Selected Foreign Text Books

外国教材精选

Prolog 高级程序设计

[美] L. 斯特林
E. 夏皮罗 著
刘家佺 邓 佑 译
郑 宇 淇 校

西安交通大学出版社

外国教材精选

Prolog
高级程序设计

〔美〕L. 斯特林
E. 夏皮罗 著

刘家佺 邓 佑 译
郑守淇 校

西安交通大学出版社

内 容 简 介

本书为我社出版的"外国教材精选"系列书之一。

该书主要介绍了逻辑程序、Prolog 语言、纯 Prolog 及其程序设计、高级 Prolog 程序设计技术以及 Prolog 的应用。重点介绍高级 Prolog 程序设计技术。书中附有大量的实用程序举例，并有较详细的说明。各章后还附有练习题供读者练习。

该书可作为计算机类大专院校师生和科技人员的重要参考书，对人工智能程序设计人员尤为适宜。

The Art of Prolog
——Advanced Programming Techniques
L. Sterling E. Shapiro
The MIT Press, Cambridge Massachusetts
London, England, 1986

Prolog 高级程序设计
[美] L. 斯特林 E. 夏皮罗 著
刘家俭 邓佑 译 郑守淇 校
责任编辑 赵丽平

*

西安交通大学出版社出版
邮政编码：710049
西安交通大学出版社印刷厂印装
陕西省新华书店经售
开本 787×1092 1/16 印张 18.5 字数：445 千字
1990 年 6 月第 1 版 1991 年 6 月第 2 次印刷
印数：3001——7000
ISBN7-5605-0249-0/TP·23 定价：6.30 元

“外国教材精选”总序

近十年来，我国高等学校教材建设在经历了从无到有、巩固提高的过程之后，目前正进入向高质量、高层次、多品种发展的欣欣向荣，百花争艳时期。现在，教材建设仍是高等学校教学改革的重要方面，这里也存在一个改革开放的问题。在这种形势下，精选国外一些有影响、有特色、特别是世界上著名大学现用的优秀教材翻译出版，无疑将对我国当前教材建设起到借鉴、促进和填补某些学科空白的积极作用。为此，西安交通大学出版社决定组织翻译出版一套“外国教材精选”系列书。

外国教材专业面广、类型繁多、层次各异，我们这套系列书在选题时以专业面较广，内容新颖或具有明显特色的教材为目标。具体原则如下：

1. 列选的教材不限国别及语种，以便博采众长。
2. 国外著名经典性教材，多次修订重版经久不衰者。
3. 最新出版，为国外著名大学所采用，有独特风格、体系，能反映国外教育动向，可供借鉴者。
4. 反映最新科技成果，能填补国内某学科教材空缺者。

根据我校具体情况，“外国教材精选”系列书将以电类教材（含电力、电子、计算机与信息科学）为主，今后随着形势的发展和需要，再进一步组织其他学科的国外先进教材翻译出版。

我们期望这套系列书，不仅是高等学校的学生和教师的良师益友；而且对已在生产科研第一线的广大科技工作者的知识更新，吸取国外科技新成果方面也大有裨益。

这套“外国教材精选”虽然从搜求原著、遴选、翻译、审校等方面都做了较细致的工作，但从浩如烟海的外国教材中精选少数形成一套系列书，对我们毕竟还是一种尝试。书源还不够充分，经验也感不足，缺点在所难免，诚挚地希望读者予以指正。

西安交通大学“外国教材精选”编委会

1988年6月

译 者 的 话

自日本宣布将 Prolog 作为新一代计算机的核心语言以来，国内外计算机界对逻辑程序设计和 Prolog 语言，产生了浓厚的兴趣。人们逐渐感觉到，用 Prolog 不仅可以解决用过程型语言难以解决的实际问题，而且它也是人工智能理论研究的重要工具。

西安交通大学出版社出版的《计算机解题逻辑》（[英] R. 科瓦尔斯基著，郑守淇译），论述了逻辑学在问题求解和计算机程序设计方面的应用。经郑守淇教授推荐，作为上面这本书的姐妹篇，我们译出了《Prolog 高级程序设计》一书。

郑守淇教授以他渊博的学识、娴熟的外语，从选定题材，审查试译稿到审订全书，都付出了辛勤的劳动。没有他的关怀、指导，要译出这本题材新颖的专著是难以想象的。为帮助读者了解实际系统及使用情况，由译者另外搜集了一些材料作为附录放在后面供读者参考。

本书第 1—14 章以及第 19 章由刘家佺译，第 15—18 章以及第 20—23 章由邓佑译，最后由刘家佺修改统一了全书。

西安交通大学出版社杨莳百，赵丽平等同志为本书的出版做了大量细致的工作，提出了很多宝贵意见，对此深表谢意。

译 者

1988年4月

目　录

第Ⅱ部分 Prolog语言

第Ⅳ部分 应用

第Ⅰ部分　逻辑程序

第1章　基本结构

逻辑程序设计的基本结构是从逻辑沿袭而来的项和语句。逻辑程序有三种基本语句：事实、规则和询问。数据结构只有一种：即逻辑项。

1.1　事　　实

最简单的一种语句称为事实。事实是说明对象之间关系的方法。例如：

```
father(abraham,isaac).
```

这个事实是说 Abraham 是 Isaac 的父亲，或者说名为 Abraham 和 Isaac 的个体之间存在着父子关系。关系的另一种名称叫谓词。个体的名称就叫原子。同样地，plus (2,3,5) 表示 2 加 3 等于 5 这个关系。大家熟悉的 plus 关系可以通过一组事实的集合来实现，这些事实定义了加法表。这个表的初始部分为

```
plus(0,0,0).    plus(0,1,1).    plus(0,2,2).    plus(0,3,3).
plus(1,0,1).    plus(1,1,2).    plus(1,2,3).    plus(1,3,4).
```

在这一章中，假如把这张表的充分大的一段作为关系 plus 的定义，它正好也是合法的逻辑程序。

贯穿本书，句法上的约定将根据需要引入。首先是字母大、小写的约定。事实中谓词和原子的名字以小写字母开头是有意义的，不允许以大写字母开头。

```
father(terach,abraham).          male(terach).
father(terach,nachor).           male(abraham).
father(terach,haran).            male(nachor).
father(abraham,isaac).           male(haran).
father(haran,lot).               male(isaac).
father(haran,milcah).            male(lot).
father(haran,yiscah).

                                 female(sarah).
mother(sarah,isaac).             female(milcah).
                                 flmale(yiscah).
```

程序　1.1

有限的事实集合构成了一个程序，这是逻辑程序最简单的一种形式。事实集合还可以是一种情况的描述。这种认识是设计数据库的基础，这将在下一章讨论。程序 1.1 根据圣经给

出了一个家族关系数据库。谓词 father、mother、male 和 female 表达了明显的关系。

1.2 询　　问

逻辑程序中语句的第二种形式是询问。询问是从逻辑程序中检索信息的手段。询问是问对象之间是否具有一种特定的关系。例如，询问 father(abraham, isaac)？就是问 abraham 和 isaac 之间是否有 father 关系。根据程序 1.1 的事实，这个询问的回答是 yes 。

就语法而言，询问和事实差不多，但可以通过上下文区别，当可能混淆时，用句号结尾表示事实，而句子结尾用问号则表示询问。我们把没有句号也没有问号的实体称为目标。事实 P. 说明了目标是真实的。询问 P？则是问目标 P 是否为真。简单的询问由单一的目标组成。

对一个程序来说，回答询问就是确定这个询问是不是该程序的逻辑推论。通过这一节我们逐步给逻辑推论下定义。逻辑推论可以通过使用演绎规则得到。最简单的演绎规则是同一性：由 P 推演出 P. 。询问是同一事实的逻辑推论。

从运算来看，用含有象程序 1.1 那样事实的程序回答简单的询问是很直接的。在隐含着询问的程序中搜索事实时，如果发现了与询问相同的事实，则回答 yes 。

如果没有找到与询问相同的事实，则回答 no 。 因为事实并不是这个程序的逻辑推论，这个回答也不反映询问的真实性，它仅仅说明我们不能根据这个程序证明此询问。对于程序 1.1, 询问 female(abraham)？ 和 plus(1,1,2)？ 将回答为 no 。

1.3 逻辑变元的代换和实例化

逻辑变元表示一个未被指定的个体，让我们看一下它在询问中的用法。假定我们想知道 abraham 是谁的父亲。一种方法就是提出一系列的询问，father(abraham,lot)？ father(abraham,milcah)？ ,…,father (abraham,isaac)？ ,… 直到有一个回答 yes 为止。使用变元使我们有一种比较好的表达方式：father(abraham,X)？,对于这样的询问，其回答是 X=isaac。如果将变元用于这种表达方式，则变元就是概括许多询问的手段。包含一个变元的询问就是问是否有这样一个变元的值，它能够使询问成为程序的逻辑推论，关于这一点下面将进一步讨论。

逻辑程序中变元所起的作用与传统的程序设计语言不同。它们代表一个未指定的，却是单一的实体，而不是存储器的存储单元。

因为已经定义了变元，所以我们可以定义项，项是逻辑程序中唯一的数据结构。定义采用归纳法。常数和变元是项，复合项（或结构）也是项。复合项包括一个函子（称为项的关键函子）和一个或多个变量的序列，这些参变量也可以是项。函子是由它的名字（它是一个原子）和若干个参变量决定的。语法上复合项具有 $f(t_1,t_2,\cdots,t_n)$这样的形式，这表示函子名为 f，而 t_i 是 n 个参变量。复合项的例子如 s(0)， hot (milk)， name (john， doe)， list(a,list(b,nil))， foo(X)， 和tree(tree(nil,3,nil)， 5,R)。

询问、目标以及更一般的项，当其中不出现变元时，则被称作基，而出现变元的则称为非基。例如 foo(a,b)是基，而 bar(X)是非基。

定义：代换是一个有限集（可能是空集），它是形为 $X_i=t_i$ 的对，其中 X_i 是变元，而 t_i 是项，且对 $i\neq j$ 则 $X_i\neq X_j$，对所有的 i 和 j，X_i 不出现在 t_j 中。

由一个对组成的代换{X=isaac}即为一例。代换可以施加到项中，把代换 θ 施加到项A中，其结果记为 $A\theta$，它是对于 θ 中每一对 X=t，通过用 t 替换 A中每次出现的 X而得到的项。

把{X=isaac}施加到项 father(abraham,X)的结果是项 father(abraham,isaac)。

定义：如果存在一个代换 θ 使得 $A=B\theta$，则A是B的实例。

根据此定义，目标 father(abraham,isaac)就是 father(abraham,X) 的一个实例。同样地，在代换为{X=sarah,Y=isaac}的情况下，mother(sarah,isaac)是 mother(X,Y)的一个实例。

1.4 存在询问

从逻辑上讲，询问中的变元被存在量词所限定，直观的意义是，询问 father(abraham, X)？可以读作："存在一个X使得 abraham 是X的父亲吗？"，更一般地，询问 $P(T_1,T_2,\cdots,T_n)$？，它包含变元 $X_1,X_2,\cdots,X_k$，可以读作："存在 $X_1,X_2,\cdots,X_k$ 使得 $P(T_1,T_2,\cdots T_n)$吗？"。为方便起见，通常忽略存在量词。

我们要介绍的下一个推理规则是概括：对于任意的代换 θ，存在询问 P 是其实例 $P\theta$ 的一个逻辑推论。事实 father(abraham,isaac) 隐含着存在一个X，使得 father(abraham,X)为真，即 X=isaac。

从操作上讲，用一个事实程序回答一个存在的、非基的询问就是寻找一个作为询问的实例的事实。其回答（问题的解）就是该实例。如果程序中没有合适的事实，则回答 no。

回答非基询问就是执行一种运算，它的输出是询问的实例。如果把代换用到询问中，那么代换就会导致解的实例，我们有时就用这样的代换表示实例。

一般地讲，存在询问也许有几个解。程序 1.1 说明了 Haran 是三个孩子的父亲，于是询问 father(Haran,X)？就有{X=lot}，{X=milcah}，{X=yiscah} 三个解。有多个解的另一询问如 plus(X,Y,4)，它是求两个数加起来等于 4。例如解是{X=0,Y=4} 和 {X=1,Y=3}等。注意，不同的变元X和Y对应着（可能）不同的对象。

上一个询问的一种有趣的变体是 plus(X,X,4)？，它要求加起来等于 4 的两个数是相同的，这个询问只有唯一的解，即{X=2}。

1.5 全称事实

变元在事实中也是有用的。假定所有圣经人物都喜欢石榴。在程序中就不必对每一个体都说明这一适当的事实：

```
likes(abraham,pomegranates).
likes(sarah,pomegranates).
⋮
```

而只要用 likes(X,pomegranates) 就可说明全部事实了。如果变元用于这种情况，则变元就

是概括许多事实的手段。事实 times(0,X,0)概括了 0 乘任何数得 0 这样的事实。

事实中的变元隐含着全称量词，它直观的意思是事实 likes(X,pomegranates)，说明对于所有的X，X都喜欢石榴。一般地 $P(T_1,T_2,\cdots,T_n)$读作如果 X_i 是出现在事实 $P(T_1,T_2,\cdots,T_n)$中的变元，那么对于所有的 $X_1,X_2,\cdots,X_k$，$P(T_1,T_2,\cdots,T_n)$为真。从逻辑上讲，从一个全称事实能推演出它的任何一个实例。例如，由 likes(X,pomegranates)能推演出 likes(abraham,pomegranates)。

第三个推演规则，叫作实例：即根据全称量词语句 P，对于任意的代换 θ，推演出 P 的实例 $P\theta$。

关于询问，通过使用相同的变元名，可以把用变元表示的两个未指定的对象，限制为相同的。事实 plus(0,X,X) 表示 0 对加法是左恒等的。它读作，对于所有X的值，0 加X等于X。还有一个类似的用法，可以把“人人喜欢自己”这个句子翻译成 likes(X,X)。

用全称量词事实回答一个基询问是直接的，即搜索一个事实，而询问正是这一事实的实例。例如，根据事实 plus(0,X,X)，对于 plus(0,2,2)？的回答是 yes。用非基事实回答一个非基询问，则涉及一个新的定义：两个项的共同实例。

定义：如果C是A的实例、并且也是B的实例，换句话说，如果存在 θ_1 和 θ_2，使得 $C=A\theta_1$，在句法上与 $B\theta_2$ 是同一的，那么C是A和B的共同实例。

例如，目标 plus(0,3,Y)和 plus(0,X,X)有一个共同实例 plus(0,3,3)，把代换{Y=3}施加到 plus(0,3,Y)，把代换{X=3}施加到 plus(0,X,X)都产生 plus(0,3,3)。

通常，用事实回答询问，就是搜索询问和事实的共同实例。如果存在，则回答是共同实例，否则回答是 no。

用共同实例通过全称事实回答存在询问，涉及到两个逻辑推理规则。事实是用实例规则从实例推演得到的，而实例是用概括规则从询问推演得到的。

1.6 合取询问和共享变元

至此，要讨论询问的一个重要推广——合取询问。合取询问是指目标合取，如 father(teraoh,X)，father(X,Y)？或一般性为 $Q_1,\cdots,Q_n$？，当只有一个目标时，就是简单询问，简单询问是合取询问的特殊情况。在逻辑上，合取询问是问“根据程序是否能推演出合取结论”。我们用‘，’表示‘and’。不要把目标中分隔自变量的逗号与用于分隔目标、表示合取的逗号混淆了。

在最简单的合取询问中，所有的目标都是基，例如，father(abraham,isaac),male(lot)？，根据程序 1.1 回答这个询问时，因为询问中的两个目标都是程序中的事实，所以回答显然是 yes。一般地说，对于程序 P，如果P含有所有的 Q_i，那么当每一个 Q_i 是基目标时，询问 $Q_1,\cdots,Q_n$？的回答是 yes。所以基合取询问并不十分有趣。

当有一个或多个共享变元时，合取询问才有趣。共享变元就是那些出现在询问中的两个不同的目标中的变量。询问 father(haran,X)，male(X)？就是一例。合取询问中变量的范围是整个合取。于是询问 p(X)，q(X)？读作：“存在一个X，使得 p(X)和 q(X) 为真吗？”，象简单询问一样，合取询问中的变量隐含着存在量词。

共享变元通过限制变量范围，可以作为约束简单询问的手段。我们已经看到 plus(X,

X,4)？的例子，其中两数之和等于4的解被限制为两数必须相同。考虑一下询问father(haran, X), male(X)？，这里询问 father(haran, X)？的解被限制为孩子应是男性。程序 1.1 说明只有一个解，{X=lot}。或者，这个询问可以被看作把询问 male(X)？的解限制为有 haran 这个父亲的个体。

从询问 father(terach,X), father(X,Y)？可以发现共享变元的一个稍有不同的用法。一方面，它把 terach 的儿子限制为那些本身就是父亲的人，另一方面，它又考虑到个体Y，他(她)们的父亲是 terach 的儿子。例如，有{X=abraham,Y=isaac}和{X=haran,Y=lot}等解答。

如果合取询问中的所有目标是程序的推论。其中共享变元在不同的目标中，实例化为相同的值，那么合取询问就是P的逻辑推论。一个充分条件就是存在一个询问的基实例，而询问是P的推论，然后这个实例通过概括去推演询问中的合取。

对于基实例的限制是不必要的，当我们在第4章讨论到逻辑程序的计算模型时会讲到。我们采用限制是为了简化下面几节的讨论。

从运算上讲，用程序P求解合取询问 $A_1, A_2, \cdots, A_n$？，就是寻找一个代换 θ，使 $A_1\theta$, …, $A_n\theta$ 是P中事实的基实例。相同的代换用于所有的目标，保证了整个询问中变元的实例是共同的。例如，对于程序 1.1，考虑一下 father(haran,X), male(X)？，把代换 {X=lot} 用于询问，则给出了基实例 father(haran,lot), male(lot)？，它是程序的推论。

1.7 规　　则

有趣的合取询问恰好也定义了关系。询问 father(haran,X), male(X)？是寻找 Haran 的儿子。询问 father(terach,X), father(X,Y)？是寻找 terach 的孙子。这给我们带来了第三个，也是最重要的逻辑程序设计语句，即规则。规则能够使我们用已存在的关系定义新的关系。

规则是具有下列形式的语句：

$$A \leftarrow B_1, B_2, \cdots, B_n .$$

其中 $n \geqslant 0$。A是规则的头，B_i 是规则的体。A和 B_i 都是目标。规则、事实和询问还称为 Horn 子句，或简称子句。注意，事实只是当 $n=0$ 时规则的一个特殊情况。事实还称为单位子句，对于体中只有一个目标的子句，即当 $n=1$ 时，我们还有一个特殊的名字，称为迭代子句。出现在规则中的变元被全称量词所限定，并且它的作用范围是整个规则。

表达儿子关系的规则是：

son(X,Y)←father(Y,X), male(X).

同样地，还可定义女儿关系：

daughter(X,Y)←father(Y,X), female(X).

祖父关系的规则是：

grandfather(X,Z)←father(X,Y), father(Y,Z).

规则可以从两个角度来看。首先，它们是用简单的询问表达新的或复杂的询问。对于含有上述 son 规则的程序，根据规则询问 son(X,haran)？可以转换成 father(haran, X), male(X)？，而且求解也如前所述。关于 son 关系的新的询问已经根据含有 father 和 male 关系的简

单询问所构成。用这种方法解释规则就是它们的过程性读法。grandfather 规则的过程性读法是："为了回答X是Z的祖父，就要回答X是Y的父亲和Y是Z的父亲"。

规则的第二种看法是源于把规则解释为逻辑公理。反箭头用来表示逻辑蕴含。son 规则读作："如果Y是X的父亲，并且X是男性，则X是Y的儿子"。依照这种观点，规则是用其它比较简单的关系定义新的、或者更复杂的关系的一种手段。谓词 son 已经用 father 和 male 定义了。规则的这种联合读法，就是我们已知的说明性读法。grandfather 规则的说明性读法是："对于所有的X、Y和Z，如果X是Y的父亲，且Y是Z的父亲，则X是Z的祖父"。

尽管从形式上看，子句中所有的变元都是全称变元，但我们有时只关心出现在子句体中的变元，而不管子句头中的变元。例如，规则 grandfather 可以读作："对于所有的X和Z，如果存在一个Y，使得X是Y的父亲，并且Y是Z的父亲，则X是Z的祖父"。我们并不打算给出这种文字变换的正式证明，而只是为了方便。无论何时，当导致混淆时，读者可凭借子句的正式读法，依此，所有变元从外部看，都是全称变元。

为了把规则结合到我们的逻辑推演框架中来，我们需要假言推理定律。假言推理是说，由B和$A \leftarrow B$，我们可以推出A。

定义：全称假言推理定律就是：根据规则

$$R=(A \leftarrow B_1, B_2, \cdots, B_n)$$

和事实

$$B_1'.$$
$$B_2'.$$
$$\vdots$$
$$B_n'.$$

如果 $A' \leftarrow B_1', B_2', \cdots, B_n'$ 是R的一个实例，则可推演出 A'。

全称假言推理包括同一性和作为特殊情况的实例化。

现在，我们就可以给出逻辑程序概念的完整定义，也能给出有关逻辑推论概念的定义。

定义：逻辑程序是规则的有限集合。

定义：如果程序P中，存在一个子句，具有基实例 $A \leftarrow B_1, B_2, \cdots, B_n, n \geqslant 0$，而 $B_1, \cdots, B_n$ 是P的逻辑推论，并且A是G的实例，那么存在量词限定的目标G是程序P的逻辑推论。

注意，当且仅当经过有限步全称假言推理规则的应用之后，能由程序P推导出目标G，G就是P的逻辑推论。

把规则 son 增加到程序 1.1 后，考虑一下询问 son(S,haran)？，把代换 {X=lot, Y=haran} 用于规则，可得到实例 son(lot,haran)←father(haran,lot),male(lot)，此规则体中的两个目标都是程序 1.1 中的事实，于是全称假言推理蕴含着具有答案 S=lot 的询问。

从运算上看，回答询问反映了逻辑推论的定义，我们可以猜出目标的基实例和规则的基实例，并且可以递归地回答与那条规则体相应的合取询问。为了用程序P证明目标A，在P中选一个规则 $A_1 \leftarrow B_1, B_2, \cdots, B_n$，并猜出一个代换 θ，使得 $A=A_1\theta$，并且对于 $1 \leqslant i \leqslant n$，$B_i\theta$ 是基，然后递归地证明每一个 $B_i\theta$，这个过程可以包含任意长的推理链，但一般讲，很难猜出正确的基实例和选择出正确的规则。我们在第 4 章将说明如何删去实例的猜测。

前述关于 son 的规则是正确的，但它是关系的不完整的说明。比如，我们无法断定 Isaac 是 Sarah 的儿子。所遗漏的是孩子既可以是父亲的，也可以是母亲的，把表达这种关系的新

规则加进去，即

son(X,Y)←mother(Y,X), male(X).

同样地，若把 father 和 mother 两种情况考虑进去，那么定义 grandparent 关系，就会有 4 条规则：

grandparent(X,Y)←father(X,Y),father(Y,Z).
grandparent(X,Y)←father(X,Y),mother(Y,Z).
grandparent(X,Y)←mother(X,Y),father(Y,Z).
grandparent(X,Y)←mother(X,Y),mother(Y,Z).

有一个更好、更紧凑地表达这些规则的方法，我们需要定义一个辅助关系—parent 作为父亲或者母亲。逻辑程序设计技巧的一部分就是决定，定义哪些中间谓词以达到关系的完整、简洁和公理化。只要抓住双亲的定义是父亲和母亲，那么定义 parent 规则就是直接的。逻辑程序可以把不同的定义，用不同的规则结合起来，如对 parent：

parent(X,Y)←father(X,Y).
parent(X,Y)←mother(X,Y).

则 son 和 grandparent 规则分别为：

son(X,Y)←parent(Y,X),male(X).
grandparent(X,Y)←parent(X,Z),parent(Z,Y).

规则头中具有相同谓词的规则的集合（如两个 parent 规则）称为过程。在后面我们会发现，用 prolog 对这些规则进行运算解释时，这种规则集合确实与传统程序设计语言中的过程或子程序类似。

输入：基询问 Q 和程序 P.
输出：如果由 P 可证明 Q，则输出 yes；否则输出为 no.
算法：对 Q 的消解式初始化
　while 消解式 $A_1, \cdots, A_n$ 非空
　begin
　选择目标 A_i，$1 \leqslant i \leqslant n$，和
　P 中子句的基实例
　　$A \leftarrow B_1, B_2, \cdots, B_k$，$k \geqslant 0$，使得 $A = A_i$（如果不存在这样的子句，则跳出 while 循环）；
　　确定新的消解式
　　$A_1, \cdots, A_{i-1}, B_1, \cdots, B_k, A_{i+1}, \cdots, A_n$
　end
　如果消解式为空，输出 yes，否则输出为 no.

图 1.1

1.8 一个简单的抽象解释器

回答询问的运算过程，在前面几节已经非正式的描述和逐步地展开。我们现在重新回顾一下这些细节，以便给出逻辑程序的抽象解释器。为了与基目标的全称假言推理的限制一致，解释器只回答基询问。

解释器执行yes/no运算。对于程序 P 和基询问 Q，如果可以由 P 推导出 Q，则输出 yes，否则输出 no。如果目标不能从程序中推导出来，那么解释器也许不能停止，在这种情况下它什么回答也不给。图 1.1 给出了解释器的步骤。

处于任何一步计算的当前目标称为消解式。解释器的迹是在计算过程中产生的消解式序

列，以及所做出的选择。程序 1.2 中有程序 1.1 的子集以及 son 和 daughter 规则，对于程序 1.2 考虑询问 son(lot,haran)?，图 1.2 是回答这个询问的迹。

迹、隐含着根据程序得到的基询问之证明，这种证明的一种更为方便的表示是用证明树。

```
father(abraham,isaac).      male(isaac).
father(haran,lot).          male(lot).
father(haran,milcah).       female(milcah).
father(haran,yiscah).       female(yiscah).
son(X,Y)←father(Y,X), male(X).
daughter(X,Y)←father(Y,X),female(X).
```

程序 1.2

我们定义一些必要的概念。

定义：程序 P 对目标 G 的基化简，是 P 中规则的基实例体对 G 的替换，该规则的头与所选目标是同一的。

此后，上述定义可以推广为一般的(非基的)化简。化简是逻辑程序设计的一个基本计算步骤，它对应于全称假言推理的应用，还对应于图 1.1 中解释器 while 循环的一次重复，在化简中被替换的目标简化了，并且导出了新的目标。

我们把这些概念与图 1.2 例子中的迹相联系，可见迹中有三步化简。第一步是化简目标 son(lot,haran)，并且产生两个导出目标，一个是 father(haran,lot)，另一个是 male(lot)。第二步是化简 father(haran,lot)，没有导出的目标。第三步在化简 male(lot)时，也没有导出目标。

证明树由节点和边组成，它们表示计算过程中被化简的目标。简单询问的证明树，其根就是询问本身。树的节点是计算过程中被化简的目标，从一个节点到导出目标的每一个节点，都存在着一个有向边。合取询问的证明树只是合取中的全部单个目标证明树的集合。图 1.3 给出了图 1.2 中程序迹的证明树。

解释器中有两个未指定的选择：即必须选择从消解式中化简的目标，和选择化简该目标的子句（和适当的基实例）。这两种选择具有极不相同的性质。

```
输入: son(lot.haran)?和程序 1.2
    消解式非空
        选择 son(lot.haran) (唯一的选择)
        选择 son(lot.haran)←father(haran.lot).male(lot).
        新的消解式是 father(haran.lot).male(lot)?
    消解式非空
        选择 father(haran.lot)
        选择 father(haran.lot).
        新的消解式是 male(lot)?
    消解式非空
        选择 male(lot)
        选择 male(lot).
        新的消解式为空
输出: yes
```

图 1.2