人工智能原理与算法

第9章-一阶逻辑的推理

中国科学院自动化研究所
国科大人工智能学院

雷 震

9 一阶逻辑的推理

一阶逻辑的推断规则

在本章中将阐述能够回答所有可解的一阶逻辑问题的算法。

- 首先介绍量词的推断规则；
- 然后介绍如何用**合一**来构建直接用于一阶逻辑的推断规则；
- 最后讨论一阶逻辑推断的3类主要算法： **前向链接**、**反向链接**和**基于归结的定理证明**。

- 命题推断与一阶推断
- 合一与一阶推断
- 前向链接
- 反向链接
- 归结

- 命题推断与一阶推断
- 合一与一阶推断
- 前向链接
- 反向链接
- 归结

9.1 命题推断与一阶推断

一阶推断转为命题推断

进行一阶推断的方法之一是将一阶知识库转换为命题逻辑并使用已知的命题推断。

- 第一步是消去全称量词，例如，“贪婪的国王是邪恶的”：

$$\forall x \ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$

- **全称量词实例化**：用**基本项**（没有变量的项）置换全称量化的变量来推断出下列语句：

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$

$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$

$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$$

$$\ldots$$

9.1 命题推断与一阶推断

全称量词实例化

- 使用**置换**来形式化地写出全称量词实例化的推断规则。用SUBST(θ, α) 表示对语句 α 应用置换 θ 后的语句。
- 对于任意变量 v 和基本项 g，规则写作

$$\frac{\forall v\ \alpha}{\text{SUBST}(\{v/g\},\alpha)}$$

上页中实例化的三条语句就是分别应用置换{x/John}、{x/Richard}和{x/Father(John)}得到的。

9.1 命题推断与一阶推断

存在量词实例化

- 存在量词实例化用一个**新的常量符号**替换存在量化的变量。
- 形式化描述：对于任意语句α、变量 v 和未在知识库其他地方出现的常量符号 k，

$$\frac{\exists v\ \alpha}{\text{SUBST}(\{v/k\},\alpha)}$$

- 例如：由语句$\exists x\ Crown(x) \wedge OnHead(x, John)$可以推断出：$Crown(C_1) \wedge OnHead(C_1, John)$
- C_1未在知识库的其他地方出现，用于命名存在语句表明存在的满足条件的对象。这个新的名称C_1被称为**Skolem常量**。
- 存在量词实例化只需要使用一次，添加了实例化的语句后，就可以丢掉存在量化的语句。

9.1 命题推断与一阶推断

约简为命题推断

- 任意一阶知识库都可以转换为命题知识库。存在量化语句能够用一个实例代替；全称量化语句可以用所有可能实例的集合代替。对于知识库：

$$\forall x \ \ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

- 若对象仅有 John 和 Richard。用所有可能的置换，{x/John}和{x/Richard}，对第一条语句应用全称量词实例化，即得到：

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$

- 命题推断与一阶推断
- 合一与一阶推断
- 前向链接
- 反向链接
- 归结

9.2 合一与一阶推断

一阶推理规则

- 一阶逻辑命题化的方法生成了不必要的全称量化语句的实例。
- 希望有一个方法有下面这样的推断过程：“给定贪婪的国王都是邪恶的这条规则，找出某个 x 使得 x 为国王且 x 是贪婪的，进而推断出这个 x 是邪恶的”。
- 更一般地说，如果存在某个置换 θ 使得每个蕴涵式前提的合取子句与知识库中的语句完全相同，那么就在应用 θ 后，断言蕴涵式的结论。
- 示例中，假设知识库中包括John是国王($King(John)$)以及所有人都是贪婪的($\forall y\ Greedy(y)$)这两条规则，那么置换θ={x/John}就能得到$John$是国王且是贪婪的这一前提。

9.2 合一与一阶推断

一般化肯定前件

- 这种可以表述为一条单独的推断规则的推断过程被称为一般化肯定前件。
- 形式化来说，对于原子语句p_i, p_i'和q，存在置换 θ 使得对所有 i 有 $SUBST(\theta, p_i') = SUBST(\theta, p_i)$，有

$$\frac{p_1', p_2', \cdots, p_n', (p_1 \wedge p_2 \wedge \cdots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

- 在示例中，以上各项分别对应

p_1'是$King(John)$	p_1是$King(x)$
p_2'是$Greedy(y)$	p_2是$Greedy(x)$
θ是$\{x/John, y/John\}$	q是$Evil(x)$
SUBST(θ, q)是$Evil(John)$	

置换

- 置换
  - 置换是一个形如{$v_1/t_1,\ldots v_n/t_n$}的有限集，其中每个v_i是变量，t_i是不同于v_i的项（常量、变量或函数）($v_i \neq t_i$)。当$i \neq j$时，$v_i \neq v_j$。
  - 无元素组成的置换称为空置换，记为ε；

- 例子：
  - {x/a, y/w, z/f(s)}, {x/g(x)}是置换；
  - {x/x}, {f(x)/y}不是置换；

实例的定义

- 置换的结果称为**实例**；
- 定义：令θ={v_1/t_1,…v_n/t_n}是一个置换，E是一个表达式，则**Eθ**是一个同时用项t_i代替E中变量v_i所得到的表达式(1≤i≤n)。 Eθ称为**E的实例**。

- 例子：
  - E=P(x, y, z), θ={x/a, y/f(b), z/c}
    - Eθ=P(a, f(b), c)
  - E=P(x, y, z), θ={x/y, y/z}
    - Eθ=P(y, z, z). Eθ≠P(z, z, z).

置换的复合

- 例子：
  - E=P(x, y, z)
  - θ={x/a, y/f(z), z/w}
  - Eθ=P(a, f(z), w)

  - λ={z/t, w/g(b)}
  - Eθλ=P(a, f(t), g(b))

  - θλ={x/a, y/f(t), z/g(b)}

9.2 合一与一阶推断

提升

- 一般化肯定前件是肯定前件的提升版——它将肯定前件从基本（无变量的）命题逻辑提升到一阶逻辑。
- 类似地，在本章中还将介绍前向链接、反向链接和归结算法的提升版。
- 提升版推断规则相比于命题化的重要优势在于，提升版推断规则只需必要的置换就可以进行特定的推断。

9.2 合一与一阶推断 - 合一

合一的定义

- 提升版的推断规则需要找出使不同的逻辑表达式相同的置换。这一过程被称作**合一**。合一算法 $UNIFY$ 接收两条语句作为输入，如果存在置换，则为它们返回一个**合一子**（即这个置换）：

$$\text{UNIFY}(p,q)=\theta \text{ 其中 } \text{SUBST}(\theta,p)=\text{SUBST}(\theta,q)$$

- 假设有查询 AskVar(Knows(John, x))：约翰认识谁？这条查询的答案可以通过找出知识库中所有与 Knows(John, x) 合一的语句来找到。当变量名重复时，需要进行**标准化分离**：

$$\text{UNIFY}(Knows(John,x),Knows(John,Jane))=\{x/Jane\}$$

$$\text{UNIFY}(Knows(John,x),Knows(y,Bill))=\{x/Bill,y/John\}$$

$$\text{UNIFY}(Knows(John,x),Knows(y,Mother(y)))=\{y/John,x/Mother(John)\}$$

$$\text{UNIFY}(Knows(John,x),Knows(x,Elizabeth))=failure$$

合一的两条语句中变量名重复

9.2 合一与一阶推断 - 合一

- **最一般合一子**
  - 使合一算法 UNIFY 的两个参数相同的合一子并不唯一。例如，$UNIFY(Knows(John, x), Knows(y, z))$可能返回$\{y/John, x/z\}$或者$\{x/John, y/John, z/John\}$。
  - $\{y/John, x/z\}$相比后者更一般化，对变量的限制更少。
  - **最一般合一子**：如果对于$UNIFY$可能返回的每一个合一子θ，都存在一个置换λ，使得$\theta = \gamma \circ \lambda$（即用$\gamma$置换后再用$\lambda$置换的结果与用$\theta$置换相同），则称$\gamma$为最一般合一子。

合一的定义

- $E_1\theta=E_2\theta$
- 定义：如果$E_1\theta=\ldots=E_n\theta$，则称置换θ为$\{E_1, \ldots, E_n\}$的合一子
- 定义：如果对$\{E_1, \ldots, E_n\}$存在这样的合一子，则称集合$\{E_1, \ldots, E_n\}$是可合一的。

- 例：
  - E={P(a,y), p(x, f(b))}, θ={x/a, y/f(b)}.
  - E={P(a,b), p(x, f(b))}

合一的定义

- 合一子不一定唯一
  - E={P(a, y), P(x, f(b))}
  - θ_1={x/a, y/f(b)}　　（唯一）
  - E={P(x, y), P(x, f(b))}
  - θ_1={x/a, y/f(b)}　　（不唯一）
  - θ_2={x/b, y/f(b)}

9.2 合一与一阶推断 - 合一

合一的定义

- 提升版的推断规则需要找出使不同的逻辑表达式相同的置换。这一过程被称作**合一**。合一算法 $UNIFY$ 接收两条语句作为输入，如果存在置换，则为它们返回一个**合一子**（即这个置换）：

$$\text{UNIFY}(p,q)=\theta \text{ 其中 } \text{SUBST}(\theta,p)=\text{SUBST}(\theta,q)$$

- 假设有查询 AskVar(Knows(John, x))：约翰认识谁？这条查询的答案可以通过找出知识库中所有与 Knows(John, x) 合一的语句来找到。当变量名重复时，需要进行**标准化分离**：

$$\text{UNIFY}(Knows(John,x),Knows(John,Jane))=\{x/Jane\}$$

$$\text{UNIFY}(Knows(John,x),Knows(y,Bill))=\{x/Bill,y/John\}$$

$$\text{UNIFY}(Knows(John,x),Knows(y,Mother(y)))=\{y/John,x/Mother(John)\}$$

$$\text{UNIFY}(Knows(John,x),Knows(x_{17},Elizabeth))=\{x/Elizabeth,x_{17}/John\}$$

- 命题推断与一阶推断
- 合一与一阶推断
- 前向链接
- 反向链接
- 归结

9.3 前向链接

一阶逻辑的前向链接算法

- 在第 7 章（命题逻辑）中展示了用于命题确定子句知识库的前向链接算法。本节拓展这个概念以涵盖一阶确定子句。
- **前向链接算法**：从知识库中的原子语句出发，在前向推理中应用假言推理规则，增加新的原子语句，直到不能进行任何推理。
- 有的逻辑语句无法被表述为确定子句，因此也无法用这种方法处理。但形如Antecedent ⇒ Consequent （前件蕴含后件）的规则足以涵盖许多有趣的真实世界系统。

9.3 前向链接

一阶确定子句

- 一阶确定子句是文字的析取式，其中必须有且仅有一个正文字。
- 这意味着确定子句要么是原子的，要么是前件为正文字的合取、后件为单个正文字的蕴涵式。
- 一阶确定子句中，存在量词不能使用，全称量词被隐式地包含在变量中。典型的一阶逻辑确定子句如下：

$$King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$

- 原子语句King(x) 和 Greedy(y)也可以看作确定子句，如Greedy(y)被解释为“每个人都是贪婪的”。

9.3 前向链接

一阶确定子句示例

- 法律规定，美国人将武器出售给敌对国家是犯罪行为。诺诺（Nono）国是美国的敌人，它拥有一些导弹，所有导弹都是韦斯特（West）上校出售给它的，而韦斯特上校是美国人。

  将证明West是罪犯

- 用一阶确定字句来表示上述事实：
  - 美国人将武器出售给敌对国家是犯罪行为：

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

  - 诺诺国拥有一些导弹：

$$Owns(Nono, M_1)$$

$$Missile(M_1)$$

9.3 前向链接

一阶确定子句示例

- 所有导弹都是韦斯特上校出售给它的：

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

- 导弹是武器：

$$Missile(x) \Rightarrow Weapon(x)$$

- 美国的敌人是“敌对的”：

$$Enemy(x, America) \Rightarrow Hostile(x)$$

- 韦斯特上校是美国人：

$$American(West)$$

- 诺诺国是美国的敌人：

$$Enemy(Nono, America)$$

9.3 前向链接

简单的前向链接算法

- 前向链接的思想是，从已知事实开始，触发所有前提被满足的规则，将结论添加到已知事实中。这一过程不断重复，直到查询得到回答（假设只需要一个回答）或没有新的事实被添加。
- 如果一个事实只是某个已知事实的**重命名**，也就是只有变量名字不同的语句，那它就不是一个“新”事实。
- 例如，Likes(x, IceCream) 与 Likes(y, IceCream) 互为重命名。它们都意味着相同的事情：“每个人都喜欢冰激凌”。

9.3 前向链接

简单的前向链接算法

一个简单的前向链接算法FOL-FC-Ask如下：

function FOL-FC-ASK(KB, α) **returns** 一个置换或false
  inputs: KB, 知识库，一个一阶确定子句集
    α, 查询，一个原子语句

  while true **do**
    new ← {} //每次迭代推断出的新语句集
    for each *rule* **in** KB **do**
      $(p_1 \wedge \cdots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(*rule*)
      for each θ 使得对于某些KB中的$p'_1, \cdots, p'_n$有SUBST $(\theta, p_1 \wedge \cdots \wedge p_n) =$ SUBST $(\theta, p'_1 \wedge \cdots \wedge p'_n)$
        $q' \leftarrow$ SUBST(θ, q)
        if q' 不能与已经在KB或*new*中的语句合一 **then**
          将q'添加到*new*
          $\phi \leftarrow$ UNIFY(q', α)
          if ϕ不为*failure* **then return** ϕ
    if *new* = {} **then return** false
    将*new*添加到KB

图 9-3　一个概念上很直观，但低效的前向链接算法。每次迭代，它都将那些用一步就可以从已经在 KB 中的蕴涵语句和原子语句推断出的原子语句添加到 KB 中。函数 STANDARDIZE-VARIABLES 用先前未使用过的变量替换其所有的参数

9.3 前向链接

- 简单的前向链接算法

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

$$Owns(Nono, M_1) \qquad Missile(M_1)$$

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

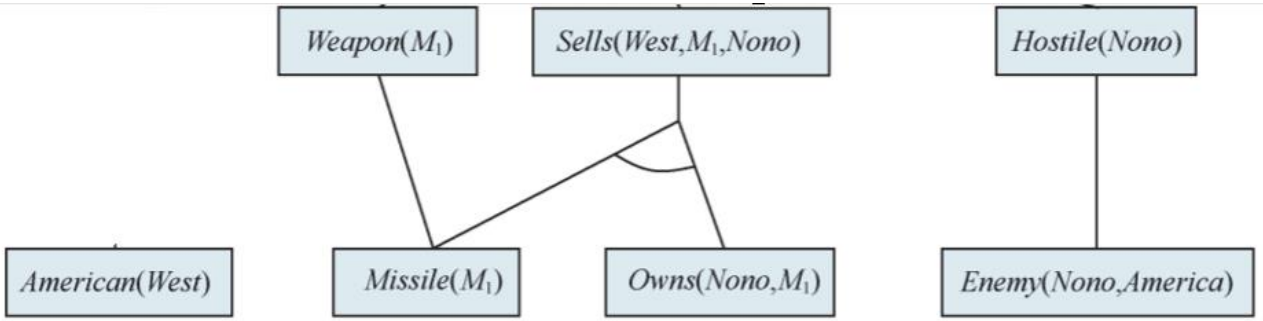
$$Missile(x) \Rightarrow Weapon(x)$$

$$Enemy(x, America) \Rightarrow Hostile(x)$$

$$American(West) \qquad Enemy(Nono, America)$$

9.3 前向链接

- 简单的前向链接算法



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

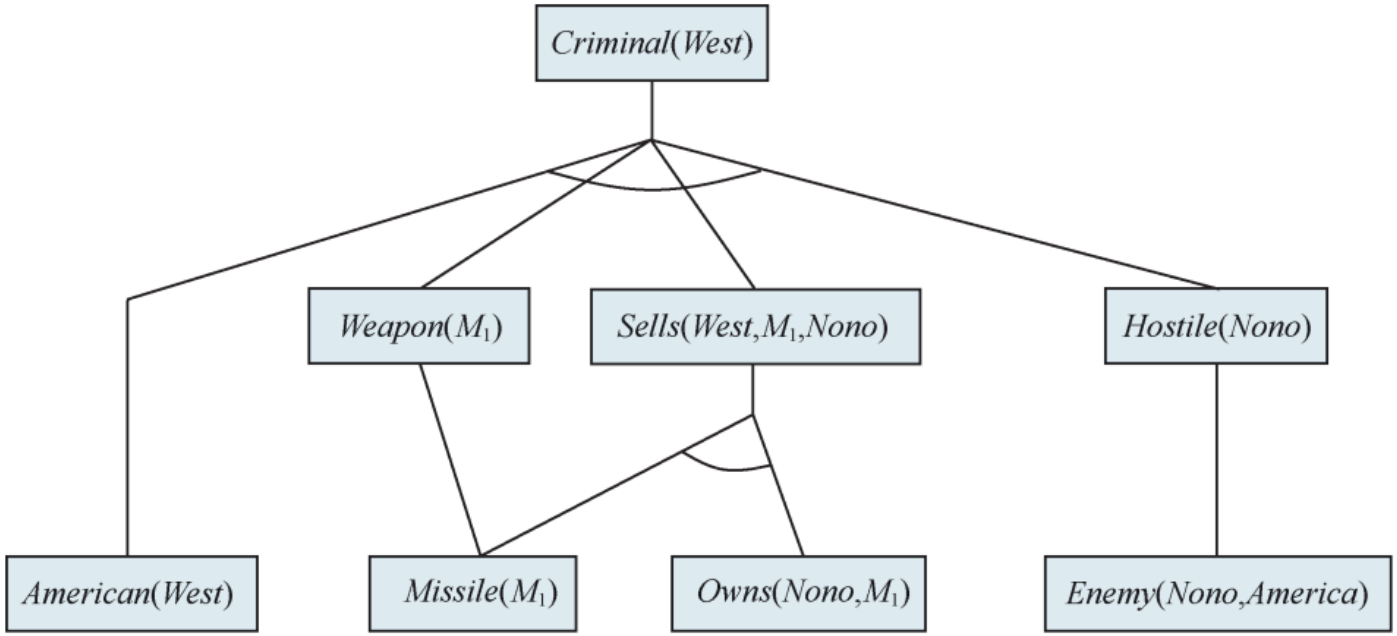
$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

9.3 前向链接

- 简单的前向链接算法



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

9.3 前向链接

简单的前向链接算法

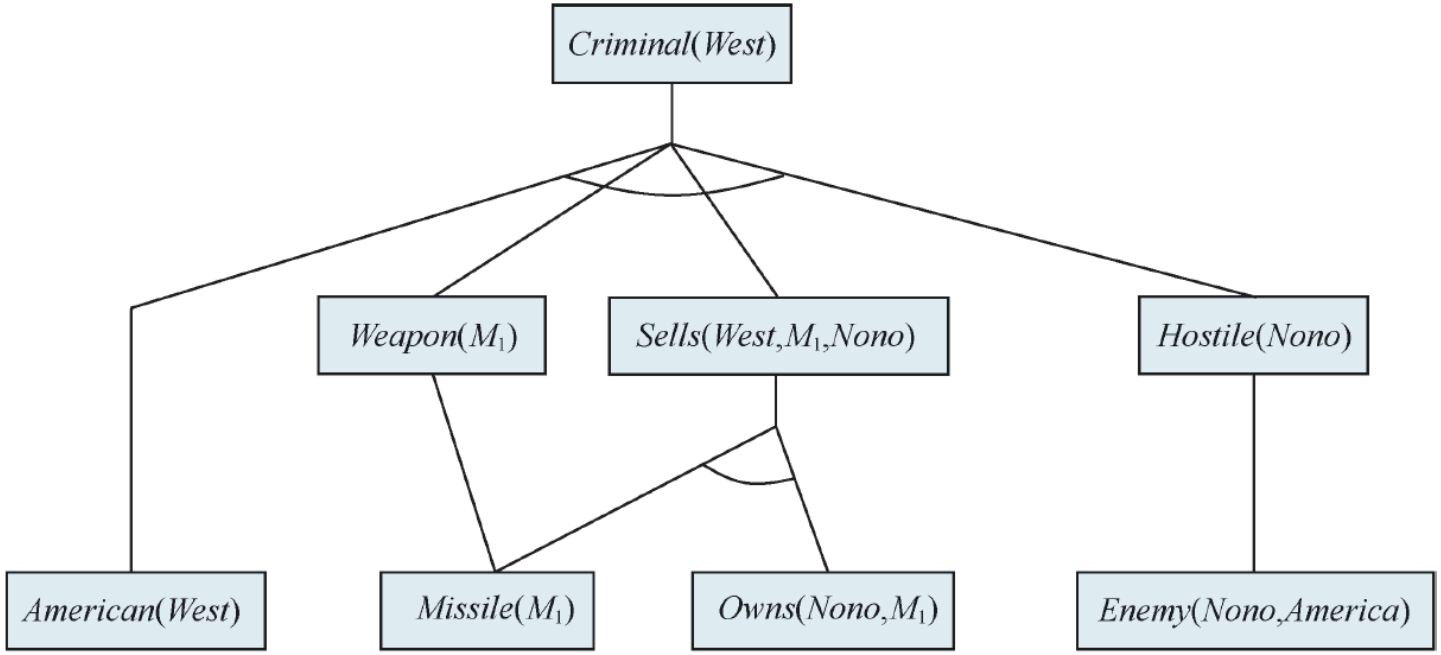


图 9-4　前向链接算法为犯罪问题生成的证明树。最早的事实出现在最下层，第一次迭代推断得到的事实在中间层，而第二次迭代推断得出的事实在最顶层

9.3 前向链接 - 高效前向链接

FOL-FC-Ask算法的特点

- FOL-FC-Ask是**可靠的**，每个推断都是对一般化肯定前件的应用，而一般化肯定前件是可靠的。
- 它对于确定子句知识库是**完备的**，也就是说，它能够对所有答案蕴涵在确定子句知识库中的查询做出回答。
- 但是，该算法是**低效的**：
  - 首先，算法的内层循环试图对知识库中的每一条规则和每一条事实进行匹配。
  - 其次，算法每次迭代都检查所有规则，尽管知识库只有少量更新。
  - 最后，算法会生成许多与目标无关的事实。

9.3 前向链接 - 高效前向链接

将规则与已知事实进行匹配

- 首先解决第一个问题，即高效地将规则的前提与知识库中的事实进行匹配。
- 考虑以下规则：

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

- 在常量时间内可以找出诺诺国拥有的所有对象，并检查它们是否是导弹；同样地，也可以先找出所有导弹，再检查它们是否被诺诺国所拥有。
- 当诺诺国拥有的对象数量远大于导弹数量时，显然后者是更好的操作。

9.3 前向链接 - 高效前向链接

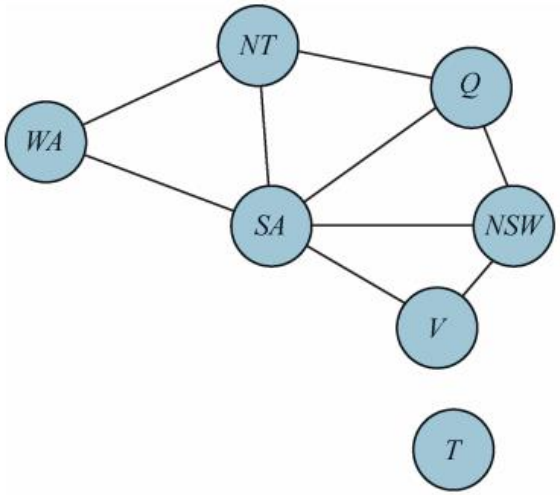
合取子句排序

- 上页所示的例子就是**合取子句排序**问题：求解规则前提的合取子句的排序，使总代价最小。
- 找出最优的排序是NP难问题，但可以用很好的启发式算法求解。
- **最小剩余值（MRV）启发式**：曾在第6章求解CSP时使用过。也称“最受约束变量”或“失败优先”启发式。
  - MRV每次选择“合法”取值最少的变量，例如，如果导弹的数量比诺诺国拥有的对象数量少，就对合取子句进行排序先查找导弹。
  - 相比随机的或静态的排序，MRV启发式通常性能更好，有时要好到1000倍以上

9.3 前向链接 - 高效前向链接

模式匹配与约束满足的联系

- 将规则和事实匹配的问题（即**模式匹配**问题）与约束满足的联系十分紧密：每个合取子句看作一条对它所包含的变量的约束。
- 拓展这种想法，就能将所有有限域 CSP 表示为单个确定子句和相关的基本事实。



（a）

$$Diff(WA, NT) \wedge Diff(WA, SA) \wedge$$
$$Diff(NT, Q) \wedge Diff(NT, SA) \wedge$$
$$Diff(Q, NSW) \wedge Diff(Q, SA) \wedge$$
$$Diff(NSW, V) \wedge Diff(NSW, SA) \wedge$$
$$Diff(V, SA) \Rightarrow Colorable()$$
$$Diff(\text{red}, \text{blue}) \quad Diff(\text{red}, \text{green})$$
$$Diff(\text{green}, \text{red}) \quad Diff(\text{green}, \text{blue})$$
$$Diff(\text{blue}, \text{red}) \quad Diff(\text{blue}, \text{green})$$

（b）

图 9-5 （a）用于为澳大利亚地图着色的约束图。（b）用单个确定子句表示的地图着色 CSP。每个地图区域都用变量表示，变量的值可以为常量 red、green、blue（使用 *Diff* 声明）

9.3 前向链接 - 高效前向链接

模式匹配与约束满足的联系

- 因为将确定子句与事实集进行匹配的问题可以等价于CSP问题，因此可以判断它是 NP 困难问题，但仍然有一些方法能简化问题的解决：
  - 真实世界知识库的大部分规则是简洁的
  - 考虑使匹配变得高效的规则的子类别，如树状结构的CSP
  - 试着消除前向链接算法中冗余的规则匹配尝试（下述）

9.3 前向链接 - 高效前向链接

增量前向链接

- 所有在第 t 次迭代中推断出的新事实必然是从至少一个在第 t-1 次迭代中推断出的新事实推得的，因为所有不需要来自第 t-1 次迭代的新事实的推断，肯定在第 t-1 次迭代时就已经得出了。
- **增量前向链接算法**：在第 t 次迭代时，仅检查前提含有合取子句 p_i 的规则，其中 p_i 能够与在第 t-1 次迭代新产生的事实 p_i' 合一。规则匹配步骤随后固定 p_i 来与 p_i' 合一，但允许规则的其他合取子句与任意先前迭代产生的事实匹配。
- 这个算法在每次迭代中生成的事实与图 9-3 所示的算法完全一致，但高效得多。

9.3 前向链接 - 高效前向链接

不相关事实

前向链接允许所有基于已知事实的推断，即使它们与目标并不相关。存在一些方法可以解决这一问题：

- 使用反向链接（下节将讨论）
- 将前向链接限制到特意挑选的规则子集上（参考命题逻辑的前向链接算法）
- **魔法集**：使用目标信息重写规则集，以便在前向推理中只考虑相关的变量绑定。具体来说，引入一个魔法谓词Magic仅对目标相关的对象为真，并将Magic加入规则前提的合取中。

- 命题推断与一阶推断
- 合一与一阶推断
- 前向链接
- 反向链接
- 归结

9.3 反向链接

反向链接算法

- 反向链接算法从目标开始反向推导链接规则，以找到支持证明的已知事实。
- 如果知识库含有形如 lhs ⇒ goal 的规则，就能证得 FOL-BC-Ask(KB, goal)，其中 lhs（left hand side）是合取子句列表。
- 反向链接是一种与或搜索：
  - FOL-BC-OR （或的部分）抓取所有可能与目标合一的子句，将子句中的变量标准化为全新变量。
  - 如果子句的 rhs （right hand side）确实能够与目标合一，就使用 FOL-BC-And （与的部分）证明 lhs 中的所有合取子句。

9.3 反向链接

反向链接伪代码

```
function FOL-BC-Ask(KB, query) returns 置换生成器
  return FOL-BC-Or(KB, query, {})

function FOL-BC-Or (KB, goal, θ) returns 一个置换
  for each ruls in Fetch-Rules-For-Goal(KB, goal) do
    (lhs ⇒ rhs) ← Standardize-Variables(rule)
    for each θ′ in FOL-BC-And(KB, lhs, Unify(rhs, goal, θ)) do
      yield θ′

function FOL-BC-And(KB, goals, θ) returns 一个置换
  if θ = failure then return
  else if Length(goals) = 0 then yield θ
  else
    rst, rest ← First(goals), Rest(goals)
    for each θ′ in FOL-BC-Or(KB, Subst(θ,  rst), θ) do
      for each θ″ in FOL-BC-And(KB, rest, θ′) do
        yield θ″
```

图 9-6　用于一阶知识库的简单的反向链接算法

9.3 反向链接

反向链接解决示例问题

$$\boxed{Criminal(West)}$$

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

$$Owns(Nono, M_1) \qquad Missile(M_1)$$

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

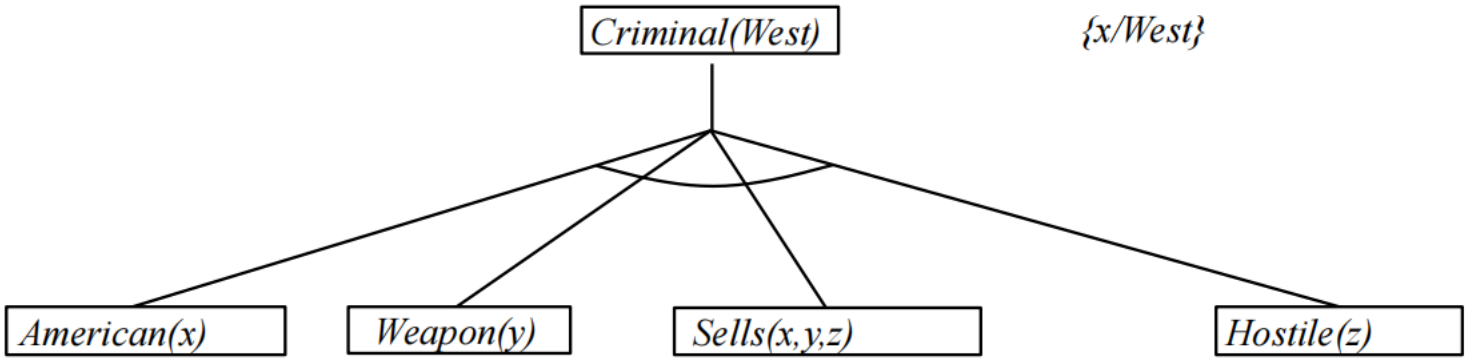
$$Missile(x) \Rightarrow Weapon(x)$$

$$Enemy(x, America) \Rightarrow Hostile(x)$$

$$American(West) \qquad Enemy(Nono, America)$$

9.3 反向链接

- 反向链接解决示例问题



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1) \qquad Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

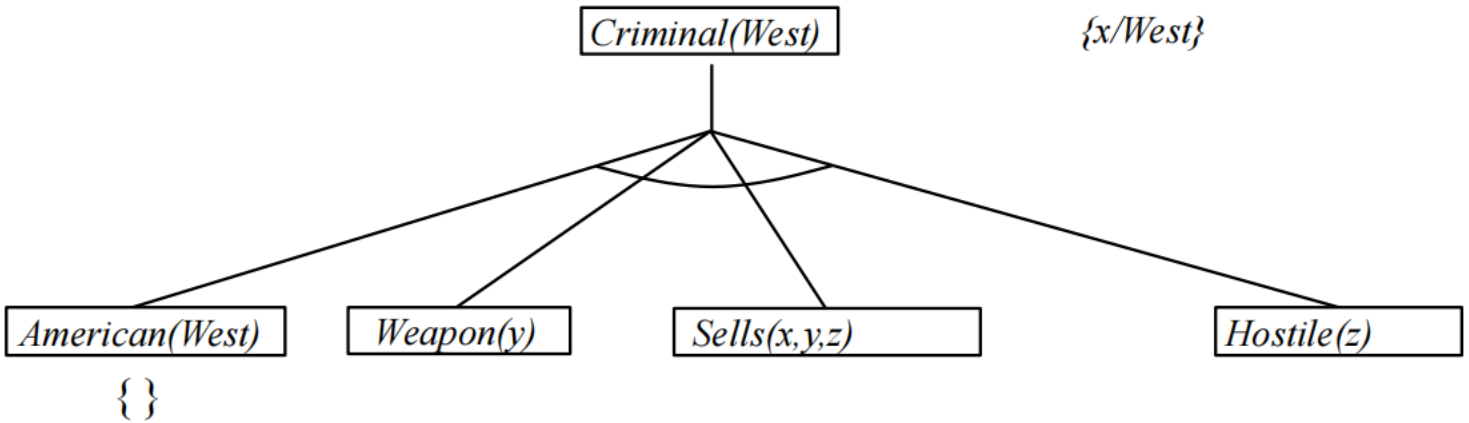
$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West) \qquad Enemy(Nono, America)$

9.3 反向链接

- 反向链接解决示例问题



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

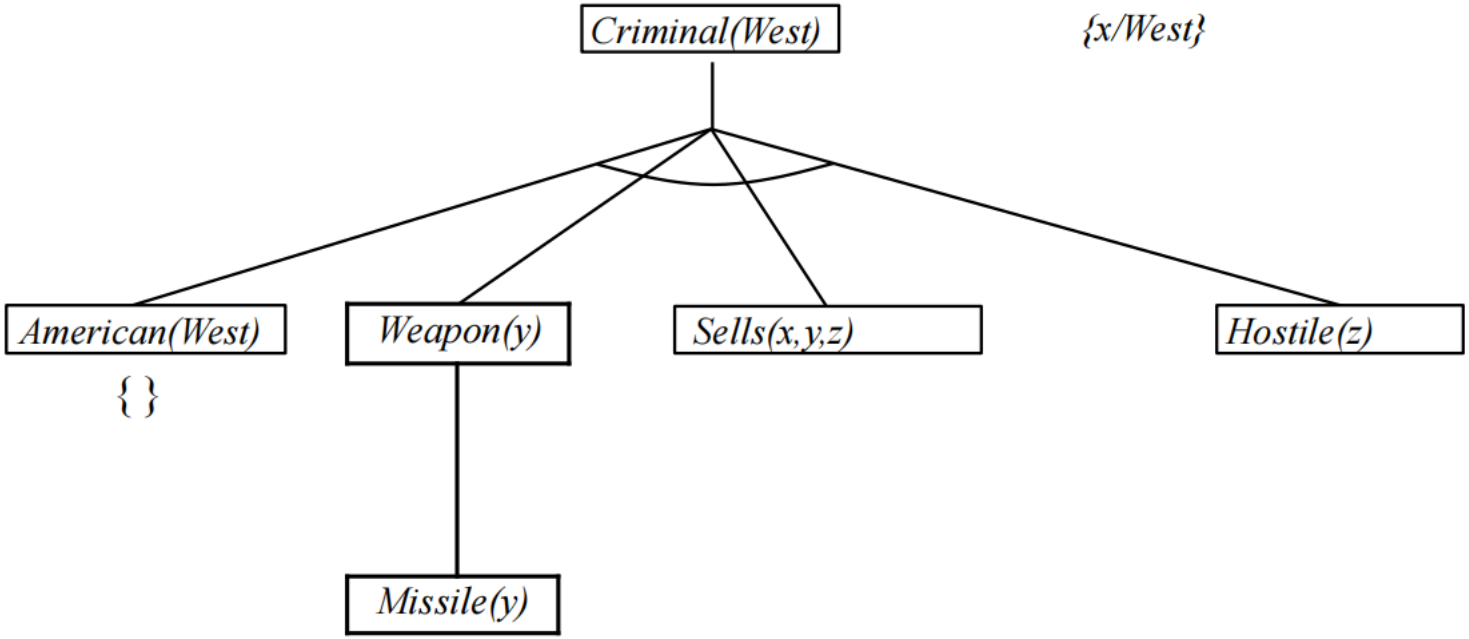
$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

9.3 反向链接

- 反向链接解决示例问题



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1) \qquad Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

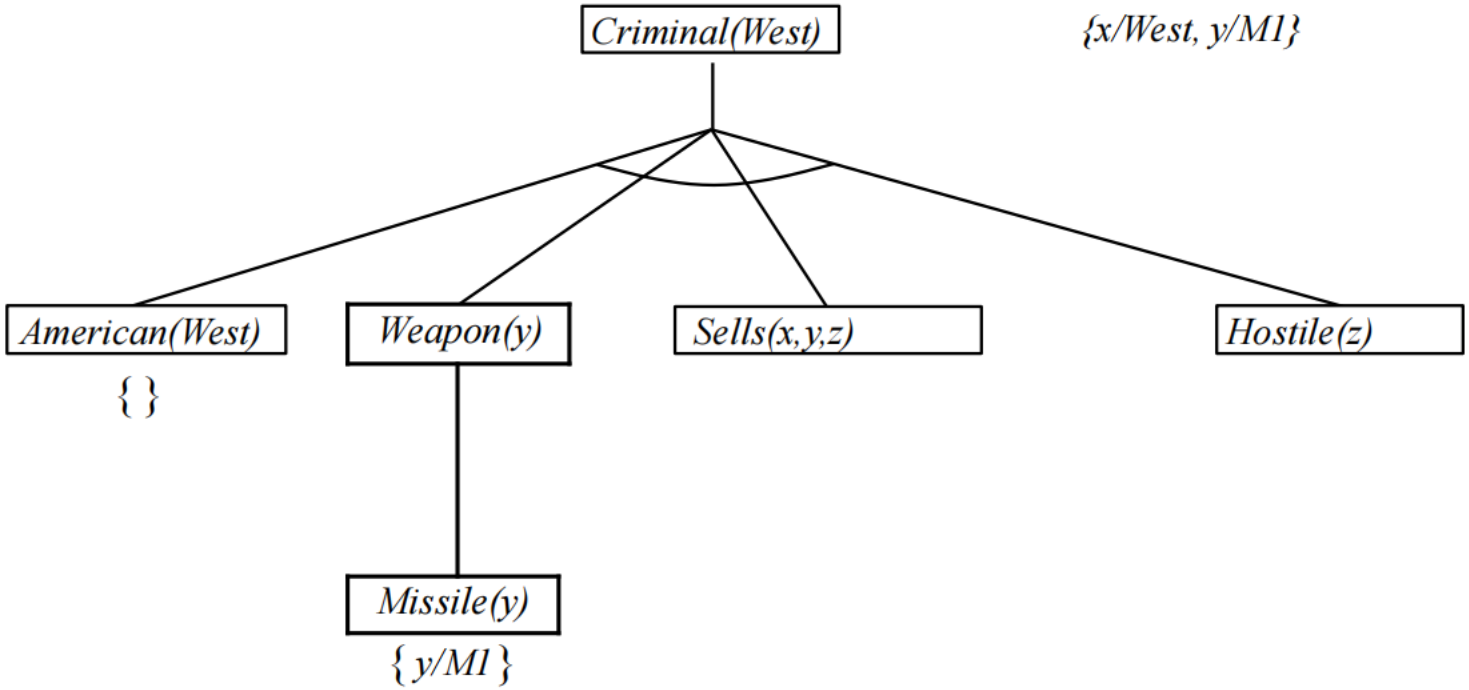
$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West) \qquad Enemy(Nono, America)$

9.3 反向链接

- 反向链接解决示例问题



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

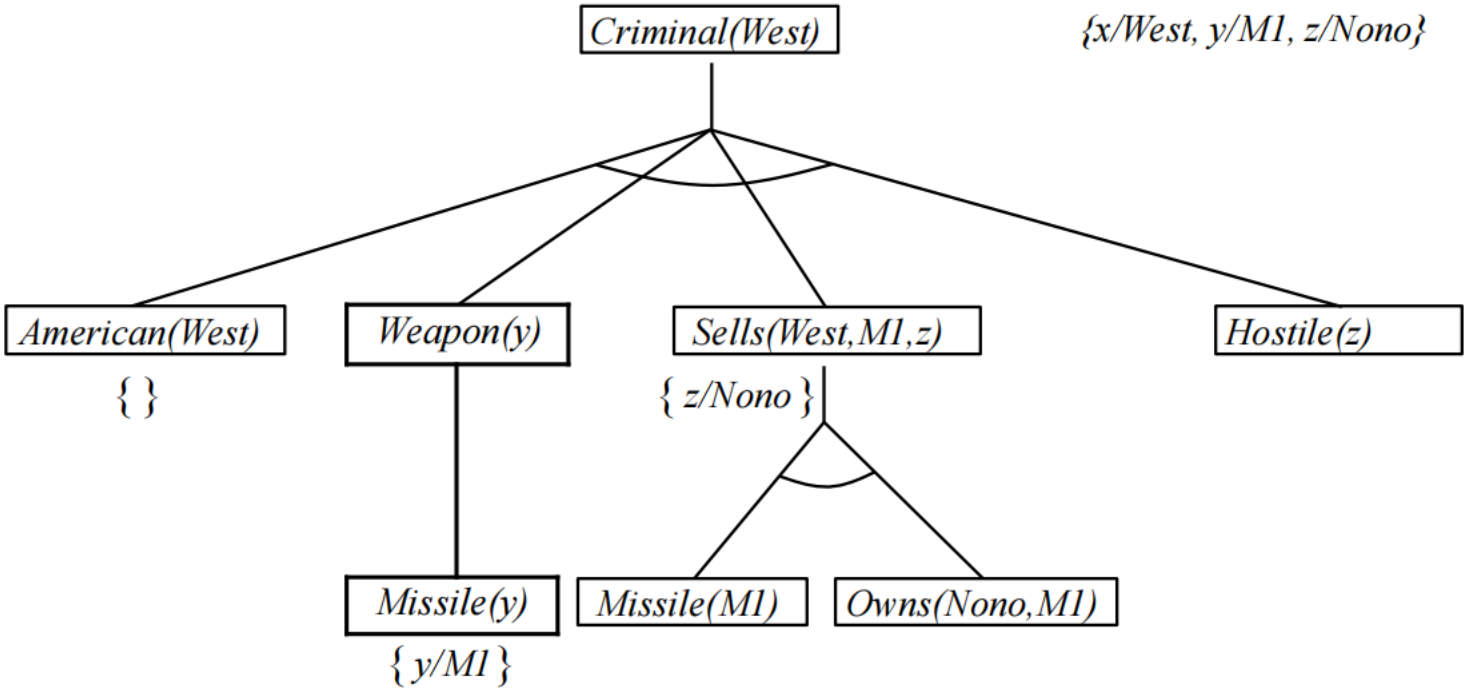
$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

9.3 反向链接

- 反向链接解决示例问题



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

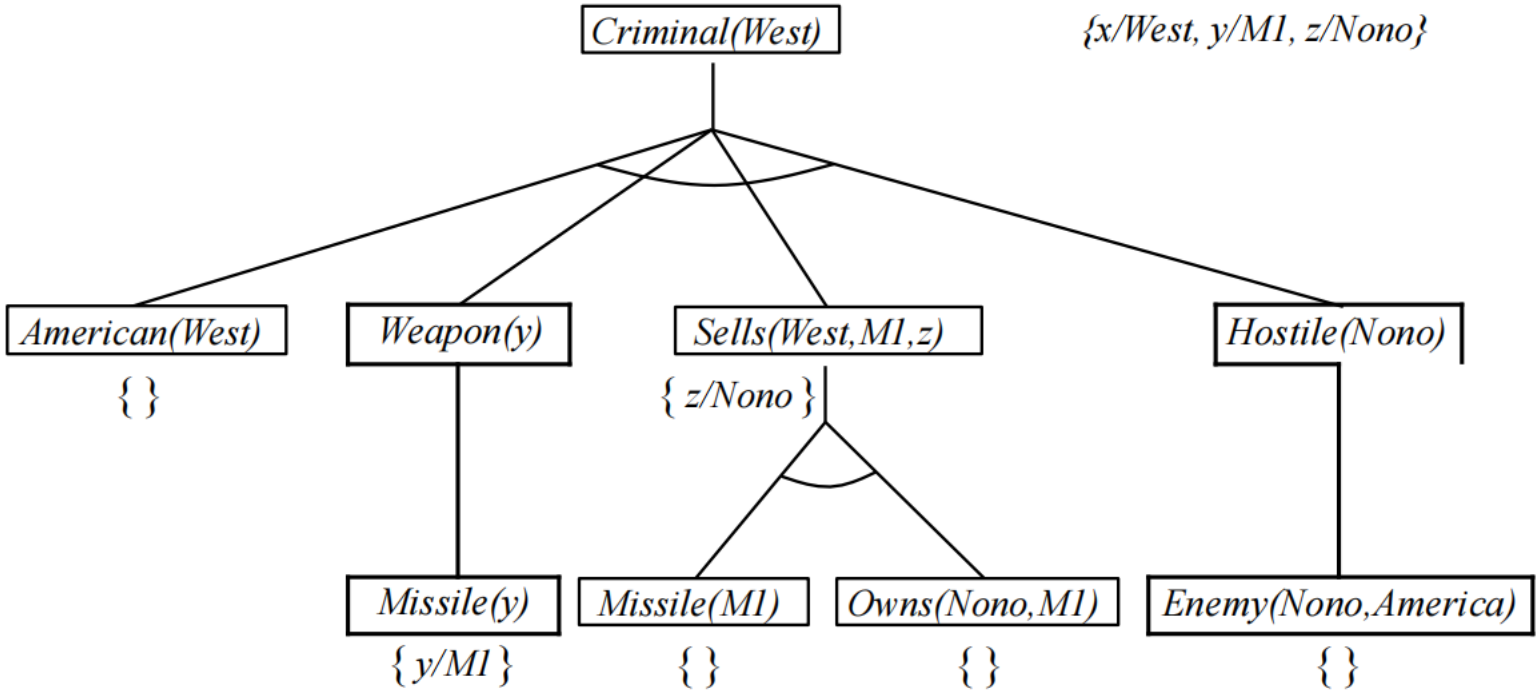
$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

9.3 反向链接

- 反向链接解决示例问题



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$

$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$ $\quad$ $Enemy(Nono, America)$

- 命题推断与一阶推断
- 合一与一阶推断
- 前向链接
- 反向链接
- 归结

9.4 归结 - 一阶逻辑的合取范式

一阶逻辑的合取范式

- 归结是唯一能够用于所有知识库而不仅是确定子句的成员。命题归结是命题逻辑的完备推断过程，现在将其拓展到一阶逻辑。
- 与命题逻辑一样，一阶逻辑的归结也要求语句必须是**合取范式**（CNF），即子句的合取式，其中每个子句是文字的析取式。文字可以包含变量，假定这些变量是全称量化的。
- 例如，语句

$$\forall x, y, z \;\; American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

转化为CNF即为

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)$$

9.4 归结 - 一阶逻辑的合取范式

合取范式转换

- 每条一阶逻辑语句都可以转换为推断上等价的 CNF 语句。特别是，CNF语句只有当原始语句不可满足时才不可满足，这是应用CNF语句的反证法证明的基础。
- 用以下示例展示CNF转换的步骤：

Everyone who loves all animals is loved by someone

$$\forall x\,[\forall y\;\; Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\;\; Loves(y,x)]$$

9.4 归结 - 一阶逻辑的合取范式

合取范式转换

- **蕴含消去**：使用 ¬P ∨ Q 取代 P ⇒ Q，以(P∧Q) ∨ (¬P∧¬Q)取代 P ⇔ Q。在例句中，需要操作两次。

$$\forall x \ \neg[\forall y \ Animal(y) \Rightarrow Loves(x,y)] \vee [\exists y \ Loves(y,x)]$$

$$\forall x \ \neg[\forall y \ \neg Animal(y) \vee Loves(x,y)] \vee [\exists y \ Loves(y,x)]$$

- **¬ 内移**：除了针对否定联结词的一般规则，还需要针对否定量词的规则，即将

$$\neg\forall x \ p \text{ 变为 } \exists x \ \neg p$$

$$\neg\exists x \ p \text{ 变为 } \forall x \ \neg p$$

从而使否定符号 ¬ 最多只作用到一个谓词上

$$\forall x \ [\exists y \ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y \ Loves(y,x)]$$

$$\forall x \ [\exists y \ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y \ Loves(y,x)]$$

$$\forall x \ [\exists y \ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y \ Loves(y,x)]$$

9.4 归结 - 一阶逻辑的合取范式

合取范式转换

- **变量标准化**：对于重复出现的变量名，更改其中一个变量的名字，从而避免在消除量词时产生歧义：

$$\forall x \ [\exists y \ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z \ Loves(z,x)]$$

- **斯科伦化**：通过消去移除存在量词的过程。这个过程类似于存在量词实例化（事实上存在量词实例化是存在量词处于最外层时的特例）。如果存在量词在全称量词的辖域内（例如$\forall y \ (\exists x \ P(x,y))$这样的形式），内层的存在量词进行实例化时应使用一个依赖于x的函数（**斯科伦函数**）：

$$\forall x \ [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

9.4 归结 - 一阶逻辑的合取范式

合取范式转换

- **全称量词消除**：斯科伦化消除了所有存在量词，此时所有剩余变量必然是全称量化的。因此消除全称量词不会损失任何信息：

$$[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

- **对∧分配∨，转化为CNF的形式**：

$$[Animal(F(x)) \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)]$$

- 从而原语句被转为了含有两个字句的CNF。对于人类来说，转化为CNF的语句非常难读，不过人类很少需要考察 CNF 语句。

Skolem范式

- 一个公式，如果量词均在全式的开头，它们的作用域延伸到整个公式的末端，则该公式叫做前束范式
- 以前束范式中消去全部存在量词所得到的公式即为Skolem标准范式

Skolem范式

- 如果用Skolem函数$f(x)$代替前束范式$\forall x \exists y \forall z(P(x) \wedge F(y,z) \wedge Q(y,z))$中的$y$，即得到Skolem标准范式：

$$\forall x \forall z(P(x) \wedge F(f(x),z) \wedge Q(f(x),z))$$

- Skolem标准型的一般形式是

$$\forall x_1 \forall x_2 \ldots \forall x_n M(x_1, x_2, \ldots, x_n)$$

其中，$M(x_1, x_2, \ldots, x_n)$是一个合取范式，称为Skolem标准型的母式。

化公式为Skolem范式的步骤

- 消去谓词公式中的蕴涵(⇒)和双条件符号(⇔)
  - 以¬A∨B代替A⇒B；以(A∧B) ∨(¬ A∧ ¬ B)代替A ⇔ B
- 减小否定符号(¬)的辖域，使否定符号“¬”最多只作用到一个谓词上
- 重新命名变元名，使所有的变元的名字均不同，并且自由变元及约束变元亦不同
- 消去存在量词。两种情况：
  - 存在量词不出现在全称量词的辖域内，此时，只要用一个新的个体常量替换该存在量词约束的变元，就可以消去存在量词
  - 存在量词位于一个或多个全称量词的辖域内，这是需要用一个Skolem函数替换存在量词而将其消去

化公式为Skolem范式的步骤

- 把全称量词全部移到公式的左边，并使每个量词的辖域包括整个量词后面公式的整个部分
- 母式化为合取范式：任何母式都可以写成由一些谓词公式和谓词公式否定的析取的有限集组成的合取

Skolem函数例子

- $(\exists x)(\forall y)(\forall z)(\exists u)(\forall v)(\exists w)P(x,y,z,u,v,w)$

$\Rightarrow(\forall y)(\forall z)(\exists u)(\forall v)(\exists w)P(a,y,z,u,v,w)$

$\Rightarrow(\forall y)(\forall z)(\forall v)(\exists w)P(a,y,z,f(y,z),v,w)$

$\Rightarrow(\forall y)(\forall z)(\forall v)P(a,y,z,f(y,z),v,g(y,z,v))$

9.4 归结 - 归结推断规则

一阶逻辑的归结推断

- 如果两个命题文字相互否定，则这两个命题文字是互补的；如果两个一阶逻辑文字中的一个能够与另一个的否定合一，则这两个一阶逻辑文字是互补的。
- 一阶子句的归结规则就是第 7 章命题归结规则的提升版：两条进行了标准化分离、没有共同变量的子句，如果它们含有互补文字则可以被归结。
- 形式化：当 l_i 能与 m_j 的否定合一（$UNIFY(l_i, \neg m_j) = \theta$），则有

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\text{SUBST}(\theta, \ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)}$$

9.4 归结 - 归结推断规则

一阶逻辑归结示例

- 假设有两个子句

  $[Animal(F(x)) \vee Loves(G(x),x)]$ 和 $[\neg Loves(u,v) \vee \neg Kills(u,v)]$

- 其中，$Loves(G(x),x)$和$\neg Loves(u,v)$是互补的，因为前者可以通过置换$\theta = \{u/G(x), v/x\}$来与后者的否定合一。因此，这两个子句可以生成归结式子句

  $$[Animal(F(x)) \vee \neg Kills(G(x),x)]$$

  这个规则叫作**二元归结**规则，因为它刚好归结两个文字。

- 归结规则通过反证法来证明结论：假设目标不成立（在知识库中引入目标的否定），通过归结推出矛盾（空子句），从而反证目标正确。下面用两个范例来说明归结如何完成一阶逻辑的证明。

9.4 归结 - 证明范例

犯罪示例的归结证明

- 第一个范例考虑9.3节中引入的犯罪示例，将该实例中的所有语句首先转为CNF的形式，转化后的语句如下：

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x,y,z) \vee \neg Hostile(z) \vee Criminal(x)$$

$$\neg Missile(x) \vee \neg Owns(Nono,x) \vee Sells(West,x,Nono)$$

$$\neg Enemy(x,America) \vee Hostile(x)$$

$$\neg Missile(x) \vee Weapon(x)$$

$$Owns(Nono,M_1) \qquad Missile(M_1)$$

$$American(West) \qquad Enemy(Nono,America)$$

9.4 归结 - 证明范例

- 犯罪示例的归结证明
  - 在进行证明时，包括目标的否定$\neg Criminal(West)$，并通过知识库中的子句进行归结，最终获得空子句，从而完成证明。

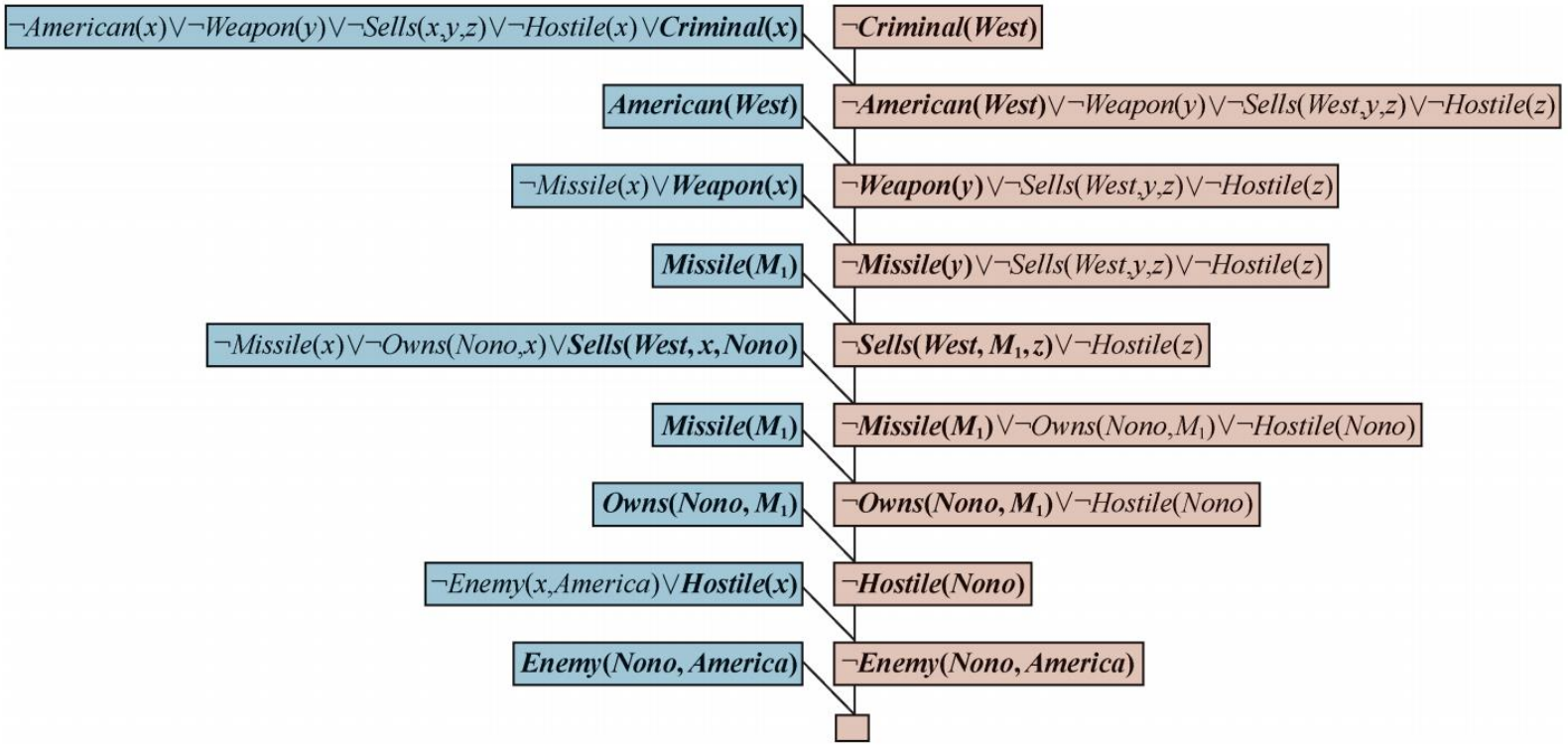


图 9-10　归结证明韦斯特有罪。每个归结步骤中，合一文字用加粗字体表示，带有正文字的子句用蓝底表示

9.4 归结 - 证明范例

好奇害死猫示例

- 第二个范例使用斯科伦化，并涉及非确定子句。这会导致更为复杂的证明结构。其自然语言描述和证明如下：

Everyone who loves all animals is loved by someone.（每个爱所有动物的人都被一些人所爱。）

Anyone who kills an animal is loved by no one.（任何害死动物的人都不被人所爱。）

Jack loves all animals.（杰克爱所有动物，）

Either Jack or Curiosity killed the cat, who is named Tuna.（要么是杰克要么是好奇心害死了那只猫，猫的名字叫 Tuna。）

Did Curiosity kill the cat?（是好奇心害死了那只猫吗？）

自然语言证明：假设好奇心没有害死 Tuna。我们知道要么是杰克要么是好奇心做了这件事，因此一定是杰克干的。现在，Tuna 是一只猫，而猫是动物，因此 Tuna 是动物。因为任何害死动物的人都不被人所爱，我们就知道没人爱杰克。但是，杰克爱所有动物；至此有人爱他；至此我们得到了一个矛盾。因此，好奇心害死了那只猫。

9.4 归结 - 证明范例

好奇害死猫示例的归结证明

- 首先将自然语言描述表示为一阶逻辑语句，同样地，在知识库中包括目标的否定：

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

B. $\forall x\ [\exists z\ Animal(z) \wedge Kills(x,z)] \Rightarrow [\forall y\ \neg Loves(y,x)]$

C. $\forall x\ \ Animal(x) \Rightarrow Loves(Jack,x)$

D. $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$

E. $Cat(Tuna)$

F. $\forall x\ \ Cat(x) \Rightarrow Animal(x)$

$\neg$G. $\neg Kills(Curiosity,Tuna)$

9.4 归结 - 证明范例

- 好奇害死猫示例的归结证明
  - 将语句全部转换为CNF范式：

A1. $Animal(F(x)) \vee Loves(G(x), x)$

A2. $\neg Loves(x, F(x)) \vee Loves(G(x), x)$

B. $\neg Loves(y, x) \vee \neg Animal(z) \vee \neg Kills(x, z)$

C. $\neg Animal(x) \vee Loves(Jack, x)$

D. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

E. $Cat(Tuna)$

F. $\neg Cat(x) \vee Animal(x)$

$\neg$G. $\neg Kills(Curiosity, Tuna)$

9.4 归结 - 证明范例

好奇害死猫示例的归结证明

- 通过不断归结子句，最终获得空子句来完成证明：

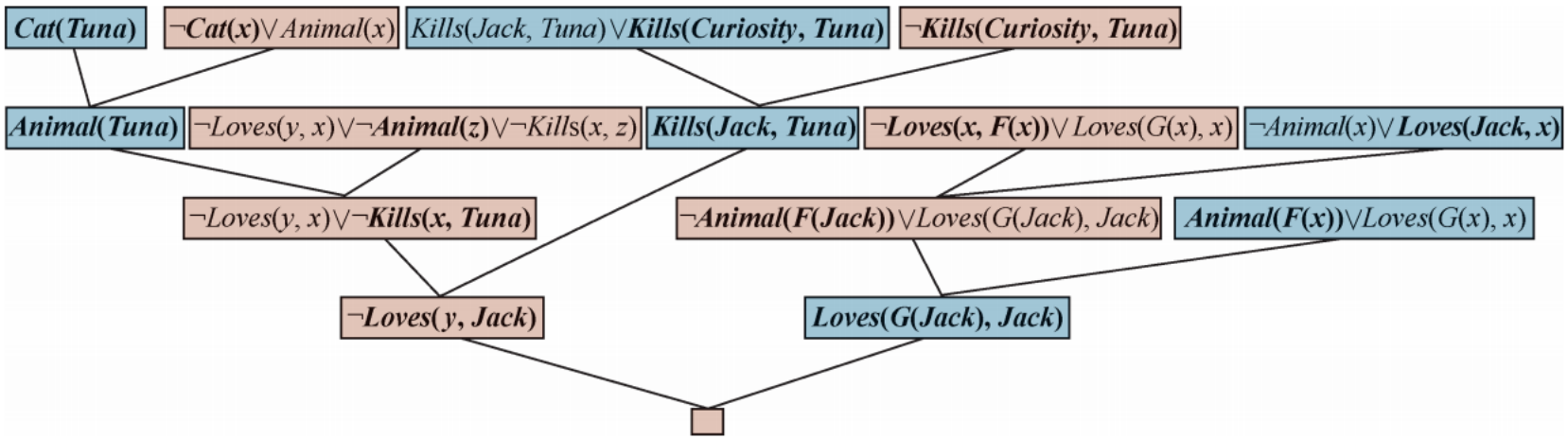


图 9-11　好奇心害死猫的归结证明。注意，在推导子句 *Loves*(*G*(*Jack*), *Jack*) 时使用了因子分解。还要注意，在右上角，合一 *Loves*(*x*, *F*(*x*))和 *Loves*(*Jack*, *x*)只有在变量标准化分离后才可以进行

9.4 归结 - 等词

等词“=”的处理：公理化等词

本章到目前为止讲述过的所有推断方法都不能在不增加额外工作的情况下处理形如 x = y 的断言。为此可以采取 3 种不同的方法。

- 第一种方法是公理化等词，也就是在知识库中写入相等关系的语句。相等是**自反的、对称的和传递的**，此外**可以在所有谓词或函数中用相等量置换相等量**。因此需要 3 类基本公理，另外每个谓词和函数都需要一条公理：

$$\forall x \quad x = x$$
$$\forall x, y \; x = y \Rightarrow y = x$$
$$\forall x, y, z \; x = y \wedge y = z \Rightarrow x = z$$

$$\forall x, y \; x = y \Rightarrow (P_1(x) \Leftrightarrow P_1(y))$$
$$\forall x, y \; x = y \Rightarrow (P_2(x) \Leftrightarrow P_2(y))$$
$$\vdots$$
$$\forall w, x, y, z \; w = y \wedge x = z \Rightarrow (F_1(w, x) = F_1(y, z))$$
$$\forall w, x, y, z \; w = y \wedge x = z \Rightarrow (F_2(w, x) = F_2(y, z))$$
$$\vdots$$

9.4 归结 - 等词

等词“=”的处理：解调

- 等词公理化会产生大量结论，其中大多数对证明没有帮助。因此第二种方法是添加推断规则而非公理。
- 最简单的规则是**解调**：
  - 如果有单元子句x=y和一个含有x的子句α，用y置换α中的x可生成新的子句。如果α中的项能够与x合一，解调就可以使用。不一定需要完全等于x。
  - 注意这种解调是有向的；给定x=y，x总是被y替换，反过来则不行。
- 例如，给定

$$Father(Father(x)) = PaternalGrandfather(x)$$

$$Birthdate(Father(Father(Bella)),1926)$$

则解调可以得到

$$Birthdate(PaternalGrandFather(Bella),1926)$$

9.4 归结 - 等词

等词“=”的处理：解调

形式化来说，对于任意项 x、y 和 z，其中 z 出现在文字 m_i 中的某处且 $UNIFY(x,z)=\theta \neq failure$，则

- **解调**：

$$\frac{x=y,\qquad m_1 \vee \cdots \vee m_n}{\text{SUB}(\text{SUBST}(\theta,x),\text{SUBST}(\theta,y),m_1 \vee \cdots \vee m_n)}$$

- **超解调**：

$$\frac{\ell_1 \vee \cdots \vee \ell_k \vee x=y,\qquad m_1 \vee \cdots \vee m_n}{\text{SUB}(\text{SUBST}(\theta,x),\text{SUBST}(\theta,y),\text{SUBST}(\theta,\ell_1 \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_n))}$$

- SUBST是对绑定表的一般置换，而SUB(x,y,m)表示在 m 中的某处用 y 替换 x
- 例子：给出$P(F(x,B),x) \vee Q(x)$和$(F(A,y)=y) \vee R(y)$，通过合一 $\theta = UNIFY(F(A,y),F(x,B)) = \{x/A, y/B\}$，可以通过超解调得到语句

$$P(B,A) \vee Q(A) \vee R(B)$$

9.4 归结 - 等词

- **等词“=”的处理：拓展合一算法**
  - 如果若干项在某种置换下可证明为相等，则它们是可合一的，其中“可证明”允许等词推理。例如，项 $1+2$ 和 $2+1$ 通常不可合一，但知道 $x+y=y+x$ 的合一算法可以用空置换合一它们。
  - 这种**等词合一**可以用针对特定公理（交换性、结合性等）设计的高效算法完成，而非通过直接用这些公理推断。

9.4 归结 - 归结策略

高效的证明策略

只要证明存在，反复运用归结推断规则总会找到一个证明。在本节中考察有助于高效找出证明的策略。

- **单元优先**：该策略对那些包含一个单文字（也称为单元子句）的语句进行归结：由于试图产生空子句，那么首先完成那些能够产生较短子句的推理可能是一个好思路。
- **支撑集**：首先尝试某些特定的优先策略对于归结很有帮助，但是一般来说减少一些潜在的归结会更有效。
  - 例如，可以坚持归结的每一步都必须涉及至少一个特殊子句集的元素，这个集合被称为支撑集。它首先确定一个称为支撑集的语句子集。归结式放到支撑集中。如果相对于整个知识库而言支撑集很小，搜索空间将会大幅度缩小。

9.4 归结 - 归结策略

高效的证明策略

- **输入归结**：在这种策略中，每次归结都是一个输入语句（来自KB或者查询）和某个其他语句的结合。（例如犯罪示例的证明就使用了输入归结，从而产生具有单条主线的结构）
- **包容**：包容法清除所有被知识库中的已有语句包容（即，比该语句更特例）的语句。包容帮助保持KB的小规模，这样才能帮助保持较小的搜索空间。
- **学习**：从经验中学习改进定理证明器。给定先前证得的定理集，训练机器学习系统来回答问题：给定前提集和证明目标，哪些证明步骤与之前成功的证明步骤类似？ DeepHOL 系统（Bansal et al., 2019）使用深度神经网络来构建目标和前提的模型来选择步骤。训练可以使用人类或计算机生成的证明作为样本，并至少需要 10 000 个证明。

本章小结

- 通过量词实例化，将一阶推断转为命题推断。
- 合一用于确定适当的变量置换，消除一阶逻辑证明的实例化步骤，使得整个过程效率更高。
- 一般化肯定前件利用合一提供了自然而且强有力的推理规则。前向链接和反向链接算法将这条规则应用于确定子句集。
- 归结能够用于所有知识库而不仅是确定子句的成员。
- 利用公理化、解调等方法处理推理过程中的 等词。

课程作业

- 假设有如下公理：
  1. $0 \leq 3$
  2. $7 \leq 9$
  3. $\forall x \;\; x \leq x$
  4. $\forall x \;\; x \leq x + 0$
  5. $\forall x \;\; x + 0 \leq x$
  6. $\forall x,y \;\; x + y \leq y + x$
  7. $\forall w,x,y,z \;\; w \leq y \wedge x \leq z \leq \Rightarrow w + x \leq y + z$
  8. $\forall x,y,z \;\; x \leq y \wedge y \leq z \Rightarrow x \leq z$
- 请分别用反向链接和前向链接证明$7 \leq 3 + 9$（注意：只用上述公理，不要用其他的数学知识），只需要给出关键步骤。

THANKS