任一启发式搜索算法或局部搜索算法来求解规划问题

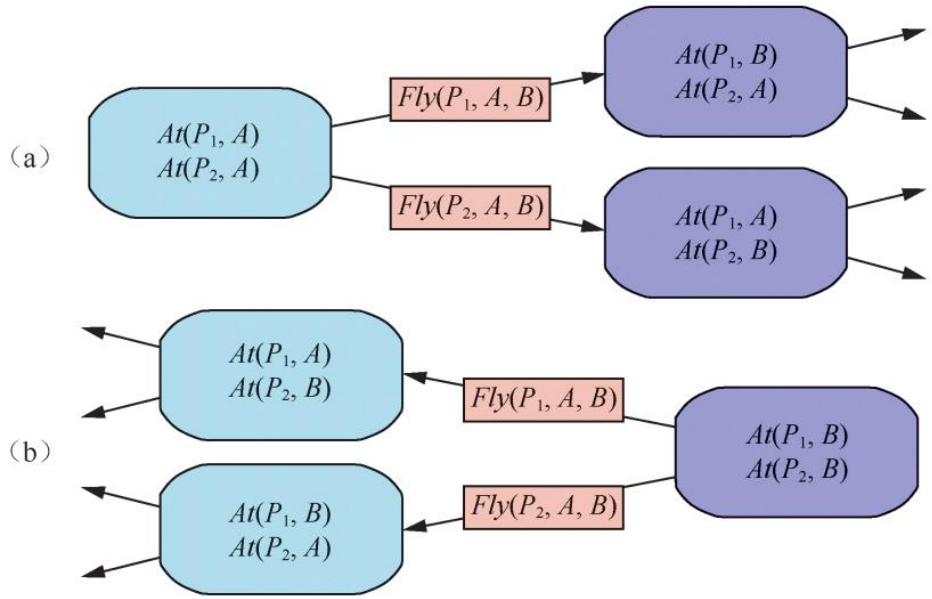


图 11-5　搜索规划的两种方法。(a) 在基本状态空间中前向（递进）搜索，从初始状态开始，利用问题的动作向前搜索目标状态集合中的一个成员。(b) 通过状态描述进行反向（回归）搜索，从目标开始，用逆动作反向搜索初始状态

11.2 前向（前进）状态空间搜索

- **前向搜索容易探索到无关动作**
- 规划问题经常有大的状态空间
- 启发式：依赖特定领域的启发知识，缩小搜索空间，使得前向搜索具有可行性
  - 10个机场，5架飞机，20件货物，目标是20件货物从A机场到B机场。
  - 很直观，解有41步，20件货物上A机场的某飞机、某飞机飞到B机场，卸载20件货物。
  - 但是他的状态空间非常大，飞机可以任意飞行，20件货物可以由不同飞机、不同批次运输。总计2000^{41}个节点

11.2 后向（后退）相关状态搜索

- 后向搜索中，从目标开始，向后应用动作，直到找到达到初始状态的步骤序列。
- **因为只考虑与目标（或当前状态）相关的动作，又被称为相关状态搜索**
- **只有当知道如何从一个状态描述后退到前驱状态描述时，后向搜索才能工作**

11.2 后向（后退）相关状态搜索

- 相关动作：
  - 前向搜索考虑“**适用动作**”，后向搜索考虑“**相关动作**”。相关动作的引入极大减少了分支，尤其是适用动作很多的时候。
  - **相关动作**：效果能与目标中的**一个文字合一**的动作，但这个效果**不能否定目标的任何部分。**
    - 复习 合一：找到对变量的一种赋值（置换），使得两条语句一致。
  - 例子：对目标¬Poor∧Famous，效果仅为Famous的动作是相关动作，而效果为Poor∧Famous的动作则不相关（因为否定了一部分目标）。

11.2 后向（后退）相关状态搜索

- 反向应用相关动作：
  - 反方向应用一个动作：给定一个目标g和一个动作a，从g通过a的反向推导会给出一个状态g’：

$$\text{Pos}(g') = (\text{Pos}(g) - \text{Add}(a)) \cup \text{Pos}(Precond(a))$$

$$\text{Neg}(g') = (\text{Neg}(g) - \text{Del}(a)) \cup \text{Neg}(Precond(a))$$

g中的正文字删去动作a**效果**中以正文字出现的流，并且添加a的前提

g中的负文字删去动作a**效果**中以负文字出现的流，并且添加a的前提

把目标中可以由a满足的状态，替换为a的前提

11.2 后向（后退）相关状态搜索

- 反向应用相关动作：
  - 目标： $At(C_2, SFO)$
  - 动作模式：

$$Action(Unload(c, p, a),$$
$$\text{PRECOND: } In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$$
$$\text{EFFECT: } At(c, a) \wedge \neg In(c, p))$$

  - 反向搜索：
    - 可以找到一种置换{c/C_2, a/SFO} 使得目标目标中的一个文字At(C_2, SFO)被满足（即合一），且没有否定目标其他部分，因此置换后，下面一个动作是一个**相关动作**：

$$Action(Unload(C_2, p', SFO)$$
$$\text{PRECOND: } In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$$
$$\text{EFFECT: } At(C_2, SFO) \wedge \neg In(C_2, p'))$$

    - 变量p被一个p′ 替代，代表某个飞机

11.2 后向（后退）相关状态搜索

- 反向应用相关动作：
  - 目标： $At(C_2, SFO)$
  - 前向动作：

$$Action(Unload(C_2, p', SFO)$$
$$\text{PRECOND: } In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$$
$$\text{EFFECT: } At(C_2, SFO) \wedge \neg In(C_2, p'))$$

  - 应用反向动作，得到新的目标状态：

$$g' = In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$$

    - 原来目标中的At(C_2, SFO)，删去动作效果中的正文字流At(C_2, SFO)，变为空，然后添加上A的前提。

11.2 规划的启发式

- 反向搜索的分支因子小于前向搜索、但反向搜索难以找到启发式方法，所以大多数系统还是倾向前向搜索+启发式方法。
- 一个启发式函数h(s)估计从状态到目标的距离，如果能为这个距离导出一个可采纳的启发式，可以使用A*搜索找到最优解
- 一个可采纳的启发式可以通过定义更容易求解的松弛问题而导出。这个更容易的问题的解的精确代价可成为原始问题的启发式

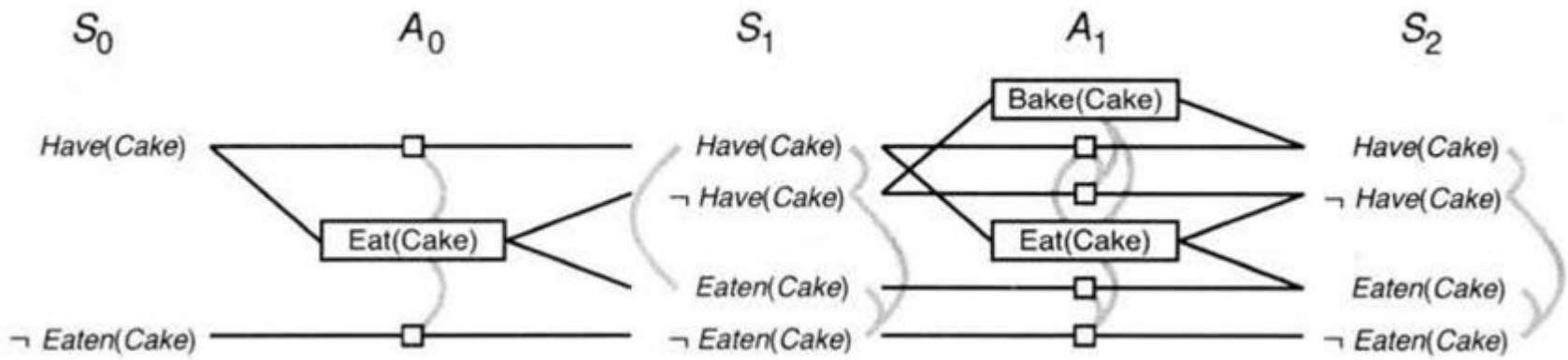
- 经典规划的定义
- 状态空间搜索规划算法
- 规划图
- 其他经典规划方法

11.3 规划图

- 前向搜索从初始状态出发，不断通过动作扩展后继状态，能够形成一个状态空间的树，然而，他是指数级的。
- 规划图是状态空间树的多项式级的近似。
- **规划图不能定性地回答是否可以从状态s_0达到目标g的问题，但可以估计出达到g需要多少步。**
- 当它报告目标不可达时，其估计总是正确的，而且从不高估步骤数，因此它是一个可采纳的启发式。

11.3 规划图

- **规划图是一个有层次的有向图：**
  - 首先是初始状态层S_0，由在S_0成立的每个流结点组成
  - 然后是层次A_0，由在S_0适用的每个动作结点组成
  - 然后是交叠的层次S_i，后面紧跟层次A_i；直到达到终止条件。



状态（S）和动作（A）交替出现

11.3 规划图

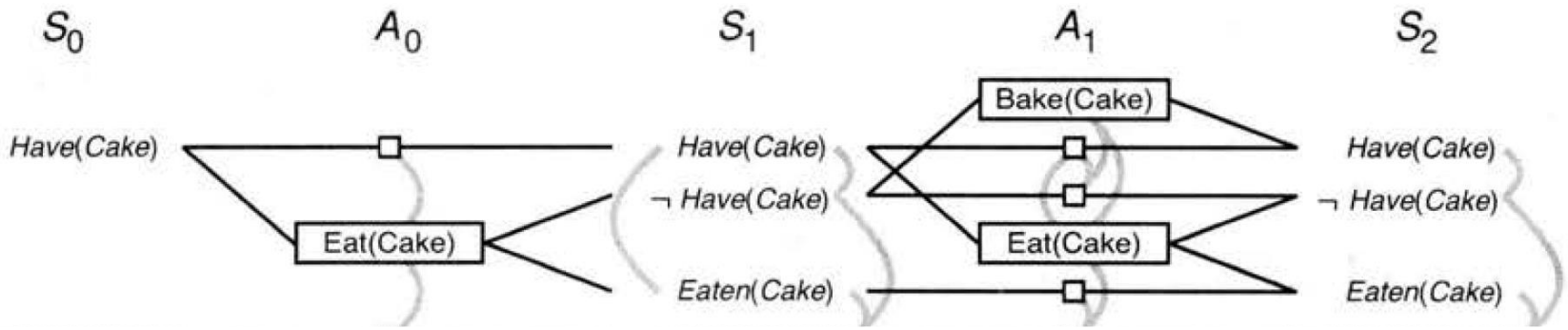
- S_i含有在时间i**有可能成立所有文字**，依赖于在前一个时间步骤执行的动作
  - 如果P或者¬P可能成立，那么两者在S_i中都要得到表示
- Ai含有在时间i**前提下有可能适用的所有动作**
  - 注：规划图只能用于命题规划问题——没有变量的规划问题。规划图被证明是解决困难规划问题的有效手段。

11.3 规划图

- 例子

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg Have(Cake)$
EFFECT: $Have(Cake))$

图 10.7 “有蛋糕而且吃蛋糕”的问题

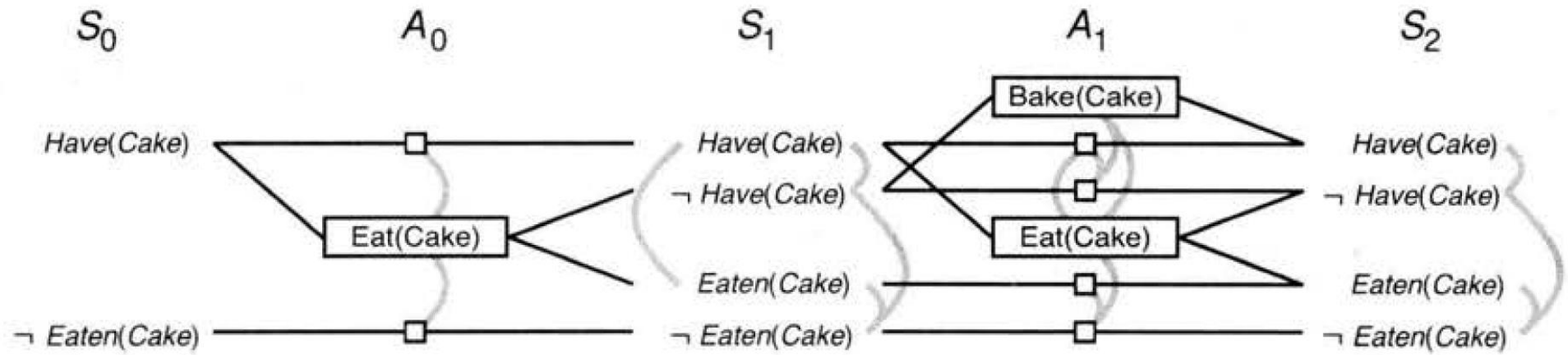


矩形：动作　　直线：前提和效果　　灰色曲线：互斥

小方格：持续动作

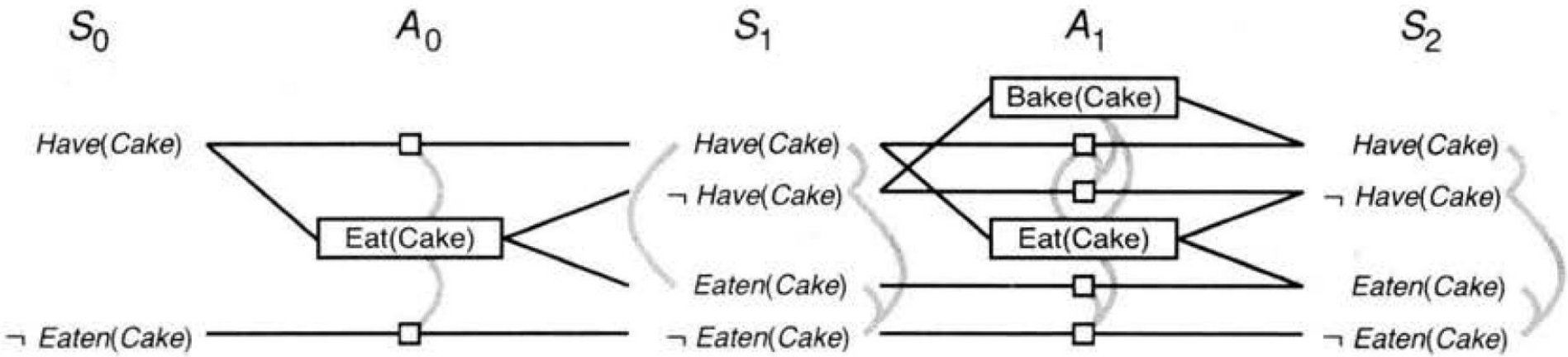
11.3 规划图

- 矩形：在层次A_i的每个动作连接到它在S_i的前提和在S_{i+1}的效果。因此一个文字出现是因为一个动作造就了它；
- 小方格：如果没有动作否定某个文字，他就可以持续。这里用持续动作（有时称为空操作）表示。



11.3 规划图

- A_0层含有在状态S_0可能发生的所有动作，但动作间有时候是有冲突的。图中灰色连线代表互斥连接。例如，Eat(Cake)与Have(Cake)或 ¬ Eaten(Cake)的持续动作是互斥的
- S_1表示信念状态：一个可能状态的集合。
- 这种方式继续下去，在状态层和动作层之间交叠，直到连续两层是一模一样的为止。这个时候，规划图达到了**稳定**



11.3 规划图

- **互斥动作**：如果下列三个条件之一成立，则两个动作间的互斥关系成立
  - 不一致效果：一个动作否定另一个动作的效果。

    例如：Eat(Cake)和Have(Cake)的持续动作具有不一致效果，因为它们对效果Have(Cake)没达成一致
  - 冲突：一个动作的效果之一是另一个动作的前提的否定。

    例如： Eat(Cake)和Have(Cake)的持续动作冲突，因为Eat(Cake)效果（不再Have(Cake)）否定了它的前提，导致他不能在同一时刻发生。
  - 竞争需要：一个动作的前提之一与其他动作的一个前提互斥

    例如：Bake(Cake)和Eat(Cake)是互斥的，因为它们对前提Have(Cake)的值是竞争的

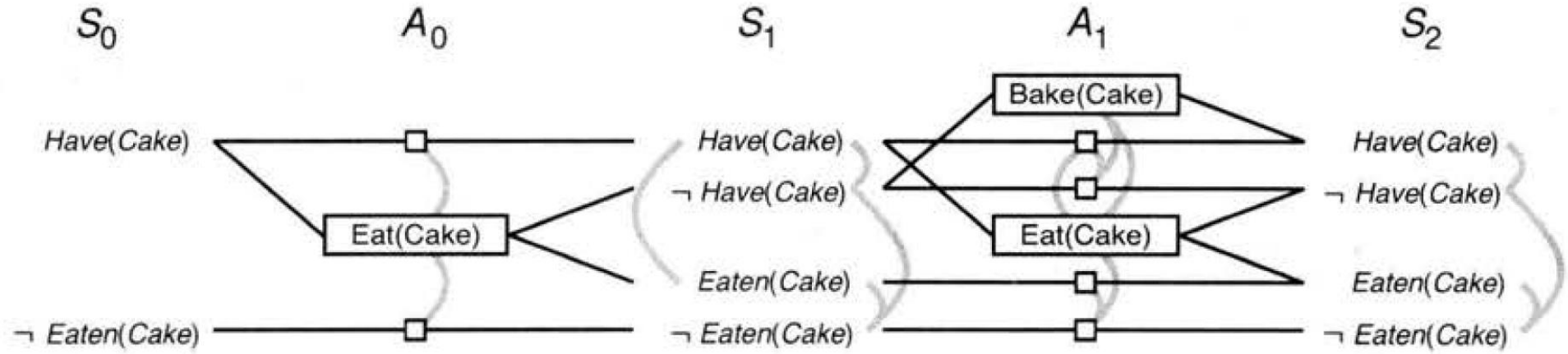
11.3 用于启发式估计的规划图

- 规划图一旦建立，就可以提供很多搜索需要的信息：
  1. 可达性：如果有目标文字没有在图的最后一层出现，那么问题是不可解的。
  2. 启发函数（层次代价）：可以估计出从状态s获得任何目标文字g_i的代价为**从初始状态s开始，到构建出的规划图中首次出现 g_i 的层**。可以用来估计搜索步数。
     - 估计并不绝对准确，因为一层可以有多个动作，而层次代价只统计层次数，而不是动作数。

- 序列化规划图
  - 要求任何时间步实际上只有一个动作会发生；通过在每对非持续动作间增加互斥连接可以实现这一点。
  - **从序列化图中提取的层次代价经常是实际代价的十分合理的估计。**

11.3 用于启发式估计的规划图

- 代价估计：最大层、层次和、集合层次
  - **最大层启发式**
    - 取所有目标中的最大的层次代价；
    - 可采纳（不会过高估计），但不一定是最准确的

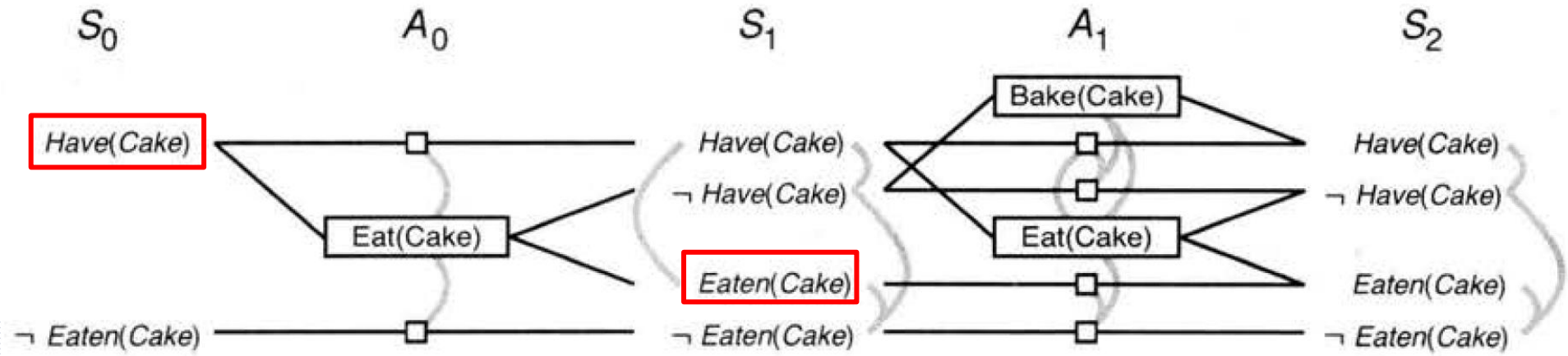


Have(cake) （0步），Eaten(Cake)：1步

Max(0,1)=1

11.3 用于启发式估计的规划图

- 代价估计：最大层、层次和、集合层次
  - **层次和启发式**
    - 遵守**子目标独立性假设**，返回子目标的**层次代价之和**；
    - 可能是不可采纳的，但在实际中对可规模分解的问题效果很好

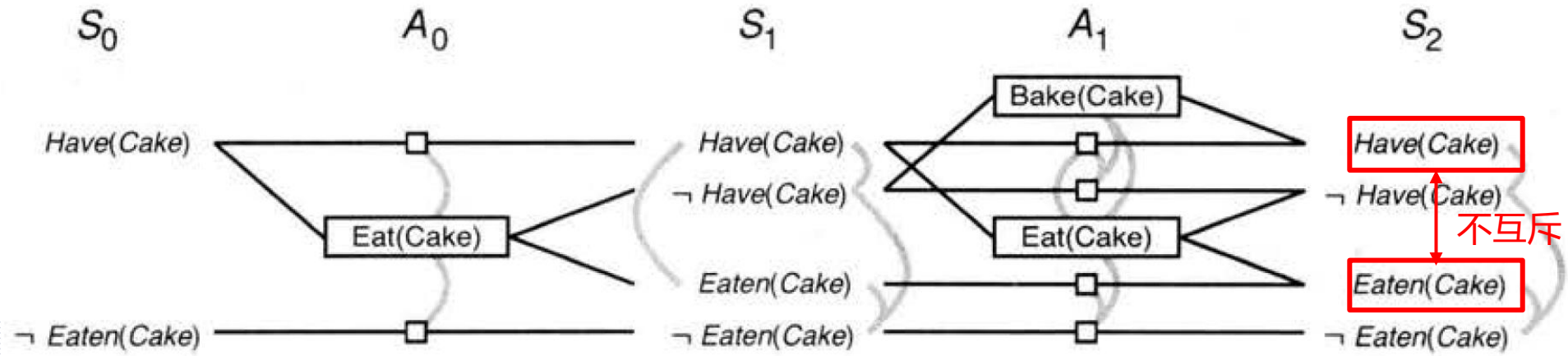


Have(cake) （0步首次出现）∧ Eaten(Cake) （1步首次出现）

0+1=1

11.3 用于启发式估计的规划图

- 代价估计：最大层、层次和、集合层次
  - **集合层次启发式**
    - 找到一个**目标中的所有文字出现的层次**，且没有任何一对目标文字是互斥的
    - 他是可采纳的，比最大层启发式占优势，对于在子规划间需要交互处理的任务效果非常好



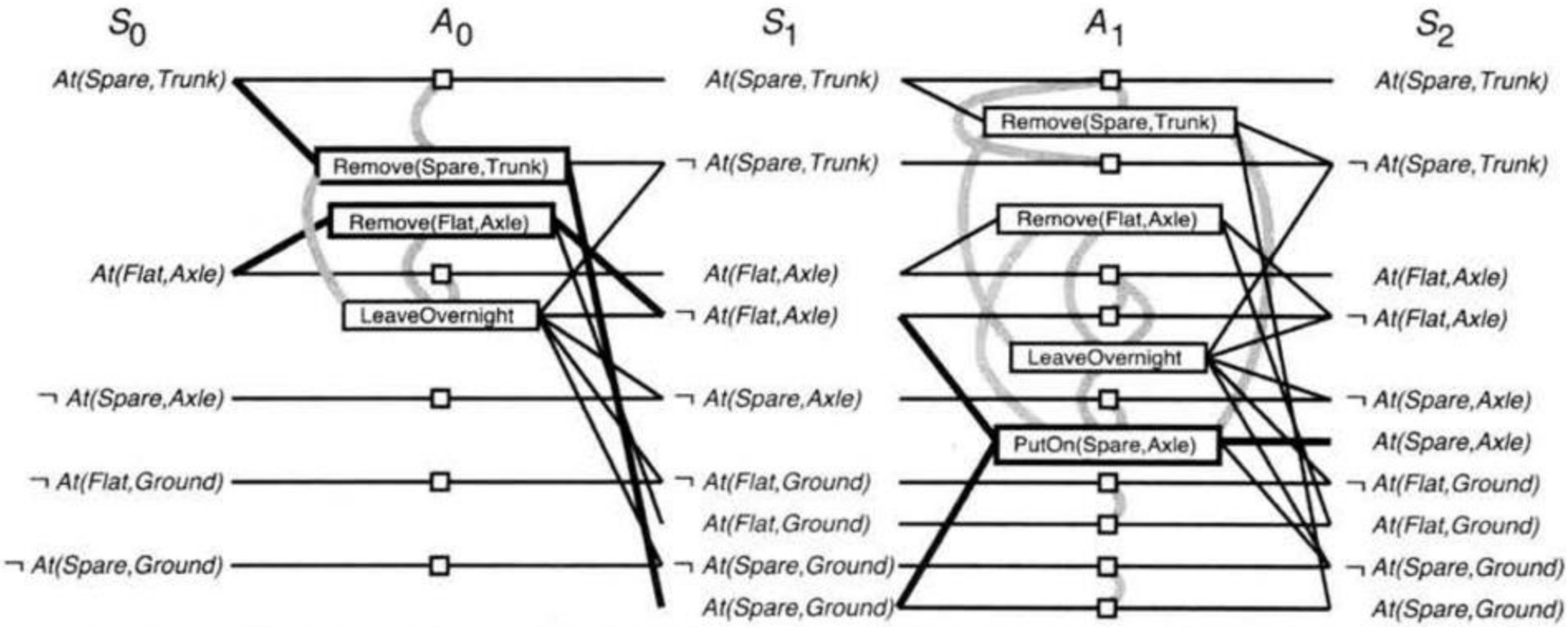
S2中首次出现，且不互斥，2步

11.3 GRAPHPLAN算法

- GraphPlan算法
  - 如何直接从规划图中抽取一个规划（动作序列），而不只是使用规划图来提供步数估算。
  - GRAPHPLAN算法反复地调用EXPAND-GRAPH向规划图增加一层。
    - 一旦所有目标在图中出现，没有互斥，GRAPHPLAN就调用EXTRACT-SOLUTION**搜索解决问题的规划**；
    - **如果提取规划失败了**（扩展到包含所有目标的一层，不代表有对应的规划），就扩展出另一层再试，当没有理由再继续的时候就以失败而终止。

11.3 例子：备用轮胎问题

- 初始At(Spare, Trunk), At(Flat, Axle)，目标： At(Spare, Axle)

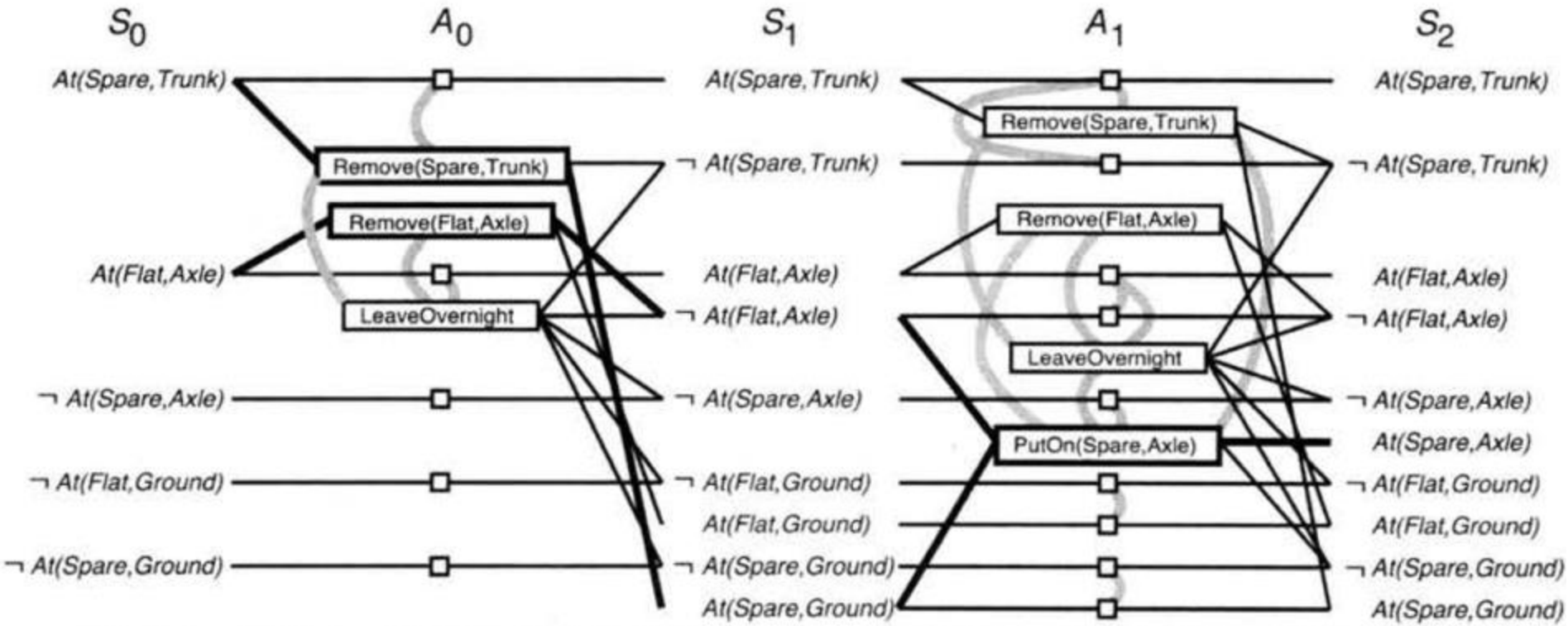


S0步：

- S0中没有At(Spare, Axle)，因此不用生成规划

11.3 例子：备用轮胎问题

- 初始At(Spare, Trunk), At(Flat, Axle)，目标： At(Spare, Axle)

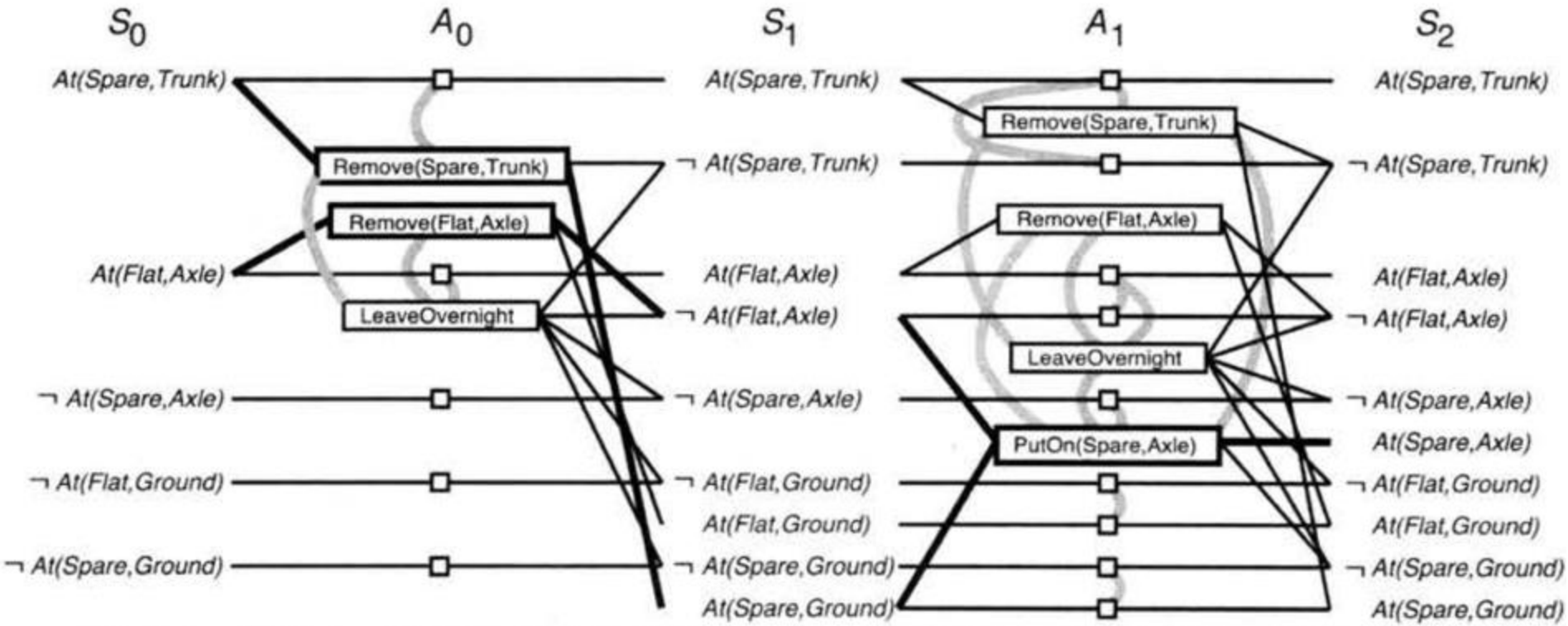


A0步：

- 扩展，在A0层尝试所有前提在S0中的动作，Remove(Spare, Trunk), Remove(Flat, Axle), LeaveOvernight
- 所有持续动作

11.3 例子：备用轮胎问题

- 初始At(Spare, Trunk), At(Flat, Axle)，目标： At(Spare, Axle)

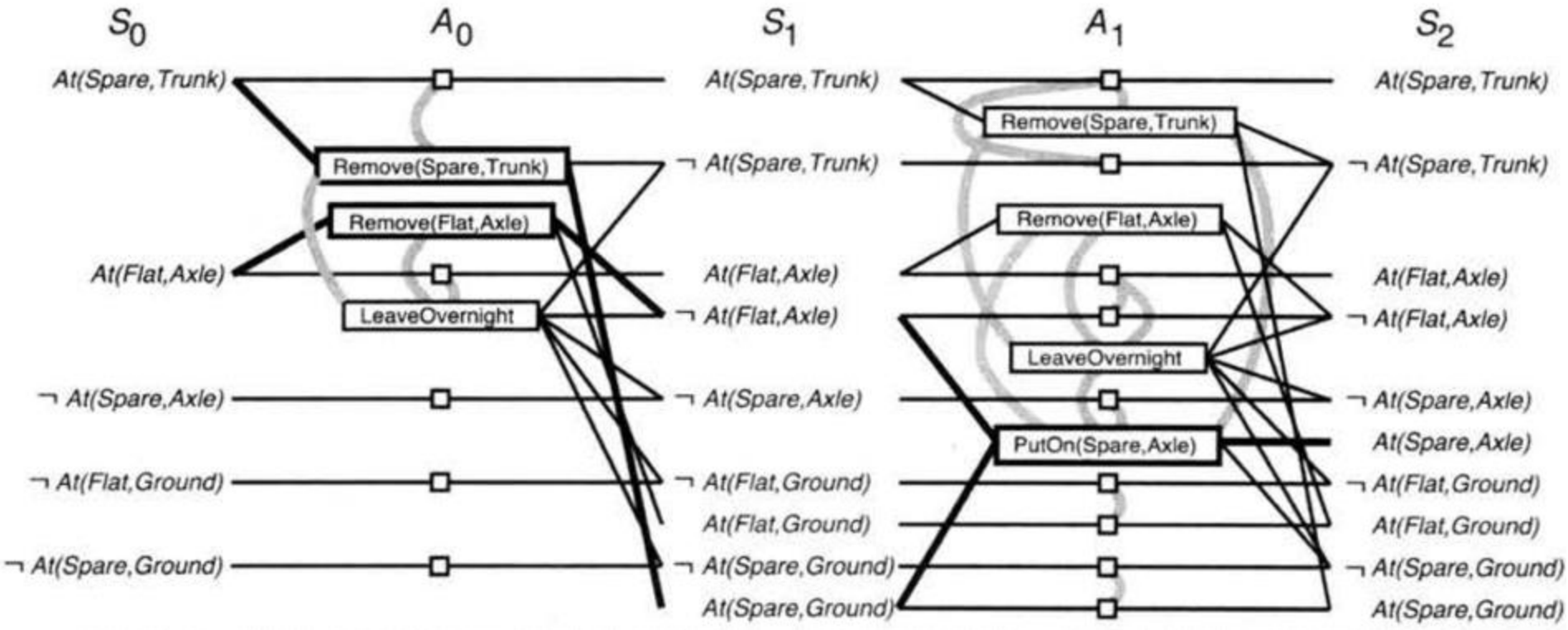


S1步：

- 得到了4个新的状态： ¬At（Spare, Truck），¬At（Spare, Truck），At(Flat, Ground)，At(Spare, Ground)
- 仍然没有目标状态，不用生成规划

11.3 例子：备用轮胎问题

- 初始At(Spare, Trunk), At(Flat, Axle)，目标： At(Spare, Axle)

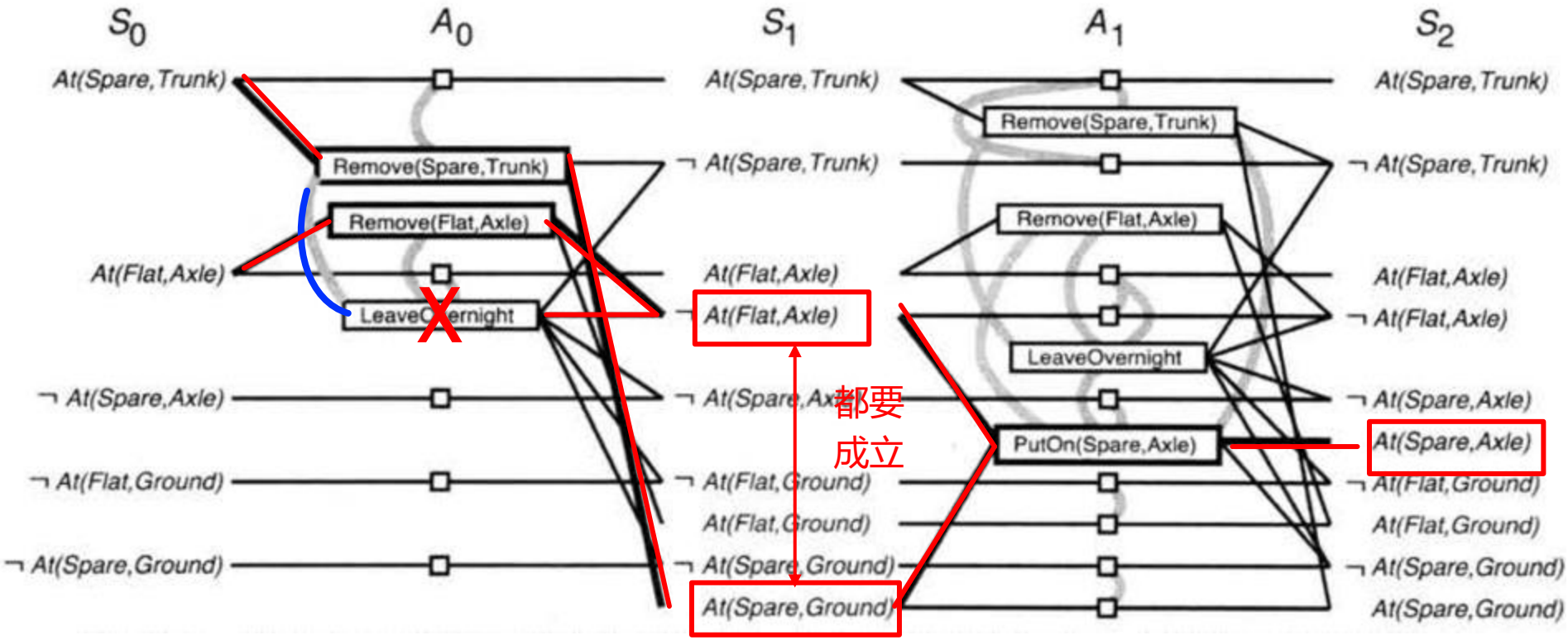


A1步：

- 所有动作都可适用，添加所有动作。
  - 注意有些动作是互斥的。比如Remove(Spare, Trunk)和LeaveOvernight，因为LeaveOvernight和At(Spare, Ground)结果冲突
- 添加持续性动作

11.3 例子：备用轮胎问题

- 初始At(Spare, Trunk), At(Flat, Axle)，目标： At(Spare, Axle)



S2步：

- 找到了目标At(Spare, Axle)，且目标间不互斥。一个解可能出现了。
- 后向搜索输出规划

11.3 GRAPHPLAN的终止

- 不能在规划图一达到稳定就停止扩展
- 规划图达到稳定后，还需要扩展多远呢?
  - 如果函数EXTRACT-SOLUTION没能找到解，那么至少有一组目标没有实现并被标注为no-good
  - 如果下一层里no-good的数量可能更少，那么应该继续
  - 一旦图和no-good都达到稳定，而又没有找到解，可以失败而终止

- 经典规划的定义
- 状态空间搜索规划算法
- 规划图
- 其他经典规划方法

11.4 其他经典规划方法

- 转换为布尔满足性问题
  - 转换为命题逻辑，用SATPLAN方法求解
- 转换为一阶逻辑推理：情景演算
  - 没有好的实际规划程序
- 转换为约束满足问题
- 转换为偏序规划的规划
  - 每个动作是图中的一个节点，每个动作的前提都有一条来自另一个动作的边，表明形成这一前提的前驱动作
  - 在规划空间而非世界状态中搜索，通过插入动作来满足条件
  - 处理具有独立子问题的规划问题的最佳方法

- 经典规划的定义
- 状态空间搜索规划算法
- 规划图
- 其他经典规划方法
- 具身智能VLA规划

11.5 规划范式的跨越

状态空间的重构

- 经典规划（符号化）：状态是因子化的。必须利用预定义的谓词（如 On(x, y), Clear(x)）来描述世界状态。
- 具身规划（VLA）：状态是连续化的。利用视觉编码器（如 ViT）提取高维特征，直接“看”懂环境。

动作与物理转换的内化

- 经典规划（显式规则）：必须人工编写动作模式, 明确定义前提和效果。
- 具身规划（隐式知识）：世界知识内化，大模型在预训练中“学到了”物理常识和因果关系 。

11.5 规划机制的重塑

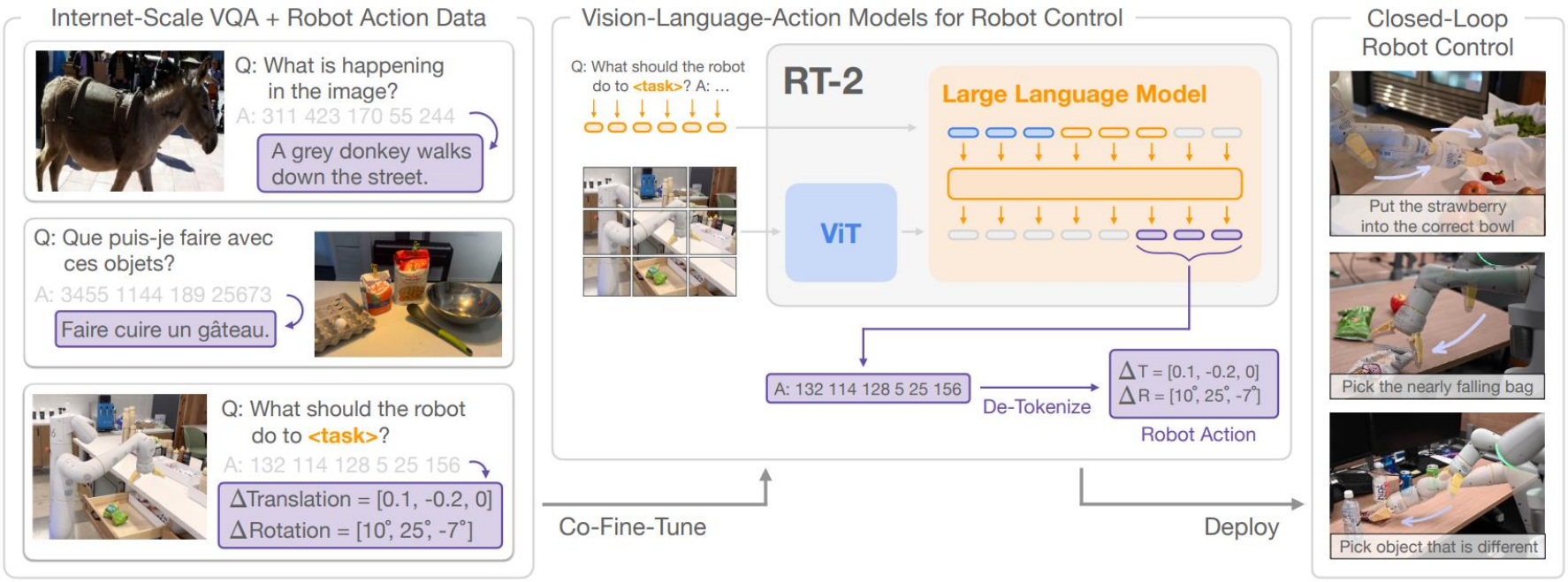
- **搜索范式 vs. 生成范式**
  - 经典规划（显式搜索）：依赖前向状态空间搜索。核心难点在于设计高效的启发式函数来指引搜索方向，以应对指数级增长的状态空间。
  - 具身规划（条件生成）：基于多模态条件概率建模。规划不再是“在树中遍历节点”，而是基于当前的视觉观测（o）和语言指令（L），直接生成符合分布的最优动作序列（A），即建模$P(A|O,L)$。

11.5 规划机制的重塑

- **核心技术：多模态对齐与表征**
  - 从“规则匹配”到“表征对齐”：经典规划依靠人工定义的规则（前提/效果）来连接符号；VLA 依靠高维向量空间的语义对齐 。
  - 统一表征空间：无论是将动作离散化为 Token（如 RT-2），还是编码为 连续特征向量（如 π_0），核心都在于将“物理动作”映射到了与“语言/视觉”共享的语义空间中。打破了感知与控制的壁垒，让模型可以用处理语言或生成图像的逻辑直接“生成”机器人的物理行为。

11.5 单系统VLA

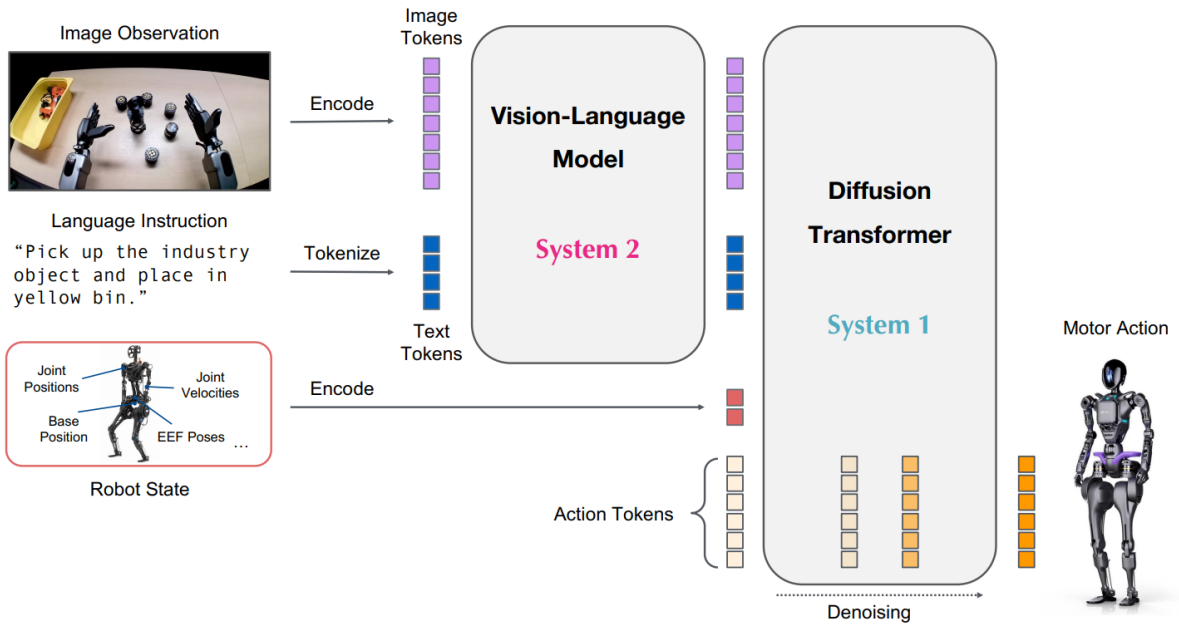
- **核心定义：感知决策一体化**
  - 紧耦合架构：无显式规划模块，感知、推理与控制在同一个 Transformer 模型中完成，表现为一种“直觉反应”。
  - 典型案例：Google DeepMind RT-2。利用互联网图文知识，实现了从语义理解到机器人控制的直接迁移。



11.5 双系统VLA

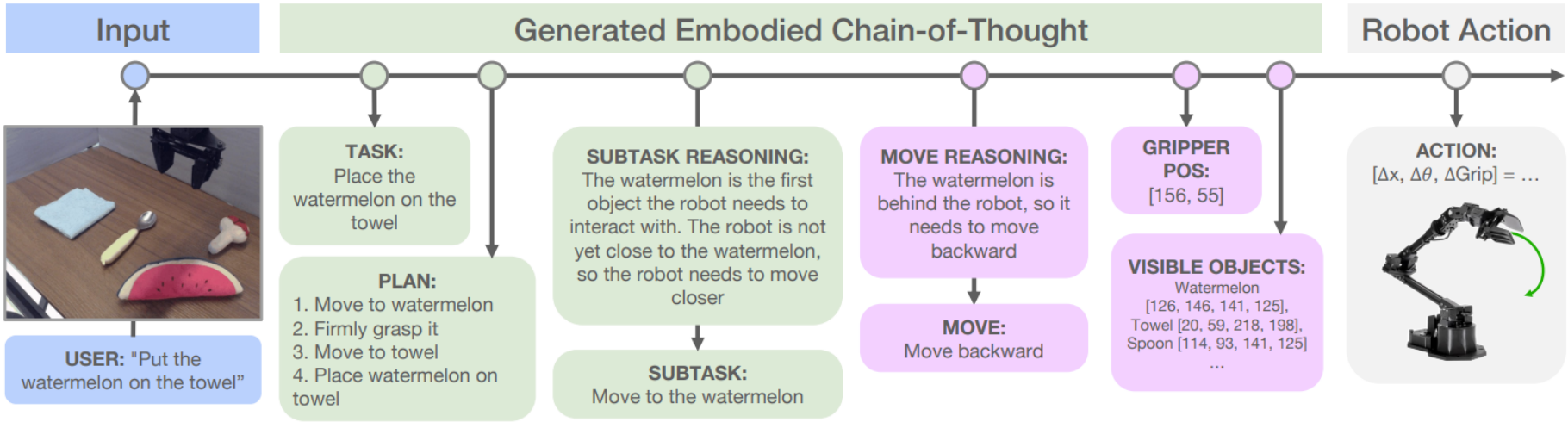
设计理念：快慢系统协作

- 系统 2 (大脑/VLM)：语义规划。负责处理抽象指令、逻辑推理，输出子目标序列。(慢思考)
- 系统 1 (小脑/Policy)：动作执行。负责将子目标转化为高频、精准的电机控制信号。(快反应)
- 典型案例：NVIDIA GR00T N1



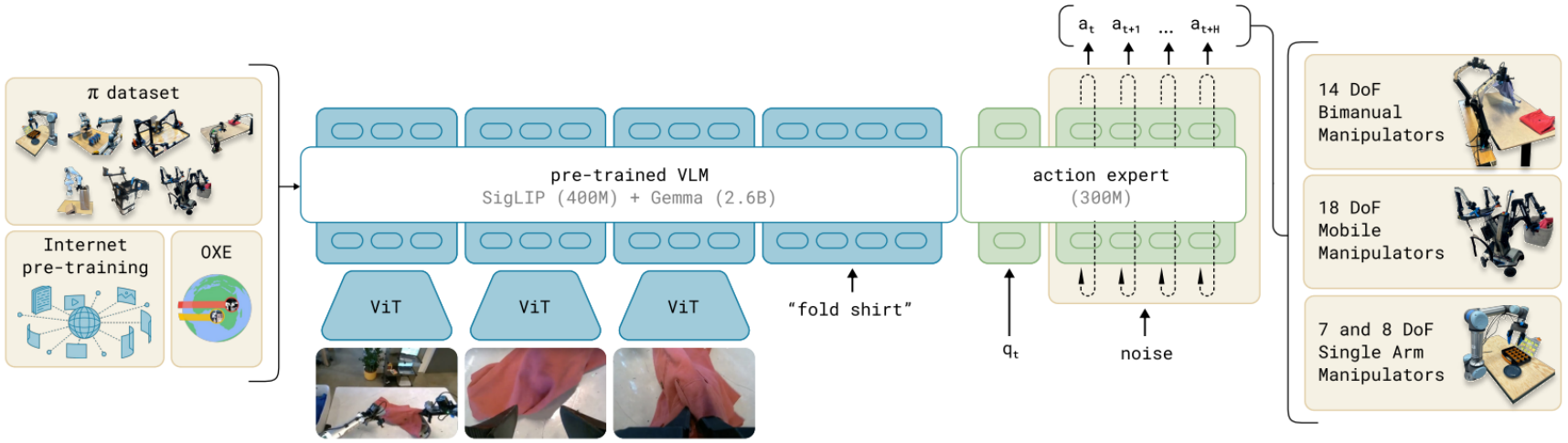
11.5 VLM 规划实例

- 显式任务分解：模型不直接输出动作，而是先生成宏观计划→子任务→物理常识推理的思维链条。
- 语义与物理的桥梁：通过中间层的移动推理（如：“西瓜在后面，所以需要向后移”），将抽象的语义指令精准转化为具体的机器人动作关节角度/末端位姿）。



11.5 前沿展望：生成式规划与世界模型

- **引入流匹配：连续动作的“生成式规划”**
  - 从离散到连续：打破了 RT-2 等模型“逐个蹦词”的限制，转向直接生成连续、平滑的时空轨迹 。如 π_0
- **引入世界模型：具身智能的“思维沙盘”**
  - 超越反应式 AI：目前的单/双系统大多是基于当前观测的“直觉反应” 。未来的规划需要反事实推理：即在行动前并在脑海中“模拟”动作后果 。



本章小结

- 在确定性的、完全可观察的、静态的环境中定义了规划问题
  - 规划系统是在状态和动作的显式因子化表示下运作的求解算法。这些表示使得推导有效的领域无关的启发式和开发强大且灵活的问题求解算法成为可能。
  - 规划领域定义语言 PDDL 用文字的合取描述初始状态和目标状态，用行动的前提和效果描述行动。
  - 状态空间搜索可以前向（递归）或反向（回归）运行。
  - 一个规划图可从初始状态开始增量式地构造出来。每一层包含一个所有在那个时间步出现的文字和动作的超集，并且对文字之间或动作之间互斥关系进行编码。

历史与趣闻

- 人工智能规划起源于对状态空间搜索、定理证明和控制理论的研究。Stripes（Fikes and Nilsson, 1971, 1993），是第一个主要的规划系统，是为 SRI 的 Shakey 机器人设计的规划器。该程序的第一个版本运行在一台只有 192 KB 内存的计算机上。它的总体控制结构以 GPS 为模型，即通用问题求解器（Newell and Simon, 1961），这是一个使用目标 - 手段分析的状态空间搜索系统。
- Strips 表示语言演变为动作描述语言（Action Description Language，ADL）（Pednault, 1986），然后是问题域描述语言（Problem Domain Description Language，PDDL）（Ghallab et al., 1998），它自 1998 年起就被用于国际规划竞赛（International Planning Competition）。其最新版本是PDDL 3.1（Kovacs, 2011）。

历史与趣闻

- 20 世纪 70 年代早期，规划器通过计算每个子目标的子规划来分解问题，然后按照某种顺序将子规划串在一起。这种方法被萨切尔多蒂（Sacerdoti, 1975）称为线性规划，很快它就被发现是不完备的。它不能解决一些非常简单的问题。
- 双向搜索（见 3.4.5 节）被认为受制于缺乏启发式，但通过使用反向搜索在目标周围创建一个围栏（perimeter），然后提出一个启发式方法向围栏前向搜索的尝试已经取得了一些成功（Torralba et al., 2016）。SymBA* 双向搜索规划器（Torralba et al., 2016）赢得了 2016 年的国际规划竞赛。

历史与趣闻

- 研究人员关注于 PDDL 和规划范式，以便能够使用领域无关的启发式方法。霍夫曼（Hoffmann, 2005）分析了忽略删除列表启发式方法的搜索空间。埃德坎普（Edelkamp, 2009）和哈斯卢姆等人（Haslum et al., 2007）阐述了如何为规划启发式方法构建模式数据库。
- 布卢姆和弗斯特（Blum and Furst, 1997）用他们的 Graphplan 系统重振了规划领域，它比当时的偏序规划快了几个数量级。布赖斯和坎班帕蒂（Bryce and Kambhampati, 2007）给出了规划图的概述。情景演算在规划中的应用由约翰·麦卡锡（McCarthy, 1963）提出，并由雷·赖特（Ray Reiter）（Reiter, 2001）加以改进。

课程作业

- 1、描述问题求解与规划之间的不同和相似之处
- 2、猴子与香蕉问题：
  - 起初猴子位于A，香蕉位于B，箱子位于C。猴子和箱子的高度都是Low，但是如果猴子爬到箱子上面，它的高度就跟香蕉一样是High。猴子可用的动作包括从一个位置走到另一个位置Go，将对象从一个地方推到另一个地方的Push，爬上一个对象的ClimbUp和爬下一个对象的ClimbDown，抓住一个对象的Grasp和放开一个对象的Ungrasp。
- a）写出初始状态描述
- b）写出六个动作模式

THANKS

Q & A