Chapter 10 classical planning

1 规划问题定义（PDDL）为一个搜索问题
2 前向搜索、后向搜索，及搜索的启发式
3 从规划图获得启发式及提取规划

规划问题定义（PDDL）为一个搜索问题

- 航空货物运输问题：货物C1在SFO机场，C2在JFK机场，飞机P1在SFO机场，飞机P2在JFK机场。将C1运到JFK，将C2运到SFO。
- 动作Load, Unload, Fly

PDDL {10.1}

- 规划：设计一个动作规划以达到目标
- ch03的基于搜索的问题求解Agent和ch07的逻辑Agent是一种规划Agent
- 用PDDL能够描述ch03和ch07无法描述的规划问题
- 将规划问题定义为一个搜索问题（要素有哪些？）

PDDL {10.1}

- **状态**表示为**流**（fluent）的合取，
- 流是基元的（无变量的）、无函数的原子。
  - 例如，$Poor \wedge Unknown$可能代表倒霉的Agent的状态；在一个包裹传递问题中的状态可以是 $At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$
  - 不是流：$At(x,y)$ 、$\neg Poor$、$At(Father(Fred), Sydney)$

PDDL {10.1}

- **动作**可用**动作模式**来描述，动作模式隐式描述了ACTIONS(*s*)和RESULT(*s*, *a*)。
- 例如直升机从一个地点飞行到另一个地点的动作模式：

  Action(*Fly*(*p*, *from*, *to*),

  PRECOND: *At*(*p*, *from*) ∧*Plane*(*p*) ∧*Airport*(*from*) ∧*Airport*(*to*)

  EFFECT: ¬*At*(*p*,*from*) ∧*At*(*p*,*to*))
- PDDL根据什么发生了变化来描述一个动作的结果；不提及保持不变的东西。动作的前提和效果都是文字（原子语句或原子语句的否定）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。

PDDL {10.1}

- 通过为所有的变量代入值而得到**基元动作**：

Action(*Fly*(P_1, *SFO*, *JFK*),

PRECOND: *At*(P_1, *SFO*) ∧*Plane*(P_1) ∧*Airport*(*SFO*)
∧*Airport*(*JFK*)

EFFECT: ¬*At*(P_1, *SFO*) ∧*At*(P_1, *JFK*))

PDDL {10.1}

- ACTIONS(*s*)：一个动作*a*能在状态*s*被执行，如果_____。

PDDL {10.1}

- ACTIONS(*s*)：一个动作*a*能在状态*s*被执行，如果*s*蕴涵了*a*的前提。蕴涵也可用集合语义来表达：*s* ⊨ *q*当且仅当*q*中的正文字都在*s*中，且*q*中的负文字都不在*s*中。用形式符号，我们可以说
  - $(a \in \text{ACTIONS}(s)) \Leftrightarrow s \models \text{PRECOND}(a)$,
- 如果状态*s*满足前提，我们称在状态*s*动作*a*是**适用的**（applicable）。

PDDL {10.1}

- RESULT(*s*,*a*)：**在状态*s*执行动作*a*的结果**定义为状态*s'*，它由一组流（fluent）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（我们称之为删除列表delete list或DEL(*a*)），并增加在动作效果中以正文字出现的流（我们称之为增加列表add list或ADD(*a*)）：
- $s' = \text{RESULT}(s,a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$

PDDL {10.1}

- **初始状态**是基元原子的合取。（对于所有状态，使用封闭世界假设，这意味着任何没有提到的原子都是假的。）
- **目标**就像前件：是可以含有变量的文字（正文字或负文字）的合取，像 $At(p,SFO) \wedge Plane(p)$。变量是存在量化的
- 当我们能找到一个动作序列，使得在蕴涵了目标的状态*s*结束，问题就得到了解决。例如，状态 $Plane(Plane_1) \wedge At(Plane_1,SFO)$ 蕴涵了目标 $At(p, SFO) \wedge Plane(p)$。

前向搜索、后向搜索，以及搜索的启发式

前向状态空间搜索 {10.2}

- 规划问题的描述定义了一个搜索问题，启发式搜索算法（第3章）或局部搜索算法（第4章）可求解规划问题。
- 前向搜索从初始状态开始搜索状态空间，寻找一个目标。
- s'=RESULT(s,a) = (s – DEL(a))∪ADD(a)

- 前向搜索的缺陷是什么？

前向状态空间搜索{10.2}

- 从规划搜索的初期（约1961年）直到约1998年，人们认为前向状态空间搜索太低效而不能实际应用。
- 首先，前向搜索**容易探索到无关动作**。其次，规划问题经常有**大的状态空间**，从而需要精确的**启发式**。尽管规划的许多现实世界应用依赖于特定领域的启发知识，还是能自动导出非常强的独立于领域的启发知识；这使得前向搜索具有可行性。

前向状态空间搜索{10.2}

- 考虑从在线书店购买《AI: A Modern Approach》这本书。有一个动作模式*Buy*(*isbn*)，效果是*Own*(*isbn*)。ISBN是10位数，因此这个动作模式表示100亿个基元动作。一个无启发的前向搜索将需要枚举这100亿个基元动作，以找到通向目标的动作。

后向相关状态搜索{10.2}

- 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。只考虑与目标（或当前状态）相关的动作（**导致当前目标状态的规划中可以作为最后一个步骤的那些动作**）。每一步考虑一组相关状态(就像信念状态(ch04))，而不是单个状态。

后向相关状态搜索{10.2}

- 给定一个基元目标描述*g*和一个基元动作*a*，从*g*经*a*回退得到状态描述*g′*，定义为
- g′ = ?

- $g' = (g - \mathrm{ADD}(a)) \cup Precond(a)$

什么样的动作是相关的？

为何不是**DEL(a)?**

后向相关状态搜索{10.2}

- 哪些动作是相关的？
- 导致当前目标状态的规划中可以作为最后一个步骤的那些动作

- 如果目标是$A \wedge B \wedge C$，一个动作的效果是$A \wedge B \wedge \neg C$，那么它是相关的吗？一个动作的效果是D，那么它是相关的吗？

后向相关状态搜索{10.2}

- 哪些动作是相关的？
- 导致当前目标状态的规划中可以作为最后一个步骤的那些动作
- 动作的效果（或者为正，或者为负）之一必须与目标的一个元素一致。动作必须不能具有任何否定目标的某个元素的效果（正或负）。
- 如果目标是$A \wedge B \wedge C$，一个动作的效果是$A \wedge B \wedge \neg C$，那么它不是相关的

后向相关状态搜索{10.2}

- 给定目标$At(C_2,SFO)$，$Unload(c,p,a)$的几个实例是相关的：我们可以选择任何特定的飞机来卸载，或者我们通过使用动作$Unload(C_2,p',SFO)$而可以不指定飞机。通过总是使用将最一般合一元置换到（标准化的）动作模式中得到的动作，我们可以减小分支因子而不会删除（漏掉）任何解。

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

后向相关状态搜索{10.2}

- 考虑目标*Own*(0136042597)，给定一个有100亿个ISBN的初始状态，以及单个的动作模式
- *A* = *Action*(*Buy*(*i*), PRECOND: *ISBN*(*i*), EFFECT: *Own*(*i*))。
- 将目标*Own*(0136042597)与（标准化的）效果*Own*(*i′*)合一，得到置换 θ={*i′*/0136042597}。然后，我们可以采用动作*Subst*(θ,*A′*)而后退，得到前驱状态描述*ISBN*(0136042597)。

规划的启发式{10.2.3}

- 启发式函数$h(s)$估计从状态到目标的距离
- 如何获得可采纳的启发式？

规划的启发式{10.2.3}

- 启发式函数$h(s)$估计从状态到目标的距离
- 如何获得可采纳的启发式？——松弛问题
  - 增加边或者合并结点
  - 针对规划问题如何设计？

规划的启发式{10.2.3}

- 启发式函数$h(s)$估计从状态到目标的距离
- 如何获得可采纳的启发式？——松弛问题
  - 增加边或者合并结点
- 忽略前提启发式（增加边/松弛前提）
- 忽略删除列表（增加边/松弛效果）
- 忽略一些流（合并结点/抽象状态）

忽略前提启发式{10.2.3}

- **忽略前提启发式（增加边）**丢弃动作中的所有前提。在每一个状态，每一个动作都变得适用，而且任何单个目标流能够一步获得（如果存在一个适用动作——否则，问题是不可能解的）
  - 因为一个动作可能抵消另一个动作的效果。因此对于许多问题，我们如此对动作进行松弛：去掉所有前提，去掉所有效果（目标中的文字除外）。然后，我们统计需要的最小动作数，使这些动作的效果的并集满足目标。

忽略前提启发式{10.2.3}

- **选择性地忽略**动作的前提是可能的。考虑8码问题。将它编制为一个使用单个动作模式*Slide*的规划问题：
- $Action(Slide(t,s_1,s_2)$ ，

  PRECOND: $On(t,s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1,s_2)$

  EFFECT: $On(t,s_2) \wedge Blank(s_1) \wedge \neg On(t,s_1) \wedge \neg Blank(s_2))$
- 去掉前提_____，那么一个动作中任何数字块能够移到任何方格，从而得到位置错误的数字块数作为启发式

忽略前提启发式{10.2.3}

- **选择性地忽略**动作的前提是可能的。考虑8码问题。将它编制为一个使用单个动作模式*Slide*的规划问题：
- $Action(Slide(t,s_1,s_2)$,

  PRECOND: $On(t,s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1,s_2)$

  EFFECT: $On(t,s_2) \wedge Blank(s_1) \wedge \neg On(t,s_1) \wedge \neg Blank(s_2))$
- 去掉前提$Blank(s_2) \wedge Adjacent(s_1,s_2)$，那么一个动作中任何数字块能够移到任何方格，从而得到位置错误的数字块数作为启发式

忽略删除列表启发式{10.2.3}

- 另一种增加边的启发式是**忽略删除列表**（ignore delete list）启发式
- 假设所有目标和前提只含有正文字（如果不是，则用新的正文字P'替换目标或前提中的负文字$\neg P$）
- 通过从所有动作中去掉删除列表（效果中的负文字），得到松弛版本。这使得没有动作会抵消另一个动作的效果;从而使得解短于原始空间的解。

忽略一些流{10.2.3}

- 通过形成**状态抽象**（state abstraction）而减少状态数的松弛方法：
- 状态抽象的最简单方式是**忽略一些流**
- 抽象状态空间中的一个解将短于原始空间中的一个解（因而将是一个可采纳的启发式）

目标分解{10.2.3}

- 定义启发式的关键思想是**分解**（decomposition）：将一个问题分成多个部分，独立地解决每个部分，再组合各部分。
- 假设目标是一组流G，我们将其划分为不相交的子集$G_1,\ldots,G_n$。然后我们找到各子目标的解规划$P_1,\ldots,P_n$。max_i COST(P_i)是可采纳的，而且有时是精确正确的：P_1可能偶然实现所有G_i。但大多数情况下，实际估计值过低。

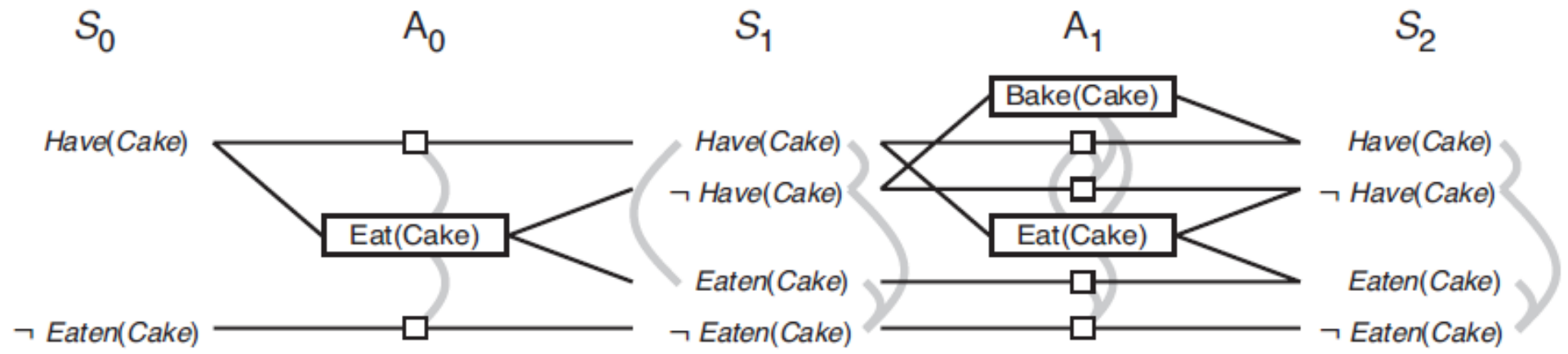
目标分解{10.2.3}

- 最好的情况是当我们确定G_i和G_j相互独立的时候。如果P_i的效果不改变P_j的所有前提和目标，那么估计值$\text{COST}(P_i)+\text{COST}(P_j)$是可采纳的，而且比最大估计值更准确。

从规划图获得启发式及提取规划

何谓规划图？{10.3}

- 一个规划图组织成有**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适用的每个基元动作结点组成；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（是什么？）。
- 规划图只能用于命题规划问题——没有变量的规划问题。



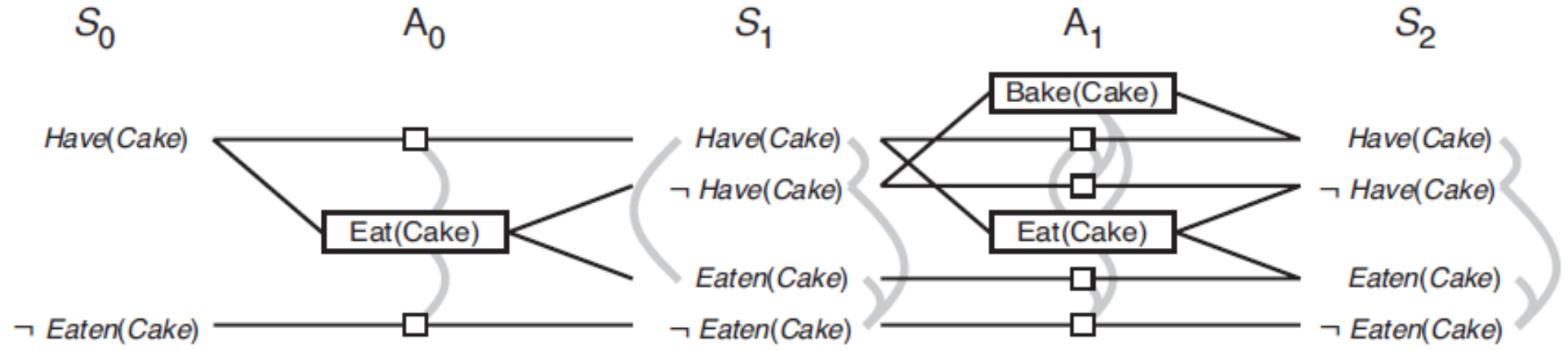
“有蛋糕吃蛋糕”的规划图{10.3}

- “有蛋糕吃蛋糕”的问题

$$
\begin{aligned}
&Init(Have(Cake))\\
&Goal(Have(Cake) \wedge Eaten(Cake))\\
&Action(Eat(Cake)\\
&\quad \text{PRECOND: } Have(Cake)\\
&\quad \text{EFFECT: } \neg Have(Cake) \wedge Eaten(Cake))\\
&Action(Bake(Cake)\\
&\quad \text{PRECOND: } \neg Have(Cake)\\
&\quad \text{EFFECT: } Have(Cake))
\end{aligned}
$$

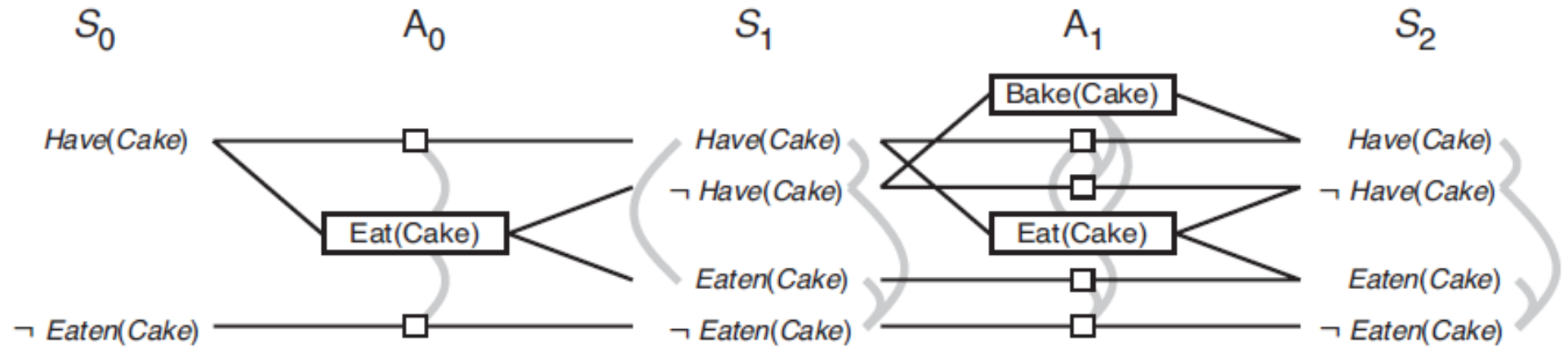
“有蛋糕吃蛋糕”的规划图{10.3}

- 矩形代表动作（小方格代表持续动作），直线代表前提和效果。灰色曲线表示互斥连接。并不是所有互斥连接都画出来了，那样会显得太零乱。一般，如果在S_i两个文字是互斥的，那么在A_i这些文字的持续动作将是互斥的，我们不需要画出互斥连接



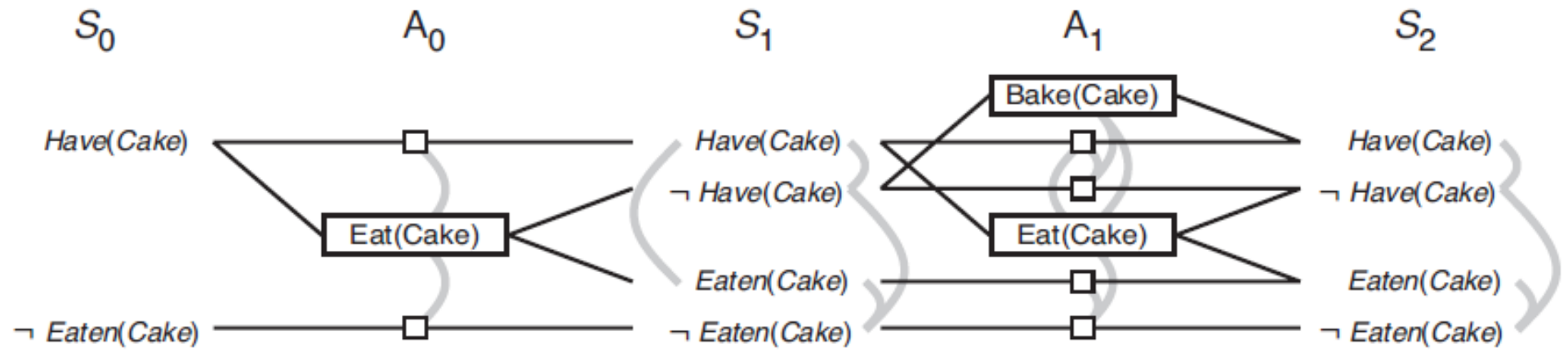
动作层与状态层的交叠何时停止？{10.3}

- 这种方式继续下去，在状态层S_i和动作层A_i之间交叠，直到连续两层是一模一样的为止。这个时候，我们说规划图达到了**稳定**（level off）。图10.8中的状态图在S_2达到稳定。



动作层与状态层的交叠何时停止？{10.3}

- 最后得到一个结构，其中每个A_i层含有S_i中适用的所有动作，以及两个动作不能在同一层都被执行的约束。每个S_i层包含A_{i-1}中动作的可能选择可能导致的所有文字，以及哪些文字对不能成对出现的约束。

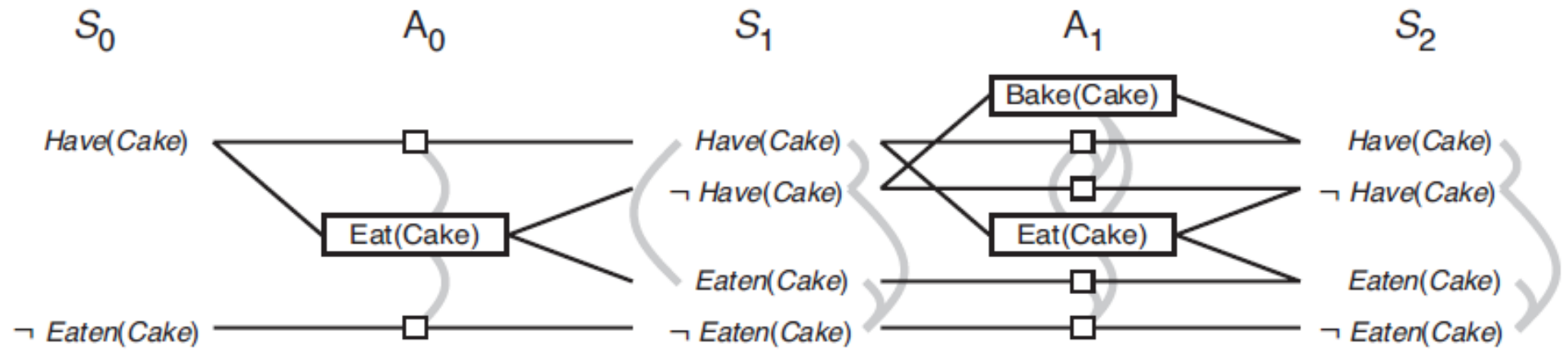


什么情况下两个动作是互斥的？{10.3}

- 在给定层次的两个动作间的互斥关系成立，如果下列三个条件之一成立：
- 不一致效果：一个动作的效果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。
- 冲突：一个动作的效果否定另一个动作的前提。例如，*Eat*(*Cake*)与*Have*(*Cake*)的持续动作冲突，因为*Eat*(*Cake*)否定了它的前提。
- 竞争需要：一个动作的前提之一与另一个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。

什么情况下两个文字是互斥的？{10.3}

- 同一层的两个文字之间的互斥关系成立，如果一个文字是另一个文字的负，或者如果得到这两个文字的每对可能的动作是互斥的。这个条件称为*非一致性支持*（inconsistent support）。例如，*Have*(*Cake*)和*Eaten*(*Cake*)在S_1处是互斥的，因为获得*Have*(*Cake*)的唯一途径——持续动作——与获得*Eaten*(*Cake*)的唯一途径——*Eat*(*Cake*)——是互斥的。



规划图有何用途？{10.3}

- 规划图用于给出更好的启发式估计。
- 用于提取规划。

如何从规划图获得单个目标文字的估计代价？{10.3.1}

- 从状态s获得任何目标文字g_i的代价为g_i在从初始状态s开始构建出的规划图中首次出现的层。我们称这为g_i的**层次代价**（level cost）。

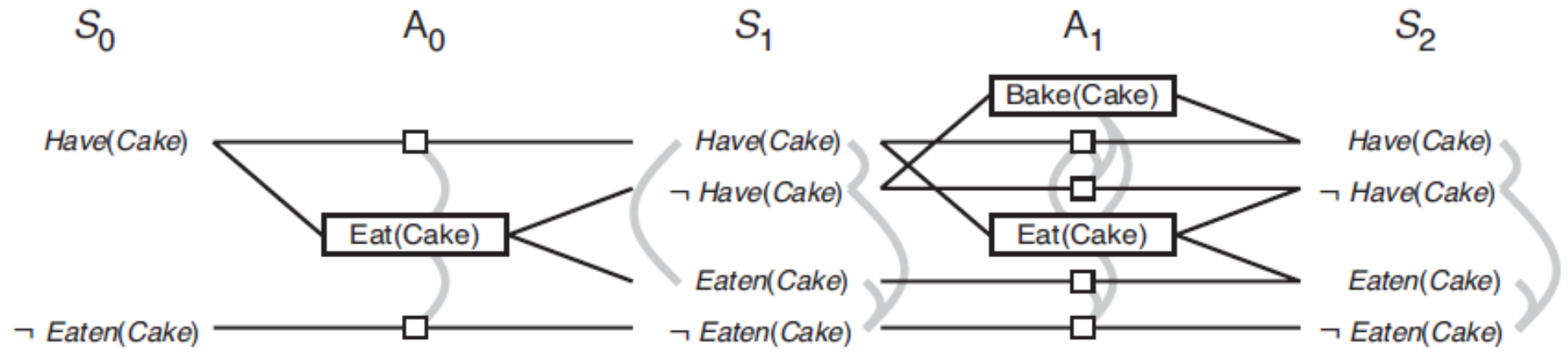
- max?
- sum?

如何从规划图获得合取目标的估计代价？{10.3.1}

- **最大层**（max-level）启发式简单取任何目标的最大层次代价；这个启发式是可采纳的。
- **层次和**（level sum）启发式遵守子目标独立性假设，返回目标的层次代价之和；这个启发式可能不是可采纳的，但在实际中对于可大规模分解的问题效果很好。

如何从规划图获得合取目标的估计代价？{10.3.1}

- 最后，**集合层次**（set-level）启发式找到合取目标中的所有文字出现在规划图中的层次，这一层没有任何一对目标文字是互斥的。对于我们的原始问题，这个启发式会给出正确的值2，对于无*Bake*(*Cake*)的问题，给出的值是无穷大。它是可采纳的，它比最大层启发式占优势（dominate）。当然，它不是完美的；例如，它忽略三个或更多文字间的交互。



如何从规划图提取一个规划:GRAPHPLAN {10.3.2}

- GRAPHPLAN算法反复地用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。

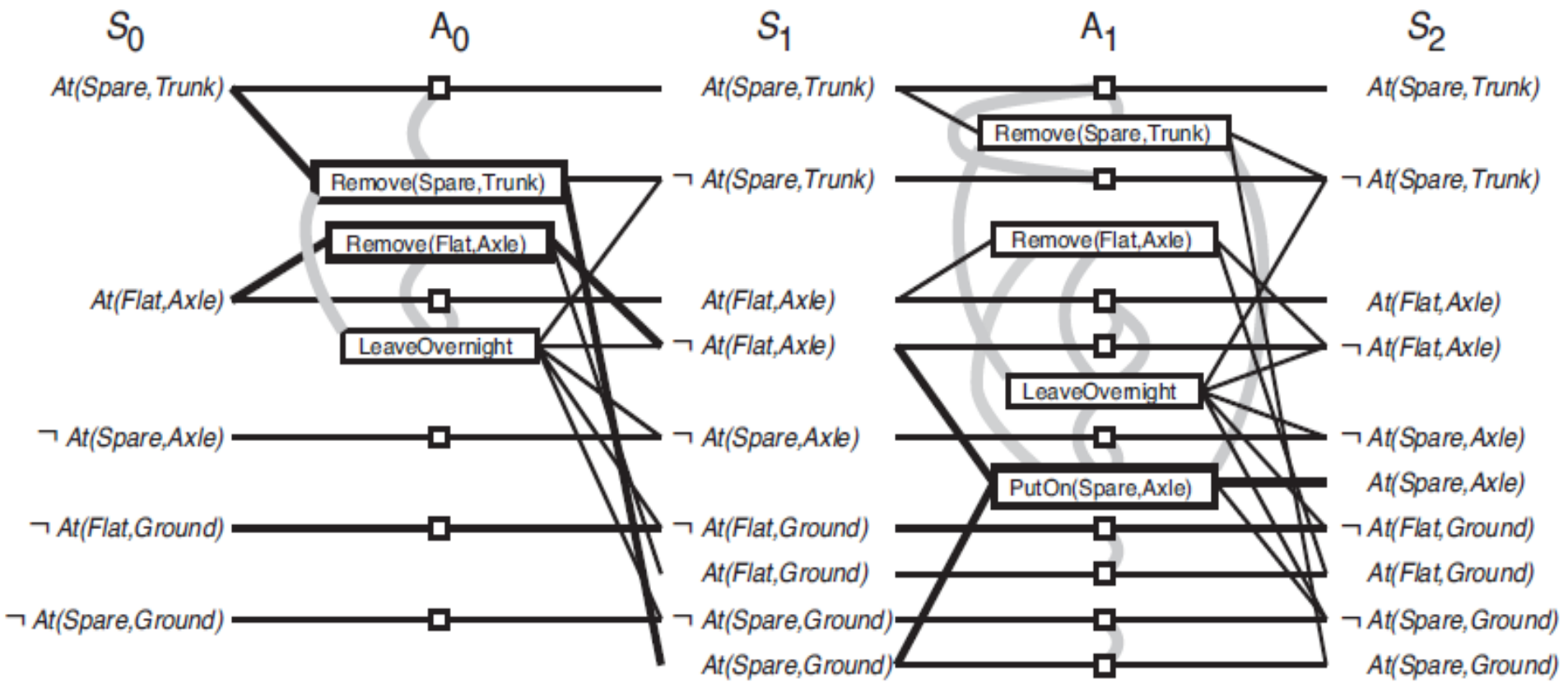
function GRAPHPLAN(*problem*) **returns** solution or failure

graph ← INITIAL-PLANNING-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*)

备用轮胎问题

$Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(obj, loc),$
PRECOND: $At(obj, loc)$
EFFECT: $\neg At(obj, loc) \land At(obj, Ground))$
$Action(PutOn(t, Axle),$
PRECOND: $Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle)$
EFFECT: $\neg At(t, Ground) \land At(t, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
$\land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))$

反复用EXPAND-GRAPH扩展出规划图。备用轮胎问题。{10.3.2}



扩展出规划图后用EXTRACT-SOLUTION找到解{10.3.2}

- 构想为一个约束满足问题
- 或定义为一个后向搜索问题

扩展出规划图后用EXTRACT-SOLUTION找到解{10.3.2}

- 扩展出备用轮胎问题的规划图后，所目标文字都在S_2中，它们都不互斥，意味着一个解可能出现了，EXTRACT-SOLUTION将试图找到这个解。我们可以将EXTRACT-SOLUTION构想为一个布尔约束满足问题（CSP），其中变量是在每一层的动作，每个变量的值在规划之内或规划之外，约束是互斥以及需要满足每个目标和前提。

扩展出规划图后用EXTRACT-SOLUTION找到解{10.3.2}

- 或者，将EXTRACT-SOLUTION定义为一个后向搜索问题，搜索中的每个状态含有指向规划图某一层的指针、以及未被满足的目标集合。

- 这个后向搜索的初始状态、给定状态下的可用动作、转移模型、目标状态是什么？

扩展出规划图后用EXTRACT-SOLUTION找到解{10.3.2}

定义这个后向搜索问题如下：

- **初始状态**是最后一层S_n，以及来自规划问题的目标集合。
- 在S_i层的状态中**可用动作**要选择A_{i-1}中的某个无冲突的、效果覆盖该状态的动作子集。**结果状态**的层为S_{i-1}，并将选择的动作集合的前提作为目标集合。“无冲突”是指一组动作任意两个都不是互斥的，它们的任意两个前提也不是互斥的。
- **目标**是达到一个在S_0的状态，使所有目标得到满足
- 每个动作的**代价**是1。

summary

能否回答以下问题：

- PDDL是如何描述一个规划问题的？
- 描述后向搜索的思想？搜索的启发式有哪些？
- 规划图是怎样的结构？ 如何从规划图得到规划的启发式？ 如何从规划图构建一个约束满足问题？ 如何从规划图定义一个后向搜索问题？