第17章　基于知识的系统

17.1　面对现实世界

既然我们已经学习了用逻辑来进行表示和推理，并且用一些比较简单的例子演示了它的使用，那么我们要问它是否能被应用到“现实世界”问题中呢？AI研究已经发现诸如医疗诊断，税金咨询和设备设计等方面的应用典型地需要大量的随手可得的主题知识。这些应用强调知识的重要性促使我们使用基于知识的系统来描述对大量知识库进行推理的程序。已经描述的方法能“按比例增大”以使其在实际应用中表现好吗？与这个问题相关的推理方法的一些理论特性是什么？在本章中，要解决其中一些问题，并讨论在现实应用中已经证明对实际推理是有用的一些方法。

逻辑推理系统有三个主要的理论特性：合理性、完备性和易处理性。为了确信一个推导的结论是“真的”，需要合理性。为了确信推论最终将产生真的结论，需要完备性。为了确信推论是可行的，需要易处理性。关于谓词演算，归结反驳是合理的和完备的。归结反驳能被用来证明，如果一个合式公式ω被一合式公式集Δ逻辑涵蕴，那么它是完备的；否则，归结反驳过程可能永不能终止。因此，我们不能用归结作为一个完全的决策过程（*decision procedure*）。进一步讲，可以证明没有其他方法总能告诉我们什么时候一个合式公式ω不是由合式公式集合Δ逻辑地派生，什么时候是。因此，我们说谓词演算是不完全决策的。当然，不完全决策性导致了谓词演算固有的不易于处理性。

但是情形变得更糟。即使对那些归结反驳终止的问题，那个过程也是NP难题——就像对一阶谓词演算的任何合理和完备推理过程一样[Börger 1989]。因此，虽然很多推理问题能被公式化为归结反驳问题，但对非常大的问题，该方法是不易处理的，这个事实导致很多人对在大规模推理问题中使用正式的逻辑方法感到绝望（例如，见[Schwartz 1987, McDermott 1987]）。然而，由于人类自己进行复杂推理，一定有启发性的和特定的公式允许易处理的计算。

研究人员已经探索了各种方式以使推理更有效。首先，我们能在坚持推理规则的合理性前，使用那些可能偶尔（我们很少希望）会“证明”一个不正确的公式过程。第二，在坚持完备性前，使用不能保证找到正确公式证据的过程。这两种修改可以使推理更有效。第三，我们能使用一种比完全谓词演算表达力弱的语言。下面要讨论的一个表达力弱的语言的例子仅仅使用了Horn子句。使用Horn子句的推理典型地讲更有效，它们可以满足很多应用。

17.2　用Horn子句进行推理

前面，我把Horn子句定义为至多有一个正文字的子句。如果有至少一个负文字和一个正文字，Horn子句就能被写为一个蕴含，它的前项是正文字的一个合取，它的后项是单个正文字，这样一个子句称为一个规则。在子句中可以没有负文字，在这种情况下，我们把它记为一个蕴含，它的前项为空，后项是一个单一正文字，这样一个子句被叫做一个事实。或者在子句中可能没有正文字，在这种情况下，我们记作后项为空、前项是一个正文字列表的蕴含，这样的子

句称为一个目标。Horn子句形成PROLOG编程语言的基础[Colmerauer 1978,Clocksin & Mellish 1987, Sterling & shapiro 1986,Bratko 1990]。在PROLOG语言中，这些子句用作该语言的语句，写成如下格式：

- 规则：λ_h :- λ_{b1}, ..., λ_{bn}

（这是表示蕴含$\lambda_{b1} \wedge ... \wedge \lambda_{bn}$ λ_h的一种特殊方式），每个λ_i是一个正文字。文字λ_h叫做子句的头，文字λ_{b1},..., λ_{bn}的有序列表叫做子句的体。

- 事实：λ_h :-
- 目标：- λ_{b1}, ..., λ_{bn}

目标和规则体中的文字是有序列表，这个顺序在一个 PROLOG程序的执行中起到重要的作用。

对PROLOG子句的推理由试图“证明”一个目标组成，它通过执行一个PROLOG程序来完成。这样一个证明通过对PROLOG事实、目标和规则执行类似于归结的操作而获得。每个归结在一个目标与一个事实或规则之间执行。

- 一个目标能与一个事实归结 —— 通过将该事实和目标中的一个文字合一。我们称这个文字是被归结的。归结式是一个由初始目标中其他文字的所有置换实例组成的一个新目标（例如，那些没有被归结的） —— 用同初始目标相同的顺序记录。置换实例通过把合一的mgu应用到所有其他的文字完成。
- 一个目标能和一个规则归结 —— 通过合一规则的头与目标中的一个文字。归结式是一个新目标，它通过把规则体中的所有文字的置换实例列表追加到目标中所有其他（没有归结的）文字置换实例表的前面而形成。

在PROLOG程序中，子句常常用下面的方式排序：第一个子句是目标子句，接着是事实，最后是规则。在执行程序时，解释器依次通过顺序中的目标文字，检查排序的每个子句，执行第一个可能的归结来检查与目标的归结，然后从新的目标子句开始。当应用一个归结产生的子句为空时，一个目标子句的证明成功。当一个仅有一个文字的目标子句和一个事实规则归结时，上述情况才会发生（因此，我们把事实放在首位以便只要可能就会成功）。如果解释器已经试过一个目标文字的所有归结，且在能被证明的新目标中没有任何结果时，一个目标子句宣布失败（不能被证明）。在这种情况下，解释器回溯到前面的目标子句，对它尝试其他的归结。这个过程容易被看成与对子句使用深度优先、回溯搜索过程的普通线性归结（用子句中的文字和子句的排序控制的归结顺序）等价。如果能很好地选择归结的子句顺序和要归结的子句中的文字，使用这些顺序的深度优先搜索是能发现一个证明的有效方式。 PROLOG程序员用有序地表达子句和有序地表达每个子句中的文字来表示深度优先搜索的归结顺序。

用一些简单的例子演示PROLOG推理。第一个不涉及变量，给出了一个过程的高级视图。回想前面用命题演算演示推理的举木块的例子。假如木块是可以举起的，电池也已被充电，那么一个证明手臂移动的PROLOG程序如下：

```
1) :-  MOVES
2) BAT_OK :-
3) LIFTABLE :-
4) MOVES :- BAT_OK，LIFTABLE
```

第一个归结在目标子句和句子4之间进行，产生新目标子句::-BAK_OK，LIFTABLE。新

目标子句与语句2归结（在新目标子句中的第一个文字上归结），产生新目标子句：-LIFTABLE。然后再与语句3归结，产生空目标子句，成功终止程序。在这种情况下，不需要回溯。

上面的证明可用图17-1中的树结构表示，在方框中的节点集对应程序语句（目标、规则或事实）。节点用程序语句中的文字标识。有两种弧：匹配弧（用双线表示）和规则弧（用单线表示）。规则弧把一个规则的体节点同该规则的头节点相连。匹配弧把一个规则（或一个目标节点）的一个体节点连向一个等价的另一个规则的标识头节点（或到一个事实）。在这样一棵树中的一个目标节点或一个体节点被证明——如果它被一个匹配弧连向一个事实节点或者连向一个其体节点均已证明的头节点。给规则弧加了一个圆形符号（U）以强调为了证明头节点，必须证明所有的体节点。图 17-1中的树是“与/或（AND/OR）”证明树的一个实例。在这样一棵树中，体节点被称为AND节点，因为它们必须全被证明（如果有另一个可能的归结，通过另一个匹配弧对一个证明树的搜索会产生另外的节点，称为OR节点。随后我们将看到带有嵌入证明树的“与/或”树的一个例子）。

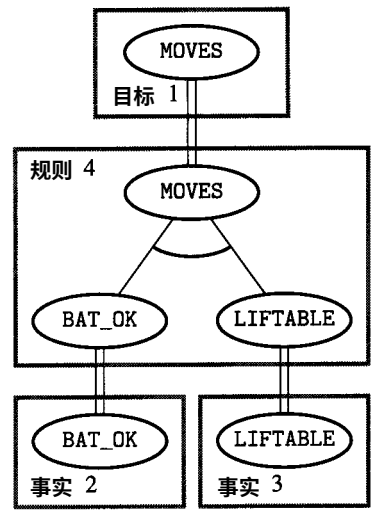


图17-1 “与/或”证明树

作为使用变量的一个例子[⊖]，回到积木问题。假如我们想使用一个积木在另一个积木上（above）的概念。用谓词演算，我们能用下面的两个公式用项on递归地定义above：

$(x, y, z)[On(x, y) \rightarrow Above(x, y)]$

$(x, y)\{(z)[On(x, z) \wedge Above(z, y)] \rightarrow Above(x, y)\}$

用这些定义，下面的PROLOG程序可用来证明当*A*在*B*上，*B*在*C*上时，A在C上：

```
1) :- Above(A,C)
2) On(A,B):-
3) On(B,C):-
4) Above(x,y) :- On(x,y)
5) Above(x,y) :- On(x,z), Above(z,y)
```

第一个可能的目标归结（用规则4）产生新目标 :- On(A, C)。目标失败，因此我们回溯到初始目标试下一个归结（用规则5）产生一个新目标 :- On(A, z)，Above(z, C)。归结这个目标中的第一个文字（用事实2）产生新目标 :- Above(B, C)。这个目标与规则4归结产生 :- On(B, C)，它和事实3归结产生空目标，过程终止。目标 :- Above(B, C)的产生可以被认为是对同一个程序的递归调用。

被PROLOG解释器执行的“与/或”演示搜索树显示在图17-2中。证明树中的节点用粗体椭圆表示。被中止的搜索部分的节点用阴影节点表示。一些匹配弧由置换标识（置换是对体节点和头节点合一时得到的），方框结构代表规则的置换实例。为了使树代表一个合法的证明，置

⊖ PROLOG 印刷约定与我们相反；在 PROLOG中，变量是大写的，常量是小写的字母和数字。为了方便与 FOPC 比较，本书坚持我的常用约定。

换必须是一致的。这里的一致性要求在整个证明树中用相同的项被置换为变量 z。当然，在PROLOG中，一致性通过在建立一个新目标时用项度量的立即置换得到。注意在证明树的规则4中实例化变量部分的使用。

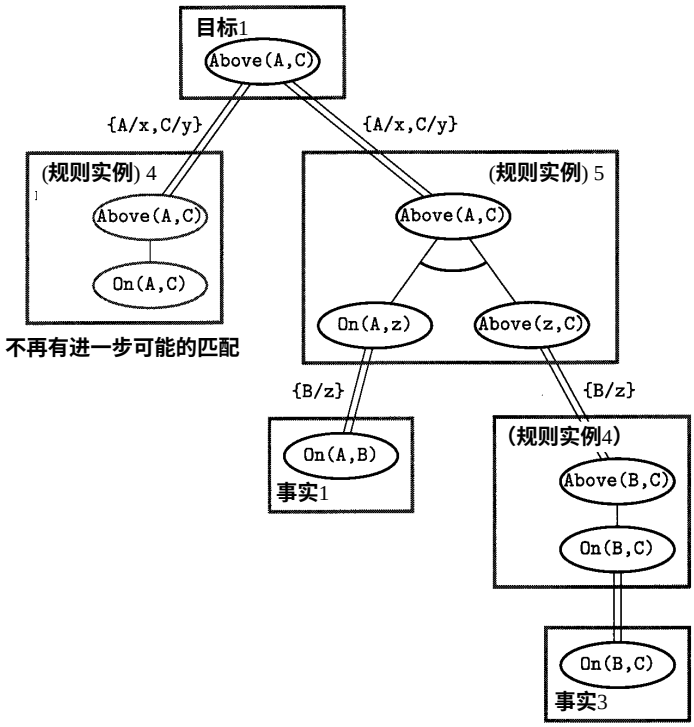


图17-2 一个“与/或”树

PROLOG语言是一门通用的编程语言，已经证明它在各种AI应用中都是有用的，尤其在自然语言处理中[Pereira & Shieber 1987]。由[Shoham 1994]写的一本课本讲解了如何用PROLOG来实现几项AI技术。和PROLOG有关的逻辑语言（如DATALOG）被用来定义“内涵关系”，用来扩展在一个关系数据库中的显式出现（外延关系）。要全面了解逻辑方法在数据库中的使用，可参见[Ullman 1988, Ullman 1989]。[Minker 1997]是一篇值得阅读的综述性文章。

这些基本的PROLOG程序说明了如何将Horn子句规则用于推理问题。在PROLOG语言中，推理“向后”进行，从要建立的目标开始，顺着规则链，最后锚定在事实上。使用目标规则和事实执行这类向后推理的AI系统被称为执行反向链。也可用正向链，在这种情况下，推理向前进行，从事实开始，顺着规则链，最后产生目标。正向链系统常常用标准的蕴含形式表示规则，它的前项由正文字构成，后项由一个正文字构成。就像PROLOG是一个基于反向链的系统一样，有一种计算机语言OPS5[Brownston, et al. 1985]是基于正向链的。已经用OPS5写出了几个实际有效的推理系统。

在正向链系统中，如果一个规则的前项文字的每一个都能与一个相应的事实（用一致的置换）合一，那么它是可应用的。由于事实是基本原子，变量仅出现在可被合一的一对表达式中。这种受限的合一形式常叫做模式匹配。一个规则应用就是将规则后项的相应实例加到事实列表中（规则的前项文字被一致地匹配到事实）。当可应用多于一个规则时，某种外部冲突归结被用来决定将应用哪个规则。当有大量的规则和事实时，必须被尝试的合一数量成为障碍。

OPS5使用一个叫RETE 的算法过程，它把规则编译成一个网络，用来通过规则的一致匹配有效地指导事实[Brownston, et al.1985,Forgy 1982]。

正向链 —— 事实触发规则的过程，它加入新事实等等 —— 非常类似第5章讨论的黑板系统的工作过程。在一个特定的形式中，它也可以作为人类认知过程的一个模型。这个事实相应于所谓的工作或短期记忆。规则相应于长期记忆。在AI中，基于这个认知模型的最完备的开发系统是那些基于SOAR形式方法的系统[Laird,Newell, & Rosenbloom 1987,Rosenbloom,Laird, & Newell 1993]。

17.3 动态知识库的维持

正向链生成规则必需的明显事实，但这些事实直到被推断出才能显式地应用到知识库中。一旦可用，这些被推断的事实就能用来演绎另外的事实，或者可以引发 agent动作。因此，直接向前推理被看作是对事实知识库做改变。这里详细讨论这个观点。

考虑一个命题演算原子知识库KB。用原子“前提（*premiss*）”代替事实，因为我们想能够推理各种“假设分析”的可能性。为具体起见，考虑一个例子，它的前提是 P和Q，规则是P∧Q R和P∨R S。注意，像前面一样，规则是带有单文字后项的蕴含，但现在它们没被限制为Horn子句。

一个使用规则的正向推理链允许我们把R和S作为新原子加到KB中。考虑这个知识库的方式之一是它像一个只有一行和几列的一维电子表格，我们可以称之为一个“扩展行”。对P、Q、R和S都有一列，在每列有一个单元放这些原子的值。在 P和 Q的每个单元中，我们能输入1（真）或0（假），而在R的值单元中，我们可以输入公式P∧Q，在S的值单元中输入公式P∨R。这些公式是每个规则的前项。一些扩展行实例显示在图 17-3中。上面的实例表示一个公式，底部显示了两个例子的单元推断值。像在电子表格中一样，一个公式的任意单元立即得到它的值，这个值从元件值计算（用正向链）而来。如果一个公式的任何元件值碰巧改变了，公式的值也自动改变。

P	Q	R	S
1	1	= P ∧ Q	= P ∨ R

P	Q	R	S
1	1	1	1

P	Q	R	S
1	0	0	1

图17-3 一个Spreadline推理系统

我们能延伸扩展行以包含任何数量的前提和规则，这样一个结构可作为一种动态知识库。推理过程被机制化，与显示的单元值无关。用这种方式实现的知识库可用于各种目的。就像电

子表格一样，它们能用来回答“假设分析”问题。某个人（或一个 agent）可能想知道，如果前提在一定的方式下改变，各种原子值将怎么改变。在 AI中，这些扩展行叫做推理维持（*reason maintenance*）系统。它们和后面要讨论的结构是所谓的真值维持系统（TMS）的实例，在规则单元值中输入的公式称为“理由（*justification*）”。它们常被表示为原子合取的析取——析取范式。各种有效的过程（像 RETE算法[Forgy 1982]）已经被设计来进行正向链后台计算，以计算所有单元的值。

一个有趣的扩展（超过传统的电子表格）使我们能够省略某些前提单元的值。不再把每个标识为1或0，而可能就是不给它们值。如果缺少前提单元的值，那么某个规则单元的值也可如此。当一个传统的电子表格面对一个带有省略值的元件公式时，它可以假定那个值是 0，或者它可以警告用户那个省略值。但是当一个动态知识库不能计算一个公式的值时（因为元件有省略值），它把公式的值也报告为省略的——留下该单元为空。在图 17-4中，展示了一个例子，对R和S的公式与图17-3中的相同。注意，在一种情况下即使缺少 S的一个元件，也能计算它的值。

P	Q	R	S
		= P ∧ Q	= P ∨ R

KB 实例

P	Q	R	S
1			1

P	Q	R	S
	1		

图17-4 一个带有缺少值的知识库

在TMS的词汇中，一个其值既不是1也不是0的原子被称为OUT。用OUT作为OUT原子的值，而不把它们的值单元留为空。当一个单元的值是 1或0时，它是IN。特别注意，把单元P的值从1改为OUT与P是False是截然不同的。这样一个改变仅仅是说知识库不再知道 P是否为真（或为假）。

允许单元值是OUT准许对规则的类型一般化。我们可以有规则，它的前项包括其值必须是OUT而不是0或1的原子。用一个例子来说明，如果 R为真，S是OUT，则我们能推论U。可以将它表示为一个类似蕴含的形式：

$R \wedge \bar{S} \Rightarrow U$

如果一个原子必须是OUT，用($\bar{}$)号作为它的上标（在TMS蕴含中的后项从来没有上标符号。即，当前项被满足时，后项是1，因此是IN）。

TMS应该被认为并非通用目的推理系统（像我们在第 13章到第16章学习的一样），而是知识库维持系统，当原子的真假改变时，它执行自动更新。 IN和OUT的使用允许一种基本的“非单调”或可废止的推理，它不被普通的推理系统支持。普通的逻辑推理是单调的，以致当一个新公式被加到知识库上时，在公式被加入前允许的所有推理仍然是有效的，加入新公式不会减

少可以证明的定理集合。但是如果对一个依赖某个别的为OUT的原子的原子φ做推理，那么当把那个别的原子加到IN后面时，我们将必须取消φ，当然，它可能导致进一步的取消（或演绎）。

然而规则的循环带来了一些困难，一是相互证明（*mutual justification*）。考虑合式公式P、P Q、Q R和R Q，如果P有真值，那么Q和R也有真值。但是如果我们不知道P的值，就不能对Q和R做出任何结论。但是在动态知识库计算中，Q和R能证明它们自己的值。考虑图17-5中的计算阶段。开始时，假设P为1，产生图中的开始KB。如果P变成OUT，扩展行计算能如图所示进行——导致R和Q证明自己而不用P的支持。另外，当一个公式中有OUT时，原子的值或者是不确定的或者过于确定。在第一种情况下，可能有不只一种方式使原子是IN或OUT；在第二种情况下，它们可能没有一致的值（更详细的内容参见[Shoham 1994, 第5章]。

一个TMS系统的一些应用要求值为1和IN的原子的某个子集的所有成员是不一致的。这些子集被叫做nogoods。一个一致的维持系统（CMS）通过改变一些IN为OUT，强迫这些约束。由于一致性能用几种不同的方式维持，故CMS是不确定的。

公式

P	Q	R
	= P ∨ R	= Q

开始知识库

P	Q	R
1	1	1

当P变为 OUT 时的计算阶段

阶段1：用公式及 R 的先前的值，计算新的Q值

P	Q	R
OUT	1	1

阶段2：给定 p 和Q 的先前的值，计算新的R值

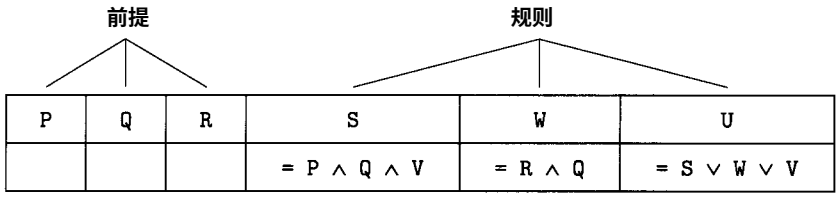
P	Q	R
OUT	1	1

图17-5 相互证明

继续使用扩展行来表示关于动态知识库中的讨论，其中的扩展行被向后而不是向前使用，即，不是从某些前提开始，用公式更新所有单元的值，而是以一个特殊单元的公式开始，寻找哪些前提值（它和某些其他公式一起被称为后台理论（ *background theory*））将使那个单元的公式为真。在这些应用中，将单元的值作为公式——按照前提原子用DNF形式表达。这些公式被称为单元的标记。具有这些连向原子的标记的知识库被称为基于假设的真值维持系统（ATMS）。每个原子的标记由各种前提值集（假设的）构成，这些前提值和后台理论一起使原子有真值。

例如，考虑图17-6顶部的扩展行。我们重新按照前提表达公式，并得到图底部的扩展行以获得ATM标记。通常，集合符号被用作标记。一个原子合取用一个集合表示，这些合取的析取用多个集合的一个列表表示。例如，在图 17-6中，为了使U为真，或者P必须为真（使V为真）或者R和Q必须为真（使W为真）。在U的一个最小标记中，我们不必包括集合{P，Q}（使S为真），因为P在标记中的出现包容了{P，Q}。

ATM能用于各种目的。一个是执行诊断，例如图 17-6中的公式可以对某个电子设备的功能建模。如果观察到U表示的部分有问题（即U有假值），可以用ATM确定P和P∧Q必定都为假，因为否则U将为真。因此我们能确定由R和Q代表的功能部分之一有问题，由P表示的部分也必然有问题。



ATMS标记

P	Q	R	S	W	U
P	Q	R	{P,Q}	{R,Q}	{P} {R,Q}

图17-6 一个TMS到一个ATMS的转换

ATM有时有其他的特征和限制。有些前提总是产生真值，因此去除标记（即，这些前提是后台理论的一部分）。标记也可能被一个假设（前提的一些子集是nogoods）所修改，因此不能总有真值。有关TMS，ATMS和它们的应用的详细情况，参见[de Kleer 1986a,de Kleer 1986b, de Kleer 1986c, Forbus & de Kleer 1993]。

17.4 基于规则的专家系统

使用事实和规则的AI推理技术的最成功应用之一是建立专家系统，专家系统包含了人类努力探索的一个专门领域的知识，如医疗、工程和商业。在[Feigenbaum,McCorduck & Nii 1988]中给出了在用的很多专家系统的调查（circa 1988）。[Dym & Levitt 1991]写了一本关于专家系统工程的教课书。严格地讲，任何具有专家功能的程序都被称为是一个专家系统。[Feigenbaum,McCorduck & Nii 1988]中有如下的定义：

在解决问题中，通过运用知识体达到专家级水平的AI程序叫做知识库系统或专家系统。经常，术语“专家系统”被约定用于知识库中包含人类专家使用的知识而不是从课本或非专家那里收集来的知识的程序。两个术语——专家系统和知识库系统更经常被同义使用。

在本书中，主要讨论基于规则的专家系统。

一个在使用环境中的专家系统的基本结构如图17-7所示（[摘自[Feigenbaum,McCorduck, & Nii 1988]）。系统的主要部分是知识库和推理引擎。根据到目前为止讨论的推理系统，知识库由谓词演算事实和有关讨论主题的规则构成（虽然它也可能包含用谓词演算的语法变种表示的知识。下一章将看到这样一个变种的例子）。推理引擎由所有操纵知识库来演绎用户要求的信息的过程构成——如归结、前向链或反向链（在某些实例中，知识库和推理机制可能被紧紧地连结在一起，就像它们在TMS中一样）。用户接口可能包括某种自然语言处理系统，它允许用户用一个有限的自然语言形式与系统交互（见第24章）。也可使用带有菜单的图形接口界面。解释子系统分析被系统执行的推理结构，并把它解释给用户（后面我们将看这种解释的一个简单例子）。

在实际应用中，这四个部分构成了一个系统。在一个专家系统结构中，一个“知识工程师”（经常是一个训练过的AI计算机科学家）与应用领域的一个专家（或几个专家）共同工作以便

把专家的相关知识表示成一种形式，以使它能被输入到知识库。这个过程经常由一个知识采集子系统协助。和其他情况一样，这个子系统检查正在增长的知识库的可能不一致和不完备信息，然后将它们表示给专家以做出决定。

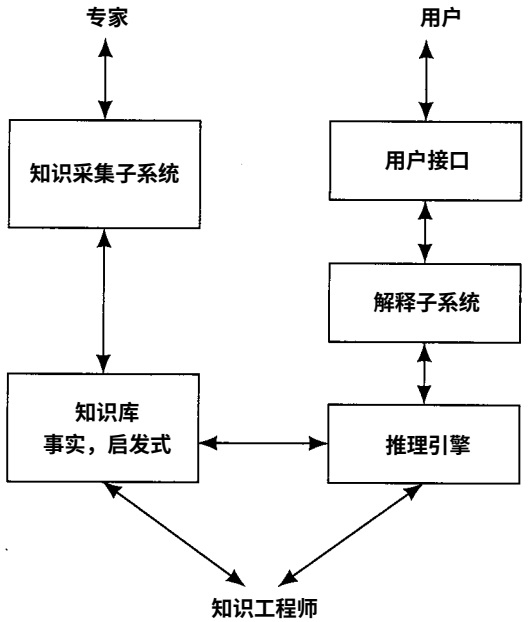


图17-7 一个专家系统的基本结构

建立系统的过程常常通过很多循环迭代进行，在循环的每一步，知识工程师和专家测试一个原型系统，看它是否能做出与专家对用户提出的典型问题相同的推理。如果系统与专家的反应不同，用解释子系统来帮助开发小组决定分歧由哪些信息和推理引起。专家可能需要更详细地说明某个信息或者提供附加的信息来包含当前的情况。这个过程继续下去直到开发小组对系统操作满意为止。建立专家系统的过程要求经验和判断。关于这个主题已经有了各种书籍；[Stefik 1995]是一本优秀的书籍。这里仅给出一些基本思想。

基于规则的专家系统常常建立在关于命题逻辑Horn子句的推理之上（也许还有某种其他的机制解决不确定性）。知识库由从专家收集来的规则构成。这些系统已被应用到很多环境中。例如，想像银行的一个贷款员使用这样一个系统帮助决定是否把一个私人贷款授于一个人。一个实际的贷款表决系统可能要考虑很多因素，远远超过我们在一个演示例子中所能想到的。但是能通过描述一个极大地简化了的版本来给出这样一个系统如何工作的大致思想。假如下面的原子将指称相关的命题：

- OK（贷款应该被批准）
- COLLAT（贷款抵押被满足）
- PYMT（申请人能做出贷款偿还）
- REP（申请人有良好的金融声誉）
- APP（抵押评估充分大于贷款额）
- RATING（申请人有良好的信贷等级）
- INC（申请人的收入超过他/她的消费）

• BAL（申请人有良好的资产负债表）

那么，下面的规则可被用来做决定：

1) COLLAT∧PYMT∧REP OK

2) APP COLLAT

3) RATING REP

4) INC PYMT

5) BAL∧REP OK

假定贷款员想确定一个申请人的OK值是否为真。为了证明OK，推理引擎用一个“与/或”证明树使用反向或正向链（或双向）进行搜索。“与/或”证明树（如果存在一个）将用OK节点做为根节点，事实（或者由用户决定是真的，或者列在一些事实数据库中）做为叶节点。根结点和叶结点通过规则连接（常常通过中间节点）。

用前述反向链中的规则，一个OK证明搜索树的上面部分如图17-8所示。用户建立OK的目标或者通过证明BAL和REP，或者证明COLLAT、PYMT和REP中的每一个来实现。证明OK的两种可选方式被两个OK节点表示在根节点下面。回忆在一个“与/或”树中被称为“或”的节点。在一个“与/或”树中，一个有“或”后继节点的节点能通过证明这些后继中的任何一个来证明。应用图示的其他规则，以证明其他的节点集。

为了本例的目的，我们假定通过询问数据库或直接让推理系统询问贷款用户，能建立BAL、RATING、APP和INC的真值（或假值）。另外，OK、PYMT、COLLAT和REP指称对推理有用的概念，但是关于这些用户和数据库却没有明确的信息（如果用户已经有这些概念的知识，他或她将不需要专家系统！）。在全面的专家系统中，可以尝试几个可选的规则来连接中间节点和叶节点。

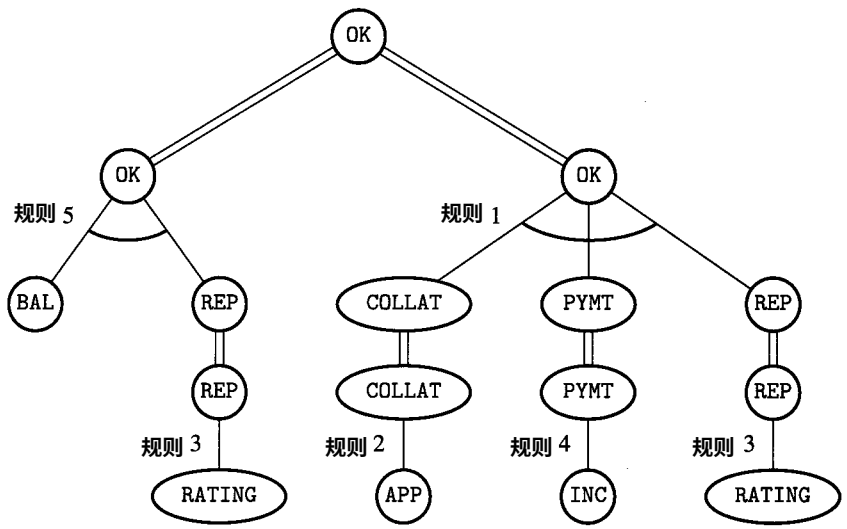


图17-8 一个证明的搜索树

如果按深度优先方式进行证明，对用户或数据库系统的查询能与搜索交替进行。例如，如果在搜索过程的早期，节点BAL被发现不为真，那么搜索将用规则1继续。顺着规则1、2、3和4向前进行，搜索过程最终产生请求查明APP、INC和RATING。如果所有这些节点的真值能被建立（从数据库或用户），那么我们证明了OK。当然，在实际应用中，证明树将更深更宽。

咨询系统通过问一些用户或数据库可能知道的命题的真值来设法回答用户的询问。通过规

则的反向链得到要回答的问题。如果一个用户想要“无偿”的信息，自底而上，通过规则的正向链能用来试图连接已经建立的证明树。

可以用同样的例子来描述系统的推理如何解释给用户。给出一个结论的解释能力对接受专家系统建议是非常重要的。证明树用来指导解释生成过程。再次引用图 17-8，在下面的对话中可以直接看到系统的反应是如何产生的（被推理系统使用的每个原子有一些与它相关的适当句子）。

用户：为什么你相信贷款应该被批准？

系统：因为贷款抵押被满足，并且申请人能做出贷款偿还，申请人有好的金融信誉。

用户：为什么你相信申请人有良好的金融信誉？

系统：因为信贷等级服务说那个申请人有良好的信贷等级。

用户：为什么你问我是否申请人的收入超过他/她的消费？

系统：因为我正试图建立是否申请人能够做出贷款偿还。

用户：为什么你要建立是否申请人能够做出贷款偿还？

系统：因为这是建立是否贷款应该被批准的标准之一。

用户：为什么你要建立贷款应该被批准？

系统：因为你问我。

在很多应用中，系统仅对不确定规则有访问取权，用户也不能确定地回答问题。例如，适于在计算机可理解的形式下编码的以下规则，被用在 MYCIN系统中[Shortliffe 1976]，该系统是诊断细菌感染的：

规则300

如果：

1) The infection which requires therapy is meningitis, and

2) The patient does have evidence of serious skin or soft tissue infection and

3) Organisms were not seen on the stain of the culture ,and

4) The type of the infection is bacterial

那么：

There is evidence that the organism (other than those seen on cultures or smears) which might be causing the infection is staphylococcus-coag-pos (.75); streptococcus-group-a (.5).

同样，下面的规则被用在PROSPECTOR系统[Duda Gaschnig, & Hart 1979 ,Campbell,et al. 1982]，它是对矿石沉淀进行推理的：

If there is a pre-intrusive, thorough-going fault system ,then there is (5, 0.7) a regional environment favorable for a porphyry copper deposit.

数字（在MYCIN中的0.75和0.5，在PROSPECTOR中的5和0.7）是表达一个规则的可信度和强度的一种方式。它们被用来计算结论的可信度。被MYCIN与PROSPECTOR使用的相似技术已被应用，并产生了良好的效果，第19章将介绍更复杂更深奥的技术以解决不确定性问题。

17.5 规则学习

既然已经知道基于规则的系统的重要性和从专家那里提取好的规则的艰难，那么很自然地，我们会问专家系统规则能否被自动学习。有两种类型的学习方法，归纳法和演绎法。两种类型

都能用于学习规则。例如，神经网络学习就是归纳法，因为学习的函数是对潜在的、未知的函数的假设。在成功的学习中，一般假设会给出大部分输入的正确输出，但是它们也可能出错。归纳规则学习方法建立一个领域的新规则——不能从任何以前的规则推导出。将介绍用于在命题和谓词演算中介绍归纳规则学习的方法。

演绎规则学习通过从以前知道的领域规则和事实推理附加的规则来扩展一个系统的性能效率。不用附加规则，系统也能推出使用附加规则推出的结论。但是使用附加规则，系统可以更有效地执行。对演绎附加规则，将解释一个叫做基于解释的广义化（EBG）技术。

17.5.1 学习命题演算规则

已经提出了几种归纳规则学习的方法，这里将描述其中一种[㊀]。首先描述Horn子句命题逻辑的一般思想，然后给出一个相似的、可以用来学习一阶逻辑Horn子句规则的技术。

再次使用批准银行贷款的简单例子。不再提供该问题的规则，而是假设给出了一个训练集合，它由大量的个人属性值构成。为了说明，考虑表 17-1中给出的数据（用1代表T，0代表F）[㊁]。例如，这个表可以从贷款应用记录和贷款员所做结论来收集。训练集中OK值为1的成员称为正例，OK值为0的叫反例。从训练集中，我们希望归纳出如下规则：

$\alpha_1 \wedge \alpha 2 \wedge \ldots \wedge \alpha_n$ OK

其中α_i是集合{APP，RATING，INC，BAL}中的命题原子。如果一规则的前项在训练集中有实例T值，我们说那个规则覆盖（*cover*）那个实例。我们能通过给规则的前项加一个原子来改变任何现存的规则，使之覆盖更少的实例。此改变使那个规则更特殊。两个规则比单个规则能覆盖更多的实例。增加一个规则使得使用这些规则的系统更为通用。我们寻找一个规则集合，它仅覆盖训练集中的所有正实例。

表17-1 银行数据

例子	APP	RATING	INC	BAL	OK
1	1	0	0	1	0
2	0	0	1	0	0
3	1	1	0	1	1
4	0	1	1	1	1
5	0	1	1	0	0
6	1	1	1	0	1
7	1	1	1	1	1
8	1	0	1	0	0
9	1	1	0	0	0

对一个规则集合的搜索在计算上是困难的。这里描述了一个“贪婪”方法，亦称为分治法（*separate and conquer*）[㊂]。首先设法找到一个规则，它仅仅覆盖正例——尽管它不覆盖所有的正例。我们从一个覆盖所有实例（正例和反例）的规则开始来对这样一个规则进行搜索，通过加原子到它的前项，逐渐使它更特殊。由于一个单一规则不可能覆盖所有的正例，我们逐渐

㊀ 规则也能从决策树（见Quinlan 1993，第5章）和神经网络（见[Towell & Shavlik 1992, Towell, Shavlik & Noordweier 1990]）中提取。

㊁ 当然，在实际应用中，我们需要比这里给出的多得多的数据。

㊂ 该方法由[Michalski 1969]提出，由一系列基于Michalski 的AQ 算法检查过。

增加规则（使它们像我们需要的一样特殊）直到整个规则集仅覆盖所有的正例。

下面说明如何使用该方法。首先从覆盖所有实例的临时规则

T OK

开始。现在，我们必须加一个原子使它包含较少的反例——朝着仅覆盖正例的方向前进。我们应该加哪一个原子（从集合{APP，RATING，INC，BAL}中）呢？有几个标准可用来做这个选择。为简单起见，决定基于一个容易计算的比率：

$$r_\alpha = n_\alpha^+ / n_\alpha$$

其中，n_α是将α原子加到规则的前项后，被（新的）前项覆盖的（正的和反的）实例的总量，n_α^+是将α原子加到规则的前项后，被（新的）前项覆盖的正例的总量。

我们选取产生最大r_α值的α值。在例子中，该值是：

$r_{App} = 3/6 = 0.5$

$r_{RATING} = 4/6 = 0.667$

$r_{INC} = 3/6 = 0.5$

$r_{BAL} = 3/4 = 0.75$

因此，我们选BAL，产生临时规则 ：

BAL OK

这个规则覆盖了正例3、4和7，但也覆盖反例1，因此必须把它进一步特殊化。用同样的技术选择另一个原子。r_α的计算必须考虑到我们已经决定了前项中的第一个部件是BAL：

$r_{App} = 2/3 = 0.667$

$r_{RATING} = 3/3 = 1.0$

$r_{INC} = 2/2 = 1.0$

其中，在RATING和INC之间有一个平局。我们可能选择RATING，因为r_{RATING}基于一个更大的取样（你应该研究一下选择INC的结果）。

规则BAL∧RATING OK仅覆盖正例，故不必再加原子到这个规则的前项。但是这个规则没有覆盖所有的正例，尤其他不覆盖正例6。因此，我们必须增加另一个规则。

为了学习下一个规则，首先从表中删去被第一个规则覆盖的所有正例，获得表17-2中所示的数据。我们用规则T OK，从这个已精减过的表开始，重复上述的所有过程。这个规则覆盖一些反例1、2、5、8和9。为了选择一个原子加到前项，我们计算

$r_{App} = 1/4 = 0.25$

$r_{RATING} = 0/3 = 0.0$

$r_{INC} = 1/4 = 0.25$

$r_{BAL} = 0/1 = 0.0$

表17-2 被减的数据

个例	APP	RATING	INC	BAL	OK
1	1	0	0	1	0
2	0	0	1	0	0
5	0	1	1	0	0
6	1	1	1	0	1
8	1	0	1	0	0
9	1	1	0	0	0

又有一个平局。任意选择APP给出规则APP OK。这个规则覆盖反例1、8和9，因此我们必须加其他的原子到前项。r的值为：

$r_{RATING} = 1/2 = 0.5$

$r_{INC} = 1/2 = 0.5$

$r_{BAL} = 0/1 = 0.0$

选择RATING给出规则APP∧RATING OK，这个规则覆盖反例9。使这个规则更特殊（用通常的方式），最终产生规则APP∧RATING∧INC OK。这两个规则，BAL∧RATING OK和APP∧RATING∧INC OK，覆盖了所有正例，且仅覆盖正例。因此我们完成了任务。

由于这个发现规则的过程使用了贪婪搜索，学习的规则有时能被简化就不惊奇了。对每个规则，通过测试来看在不改变其余规则在训练集上做出的决定的情况下，规则是否能被遗弃。如果没有任何影响（或者数据中有噪声时，对精度有很小的影响），那个规则可以被删除。同样，对规则中的每个原子，我们测试那个原子能否被移走而只有最小的影响。确实，如果数据有噪声，我们宁愿修改学到的规则仅覆盖所有正例的标准。我们可能允许每个规则覆盖“主要的”正例——允许（当必须考虑噪声数据时）每个规则包含少量的反例。同样，学到的规则集可以允许未能覆盖少量的正例。这些“修剪”操作和噪声容限修改有助于把过高的冒险最小化。

用伪代码简洁地描述这个规则学习过程，称之为通用分治算法（GSCA）。在算法中：

Ξ 是初始的二值化特征训练实例集，每个特征用一个原子γ的值标识；

π 是将要学习的规则集；

ρ 是规则之一；γ是它的后项，（原子的合取）Γ是它的前项；

α 是从 Ξ 中的特征之一提取的原子。

通用分治算法GSCA：

1) 初始化 $\Xi_{cur} \leftarrow \Xi$。

2) 初始化π← 空的规则集。

3) repeat 外部循环加入规则直到π覆盖所有（或大部分）正例。

4) 初始化 Γ← T。

5) 初始化ρ←Γ γ。

6) repeat 内部循环把原子加到Γ中，直到ρ仅覆盖（或主要）正例。

7) 选择一个原子α加到Γ，这是一个用于回溯的非确定性选择点。

8) Γ←Γ∧α。

9) until ρ仅包含（或主要）Ξ_{cur}中的正例。

10) π←π, ρ。把规则ρ加到规则集。

11) $\Xi_{cur} \leftarrow \Xi_{cur}$ —— (Ξ_{cur}中被π包含的正例)。

12) until π覆盖所有（或大部分）Ξ中的正例。

在银行贷款例子中，学到的规则与最初“专家”为这个问题给出的规则是一致的。然而，用这样一个小的训练集学到的特殊规则可能与一个专家对那个问题的直觉判断不一致（例如，如果平局中我们选用不同的项，就会学到不同的规则）。对学到的规则和专家给出的规则进行比较，揭示了学到的规则不会提到专家没有估量或问到的任何谓词（COLLAT、PYMT和REP），它们没有被数据中的特征表示。中间谓词通过封装更原始谓词的一般出现组简化了知识表示和推理。学习这些中间谓词是一个重要的研究课题。关于“谓词发明”已经做了一些初步的工作

(例如，见[Muggleton & Buntine 1998]。即使没有产生涉及中间谓词的规则，规则学习常常也能被用来加速专家系统的建立过程。

17.5.2 学习一阶逻辑规则

上面描述的规则学习技术产生命题逻辑中的规则。虽然很多专家系统建立在命题逻辑的基础之上，但是更全面的专家系统是通过使用带有通用量化变量的规则而得到的。归纳逻辑编程(ILP)的子领域集中于FOPC(因此与PROLOG程序一样)中Horn子句的归纳学习方法。已经开发了学习Horn子句的几个方法——有些对能产生的Horn子句的类型有一定的限制。对这个主题的全面解决远远超过了这本书的范围，但是本书简短地构画出使用分治方法的一个有代表性的ILP技术。描述的方法是基于一个叫FOIL[Quinlan 1990.]的系统(关于本领域的全面和易读的论述，参见[Lavrač & Džeroski 1994]。[Muggleton 1992]是一本ILP论文集，[Muggleton & De Raedt 1994]是一个标准的参考)。

为了使讨论符合大部分归纳逻辑编程(ILP)著作习惯，本书使用类似PROLOG的符号，然而为了与FOPC约定相一致，继续用大写字母表示常量，小写字母表示变量符号。在归纳逻辑编程中，我们的目标是学习由Horn子句ρ构成的程序π，每个ρ的形式是ρ：-α_1，α_2，...,α_l，其中α_i是与基本原子事实合一的原子公式。其思想是当π的变量被限制到一些值集(已知它们是在我们试图学习的关系中)时，它的值应该评估为真；这些值是训练集合的正例Ξ^+。当π的变量被约束到不在关系中的值时，π的值应该评估为假，这些值集是反例Ξ^-。就像在学习命题规则中一样，我们想让π覆盖正例而不覆盖反例。将与α合一的基本原子事实构成后台知识(*background knowledge*)，假定它们已被给出——以能被运行和评估的补充PROLOG程序的方式，或者显式地用事实列表的形式。

做为一个例子，假设在一个大楼里运送东西的一个机器人通过经验发现，在某些位置对之间走动是容易的，但要走到其他位置对去就不是那么容易了，我们在图17-9中显示了这样一个大楼的部分地图。在这个大楼中，位置*A*、*B*和*C*是交叉点(*junction*)，所有其他的位置是商店。

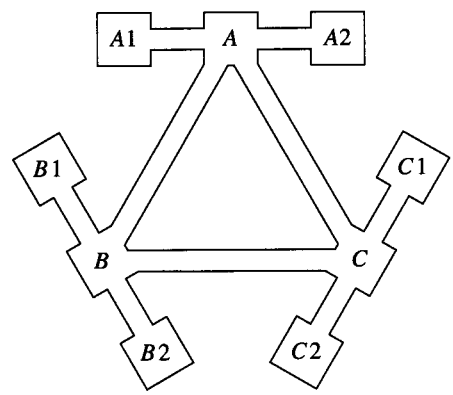


图17-9 一个建筑的部分图

假定机器人已经汇编了位置对的一个训练集Ξ，每对位置都被标识了，不管它是否容易导航。假定机器人有关于这些位置属性和位置之间关系的一些信息。特别地，对任何位置，我们知道它是不是一个交叉点；对任何位置对，知道它的一个成员是不是连向一个交叉点的商店。这些关系用表达式(Junction(x)和Shop(x, y))表示。我们想让一个学习程序学习一个程序Easy (x, y)，它覆盖Ξ中的正例但不覆盖反例。Easy (x, y)能使用后台子表达式Junction (x)和Shop(x, y)。

为了使我们例子具体化，假定训练集由下面的Easy (Ξ^+)正例：

$$\{<A,B>,<A,C>,<B,C>,<B,A>,<C,A>,<C,B>,$$
$$<A,A1>,<A,A2>,<A1,A>,<A2,A>,<B,B1>,<B,B2>,$$
$$<B1,B>,<B2,B>,<C,C1>,<C,C2>,<C1,C>,<C2,C>\}$$

和下面的Easy(Ξ)反例：

$$\{<A,B1>,<A,B2>,<A,C1>,<A,C2>,<B,C1>,<B,C2>,$$
$$<B,A1>,<B,A2>,<C,A1>,<C,A2>,<C,B1>,<C,B2>,$$
$$<B1,A>,<B2,A>,<C1,A>,<C2,A>,<C1,B>,<C2,B>,$$
$$<A1,B>,<A2,B>,<A1,C>,<A2,C>,<B1,C>,<B2,C>\}$$

构成。

假定机器人能对Junction(x)和Shop(x, y)的任何变量值进行一段评估。特别地，对Ξ中命名的所有位置，只有集合{A，B，C}会给Junction一个真值。合并成一段在Ξ中提到的所有其他位置都给Junction一个假值。

对Ξ中命名的所有位置，只有下面（排序的）位置对给Shop真值：

$$\{<A1,A,>,<A2,A>,<B1,B>,<B2,B>,<C1,C>,<C2,C>\}$$

Ξ中提到的所有其他位置对Shop都有假值。

下面的PROLOG程序正好覆盖了训练集中的所有正例，且不包含反例：

```
Easy(x, y) :- Junction(x), Junction(y)
           :- Shop(x, y)
           :- Shop(y, x)
```

就像其他的学习问题一样，我们希望推理程序能具有一般性，即，如果表达用到的一些参数没有在训练集中出现（但对需要的后台关系，我们有它们的值），我们想让程序做出好的猜测（想像大楼里未在训练集中表达的其他地面和侧楼具有重复的结构）。

待解释的学习过程使用了一种GSCA算法。从一个没有体的单个规则的程序开始，给规则体增加文字直到它仅（主要）覆盖正例，然后用同样的方式增加其他规则，直到程序覆盖所有（或大多数）（有很少的例外）正实例（附加的放松办法可应用于数据有噪声的情况）。

我们有无限个可能文字能加到一个子句体中。实际的ILP系统用各种方式约束文字。典型的允许增加的条件是：

1) 文字要用在后台知识中。

2) 文字的参数是子句头中参数的子集。

3) 文字要引入一个新的独特的变量，它不同于子句头中的变量。

4) 一个文字使子句头中的一个变量等同于另外一个这样的变量或者一个在后台知识中提到的项。（这种可能性等价于通过一个置换形成一个特例）。

5) 一个文字要和子句头中的文字一样（除了它的参数）（这种可能性允许递归程序，在一些系统中不允许这种情况，在此我也不谈论它）。

在机器人导航的例子中，考虑加到一个子句中的文字是：

```
Junction(x)
Junction(y)
Junction(z)
shop(x, y)
shop(y, x)
shop(x, z)
shop(z, y)
```

(x=y)

遵从这里讨论的方法（通过加文字使子句具体化）的ILP系统会有很好定义的方法，以便计算可能加到一个子句的文字。选择加入哪个文字是更复杂的事情；大多数系统使用类似于在命题逻辑规则学习例子中的记分方法。

使用机器人导航的例子，演示了GSCA的ILP方法如何工作。知道谓词Easy是一个二位谓词，算法的内部循环把第一个子句初始化为Easy（x, y）:-。这个子句包含所有的训练实例（正的和反的），因此我们必须加一个文字到它的（空）体。为了演算算法而不考虑如何选择一个要加入的原子公式，假定算法加入Junction(x)。Ξ中下面的正例被Easy(x, y):- Junction(x)覆盖：

$$\{<A,B>,<A,C>,<B,C>,<B,A>,<C,A>,<C,B>,$$
$$<A,A1>,<A,A2>,<B,B1>,<B,B2>,<C,C1>,<C,C2>\}$$

为了计算这个覆盖，使用后台关系Junction给出的城市作为基本事实，为Ξ中所有的城市对解释逻辑程序Easy(x, y):- Juntion(x)。下面的反例也被覆盖：

$$\{<A,B1>,<A,B2>,<A,C1>,<A,C2>,<C,A1>,<C,A2>,$$
$$<C,B1>,<C,B2>,<B,A1>,<B,A2>,<B,C1>,<B,C2>\}$$

因此，必须加入另一个文字。假定接下来加入 Junction(y)。下面的正例被Easy(x, y):- Junction(x), Junction(y)覆盖：

$$\{<A,B>,<A,C>,<B,C>,<B,A>,<C,A>,<C,B>\}$$

不再有任何Ξ中的反例被覆盖，因此我们用子句Easy(x,y):- Junction(x),Juntion(y)终止内部循环的第一次循环。

但是仅由这个子句组成的程序π不能覆盖下面的正例：

$$\{<A,A1>,<A,A2>,<A1,A>,<A2,A>,<B,B1>,<B,B2>,$$
$$<B1,B>,<B2,B>,<C,C1>,<C,C2>,<C1,C>,<C2,C>\}$$

被Easy(x, y):- Junction(x), Junction(y) 覆盖的正例从Ξ中删除，形成下次内部循环中要用的Ξ_{cur}。Ξ_{cur}由Ξ中所有的反例以及还没被覆盖的正例（前面列出的）构成。为了设法覆盖它们，内部循环建立另一个子句，首先赋予Easy(x, y):-。这个子句覆盖所有的反例，因此必须加文字，假定我们加入Shop(x, y)。子句Easy(x, y):- Shop(x, y)不覆盖任何反例，因此我们完成了另一个内部循环。

Easy(x, y):- Shop(x, y)覆盖Ξ_{cur}中的下述正例：

$$\{<A1,A>,<A2,A>,<B1,B>,<B2,B>,<C1,C>,<C2,C>\}$$

将这些实例从Ξ_{cur}中移走，并为下一次通过内部循环做准备。程序现在包含两个子句：

```
Easy(x, y) :- Junction(x), Junction(y)
              :- Shop(x, y)
```

由于没有覆盖下述的正例，该程序仍然不合适：

$$\{<A,A1>,<A,A2>,<B,B1>,<B,B2>,<C,C1>,<C,C2>\}$$

在下一次通过内部循环后，我们加入子句Easy(x, y):- Shop(y, x)。现在程序包含所有

的、且都是正的实例。因此过程终止于

```
Easy(x, y) :- Junction(x), Junction(y)
           :- Shop(x, y)
           :- Shop(y, x)
```

这个程序能应用于（也许以好的一般化）除了在 Ξ中命名以外的其他位置 —— 只要我们能对这些其他的位置评估Junction和Shop的关系就行。

17.5.3 基于解释的一般化

现在转向演绎学习，一种从以前的规则和事实推导规则的方法。用一个积木世界的例子解释这个方法。假定我们关于这个世界的一般知识包括下述规则：

```
Green(x) ⊃ Heavy(x)
Heavy(x) ⊃ ¬Pushable(x)
```

关于这个世界的事实之一是：

```
Green(A)
```

假定要求我们证明 ¬ Pushable(A)。在这个演示例子中，证明是简单的。然而在实际问题中它可能更复杂。在图17-10中显示了归结反驳。

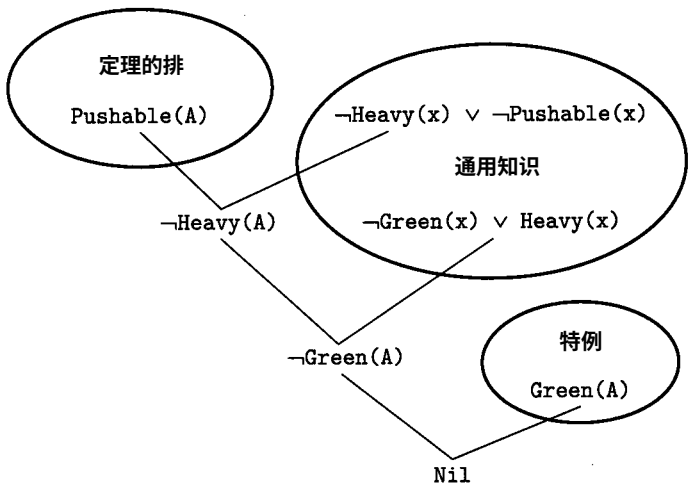


图17-10 一个用在EBG中的归结反驳

根据证明，我们能对结论构造一个解释。一个解释是用在证明中的事实集合。在这种情况下，对¬Pushable(A)的解释是Green(A)。用这个解释，我们能构造一个新规则：

```
Green(A) ⊃ ¬Pushable(A)
```

但是这个规则的应用十分有限，因为它只适用于积木 *A*。然而，我们注意到证明能被一般化 —— A可以是任何东西，证明将仍然成立。把常量 A用变量 x代替使解释一般化，产生 Green(x)。仿照 ¬Pushable(A)的证明结构，不用附加的搜索，我们看到用 Green(x)代替Greeen(A)，能证明 ¬Pushable(x)。那么，这个过程的结果，即我们称为基于解释的一般化(EBG)的结果是生成规则

```
Green(x) ⊃ ¬Pushable(x)
```

这个规则可以单独由一个一般规则建立，因此我们并没有学到任何新东西。但是 EBG过程可以指导我们从基于需要的规则去建立适用于特殊情况的规则。对更全面的 EBG讨论，参见[Minton, et al. 1989]。

在执行EBG的过程中，我们假定一般规则对其他相似的情况也是有用的，因此值得推导和保存。当然，更多的规则会随着分枝因子的增加而减慢推理过程。因此 EBG必须小心使用——可能要通过保持有关学到的规则的实用（*utility*）信息而获得。Minton按照由规则提供的搜索工作的平均节省、规则被使用的频繁程度及决定规则是否可应用的代价来定义实用 [Minton 1988, Minton 1990]。他的PRODIGY系统只保留高实用的规则。

17.6 补充读物和讨论

[Levesque & Brachman 1987]讨论了在逻辑表示的可表达性和逻辑推理方法的易处理性之间的权衡。[levesque 1986]指出推理的难处理是由于在逻辑语言中存在析取和否定。他建议用鲜明的表示省却这些结构。如果不包含函数符号，只使用 Horn子句的表示（称为 Horn 理论）是易处理的。这些约束在一个叫 DATALOG的语言中可以看到，DATALOG用来增加数据库的信息[Ullman 1989]。[Gogic et al. 1995]对各种命题逻辑和它的非单调扩展性的可表达性 、易处理性和简洁性进行了比较。

[Selman & Kautz 1991]提出使用“近似理论”加速某些推理过程。它们是 Horn最大较低约束（GLB）和Horn最低较高约束（LUB）理论。如果一个公式不遵循一个LUB理论，它就不遵循原始理论。另一方面，如果遵循一个 GLB理论，它也遵循原始理论。[Kautz , Kearns , & Selman 1993]建议使用特征模型作为一个可行的方式对Horn理论进行推理。

下一章将阐述术语逻辑（*terminological logic*）——通过约束表达语言获得易处理性的另一种方式。

作为对约束语言的一个可选方法，我们可以使用近似推理方法——它可能不是合理的和完备的。这方面的例子有随机模型采样[Khardon & Roth 1998]。

使用一阶逻辑（和归结方法）作为一门编程语言（和它的解释器）的第一个建议要归功于[Green 1969a]。[Kowals ki 1974]独自详细研究了这个思想，并激发了 PROLOG的发明。Alain Colmerauer开发出了PROLOG的第一个解释器[Roussel 1975,Colmerauer 1973]。一个有效的实现归功于[Warren,Pereira,& Pereira 1977]。关于这个主题的主要论文期刊是《Journal of Logic》。

“与/或”图结构是求解问题方法的基础，该方法把问题简化为子问题的可选集合（就像在通过规则向后推理中一样）。[Nilsson 1980 第3章]介绍了搜索“与/或”图的一个算法AO*，它基于 [Nilsson 1969,Martelli & Montanari 1973] 的早期工作。[Davis 1980]描述了使用元规则控制基于规则的专家系统“与 /或”树结构上的搜索，也可参考 [smith,Genesereth，& Ginsberg 1986, Smith 1989]。

决定最小的ATMS标记是NP完全问题[Selman & Levesque 1990],即使在后台理论仅包含Horn子句的情况下。然而，[Kautz, Kearns, & Selman 1993]证明了这样的标记能在多项式时间内被计算——如果计算是基于特征模型的。其他有关 TMS的著作，参见[Doyle 1979,de Klear 1986a, de Kleer 1986b, de Kleer 1986c, Forbus & de Kleer 1993，Shoham 1994]。

作为对专家系统的建立过程的一个概述，可参见 [Bobrow,Mittal, & Stefik 1986]。[Stefik 1995]是一本涉及知识库系统建立的很多方面的课本（它也包含有关 AI的很多其他材料）。尤其

有用的是Stefik的注释参考文献，它里面包含了该领域重要论文的概括短文（附录A）。

专家系统的一个例子是配置 DEC公司的VAX-11/780系统，参见[McDermott 1982]。[Leonard-Barton 1987]介绍了专家系统在DEC公司的开发历史和使用。有关应用方面正在进行的工作在人工智能创新应用（IAAI）的年会论文集中有报告，有关专家系统的论文定期出现在IEEE的专家系统中。

关于谓词（概念）创新的更多情况，参见[Kautz & Selman 1992]和[Muggleton & Buntine 1988]。ILP应用的一个给人印象深刻的例子是在GOLEM系统中[Muggleton, King, & Sternberg 1992]蛋白质二次结构的预测。国际归纳逻辑编程研究组的学报包含该主题的当前论文。

知识表示和推理（KRR）国际会议论文集包含了描述当前研究成果的论文。[Brachman & Levesque 1985]是关于知识表示的重要论文集。

习题

17.1 下面的问题与PROLOG有关：

1) 解释一下为什么一个运行PROLOG程序的PROLOG解释器不能用来证明一个原子的否定。

2) 有些推理系统（如PROLOG），当它们不能证明原子ϕ时，就得出结论$\neg\phi$。这种否定类型被称为失败否定。解释一下为什么一般地我们不能因证明P为假而得出$\neg P$的结论。

17.2 如本章所描述的，PROLOG解释器在每一步执行一个指定的归结。如果一个PROLGO程序中的所有语句能被转化为文字的析取，描述一个归结反驳系统的控制策略，该系统能像PROLOG解释器一样执行同样的归结。

17.3 Tom声称是Paul Revere的后代。检验Tom声明的一个较容易方式是：通过说明Revere是Tom的祖先之一（向后搜索）或说明Tom是Revere的后代之一（向前搜索）？为什么？你的答案有例外吗？

17.4 在习题16.12描述的电路中，假如$W1$、$W2$和$W4$是“on”。描述一下如何用一个ATMS系统来诊断这个电路的可能故障。

17.5 一个归纳学习系统观察到每次Q为真时，下面的公式之一也是真：

P(A，B)

P（C，B）

P（D，B）

这个学习系统决定建立规则（ x）（P（x, B） Q）

这个归纳一般化的合理性是什么？为什么系统不建立规则(x, y)(P(x, y) Q)?

17.6 一个专家系统被用来评估人们的信用风险，它基于大量事实，这些事实组合在一起产生一个数字分数。为了被判断为一个好的信用风险，分数必须比某个N值还要大。专家系统的设计者没有说N值是什么，但我们已经知道Pete被判断为是一个好的信用风险，Joe的分数比Pete的高。通过用下面的领域规则：

$\forall(x, y, z)[\mathtt{Greater}(x, y) \wedge \mathtt{Greater}(y, z) \supset \mathtt{Greater}(x, z)]$

$\forall(x)[\mathtt{C}(x) \supset \mathtt{Greater}(\mathtt{score}(x), \mathtt{N})]$

$\forall(x)[\mathtt{Greater}(\mathtt{score}(x), \mathtt{N}) \supset \mathtt{C}(x)]$

和下面的数据：

C(Pete)

Greater(score(Joe), score(Pete))

我们能证明Joe也会被判断为是一个好的信用风险。

首先，用这些公理和数据证明 Joe是一个好的信用风险，然后用基于解释的一般化技术对你的证明建立规则

$[\forall(x, y)\{[\text{Greater}(\text{score}(x), \text{score}(y)) \wedge C(y)] \supset C(x)\}$