基于解释的算法构架的学习*

费宗铭 张家重 徐家福

(南京大学计算机软件研究所,南京 210008)

摘要 本文提出了将解释学习方法用于学习算法构架的思想,以提高软件自动化系统从功能规格说明转换到设计规格说明的能力.文中给出了算法构架的表示,操作性的定义及其处理方法.系统从用户给出的一个问题的解中学习算法构架,用于解决一类问题,系统的学习效果表现为通过学习能够解决原来不能解的问题.

关键词 程序转换,规格说明,解释学习.

软件开发是知识密集型的活动,有关知识的获取、管理和使用是其核心问题.人们在处理这些知识时能力上的局限性,使软件开发出现人们难以驾驭的局面.利用计算机这一工具去延拓我们的能力,去处理有关的知识,是解决这一问题的重要途径.近年来,软件自动化方面的一系列工作[1,2]正是瞄准这一方向.其中,从刻划"做什么"的软件功能规格说明,自动或半自动地生成反映"如何做"的软件设计规格说明(亦即算法设计自动化)是关键.现有系统在处理这一问题时,依赖于系统的知识,正是这些知识的多少,决定了系统的能力.

现有工作的不足之处是适用面较窄,其能力在系统设计完成,投入运行时就已确定.当待解问题超出系统预先有的知识范围时,系统就不能解决,表现出脆弱性.在这种情况下,这些系统的通常处理方法是由用户直接给出设计规格说明.相应地,出现的另一问题是,系统在遇到同类甚至相同问题时,仍需用户重复干预.

造成上述两方面不足的重要原因是系统缺乏学习的能力,系统不能从解题过程中自动获取有用的知识,提高自身的能力.

本文工作的出发点就是在现有系统中引入学习机制,以克服上述不足.自学习软件自动化系统 NDSAIL 是我们在这方面的尝试[3].本文着重考虑其中用解释学习方法学习算法构架部分,即通过对用户设计的算法实例的分析,使用解释学习方法,学习算法设计的策略,扩充系统的知识和能力,学习结果表现为系统能解决原来不能解的问题.

我们采用算法构架的形式表示算法设计的策略知识,是基于这样的分析,人们在设计算法时,一般先设计其总体结构,再确定其中的待定部分.这里的算法构架即是表示了算法的

* 本文 1991-10-21 收到,1992-01-24 定稿

作者**费宗铭**,29 岁,1991 年博士毕业于南京大学,主要研究领域为软件自动化,机器学习.**张家重**,29 岁,1991 年硕士毕业于南京大学,主要研究领域为软件自动化,机器学习.**徐家福**,70 岁,教授,主要研究领域为软件自动化,机器学习,自然语言理解等.

本文通讯联系人:费宗铭,南京 210008,南京大学计算机软件研究所

© 中国科学院软件研究所 http://www.jos.org.cn

总体结构,用算法构架表示设计知识的早期工作见[3].

解释学习是新近提出但研究较多的学习方式[4],它基于对单个实例的分析,应用领域知识,学习目标概念的描述.

解释学习是保正确性的,即在领域理论正确的前提下,其学到的概念描述亦是正确的.算法设计自动化中,系统设计的结果正确性必须得到保证,这首先要求利用的设计知识是正确的,因此,作为学习这些知识的方法应该是保正确性的.这是我们采用解释学习方法的原因之一.

从机器学习的角度,算法构架学习中的训练实例是用户给出的具体问题及其解,学到的内容为算法构架,要用户给出反映同一设计思想的多个实例较为困难,至少是繁琐的.解释学习将多个实例的要求转移到对领域理论的要求,减轻了用户的负担.这也是采用解释学习方法的重要原因.

使用解释学习方法学习算法构架的难点在于,研究对象为算法构架,相应的表示机制不仅要刻划算法所处理的数据之间的关系,反映算法构架各部分之间的相互约束形式,而且须能刻划抽象的算法构架间的关系,以适应解释和推广的要求.解释学习中另一关键问题是操作性准则的处理,必须使得学到的算法构架既有一定的适用面,又使系统能有效地用于解决实际问题.

基于上述两点,本文讨论了算法构架的表示机制 FuncL,给出了使用解释学习方法学习算法构架时具体的处理方法(特别是操作性的处理),最后给出了一个实例.

1 表示机制 FuncL

1.1 描述

FuncL 的基本组成单位是函数,每一函数有一相应的输入输出类型.若函数 f 的入/出类型分别为 T_{in},T_{out},则记为 $f:T_{in}\rightarrow T_{out}$.

首先,我们给出 FuncL 的一些基本概念.

(1)FuncL 的类型集 Type 构成如下:

a)若 T 为类型变量或 FGSPEC 基本类型,则 $T\in Type$.

b)若 $T\in Type$,则(set T),(seq T),(array T i_1 i_2)$\in$Type.

其中 i_1,i_2 为整数,且 $i_1\leqslant i_2$.(set T)、(seq T)分别为分量类型为 T 的集合和序列类型,(array T i_1 i_2)为分量类型为 T,下界、上界分别为 i_1 和 i_2 的数组类型.

c)若 $T_i(i=1,\cdots,n)\in Type$,则(cart $T_1\cdots T_n$)$\in$Type,它表示类型 $T_1,\cdots,T_n$ 的笛卡尔积.

注:后面我们将用到 simptype,Tv 和 comptype,simptype 表示 FGSPEC 原始函数集,Tv 取值可为任何类型,而 comptype 可为 set 和 seq 之一.

(2)FuncL 的函数类型 functype::=Type→Type.

(3)函数型把已有函数组合构成新的函数,这里介绍其中的三个:

a)复合 comp

若 $f_1:T_1\rightarrow T_2$,$f_2:T_2\rightarrow T_3$,则(comp f_2 f_1):$T_1\rightarrow T_3$.

© 中国科学院软件研究所 http://www.jos.org.cn

且对 $x\in T_1$,有$(\text{comp } f_2\ f_1)(x)=f_2(f_1(x))$.

b)构造 tuple

若 $f_i:T\to T_i$. $(i=1,\cdots,n)$则$(\text{tuple } f_1\cdots f_n):T\to(\text{cart } T_1\cdots T_n)$,

且对 $x\in T$,有$(\text{tuple } f_1\cdots f_n)(x)=(f_1(x),\cdots,f_n(x))$.

c)条件 cond

若 $B_i:T_1\to\text{bool}$,$f_i:T_1\to T_2(i=1,\cdots,n)$,则$(\text{cond}(B_1\ f_1)\ \cdots\ (B_n\ f_n)):T_1\to T_2$

且对 $x\in T_1$,

$$\text{有}(\text{cond}(B_1\ f_1)\ \cdots\ (B_n\ f_n))(x)=\begin{cases}f_1(x), & \text{若 } B_1(x)\text{成立}\\ \cdots & \cdots\\ f_n(x), & \text{若 } B_n(x)\text{成立.}\end{cases}$$

(4)算法构架在 FuncL 中的表示为按上述规则构造所得的带有入/出类型的函数.

1.2 算法构架(Scheme)间的关系

上述表示机制以函数型的形式严格地刻划了算法构架各部分之间的相互约束形式,本节通过定义算法构架间的偏序结构刻划其相互间的关系.

定义. 类型 Type 上的偏序关系是下列关系的自反传递闭包:

(1)对任何 $t\in\text{simptype}$,$t<\text{simptype}$.

(2)若 set,seq,array,cart 的分量类型具有序关系"<",则构造的复合类型亦保持这种关系.

(3)对任何 $t\in\text{comptype}$,$pt\in\text{Type}$,

有 $t(pt)<\text{comptype}(pt)$.

(4)$\text{simptype}<Tv$,

对任何 $T\in\text{Type}$, $\text{comptype}(T)<Tv$,

对任何 $T_1\cdots T_n\in\text{Type}$,$(\text{cart } T_1\cdots T_n)<Tv$.

定义. 算法构架之间的偏序关系是下列关系的自反传递闭包.

(1)若 $f:T_1\to T_2$ 为函数常量,$vf:T_1\to T_2$ 为函数变量,

则 $f<vf$ (且称为常变偏序).

(2)若 $vf_1:T_{11}\to T_{12}$,$vf_2:T_{21}\to T_{22}$,且 $T_{11}<T_{21}$,$T_{12}<T_{22}$.

则 $vf_1<vf_2$(且称为类型偏序)

(3)对任何常元或变元 B_i,f_i,设 f 为函数变元,则有

a)$(\text{comp } f_1\cdots f_n)<f$

b)$(\text{tuple } f_1\cdots f_n)<f$

c)$(\text{cond } (B_1\ f_1)\cdots(B_n\ f_n)<f$

(4)comp, tuple, cond 保持偏序关系(对常变偏序而言)

若 $f_i<f'_i(1\leqslant i\leqslant n)$(对 cond 或者 $B_i<B'_i$),

则$(\text{comp } f_1\cdots f_i\cdots f_n)<(\text{comp } f_1\cdots f'_i\cdots f_n)$,

$(\text{tuple } f_1\cdots f_i\cdots f_n)<(\text{tuple } f_1\cdots f'_i\cdots f_n)$,

$(\text{cond}(B_1\ f_1)\ \cdots\ (B_i\ f_i)\ \cdots\ (B_n\ f_n))<(\text{cond}(B_1\ f_1)\ \cdots\ (B'_i\ f'_i)\ \cdots\ (B_n\ f_n))$.

(5)comp,tuple,cond 保持偏序关系(对类型偏序而言)

© 中国科学院软件研究所 http://www.jos.org.cn

若 $f_i<f'_i(i=1,\cdots,n)$, $B_i<B'_i(i=1,\cdots,n)$,

则 $(\text{comp } f_1\cdots f_n)<(\text{comp } f'_1\cdots f'_n)$,

$(\text{tuple } f_1\cdots f_n)<(\text{tuple } f'_1\cdots f'_n)$,

$(\text{cond}(B_1\ f_1)\cdots(B_n\ f_n))<(\text{cond}\ (B'_1\ f'_1)\cdots(B'_n\ f'_n))$.

例如:设 f_1,f_2,f_3 为函数常量,vf,vf_1,vf_2,vf_3 为函数变量,则有如下的偏序关系(图 1,其中 $t_1<t_2$ 表示为 $t_1\rightarrow t_2$).

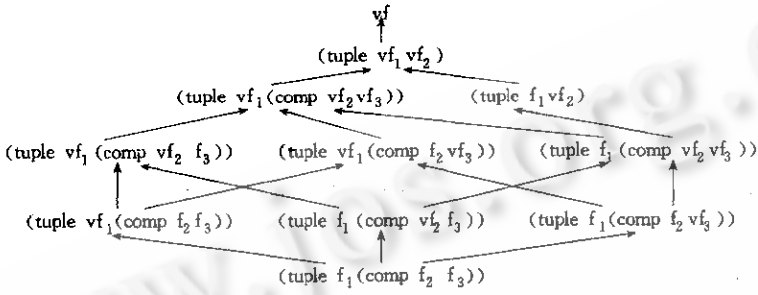

图1　算法构架间的关系

2　算法构架的学习

我们已定义了算法构架及其上的偏序结构,明确了从类型约束角度合法的算法构架的构成规则.可以把算法构架的构成规则看成领域知识,而它本身是要学到的目标概念.用户给出的具体的解是训练实例,经过对训练实例的解释和推广,可得到可操作的算法构架的概念.这里的可操作重在可以用它来指导问题分解过程.

算法构架的学习的特殊性在于,其构成单位是函数,变与不变是就函数常量和函数变量而言.对具体算法构架的解释可利用算法构架构成规则及其间偏序结构,分析其构造的合法性,包括类型合法性,并建立推广路径.下面给出一些构成规则,并用之对一算法形式予以解释.(规则以类似于 PROLOG 的形式给出).

$\text{is\_scheme}(S,T_1,T_2)$

　　$:-\text{is\_comp}(S),\text{is\_first}(S,S_1),\text{is\_second}(S,S_2),$

　　$\text{is\_type}(T_3),\text{is\_scheme}(S_1,T_3,T_2),\text{is\_scheme}(S_2,T_1,T_3).$

$\text{is\_scheme}(S,T_{in},T_{out})$

　　$:-\text{is\_tuple}(S,n),\text{is\_1st\_t}(S,S_1),\text{is\_type}(T_1),$

　　$\text{is\_scheme}(S_1,T_{in},T_1),\cdots,\text{is\_nth\_t}(S,S_n),$

　　$\text{is\_type}(T_n),\text{is\_scheme}(S_n,T_{in},T_n),$

　　$\text{is\_cart\_type}(T_{out},T_1,\cdots,T_n)$

这两条规则分别用来解释复合和构造,对一具体算法的解释结构,则可由上述规则产生.

例如:对具体算法(comp add(tuple square Id)):int→int,其中,square:int→int;Id:T→T;add:int×int→int;则可构造解释结构如图 2(其中 is--scheme(S2,T1,T3)解释部分

© 中国科学院软件研究所 http://www.jos.org.cn

从略).

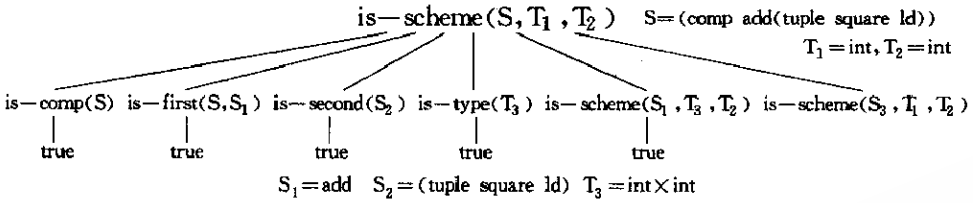

图2　一算法的解释结构

算法构架学习机制的推广过程将解释结构中部分个体变量化,并兼顾操作性准则,进行结构推广.下面,我们给出算法构架的操作性定义.

定义. 一个算法构架是可操作的,可递归定义如下:

1. 若 f 为函数变量或函数常量,则 f 是可操作的.

2. 若 $f_1:T_1\to T_2$, $g:T_2\to T_3$ 之一是可操作的,而另一函数为函数常量,则(comp g f)是可操作的.

3. 若 $f_i:T\to T_i$ 为可操作的,而 $f_j(j=1,\cdots,n\wedge j\neq i)$为函数常量,则(tuple $f_1\cdots f_i\cdots f_n$)为可操作的.

4. 若各分支为可操作的,则(cond(B_1 f_1) … (B_n f_n))为可操作的.其中分支(B_i f_i)为可操作 iff(B_i 可操作的 $\wedge$ f_i 为函数常量)$\vee$(B_i 为函数常量 $\wedge$ f_i 为可操作的).

满足可操作性的最简单的方法是不作任何推广,即保留所有实体,则产生的算法构架为原来函数,这太特殊,无一般性.另一极端是将所有个体去掉,推广为单个函数,这由于没有反映解决问题的任何方法,过于一般,亦不为我们所取.在这两个极端之间,按照上节定义的偏序关系而构成的格结构中,须寻求一个既能够反映设计思想,但又可操作的算法构架.例如,对第 1 节中的例子(tuple f_1(comp f_2 f_3)),满足可操作性的推广有如下几个:vf,(tuple f_1 vf_2),(tuple vf_1(comp f_2 f_3)),(tuple f_1(comp vf_2 f_3)),(tuple f_1(comp f_2 vf_3)),(tuple f_1(comp f_2 f_3)).

在这些满足操作性准则的构架中,选择怎样的具体形式,依赖于系统的知识和采取的策略.我们主要根据与数据类型相关性的原则,保留且仅保留那些反映算法结构的函数常量,而将其他函数常量推广.如对自然数类型而言,其零元和后继函数就是比较反映其构成结构的操作,对序列类型,取首、取尾操作亦然.我们注意到,对应于一种数据类型,各操作相关性是相对其他数据类型的操作及该类型的其他操作而言的,并非截然需保留还是不保留,这反映了一种多值的特点.这第一步所作的推广包括数据类型的参量化和函数的变量化,属变量回归的范围,只是对象不是个体变元而是函数变元.

当一个复合结构中有两个待定部分时,就须向上推广一层.因为在遇到问题时,就无法确定其中任何一个具体形式.如在(tuple f_1(comp f_2 f_3))中,若我们只保留 f_1,而 f_2,f_3 都需推广,则为(tuple f_1(comp vf_2 vf_3)),这时出现两个待定函数 vf_2,vf_3,就将之推广为(tuple f_1 vf_2).这里根据算法构架结构的性质,按可用的原则,推广了算法构架的组成形式,实质上做了结构推广.系统对数推广暂未考虑.

© 中国科学院软件研究所 http://www.jos.org.cn

在 NDSAIL 系统中，基于解释的算法构架的学习扩充了系统的算法构架库，使得系统能够解决原来不能解的问题，在一定意义上提高了系统的能力.

例如，给定两个例子：(1)求对数，(2)求两个自然数的整除商. 单独给例(1)和例(2)，系统都不能解决. 但不管用户给出任一问题的解，系统都能学习算法构架，用于解决另一问题(实质上能解决包括该问题在内的一类问题). 而在无学习功能的系统中，是不可能达到的.

例 1：求对数.

```
spec :int→int
        n→y
pre  :grt(n,0)
post :and(le(exp(2,y),n),less(n,exp(2,add(y,1))))
```

其解以树形结构表示为图 3.

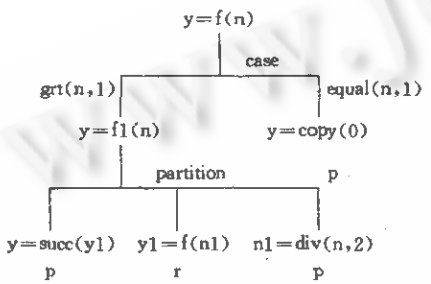

图3　求对数问题的解

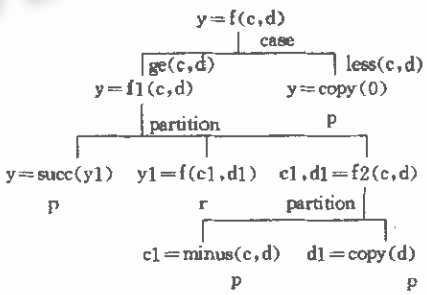

图4　求整除商问题的解

学到算法构架：

```
f=(cond (op1 (comp copy 0))
        (op2 (comp succ (comp Rec op3))))
```

其中：op1，op2：int→bool；op3：int→int.

用于求解例 2：求 c，d 的整除商

```
spec :int×int→int
      c,d→y
pre  :and(grt(c,0),grt(d,0))
post :and(le(mul(y,d),c),less(c,mul(add(y,1),d)))
```

可得出其解为树形结构为图 4.

结　语：本文讨论了用解释学习的方法学习算法构架，用于扩充系统的解题能力. 给出了处理这一比较复杂问题时的表示机制及处理方法. 与同类工作[5,6]相比，在保证结果正确性，便于自动实现，提高系统能力等方面有所进步.

进一步的工作可将领域知识结合到算法设计中，使得学到的算法构架能更有效地使用.

参考文献

1　Bauer F L *et al*. Formal program construction by transformations—computer—aided. Intuition—Guided Program-

© 中国科学院软件研究所 http://www.jos.org.cn

ming. IEEE—SE，1989；15(2)：165—179.

2 Smith D R. KIDS：a semiautomatic program development system. IEEE—SE，1990；16(9)：1024—1043.

3 Gerhart S L. Knowledge about programs：a model and case study. Proc. Int. Conf. on Reliable Software，Los Angeles，Calif，1975：88—95.

4 Mitchell T *et al*. Explanation—based generalization：a unifying view. Machine Learning，1986；1(1)：47—80.

5 Steier D. Automating algorithm design within a general architecture for intelligence. Ph. D. Dissertation，School of Computer Science，Carnegie—Mellon University，Pittsburgh，PA，1989.

6 Dershowitz N. Programming by analogy. In：Michalski R S *et al* (Eds)，Machine Learning：An Artificial Intelligence Approach，Vol. II，Morgan Kaufmann，Los Altos，Calif，1986.

EXPLANATION—BASED ALGORITHM SCHEME LEARNING

Fei Zongming，Zhang Jiazhong and Xu Jiafu

(*Institute of Computer Software，Nanjing University，Nanjing* 210008)

Abstract　This paper proposes the idea that explanation—based learning method be used to learn algorithm schemes，with the aim to enhance the ability to transform from functional specification to design specification of software automation system. It gives representation of algorithm schemes，definition of operational criterion and processing method in detail. System can use algorithm schemes learned to solve other problems，which is impossible without learning.

Key words　Program transformation，specification，explanation—based learning.

© 中国科学院软件研究所 http://www.jos.org.cn