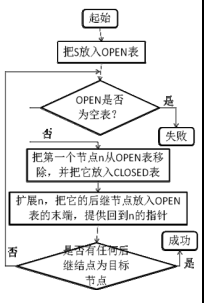
- **宽度优先搜索算法：** $h(n)=0,\ g(n)=d(n),\ f(n)=g+h$

算法对 Open 表的操作规则是**先进先出**，可以使用数据结构中的**队列**来实现。



（1）把起始节点放到Open表中，如果该起始节点为一目标节点，则求得一个解，算法结束。

（2）如果Open 是个空表，则没有解，失败退出；否则继续。

（3）把第一个节点（节点n）从Open 表移出（先进先出），并把它放入Closed 的扩展节点表中。

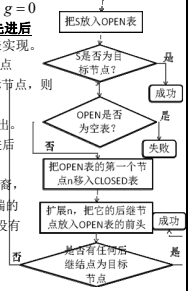
（4）扩展节点n。如果n 没有后继节点，则转向步骤（2）

（5）把n 的**所有后继节点放到Open 表的末端**(末端的节点比前端的节点后移也），并提供从这些后继节点回到n 的指针。

（6）如果n 的任一个后继节点是一个目标节点，则找到一个解，成功退出；否则转向步骤（2）。

- **深度优先搜索算法：** h 很大，$g=0$

算法对 Open 表的操作规则是**先进后出**，可以使用数据结构中的**栈**来实现。



（1）把起始节点S 放到未扩展节点Open 表中。如果此节点为一目标节点，则得到一个解。

（2）如果Open为空表，则失败退出。

（3）把Open 中第一个节点（先进后出）n 从Open 表移到Closed 表。

（4）扩展节点n ，产生其全部后裔，并把它们放入Open 表的前端(末端的节点比前端的节点后移也)。如果没有后裔，则转向步骤（2）。

（5）如果后继节点中有任一个为目标节点，则求得一个解，成功退出；否则，转向步骤（2）。

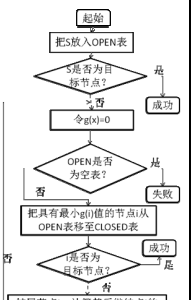
- **等费用算法：** $h(n)=0$

宽度优先+Open(小顶堆)排序

（1）把起始节点S 放到未扩展节点表Open 中。如果此起始节点为一目标节点，则求得一个解；否则令g(S)=0。

（2）如果Open 是个空表，则没有解而失败退出。

（3）从Open 表中选择一个节点m，使其$g(m)$为最小。如果有几个节点都合格，那么就要选择一个目标节点作为节点m（要是有目标节点的话）；否则，就从中任选一个作为节点m 。把节点m 从Open 表移至扩展节点表Closed 中。



（4）如果节点m 为目标节点，则求得一个解。

（5）扩展节点m 。如果m 没有后继节点，则转向步骤（2）。

（6）对于节点m的每个后继节点n，计算g(n)=g(m)+c(n, m)，并把所有后继节点n 放进Open 表。提供回到节点m 的指针。

（7）转向步骤（2）。

- **爬山法：（贪婪）** $g=0,\ h(n)$ **为与目标节点的距离**

（1）把起始节点S 放到未扩展节点表Open 中。如果此起始节点为一目标节点，则求得一个解；否则计算$h(S)$。

（2）如果Open 是个空表，则没有解而失败退出。

（3）从Open 表中选择一个节点m，使其$h(m)$为最小。如果有几个节点都合格，那么就要选择一个目标节点作为节点m（要是有目标节点的话）；否则，就从中任选一个作为节点m 。把节点m 从Open 表移至扩展节点表Closed 中。

（4）如果节点m 为目标节点，则求得一个解。

（5）扩展节点m 。如果m 没有后继节点，则转向步骤（2）。

（6）对于节点m的每个后继节点n，计算$h(n)$，并把所有后继节点n放进Open表。提供回到节点m 的指针。

（7）转向步骤（2）。

爬山法**不总是**能够找到最优解。

- ***A* 算法：** g **已走过的路，** h ***与目标距离，估计** $f(n)=g+h$

（1）把起始节点S 放入Open 表, 记f(S)= h(S), 令Closed 为空表

（2）若Open 为空表，则宣告失败。

（3）选取Open 表中具有最小f 值的节点为最佳节点BestNode，并把它放入Closed 表。

（4）若 BestNode 为一目标节点，则成功求得一解。

（5）若BestNode不是目标，则扩展之，产生后继节点 Successor

（6）对每个Successor 进行下列过程：

（a）建立从Successor 返回BestNode 的指针。

（b）计算g(Successor)=g(BestNode)+ c(Successor,BestNode)。

（c）如果Successor 在Open 表中，则称Open 表中该节点为Old。

（d）比较新旧路径代价。如果g(Successor) <g(old)，则用Successor替换Old节点。

（e）若与Old 节点的代价低或一样，则停止扩展节点。

（f）若 Successor 不在Open 表中，则看其是否在Closed 表中。

（g）若Successor 在Closed 表中，则转向（c）。

（h）若Successor 既不在Open 表中，又不在Closed 表中，则把它放入Open 表中，然后转向（7）。

（7）计算该节点f 值。

（8） 转向（2）

- ***A** 算法：估计* $h^*(n)$ *的下界* $h(n)$ **，完备性可采纳性最优性**
- **P问题**：有多项式时间算法解决的判定问题。
- **NP问题**：对问题的一个猜想存在多项式时间算法来验证的判定问题。
- **多项式可简化**：问题A 对于问题B 是多项式可简化的：A 的算法中要调用B 作为子程序，如果把B 的子程序作为一个步骤，A 的算法是多项式时间算法。记为：A∝B。
- **NP 完全问题**：判定问题$P_1 \in NP$：对所有其它判定问题

$P_m \in NP$，有$P_m \propto P_1$，则P_1是NP 完全的。NP 完全问题是NP中最难的问题。

- **NP 难题**：如果$P_1 \propto P_2, P_1 \in NP$ 完全问题，则P_2 是NP 难

- **问题归约描述**：（1）一个初始问题描述。（2）一套把问题变成子问题的算符。（3）一套本原问题描述
- **与或图的启发式搜索算法：**

1) 初始节点为s，建立一个搜索图G ，开始时图G 只包括s，计算h(s)，若s是叶结点，则标记能解。

2) 重复下面循环，直到 s 已被标记为可解节点：

3) Begin

4) 根据连接符标记（指针）找出一个待扩展的局部解图G'

5) 选G'中的任一非终结点n作为当前结点。

6) 扩展节点 n ，生成其后继结点集$\{n_j\}$，并且把它们作为n 的后继添加到G中。对每一个不曾在G 中出现过的节点x_j，计算$h(x_j)$。把其中属于终节点的后继节点标记为可解节点。

7) 建立含 n 的单一结点集合S 。

8) 重复下面循环，直到S为空

9) Begin

10) 从 S 中删除一个节点m ，该节点在G 中的后继不出现在S中。

11) 修改m 的代价值h(m)：

对m 指向结点集$\{n_{i1}, n_{i2}, \cdots, n_{id}\}$的每一个连接符i分别计算

$h_i(m)=C_i + h(n_{i1}) + h(n_{i2}) + \cdots + h(n_{id})$

h:=min$_i$h$_i$(m)，即对m的i个连接符，取代价最小的那个值作为该节点的代价h(m)。

把指针加到$h_i(m)$最小的连接符上，或把指针修改到$h_i(m)$最小的连接符上。

若该连接符的所有子结点都是可解的，则m 也标为可解。

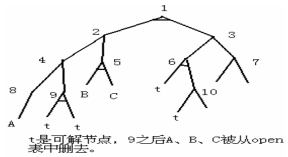
12) 如果m 可解或修正后的代价值与原来估算的代价值不同,则把m的所有父辈结点插入到S 中。

13) end

14) end

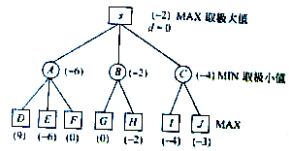
- **与或图的宽度优先搜索**：

1.当找到一个**可解**节点A时，向上回溯确定，A的祖先是否可解，直到不能再回溯为止。然后**去掉**open表中以最高可解节点为祖先的节点。2. 当找到一个**不可解**节点B时，向上回溯确定，B的祖先是否不可解，直到不能再回溯为止。然后**去掉**open表中以最高不可解节点为祖先的节点。

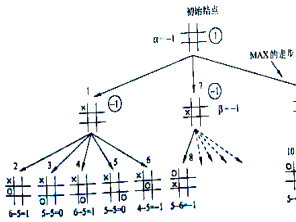


t是可解节点，9之后A、B、C被从open表中删去。

- **博弈树搜索：** 极小极大搜索过程



- **α－β剪枝：**



极大值层的下界值称为α，极小值层的上界值称为β。

① α 剪枝：若任一极小值层结点的β 值小于或等于它任一先辈极大值层结点的α 值，即α（先辈层）≥ β（后继层），则可中止该极小值层中这个MIN 结点以下的搜索过程。这个MIN结点最终的倒推值就确定为这个β 值。

② β 剪枝：若任一极大值层结点的α 值大于或等于它任一先辈极小值层结点的β 值，即α（后继层）≥ β（先辈层），则可以中止该极大值层中这个MAX结点以下的搜索过程。这个MAX结点的最终倒推值就确定为这个α 值。

- **命题逻辑：析取(或,并)∨，合取(与,交)∧。**

能判断真假（不是既真又假）的陈述句为命题。

A∨ (A∧ B) <=> A；A∧ (A∨ B) <=> A；A→ B <=>~ A∨ B；A↔ B <=> (A→ B)∧ (B → A)；(A→ B) ∧ (A→~ B) <=>~ A

+ 不含任何联结词的公式为**原子公式**。
+ 原子或原子的否定形式称为**文字**。任何文字的**析取式**为**子句**。
+ 仅由有限个文字构成的合取式称为**简单合取式**。
+ 仅由有限个文字构成的析取式称为**简单析取式**。
+ 仅由有限个简单析取式构成的合取式称为**合取范式**

如：P ∧ (P ∨Q) ∧ (~ P ∨Q)

+ 仅由有限个简单合取式构成的析取式称为**析取范式**

如：P ∨ (P ∧Q) ∨ (~ P ∧Q)

+ **逻辑公式的子句集S**：合取范式形式的所有子句（元素）的集合。

如：$p\wedge(p\vee q)\wedge(\neg p\vee q) \rightarrow S=\{p,\ p\vee q,\ \neg p\vee q\}$

+ **合取范式的求法：**①削去对于(~,∨,∧)来说冗余的联结词；②内移或削去否定号；③利用分配率。
+ **归结原理**：如p∨q和r∨~ q都为真，则p∨r为真。
+ **归结法证明过程**：①建立待归结命题公式。首先根据反证法将所求证的问题转化成为命题公式，求证其是矛盾式（永假式）。②求取合取范式。③建立子句集。④归结，对子句集中的子句使用归结规则：a）归结式作为新子句加入子句集参加归结；b）归结式为空子句□，停止。若得到空子句，表示S是不可满足的（矛盾），故原命题成立。

- **谓词逻辑**

+ 约束变量换名规则：

$(\forall x)P(x) <=> (\forall y)P(y)$，$(\exists x)P(x) <=> (\exists y)P(y)$

$(\forall x)P(x,z) <=> (\forall y)P(y,z)$，$(\exists x)P(x,z) <=> (\exists y)P(y,z)$

+ **量词否定**等值式：

$\sim(\forall x)P(x) <=> (\exists y)\sim P(y)$

$\sim(\exists x)P(x) <=> (\forall y)\sim P(y)$

+ **量词分配**等值式：注意其他两种不成立

$(\forall x)(P(x) \wedge Q(x)) <=> (\forall x)P(x) \wedge (\forall x)Q(x)$

$(\exists x)(P(x) \vee Q(x)) <=> (\exists x)P(x) \vee (\exists x)Q(x)$

+ 量词**辖域收缩与扩张**等值式：

$(\forall x)(P(x) \vee Q) <=> (\forall x)P(x) \vee Q$

$(\forall x)(P(x) \wedge Q) <=> (\forall x)P(x) \wedge Q$

$(\forall x)(P(x) \rightarrow Q) <=> (\exists x)P(x) \rightarrow Q$

$(\forall x)(Q \rightarrow P(x)) <=> Q \rightarrow (\forall x)P(x)$

$(\exists x)(P(x) \vee Q) <=> (\exists x)P(x) \vee Q$

$(\exists x)(P(x) \wedge Q) <=> (\exists x)P(x) \wedge Q$

$(\exists x)(P(x) \rightarrow Q) <=> (\forall x)P(x) \rightarrow Q$

$(\forall x)(Q \rightarrow P(x)) <=> Q \rightarrow (\exists x)P(x)$

$(\exists x)(P(x) \wedge Q(x)) \neq (\exists x)P(x) \wedge (\exists x)Q(x)$

$(\forall x)(P(x) \vee Q(x)) \neq (\forall x)P(x) \vee (\exists x)Q(x)$

+ **前束范式**：谓词公式P 如果具有以下形式，即把所有的量词都提到最左端去，$(M_1x_1)(M_2x_2)\ldots(M_nx_n)P(x_1,x_2,\ldots,x_n)$则称为P的前束范式，其中$M_i$ 表示量词$\forall$ 或者$\exists$。量词的削去和引入的基本思想是**任意量词可以削去**，**存在量词可以用常量**表示。

如：$\forall x\forall y\forall t((P(x,z) \rightarrow Q(y)) \rightarrow R(t,z))$

+ **Skolem 标准型：**转化过程为，依据约束变量换名规则，首先把公式变型为**前束范式**，然后依照量词消去原则消去或者略去所有量词。量词消去原则为：① 消去存在量词“∃”，即将该量词约束的变量用任意常量(a,b等) 或任意变量的函数($f(x)$, $g(y)$等)代替。如果存在量词左边没有任何任意量词，则只将其改写成为常量；如果左边有任意量词的存在量词，消去时该变量改写成为任意量词的函数。②略去任意量词“∀”，简单地省略掉该量词。

+ **子句集S 的求取过程：**

① 将谓词公式G 转换成前束范式；

② 消去前束范式中的存在量词，略去其中的任意量词，

生成Skolem 标准型（Skolem 标准型必须满足合取范式）；

③ 将 Skolem 标准型中的各个子句提出，表示为集合形式。

+ **归结过程控制策略：**

1. **删除策略**：在归结过程中可以随时删除以下子句：

① 含有永真式的子句；

② 被子句集中其他子句归类的子句。

2. **支撑集策略**：

设有不可满足子句集S 的子集T ，如果$S-T$ 是可满足的，则称T 是S 的支撑集。

采用支撑集策略时，从开始一直到得到空子句的整个归结过程中，只选取不同时属于$S-T$ 的子句对，在其间进行归结。就是说，至少有一个子句来自于支撑集T 或由T 导出的归结式。

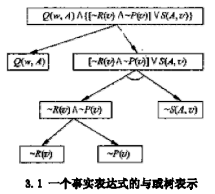
+ **规则正向演绎系统：**

事实(复杂) 推出 目标

注意：此处是反的：

与关系 -> 或连接符

或关系 -> 与连接符



当正向演绎系统产生一个含有以目标节点作为终止的解图时，此系统就成功地终止。

图 3.1 一个事实表达式的与或树表示

+ **规则逆向演绎系统**

目标(复杂) 推出 事实

注意：此处是正的：

或关系 -> 或连接符（**1连接符**）

与关系 -> 与连接符（**k连接符**）

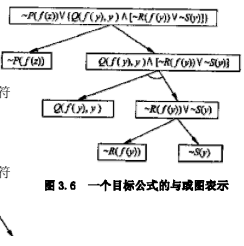


图 3.6 一个目标公式的与或图表示

逆向系统中的事实表达式均限制为文字**合取形**，它可以表示为一个文字集。

- **概念学习：**

+ **FIND-S:寻找极大特殊假设**

1. 将 h 初始化为 H 中最特殊 S 的假设
2. 对每个正例 x

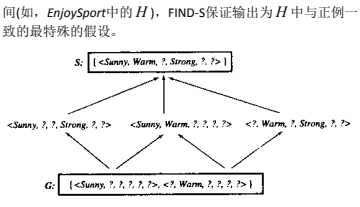
对 h 的每个属性约束a_i

如果 x 满足约束a_i，那么不做任何操作

否则将 h 中a_i替换为能够令 x 满足的下一个更一般约束

3. 输出结果S

+ FIND-S算法的**特点**是：对以属性约束的合取式描述的假设空间(如，*EnjoySport*中的H)，FIND-S保证输出为H 中与正例一致的最特殊的假设。



一般边界G：是在H 中与D 相一致的最一般成员的集合。

特殊边界S：在H 中与D 相一致的最特殊成员的集合。

+ **候选消除算法：**同时找到**特殊边界**S 和**一般边界**G 。

1.初始化： 将G集合初始化为H中最一般假设集

$G_0 \leftarrow \{\langle ?,?,?,?,?,? \rangle\}$

将S集合初始化为H中最特殊假设集合

$S_0 \leftarrow \{\langle \varnothing,\varnothing,\varnothing,\varnothing,\varnothing,\varnothing \rangle\}$

2.对每个训练样本d，进行以下操作：

2.1如果d是一正例

从G中移去所有与d不一致的假设

对S中每个与d不一致的假设s

+ 从S中移去s
+ 把所有比s稍微一般的h加入到S中，

其中h满足：①.h与 d一致，而且G的**每个成员**比h更一般. ②.从S中移去所有这样的假设：它比S中另一假设更一般

2.2如果d是一个反例

从S中移去所有与d不一致的假设

对G中每个与d不一致的假设g

+ 从G中移去g
+ 把所有比g稍微具体的h加入到G中，

其中h满足： ①.h与d一致，而且S的**某个成员**比h更特殊. ②.从G中移去所有这样的假设：它比G中另一假设更特殊。

+ **Remarks**：无偏的学习是没有用的.

上面两种学习算法都是有偏的，因为它们在考虑假设空间的时候都是考虑为属性的合取形式。定义无偏的学习器，将假设空间里的各种假设的析取形式作为判别器。但无偏的学习是没有用的。

如考虑上面提到的 EnjoySport 例子，我们给学习器提供了 3 个正例(x_1,x_2,x_3) 以及两个反例(x_4,x_5)。这时，变型空间的 S 边界包含的假设正好是三个正例的析取：$S:\{x_1 \vee x_2 \vee x_3\}$ 因为这是能覆盖 3 个正例的最特殊假设。相似地，G 边界将由那些刚好能排除掉反例的那些假设组成。$G:\{\neg(x_4 \vee x_5)\}$

S 边界总是所有正例的析取式，G 边界总是所有反例的析取的否定式。每一个新样本都会被变型空间中刚好半数的假设划分为正例，而被另一半划分为反例。

- **决策树：**

+ **CART算法：**

$P(w_j)$ **是节点** N **处属于** w_j **类样本占总样本数的比例.**

熵不纯度：$i(N) = -\sum_j P(\omega_j)\log_2(P(\omega_j))$

Gini不纯度：$i(N) = 1 - \sum_j P^2(\omega_j)$

错分不纯度：$i(N) = 1 - \max_j P(\omega_j)$

不纯度的变化：$\Delta i(N) = i(N) - \sum_{j=1}^{B} P(N_j) i(N_j)$

$i(N)$ 为上层节点的不纯度,

$P(N_j)$ 为分到j节点的元素占总元素的比例

$i(N_j)$ 为在下层某节点的不纯度

寻找查询使得$\Delta i(N)$最大。即**不纯度函数取值最小**.

+ **剪枝**：先让决策树充分生长，然后依次检测每相邻两个叶节点，如果将这两个叶节点合并造成的不纯度的增加小于某个阈值，则合并这两个叶节点。

+ **复杂度**：假如给定n 个d 维训练样本，建树的平均时间复杂度为$O(dn(\log n)^2)$。识别的平均时间复杂度为O(logn)。

- **ID3算法**

1. 可以考虑实数变量，将其按区间划分.
2. 多叉树
3. 问题深度与样本数有关
4. 不考虑剪枝

- **C4.5算法**

1. 考虑剪枝：建议一组规则，对其排序.
2. 当高优先级满足时，即退出

+ **消解推理步骤**:

□要证A⇒B成立 -> 只要证A∧~B不可满足(永假)□

①化A∧~B为合取范式

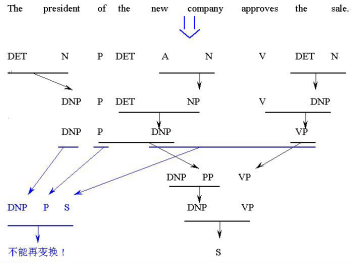
②子句集S={ }

③消解规则用于S，消解式入S中.

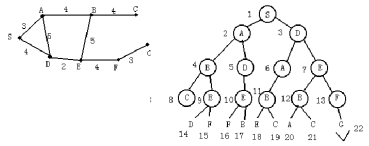
④重复③，直到S中出现空子句。

例题：

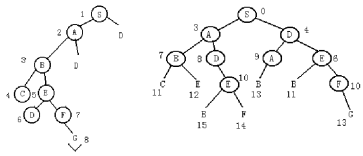
- **句法分析**，已知语法规则：of→P，N→NP，approves →V，new→A，president→N，company→N，sale →N，the→DET，A NP→NP，V DNP →VP，P DNP →PP，DNP PP→DNP，DET NP→DNP，DNP VP→S



- 宽度优先搜索：



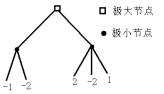
- 深度优先搜索：　　等费用搜索：



1.所有的有理数都是实数：P(x):x是有理数；Q(x):x实数(∀x)(P(x)→Q(x))而不是(∀x)(P(x)∧Q(x)).

2. 有的实数是有理数：(∃x)(Q(x)∧P(x))而不是(∃x)(Q(x)→P(x)).

- **博弈树搜索：**



用α－β过程剪枝：

先遍历根节点的左子树，得到左边那个极小节点的α=-2；
接着遍历右子树，当遍历到-2那个节点是知右子树的β不会大于-2，因此右子树剩下的节点被剪掉。
总共遍历的节点数为四个。
P(x)：x是有理数，Q(x)：x实数

+ 所有的有理数都是实数。

(∀x)(P(x)→Q(x)) 而不是(∀x)(P(x)∧Q(x))

+ 有的实数是有理数

(∃x)(Q(x)∧P(x)) 而不是(∃x)(Q(x)→P(x))

+ 转化为析取和合取范式

1) (⌝P∨Q)→(P↔⌝Q)
= (⌝P∨Q)→((P∧Q)∨(⌝P∧Q))
= ⌝(⌝P∨Q)∨((P∧Q)∨(⌝P∧Q))
= (P∧Q)∨(P∧Q)∨(⌝P∧Q) (析取范式)
= (P∧(Q∨Q))∨(⌝P∧Q)
= P∨(⌝P∧Q)
= (P∨⌝P)∧(P∨Q) (合取范式)
= P∨Q (析取范式)

2) (P∧⌝Q∧S)∨(⌝P∧Q∧R) (析取范式)
= (P∧⌝Q∧S)∨((⌝P∧Q)∧R)
= (((P∧⌝Q)∧S)∨(⌝P∧Q))∧((P∧⌝Q∧S)∨R)
= (((⌝P∧Q)∨(P∧⌝Q))∧((⌝P∧Q)∨S))∧((R∨(P∧⌝Q))∧(R∨S))
= (P∨Q)∧(⌝P∨Q)∧((⌝P∧Q)∨S)∧(R∨(P∧⌝Q))∧(R∨S)
= (P∨Q)∧(⌝P∨Q)∧(⌝P∨S)∧(R∨⌝Q)∧(P∨R)∧(Q∨S)∧(R∨S) (合取范式)

+ 化成子句集

1) G = (((P∨~Q) →R) → (P∧R))
= ~((P∨~Q) →R) ∨ (P∧R)
= ~(~ (P∨~Q) ∨ R) ∨ (P∧R)
= ((P∨~Q) ∧ ~R) ∨ (P∧R)
= (P∨~Q∨(P∧R))∧(~R∨(P∧R))
= (P∨~Q∨P) ∧ (P∨~Q∨R)
∧ (~R∨P) ∧ (~R∨R)
= { P∨~Q, P∨~Q∨R, ~R∨P, ~R∨R}
= { P∨~Q, P∨~Q∨R, ~R∨P}

2) G= (∀x){P(x) → (∀y)[P(y) →P(f(x, y))]
∧ ~ (∀y)[Q(x, y) →P(y)]}
= (∀x){ ~ P(x) ∨ (∀y)[~P(y) ∨ P(f(x, y))]
∧~ (∀y)[~Q(x, y) ∨P(y)]}
= (∀x){ ~ P(x) ∨ (∀y)[~P(y) ∨ P(f(x, y))]
∧ (∃y)[Q(x, y) ∧ ~P(y)]}
= (∀x){ ~ P(x) ∨ (∀y)[~P(y) ∨ P(f(x, y))]
∧ (∃z)[Q(x, z) ∧ ~P(z)]}
= (∀x){ ~ P(x) ∨ (∀y)[~P(y) ∨ P(f(x, y))]
∧ (∃z)[Q(x, z) ∧~P(z)]}
= (∀x) (∀y) (∃z){~P(x)∨~P(y)∨P(f(x, y)),
Q(x, z), ~P(z) }

+ 使用归结法证明下列命题的正确性：

1) P → (Q→P)
2) (P → (Q→R)) → ((P→Q) → (P→R))
3) (Q→~P) → ((Q→P) → ~Q)

解：消解推理步骤：要证 A⇒B 成立 ->
只要证 A∧~B 不可满足(永假)
①化 A∧~B 为合取范式　②子句集 S={ }
③消解规则用于 S，消解式入 S 中.
④重复③，直到 S 中出现空子句。

1) S = ~(Q→P) ∧ P
= Q ∧ ~P ∧ P
= {~P, Q, P}
= {F}　　正确

2) S= ~((P→Q) → (P→R)) ∧ (P → (Q→R))
= ~(~(~P∨Q) ∨ (~P∨R)) ∧ (~P ∨ (~Q∨R))
= ((~P∨Q) ∧ (P∧~R)) ∧ (~P ∨ (~Q∨R))
= { ~P∨Q, P, ~R, ~P∨~Q∨R }
= { Q, P, ~P∨~Q }
= {F}　　正确

3) S = ~((Q→P) → ~Q) ∧ (Q→~P)
= ~(~(~Q∨P) ∨ ~Q) ∧ (~Q∨~P)
= ((~Q∨P) ∧ Q) ∧ (~Q∨~P)
= {~Q∨P, Q, ~Q∨~P}
= {P, ~P}
= {F}　　正确

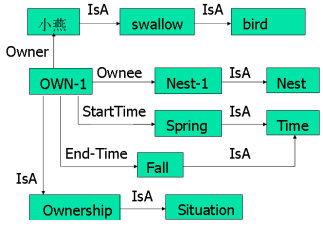
+ Skolem 子句形

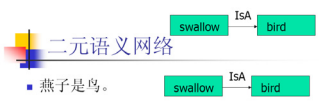
1) ((∃x)P(x)∨(∃x)Q(x))→(∃x)(P(x)∨Q(x))
= ~((∃x)P(x)∨(∃x)Q(x)) ∨ (∃x)(P(x)∨Q(x))
= ((∀x)~P(x) ∧ (∀x)~Q(x)) ∨ (∃x)(P(x)∨Q(x))
= (∀x) (~P(x)∧~Q(x)) ∨ (∃x)(P(x)∨Q(x))
= (∀x)(∃y) { (~P(x)∧~Q(x))∨(P(y)∨Q(y)) }, y=f(x)
=(∀x)(∃y){(~P(x)∨P(y)∨Q(y))∧(~Q(x)∨P(y)∨Q(y))}
S={ (~P(x)∨P(f(x))∨Q(f(x))), (~Q(x)∨P(f(x))∨Q(f(x)))}

2) (∀x)(P(x)→(∀y)((∀z)Q(z,y)→~(∀z)R(y,z)))
= (∀x)(~P(x) ∨ (∀y)(~(∀z)Q(z,y) ∨~(∀z)R(y,z)))
= (∀x)(~P(x)∨(∀y)((∃z) ~Q(z,y) ∨ (∃z) ~R(y,z)))
= (∀x)(~P(x) ∨ (∀y)(∃z)(~Q(z,y) ∨ ~R(y,z)))
= (∀x)(∀y)(∃z)(~P(x) ∨ ~Q(z,y) ∨ ~R(y,z))
= (∀x)(∀y)(~P(x) ∨ ~Q(f(x, y), y) ∨ ~R(y, f(x, y)))
S = { ~P(x) ∨ ~Q(f(x, y), y) ∨ ~R(y, f(x, y)) }

3) (∀x)P(x)→(∃x)(((∀z)Q(x,z))∨(∀y)R(x,y,z))
= ~(∀x)P(x)∨(∃x)(((∀z) Q(x,z)) ∨ (∀y)R(x,y,b))
= (∃x)~P(x)∨(∃x)(((∀z) Q(x,z)) ∨ (∀y)R(x,y,b))
= (∃x){ ~P(x)∨(((∀z) Q(x,z)) ∨ (∀y)R(x,y,b)) }
= (∃x)(∀z)(∀y){ ~P(x) ∨ (Q(x,z) ∨ R(x,y,b) }
S = { ~P(a) ∨ Q(a,z) ∨ R(a,y,b) }

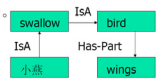
+ **二元语义网络**：“小燕”从春天到秋天占有一个巢





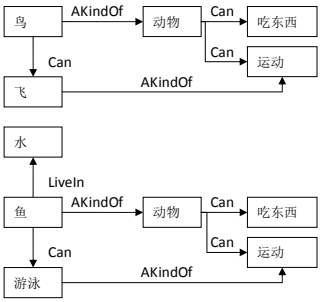
- 燕子是鸟。

- 燕子是鸟。
- “小燕”是一只燕子。
- 鸟有翅膀。



(混用)
IsA关系：实例 -> 类
AKindOf关系：派生类 -> 类

动物会运动、能吃东西；鸟是一种动物，会飞；鱼是一种动物，生活在水里，会游泳。



EnjoySport 候选消除算法，每一步的S、G边界

初始化

S_0: {<Ø, Ø, Ø, Ø, Ø, Ø >}

G_0: {<?,?,?,?,?,?>}

X1:<Sunny Warm Normal Strong Warm Same >, +
X2:<Sunny Warm High Strong Warm Same >, +
X3:<Rainy Cold High Strong Warm Change >, -
X4:<Sunny Warm High Strong Cool Change >, +

X1(+)S 边界学习；G 满足，不变.

S_0: {< Sunny Warm Normal Strong Warm Same >}

G_0: {<?,?,?,?,?,?>}

X1:<Sunny Warm Normal Strong Warm Same >, +

X2:<Sunny Warm High Strong Warm Same >, +
X3:<Rainy Cold High Strong Warm Change >, -
X4:<Sunny Warm High Strong Cool Change >, +

X2(+)S 边界学习，不满足->?；G 满足，不变.

S_0: {< Sunny Warm ? Strong Warm Same >}

G_0: {<?,?,?,?,?,?>}

X1:<Sunny Warm Normal Strong Warm Same >, +
X2:<Sunny Warm High Strong Warm Same >, +

X3:<Rainy Cold High Strong Warm Change >, -
X4:<Sunny Warm High Strong Cool Change >, +

X3(-)S 边界满足(不成立)，不变；G 边界学习

S_0: {< Sunny Warm ? Strong Warm Same >}

G_0: {<Sunny,?,?,?,?,?>,<?,Warm,?,?,?,?>,<?,?,Normal,?,?,?>,<?,?,?,Weak,?,?>,<?,?,?,?,Cool,?>,<?,?,?,?,?,Same>}

X1:<Sunny Warm Normal Strong Warm Same >, +
X2:<Sunny Warm High Strong Warm Same >, +
X3:<Rainy Cold High Strong Warm Change >, -

X4:<Sunny Warm High Strong Cool Change >, +

根据(S<G)，消除 G 边界不满足的部分

S_0: {< Sunny Warm ? Strong Warm Same >}

G_0: {<Sunny,?,?,?,?,?>,<?,Warm,?,?,?,?>,<?,?,?,?,?,Same>}

X1:<Sunny Warm Normal Strong Warm Same >, +
X2:<Sunny Warm High Strong Warm Same >, +
X3:<Rainy Cold High Strong Warm Change >, -

X4:<Sunny Warm High Strong Cool Change >, +

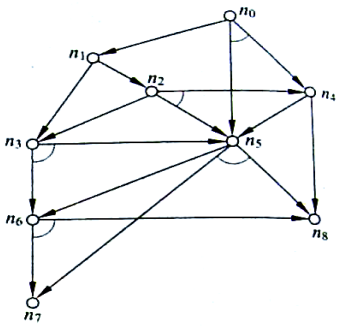
X4(+)S 边界学习；G 边界学习，删除不满足的部分.

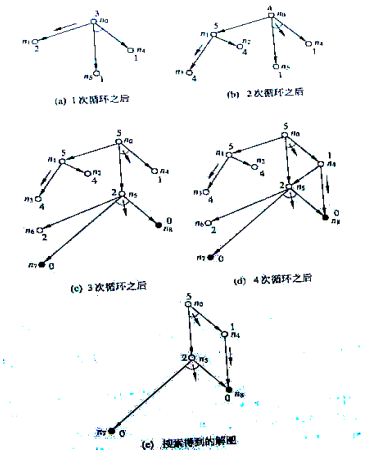
S_0: {< Sunny Warm ? Strong ? ? >}

G_0: {<Sunny,?,?,?,?,?>,<?,Warm,?,?,?,?>}

X1:<Sunny Warm Normal Strong Warm Same >, +
X2:<Sunny Warm High Strong Warm Same >, +
X3:<Rainy Cold High Strong Warm Change >, -
X4:<Sunny Warm High Strong Cool Change >, +

- *AO*：与或图启发式搜索算法





开始的时候认为各节点的代价如下：

h(n)=3，h(n1)=2，h(n2)=4，h(n3)=4，h(n4)=1，h(n5)=1，h(n6)=2，h(n7)=h(n8)=0（目标结点）

- *A-B* 剪枝理论效率

枝技术，搜索树的端结点数 $N_D = B^D$；若使用 $\alpha-\beta$ 剪枝技术，可以证明理想条件下生成的端结点数最少，有

$$N_D = 2B^{D/2} - 1 \quad (D \text{ 为偶数})$$

$$N_D = B^{(D+1)/2} + B^{(D-1)/2} - 1 \quad (D \text{ 为奇数})$$