AIP Review

- Week 1: Planning Basic
  - 1.1 Forward Search Planning
    - 1.1.1 Breadth-first search
    - 1.1.3 Depth-first search
  - 1.2 Heristic Search Planning
    - 1.2.1 Best First Search (Greedy)
    - 1.2.2 A*
    - 1.2.3 Weight A*
  - 1.3 Classical(STRIPS) Planning Problem
  - 1.4 Scanalyzer Domain
- Week 2: Improving Heuristic Seach I
  - 2.1 Problem Relaxation and Relaxed Planning(RPG Heuristic)
    - 2.1.1 What is domain independent and domain dependent planning?
    - 2.1.2 Why we hope Domain Independent Heurist?
    - 2.1.3 Delete Relaxation:
    - 2.1.4 Why we need Relaxed Planning rather than Original Planning?
    - 2.1.5 Building a Relaxed Planning Graph (RPG)
    - 2.1.6 Extract a Solution from RPG
    - 2.1.7 FF Planner
      - 2.1.7.1 Enforced Hill-climbing(EHC)
  - 2.2 Planning Landmarks
    - 2.2.1 Fact Landmark
    - 2.2.2 Action Landmark
    - 2.2.3 Disjunction Action Landmark
    - 2.2.4 Finding Landmarks #1: Deletion Relaxation
    - 2.2.5 Finding Landmarks #2: Backchaining
    - 2.2.6 Finding Landmarks #3: RPG Propogation
    - 2.2.7 Landmarks Orderings
      - 2.2.7.1 Sound Ordering
      - 2.2.7.2 Unsound Ordering (used in heuristics)
  - 2.3 Landmark Uses
    - 2.3.1 Landmarks as Subgoals
    - 2.3.2 Landmarks as Heuristics Estimate
      - 2.3.2.1 The Landmark-count Heuristic (LM-Count)
      - 2.3.2.2 LAMA
- Week 3: Improving Heuristic Seach II & Optimal Planning I
  - 3.1 Dual Open Lists Search
    - 3.1.1 Identidem
  - 3.2 hAdd & hMax heuristic
- Week 4: Optimal Planning II & Non-Forward Search I
  - 4.1 SAS+ Planning
    - 4.1.1 Invariants
    - 4.1.2 Mutal Exclusion
    - 4.1.3 Finite-Domain State Variables
    - 4.1.4 Finite-Domain Representation（FDR)
    - 4.1.5 SAS+
    - 4.1.6 Domain Transition Graphs (DTGs)
  - 4.2 Pattern Databases
      - 4.2.1 Abstracts

4.2.2 How to Use Pattern Database
4.3 Cost Partitioning
Week 5: Non-Forward Search II & Planning Under Uncertainty
5.1 Graph Plan
5.1.1 Valid Plan
5.1.2 Mutex relations
5.1.2.1 Interference
5.1.2.2 Competing Needs
5.1.2.3 Inconsistent Support
5.1.3 Backwards Planning graph
5.1.4 RPG vs Graph Plan
5.2 Planning as SAT
5.2.1 Frame Problem and Frame Axioms
5.2.2 Exclusion Axioms
5.2.3 Example
5.3 POP
5.4 HTN Planning
5.4.1 STN
5.4.1.1 Total Ordering STN
5.4.1.2 Patial Ordering STN
5.4.2 HTN
5.4.3 STN Planning VS HTN Planning
Week 6: Planning with Preference and Numeric Planning
6.1 Numeric Planning
6.2 Metric-FF
6.3 Numeric Planning

AIP Review

- Classical Planning: The fundamentals
- Improving Search
  - Heuristic Design (RPG/LAMA)
  - Dual Openlist Search
- Optimal Planning
  - SAS+ Planning
  - Pattern Databases
- Non-forward Search
  - Graph Plan
  - SAT Planning
  - POP Planning
  - HTN Planning
- Planning Under Uncertainty
  - Markov Decision Process (MDP)
  - Partially Obsearvable Markov Decision Process (POMDP)

Week 1: Planning Basic

1.1 Forward Search Planning

Forward search from the initial state, with a closd list, builds a tree.

Closed list: states already dealt with

Open list: states to deal with

1.1.1 Breadth-first search

FIFO QUEUE: push states onto the back, pop states from the font

1.1.3 Depth-first search

STATCK: push states onto the back, pop states from the back

1.2 Heristic Search Planning

1. What is heristic?

   A function that takes a state and returns an estimate of its distance from the goal. Perfect heuristics would tell us which sucessor to follow and which path should explore first.

2. How can you tell if a heuristic is **admissible**?

   A heuristic function is said to be admissible if it **never overestimates** the cost of reaching the goal.

   ```
   f(n) = g(n) + h(n), 其中h(n):estimaye of current to the goal; g(n):cost of path from init to current
   h(n) <= h*(n),      其中h*(n)表示到goal state的真正距离real cost
   ```

   e.g: Cost(A -> C) = 1, Cost(C -> Goal) = 3, Cost(A -> Goal) = 4

   if h is admissible, then: $h(A) \leq 4; h(C) \leq 3; h(G) \leq 0$

3. How can you tell if a heuristic is **consistant**?

   e.g: Cost(A -> C) = h(c), Cost(C -> Goal) = h(g), Cost(A -> Goal) = h(a)

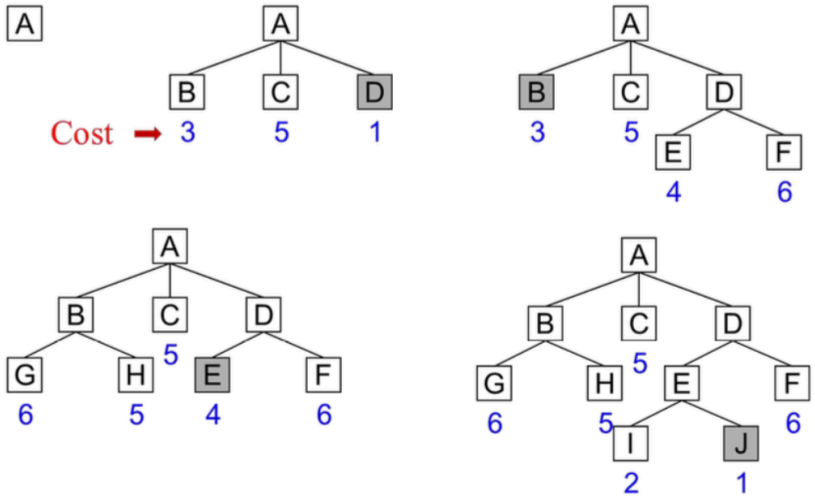
   if h is admissible, then: $h(a) \leq h(c) + h(g)$

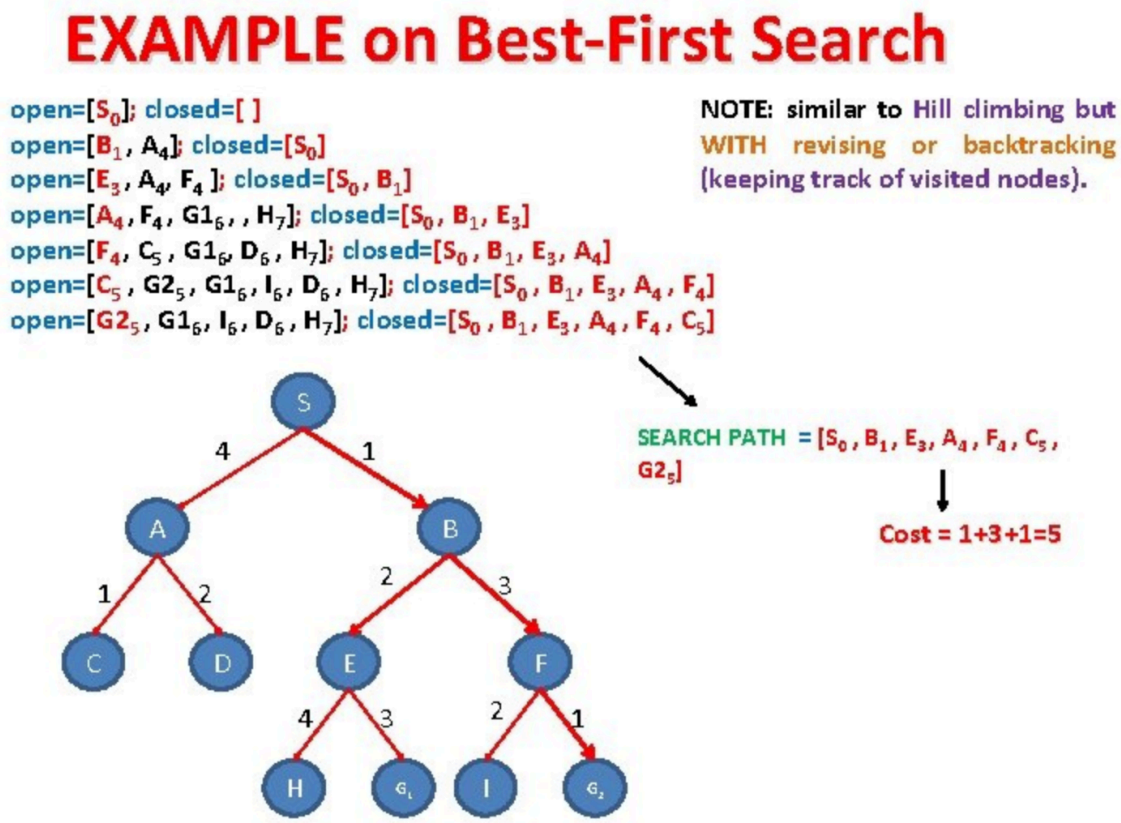
4. How to implement Heuristic Search?

Using Prioritise open nodes with heuristic

1.2.1 Best First Search (Greedy)

Expand node with minimum h: Priority Queue **sorted** by $h(n)$





1.2.2 A*

Expand node with minimum (g+h): Priority Queue **sorted** by $g(n) + h(n)$

1.2.3 Weight A*

Expand node with minimum (g+Wh): Priority Queue **sorted** by $g(n) + Wh(n)$

1.3 Classical(STRIPS) Planning Problem

Classical(STRIPS) planning problem is a tuple<P,A,I,G> where:

- P: propositions
- A: Actions
    - Preconditions

    - Add Effects
    - Delect Effects
- I: Initial States
- G: Goal Statess

1.4 Scanalyzer Domain

TODO

Week 2: Improving Heuristic Seach I

2.1 Problem Relaxation and Relaxed Planning(RPG Heuristic)

2.1.1 What is domain independent and domain dependent planning?

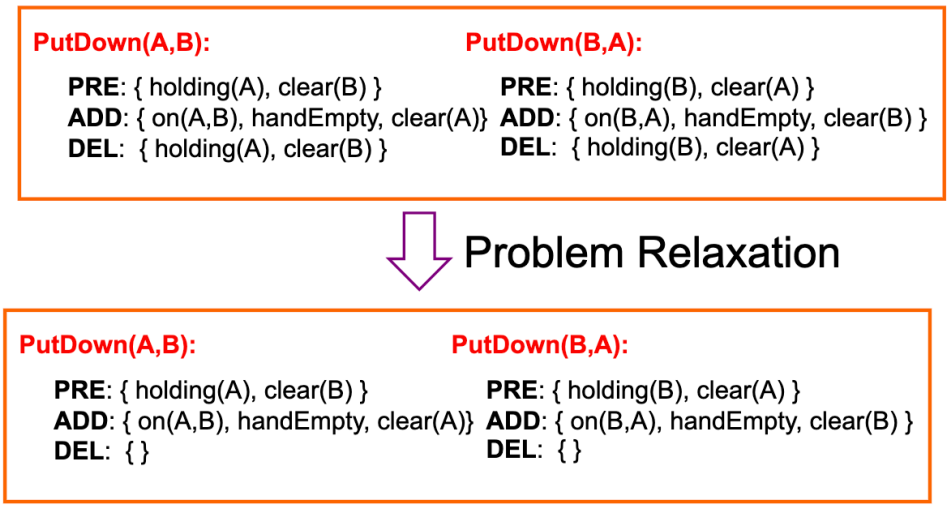
- Domain Independent: Planning system that addresses problems without specific knowledge of the domain.
- Domain Dependent: Using domain-specific knowledge

2.1.2 Why we hope Domain Independent Heurist?

Because Domain Independent Heurist are formulated on a general problem description language (PDDL, STRIPS...), which means that they can be applied to every problem that can be expressed in that language.

2.1.3 Delete Relaxation:

删掉所有动作的删除效果，**使Original Planning变成Relaxed Planning**



holding(A)表示A被hand拿起，clear(B)表示B表面没有东西。执行动作PutDown(A,B)后的效果分成ADD和DELETE,在Relaxation后只需保留ADD效果即可。

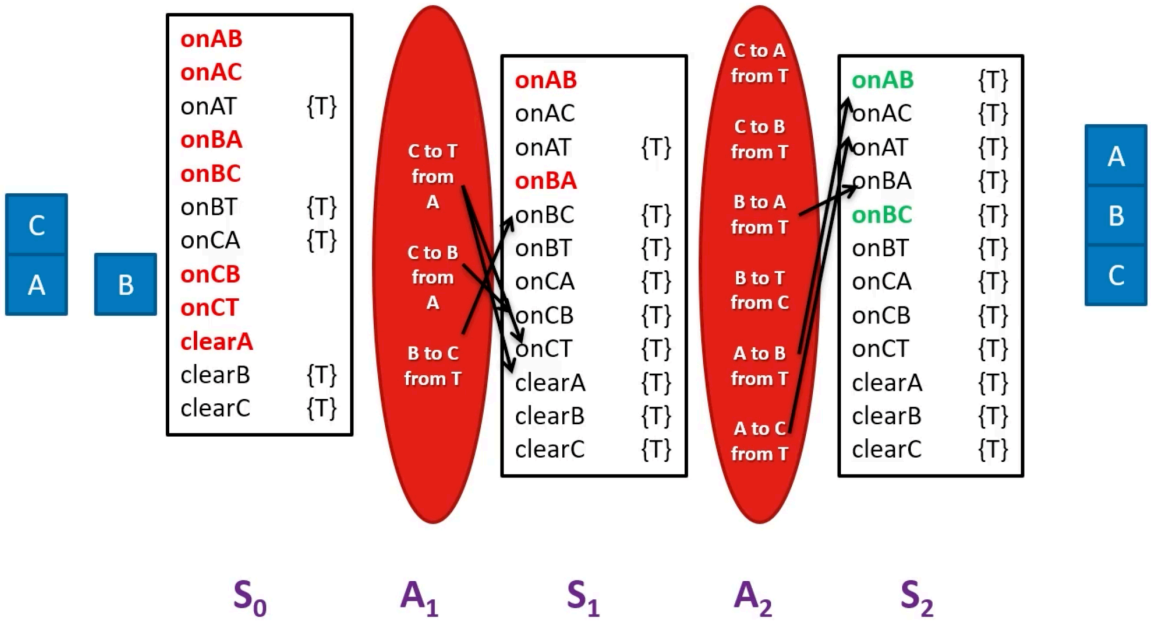
2.1.4 Why we need Relaxed Planning rather than Original Planning?

因为在放松的规划图中，因为忽略了动作的删除效果，任何一对命题节点之间不存在互斥关系，任何一对动作节点之间也不存在互斥关系。

$h(S_{current} = cost(\#action)$: Take relaxed-plan length to be the heristic value.

The length of optimal solution for the relaxed problem is admissible heuristic for original problem. WHY?

2.1.5 Building a Relaxed Planning Graph (RPG)



图解：

- S_0, S_1, S_2表示Fact Layer，其中包含当前所有的状态; **注意：已经忽略掉了delete effects，因此是Relaxed Planning Graph**
- A_1, A_2表示Action Layer，其中包含当前所有可执行的动作
- Fact Layer中{T}表示True，该状态为TRUE，其他未标注红体字为False

2.1.6 Extract a Solution from RPG

At each fact layer f(n), we have goals to achieve g(n) : 在实现目标之后，开始倒推，goal的action的precondition加入前面一层的goal，这样就可以知道每一层的goal，这样就是一种智能搜索。

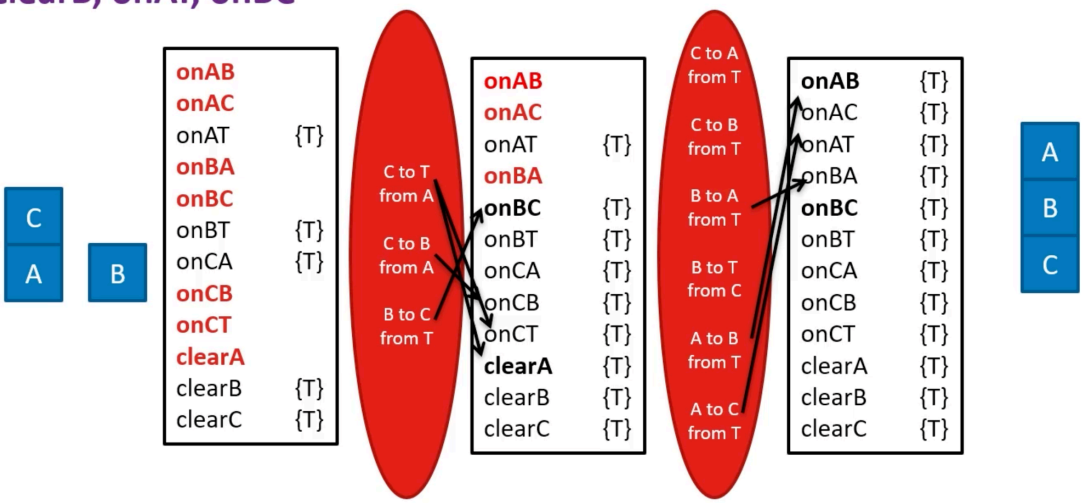
流程：

We start with g(n) containin the problem goals:

- For each fact in g(n):
  - if it was in f(n-1), add it to g(n-1)
  - Otherwise, choose an action from a(n), and add its preconditions to g(n-1)
- Stop when at g(0)

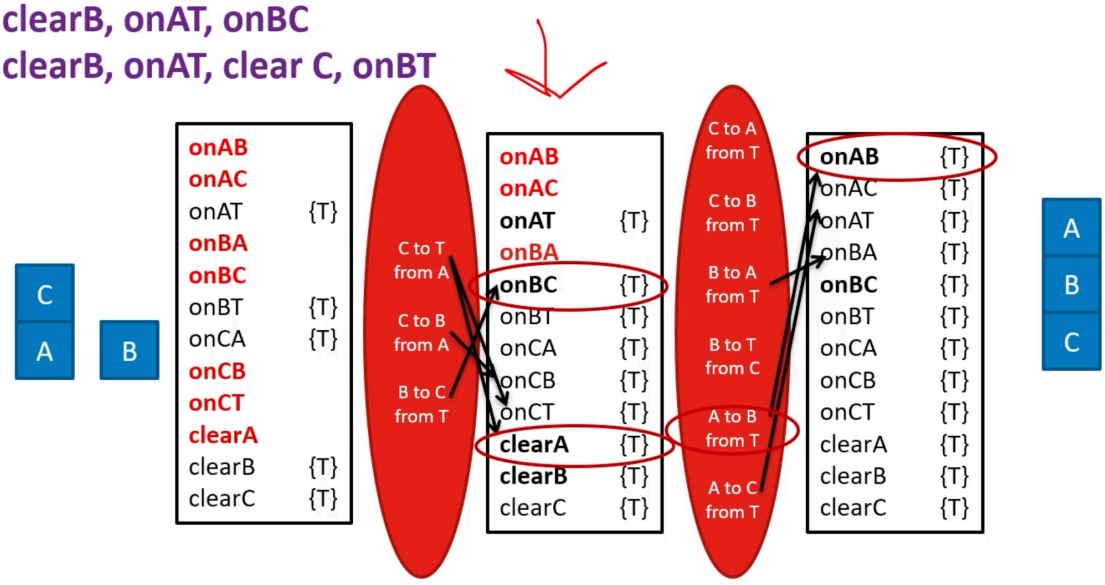
例子：

- **G(2) = onAB, onBC**
- **G(1) = clearA, clearB, onAT, onBC**
- **Solution:**
  - A to B from T



- 从最后一层g(2)开始, g(2)包含最终的goal：onAB, onBC; 左上角可见*G(2) = onAB, onBC*
- 由于onAB存在于上一层的goal(g(1))中, 所以直接加到g(1)中即可；onBC没有存在于上一层的goal(g(1))中,所以找到造成onBC的动作（A to B from T）的preconditions(clearA,clearB,onAT)加入到g(1)中，左上角可见*G(1) = clearA,clearB,onAT + onBC*
- 由于A to B from T这个动作造成了onBC这个状态，并且onBC是我们需要的最终结果，因此A to B from T是最终的答案之一
- 再看接下来需要实现的g(1)层goal：

- **G(1) = clearA, clearB, onAT, onBC**
- **G(0) = clearA, clearB, onAT, clear C, onBT**
- **Solution:**
  - C to T from A
  - B to C from T
  - A to B from T



- 在g(1)中的四个目标clearA,clearB,onAT, onBC中，由于onAT和clearB已经存在于上一层goal(g(0))中，所以就直接加入到g(0)中即可
- 只有clearA和onBC不在上一层goal(g(0))中，找到对应的动作：'C to T from A' 和 'B to C from T' 和与之对应的preconditions: ClearC, ClearA和 onBT, 因此*g(0) = onAT, clearB, ClearC, ClearA, onBT*
- 动作'C to T from A' 和 'B to C from T' 加入到Solution中

- Final relaxed solution：
  - Put C on T from A;
  - Put B to C from T;

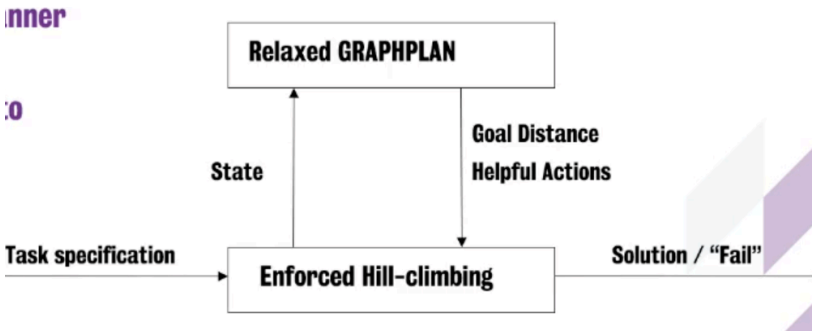
- Put A to B from T

2.1.7 FF Planner

FF is a forward-chaining heuristic search-based planner

FF use the Relaxed Planning Graph(RPG) heuristic to guide search

系统概述:



采用Enforced Hill-climbing(EHC)作为主要的搜索算法，使用放松图规划作为主要的启发式信息产生模块。

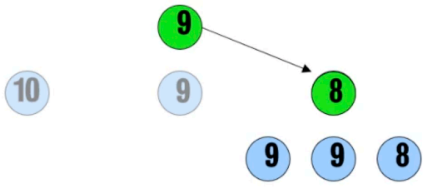
加强爬山算法不断产生状态，并将这些状态发送给放松图规划模块以获取启发信息。

放松图规划模块以加强爬山模块提供的失态，为初始状态求解一个到目标状态的放松规划，并把求得的放松规划的长度作为状态s到目 标状态的距离估计反馈给加强爬山模块，同时反馈的还有对于状态。到达目标状态有帮助的动作。

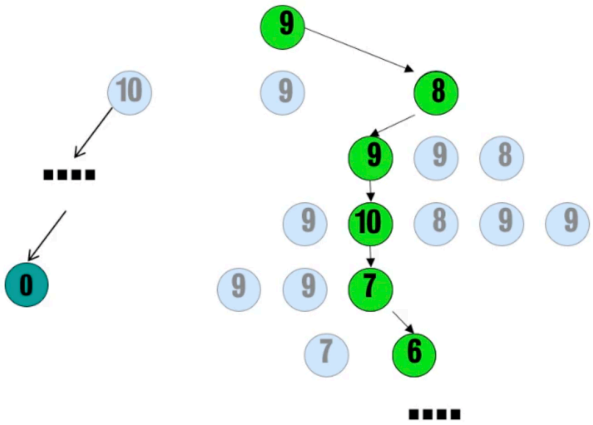
2.1.7.1 Enforced Hill-climbing(EHC)

- Basic idea: **lower heuristic = better**
- Always try and find states with the best heuristic value seen so far:
  - Start with best = h(S;nit), aim to get to best = 0.
    1. Expand a state S
    2. If we have a successor state S' with h(S') « best: S=S' return to Step 1;
    3. If no state is found **Breadth-first search** until one is found; return to Step 1;

  举例:



此时，没有比8更小的值进行下一步，则通过Breadth-first来寻找比8小的值:



但是，如果最小的值在另外一侧，EHC将没有答案。

When EHC fails, FF resorts to systematic best-first search from the initial state!!!

```
- EHC is incomplete, but fast
- FF is complete, but slower
```

2.2 Planning Landmarks

The key feature of a landmark is that it must be true at some point on any solution path to the given planning task

- Understanding and exploiting constraints that encapsulate facets of every possible solution of a problem.
- Some of these constraints can be denoted as landmarks of the solution.
- Landmarks can be ordered to dictate the order in which they should be achieved.
- We aim to discover landmarks and their ordering automatically from the problem.

2.2.1 Fact Landmark

A variable takes a particular value in at least one state

在fact layers中每次都满足的fact就是fact landmark

2.2.2 Action Landmark

An action must be applied in the solution

An action set A is a landmark if all plans include an action from A.

2.2.3 Disjunction Action Landmark

One of a set of possible actions must be applied

2.2.4 Finding Landmarks #1: Deletion Relaxation

- Remove an action to see if RPG is still success
- Need to try every action

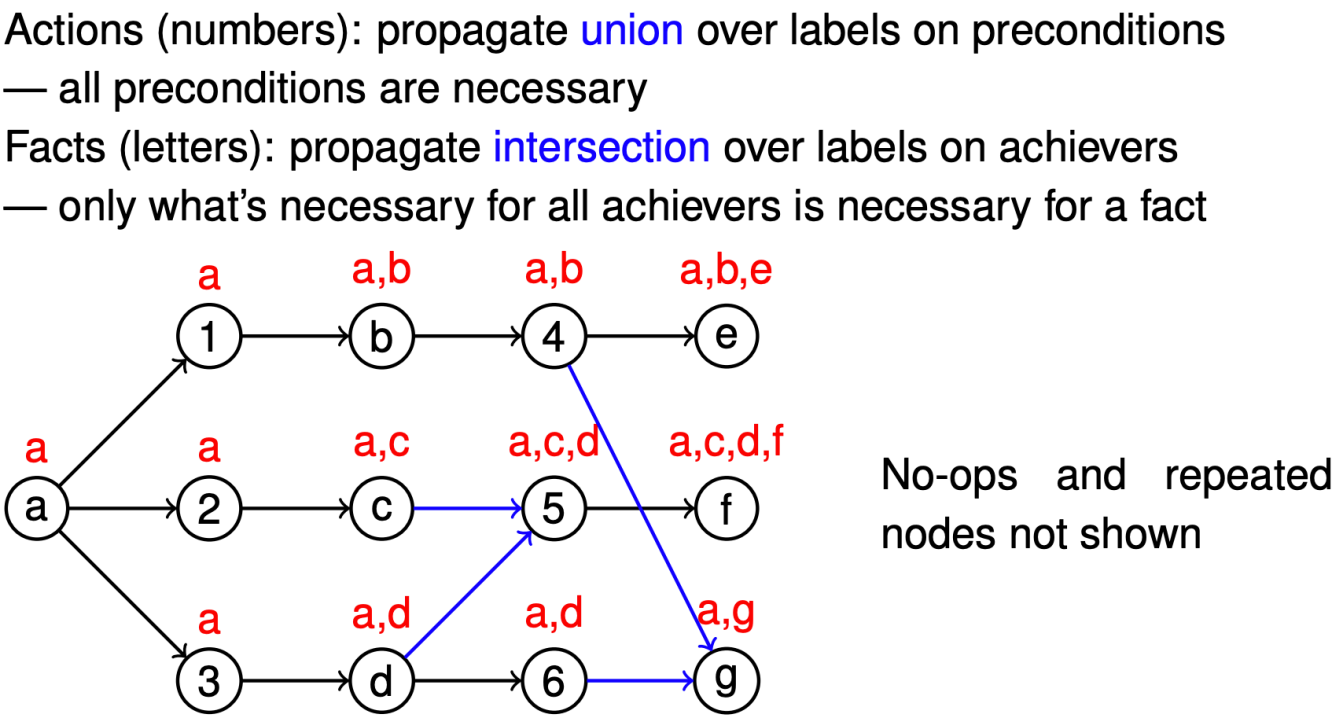
2.2.5 Finding Landmarks #2: Backchaining

- Treat each **goal** is a landmark
- If goal B is a landmark and all actions that achieve B share A as preconditions, then
  - A is a landmark
  - A is also ordered before (must be achieved before) the goal B
- Repeat for all landmarks found

2.2.6 Finding Landmarks #3: RPG Propogation

Find landmarks through forward propagation in relaxed planning graph (Zhu & Givan 2003)

- Finds causal landmarks only (preconditions for actions)
- Finds all causal delete-relaxation landmarks in polynomial time
  - Propagate information on necessary predecessors
  - Label each fact node with itself
  - Propagate labels along arcs



- 字母表示fact; 数字表示action
- 已知initial state中的fact (a)是landmark，往后传播，每层的action造成的effect和preconditions也是landmark，到最后的final goal中就包含了所有的landmarks
- Goal nodes in final layer: labels are landmarks; 比如，如果g是final goal的话，landmarks就是a

2.2.7 Landmarks Orderings

LO告诉我们要先达到哪个landmark状态才能到达goal

2.2.7.1 Sound Ordering

保证肯定能到达goal的ordering

- Necessary Ordering $A \rightarrow_n B$: 不实现A就没法实现B
- Greedy-necessary Ordering $A \rightarrow_{gn} B$: 实现A在第一次实现B之前
- Natural Ordering $A \rightarrow B$:

2.2.7.2 Unsound Ordering (used in heuristics)

不能保证可以到达goal的ordering

- Reasonable Ordering $A \rightarrow_r B$： 先走B的话会再走一遍B才能到A
- Obedient Reasonable Ordering$A \rightarrow_{or} B$

举例：

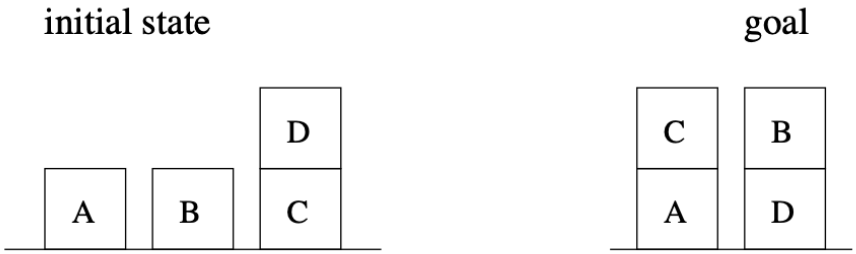


Figure 1: Example *Blocksworld* task.

- Necessary Ordering:
  - $clear(C) \rightarrow_n holding(C)$ 是necessarily ordered的关系, 因为达到holdingC这个状态的**上一步**必须且总是clearC
  - $holding(C) \rightarrow_n on(CA)$ 是necessarily ordered的关系， 因为on(C A)的**上一步**必须且总是holdingC
- Greedy necessary Ordering:
  - $clear(D) \rightarrow_{gn} clear(C)$, but not a necessary order: 实现clearD在第一次实现clearC之前
- Reasonable Ordering：
  - $on(B, C) \rightarrow_r on(A, B)$: if one achieves on(A, B) first then one has to unstack A again in order to achieve on(B, C).

2.3 Landmark Uses

We assume that landmarks and orderings are discovered in a pre-processing phase, and the same landmark graph is used throughout the planning phase

2.3.1 Landmarks as Subgoals

- Given landmark set and partial ordering
- Set the $\vee$ of first landmark as goals to plan, then repeat

Conclusions:

- Fast, but may take longer plans
- Incomplete: may lead to dead-ends

2.3.2 Landmarks as Heuristics Estimate

2.3.2.1 The Landmark-count Heuristic (LM-Count)

LM-Count: **A path-dependent heuristic**

路标计数启发式(landmark-counting heuristic)是**从当前状态到达目标还需要实现的landmark个数**, $l = n - m + k$ 作为状态估值,其中,n 是给定问题提取出的路标总数,m 为当前已经到达的landmark个数,k 为已到达但还需要再次实现的landmark个数.

公式解析：$L(s,p) = (L \backslash Accept(s,p)) \vee ReqAgain(s,p)$

- $L(s,p)$: 从状态 s沿着路径p还需要实现landmarks个数
- L：问题中发现的所有landmarks
- $Accept(s,p)$: 以后不需要被实现节点就是被accpet的节点
  - A为true并且A之前的节点被accepted (初始状态都为accept的)
  - 一个accepted的landmark在以后的状态s中会继续保持accepted的状态
- $ReqAgain(s,p)$: 已到达但还需要再次实现的landmark
  - A is false in s and is a goal （一开始所有landmark都为false，所以goal都是ReqAgain）
  - A is false in s and $A \rightarrow_{gn} B$, and B is not accepted: false and is a greedy-necessary predecessor of some other landmark m, which is not yet accepted.
- 所以LM-count就是那些还没有achieve的landmarks，因为从所有的Landmarks中去除掉不需要被achieve节点再加上那些目前还需要achieve的landmark，所以总体就是还没有achieve的landmarks

LM-Count是**inadmissible heuristic**: 因为一个action会achieve不止一个landmark

2.3.2.2 LAMA

LAMA是以路标计数启发式(landmark-counting heuristic)为启发函数设计出的规划系统

LAMA is a propositional planning system based on heuristic search. The core feature of LAMA is the use of landmarks as a pseudo-heuristic and for generating preferred operators.

Week 3: Improving Heuristic Seach II & Optimal Planning I

3.1 Dual Open Lists Search

If one heuristic is on a plateau, use the second heuristic to find the path

3.1.1 Identidem

针对EHC中不一定找到最优结果问题使用local-seach方法解决

- 当到达plateau无法找到更合适的节点时
  - 任意选择一个子节点（甚至比当前节点更差的节点也可以）
  - 如果d steps之后还没找到更好的节点，再从新选择一个节点，继续寻找
  - 如果当前d steps内没有找到更好的节点，就往更深的d+1 steps中继续寻找
- Identidem select successor states using **roulette selection**
- EHC selects successor states using **the lowest heuristic value**

解决了掉入plateau和local minima无法找到最优路径问题

3.2 hAdd & hMax heuristic

- h^{Add}: find the lowest cost path for all the preconditions of the goal. then pick the **highest value** among them as the value of the goal (Inadmissible)
- h^{Max}: find the lowest cost path for all the preconditions of the goal, but **add** their values as the value of the goal (Admissible)

Week 4: Optimal Planning II & Non-Forward Search I

4.1 SAS+ Planning

4.1.1 Invariants

A truck cannot be in two places at once/ you cannot at uni and at home at once

$\phi = not(at - uni \wedge at - home)$

4.1.2 Mutal Exclusion

How to find invariants? through Mutal Exclusion!!

$\phi = not(at - uni \wedge at - home)$ Is a mutex

4.1.3 Finite-Domain State Variables

举例：$v = below - a, dom(below - a) = b, c, table$

表示哪些block可以在a的下面，domain：below c可能是b,c,table 所以一个式子可以代表三个状态 below-a:{b,c,table}

4.1.4 Finite-Domain Representation（FDR）

我们可以将propositions通过FDR相互转化，比如

$above - a = nothing \vee not(below - b = c)$

等价于$A - clear \vee not(B - on - C)$

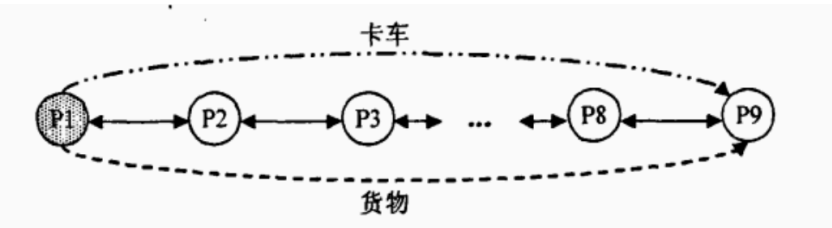
4.1.5 SAS+

我们使用FDR将SAT问题(布尔可满足性问题)转化为SAS+ planning问题

In SAS+, $< V, O, s_0, s^* >$

- V：状态变量集，对于V中的每一个状态v都有一个值域domain, D_v
- O: a set of Operators(Actions) 每个操作o=<pre,eff>具有前提pre和效果eff
  - Prevail conditions: variables won't change
  - Pre-post conditions: variables that change
- s_0 : 初始状态, 是对V中所有状态的赋值
- s^*: 目标状态

举例：



节点表示不同的城市，灰色的节点表示卡车初始所在地，弧表示两个不同城市之间的公路，两种虚线弧分别指向货物和卡车从 初始地所要到达的目标地，其中灰色的地点表示货物初始所在地。该问题中共有9个不同的城市，每相邻的两个城市间有双向的公路，有一辆卡车，初始位置在p1，货物的初始位置也为P1，目标是货物和卡车都在P9。

- 状态变量集：V={v_c，v_t}，其中v_c表示货物所在地，值域为(P1,P2, P3, P4, P5. P6,P7,P8,P9,t)，其中$v_c = t$表示货物在卡车里; v_t表示卡车所在地，值域为{P1, P2, P3, P4, P5, P6, P7,P8, P9)
- 操作集：O= {
  $< \{v_t = P1\}, \{v_t = P2\} >, < \{v_t = P2, v_c = P2\}, v_c = t >, < \{v_t = P8, v_c = t\}, \{v_c = P8\} >, \ldots$
  }没有列出O中的所有操作。而是对每种操作各举一例， 这三个操作分别表示卡车从PI到P2的move操作，在P2地装载货物操作 load, 以及在P8地卸载货物操作 unload。其余操作类似。
- 初始状态：s_0: $\{v_c = P1, v_t = P1\}$
- 目标状态：$s^* = \{v_c = P9, v_t = P9\}$

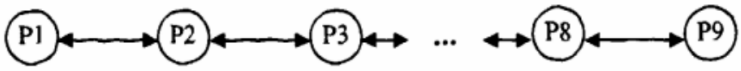
4.1.6 Domain Transition Graphs (DTGs)

域转换图表达了**同一状态变量**的不同取值之间的转换关系，需要对每个状态变量进行构建

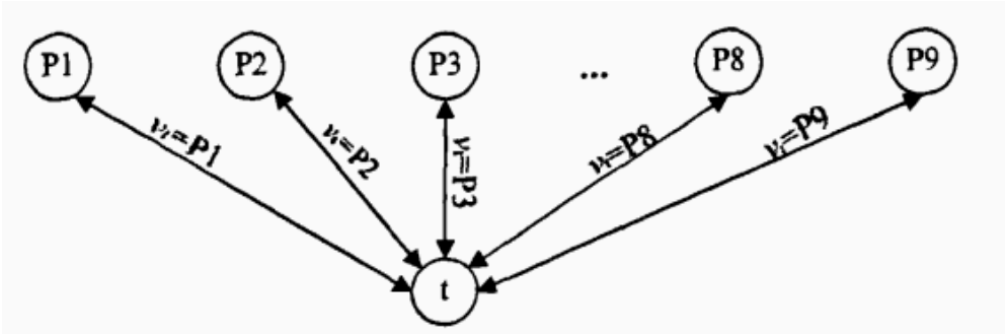
因此DTG可以表示一个变量是否可以从其值域中的一个值转换为另一个值，以及转移是否以其他变量的值为条件

因此使用DTG可以帮助我们visualize assignment of variables to values during search

上图例子中，变量v_t的DTG如下图所示，表示卡车在九个城市之间的移动情况。图中的转移都没有条件，因为卡车的移动不依赖于其它变量（v_c）。



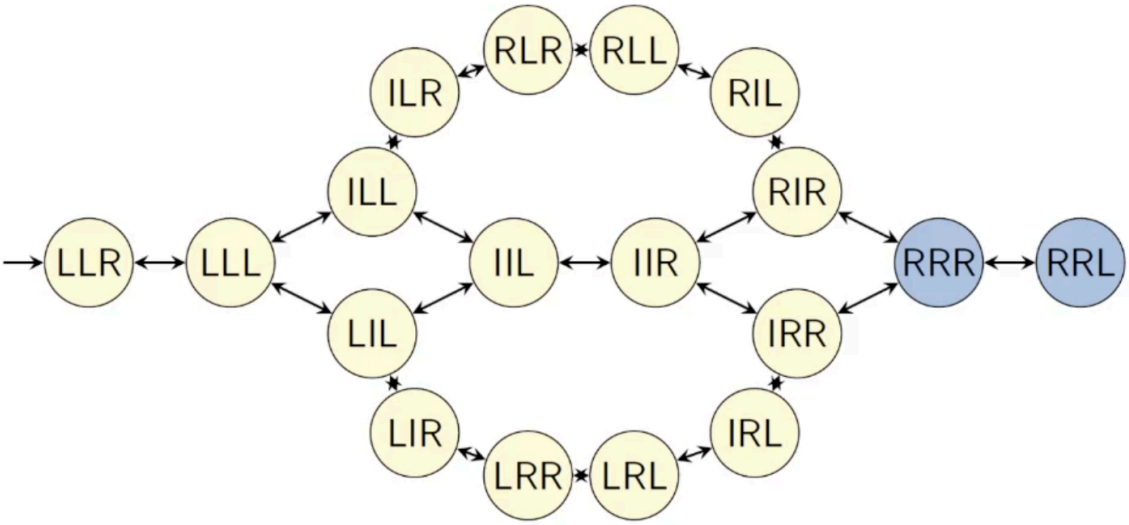
变量v_c的DTG如下图所示，表示货物所有可能的转移。如货物从卡车中（$v_c = t$）转移到P1($v_c = P1$)需要以卡车在P1($v_t = P1$)为条件，相对应的操作是unload：反之，从$v_c = P1$到$v_c = t$的转移与操作load相对应，其它转移情况类似，显然，货物的转移以卡车的转移为条件。



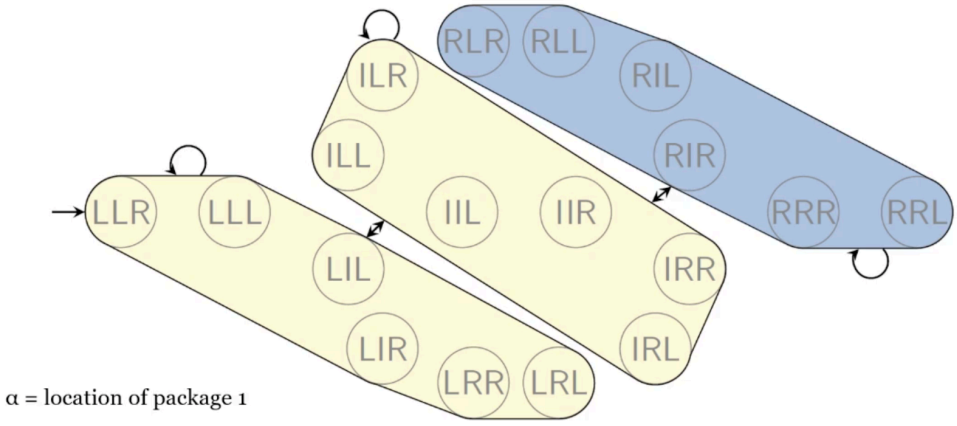
4.2 Pattern Databases

- Pattern Database是一种抽象的启发式，所谓**抽象的启发式**是指启发式的值是抽象后的规划任务的最优耗散值。
- 这里的抽象是将原始问题的问题空间抽象为较小的空间，但该空间仍能反应出原问题的实质，通过抽象得到的空间为原问题提供heuristics value，帮助原问题寻找最优解
- 通过使抽象空间足够小从而使得在抽象后的问题空间利用盲目搜索获取最优耗散解成为可行。在抽象后的状态空间中,每个状态与目标的距离即状态到达目标所需的最优耗散是被提前计算好并存储在内存中的,因此在搜索过程中启发式的值可通过简单的查表来获取。

4.2.1 Abstracts



三个字母，前两个表示package在哪，最后一个字母表示truck在哪，目标是把俩package送到R



如果我们只关心第一个package在哪，就把问题转换到了子问题空间。可以分成三个类别，package在L，在truck上，在R。

4.2.2 How to Use Pattern Database

- Create a set of all state propositions in mutex groups
- Construct abstract problem spaces
- Construct Pattern Database
- Run planning process from initial state

Example:

- **Initial State**

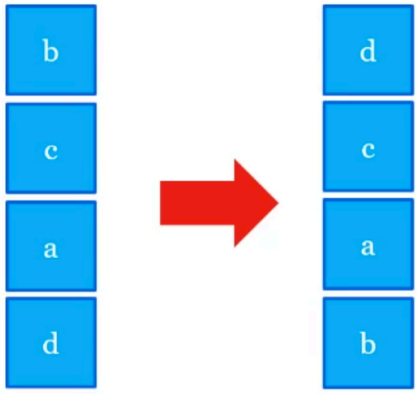
(clear b)
(ontable d)
(on b c)
(on c a)
(on a d)

(pick-up a)

P = (clear a), (ontable a), (handempty);
A = (holding a); and
D = (ontable a), (clear a), (handempty)

- **Goal State**

(on d c)
(on c a)
(on a b)



操作集O：pick-up, put-down, stack, and unstack；每个operator is defined by a precondition list P , an add list A, and a delete list D。比如: (pick-up a)

```
P = {(clear a); (ontable a); (handempty)}
A = {(holding a)}
D = {(ontable a), (clear a); (handempty)}
```

1. **Create a set of all state propositions in mutex groups：**

```
G1 = {(on c a); (on d a); (on b a); (clear a); (holding a)}
G2 = {(on a c); (on d c); (on b c); (clear c); (holding c)}
G3 = {(on a d); (on c d); (on b d); (clear d); (holding d)}
G4 = {(on a b); (on c b); (on d b); (clear b); (holding b)}
G5 = {(ontable a); true}
G6 = {(ontable c); true}
G7 = {(ontable d); true}
G8 = {(ontable b); true}
G9 = {(handempty); true}
```

对于一个block而言，它上方有其他block，它上方是空的，和它被正被抓起这三种情况是互斥的，即它只能处于一种情况中。因此，有些谓词实例不可能在一个状态中同时出现。比如(on B A) (clear A)和(holding A)很明显是互斥的，在同一状态中只会有一个出现。

如果我们找出互斥的谓词实例并把它们放在一个集合里，如 (on B A), (on CA), (on DA), (clear A),(holding A)这样的一个谓词实例集所需要的编码位数，应该是log5=3而不是5位，节省了两位，对于物体越多的问题来说，能节省的编码位数越多。

G1-G4是对on+clear+holding实例化的结果；G5-G8：是对ontable实例化的结果；G9为handempty。G5 至G9 都含有一个额外的 true，用于表示状态中不含 true 所在集合中的其余谓词实例：

- G1: what is above A?
- G2: what is above C?
- G3 : what is above D?

- G4: what is above B?
- G5: what is under A? either (ontable a) or true: A is either on the table, or it is not, and that pair of options can be encoded with a single variable with two values in its domain.

2. **Construct abstract problem spaces**

   构造abstract就要通过忽略一些variable的取值，把多种不同的state抽象成为同一个state，那忽略哪些variable的取值呢？最简单的办法是分奇偶，强行分出。

   因此把goal state分成奇偶两部分：Π_{odd} = (on a b), (on d c)和Π_{even} = (on c a)

3. **Construct Pattern Database**

优点：

1. 一旦模式数据库建立好，要得到启发值只需要对表格进行一次简单的查找即可
2. PDB一旦建立好了就能用于相同的domain里，目标一致初始值不一致的问题中

4.3 Cost Partitioning

Week 5: Non-Forward Search II & Planning Under Uncertainty

5.1 Graph Plan

图规划

5.1.1 Valid Plan

规划问题的一个中心任务是找出有效规划。这里所说的有效规划，是指：

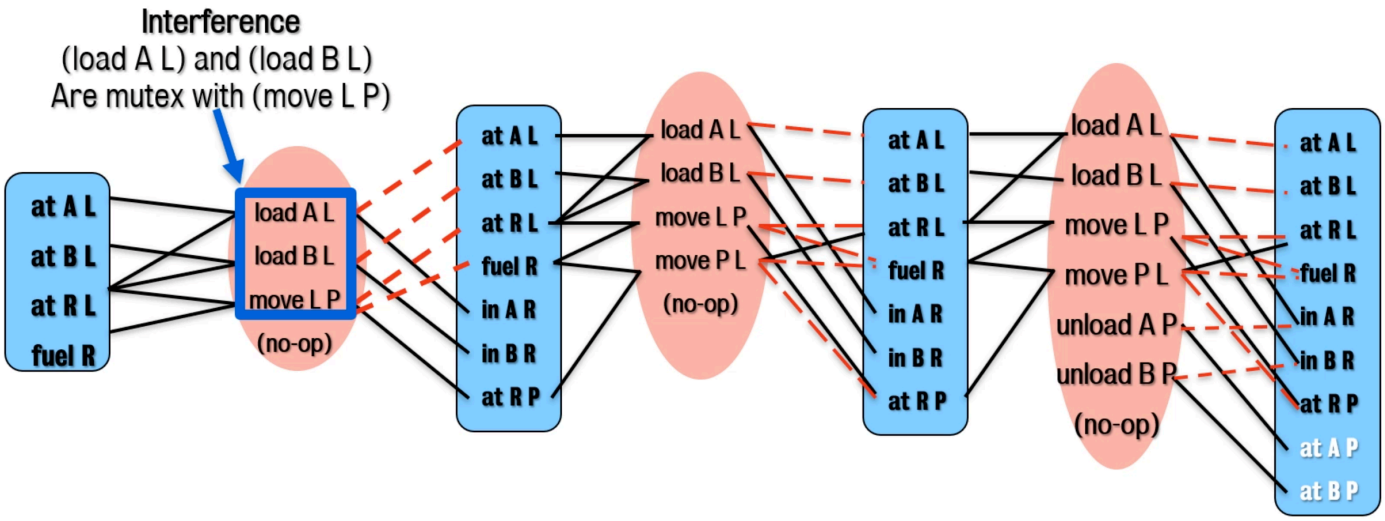
- Actions at the same level don't interfere with one other
- Each action's preconditions are true at that point in the plan
- Goals are satisfied at the end of the graph

5.1.2 Mutex relations

如果没有valid plan包含action 1和action 2，那他俩是mutex的

5.1.2.1 Interference

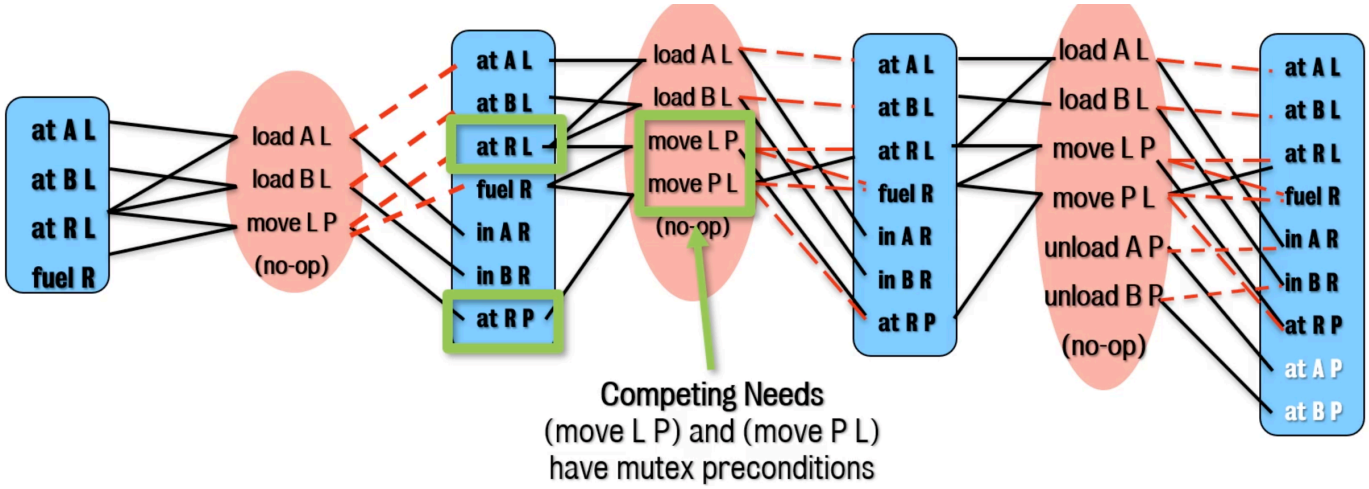
两个动作 effect negates precondition



{Load A L, Load B L}与{Move L P}互斥：因为{Move L P}删除了{Load A L, Load B L}的前提条件{At R L} (红色虚线表示删除effects)

5.1.2.2 Competing Needs

两个动作 preconditions are mutex



{Move L P}与{Move P L}互斥：因为{Move L P}的前提条件{at RL}与{at R L}的前提条件{at RP}互斥，Rocket不可能同时在R和L

5.1.2.3 Inconsistent Support

all ways of creating them are mutex

5.1.3 Backwards Planning graph

Backwards Planning graph to generate the final plan

- 查看规划图最后一步是否包含目标集G = {at A P, at B P}
- 找出生成各目标的动作集
  - at A P : {unload A P}
  - at B P : {unload B P}
  - 查看这两个动作是否互斥，如果不互斥这两个actions的前提条件构成了又一目标集G1={in A R, at R P, in B R}
  - 按照步骤依次往下搜索

- load A L, load B L
- move L P
- unload A P, unload B P

5.1.4 RPG vs Graph Plan

Graph Plan is solving the actual planning problem, and does so by ensuring the facts and actions do not have any contradictions. Meanwhile, the RPG is using the same approach but is only interested in solving the relaxed problem. So it doesn't care if the goal facts are technically not achievable at that point. In fact, solving the relaxed version of this problem in the slide results in several actions being selected in g(n-1). It would have to move the rocket and unload A and B in the same action layer, which has an ordering problem the relaxed plan ignores

5.2 Planning as SAT

步骤：规划问题形式化为SAT问题，对SAT问题进行简化并求解，然后进行计划抽取

5.2.1 Frame Problem and Frame Axioms

解释性框架公理：所谓框架问题，是指动作只改变在其效果中出现的“流”，而对那些不在其效果中 出现的“流”则保持不变。例如，移动一个集装箱，只会引起被移动的集装箱的位置的改变，而其他集装箱的位置则保持不变。 对于框架问题，也可以等价地认为，如果某一“流’被改变，则定存在某一以该“流”为效果的动作被执行了。例如，如果“流”f在时间步i不成立,而在时问步i+1成立,则一定存在一在时间步i执行的动作，该动作的正效果中含有f。类似地，如果“流’f在时间步i成立,而在时间步i+1不成立,则一定存在一时间步执行的动作，该动作的负效果中含有f。由此,可以得到表示由于动作而引起状态变化的命题公式集合,这些命题公式集合枚举了引起状态变化的所有可能的动作集合，因此,称这些命题公式集合为解释框架公理。

5.2.2 Exclusion Axioms

完全排斥公理：某一时间步只能有一个动作发生

5.2.3 Example

如何将规划问题形式化为SAT问题？？

假定有一个机器人搬运车r_1, 两个位置l_1和l_2。r_1 能在这两个位置之间移动。在初始状态，r_1位于位置l_1，而在目标状态, r_1位于位置l_2。 移动r_1的动作可以表示成下面形式： $move(r, l, l')$ precond: at (r, l) effects: at (r, l'), ¬ at (r, l) 显然,对于上述规划问题，长度为1的规划就已足够实现规划目标，因此,可将规划的长度限定为 n=1。可将初始状态和目标状态分别编码成命题公式 (init) 和 (goal)：

- (init): $at(r_1, l_1, 0) \wedge \neg at(r_1, l_2, 0)$
- (goa): $at(r_1, l_2, 1) \wedge \neg at(r_1, l_1, 1)$

- 动作可被编码成：

(Move1): $move(r_1, l_1, l_2, 0) => at(r_1, l_1, 0) \wedge at(r_1, l_2, 1) \wedge \neg at(r_1, l_1, 1)$

(Move2): $move(r_1, l_2, l_1, 0) => at(r_1, l_2, 0) \wedge at(r_1, l_1, 1) \wedge \neg at(r_1, l_2, 1)$

- 解释性框架公理可被编码为：

(at1): $\neg at(r_1, l_1, 0) \wedge at(r_1, l_1, 1) => move(r_1, l_2, l_1, 0)$

(At2): $\neg at(r_1, l_2, 0) \wedge at(r_1, l_2, 1) => move(r_1, l_1, l_2, 0)$

(At3): $at(r_1, l_1, 0) \wedge \neg at(r_1, l_1, 1) => move(r_1, l_1, l_2, 0)$

(At4): $at(r_1, l_2, 0) \wedge \neg at(r_1, l_2, 1) => move(r_1, l_2, l_1, 0)$

- 完全排斥公理：

$\neg move(r_1, l_1, l_2, 0) \vee \neg move(r_1, l_2, l_1, 0)$

将规划问题按照这五部分形式化为SAT问题，得到该命题公式的一个模型，从该命题公式的模型中，可将那些被赋值为真的表示规划动作的命题变量排列成序列$< a_1(0), \ldots, a_n(n-1) >$的形式，其中$a_i(j)$表示在时间步j的动作$a_i$所对应的命题变 量,0≤i≤n-1。这样，就可以得到规划解$< a_1, \ldots, a_n >$

将规划问题编码成命题公式后,规划问题是否有规划解的问题就转换为命题公式是否可满足的问题，也就是说,规划的求解就转换为寻找命题公式的一个模型。有关如何在命题公式中寻找其模型的问题，命题可满足领域中的研究者已经对此进行了广泛的研究，并有很多成熟快速的 SAT solver求解器出现。事实上，将规划问题转换成 SAT 问题来求解的研究动机也主要是源手 SAT研究领域的快速发展。

5.3 POP

5.4 HTN Planning

分层任务规划HTN：Hierarchical Task Network

在 HTN 规划器中，规划的目的不是要达到某一目标状态集合,而是要完成某一任务集合。也就是说，与 STRIPS 方法和 PDDL 方法不同的是,HTN 语言不是直接告诉规划器规划目标,而是将规划目标转换成任务集合的形式。同时,规划系统的输入不仅包含与经典规划器类似的动作集合,还包含一方法集合。所谓方法就是以处方(prescription)的形式告诉系统如何将 某一类任务分解成更小的子任务集合。**规划的过程就是递归地将那此非原子任务分解成越来越小的子任务，直到出现那些可 以直接执行规划动作就能完成的原子任务为止**。同时,HTN 规划器也是以方法的形式执行动作。

两种tasks

- Primitive Tasks: Single actions
- Compound Tasks: collection of one or more tasks

以Dockerworker为例：

```
Recursive-Move(p,q,c,x)
  Task: move - stack(p,q)
  Pre: top(c,p), on(c,x)
  Subtasks:
      move-topmost-container(p,q)
      move-stack(p,q)
```

Task(m)表示m方法可用于解决哪一类任务

Subtasks(m)表示完成task(m)所需完成的子任务

5.4.1 STN

STN(Simple Task Network)简单任务网络规划是简化版的HTN

5.4.1.1 Total Ordering STN

全序STN规划：全序不允许对不同任务的子任务进行交叉

5.4.1.2 Patial Ordering STN

偏序STN规划：子任务之间没有严格的序列关系并且子任务可以任意交叉

5.4.2 HTN

在STN中有一个顺序约束Ordering constraint，通过任务网络图中的边来表示,而方法的前提条件则没有在任务网络图中被显示地 表示出来：取而代之的是,我们通过构造能满足方法前提条件的任务网络来实现方法的前提条件。或许只有像TFD(全序前向分解)或 PFD这样的前向分解程序才适合与STN一起使用,但有些情况下我们并不希望采用这种前向分解程序。而HTN 规划在如何构造任务网络这方面比 STN 规划更加自由，因此HTN规划是对 STN 规划的推广。

为了能更自由的构造网络，需要把那些尚未满足的约束显式的表示在任务网络中

5.4.3 STN Planning VS HTN Planning

Week 6: Planning with Preference and Numeric Planning

PDDL2.1 introduced numerical planning and temporal planning

6.1 Numeric Planning

6.1.1 名词解释：

- Fluents: variables with changing values
- Propositional fluents: varaibles with Boolean vaule
- Numeric fluents: fluents with numeric values

6.1. 2 定义：

- Numeric fluent definition

  ```
  (:functions (functionName ?parameter1 - type ...) ... )
  ```

- Numeric Preconditions

  ```
  <, >, <=, >=, =, +, -, *, /.
  ```

  e.g:

  ```
  (>= (fuel) (* 5 (distance ?from ?to)))
  (<= (height truck) (height barrier1))
  (>= (number-of-widgets) 3)
  ```

- Effects can modifying numeric fluent using

  ```
  increase, decrease, assign, scale-up, scale-down
  ```

  e.g:

```
(decrease (fuel) (* 5 (distance ?from ?to)))
(increase (height truck) (height barrier1))
(assign (battery-charge b) (max-charge b))
```

- Metric function(达到goal state的函数)

  ```
  (:metric minimize (...))
  ```

  e.g：

  ```
  (:metric minimize total-time)
  ```

  Or artimetic expression over the numerical varaible

  ```
  (:metric minimize (+(*2 total-time)(* 4 total-fuel-consumed)))
  ```

- Durative action

  ```
  (at start ...) preconditions and changes when action is initiated
  (at end ...) preconditions and effects when action is terminated
  (over all ...) invariance condition to be true throughout the action
  ```

PDDL2.1使用numeric fluents例子：

```
(:action load
 :parameters (?t - truck ?p - package ?loc - location)
 :precondition (and (at ?t ?loc) (at ?p ?loc)
                    (<= (volume ?p) (empty_capacity ?t)))    # package的体积要小于等于空车时可载物的容
量
 :effect (and (not (at ?t ?loc) (at ?p ?loc))
              (decrease (empty_capacity ?t) (volume ?p))
              (increase (weight ?t) (weight ?p)))

假设我们已经定义了以下volume, empty_capacity和weight函数：
(:functions (volume ?p - package) (empty_capacity ?t - truck) (weight ?o - location))
```

6.2 Metric-FF

Metric-FF is a domain independent planning system developed by Joerg Hoffmann.

6.2.1 Relaxation Heuristic

- Good search heuristics often use relaxations
  - simplification of the problem
  - easier to solve
  - approximates the original problem
- In planning a good relaxation is to ignore negative action effects
  - FF (Hoffmann) implements a very effective example of this

- This idea can be extended to ignoring decreasing numeric effects
    - MetricFF (Hoffmann)

6.2.2 Relaxed graph construction (Reachability analysis)

例子：

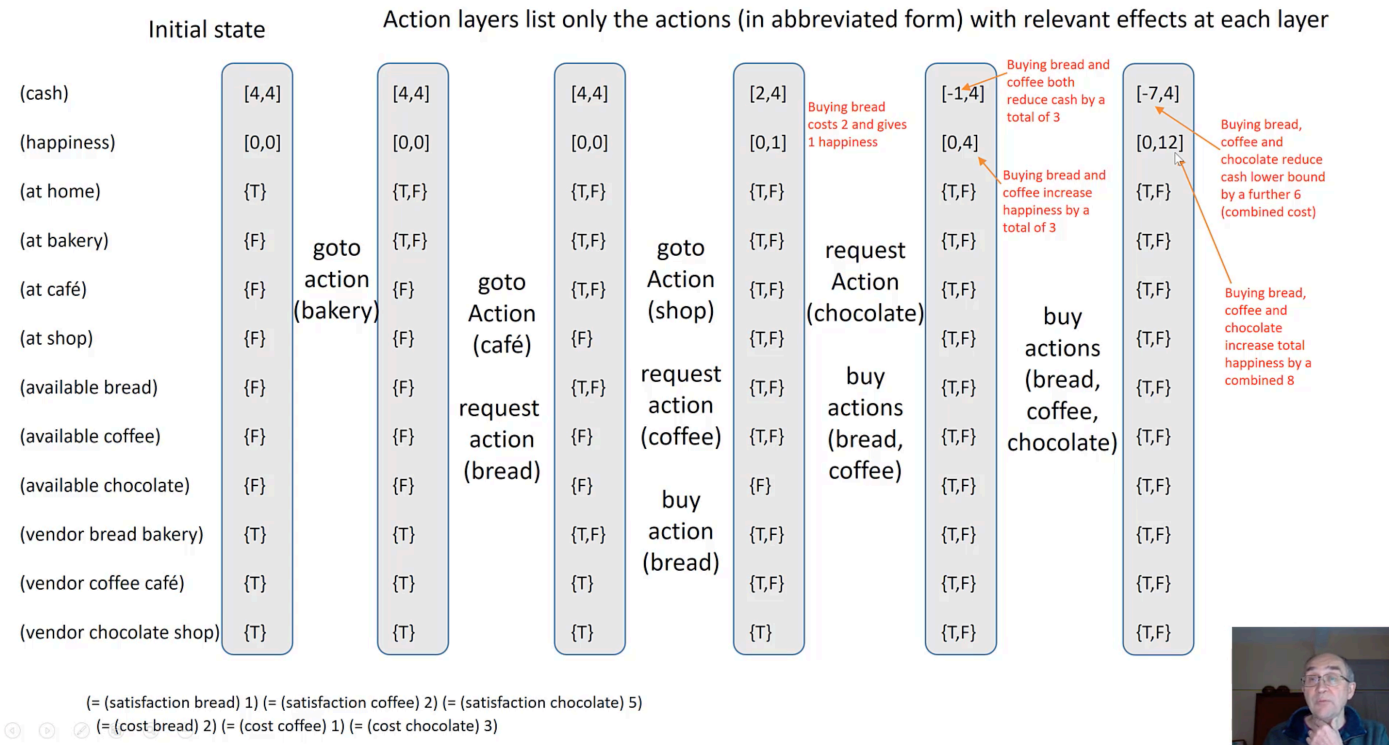
```
(:action buy
parameters (?g - good)
precondition (and (>= (cash) (cost ?g)) (available ?g))
effect (and (not (available ?g)) (decrease (cash) (cost ?g)) (increase (happiness) (satisfaction ?
g))))
(:action goto
parameters (?f?t-location)
precondition (and (connected ?f ?t) (at ?f))
effect (and (not (at ?f)) (at ?t)))
(:action request
parameters (?g - good ?loc- location)
:precondition (and (at ?loc) (vendor ?g ?loc))
:effect (and (not (vendor ?g ?loc)) (available ?g)))
```

```
Initially: (at home) (vendor bread bakery) (vendor coffee café) (vendor chocolate shop)
           (= (satisfaction bread) 1) (= (satisfaction coffee) 2) (= (satisfaction chocolate) 5)
           (= (cost bread) 2) (= (cost coffee) 1) (= (cost chocolate) 3) (= (cash) 4) (= (happiness)
0)
           (connected home bakery) (connected bakery cafe) (connected café shop)
Goal: (>= (happiness) 6)
```

Notice that locations are connected in order: Home -> Bakery -> Cafe -> Shop

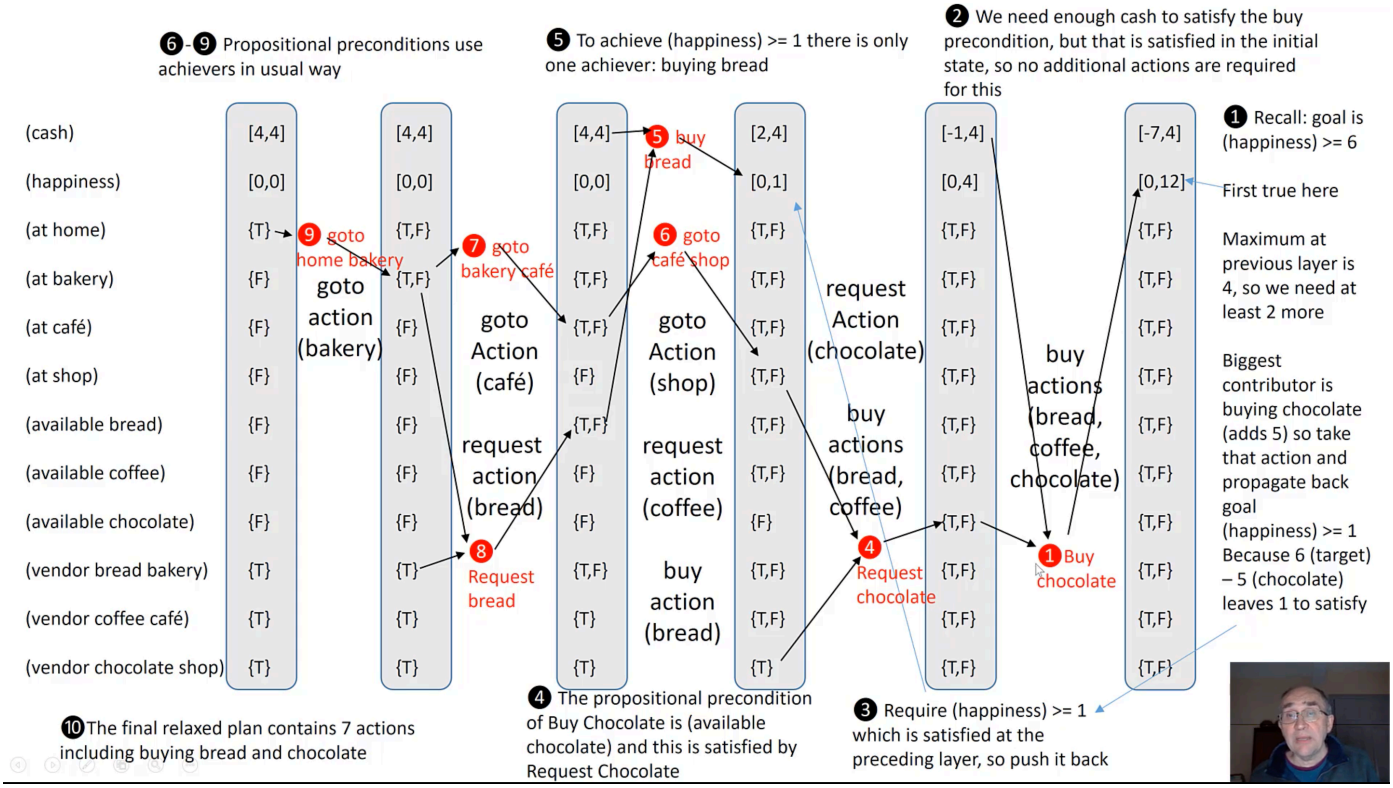
Relaxed Graph:

- Increase the upper bound when increase effects
- Decrease the lower bound when decrease effects



6.2.3 Extracting a relaxed plan

- Find the first interval that satisfy the nemerical goal
- Choose the biggest contributor(action) to the goal
- $goal_{t-1} = goal_t -$ the biggest contribution
- Find the first interval that satisfy $goal_{t-1}$ Then repeat



6.3 Numeric Planning

Planning Language

Fike 和 Nilson 的 STRIPS[15]系统使得规划可以非常容易地进行描述和操作，但随着规划技术的应用，人们发觉 STRIPS 的表达能力非常有限，它不能满足一些实际问题的模型化要求。设计一种能够刻画、模型化一个实际问题的规划问题定义语言成为了规划技术应用的关键，1996 年 E.Pednault[24]提出了动作描述语言——ADL（Action Description Language），ADL 除了具有 STRIPS 的表达能力外，还具有表达条件效果、量化效果等能力。

1998 年 Drew McDermott[25]等提出了规划领域定义语言（The planning Domain Definition Language，PDDL），它逐渐地成为公认的国际智能规划比赛的标准。PDDL

语言不仅给出了规划问题定义的语法，也从语义的角度给出了规划的定义。PDDL 语言的表达能力非常强，它不仅能够表示 STRIPS、ADL 以及其它语言，还能够刻画规划问题的时间和数值方面的属性，给规划器的发展提出了挑战，指明了发展的方向。自 1998 年以来，PDDL 已经作为国际规划大赛的标准语言使用。

目前，PDDL 仍在不断改进中，最先提出的版本是 PDDL1.2，它包含 STRIPS 和 ADL 域；PDDL2.1[26]增加了数值表达、规划尺度和持续动作等新的表达能力；PDDL+[27]对 PDDL2.1 进行重要扩展，采用离散的方法对时间和数值变量模型化；PDDL2.2[28]保留了 PDDL2.1 前三层的表达能力，新增了导出谓词和初始化时间命题的表达能力；PDDL3.0[29]中新增偏好和软约束等特性；PDDL3.1[30]在 PDDL3.0 的基础上新增了对象变量和数值动作代价。