演绎数据库 逻辑程序 数据库

计算机科学1996Vol.23№5

演绎数据库和逻辑程序中的否定

周傲英 施伯乐
(复旦大学计算机科学系 上海200433)

摘 要 This paper analyses the problems due to the introduction of negation into deductive databases and logic programs. Various approaches coupling with negation are surveyed and compared. For stratifiable logic programs, various stratifiabilities and the relationship among them are discussed. For general programs, the declarative semantics based on well-founded model and stable model are presented, and some comparison and analysis are followed.

关键词 Deductive database, Logic Program, Negation.

演绎数据库是数据库技术发展的重要方向之一。Datalog 是典型的演绎数据库语言，实质上是不带函词的一阶 Horn 子句语言，在演绎数据库中充当着查询语言和知识表示语言的双重角色。在理论基础方面，演绎数据库和逻辑程序有许多共同之处，都是基于一阶谓词逻辑的。

对常识推理或非单调推理的支持为演绎数据库和逻辑程序的研究提出了挑战。应付这种挑战的具体体现便是在 Datalog 中引入否定子目标，否定的引入远不象初看起来那么容易，会使得 Datalog 的语义定义方案变得无意义或不适用，从而会带来概念和思想的改动。针对这种情况，本文讨论了封闭世界假设(CWA)及失败即否定(NFF)等早期的朴素的解决办法。分层概念的引入是演绎数据库和逻辑程序研究的一个重要进展。本文描述了分层、局部分层、松散分层等概念及其间的关系，并针对各类分层分别讨论了迭代不动点语义、完美模型语义、良基模型语义、稳定模型语义等定义方案。本文最后还讨论了各种语义间的联系及有关的过程性求解算法。

1 Datalog 及其语义

严格说来，演绎数据库通常不带函词，但在许多场合，人们并不区分“演绎数据库”和“逻辑程序”这两个术语，甚至混合使用。一个演绎数据库是一组规则的集合，规则的语法可以对应到一阶逻辑语法，即由逻辑连接词构成的良构公式(wff)。Datalog 就是一个典型的演绎数据库语言，其语法简单、直观，语义明确，关于演绎数据库和逻辑程序的许多研究乃至系统实现都是围绕 Datalog 而展开的。

Datalog 的模型论语义、证明论语义、操作语义都很简单直观，并且从各种观点出发得出的语义结果都是一致的。基于模型论观点，演绎数据库中的规则定义了可能的世界或称“模型”，用以说明与谓词相关的各种可能实例的真假。对于 Datalog，项、Herbrand 空间、Herbrand 基等概念类似于一阶逻辑。解释是关于谓词的各种可能实例(即 Herbrand 基)的真假说明，对一个 Datalog 程序 P 而言，解释(I)可以看作是其 Herbrand 基(记为 B_P)的一个子集，包含在 I 中的实例为真，余下的(即 B_P-I)中的实例为假。这时，解释将 B_P 分为真、假两部分，称为完全解释。在一般情况下，解释可能将 B_P 分成真、假和未定义三部分，称为部分解释。对一个解释 I 而言，如果 Datalog 程序 P 中的任一规则都是 I 的逻辑后承，或称任一规则都被 I 满足，记为 $I\models P$，则称 I 是 P 的一个模型。一个 Datalog 有多个模型，模型集合在模型间的集合包含关系下构成一个完全格。Datalog 程序 P 的模型的交仍然是 P 的模型(模型相关定理)，所有模型的交就是 P 的最小模型(最小模型存在定理)。最小模型就是 P 的模型论语义。

不动点语义是基于直接后承算子而定义的，一般地，对一个程序 P 而言，其直接后承算子 T_P 定义如下：

$T_P\uparrow(I)=\{head(r)\mid r\in inst\_P, \forall L\in body(r), I\models L\}$

其中，r 表示实例化的规则，Inst_P 代表 P 的实例化结果，head(r)代表 r 的规则首，body(r)代表 r 的规则体中子目标的集合。对 Datalog 而言，直接后承算子 T_P 可定义为，

$T_P\downarrow(I)=\{head(r)\mid r\in inst\_P, \forall L\in body(r), I\in L\}$

在 Datalog 情形下，T_P 具有以下性质：(1)单调性：$I_1\subseteq I_2\Rightarrow T_P(I_1)\subseteq T_P(I_2)$。(2)连续性：$T_P(lub(X)>=lub(T_P(X))$。这些性质保证了 T_P 的不动点存在，$lfp(T_P)=T_P\uparrow\omega(\Phi)$，且与 P 的最小模型相同。

模型论语义和不动点语义都是说明性语义，不同之处在于前者侧重于公式(规则)的逻辑含义，后者更偏重于实际可操作性(Bottom-up 计值法就是 T_P 的直接实现)。

证明论语义则是把规则看作证明过程中使用的公理进行证明推导所得到语义。SLD 带选择函数的线性归结方法就是针对 Horn 子句而提出的证明论方法；SLDNF 则是 Clark 为解决否定事实的推导而提出的将 SLD 与否定即失败假设一起使用的证明论方法；SLS 是新近针对分层逻辑程序而提出的带选择函数的线性归结方法。

操作语义也称计算语义或过程性语义，其目标是给出一个可执行的算法，通过算法的执行而得的语义结果就是操作语义。Prolog 部分实现了 SLDNF；Datalog 中的 Semi-naive 计值法通过求解直接后承算子 T_P 是最小不动点来求解程序的语义。

从以上讨论可以看出，对一个演绎数据库系统或逻辑程序系统来说，说明性语义和过程性语义都是相辅相承、同等重要的。通过对前者的研究，人们可以明确程序的逻辑含义，可以与人们的主观愿望相比较；而后者则是实际应用所必需的，二者缺一不可。

2 否定的引入及问题

演绎数据库是关系数据库的推广和发展，Datalog 可以表达关系语言无法表达的传递闭包问题，这是 Datalog 表达能力较强的一个方面，但 Datalog 却无法表达关系语言极易表达的关系补运算。为了克服这一缺陷，可以在 Datalog 规则的子目标中引入否定。此外，在人工智能和自动推理领域，人们在常识推理或非单调推理研究方面取得了卓著的成果，提出了默认逻辑、限定(Circumscription)理论、封闭世界假设、自知逻辑等思想和理论。这也促使了在 Datalog 中引入否定子目标，以期在演绎数据库系统中支持非单调推理。

否定的引入并不象初想起来那么简单，它会带来许多问题。先看看不动点语义的问题。假设 P={A,C:-A,¬B}，如果 $T_P\downarrow$ 和 Datalog 一样定义为 $T_P\downarrow$，则 $T_P\downarrow(\Phi)=\{A,B\}$，这显然是不合理的，不符合人们的主观愿望。考虑这种情况，$T_P\downarrow$ 似乎应定义如下：

$T_P\downarrow(I)=\{head(r)\mid r\in Inst\_P, \forall L\in pos(body(r)L\in I, \forall L\in neg(body(r)\neg L\notin I\}$

其中 pos(X)，neg(X)分别代表闭文字集 X 中的正文字集和否定文字集。

这样一来，$T_P\downarrow$ 就不再具有单调性。下例说明这一点，假设 $P=\{B\leftarrow\neg A\}$，$I_1=\Phi$，$I_2=\{A\}$，则 $I_1\subseteq I_2$，$T_P\downarrow(I_1)=\{B\}$，$T_P\downarrow(I_2)=\Phi$，故，$T_P\downarrow(I_1)\not\subseteq T_P\downarrow(I_2)$。因此，$T_P\downarrow$ 就不一定存在不动点。

再来看看，否定的引入给模型论语义带来的问题。假定 P={A,C:-A,¬B}，则 M={A,C}，M′={A,B}是 P 的两个不可比的极小模型。同样，P′={A,B:-A,¬C}也有两个相同的极小模型。到底选哪一个作为模型论语义呢？直观上看来，M，M′分别是 P 和 P′的语义。这样，最小性已不能作为选择的标准，人们又给出了一个支撑性(supportedness)标准。程序 P 模型 M 满足支撑性，则有 $A\in M\Rightarrow r(r\in Inst\_P, A=head(r), M\models body(r))$，即，$M\in T_P\downarrow(M)$。支撑性的引入可以解决 P，P′的问题。

假设 P={A:-B,B:-A}，则 P 有两个 Herbrand 模型 M=Φ，M′={A,B}，它们都满足支撑性。实证主义模型(positivist model)要求同时满足支撑性和最小性。这里 M 是 P 的实证主义模型，也符合主观愿望。通常，支撑性和最小性可能会互相矛盾，从而导致实证主义模型不存在。

3 早期的解决办法

Reiter 在1978年给出了逻辑数据库的三个特征：唯一命名假设(UNA)、域封闭公理(DCA)和封闭世界假设(CWA)，其中 CWA 是逻辑数据库区别于数理逻辑的最显著特征。CWA 对应于现实世界的一种不完全的表示，在这一假设下，只需显式说明真

事实，未说明为真的皆为假。在关系数据库中，CWA被隐式采纳，表达了一种常识。逻辑数据库是非单调的。CWA是默认推理的一种特殊情形，可相应地有前提、理由和推论。

NFF(Negation as Finite Failure)是Clark在1978年为处理否定事实而提出的。SLD归结加入NFF后产生的SLDNF在归结过程中只能选择否定基文字，不能处理带变量的否定文字。Prolog实现了一种受限的SLDNF。SLDNF方法具备有效性，但不具备完备性。程序完备(program completion)是Clark针对SLDNF提出的一种模型化方法，其实质是在规则中用iff代替if，结果也会带来许多问题。

4 分层的否定

4.1 可分层(stratifiable)的程序

一个程序P，如果程序中规则的否定子目标的谓词符号不出现在任何规则的首部，则称P是半肯定(semi-positive)逻辑程序，其不动点存在，且lfp(T⊢)=T⊢↑ω(Φ)。这种程序中的否定只是关于EDB谓词的。

如果程序P存在一个划分$P^*=\{P_n,pred_n)n\geq 1\}$，并且①$pred_i\neq\Phi$，$pred_i\cap pred_j=\Phi(i\neq j)$，②$P_i(1\leq i\leq n)$是半肯定程序，③$P_i(1\leq i\leq n)$的规则首部出现的谓词都在$pred_i$中，则称$P^*$是P的一个分层；称P是一可分层的程序。逻辑程序的可分层性属于语法性质，是多项式可判定的。

可分层程序的语义可以用迭代不动点来定义。设可分层程序P有一个分层$P^*=\{(P_i,pred_i)1\leq i\leq n\}$，则$M_1=T\vdash_{P_1}\uparrow\omega(\Phi)$，$M_{(i+1)})=T\vdash_{P_{i+1}}\uparrow\omega(M_i)\cup M_i(i\geq 1)$。$P^*$的迭代不动点是$M^*=M_n$。可分层程序的迭代不动点独立于分层；迭代不动点语义与P的任一分层P^*的迭代不动点一致；并且迭代不动点满足支撑性和最小性。

4.2 局部可分层的程序

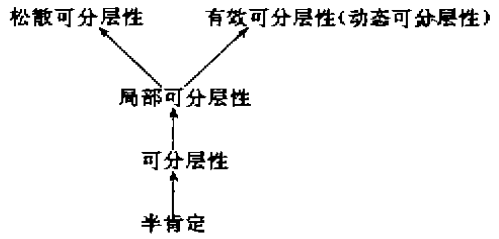
一个程序P是局部可分层的，当且仅当P的实例化Inst_P是可分层的。设P={even(0)，even(suc(X)；-even(X))}，这是一个利用后继函数suc定义偶数的程序。它有直观的明确语义。P实例化后是可分层的，所以P是局部可分层的。可以看出，局部可分层性仅局限于带函词的逻辑程序。对于不带函词的逻辑程序，只要规则首部的谓词参量位置上不出现常量，那么局部可分层性和可分层性是等价的。局部可分层性是不可判定的，它等价于图中无穷路径的判定。

4.3 松散可分层的逻辑程序

假定把程序写成P={suc(0,1)，suc(1,2)，…，even(0)，even(X)；-suc(Y，X)，—even(Y)}。显然，这只是一种写法上的改变，直观上的语义同样很明确。这里，程序P是不可分层的，也是不可局部分层的。但仔细考虑Inst_P，我们会发现suc(X，X)是不成立的，所以貌似的否定依赖实际上是不成立的。这种情况属于松散可分层。局部可分层性不考虑Inst_P，松散可分层性需考虑Inst_P才能决定。

4.4 各种可分层性之间的关系

对一个逻辑程序P而言，其各种可分层性之间的关系如下图所示(箭头方向表示放松)



5 逻辑程序的模型语义

5.1 局部可分层程序的完美模型语义

上面已经讨论过，对可分层程序可以用迭代不动点定义其语义。对局部可分层程序，先定义基原子公式间的优先(priority)关系(<，≤)。对程序P，根据Inst_P构造基原子间的依赖图，若A依赖于B，则A≤B；若A通过一条否定弧依赖于B，则A<B。在优先关系的基础上，再定义模型间的优选(preferability)关系(≪)。设M、N是P的两个模型。M≪N iff M≠N且A∈M−N，B∈N−M，s.t. A<B。≪下的最小模型就是局部可分层程序P的完美程序。完美模型(prefect model)体现了人们在书写程序时的主观愿望。

完美模型的性质也说明了其合理性。局部可分层的程序有唯一的完美模型，完美模型是关于集合包含关系的最小模型。对可分层的程序而言，其完美模型与其迭代不动点是一致的。这说明完美模型语义是一合适的推广。完美模型语义与优先限界推理也不谋而合。

5.2 良基模型(well-founded model)语义

良基模型语义实质上是一不动点语义，是针对任意的不受限制的带否定子目标的逻辑程序而提出的。良基模型通常是一个部分模型，记为 $M=\langle M^+, M^-, M^I\rangle$，$M^+, M^-, M^I$ 分别代表 Herbrand 基中为真、为假及未知的事实。

先来看看如何导出为假的事实，为此定义无基集(unfounded set)。对一个部分解释 $I=\langle I^+, I^-, I^I\rangle$，$A\subseteq B_P$ 是 P 关于 I 的一个无基集，如果每一原子 $P\in A$ 都满足以下条件(对于 Inst_P 中首部为 P 的规则，下列条件至少有一成立)：

(1)规则体内某个(正或负)子目标 q 在 I 中为假，即，$q\in\neg\cdot I^+\cup I^-$；

(2)规则体中某个正的子目标出现在 A 中。

程序 P 关于 I 的所有无基集的并，记为 $U_P(I)$，也是一个无基集，称为最大无基集 GUS。直观可见，U_P 是用来导出否定事实的。

在此基础上，对于 $I=\langle I^+, I^-, I^I\rangle$，可以如下定义变换 T_P, U_P 和 W_P。

(1)$p\in T_{P(I)}$ iff$\exists\ r(r\in \text{Inst_P}, \text{head}(r)=p, \forall\ L\in \text{body}(r), L\in\neg\cdot I^-\cup I^+)$；

(2)$U_{P(I)}$是 P 关于 I 的 GUS；

(3)$W_{P(I)}=T_{P(I)}\cup\neg\cdot U_{P(I)}$。

T_P, U_P, W_P 都是单调的。P 的良基模型 WFM $(P)=W_P\uparrow\omega(\Phi)$。

5.3 稳定模型(stable model)语义)

稳定模型是一完全模型，又称默认模型(default model)，其语义与默认逻辑(default logic)和自知逻辑(autoepistemic logic)有着密切的联系。稳定模型实质上是通过转换和检测二步来定义的。

对于程序 P 的一个完全模型 M，稳定性转换 $S_P(M)$分为以下三步(对 Inst_P 中的每一规则)：

(1)如果规则中有子目标$\neg$A，而 $A\in M$，则删除这一规则；

(2)在第(1)步剩下的规则中去掉体内所有的否定子目标，得结果 P'；

(3)对第(2)步产生的 Horn 程序使用 $T_{P'}\uparrow\omega(\Phi)$。如果 $M=S_{P(M)}$，则称 M 是 P 的稳定模型。

讨　论

良基模型和稳定模型都是针对一般的非可分层逻辑程序而提出来的。前者是一个部分模型，对任意的程序总是唯一存在的；后者是一个完全模型，对有些程序可能不存在，而对某些程序则可能有多个。它们都是完美模型在一般逻辑程序情形下的合理的推广。比较研究表明，如果一个程序的良基模型是完全的，则它和该程序的稳定模型一致。此外，如果一个程序有多个稳定模型，那么该程序的良基模型肯定是部分的，并且在每个稳定模型中都为真(假)的部分一定是良基模型的真(假)部分，余下的是未定义部分。因而，稳定模型适合于抽样查询。

关于良基模型的研究表明，其"未定义"部分包括"析取"，"因子提取"和"难以赋值"三部分，其中仅有最后一部分确实应赋以"未定义"值，所以说良基模型是"过于未定义的"(over-undefined)。良基模型对于弱分层的、动态分层或有效分层的程序是完全的。

良基模型语义的过程性求解也是近来的研究热点，交替不动点(alternating fixpoint)给出了良基模型的构造性定义；振荡不动点(oscillating fixpoint)则采用自底向上的方法求解；自顶向下求解的方法是 SLS。

参考文献

[1]Apt Kr, Bol R. N., Logic Programming and Negation: A Surevy, J. of Logic Programming, 19(1)1994

[2]Bidoit N., Negation in Rule-based Languages: A Survey, TCS, 78(1)1991

[3]Shepherdson J. C., Negation in Logic Programming for General Logic Programs, in: Minker, J(ed.), Foundation of Deductive Databases and Logic Programming, Margankauffman, 1988

[4]Gelder A. V., The Alternating Fixpoint of Logic Programs with Negation, JCSS, 47(1)1993

[5]Shi B. Zhou, A Bottom-up Evaluation of Datalog with Negation, JCST, 9(3)1994