规划(Planning)

- 规划的STRIPS表示语言
- 规划作为搜索

*Slides based on those of Hector Levesque and Sheila McIlraith

动因

- 目前我们学习了问题求解的搜索方法、及知识表示和推理
- 然而，智能体并不仅仅是被动的问题解决者或推理者
- 智能体必须在世界中采取行动
- 我们希望他们能以智能的方式行动
  - 采取有目的的动作,
  - 预测这些动作的效果,
  - 把动作组合起来以实现复杂的目标

火星探测器(MER)的活动规划

- 操作MER 是一项具有挑战性、时间紧迫的任务。
- 每天，操作团队必须制定一个新的规划，描述探测器第二天的活动。
- 这些规划必须遵守资源限制、安全规则和时间限制。
- 自动规划技术用于生成这些规划。

规划应用于自动制造

- 金属板折叠机
- 规划折叠的顺序

规划应用于桥牌

- 桥牌男爵是一个桥牌计算机程序。
- 它赢得了1997年计算机桥牌的世界冠军。
- 它使用AI规划技术来规划其叫牌动作。

其他应用

- 调度
  - 供应链管理
  - 哈勃太空望远镜调度器
  - 工作流管理
- 空中交通管制
  - 规划飞机在跑道和航站楼之间的路线
  - 飞机必须保持安全距离
  - 最小化滑行和等待时间

其他应用

- 角色动画
  - 从高级规范生成逐步的角色行为
- 基于规划的智能用户界面
- Web服务组合
- 基因组重排

规划

- 为了做规划，我们需要对做了一些行动之后的世界状态进行推理。
- 现在我们需要对动态环境进行推理。
  - in(robby,room1), lightOn(room1)为真：在robby执行了turnOffLights 动作后，它们是否为真?
  - in(robby,room1)为真：robby需要做什么动作才能使in(robby,room3)为真?
- 关于动作的效果进行推理，计算采取什么动作可以达到某种效果是决策的核心。

不确定性规划

规划中的一个主要复杂因素是不确定性规划。

- 我们对世界的知识几乎肯定是不完备的。我们可能希望对此进行概率建模。
- 感知会有噪音（尤其是在机器人应用中）。
- 动作和效应器也会出现错误(效果的不确定性)。

经典规划

在这门课中，我们考虑经典规划。

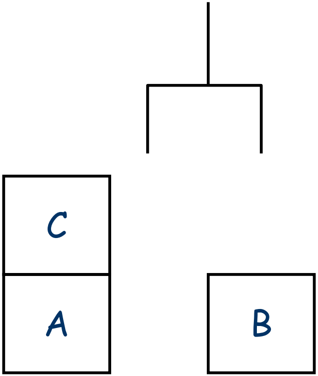
- 没有不完备或不确定的知识。
- 假设关于初始状态的完备信息，通过封闭世界假设(CWA)。
- 假设动作的前提条件是基原子的合取。
- 假设动作效果仅限于使基原子变为真或变为假。没有条件效果，没有析取效果，等等。

封闭世界假设(Closed World Assumption, CWA)

- 用于表示世界状态的知识库是基原子的一个集合。(就像数据库一样)
- 封闭世界假设(CWA)包含以下两点:
  - KB中提到的常量是所有的论域对象。
  - 如果一个基原子不在KB中，那么它为假。
- 这提供了关于世界状态的完备信息。

查询封闭世界知识库

- CWA将知识库视为数据库:
  - e.g., 如果$employed(John, CIBC)$不在数据库中，我们得出结论$\neg employed(John, CIBC)$为真。
- 这样一个KB被称为一个CW-KB(封闭世界知识库)
- 给定一个CW-KB，我们可以计算任意复杂的一阶公式的真值。
  - CW-KB构成一个解释
- 这个过程与数据库中的查询处理非常相似。



KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

Arbitrary queries:

1. clear(c) $\wedge$ clear(b)?
2. $\neg$on(b,c)?
3. on(a,c) $\vee$ on(b,c)?
4. $\exists$X.on(X,c)? (D = {a,b,c})
5. $\forall$X.ontable(X) $\rightarrow$ X = a $\vee$ X = b?

STRIPS表示

- STRIPS(Stanford Research Institute Problem Solver, 斯坦福研究所问题求解器)是由Fikes和Nilsson在1971年开发的一个自动规划程序。
- 这个名字后来被用来指称这个规划器的输入语言。
- 动作被建模为改变世界状态的方式。
- 因为世界状态是用CW-KB表示的，所以STRIPS动作表示更新CW-KB的方式。
- 一个动作产生一个新的KB，描述新的世界状态—执行动作所产生的世界状态。

STRIPS动作

- STRIPS使用3个表表示一个动作。
  - 一个动作前提条件的表
  - 一个动作添加效果的表
  - 一个动作删除效果的表
- 这些表包含变量，因此可以用一个规范来表示一类动作。
- 变量的每个实例化产生一个特定的动作。

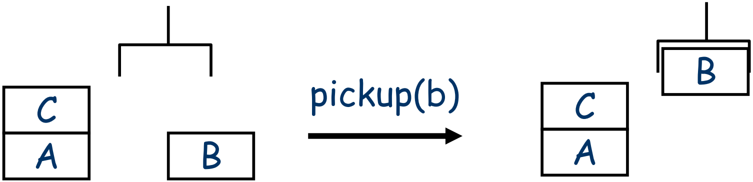
STRIPS动作: 示例

- pickup(X):
  - Pre: {handempty, clear(X), ontable(X)}
  - Adds: {holding(X)}
  - Dels: {handempty, clear(X), ontable(X)}
- “pickup(X)” 被称为一个STRIPS操作.
- 一个实例(e.g. “pickup(a)”)被称为一个动作。

STRIPS动作的操作

- 一个特定的STRIPS动作(基实例)在一个状态(a CW-KB)下是可应用的
    - 如果其前提条件表中的每个事实在KB中为真。
    - 这只需要测试前提条件表中的每个原子事实是否都在KB中。
- 如果该动作是可应用的，则以下列方式生成新的状态
    - 从KB中移除删除表中的所有事实，然后
    - 加入添加表中的所有事实到KB中。

Strips动作的操作：示例



pre = {handmpty,
clear(b),
ontable(b)}

add = {holding(b)}

del = {handmpty,
clear(b),
ontable(b)}

KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

KB = { holding(b),
clear(c),
on(c,a),
ontable(a)}

STRIPS积木世界操作(1)

- pickup(X) (从桌上)
  Pre: {clear(X), ontable(X),handempty}
  Add: {holding(X)}
  Dels: {clear(X), ontable(X),handempty}
- putdown(X) (在桌上)
  Pre: {holding(X)}
  Add: {clear(X), ontable(X), handempty}
  Del: {holding(X)}

STRIPS积木世界操作(2)

- unstack(X,Y) (从积木塔中捡起)
  Pre: {clear(X), on(X,Y), handempty}
  Add: {holding(X), clear(Y)}
  Del: {clear(X), on(X,Y), handempty}
- stack(X,Y) (放在一块积木上)
  Pre: {holding(X),clear(Y)}
  Add: {on(X,Y), handempty, clear(X)}
  Del: {holding(X),clear(Y)}

8数码作为一个规划问题

常数

- 用一个常数表示每个位置: P1,⋯,P9

P1	P2	P3
P4	P5	P6
P7	P8	P9

- 用一个常数表示每个数码和空格: B,T1, ⋯, T8.

8数码

谓词

- adjacent(X,Y): 位置X紧挨着位置Y
  - *e.g.*, adjacent(P5,P2), adjacent(P5,P4), adjacent(P5,P8), . . .
- at(X,Y): 数码X在Y位置
  - *e.g.*, at(T1,P1), at(T2,P2), at(T5,P3), . . .

P1	P2	P3
P4	P5	P6
P7	P8	P9

1	2	5
7	8	
6	4	3

8 数码

A single operator (creating lots of ground actions)

slide(T,X,Y)

Pre: {at(T,X), at(B,Y), adjacent(X,Y)}

Add: {at(B,X), at(T,Y)}

Del: {at(T,X), at(B,Y)}

at(T1,P1), at(T5,P3), at(T8,P5), at(B,P6), …,

at(T1,P1), at(T5,P3), at(B,P5), at(T8,P6), …,

1	2	5
7	8	
6	4	3

→ slide(T8,P5,P6)

1	2	5
7		8
6	4	3

STRIPS没有条件效果

- 和动作描述语言(Action Description Language, ADL)不同，STRIPS没有条件效果。
- *e.g.*, 如果我们允许条件效果，我们可以将putdown(X) 视为stack(X,Y)，其中Y=table，
- 由于STRIPS没有条件效果，对于每种类型的条件，我们必须引入一个额外的动作
- 我们将条件嵌入到前提条件中，然后相应地改变效果。

比STRIPS表达能力更强的语言

- STRIPS操作不是很有表达能力。
- ADL(Action Description Language，动作描述语言)扩展了STRIPS的表达能力。
- ADL操作在STRIPS基础上添加了一些特征。
  - 它们的前提条件可以是任意公式。
  - 它们可以有条件效果和全称效果。

ADL操作举例

积木世界假设:
桌子有无限的空间，所以总是clear的。

- 如果我们在桌(Y=table)上放一块积木，我们不能删除clear(table)，
- 但如果Y是一块普通的积木，我们必须删除clear(Y)。

move(X,Y,Z)
Pre: $on(X, Y) \wedge clear(Z)$
Effs: ADD[on(X,Z)]
DEL[on(X,Y)]
Z $\neq$ table $\rightarrow$ DEL[clear(Z)]
Y $\neq$ table $\rightarrow$ ADD[clear(Y)]

ADL操作举例

move(c,a,b)
Pre: on(c,a) $\wedge$ clear(b)
Effs: ADD[on(c,b)]
DEL[on(c,a)]
b $\neq$ table $\rightarrow$ DEL[clear(b)]
a $\neq$ table $\rightarrow$ ADD[clear(a)]

KB = { clear(c), clear(b),
on(c,a),
on(a,table),
on(b,table)}

KB = { on(c,b)
clear(c), clear(a)
on(a,table),
on(b,table)}

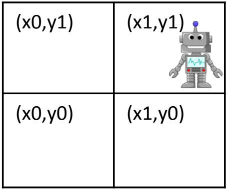
具有全称效果的ADL操作

clearTable()
Pre:
Effs: $\forall X.on(X, table) \rightarrow DEL[on(X, table)]$

KB=$\{on(a, table), on(b, table), on(c, table)\} \Rightarrow$ KB=$\{\}$

Exercise

如图所示，有4个房间。
开始时，机器人在房间$(x1, y1)$。
机器人的目标是访问所有的房间。



我们使用4个常量：$locx0y0, locx0y1, locx1y0, locx1y1$，
3个谓词：

1. $at(loc)$：机器人在房间loc；
2. $visited(loc)$：机器人访问过房间loc；
3. $connected(loc1, loc2)$：房间$loc1$与房间$loc2$相邻。

假设只有一个动作：$move(from, to)$：机器人从房间$from$移动到房间to，前提是这两个房间相邻。

试写出$move$动作的STRIPS表示，初始知识库，目标，及一个解

规划作为搜索问题

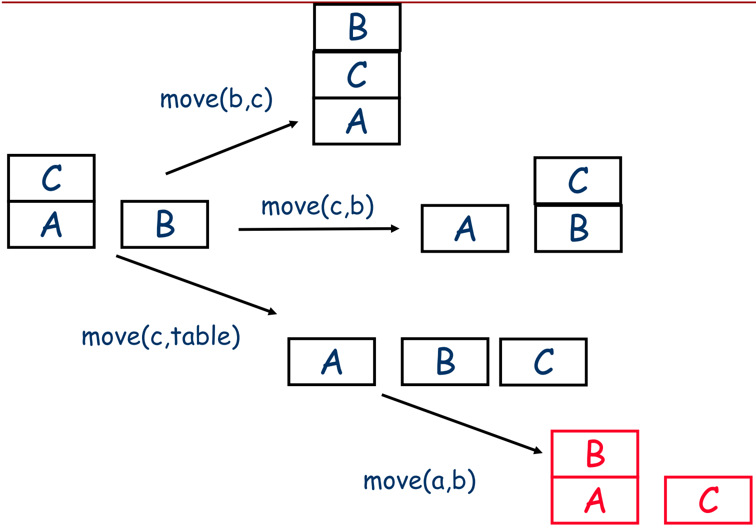
- 给定
  - 一个CW-KB表示初始状态
  - 一组STRIPS或ADL操作
  - 一个目标条件(用公式表示)
- 规划问题是：确定一个动作序列，当应用于初始的CW-KB时，将产生满足目标的更新的CW-KB。
- 这就是所谓的经典规划任务。

规划作为搜索

这是一个搜索问题，其中我们的状态表示是一个CW-KB。

- 初始CW-KB是初始状态。
- 动作是将状态(CW-KB)映射到新状态(更新的CW-KB)的操作。
- 可以检查任意状态(CW-KB)是否满足目标。
- 通常，目标是基原子的合取，因此我们只需要检查是否所有这些原子都包含在CW-KB中。

一个示例



问题和解决方法

- 问题: 搜索树通常相当大
  - 随机地重新放置9块积木需要数千个CPU秒
- 但是这种表示呈现出某种结构。
  - 每个动作只影响一小部分事实
- 规划算法的设计利用了动作变化的“局部性”。
- 我们将学习：使用放松规划启发式函数的启发式搜索。
- 启发式是论域无关的。因此，该技术属于规划的论域无关启发式搜索方法。

STRIPS表示

- 世界状态用CW-KB表示, *i.e.*, 带CWA的基原子列表
  - KB中提到的常量是所有的论域对象
  - 如果一个基原子不在表中，则为假
- 一个动作用3个表表示: Pre, Add 和Del
- ADL对STRIPS的扩展：任意前提条件，条件效果，全称效果

- 给定
  - 一个表示初始状态的CW-KB,
  - 一组STRIPS操作
  - 一个目标条件
- 确定一个动作序列将初始CW-KB转换为一个满足目标的CW-KB

放松问题

- 我们假设
  - 每个动作的前提条件是正事实的集合，以及
  - 目标是正事实的集合。
- 回想一下，我们可以通过解决8数码的放松版本来获得启发式，其中我们放松了下列限制之一：
  - 仅移动到相邻位置
  - 仅移动到空白位置
- 这里的想法是类似的：忽略动作的删除效果表。
- 这产生了一个“放松问题”，可以产生有用的启发式估计。

放松的STRIPS积木世界操作

- pickup(X)
  Pre: {handempty, ontable(X), clear(X)}
  Add: {holding(X)}
  ~~Del: {handempty, ontable(X), clear(X)}~~
- putdown(X)
  Pre: {holding(X)}
  Add: {handempty, ontable(X), clear(X)}
  ~~Del: {holding(X)}~~
- unstack(X,Y)
  Pre: {handempty, clear(X), on(X,Y)}
  Add: {holding(X), clear(Y)}
  ~~Del: {handempty, clear(X), on(X,Y)}~~
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {handempty, clear(X), on(X,Y)}
  ~~Del: {holding(X),clear(Y)}~~

放松问题

定理. 放松问题的最优规划的长度≤原问题的最优规划的长度。

证明:

- 令P是原问题，P'是放松的问题.
- 我们证明$Sols(P) \subseteq Sols(P')$. 证明在附录中
- 因而$Minlen(Sols(P')) \leq Minlen(Sols(P))$.

计算启发式

- 根据定理，最优放松规划的长度可以作为A*的可采纳启发式。
- 然而，计算最优放松规划是NP难的
- 那么我们如何计算启发式呢？
- 从状态S构建一个达到目标的分层结构。
- 计算一个放松的规划需要多少动作。
- 用它作为我们对S到目标距离的启发式估计。

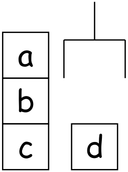
可达性分析

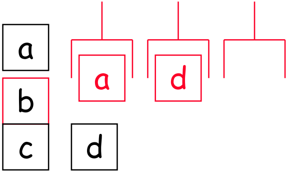
- 开始于初始状态S_0.
- 在状态层和动作层之间交替。
- 确定其前提条件包含在S_0中的所有动作。
- 这些动作构成第一个动作层A_0。
- 下一个状态层$S_1 = S_0 \cup$所有由A_0中的动作添加的事实。
- 继续此过程

可达性分析

- 通常，
  - A_i: 前提条件在S_i中但不在A_{i-1}中的动作集
  - $S_i = S_{i-1}$ ∪ A_i中所有动作的添加表
- 直观地
  - A_i中的动作是可以在某个规划的第i步执行的动作，
  - S_i中的事实是可以通过长度为i的规划实现的事实
- 某些动作/事实具有此属性。但不是全部！
- 继续构造状态和动作层直到
  - 目标G包含在状态层中，或
  - 状态层不再改变（达到不动点）。

示例





on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty

S_0

unstack(a,b)
pickup(d)

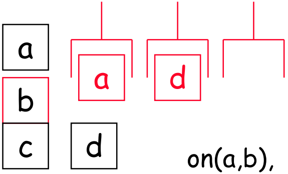
A_0

on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
handempty,
clear(d),
holding(a),
clear(b),
holding(d)

S_1

this is not a state as some of these facts cannot be true at the same time!

示例



on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty,
holding(a),
clear(b),
holding(d)

S_1

unstack(a,b)
pickup(d)

from $\boxed{A_0}$

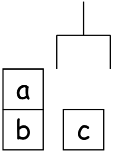
putdown(a),
putdown(d),
stack(a,b),
stack(a,a),
stack(d,a),
stack(d,b),
stack(d,d),
unstack(b,c)
...

Impossible, but we don't know because we ignore dels.

$\boxed{A_1}$

Reachability

to reach on(c,b) requires 4 actions



S_0	A_0	S_1	A_1	
on(a,b), ontable(c), ontable(b), clear(a), clear(c), handempty	**unstack(a,b)** **pickup(c)**	on(a,b), ontable(c), ontable(b), clear(a), clear(c), handempty, holding(a), clear(b), holding(c)	**stack(c,b)** …	on(c,b), …

but stack(c,b) cannot be executed after one step

状态层和动作层的性质

命题. 令$a_0, \ldots, a_{n-1}$是一个在S_0下可应用的动作序列。
令$s_0 = S_0$，并且对$i < n$，$s_{i+1} = s_i \cup add(a_i) - del(a_i)$。
则对$i < n$，存在$j, k \leq i$ s.t. $a_i \in A_k$ and $s_i \subseteq S_j$.

证明在附录中

不可解性

定理. 假设状态层停止更改，并且目标没有得到满足。那么原规划问题是不可解的。

证明:

- 假设$a_0, \ldots, a_{n-1}$是原问题的一个解。
- 则$Goal \subseteq s_n$.
- 根据命题，存在$m \leq n$ s.t. $Goal \subseteq s_n \subseteq S_m$.
- 这与假设相矛盾。

CountActions

- 假设目标G包含在状态层中
- 我们想计算一个好的放松规划
- 其思想是为每个i选择A_i的一个极小子集

CountActions(G,S_K):
/* 这里G包含在S_K中，我们计算实现G的一个放松规划中包含的动作的数量。*/

- 如果$K = 0$ 返回0
- 将G拆分为$G_P = G \cap S_{K-1}$ 和$G_N = G - G_P$
  - G_P包含先前实现的（在S_{K-1}中），
  - G_N包含G刚实现的部分（仅在S_K中）。
- 找到一个添加效果覆盖G_N的极小动作集A。
  - 不能包含冗余动作，
  - 但可能不是最小集合
    (计算最小动作集是集合覆盖问题，是NP难的)
- NewG := G_P U A的前提条件.
- 返回CountAction(NewG,S_{K-1})+size(A)

集合覆盖问题

- 给定一个集合$\{1, 2, ..., n\}$(称为全域)和并集等于全域的m个集合的集合S
- 确定S的一个最小子集，使其并集等于全域.
- *e.g.*, $U = \{1, 2, 3, 4, 5\}$, $S = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$. 一个最小覆盖是$\{\{1, 2, 3\}, \{4, 5\}\}$.

一个示例

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

Goal: f_6,f_5,f_1

Actions:

$[f_1]a_1[f_4]$

$[f_2]a_2[f_5]$

$[f_2,f_4,f_5]a_3[f_6$

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$
$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$
$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$
$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$
$\mathbf{S_2} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$G = \{f_6, f_5, f_1\}$
$G_N = \{f_6\}$ (newly achieved)
$G_p = \{f_5, f_1\}$ (achieved before)

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$\mathbf{G = \{f_6,f_5, f_1\}}$

We split G into G_P and G_N:

CountActs(G,S_2)

$G_P = \{f_5, f_1\}$ //already in S1

$G_N = \{f_6\}$ //New in S2

$A = \{a_3\}$ //adds all in G_N

//the new goal: $G_P \cup Pre(A)$

$G_1 = \{f_5,f_1,f_2,f_4\}$

Return

1 + CountActs(G_1,S_1)

一个示例

Now, we are at level S1

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], [f_2]a_2[f_5]\}$

$\mathbf{S_1} = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$S_2 = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$G_1 = \{f_5,f_1,f_2,f_4\}$

We split G_1 into G_P and G_N:

$\mathbf{G_N} = \{f_5,f_4\}$

$\mathbf{G_P} = \{f_1,f_2\}$

CountActs(G_1,S_1)

$G_P = \{f_1,f_2\}$ **//already in S0**

$G_N = \{f_4,f_5\}$ **//New in S1**

$A = \{a_1,a_2\}$ **//adds all in G_N**

//the new goal: $G_P \cup$ Pre(A)

$G_2 = \{f_1,f_2\}$

Return

2 + CountActs(G_2,S_0)

接下来，我们处于S_0 级，只需返回0

所以，总计CountActs(G,S2)=1+2+0=3

定理. 假设$Goal \subseteq S_k$. 对$i < k$, 令A'_{i-1}为调用CountActions(G_i,S_i)得到的A。
则$A'_0, \ldots, A'_{k-1}$是一个放松规划。

证明在附录中

CountActions

然而，CountActions并不能计算最佳放松规划的长度，因为

- 选择使用哪个动作集来实现G_N（“G的刚实现部分”）并不一定是最优的
- 即使我们在每个阶段都选择了一个最小集合A，我们也可能无法得到整个规划的最小动作集合，因为在每个阶段选择的集合A会影响下一阶段可以使用的集合！

CountActions

计算一个最优的放松规划是NP难的。

- 因此，CountActions不可能在不增加计算难度的情况下改进为可采纳的启发式。
- 从经验上看，CountActions的细化在一些规划论域表现非常好。

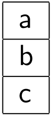
练习：积木世界规划

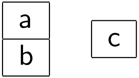
有一组积木：一个积木可以在桌上，也可以在另一个积木的上面
有三个谓词：

- $clear(x)$：积木x的上面没有积木；
- $on(x, y)$：积木x在积木y上；
- $onTable(x)$：积木x在桌上.

有三个动作：

- $move(x, y, z)$：将积木x从积木y上移动到积木z上，如果x在y上，且x和z都是clear的；
- $moveFromTable(x, y)$：将积木x从桌上移动到y上，如果x在桌上，x和y是clear的；
- $moveToTable(x, y)$：将积木x从积木y上移动到桌子上，如果x在y上，并且x是clear 的.

初始状态是：

目标状态是：

一个练习：积木世界规划

1. 写出动作的STRIPS表示、初始知识库和目标。
2. 使用可达性分析来计算初始状态的启发式值。画出状态层和动作层。对于CountActions的每次调用，给出G, G_P, G_N和A的值。

Proof that $Sols(P) \subseteq Sols(P')$

- Let $a_0, \ldots, a_{n-1}$ be a solution to P.
- We show that it is also a solution to P'.
- Let s_0 denote the initial state.
- For $i < n$, let $s_{i+1} = s_i \cup add(a_i) - del(a_i)$.
- Then $Goal \subseteq s_n$, and for $i < n$, $pre(a_i) \subseteq s_i$.
- Let $s'_0 = s_0$, and for $i < n$, let $s'_{i+1} = s'_i \cup add(a_i)$.
- We show by induction on i that for $i \leq n$, $s_i \subseteq s'_i$.
- Thus $Goal \subseteq s_n \subseteq s'_n$, and for $i < n$, $pre(a_i) \subseteq s_i \subseteq s'_i$.
- So $a_0, \ldots, a_{n-1}$ is also a solution to P'.

We show by induction on i that for $i \leq n$, $s_i \subseteq s'_i$.

- Basis: $i = 0$, obviously.
- Induction: Let $i < n$. Assume $s_i \subseteq s'_i$. We prove $s_{i+1} \subseteq s'_{i+1}$.
- Since a_i is applicable in s_i, $pre(a_i) \subseteq s_i$.
- Hence $pre(a_i) \subseteq s_i \subseteq s'_i$. So a_i is applicable in s'_i.
- So $s_{i+1} = s_i \cup add(a_i) - del(a_i) \subseteq s'_i \cup add(a_i) = s'_{i+1}$.

命题. 令$a_0, \ldots, a_{n-1}$是一个在S_0下可应用的动作序列。
令$s_0 = S_0$，并且对$i < n$，$s_{i+1} = s_i \cup add(a_i) - del(a_i)$。
则对$i < n$，存在$j, k \leq i$ s.t. $a_i \in A_k$ and $s_i \subseteq S_j$.

We prove by induction on i:

- Basis: $i = 0$, obviously.
- Induction: Let $i < n$. Assume there exists $j \leq i$ s.t. $s_i \subseteq S_j$.
- Since $pre(a_i) \subseteq s_i$, $pre(a_i) \subseteq S_j$.
- Let k be the least $u \leq j$ s.t. $pre(a_i) \subseteq S_u$.
- Then $a_i \in A_k$.
- So $add(a_i) \subseteq S_{k+1} \subseteq S_{j+1}$.
- Thus $s_{i+1} \subseteq s_i \cup add(a_i) \subseteq S_j \cup S_{j+1} = S_{j+1}$.

定理. 假设$Goal \subseteq S_k$. 对$i < k$, 令A'_{i-1}为调用CountActions(G_i,S_i)得到的A。
则$A'_0, \ldots, A'_{k-1}$是一个放松规划。

Proof: We prove by induction on $i \leq k$ that $A'_0, \ldots, A'_{i-1}$ is a relaxed plan for achieving G_i.

- Basis: $i = 0$. We have $G_0 = G_1 \cap S_0 \cup pre(A'_0)$. Since $Pre(A_0) \subseteq S_0$, $G_0 \subseteq S_0$. So the empty plan achieves G_0.
- Induction: Let $i < k$. Assume $A'_0, \ldots, A'_{i-1}$ achieves G_i.
- We have $G_P = G_{i+1} \cap S_i$, $G_i = G_P \cup pre(A'_i)$, and $G_N = G_{i+1} - G_P \subseteq add(A'_i)$.
- So $pre(A'_i) \subseteq G_i$, $G_{i+1} - G_i \subseteq G_{i+1} - G_P \subseteq add(A'_i)$.
- Thus $A'_0, \ldots, A'_{i-1}, A'_i$ is a relaxed plan for achieving G_{i+1}.