一个基于抽象解释的部分演绎过程*

刘椿年

(北京工业大学计算机科学系,北京 100022)

摘要 本文提出一种新的基于抽象解释的逻辑程序部分演绎方法.在一遍预处理里,针对给定的程序 P 和目标 G,同时进行 $P\cup\{G\}$ 的部分演绎和抽象解释,以抽象解释控制部分演绎的展开过程.只要抽象论域是有穷的,部分演绎必定终止,而且 P' 带有抽象解释估算出的关于其运行性质的信息,便于对 P' 作进一步优化.

关键词 逻辑程序设计,部分演绎,抽象解释.

部分演绎(Partial Deduction)和抽象解释(Abstract Interpretation)是逻辑程序设计的两个重要技术.尽管在它们各自领域里已出现了大量的研究成果,但在它们的相互联系和结合方面尚鲜有报导.本文提出一个基于抽象解释的部分演绎方法,它具有如下特点:

1. 在一遍预处理中对 $P\cup\{G\}$ 同时进行抽象解释和(部分的)具体解释,以抽象解释控制部分演绎的展开过程.只要抽象论域是有穷的,部分演绎必定终止.

2. 部分演绎程序 P' 的时空效率权衡这一棘手问题可通过选用不同的抽象解释获得一定程度的解决.

3. 抽象解释在估算 P 的某动态性质的同时也为 P' 估算出同一信息,故而 P' 亦能以各种方式利用这一信息,从而被进一步优化(如 P' 的编译优化和 AND-并行优化).

本文假定读者了解逻辑程序设计的一般概念[1].为了叙述的简便,我们只讨论纯 Prolog,但有关结果可容易地推广到带 not 和预定义谓词的全 Prolog.本文依据文献[1],将不带 not 的纯 Prolog 程序(目标)称为定程序(定目标),简称程序(目标).

1 部分演绎的概念和方法

定义 1. 令 P 为程序,$G=?-A$ 为目标,$G_0=G_1,G_2,\cdots,G_n$ 为 $P\cup\{G\}$ 的一个(可能是不完全的)SLD 推导,其中 $G_n=?-B_1,B_2,\cdots,B_t$,所用置换序列为 $\theta_1,\theta_2,\cdots,\theta_n$.令 θ 为 $\theta_1,\theta_2,\cdots,\theta_n$ 在 G 中变量上的置换,则称子句

$$A\theta:-B_1,B_2,\cdots,B_t.$$

为从 G 到 G_n 的推导的结式(resultant).

* 本文 1993-03-17 收到

本研究是国家自然科学基金资助项目.作者刘椿年,50 岁,教授,主要研究领域为人工智能语言,软件方法学,知识工程等.

本文通讯联系人:刘椿年,北京 100022,北京工业大学计算机科学系

© 中国科学院软件研究所 http://www.jos.org.cn

定义 2. 令 T 为 $P\cup\{?-A\}$的一棵SLD树. 在 T 的每一非失败分枝上恰好选一个节点 $G_i(i=1,2,\cdots,r)$,令 R_i 为从 $?-A$ 到 G_i 的结式. 则子句集$\{R_i|i=1,2,\cdots,r\}$称为 A 在 P 中的部分演绎. 又令 $S=\{A_1,A_2,\cdots,A_m\}$为原子公式有穷集. S 在 P 中的部分演绎是 $A_j(j=1,2,\cdots,m)$在 P 中的部分演绎的并集. 将 S 中所含各谓词在 P 中的原定义换成 S 在 P 中的部分演绎,我们得到一个新程序 P',称为 P 对 S 的部分演绎(程序).

定义 3. 若 $P'\cup\{G\}$的任一原子公式的谓词符号出现在 S 中,该原子公式必是 S 中某原子公式的实例,则称 $P'\cup\{G\}$ 是 S-封闭的.

部分演绎程序 P'的合理性由归结原理的合理性自然地加以保证. P'的完备性则需附加一定的条件才能成立. 文献[2]中证明了,只要 $P'\cup\{G\}$ 是 S-封闭的,P'和 P 对求解 G 来说在计算上是等价的(即得出相同的解集合).

部分演绎系统构造的难点在于 P'的时空效率权衡. 一般来说,若限制展开的条件较宽,则部分演绎较深,所得 P'速度较快,但其程序量可变得过于庞大,甚至在某些递归的情况下造成部分演绎过程不终止;若限制展开的条件过严,则可能使部分演绎过程过早地终止,从而使 P'在速度方面没有什么提高. 要想设置一个对各种各样的源程序都是"最佳"的控制展开过程的条件是很困难的. 因此国际上已实现的几个部分演绎器(如文献[3])都是半自动的,要求用户以某种方式人为地参与展开或停止的决策. 最近,文献[4]提出了一个全自动化的部分演绎过程,我们将在第 3 节比较它与本文的工作.

2 抽象解释的概念与过程

给定程序 P 和目标 G,抽象解释是在某抽象论域 Δ 上模拟 $P\cup\{G\}$的执行,以估算出 P 真正运行时的某些性质. Δ 的一个元素是具体论域 U 中一组具体项的抽象表示,而具体置换 θ 的抽象表示则是抽象置换 $\xi=\{\delta_1/v_1,\cdots,\delta_n/v_n\}$,其中 $\delta_i\in\Delta,v_i$ 为变量. 给定一组具体对象,抽象化函数 α 将它们映射到一个抽象对象;反过来,具体化函数 γ 将一个抽象对象映射到一组具体对象. 最后,对逻辑语言解释中的每一基本操作 f,我们定义一个抽象操作 abs_f. 为了使抽象解释是合理的,抽象操作的结果应是具体操作的结果的正确抽象(即具体操作的任一结果必属于其对应的抽象操作结果所代表的具体对象集合).

为了得到逻辑程序的抽象解释过程,我们首先要识别出它的具体解释中的基本操作. 它们是(其中 T,T',T_1,T_2 为项的元组,θ,θ' 为置换):

1. 置换施行 $s(\theta,T,T')$,即 $T'\leftarrow T\theta$;

2. 合一 $u(\theta,T_1,T_2,\theta')$,即 $\theta'\leftarrow\theta\cdot\mathrm{mgu}(T_1\theta,T_2\theta)$.

它们对应的抽象操作的形式是(其确切的定义依赖于各个特定的抽象解释):

1. $abs\_s(\xi,T,A)$:将抽象置换 ξ 作用于具体项元组 T,得抽象项元组 A;

2. $abs\_u(\xi,T,A,\xi')$:T 和 A 在 ξ 环境下进行抽象合一,得到新的抽象置换 ξ'.

设有子句(其中 T_i 表示项的元组):

$$p(T_0):-q_1(T_1),q_2(T_2),\cdots,q_n(T_n) \tag{1}$$

对它的调用为 $p(X_{in})$,则(1)的具体解释将返回某 $p(X_{out})$. 这一具体解释过程可用上述的两个基本操作描述出来(细节略). 抽象解释的出发点是子句(1)和调用 $p(X_{in})$的抽象形式

© 中国科学院软件研究所 http://www.jos.org.cn

$p(A_{in})$(称为调用型)，而它所得到的是返回 $p(X_{out})$的抽象形式 $p(A_{out})$(称为返回型). 在具体解释过程中，将 s 和 u 换成它们的抽象形式 abs_s 和 abs_u，就可以立即得到抽象解释过程：

1. $abs\_u(\alpha(\{\varepsilon\}),T_0,A_{in},\xi_0)$；　　%% ε 为空置换，即初始时(1)中变量均是自由的
2. $abs\_s(\xi_0,T_1,A_{1,in})$；　　%% 得子目标 q_1 的调用型 $q_1(A_{1,in})$
3. 递归处理调用型 $q_1(A_{1,in})$；　　%% 得其返回型 $q_1(A_{1,out})$；
   $abs\_u(\xi_0,T_1,A_{1,out},\xi_1)$；
   $abs\_s(\xi_1,T_2,A_{2,in})$
4. 类似地对 $q_2,\cdots,q_n$ 做第 3 步；　　%% 最后得到 q_n 的返回型 $q_n(A_{n,out})$；
   $abs\_u(\xi_{n-1},T_n,A_{n,out},\xi_n)$；
5. $abs\_s(\xi_n,T_0,A_{out})$.　　%% 得调用型 $p(A_{in})$的返回型 $p(A_{out})$

抽象解释所要估算的动态性质实际上被表示在各个谓词的调用型中，而返回型只是抽象解释过程中所需的中间信息. 通常，抽象论域 Δ 是有穷的，亦即只有有穷多个不同的调用型，因此可用定点计算的方法在有穷步内求出所有可能的调用型. 比如，我们可维护一个对子〈调用型，返回型〉的表(称为外延表 Extension Table[5])，凡能通过查表解决的计算就不再重复进行了.

3　一个基于抽象解释的部分演绎过程

我们现在提出一个基于抽象解释的部分演绎过程，它在一遍预处理中同时对 $P\cup\{G\}$ 进行具体解释和抽象解释，以后者控制前者的展开/停止的决策从而达到部分演绎的目的. 设已确定了一种抽象解释(即给出了上节开头指出的各个成份，特别是抽象操作 abs_s 和 abs_u 的确切定义). 下面分步骤给出我们的基于抽象解释的部分演绎过程. 首先是一个实质性例程 $aipd(C,p(X),p(A_{in}),p(A_{out}),PR)$. 它的各变元含义为：

1. 输入变元：子句 C，形如 $p(T_0):-q_1(T_1),q_2(T_2),\cdots,q_n(T_n).(n\geqslant 0)$；
2. 输入变元：对 C 的调用 $p(X)$；
3. 输入变元：调用型 $p(A_{in})$，即 $p(X)$的抽象形式；
4. 输出变元：C 此次调用的一个估算返回型 $p(A_{out})$. $aipd$ 是一个不确定过程，每次给出一个解并标记回溯点；
5. 输入/输出变元：部分结式 PR—正在构造的结式当前已积累的子目标和它们的调用型.

算法 1. function $aipd(C,p(X),p(A_{in}),p(A_{out}),PR)$:bool;

1. if X 与 T_0 不可合一　　%% 对程序项的引用均指其在具体解释中的当前实例
   then return(false)　　%% 舍弃 C
   else 做具体合一；
2. $abs\_u(\alpha(\{\varepsilon\}),T_0',A_{in},\xi_0)$；　　%% 抽象操作只用项 T_0 的原形 T_0'，下同
3. if $n=0$ then goto 7　　%% C 为无条件子句
   else $k:=1$；　　%% 准备处理子调用 q_1

© 中国科学院软件研究所 http://www.jos.org.cn

4. $abs\_s(\xi_{k-1}, T_k', A_{k,in})$; %% T_k'是T_k的原形

5. if $q_k(A_{k,in}) \in ET$ %% ET是外延表

then 不再展开q_k,从ET中查出一个返回型$q_k(A_{k,out})$; %% 回溯点

$PR:=PR++[q_k(T_k), q_k(A_{k,in})]$; %%记录结式的一个子目标及其调用型

else 对q_k的各定义子句C_i递归调用$aipd(C_i, q_k(T_k), q_k(A_{k,in}), q_k(A_{k,out}), PR)$;

if 均失败 then return(*false*)

else 得一返回型$q_k(A_{k,out})$; %% 回溯点

6. $abs\_u(\xi_{k-1}, T_k', A_{k,out}, \xi_k)$;

$k:=k+1$;

if $k \leqslant n$ then goto 4; %% 下一个子目标

7. $abs\_s(\xi_n, T_0', A_{out})$; %% 得$C$的一个返回型$p(A_{out})$

$ET:=ET++\langle p(A_{in}), p(A_{out})\rangle$;

return(true).

设有程序P和目标$G=?-A$. $A=p(X)$为原子公式,其调用型为$p(A_{in})$. 下面的算法可求出A在P中的一个部分演绎P_A,且P_A的各子句及各谓词调用点均附有调用型信息.

算法 2. 求$A=p(X)$在P中的(带有调用型信息的)部分演绎P_A

1. 记p的定义子句集为D_p. 对每一子句$C_i \in D_p$做:

(i)$PR:=[]$;

(ii)if $aipd(C_i, p(X), p(A_{in}), p(A_{out}), PR)$=false

then skip

else (a)生成一个带调用型信息的结式:

"$p(X)$ {调用型 $p(A_{in})$} :- PR中所含的子调用及其调用型."

(b)用强迫回溯的方法使上述对$aipd$的调用逐次给出它所有的解,对每一个解生成一个结式;(即算法2以算法1为例程,并用强迫其回溯的方法得到所有的结式. 这种控制策略可用Prolog语言方便地加以实现)

2. A在P中的部分演绎PA即为1.产生的各结式的集合.

部分演绎系统的最外层的过程是求P对原子公式有穷集S的部分演绎程序P'. 它直接按定义1.2构造,并显式验证定义1.3的S-封闭性条件以保证P与P'的计算等价性,限于篇幅我们不再详述. 下面将本文的工作与有关工作作一对比.

文献[4]是我们看到的唯一的一篇讨论全自动化部分演绎过程的文献. 它提出了四种控制展开过程的计算规则,其中以实例证明(但无理论证明)为"最好"的规则是:"选取归结式中最左的其任何变体均未在当前SLD分支上出现过的子目标加以展开;无可选择时本分支的展开自动停止". 但它的四种规则均不能保证部分演绎过程的终止性. 我们的方法是用静态的抽象解释信息而不是用动态的实际执行信息来控制展开,因此只要抽象论域Δ是有穷的,部分演绎过程必定终止. 至于到底什么是"最佳"控制策略,文献[4]已指出目前尚无理论分析的基础,我们只能用实例来测试(如见下一节的实例分析).

文献[6]曾提出了基于抽象解释的部分演绎的思想. 但它所建议的方法(详见文献[6])涉及两遍预处理,所得的部分演绎程序P'未能继承抽象解释估算出的信息,而且包含有冗

© 中国科学院软件研究所 http://www.jos.org.cn

余的子句. 我们的方法吸取了文献[6]中提出的"抽象解释可用来控制部分演绎过程"的基本思想，但仅使用一遍扫描同时进行抽象解释和(部分的)具体解释，所得的部分演绎程序 P' 带有调用型的信息，且不含冗余子句(因真正执行时无用的子句在部分演绎时就被舍弃了，见算法 1).

4　一个应用实例

我们首先要选择一种抽象解释方法，这在文献中可以找到很多[7]. 为了说明本文的主要论点，我们取文献[8]中的一个比较简单的方法并加以改进. 这里 $\Delta=\{f,u,g\}$，其中 f 代表自由变量集，g 代表基项集，u 代表其它项的集合. 显然有如下的例化强度序《(x《y 意为 y 的例化比 x 的例化更充分)：f《u《g. 抽象操作的确切定义为：

1. $abs\_s(\xi,T,A)$：若 $T=\langle T_1,\cdots,T_n\rangle$，则 $A=\langle\Xi(T_1),\cdots,\Xi(T_n)\rangle$. 其中 Ξ：项集→Δ 是 ξ：变量集→Δ 的自然推广：若 T 是变量，则 $\Xi(T)=\xi(T)$；若 T 是常数，则 $\Xi(T)=g$；若 T 是复合项，则 $\Xi(T)=g$ 当且仅当 T 的所有真子项均被映射到 g，否则 $\Xi(T)=u$.

2. $abs\_u(\xi,T,A,\xi')$：若变量 v 出现在 T 中，则 $\xi'(v)=\max\{\xi(v),A[i]\mid v$ 出现在 $T[i]$ 中$\}$；否则 $\xi'(v)=\xi(v)$. 这里 max 是在《序下取最大值，反映了在程序运行时变量的例化只会越来越强的事实.

上述抽象解释用来控制部分演绎的展开过程时，往往在很浅的地方就停止了. 我们可以把基项集 g 细分为程序运行时可能产生的所有基项，使部分演绎的展开深度大大增加，但这可能产生无穷多个基项，从而使部分演绎的过程不终止. 作为折中，我们把长度大于 2 的表如$[a,b,c,\cdots]$抽象为$[a,b|g]$(对其它复合项也作类似的截断)，扩大 Δ 使之包含所有这些有穷多个抽象化的基项，并相应地修改 abs_s 和 abs_u 的定义，就得到一个更合于我们的需要的抽象解释.

下面考虑一个应用实例. 我们用复合表来表示 $n*m$ 矩阵，每一子表代表矩阵的一行. 例如$[[X1,X2],[X3,X4],[X5,X6]]$是一个 $3*2$ 矩阵. 通用的求矩阵转置的程序 P 为：

```
t(M,[]) :-n(M).
t(M1,[Row|M2]):-m(M1,Row,M3),t(M3,M2).
m([],[],[]).
m([[X|R]|M1],[X|Row],[R|M2]) :- m(M1,Row,M2).
n([]).
n([[]|M) :- n(M).
```

我们想用部分演绎的方法得到一个针对两列矩阵转置的较快的程序 P'. 为此考虑目标 $G=?-t([[X1,X2]|T],A)$. 如果我们知道 P 总是用来求具体矩阵的转置，则可断定 G 的调用型为 $t(g,f)$，尽管它的语法形式中含有很多变量. 在我们改进的方法中，调用型可细化为 $t([[g,g]|g],f)$. 于是，按第四节的基于抽象解释的部分演绎算法，就得到下面的部分演绎程序 P'(它附有调用型信息 CP，在输出 CP 时为简单起见把基项又简化为 g)：

```
t([[X1,X2]],[[X1],[X2]]).                                  CP: t(g,f)
t([[X1,X2],[X3,X4]|T],[[X1,X3|R1],[X2,X4|R2]]) :-          CP: t(g,f)
m(T,R1,M1),                                                CP: m(g,f,f)
```

© 中国科学院软件研究所 http://www.jos.org.cn

$m(M1,R2,M2)$, $CP: m(g,f,f)$

$n(M2)$. $CP: n(g)$

%% m 和 n 的定义同 P,但也附有调用型信息,略.

文献[4]中也考虑了这个实例.在其给出的四种控制策略中,最好的一种所产生的部分演绎程序 P''与我们的 P'类似,但多用了一条子句.而且,由于 P'带有调用型信息,使它可被进一步优化.

5 结束语

本文提出的基于抽象解释的部分演绎方法生成带调用型信息的部分演绎程序,是将抽象解释和部分演绎两个领域结合起来的一项新的研究.该方法已在 Sun SPARC 工作站上用 SWI—Prolog 系统[9]实现,第 4 节的实例是实际运行的结果.

由于部分演绎程序 P'的质量评价(时空权衡)的内在复杂性,很难找到一种抽象解释普遍适合于各种各样源程序的部分演绎.由第 4 节实例的提示,可能基于项的类型分析的信息是比较有希望的,这是下一步工作的目标之一.由于我们的实现中 abs—s 和 abs—u 的定义是独立的模块,引进一种新的抽象解释只需改写这个模块.实际上我们已试验了数种抽象解释,每次涉及的改写工作量是有限的.

参考文献

1 Lloyd J W. Foundations of logic programming. 2nd ed. Springier—Verlag, 1987.

2 Lloyd J W, Shepherdson J C. Partial evaluation in logic programming. Journal of Logic Programming, 1991.

3 Komorowski J. Synthesis of programs in the partial deduction framework. In: Lowry M R and McCartney R D eds. Automating Software Design, AAAI Press/MIT Press, 1991: 377—403.

4 Benkerimi K, Lloyd J W. A procedure for the partial evaluation of logic programs. In: Proc. of 1990 North American Conf. in Logic Programming, Texas, MIT Press, 1990: 343—358.

5 Hermenegildo M V *et al*. Global flow analysis as a practical compilation tool. Journal of Logic Programming, 1992, **13**: 349—366.

6 Gallagher J *et al*. Specialization of prolog and FCP programs by abstract interpretation. In: New Generation Computing, 1988, **6**: 159—168.

7 Cousot P, Cousot R. Abstract interpretation and application to logic programs. Journal of Logic Programming, 1992, **13**: 103—179.

8 刘椿年,李晨.一类受限 Prolog 程序的抽象解释及其应用.计算机学报,1988,**11**(**12**):717—724.

9 Wielemaker J. SWI—Prolog 1.4 reference manual. University of Amsterdam, 1990.

A PARTIAL DEDUCTION PROCEDURE BASED ON ABSTRACT INTERPRETATIONS

Liu Chunnian

(*Department of Computer Science, Beijing Polytechnic University, Beijing* 100022)

© 中国科学院软件研究所 http://www.jos.org.cn

Abstract This paper presents a novel approach to partial deduction (PD) of logic programming, which is based on abstract interpretations (AIs). In asingle preprocessing phase, given a program P and a goal G, PD and AI for $P \bigcup \{G\}$ are carried out simultaneously, with AI controlling the process of auto—unfolding in PD. PD always terminates if the abstract domain of AI is finite. Moreover, P' has got the information (collected by AI) about its operationalbehavior, which can be used for further optimization of P'.

Key words Logic programming, partial deduction, abstract interpretation.

© 中国科学院软件研究所 http://www.jos.org.cn