人工智能笔记

陈鸿峥

2019.10*

目录

1 简介

1.1 概述

- 1997 Deep Blue
- 2011 IBM Watson
- 2016 Google DeepMind

---

*Build 20191015

什么是AI？

- 像人类一样思考(thinking humanly)：中文屋子
- 理智思考(thinking rationally)
- 像人类一样行为(acting humanly)：图灵测试(1950)
- 理智行为(acting rationally)

常见术语

- 强AI：机器像人类一样思考
- 弱AI：机器有智能的行为
- 通用AI(AGI)：能够解决任何问题
- 窄AI：专注于某一特定任务

不以模拟人类作为实现人工智能的最好方法

- 计算机和人类的体系结构不同：数值计算、视觉、并行处理
- 对人类大脑的了解太少了！

1.2 历史

- 1950-70: Early excitement, great expections
  - Samuel(1952)跳棋程序
  - Newell(1955)逻辑理论家
  - Dartmouth会议(1956)：AI诞生
- 1970-90: Knowledge is power
- 1990-: rise of machine learning "AI Spring"
- 2010-: Deep learning

2 搜索

搜索主要包括无信息(uninformed)搜索和有信息搜索。

- 状态空间(state space)
  - 传统搜索：状态空间可见、动作确定性
  - 非传统搜索：局部搜索、模拟退火、爬坡
- 动作(action)：不同状态之间的转换
- 初始状态(initial state)
- 目标/期望(goal)

树搜索，边界集(frontier)是未探索的状态集合

Algorithm 1 Tree Search

```
procedure TreeSearch((Frontier, Successors, Goal?))
    if Frontier is empty then
        return failure
    Curr = select state from Frontier
    if Goal?(Curr) then
        return Curr
    Frontier' = (Frontier − {Curr}) ∪ Successors(Curr)
    return TreeSearch(Frontier',Successors,Goal?)
```

搜索需要关注的几个特性：

- 完备性：若解存在，搜索是否总能找到解
- 最优性：是否总能找到最小代价的解
- 时间复杂性：最大需要被**生成或展开**[1]的结点数
- 空间复杂性：最大需要被存储在内存中的结点数

2.1 无信息搜索

2.1.1 宽度优先搜索(BFS)

将后继加入边界集的后面，b为最大状态后继数目/分支因子(branching factor)，d为最短距离解的行动数（注意是**边数**，而不是层数！）

- 完备性与最优性：所有短路总在长路前被探索，某一长度只有有限多条路径，最终可以检测所有长度为d的路径，从而找到最优解
- 时间复杂度：$1 + b + b^2 + \cdots + b^d + (b^d - 1)b = O(b^{d+1})$，最差情况在最后一层的最后一个节点才探索到最优解，从而前面b个节点都要展开第$d + 1$层
- 空间复杂度：$b(b^d - 1) = O(b^{d+1})$，需要将边界集都存储下来

2.1.2 深度优先搜索(DFS)

将后继加入边界集的前面，即总是展开边界集中最深的节点

- 完备性
  - 无限状态空间：不能保证
  - 有限状态空间无限路径：不能保证
  - 有限状态空间+路径/重复状态剪枝：可以保证

[1]而不是探索的结点数目

- 最优性：因完备性不能保证，故最优性也不能保证
- 时间复杂性：$O(b^m)$，其中m为状态空间的最长路的长度（若$m >> d$，则非常糟糕；但如果有大量解路径，则会快于BFS）
- 空间复杂性：$O(bm)$，**线性空间复杂性**是DFS最大的优点。边界集只包含当前路径的最深节点以及回溯节点（backtrack points为当前路径上节点的未探索的兄弟sibling）

2.1.3 一致代价(Uniform-cost)

一致代价搜索(Uniform cost search, UCS)[2]的边界集以路径开销升序排序，总是先展开最低开销的路径。如果每一个动作都是一样的代价，则一致代价等价于BFS。

- 完备性与最优性：假设所有转移都有代价$\geq \varepsilon > 0$，所有更低代价的路径都在高代价路径之前被展开，只有有限多的路径开销小于最优解的开销，故最终一定会到达最优解
- 时间复杂性：$O(b^{C^\star/\varepsilon+1})$，对应着BFS中$d = C^\star/\varepsilon$，其中$C^\star$为最优解的开销，最坏情况就是每一层开销都很小为$\varepsilon$
- 空间复杂性：$O(b^{C^\star/\varepsilon+1})$

2.1.4 深度受限搜索(Depth-limited)

执行只在最大深度执行DFS，因此无穷路径长不会存在问题

- 完备性与最优性：不能保证，若解的深度大于L
- 时间复杂度：$O(b^L)$
- 空间复杂度：$O(bL)$

2.1.5 迭代加深搜索(Iterative Deepening)

迭代加深搜索(Iterative Deepening Searching, IDS)逐渐增加最大深度L，对每一个L做深度受限搜索

- 完备性：可以保证
- 最优性：如果开销一致[3]，则可以保证
- 时间复杂性：$(d+1)b^0 + db + (d-1)b^2 + \cdots + b^d = O(b^d)$，第0层搜了$(d+1)$次，可以看到时间复杂度是**比BFS优**的
- 空间复杂性：$O(bd)$，同DFS

[2]至于为什么叫Uniform，可以看https://math.stackexchange.com/questions/112734/in-what-sense-is-uniform-cost-search-uniform和https://cs.stackexchange.com/questions/6072/why-is-uniform-cost-search-called-uniform-cost-search，比较合理的解释是到达同一结点的cost都被认为是相同的（寻找最优解时）。一致的算法总是选择边界集中第一个元素。

[3]若开销不一致，则可以采用代价界(cost bound)来代替：仅仅展开那些路径开销小于代价界的路径，同时要记录每一层深搜的最小代价。这种方式的搜索开销会非常大，有多少种不同路径开销就需要多少次迭代循环。

2.1.6 双向搜索(Bidirectional)

从源结点和汇结点同时采用BFS，直到两个方向的搜索汇聚到中间。

- 完备性：由BFS保证
- 最优性：若一致代价则可保证
- 时间复杂性：$O(b^{d/2})$
- 空间复杂性：$O(b^{d/2})$

2.1.7 环路/路径检测

- 环路(cycle)检测：检测当前状态是否与已探索的状态重复(BFS)
- 路径(path)检测：只检测当前状态是否与该路径上的状态重复(DFS)

注意不能将环路检测运用在BFS上，因为开销太大。

环路检测运用到UCS上依然**可以保证最优性**[4]。因为UCS第一次探索到某一状态的时候已经发现最小代价路径，因而再次探索该状态不会发现路径比原有的更小。

2.1.8 总结

	BFS	UCS	DFS	Depth-limited	IDS	Bidirectional
完备性	✓	✓	✗	✗	✓	✓
时间复杂度	$O(b^d)$	$O(b^{\lfloor C^\star/\varepsilon \rfloor+1})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
空间复杂度	$O(b^d)$	$O(b^{\lfloor C^\star/\varepsilon \rfloor+1})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
最优性	✓	✓	✗	✗	✓	✓

例 1. *N个传教士和N个食人族要过河，他们都在河的左岸。现在只有一条船能够运载K个人，要把他们都运往右岸。要满足无论何时何地，传教士的数目都得大于等于食人族的数目，或者传教士数目为0。*

分析. 考虑对问题形式化为搜索问题

- 状态(M,C,B)，其中M为左岸传教士数目，C为左岸食人族数目，$B=1$指船在左岸
- 动作(m,c)指运m个传教士和c个食人族到对岸
- 先决条件：传教士数目和食人族数目满足限制
- 效果：$(M,C,1) \overset{(m,c)}{\Longrightarrow} (M-m,C-c,0)$
  $(M,C,0) \overset{(m,c)}{\Longrightarrow} (M+m,C+c,1)$

2.2 有信息搜索

在无信息搜索中，我们从不估计边界集中最有期望(promising)获得最优解的结点，而是无区别地选择当前边界集中第一个结点。然而事实上，针对不同问题我们是有对结点的先验知识(apriori knowledge)的，即从当前结点到目标结点的开销有多大。而这就是有信息搜索(informed)，或者称为启发式搜索(heuristics)。

[4]注意这在启发式搜索中不一定成立

关键在于领域特定启发式函数$h(n)$的设计，它估计了从结点n到目标结点的开销(cost)。注意满足目标状态的结点$h(n) = 0$。

2.2.1 贪心最优搜索(Greedy Best-First Search)

直接使用$h(n)$对边界集进行排序，但这会导致贪心地选择**看上去**离目标结点开销最小的路径。

如果存在环路，贪心最优搜索是不完备的，会陷入死循环。

2.2.2 A*搜索

综合考虑当前已走的开销和未来估计的开销。定义一个估值函数

$$f(n) = g(n) + h(n)$$

其中$g(n)$为路径到节点n的代价，$h(n)$为从n到目标节点的代价，采用$f(n)$对边界集内的节点进行排序。

$f(n)$需要满足下列两个性质。

定义 1 (可采纳的(admissibility)). 假设所有代价$c(n1 \to n2) \geq \varepsilon > 0$，令$h^\star(n)$为从$n$到目标节点$\infty$的最优解[5]，若

$$\forall n: \ h(n) \leq h^\star(n)$$

则称$h(n)$是可采纳的。即一个可采纳的启发式函数总是**低估**了当前结点到目标结点的真实开销（这样才能保证最优解不被排除）。

定义 2 (一致性(consistency)/单调性(monotonicity)). 若对于所有的结点n_1和n_2，$h(n)$满足（三角不等式）

$$h(n_1) \leq c(n_1 \to n_2)_h(n_2)$$

则称$h(n)$是单调的。

定理 1. 一致性蕴含可采纳性

分析. 分类讨论

- 当结点n没有到目标结点的路径，则$h(n) \leq h^\star(n) = \infty$恒成立
- 令$n = n_1 \to n_2 \to \cdots \to n_k$为从结点$n$到目标结点的最优路径，则可以用数学归纳法证明$\forall i: h(n_i) \leq h^\star(n_i)$，如下从后往前推

$$h(n_i) \leq c(n_i \to n_{i+1}) + h(n_{i+1}) \leq c(n_i \to n_{i+1}) + h^\star(n_{i+1}) = h^(n_i)$$

定理 2. 可采纳性蕴含最优性

分析. 假设最优解有开销$C^\star$，则任何最优解一定会在开销大于$C^\star$的路径之前被展开。因此在最优解展开之前的路径一定有开销$\leq C^\star$，最终我们一定会检测到最优解，而且次优解不会在最优解之前被检测。

[5]如果没有路径则$h^\star(n) = \infty$

做环检测可能导致找不到最优解

但如果满足单调性，有以下几个性质

命题 1. 路径上的f一定是非递减的

分析.

$$\begin{aligned} f(n) &= g(n) + h(n) \\ &\le g(n) + c(n \to n') + h(n') \\ &= g(n') + h(n') \\ &= f(n') \end{aligned}$$

命题 2. 如果n_2在n_1之后被扩展，则$f(n_1) \le f(n_2)$

分析. n_2在边界上 n_1

命题 3. 当n在任何小于f值得路径之前被展开

分析.

命题 4. $A^\star$算法第一次展开某个状态时，它已经找到了到达那个状态的最小开销路径。

分析.

若满足单调性，则进行环检测不会破坏最优性

2.2.3 迭代加深$A^\star$(IDA)算法

$A^\star$算法有和BFS或UCS同样的空间复杂性问题，而迭代加深$A^\star$算法同样解决空间复杂度的问题

2.2.4 构造启发式函数

常常需要考虑一个更加简单的问题，然后让$h(n)$为到达一个简单问题解的开销

例 2. 现有积木若干，积木可以放在桌子上，也可以放在另一块积木上面。有两种操作:

1. `move(x,y)`: 把积木x放到积木y上面，前提是积木x和积木y上面都没有其他积木
2. `moveToTable(x)`: 把积木x放到桌子上，前提是积木x上面无其他积木，且积木x不在桌子上

设计一个可采纳的启发式函数$h(n)$

分析. $h(n) = h_1(n) + h_2(n)$

2.3 博弈树搜索

博弈的一些前提

- 两个博弈玩家
- 离散值：游戏和决策都可以映射到离散空间

- 有限的：只有有限的状态和可能的决策
- 零和博弈：完全竞争，即如果一个玩家胜利，则另外一个失去同样数量的收益
- 确定性的：没有牵涉到概率性事件，如色子、抛硬币等
- 完美信息博弈：状态的所有方面都可以被完全观察，即没有隐藏的卡牌

剪刀石头布是简单的一次性(one-shot)博弈

- 一次移动
- 在博弈论中称为策略或范式博弈(strategic/normal form)

但很多游戏是牵涉到多步操作的

- 轮回(turn-taking)游戏，如棋类
- 在博弈论中称为扩展形式博弈(extensive form)

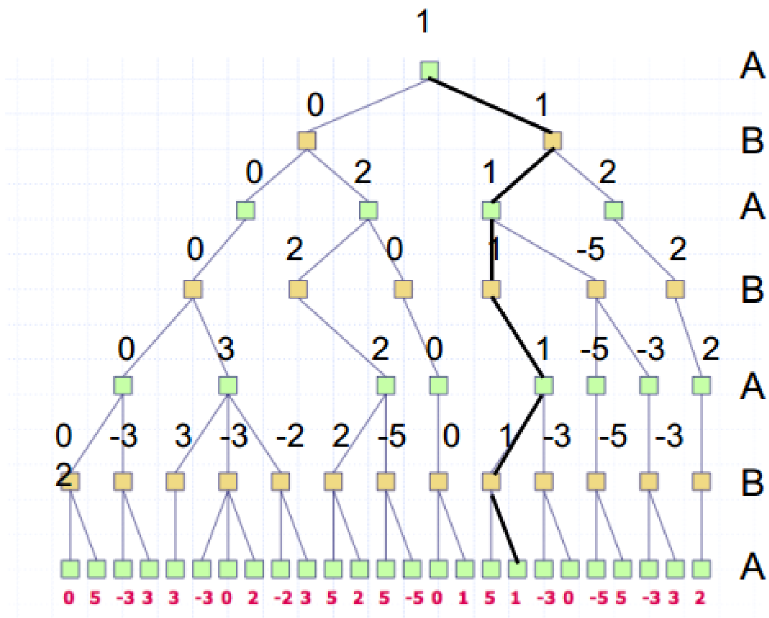
两个玩家A（最大化己方收益）和B（最小化对方收益）

- 状态集合$\mathcal{S}$
- 初始状态$I \in \mathcal{S}$
- 终止位置$T \subset \mathcal{S}$
- 后继：下一可能状态的集合
- 效益(utility)/收益(payoff)函数$V : T \mapsto \mathbb{R}$，表明终止状态对A玩家有多好，对B玩家有多坏（都站在A角度给出）

minimax算法：自己选max，对方选min

- 构建整棵博弈树，然后将终止/叶子结点标上收益
- 回溯整棵树，然后将每个结点都标记上收益

$$U(n) = \begin{cases} \min\{U(c) : \ c \text{ is a child of } n\} & n \text{ is a Min node} \\ \max\{U(c) : \ c \text{ is a child of } n\} & n \text{ is a Max node} \end{cases}$$



用DFS可以遍历整棵树，同时保持线性的空间复杂度，每次回溯时更新结点为min/max即可

α-β剪枝

- 只要当前Max结点的值$\geq$祖先某一Min结点的值，就可以在该Max结点上做α剪枝
- 只要当前Min结点的值$\leq$祖先某一Max结点的值，就可以在该Min结点上做α剪枝

Algorithm 2 Alpha-Beta Pruning

```
procedure AlphaBeta(n,Player,alpha,beta)
    if n is TERMINAL then
        return V(n)                                    ▷ Return terminal states utility
    ChildList = n.Successors(Player)
    if Player == MAX then
        for c in ChildList do
            alpha = max(alpha, AlphaBeta(c,MIN,alpha,beta))
            if beta ¡= alpha then
                break
        return alpha
    else                                               ▷ Player == MIN
        for c in ChildList do
            beta = min(beta, AlphaBeta(c,MAX,alpha,beta))
            if beta ¡= alpha then
                break
        return beta
                          ▷ Initial call: AlphaBeta(START-NODE,Player,−∞,+∞)
```

可以证明，如果原始情况需要访问$O(b^D)$个结点，则经过α-β剪枝后只需访问$O(b^{D/2})$个结点。

但在现实生活的游戏中，即使采用了α-β剪枝，博弈树也太过庞大。如棋类的分支因子大致是35，深度为10的树已经到2.7×10^{14}个结点。因此不能将整棵博弈树展开，需要采用一些启发式方式进行估计。

评价函数(evaluation)的一些需求：

- 对于终止状态，评价函数的序应与真实的收益函数相同
- 对于非终止状态，评价函数则应该与真实的胜率相关联
- 计算时间不能花太长
- 通常取多个特征，然后进行加权求和（先验知识）

在线(online)/实时(real-time)搜索

- 没有办法展开全部的边界集，因此限制展开的大小（在没找到去目标的真实路径就做出决定/直接选一条路就开始走）
- 在这种情况下，评价函数不仅仅引导搜索，更是提交真实的动作
- 虽然找不到最优解，但是求解时间大大缩减

3 限制可满足性问题

在搜索问题中，状态表示是个黑箱，可以有多种多样的方法来表达。但实际上我们可以有特定的状态表示方法来解决大量不同的问题，在这种情况下的搜索算法可以变得很高效。

限制可满足性问题(Constraint Satisfaction Problem, CSP)指每一个状态都可以用一组特征值向量表示的问题。

- k个特征/变量的集合$V_1, \ldots, V_n$
- 每一个变量都有一个包含有限值的论域$\mathrm{dom}[V_i]$，如

$$\mathrm{height} = \{\mathrm{short}, \mathrm{average}, \mathrm{tall}\}$$

- 一组限制条件$C_1, \ldots, C_m$
  - 每个限制条件都有一个作用域(scope)，表示作用在什么变量上，如$C(V_1, V_2, V_4)$
  - 相当于一个布尔函数，从变量赋值到布尔值的映射，如

$$C(V_1 = a, V_2 = b, V_4 = c) = True$$

  - 布尔函数可以以表形式给出，或以表达式形式给出，如$C(V_1, V_2, V_4) = (V_1 = V_2 + V_4)$
- 一个状态可以通过给每一个变量赋值得到

CSP不关心到目标状态的移动步骤，而只关心是否存在这样一组变量满足目标。

3.1 回溯(backtracking)搜索

对每一个变量分别赋值，深搜方式，同时结合启发式函数，用于在每一步选择不同的赋值变量。

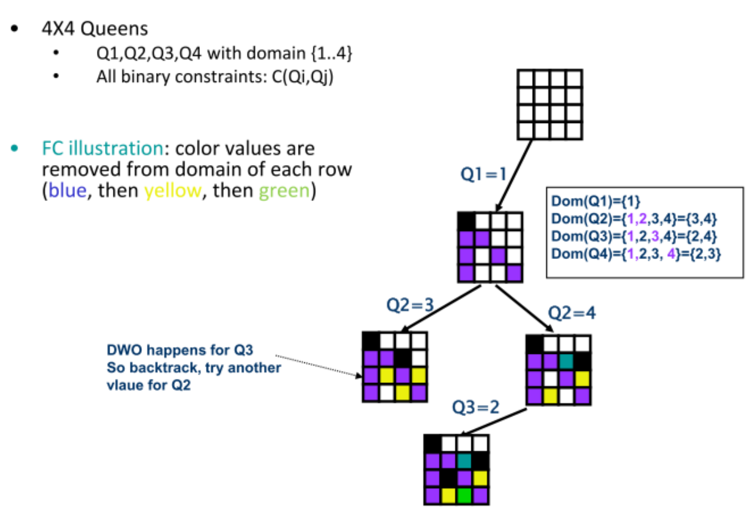
```
BT(Level)
  If all variables assigned
      PRINT Value of each Variable
      RETURN or EXIT (RETURN for more solutions)
                     (EXIT for only one solution)
  V := PickUnassignedVariable()
  Assigned[V] := TRUE
  for d := each member of Domain(V)  (the domain values of V)
      Value[V] := d
      ConstraintsOK = TRUE
      for each constraint C such that
              a) V is a variable of C and
              b) all other variables of C are assigned:
                 ;(rarely the case initially high in the search tree)
           IF C is not satisfied by the set of current
              assignments:
              ConstraintsOK = FALSE
      If ConstraintsOk == TRUE:
           BT(Level+1)

  Assigned[V] := FALSE  //UNDO as we have tried all of V's values
  return
```

回溯的问题在于不能提前探测到某一变量已经没有可以赋值了，导致依然要进入一层进行搜索。因此考虑前瞻式算法，即限制传播(propagation)。

- 甚至可以在还未进行搜索之前就采用
- 传播本身需要耗费资源，因此这里存在一个权衡

3.2 向前检测

1. 选择一个未被赋值的变量V。这里可以采用最小剩余值(Minimum Remaining Values, MRV)作为启发式函数，即先选论域小的变量。
2. 选择论域dom$[V]$中的值对V进行赋值d
3. 将d向前传递给含有V的限制C，主要考察那些**只剩一个未赋值变量**X的限制
4. 检测`FCCheck(C,X)`是否出现论域清空(Domain Wipe Out, DWO)，即X没得选值了。这里`FCCheck`做的则是核心的限制传播部分，当$V=d$后把X不能取的值删去
5. 如果不出现DWO，则进入下一层（选择新的未赋值变量赋值）
6. 否则需要恢复当层`FCCheck`剪枝的部分，即d不可取，V要重新取值



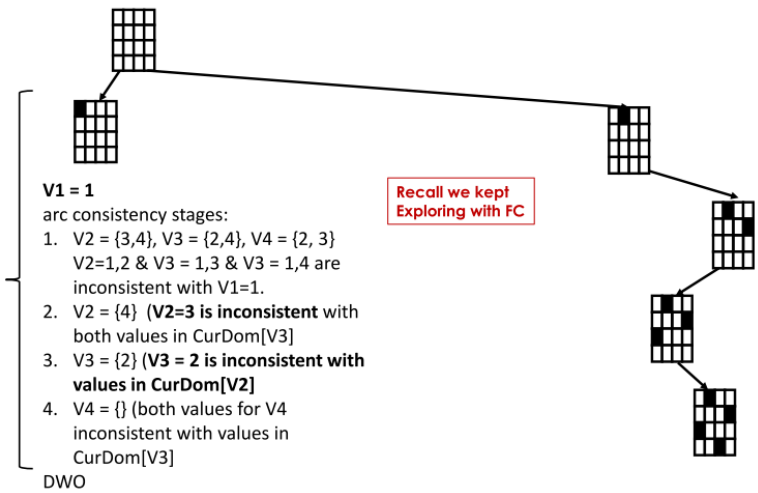
3.3 一般性边一致性(GAC)

定义 3 (一致)**.** 限制$C(X,Y)$是一致的，当且仅当对于所有X的值都存在某些Y满足C，即$\forall X \exists Y : C(X,Y)$。

定义 4 (一般性边一致性(Generalized Arc Consistency, GAC))**.** 限制$C(V_1,V_2,\ldots,V_n)$是关于V_i边一致的，当且仅当$\forall V_i, \exists V_1,\ldots,V_{i-1},V_{i+1},\ldots,V_n$满足$C$。限制$C$是$GAC$的当且仅当对于每一变量都是$GAC$的。一个$CSP$是$GAC$的当且仅当它的限制都是$GAC$的。

有GAC算法：

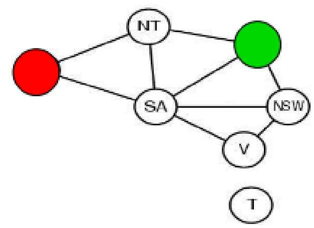
- 在$V_i = d$下，没有其他变量赋值能够满足该限制，则d是边不一致的，进而可以被剪枝剪掉。
- 注意当从论域中移除一个值时可能导致新的不一致，故需要采用队列的方式，不断将需要检测边一致性限制添加，直到队列为空，即限制条件变为GAC。
- 近似可理解为将后续搜索步骤都前移到剪枝部分。



向前检测和边一致性检测的区别如下

- Assign *{Q=green}*
- Effects on other variables connected by constraints with Q
    - *NT can no longer be green = {B}*
    - *NSW can no longer be green = {R, B}*
    - *SA can no longer be green = {B}*
- *DWO there is no value for SA that will be consistent with NT ≠ SA and NT = B*

Note Forward Checking would not have detected this DWO.



4 知识表示与推理

一阶逻辑(First-Order Logic,FOL)

- 个体/常量(0-ary)
- 类型(unary)谓词：$A(x), B(x)$
- 关系（二元谓词）：$L(x, y)$

定义 5 (项(term)). 每一个变量都是一个项。若$t_1, \dots, t_n$都为项，且f为n参数的的函数，则$f(t_1, \dots, t_n)$是一个项。

定义 6 (公式(formular)). 公式包括以下几种情况：

- 若$t_1, \dots, t_n$都是项，且P是n元的谓词符号，则$P(t_1, \dots, t_n)$是一个公式
- 若t_1, t_2都是项，那么$(t_1 = t_2)$是一个原子公式
- 若α, β都是公式，v是一个变量，则$\neg\alpha, (\alpha \wedge \beta), (\alpha \neg \beta), \exists v.\alpha, \forall v.\alpha$都是公式

定义 7 (句子(sentence)). 没有自由变量的公式

定义 8 (替换). $\alpha[v/t]$表示α中所有自由出现的v都用项t替代

定义 9 (解释(interpretation)). 一个解释是一个对*(pair)*$\mathcal{I} = \langle D, I \rangle$，其中

- D是论域，可以是任何非空集
- I是从谓词到函数符号的映射
- 如果P是一个n-参数的谓词符号，$I(P)$是一个在D上的n-参数的关系，即$I(P) \subset D^n$

定义 10 (赋值(denotation)). 变量指派*(assignment)*μ是一个从变量集合到论域D的映射

$$\|v\|_{\mathcal{I},\mu} = \mu(v)$$
$$\|f(t_1,\ldots,t_n)\|_{\mathcal{I},\mu} = I(f)(\|t_1\|_{\mathcal{I},\mu},\ldots,\|t_n\|_{\mathcal{I},\mu})$$

定义 11 (满足). $\mathcal{I},\mu \models \alpha$读作$\mathcal{I},\mu$满足$\alpha$

- $\mathcal{I},\mu \models \alpha \iff \langle \|t_1\|_{\mathcal{I},\mu},\ldots,\|t_n\|_{\mathcal{I},\mu}\rangle \in I(P)$
- $\mathcal{I},\mu \models (t_1 = t_2) \iff \|t_1\|_{\mathcal{I},\mu} = \|t_2\|_{\mathcal{I},\mu}$

定义 12 (子句(clause)). 文字*(literal)*是原子公式或它的取反，一个子句是文字的析取*(disjunction)*，如$p \vee \neg r \vee s$，写作$(p,\neg r,s)$。特殊地，空子句$()$代表为假。公式*(formula)*则是子句的合取*(conjunction)*。

归结(resolution) 反驳(refutation)

$$\vdash$$

- 消除蕴含：$A \to B \iff \neg A \vee B$
- 将非向内推：德摩根定律
- 标准化变量：重命名变量使得每一个量词都是唯一的
- 消除存在量词(skloemize)：引入新的函数符号，如$\forall x P(x)$改为$P(g(y))$
- 将所有量词带到最前面：只有全局量词，且名字均不同
- 析取分配到合取
- 压平
- 转化为子句：将量词全部移除

定义 13 (MGU). 两个公式f和g的替换σ

-
-

计算MGU的算法：不断代入新的元，使其一致
利用归结（两条文字合一变真删除）看是否能得到空子句
答案抽取(answer extraction)

- 将询问$\exists x P(x)$用$\exists x[P(x) \wedge \neg \text{answer}(x)]$替换（因为取非后变成$\forall x P(x) \implies \text{answer}(x)$）
- 直到获得任意子句只包含答案的谓词

例 3. 对下列查询进行归结及答案查询

- *Whoever can read $R(x)$ is literate $L(x)$*
- *Dolphins $D(x)$ are not literate*
- *Flipper is an intelligent dolphin $I(x)$*

Who is intelligent but cannot read?

分析. 对语句进行形式化

$\forall x(R(x) \to L(x))$	*1*	$(\neg R(u), L(u))$
$\forall x(D(x) \to \neg L(x))$	*2*	$(\neg D(v), \neg L(v))$
$D(Flip) \wedge I(Flip)$	*3*	$D(Flip)$
	4	$I(Flip)$
$Q{:}\exists x(I(x) \wedge \neg R(x))$	*5*	$(\neg I(y), R(y), answer(y))$
$R[4,5]/y = Flip$	*6*	$(R(Flip), answer(Flip))$
$R[1,6]/u = Flip$	*7*	$(L(Flip), answer(Flip))$
$R[2,7]/v = Flip$	*8*	$(\neg D(Flip), answer(Flip))$
$R[3,8]$	*9*	$(answer(Flip))$

因此得到$Flipper$是聪明的但是不能阅读

一组子句是否可满足是NP完全的[Cook,1972]

5 规划

智能体应该能够对世界做出动作(action)，而不仅仅是通过搜索解决问题或推理及知识表示。核心是对动作的效果进行推理，并且计算什么动作能够达成特定的效果。

情景演算(Situation Calculation, SitCalc)三个基本部分

- 动作(action)
- 情景(situations)：动作序列，$do(a,s)$为动作、情景到新情景的函数映射，S_0为初始情景

$$do(put(a,b), do(put(b,c), S_0))$$

要区别情景与状态(state)，如将硬币转两次，情景/动作历史不同，但状态都是一样的

- 流(fluent)：从情景到情景的谓词或函数（动态变化过程）
- 条件(precondition)：动作执行的前提条件
- 影响(effect)：执行动作后改变的流。如下在情景s下执行修复动作后，x就不是破碎的

$$\neg Broken(x, do(repair(r,x)), s)$$

只陈述了执行动作的影响，而没有阐述没影响的部分

框架(frame)问题：找到一种高效的方法来确定动作的非效果(non-effects)而不是显式地将它们全部写下来，用一阶逻辑

利用归结进行情景演算

传统的规划没有完全或确定的信息，假设对于初始状态有完整的信息。

定义 14 (封闭世界假设(Closed-World Assumption, CWA)). 用于表示世界状态的知识库是一系列正的真实原子事实（与数据库类似）。如$emp(A,C)$不在数据库中，则$\neg emp(A,C)$为真

STRIPS(Stanford Research Institute Problem Solver)：

- 世界被表示成封建世界知识库(CW-KB)，一个STRIPS的动作表示成更新CW-KB的方式
- 一个动作生成新的KB，用以描述新的世界

在SitCalc中我们可能有不完全的信息（用一阶逻辑公式表示），而在STRIPS中，我们有完整的信息（用CW-KB）表示

例子如下： $pickup(X)$:

- $Pre : handempty, clear(X), ontable(X)$
- $Adds : holding(X)$
- $Dels : handempty, clear(X), ontable(X)$