第16章　谓词演算中的归结

16.1　合一

合式公式（ ξ_1，ξ_2，…，ξ_n）（$\lambda_1 \vee \lambda_2 \vee \cdots \vee \lambda_k$）可缩写为$\lambda_1 \vee \lambda_2 \vee \cdots \vee \lambda_k$，其中$\lambda_1$，$\lambda_2$，…，$\lambda_k$是可能包含变量$\xi_1$，$\xi_2$，…，$\xi_n$的文字。也就是说，仅仅去掉了全称量词，并假定$\lambda_i$中任何变量全称量化（后面将说明如何能首先消除任何存在的量化变量）。这种缩写形式的合式公式叫做子句。有时，用集合符号{λ_1，λ_2，…，λ_k}表达一个子句，并假定集合中的元素是析取的。

如果两个子句的文字相匹配但是互补，我们能归结它们——就像在命题演算中一样。如果一个子句中有一个文字λ（ξ）（ξ是一个变量），而另一个子句中有一个互补文字 $\neg\lambda$（τ），τ是不包含ξ的某个项，我们能把第一个子句中的所有ξ用τ代替，然后对互补文字进行命题归结以产生那两个子句的归结式。

举例：考虑两个子句P（f（y），A）∨Q（B，C）和 ¬P（x，A）∨R（x，C）∨S（A，B）。用f（y）代替第二个子句中的x产生 ¬P（f（y），A）∨R（f（y），C）∨S（A，B）。现在，两个子句中的第一个文字刚好互补，因此我们能对文字（f(y)，A）执行一次归结，产生归结式R(f(y)，C)∨S(A，B)∨Q(B，C)。

用一个被称为合一的方法计算适当的置换。合一在AI中是一个极其重要的方法。为了描述它，必须先讨论一下置换。

一个表达式项可能是变量符号、对象常量或者函数表达式，后者包含函数常量和表达式项。一个表达式的置换实例通过置换那个表达式中的变量项而得到。因此，P[x，f(y)，B]的四个置换实例是：

```
P[z，f(w)，B]
P[x，f(A)，B]
P[g(z)，f(A)，B]
P[C，f(A)，B]
```

上面第一个实例称为原始文字的字母变种（*alphabetic variant*），因为我们仅仅用另外的变量代替了P[x，f(y)，B]中出现的变量。第4个叫基例（*ground instance*），因为文字中没有一项包含变量（一个基项是一个不包含任何变量的项）。

我们能通过一组有序对$s = \{\tau_1/\xi_1, \tau_2/\xi_2, \ldots, \tau_n/\xi_n\}$来表达任何置换。$\tau_i/\xi_i$对意思是说$\tau_i$项替换在整个置换范围内的$\xi$的每次出现。而且，变量不能被一个包含相同变量的项代替。使用前面P[x，f(y)，B]的四个实例的置换是：

```
s1 = {z/x，w/y}
s2 = {A/y}
s3 = {g(z)/x，A/y}
s4 = {c/x，A/y}
```

用ωs来指称一个使用置换s的表达式ω的一个置换实例。因此，P[z，f(w)，B] = P[x，f(y)，

B]s1。两个置换s1和s2的组合用s1s2指称，它指的是这个置换通过先把s2应用到s1各项，再加上不含出现在s1中变量的所有s2对而得到，因此：

{g(x, y)/z} {A/x, B/y, C/w, D/z} = {g(A, B)/z, A/x, B/y, C/w}

可以看出，把s1和s2连续地应用到ω表达式和把s1s2应用到ω 是相同的，即：(ω s1) s2 = ω (s1s2)。也能看出，置换组合是符合结合律的。即：(s1s2) s3 = s1 (s2s3)。

举例 ω是P (x, y)，s1是{f(y) / x}，s2是{A/y}。那么

(ωs1) s2 = [p(f(y), y)]{A/y} = P(f (A), A)

和

ω(s1s2) = [P(x, y)]{f(A) / x, A/y} = P(f(A), A)

一般地讲，置换不符合交换律，即s1s2 = s2s1是不成立的。因此，改变应用置换顺序会产生差异。

例如（使用前面例子中的ω、s1和s2）

ω (s1s2) = P (f(A), A)

ω (s2s1) = [P(x, y)]{ A/y, f(y) / x } = P(f(y), A)

当一个置换s被应用到一个表达式集合{ω_i}的每一个成员时，用{ω_i}s表示置换实例集合。如果存在一个置换s，它使$\omega_1 s = \omega_2 s = \omega_3 s = \cdots$，我们说表达式集合{$\omega_i$}是可以合一的（*unifiable*）。在这种情况下，s被称为{ω_i}的一个合一式(*unifier*)，因为它的使用把集合压缩成为一个单元素集合。例如：s = {A/x, B/y}把集合{p[x, f(y), B], P[x, f(B), B]}合一产生{p[A, f(B), B]}。虽然s = {A/x, B/y}是集合{p[x, f(y), B], p[x, f(B), B]}的一个合一式，但在某种意义上它不是最简单的合一式。我们注意到确实不必用A置换x来获得合一。最一般（或最简单）的合一式（mgu），{ω_i}的g有下面的特性：如果s是产生 {ω_i}s的{ω_i}的任意合一式，那么存在一个置换s'以使{ω_i}s = {ω_i}gs'。而且，经一个最一般的合一式产生的通用实例除了字母变化外是惟一的。

有很多算法可以用来找到一个可以合一的表达式的有限集合的 mgu，并且当那个集合不能被合一时能返回失败。这里给出的算法UNIFY是从[Chang & Lee 1973，P. 77]给出的一个算法改编的。它工作在一个列表结构的表达式集合上，在这些表达式中，每个文字和项作为一个列表项。例如：¬ P (x, f(A, y)写为 (¬P x (f A y))) 列表结构形式，表达式 ¬P是列表中的第一个顶级表达式，(f A y) 是第三个顶级表达式。

UNIFY的基础是分岐集（*disagreement set*）的思想。一个非空的表达式集合$\mathcal{W}$的分岐集由下面的方法得到：首先定位第一个符号（从左边计数），在这个位置不是$\mathcal{W}$中的所有表达式有完全相同的符号，然后从$\mathcal{W}$的每个表达式中提取从占据那个位置的符号开始的子表达式，各个子表达式集合构成$\mathcal{W}$的分岐集。例如，两个列表{(¬P x (f A y)), (¬p x (f z B))}集合的分岐集是{A, z}。分岐集能用置换A/z产生协调。

UNIFY (Γ) (Γ是一个列表表达式集合)

1) k ← 0，Γ_k ←Γ，σ_k ← ε (初始化步骤；ε是空的置换)。

2) 如果Γ_k是一个单元素集合，用Γ 的mgu σ_k 退出；否则继续。

3) D_k ←Γ_k的分岐集。

4) 如果在D_k中存在元素v_k和t_k，v_k是一个不会出现在t_k中的变量，则继续；否则，失败退出，Γ是不可以合一的。

5) $\sigma_{k+1} \leftarrow \sigma_k\{t_k/v_k\}$，$\Gamma_{k+1} \leftarrow \Gamma_k\{t_k/v_k\}$(注意$\Gamma_{k+1} = \Gamma_k\sigma_{k+1}$)。

6) $k \leftarrow k+1$

7) 转到第2步。

[Chang & Lee 1973, p.79]证明了UNIFY或者能找到一个可以合一表达式集合的一个最一般的合一式；或者当表达式不能合一时，能报告失败。算法产生mgu作为一个列表结构表达式对，但这些能被方便地转换为一阶逻辑中使用的形式。有几个算法可以执行合一化过程，包括一个以线性时间运行的算法[Paterson & Wegman 1978]。

作为例子，列出了几个文字集合的最一般置换实例（它们由mgu获得）。可以对文字集合中的每一个逐步执行UNIFY算法。

文字集合	最一般置换实例
{P(x), P(A)}	P(A)
{P[f(x), y, g(y)], P[f(x), z, g(x)]}	P[f(x), x, g(x)]
{P[f(x, g(A, y)), g(A, y))], P[f(x, z), z]}	p[f(x, g(A, y)), g(A, y)]

在UNIFY的第4步，检查是否一个变量出现在我们将要置换的一个项中。这样的置换会导致一个无限循环，因此不允许。作为一个例子，假如我们企图合一 P(x, x)和P(f(z), z)。UNIFY首先用f(z)置换这些表达式中的x，产生P(f(z), f (z))和P(f(z), z)。如果不进行检查，它将用f(z)置换z产生P(f(f(z)), f(f(z)))和P(f(f(z)), f(z))，等等。

16.2 谓词演算归结

假如γ_1和γ_2是两个子句（表示为文字集合）。如果γ_1中有一个原子ϕ，γ_2中有一个文字$\neg\psi$，并使ϕ和ψ有一个最一般合一式，那么这两个子句有一个归结式ρ，它通过把置换μ与γ_1和γ_2减去互补其文字的并集而得到。

即：$\rho = [(\gamma_1 - \{\phi\}) \quad (\gamma 2 - \{\neg\psi\})]\mu$

在两个子句被归结前，为了避免变量混淆，我们对每个子句中的变量重命名以使一个子句中的变量不会出现在另一个中。例如：假如我们正在归结P（x）∨Q(f(x))与R(g(x)) ∨ ¬Q(f(A))，首先重写第二个子句，比如说为 R（g(y)）∨ ¬ Q(f(A))，然后执行归结获得P(A)∨ R(g(y))。变量重命名被称为对变量进行标准化(*standardizing the variables apart*)。

下面是一些例子。

{P(x), Q(x, y)}和{¬P(A), R(B, z)}归结产生{Q(A, y), R(B, z)}。

{P(x, x), Q(x), R(x)}和{¬P(A, z), ¬Q(B)}可用两种不同的方式归结，分别产生{Q(A), R(A), ¬ Q(B)}和{P(B, B), R(B), ¬P(A, z)}。

有时需要对谓词演算归结有一个稍强的定义。例如，考虑两个子句 {P(u), P(v)}和{¬P(x), P(y)}。这两个子句各自有基例P(A)和 ¬P(A)（由置换A/u，A/v，A/x，A/y获得）。从这些基例中，能推断出空子句，因此我们应该能从初始子句推断它，但是刚刚给定的归结规则不能做到这些。更强的规则如下。

假定γ_1和γ_2是两个子句（再次表示为文字集合）。如果有γ_1的一个子集γ_1'和γ_2的一个子集γ_1'使得γ_1'的文字能与γ_1'的文字的否定式用最一般合一式μ合一，那么这两个子句有一个归结式ρ，它通过把置换μ与γ_1和γ_2减去其互补子集的并集而得到。即：

$\rho = [(\gamma_1 - \gamma_1') \quad (\gamma_2 - \gamma_1')]\mu$

用这个归结定义，归结子句{P(u)，P(v)}和{¬p(x)，¬P(y)}产生空子句。

16.3 完备性和合理性

谓词演算的归结是合理的。也就是说，如果ρ是两个子句ϕ和ψ的归结式，那么{ϕ，ψ}⊨ρ。这个事实的证明不比命题归结的合理性证明难。就如同在命题演算中一样，归结的完备性相对更难一些。我们不能只根据归结就推论出被一个给定集合逻辑涵蕴的所有公式。例如：我们不能从P(Ac中推断P(A)∨ P(B)。在命题归结中，我们用归结反驳克服这个困难，在谓词演算中也能如此。然而，为了保证反驳的完备性，我们必须用刚刚给定的更强的归结规则。

16.4 把任意的合式公式转化为子句形式

就像在命题演算中一样，任何合式公式能被转化为子句形式。步骤如下：

1) 消除蕴含符号（如在命题演算中一样）。
2) 减少否定符号的范围（如在命题演算中一样）。
3) 变量标准化，由于量词范围内的变量像“ 哑元 ”，因此它们能被更名，以使每个量词有它自己的变量符号。

   举例：(x) [¬ P (x) ∨ (x)Q (x)]可以改写为 (x) [¬P(x) ∨ (y)Q (y)]
4) 取消存在量词。

   举例：在 (x) [(y)Height(x，y)]中，存在量词在一个全称量词范围内，因此，y的“ 存在 ”可能依赖 x的值。例如，如果 (x) [(y)Height(x，y)]的意思是“ 每个人*x*有身高*y* ”，那么很明显身高与人有关。用某个未知的函数 h(x)显式定义这个依赖关系，h(x)把x的每个值映射为存在的y值。这样的一个函数叫做 Skolem函数[⊖]。如果我们把Skolem函数用在y存在的位置，就能取消存在量词，并写为 (x) Height[x，h(x)]。

   从一个合式公式中消除一个存在量词的一般规则是用一个 Skolem函数代替存在量词作用范围内变量的每一次出现，Skolem函数的参数是那些被全称量词约束的变量，全称量词的范围包括要被消除的存在量词的范围。用在 Skolem函数中的函数符号必须是“ 新的 ”，不能是已经出现在任何合式公式中被用在归结反驳中的符号。

   因此，我们能从

$$[(\forall w)Q(w)] \supset (\forall x)\{(\forall y)\{(\exists z)[P(x, y, z) \supset (\forall u)R(x, y, u, z)]\}\}$$

经消除 (z)，产生

$$[(\forall w)Q(w)] \supset (\forall x)\{(\forall y)[P(x, y, g(x, y)) \supset (\forall u)R(x, y, u, g(x, y))]$$

我们能从

$$(\forall x)\{\neg P(x) \vee \{(\forall y)[\neg P(y) \vee P(f(x, y))] \wedge (\exists w)[Q(x, w) \wedge \neg P(w)]\}\}$$

经消除 (w)，产生

$$*(x)\{\neg P(x) \vee \{y[\neg P(y) \vee P(f(x, y))] \wedge [Q(x, h(x)) \wedge \neg P(h(x))]\}\}$$

⊖ Skolem 函数是根据逻辑学家 Thoralf Skolem [Skolem 1920] 的名字命名的。

其中g(x，y)和h(x)都是Skolem函数。

如果要被消除的存在量词不在任何全称量词的范围内，那么我们使用一个没有参数的Skolem函数，它只是一个常量。因此，(x)P(x)成为P(Sk)，常量符号Sk用来指向我们知道存在的那个项。另外，Sk是一个新的符号常量且没有用在其他函数中，这是必要的。

为了从一个合式公式中消除所有的存在量化变量，依次对每个子公式使用前述的过程。从一组合式公式中消除存在量词产生公式集合的Skolem范式。

注意，一个合式公式的Skolem范式并不等价于原始的合式公式！公式(x)P(x)被它的Skoem范式P(Sk)逻辑涵蕴，但反过来就不行。作为另一个例子，注意[P(A)∨P(B)]⊨(x)P(x)，但是[P(A)∨P(B)]⊭P(Sk)。公式集合Δ是可以满足的，当且仅当Δ的Skolem范式是可以满足的。或者，对归结反驳更有用的是，Δ是不可满足的，当且仅当Δ的 Skolem范式是不可满足 [Loveland 1978，PP.41 以后]。

5) 转化为前束范式。在这个阶段，没有任何残留的存在量词，每个全称量词有它自己的变量符号。现在，我们可以把所有的全称量词移到合式公式的前面，并且让每个量词的范围包括跟随它的合式公式的全部。这样的合式公式被称为是在前束范式中。前束范式的一个合式公式包括一个称为前束范式的量词，它后面跟随一个称为矩阵（*matrix*）的没有量词的公式。前面标有 * 的合式公式例子的前束范式是：

$$(\forall x)(\forall y)\{\neg P(x) \vee \{[\neg P(y) \vee P(f(x,y))] \wedge [Q(x,h(x)) \wedge \neg P(h(x))]\}\}$$

6) 将矩阵写成一个合取范式。像命题演算一样，我们可以通过重复使用分配律，用($\omega_1 \vee \omega_2$) ∧ ($\omega_1 \vee \omega_3$)代替$\omega_1 \vee (\omega_2 \wedge \omega_3)$，把任何矩阵改写成合取范式。

将前述合式公式例子的矩阵改写成合取范式，它变为

$$(\forall x)(\forall y)\{[\neg P(x) \vee \neg P(y) \vee P(f(x,y))] \wedge [\neg P(x) \vee Q(x,h(x))] \wedge [\neg P(\vee \neg P(h(x))]\}$$

7) 消除全称量词。由于用在合式公式中的所有变量必须是在一个量词范围内，保证在这一步保留的所有变量都被全称量化。而且，全称量化的顺序并不重要，因此可以删去明确出现的全称量词，并且根据约定可以保证矩阵中的所有变量都被全称量化。现在，只留下了一个合取范式的矩阵。

8) 消除∧符号。可以用合式公式集合{ω_1，ω_2}代替表达式($\omega_1 \wedge \omega_2$)，删除显式出现的∧符号。重复代替的结果是获得一个有限的合式公式集合，合式公式中的每一项是一个文字析取项。只包含文字析取项的任何合式公式被称为一个子句，我们的合式公式例子被转化为下面的子句集合：

$$\neg P(x) \vee \neg P(y) \vee P[f(x,y)]$$

$$\neg P(x) \vee Q[x,h(x)]$$

$$\neg P(x) \vee \neg P[h(x)]$$

9) 变量更名。变量符号可以被更名，以使没有任何变量符号出现在多于一个的子句中。注意到(x)[P(x)∧Q(x)]等价于[(x)P(x)∧((y)Q(y))。现在子句是：

$$\neg P(x1) \vee \neg P(y) \vee P[f(x1,y)]$$

¬P(x2) ∨ Q[x2, h(x2)]

¬P(x3) ∨ ¬P[h(x3)]

一个子句的文字可以包含变量，但这些变量总是被理解为是全称量化的。

16.5 用归结证明定理

当归结用作一个定理证明系统的推论规则时，可以将要从中证明一个定理的合式公式集合首先转化成子句。可以证明如果合式公式ω从一个合式公式集合Δ中逻辑地产生，那么同样也从Δ中的合式公式转换成的子句集合中逻辑地产生。反之亦然[Davis & Putnam 1960]。因此，子句是一个表达谓词演算合式公式的完备的通用形式。

归结反驳能被证明既是合理的也是完备的[Robinson 1965]（也见[Chang & Lee 1973，p.85]）。因此，为了证明来自Δ 的一个合式公式ω，我们像在命题演算中一样进行。对ω取反，把这个否定转换为子句形式，并把它加到Δ的子句形式中；然后应用归结直到推出空子句。我们能用任何在命题演算中讨论的与归结有关的排序和精炼策略。

回到上一章提到的搬箱子机器人的一个例子。假如这个机器人知道27号房间中的所有箱子都比28号房间中的小。即：

1) (x, y) {[Package(x)∧Package(y)∧Inroom(x, 27)∧Inroom(y, 28)] Smaller(x), y}

缩写谓词符号使公式更紧凑。转换为子句形式，产生：

2) ¬P (x) ∨ ¬P(y)∨ ¬I(x, 27)∨ ¬I(y, 28)∨S(x, y)

假定机器人知道箱子A在27号或28号房间中（但不知道是哪一个）。它知道箱子B在房间27中且B不比A小。

3) P (A)

4) P (B)

5) I (A, 27) ∨ I (A, 28)

6) I (B, 27)

7) ¬ S (B, A)

用归结反驳，机器人能证明箱子A在27号房间中。图16-1是一棵证明树。待证明的合式公式的否定被标在左上角；和机器人明确知道的相对应的合式公式被标在图中沿右手方向。归结期间的置换在图中也已表示出来。

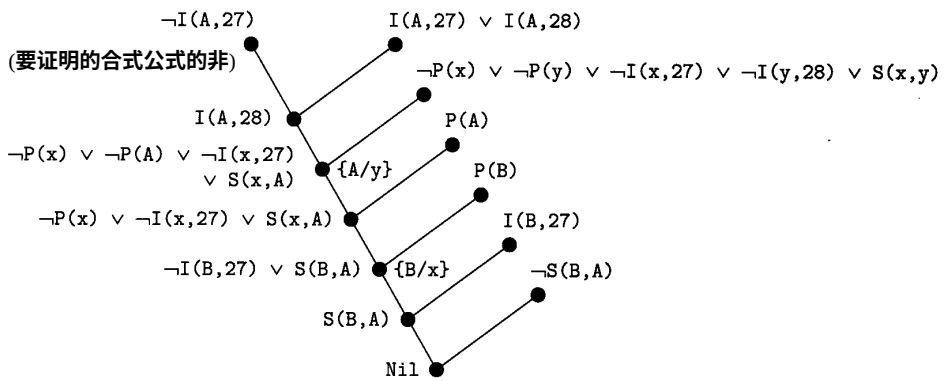


图16-1 一个归结反驳

16.6 回答提取

为了用归结回答由谓词演算合式公式表示一个领域知识的问题，通常要做的不只是证明一个定理。例如，假定必须证明形如（ ξ）ω（ξ）的一个定理。我们可能也想得到已经证明存在的ξ。为了做到这些，必须跟踪在反驳过程中所做的置换，可以用一个回答文字策略跟踪那个置换。将文字Ans(ξ_1，ξ_2，...)加到要证明的定理的否定子句，执行归结直到只剩下一个回答文字。在Ans文字中的变量是出现在待证明定理的否定子句形式中的所有变量。在证明期间代替Ans文字中变量的项，将成为被证明合式公式中存在量化变量的实例。回答提取由[Green 1969b]发明。它后来被[Luckham & Nilsson 1971]分析并扩展。也可参见[Nilsson 1980，pp 175以后]。

在图16-2中，给出了如何使用Ans文字的一个例子，现在要证明合式公式（ u)I(A，u)，它可以看作是机器人问它自己："A究竟在哪个房间？"。

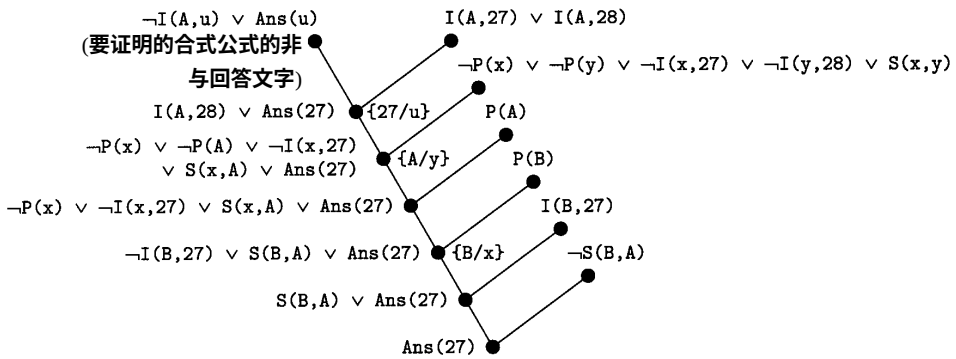


图16-2 回答提取

16.7 等式谓词

用在一个知识库公式中的关系常量常常有意向含意（即，关系），但是这些关系仅仅被知识库的模型集合所约束，根本不会被用在关系常量中的特殊符号所约束。归结反驳的结果将和意图含意相一致，只要知识库适当地约束实际的关系。

然而，有一些重要且经常出现的关系，我们可能想让所有的知识库描述它们。例如等式关系，我们可以使用前缀形式Equals(A，B)或者中缀形式A = B来表示一个关系常量。仅仅因为在知识库中包括了公式Equals(A，B)，并不意味着能够从P（A）中推出P（B），正如我们也不能归结¬ Q(B，A)和Q(A，B)一样。没有附加的公式，知识库无法知道Equals是什么。

有一些著名的等式属性能用公式表达，我们能把知识库加到这些公式中。这些等式关系是：

- 自反性(x)Equals(x，x)
- 对称性(x，y)[Equals(x，y) Equals(y，x)]
- 传递性（ x，y ，z）[Equals(x，y) ∧ Equals(y，z) Equals(x，z)]

即使有这些公理，我们也不能从P（A）和Equals（A，B）证明P（B)。需要的是能把相等项置换为任何表达式的相等项。为了支持这种置换，知识库将另外需要显式列出所有允许的置换（对每个谓词常量和函数常量）——这是一个不切实际的要求。

实际上有几个可能的选择。最强的一个叫调解（*Paramodulation*）[Wos & Robinson 1968]，[Chang & Lee 1973，pp.168~170]。调解是一个特定的等式推论规则，在知识库包含等式谓词的

情况下它和归结混合使用。规则定义如下：

假如γ_1和γ_2是两个子句（表示为文字集合）。如果$\gamma_1 = \{\lambda(\tau) \quad \gamma_1'\}$，$\gamma_2 = \{\text{Equals}(\alpha, \beta) \cup \gamma_2' \quad \}$ τ、α和β是项，γ_1' 和 γ_2' 是子句，$\lambda(\tau)$是一个包含项τ的文字（可能在其他条目之中）。如果τ和α有一个最一般合一式σ，那么推导出γ_1和γ_2的二元调解式（*binary Paramodulant*）：

$$\pi = \{\lambda\sigma[(\beta\sigma)] \cup \gamma_1'\sigma \cup \gamma_2'\sigma\}$$

其中 $\lambda\sigma[(\beta\sigma)]$表示用$\beta\sigma$代替在$\lambda\sigma$中单个$\tau\sigma$出现的结果（等式的对称性被一对倒置$\alpha$和$\beta$角色的规则处理）。

在操作过程中，规则不像它的正式定义显示的那样复杂。作为第一个例子（用大锤砸坚果），从P（A）和（A = B）证明P（B）（为简短，用中缀符号）。对一个反驳类型的证明，我们必须从子句 P（B）、P（A）和（A = B）推导出空子句。对上面两个子句使用调解法，$\lambda(\tau)$是P（A），τ是A，α是A，β是B。由于A（在τ中）和A（在α中）普通地合一而没有置换，二元调解式是P（B），它是用β（即B）代替τ(即A)产生的一个结果。用$\neg$P（B）归结这个二元调解式产生了空子句。

把下面的例子（来自[Chang & Lee 1973，p.170]）留给读者去思考：

P(g(f(x)))∨Q(x) 和 [f(g(B)) = A]∨R(g(C))的二元调解式是

$$P(g(A)) \vee Q(g(B)) \vee R(g(C))$$

对这种允许的调解式作一个轻微的扩展（超过二元的），能够证明与归结反驳组合的调解对包含等式谓词的知识库是完备的。

对那些不需要用相等置换相等的问题，就不再需要调解的能力了。在很多情况下，我们用一个叫做谓词评估的技术来处理。如果一个外部处理能返回一个等式谓词的真值，我们可以用适当的T或F代替那个谓词（当它出现在一个公式中时）。在归结反驳中，包含文字T的子句能被取消。任何子句中的文字F能被消除（它总能被一个假定的无所不在的T所归结。）

例如，考虑问题（它可能是关于一个推东西的机器人）：证明如果一个箱子A在一个特定的屋子*R*1中，那么它不能在一个不同的屋子*R*2中。机器人在它的知识库中将有如下的语句：

(x，y，u，v) [In(x，u)∧(u ≠v)] ¬In(x ，v)

In(A，R1)

⋮

它试图证明¬In (A，R2)。把第一个公式转化为子句形式，产生

¬In (x，u)∨(u = v)∨ ¬ In(x ，v)

该策略延迟处理等式谓词，直到它们只包含基本项。用待证明的合式公式的否定归结子句，产生

(R2 = v) ∨¬In(A ，v)

用给定的合式公式 In(A，R1)归结这个结果，产生：

(R2 = R1)

现在，我们可以想像知识库事实上包含合式公式 ¬(R2 = R1)。如果这样，我们能用它产生空子句，完成反驳。但是在有*M*个房间的大建筑物中，我们将需要 $\frac{M(M-1)}{2}$ 个这种不等式。如果机器人必须做涉及到数字的推理，它可能需要一个大得无法管理的合式公式集合，如 ¬（3724 = 4861），等等。并非将所有的这些合式公式显式地放在知识库中，而是提供一个例程（即一个程序），它能够对所有的（基本）α和β的表达式（$\alpha = \beta$）进行评估，这样将会更好。在例子中，

程序对表达式（R2= R1）将返回F（或Nil），反驳将完成。

几个其他常用的关系（大于，小于，……）和函数（加法，减数，除法，……）能被直接评估而不必用公式推理。因此，在自动推理系统中表达式评估是一个强大而高效的工具。

16.8 补充读物和讨论

有些人并不愿意寻求归结推论规则，而宁愿使用自然演绎方法[Prawitz 1965]。这些被称为是“自然的”，因为推论是在那些多少“似乎”没有转化为规范形式的句子上进行。[Bledsoe 1977]讨论了一些非归结方法。

在把归结和其他推理方法应用到数学定理证明中，Larry wos和后来的Woody Bledsoe一直是领先者。这个工作产生了新的归结策略，像支持集[Wos，Carson，& Robinson 1965]和单元首选。强大的定理证明系统（其中有些已经解决了数学中的开放问题）的例子能在[Wos & Winker 1983，Boyer & Moore 1979，Stickel 1988，McCune 1992，McCune 1994，Wos 1993]中发现。[Wos，et al 1992]是一本有关计算机定理证明和它在问题解决应用中的教科书。

定理证明也已被应用于验证和综合给定规范说明的计算机程序中。[Manna & Waldinger 1992]是一本处理自动程序综合的教程。[Manna & Waldinger 1985，Manna waldinger 1990]是处理逻辑和编程的书。[Lowry & McCartney 1991]是有关程序合成的文集。

谓词评估是一个叫语义连接的更一般过程的一个实例。在语义连接中，数据结构和程序与谓词演算语言的元素关联（即连接到）。然后，被连接的结构和过程能被用来在一种与它们的意图解释对应的方式下评估语言中的表达式[Weyhrauch 1980，Myers 1994]。

《Journal of Automated Reasoning》中包含了定理证明技术的理论和应用。

习题

16.1 判断下面的表达式对是否能够合一，并给出每个可能的合一对的最一般合一式：

1) P(x，B，B)和P（A，y，z）
2) P(g(f(v))，g(u))和P（x，x）
3) P(x，f(x))和P(y，y)
4) P(y，y，B)和P(z，x，z)
5) 2 +3 = x 和 x = 3+3

16.2 解释一下为什么P（f(x，x)，A）与P(f(y)，f(y，A))，A）不能合一。

16.3 把下面的表达式转换成子句形式：

1) ((x) [P(x)]∨(x)[Q(x)] (x)[P(x)∨Q(x)]
2) (x)[P(x) (y)[(z)[Q(x，y)] ¬(z)[R(y，x)]]]
3) (x)[P(x)] (x)[(z)[Q(x，z)]∨(z)[R(x，y，z)]]
4) (x)[P(x) Q(x，y)] ((y)[P(y)]∩(z)[Q(y，z)])

16.4 给定下面的一段话：

Tony，Mike，and John belong to the Alpine Club. Every member of the Alpine Club is either a skier or a mountain Climber or both. No mountain Climber likes rain，and all skiers like snow. Mike dislike whatever Tony likes and likes whatever Tony dislikes .Tony likes rain and snow.

用一种方式通过谓词演算语句表达这个信息，在这种方式中，把问题：“Who is a member of the Alpine Club who is a mountain climber but not a skier?”表达为一个谓词演算表达式。用归结反驳与回答提取回答它。

16.5 证明（ x）λ(x)├ λ(τ)，τ是一个基本项。

16.6 通过一个归结反驳，证明合式公式（ x)P(x)逻辑上产生于合式公式 [P(A1)∨P(A2)]。存在一个证明(x)P(x)的Skolem范式P（Sk）逻辑上产生于[P(A1）P(A2)的归结反驳吗？

16.7 Sam、Clyde和Oscar是大象。关于它们，我们知道下面的事实：

1) Sam是粉红色的；

2) Clyde是灰色的且喜欢Oscar；

3) Oscar 是粉红色或者是灰色（但不是两种颜色）且喜欢Sam。

用归结反驳证明一个灰色大象喜欢一个粉红色大象，即，证明：

（ x，y）[Gray(x)∧Pink(y)∧likes(x,y)]

16.8 对下面的每个公式，或者证明它是永真的，或者给出它的永真性的一个反例（提示：不要立即进行归结证明）。

1) {[(x)P(x)] Q(A)} {(x)[P(x) Q (A)]}

2) {[(x)P(x)] Q(A)} {(x)[P(x) Q (A)]}

3) (x)[P(x) Q(A)] (x)[P(x) Q(A)]

16.9 把下面的公式转化为既没有存在量词也没有Skolem函数的形式。

(x，y){(z)[On(x，z) ∧ Above(z，y)] Above(x，y)}

16.10 如果给定下面的条件，用一组子句上的归结反驳证明有一个绿色对象：

- 如果可推动的对象是蓝色的，那么不可推的对象是绿的。
- 所有的对象或者是蓝色的或者是绿色的，但不能是两色的。
- 如果有一个不可推动的对象，那么所有可推动的对象都是蓝色的。
- 对象*O*1是可推动的。
- 对象*O*2是不可推动的。

1) 把这些句子转换成一阶谓词演算中的表达式。

2) 把上述谓词演算表达式转换为子句形式。

3) 把上述子句形式的表达式与要证明的语句的否定子句形式组合起来，然后给出用在获得一个归结反驳中的步骤。

16.11 函数cons(x，y)表示由把元素x插在列表y的头部形成的列表。我们用Nil表示空列表；列表（2）由cons（2，nil）表示；列表（1、2）由cons(1,cons(2，Nil))表示；等等。公式Last(L，e)意指e是列表L的最后一个元素。我们有下面的公理：

- (" u) [Last(cons(u ，Nil)，u)]
- (" x，y，z)[Last(y，z) … Last(cons(x，y)，z)]

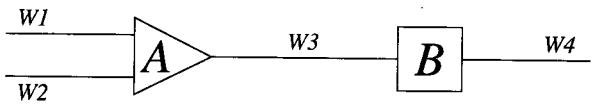
1) 从这些公理中用归结反驳证明下面的定理：

（ v）[Last(cons(2，cons(1，Nil))，v)]

2) 用回答提取找到v，列表（2，1）的最后一个元素。

16.12 下面的逻辑电路有4条线，*W*1，*W*2，*W*3和*W*4。它有一个“与门”*A*和一个“反相器”

B。输入线*W*1和*W*2或者是“ on ”或者是“ not ”。与门*A*运行“ OK ”，则仅当*W*1和*W*2都是“ on ”时*W*3才是“ on ”。如果反相器*B*运行“ OK ”，仅当*W*3是“ on ”时*W*4才是“ on ”。



1) 使用形如OK(A)、ON(W1)的表达式描述所定义的电路功能。
2) 使用描述电路功能的公式，假定所有的元件都是运行正确的，并且*W*1和*W*2是“ on ”，用归结法证明*W*4不是“ on ”。
3) 再次使用描述电路功能的公式，给定*W*1和*W*2是“ on ”，且W_4也是“ on ”，用归结法证明或者与门或者反相器不是运行正确的。