计算机科学1995 Vol.22 №.5

面向问题的系统化程序设计方法及其描述工具

何明昕
（暨南大学计算机科学系 广州510632）

摘 要 Based on the inside considerations of researches and practices about program designing and ralated areas, this paper presents a systematic program designing method which is problem-oriented and produces programs progressively by stepwise problem resolving and refinements. The basic problem resolving strategies are summarizied and a progamming description tool called program production graph is presented as a partner of the programming method.

关键词 Method to design programs, Program production diagram, Problem resolving strategy, Programming description tools, Problem solving.

一、引 言

回顾以程序设计为核心的计算科学的研究、实践和教学，虽然有越来越多的人的参与和林林总总的成果发表，而“程序设计危机”并没有得到根本的改善，为此，W. J. Dijistra 郑重指出“真正讲授计算科学的严酷性”[1]，我们需要寻求简明实用有效的系统化程序设计和表达方法。

分析总结软件和程序设计方法及人工智能等相关领域的研究和实践，我们可以得到以下几点认识：

1. 程序设计是将待解的问题转化为可由计算机直接或间接执行的指令或语句组成的求解序列并涉及问题领域和计算机领域的综合思维过程。或许由于问题领域的多样性和复杂性，目前有关的研究及教学有失偏颇和人为的脱节。我们不仅需要程序设计语言之类的编程环境的语法以及语义的种种细节，更应该充分认识问题求解方法和策略在程序设计中的重要性，充分认识程序设计的综合属性，并借鉴人工智能及相关学科的研究成果，开展面向问题的程序设计方法研究。

2. 程序的规格说明（问题描述）与实现（求解步骤）之间存在不可避免的交织[3]。事实上，任一层次的描术总是另一较高层次的实现；任一层次的实现总是另一较低层次的描述；对人来说最低层次的程序实现，对计算机而言便是最高层次的描述；对程序的部分描述的分解精化可以得到程序的部分实现。实践表明，人们只能同时处理有限数目的对象，这数目一般不超过十。因此，我们应该放弃程序的形式化描述一步到位的理想；而逐步分解、各个击破是人类解决复杂问题的最重要的思想方法，也是系统化程序设计的最重要的指导思想。

3. 必须区分程序设计的结果（即程序）的形式与产生这些结果（程序设计）的过程中所使用的方法和表达形式[1]。程序设计可以被看作是用符号模拟问题求解策略和步骤的过程。最终的符号形式，即交付给计算机的程序，必须严格形式化，能够被相应的编程环境（例如某种程序设计语言的编译器）所接受、经过一定的处理（如编译连接）后，能被正确地执行，从而完成问题的计算。而程序设计过程中采用的方法和记录思维过程的符号形式，并不要求严格形式化。一般地，程序设计语言和描述语义的各种形式化方法本质上是面向机器的，对人而言会带来明显的人为的复杂化倾向，并且可能人为增加产生错误的机会。实践表明，在程序设计过程中尽量推迟程序最终实现语言的细节因素的考虑，对提高程序设计的效率和软件程序的质量都是十分有益的。在生成最终的程序之前，可以采用任何简明、清晰而有条理的文字、符号和描述工具来精确刻画问题求解的策略和步骤的算法，称为算法程序[2]。算法程序从本质上是面向人，而不是面向机器的。由于算法程序可以很容易地转化成编程环境可接受的程序代码，因此可将程序设计的重点放在算法程序的设计上。

4. 程序设计方法和工具不仅应该有理论上的指导意义和优越性，更应该具有实际的可操作性。即能够真正有助于程序设计的思维活动，而不仅仅象程序流程框图那样，程序编完之后再画常常成为供人

们理解程序之参考的"马后炮".好的描述工具应该与系统化的程序设计方法协调配合,准确地刻画程序设计的思路和结果,形式要求简明清晰、易写易读,其结果应该很容易地转换为最终程序.

根据以上的认识,我们可以将程序设计看作这样一个过程,即从待解的问题出发,采用多层次的(问题)描述/(程序)实现的方法,逐步分解精化问题的求解策略和步骤,最终得到需要的程序.本文对这种面向问题、通过逐步分解精化、系统地生成程序的方法作了概要描述,初步归纳了问题分解的策略,并给出一种与系统设计方法配套的、以改进 Warnier-Orr 图[1]为基础的、简便高效的程序设计辅助描述工具,即树形的程序生成关系图.由于每层的实现都由一串线性序列符号完整地表达,不仅有效地避免了已实现部分的任何重写工作,而且便于利用软件工具将程序生成关系图自动转换为伪码表示的算法程序.

二、基本策略

设待解的问题为 P,求解 P 的程序为 S,程序设计便是由 P 逐步生成 S 的过程.P 是 S 的最高层的描述,S 是 P 的最低层的实现.

P 的求解可分解为若干子问题的求解,即

$$P=D(P1,P2,\cdots,Pn)$$

P1,P2,…,Pn 是由 P 分解派生出来的比 P 的规模更小或更容易求解的子问题.D 描述了 P 与 P1,P2,…,Pn 之间的关系,所有 Pi(i=1,2,…,n)的解的综合便构成了 P 的解;D 同时也描述了 P 的分解策略,下节专门论述.

对应求解 P 的程序 S 可由求解 Pi(i=1,2,…,n)的程序 Si(i=1,2,…,n)综合扩展而成,即

$$S=F(S1,S2,\cdots,Sn)$$

其中 F 描述了 S1,S2,…,Sn 与 S 的构成关系,通常它对应于一定的程序控制结构、或者程序框架,也可以表现为对 S1,S2,…,Sn 的结果的一个处理过程.F 与问题 P 的分解策略一一相关,是 D 的实现,同样在下节讨论.

对 P1,P2,…,Pn 可以进行类似的分解,直到每一个(子)问题都是可由计算机基本操作/语句完成的基本问题.

问题 P 的分解过程及结果可以表示成如图1所示的一棵问题分解树.对应程序 S 的程序构成关系如图2所示.

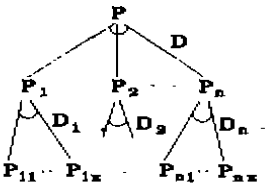

图1 问题分解树

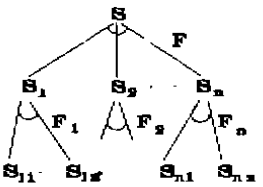

图2 程序构成关系图

程序 S 的程序构成关系图与其对应的 P 的问题分解树具有同样的结构.在面向问题分解的程序设计过程中,通过逐层的问题分解,只要及时地将问题分解策略及解的构成关系 D 转换成程序的控制结构或程序框架 F,用一种适当的表达方式和描述工具将 S1,S2,…,Sn 与 F 描述成为一种线性的序列关系,便只需要画一种树形的问题分解暨程序生成关系图(见下节).当树中所有的叶结点都成为一个基本问题即一个计算机的基本操作或程序语句之后,由所有叶结点的顺序遍历而得到的操作/语句序列便是我们设计出的程序.

如果我们把程序的设计与最终实现分开,只考虑算法程序的程序设计,只须将问题分解到程序员能够准确理解并能用手工或软件工具容易地转换为程序语句的程度为止.

三、问题分解基本策略与程序结构描述工具

由上节可知,我们把程序设计看成与逐步问题分解同步的逐步程序生成过程.程序设计的关键,是如何分解问题,以及如何将分解策略暨解的构成关系转换成对应的程序构成关系暨程序结构.本节讨论问题分解的基本策略和与之对应的程序结构及其描述工具.我们借鉴了 Warnier-Orr 图简明清晰的基本形式,并对它的记号和意义作了较大的改进.我们使用的程序生成关系图的每一个节点,都由其子节点的线性符号序列完整地实现.而对 Warnier-Orr 图,有时某个节点的细化子节点的全体只是其父节点的部分实现.程序生成关系图更清晰完整地描述了程序的逐步生成关系,而且便于开发程序生成关系图编辑工具,以及将其自动转换为伪码程序的软件工具.

为了叙述方便,我们用

$$\{Q\}S\{R\}$$

表记程序的效果,Q 为程序 S 执行前的环境,通常用谓词表示的前置条件来描述,它对应于问题的已知条件;程序 S 是问题 P 的求解策略和步骤;R 为 S 执

行后的环境，通常用谓词表示的后置条件来描述，它对应于问题的解。在不引起混淆的情形下，我们可以非正式地称 Q 是 P 的已知条件，R 是 P 的解。这样，S1，S2，…，Sn 的执行效果可分别表示为

$$\{Qi\}Si\{Ri\} \quad (i=1,2,\cdots,n)$$

1. 并分解策略和顺序结构

如果问题 P 的解可由其派生子问题 P1，P2，…，Pn 的解的全体简单组合，或由它们的顺序求解而得到，我们称为并分解策略。

考查前者，对应的 S 以及 S1，S2，…，Sn 满足：

$$Q=>Qi(i=1,2,\cdots,n),$$

$$R1\wedge R2\wedge\cdots\wedge Rn=>R$$

这是简单的并分解。我们可以任意指定 P1，P2，…，Pn 的求解顺序，当作后者的一种特例来处理，或用并行程序来处理。

对于后者，P1，P2，…，Pn 是 P 的有序的求解步骤，这是顺序并分解。对应的 S 以及 S1，S2，…，Sn 满足：

$$Q=>Q1$$

$$Ri=>Qi+1(i=1,2,\cdots,n-1)$$

$$Rn=>R$$

对应于问题并分解策略的程序结构是程序的顺序结构，即

$$S=\text{“}S1;S2;\cdots;Sn\text{”}$$

我们用如图3(a)所示的记号来表记程序的顺序结构。对于简单并分解，为了强调 S1，S2，…，Sn 的与顺序无关的并行性，可由图3(b)的记号表示。

2. 或分解策略与选择结构

如果问题 P 根据不同的情形可用不同的方法和步骤来求解，即 P 的解是其派生子问题 P1，P2，…，Pn 的解中的一个，则称为问题的或分解策略。

与 P 对应的 S 以及 S1，S2，…，Sn 满足，

$$(Q=>Q1\wedge R1=>R)\vee(Q=>Q2\wedge R2=>R)\vee\cdots\vee(Q=>Qn\wedge Rn=>R)$$

我们可用程序的选择结构来刻画问题的或分解策略。如果用逻辑表达式 B1，B2，…，Bn 来分别表记前提条件 Q1，Q2，…，Qn，则 S 与 S1，S2，…，Sn 的关系可用卫式语言表示为：

$$S=\text{if } B1\to S1\square B2\to S2\cdots\square Bn\to Sn \text{ fi}$$

我们用图4所示的线性序列符号来表示程序的一般选择结构。其中?为选择分支标志，⊕为分支相关标志，每个选择分支的排列顺序通常是任意的。使用⊕的目的是为了减少分解的层次并避免多层并列带来的歧义性。

选择结构可用 Pascal 语言的 case 语句、C 语言的 switch 语句、PL/1语言的 SELECT 语句以及各种程序设计语言都有的 if 条件语句来实现。通常的 if 条件语句只是一般选择结构的一种简单情形，即只有两个选择分支。

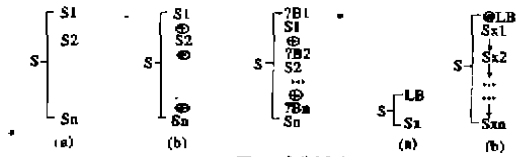

图3 并分解对应的程序结构 a 顺序结构 b 并行结构

图4 或分解对应的程序的选择结构

图5 重复分解对应的程序的循环结构

3. 重复求解策略与循环结构

如果问题 P 的解可由一个子问题序列 P1，P2，…，Pn 的顺序求解而得到，而 P1，P2，…，Pn 各自的求解方法完全相同，则 P 的求解可化为简单问题 Px 的重复求解。重复求解策略可认为是顺序并分解策略的一种特殊情形的缩写形式。

与重复求解策略对应的程序结构是循环结构。记求解 Px 的程序为 Sx，则 S 与 Sx 的关系为

$$S=\text{do LB Sx od}$$

其中 LB 为循环方式及相应的循环条件。循环结构可由图5(a)所示的线性序列符号来表示。其中的@为循环结构标志。

常用的循环结构有以下三种循环方式：

(1)while B do S

(2)repeat S until B 或 do S until B

(3)for (Init,B,Step)S 或 do(i0,in,istep)S

对应的 LB 分别表记为：

(1)WB

(2)UB

(3)(Init,B,Step)或(i0,in,istep)或(i0,in)

为了使符号更紧凑，我们可将 Sx 的描述省略，直接记录 Sx 的实现(即下一层描述“Sx1；Sx2；…；Sxn”)，如图5(b)。使用↓表记程序的顺序结构关系，与使用⊕的目的相同，是为了减少分解的层次，并避免多层并列带来的歧义性。

至此我们讨论了并分解、或分解、重复求解三种基本的问题分解策略及其对应的程序结构。程序的结构定理表明，任何一个具有单入口和单出口的正统程序，都函数等价于以顺序、选择、循环三种基本结构为控制结构的一个结构化程序。由本文介绍的

方法生成的程序是完全结构化的，因而对正统程序是完备的。与问题分解有关的策略及程序设计方法下节再作进一步讨论。

一个具体问题的计算究竟要分为多少层次的描述/实现来生成程序，通常是不能预先确定的，因为这与求解方法的选择和程序员的思路都是密不可分的。在面向问题的程序设计中，描述工具应具有必要的灵活性，支持多种程序结构混合的线性序列表达。图6给出的几个程序生成关系图都是等价的，图(b)省略了一些中间结点的描述，图(c)和图(d)比图(a)和图(b)层次更少，图(c)比图(b)具有更好的可读性。

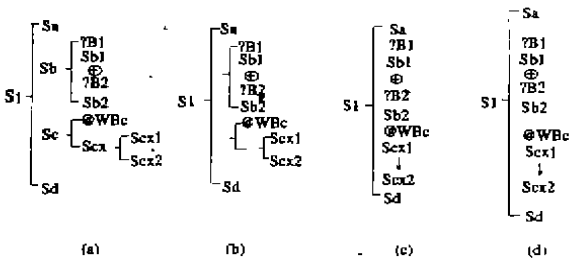

图6　几个等价的程序生成关系图

四、进一步的讨论

1. 一般分解策略及对应的程序结构

对问题 P，一般的分解策略及解的构成关系 D 可以分为三个部分，即问题分割、子问题的求解步骤以及解的综合。求解 P 的程序 S 与求解 P1，P2，…，Pn 的程序 S1，S2，…，Sn 的构成关系 F 也相应地可分解为三部分，即问题分割程序 Sd、子问题的求解步骤对应的程序结构 Fb 以及解的综合程序 Sc。于是 S 与 S1、S2、…、Sn 的关系可由下式表示：

$$S=\text{“}Sd;Fb(S1,S2,\cdots,Sn);Sc\text{”}$$

其中 Fb 为上节讨论过的顺序、选择、循环三种基本程序结构。

2. 程序块和程序生成关系子图

在逐步分解精化问题的程序设计过程中，如果问题 Px 具有较强的独立性或 Px 是多个问题的派生子问题，为了降低程序的复杂性、避免程序代码的重复，通常将求解 Px 的程序 Sx 设计成独立的程序块(子程序、过程或函数)。对应 Sx 的逐次描述/实现生成关系将由独立的程序生成关系子图来描述。在调用 Sx 的程序生成关系图上，我们用⊙来记程序的模块调用。这样，一个完整的程序将由若干树形的程序生成关系(子)图组成的森林来描述。

3. 分而治之和递归程序

将问题 P 分解成若干与 P 性质相同但规模更小的子问题，这种求解策略是众所周知的分而治之。分而治之的问题分解的一般形式为

$$P=D(\cdots,P,\cdots)$$

与之对应的程序结构是程序的递归

$$S=F(\cdots,S,\cdots)$$

其中对应子问题求解的 S 以模块调用的形式出现。参看下节的例子(略)。

4. 数据结构和程序结构

对某些问题，可以根据输入数据或输出数据的结构来确定问题分解的策略和求解步骤，从而导出程序的结构。M. A. Jackson 提出的结构程序设计方法(SD 方法)[5]和 J. A. Warnier 提出的逻辑地构造程序的方法(LCP 方法)[6]，都是以数据的逻辑结构为出发点的。LCP 方法还主张以判定表作为程序设计的辅助工具，它对应于问题的或分解策略，适当的使用可使逻辑思路更加清晰。

由于 Warnier-Orr 图也可以很好地描述数据结构，可利用它来描述输入接口数据、输出接口数据以及主要的内部数据。程序生成关系图和 Warnier-Orr 数据图共同组成了对程序的完整描述。

五、结束语

本文讨论的面向问题分解的系统化程序设计方法，综合了结构化逐步求精的程序设计思想和人工智能的问题求解与知识工程技术，体现了程序设计必须面向问题、面向人的思维活动的原则，充分考虑了方法的可操作性，因而具有较强的实用性。以改进的 Warnier-Orr 图为基础的程序生成关系图作为程序设计描述工具，由于其树形逐层线性序列描述的完整性，不仅具有清晰、紧凑、简明、易用等面向人的优点，而且可由软件工具自动地转换为伪码程序。我们正在开发程序生成关系图编辑工具、以及将其自动转换为伪码程序的软件工具。

本文介绍的方法，重点集中在算法程序设计这个程序设计的难点之上，没有细致考虑数据结构因素和程序语言的因素。作为软件程序设计的一个环节，其方法对于结构化的分析/设计范型和面向对象的分析/设计范型的软件开发实践以及专家系统的建造，都具有实用的意义。

(下转第13页)

$=T_{P_i}^{M_{i-1}} \uparrow \omega(\Phi)$是$\overline{P}_i$的被支持模型。

证明：由引理3.2，M_i是P_i的模型。下面证明M_i是被支持的。对任意的$A \in M_i$：

1)如果$A \in M_{i-1}$，因为M_{i-1}是被支持的，故存在P_{i-1}中的子句$A_1 \leftarrow L_1, \cdots, L_m$及基置换$\theta$，使得$A = A_1\theta$且$M_{i-1} \models L_j\theta (j=1,2,\cdots,m)$。若$L_j$是正文字，因为$M_{i-1} \subseteq M_i$，故必有$M_i \models L_j\theta$，若$L_j$是负文字，由引理3.3，有$M_i \models L_j\theta$。故对任意的$L_j$总有$M_i \models L_j\theta$。

2)如果$A \in M_i$而$A \notin M_{i-1}$，则存在n，使得$A \in T_{P_i}^{M_{i-1}} \uparrow n(\Phi)$，故存在$P_i$中的子句C及基置换$\theta$，使得$Pos(C\theta) \subseteq T_{P_i}^{M_{i-1}} \uparrow (n-1)(\Phi)$并且或者$Neg(C\theta) \cap M_{i-1} = \Phi$，$A = Head(C\theta)$或者$Neg(C\theta) \cap M_{i-1} \neq \Phi$，$A \in Neg(C\theta) \cap M_{i-1}$。由$A \notin M_{i-1}$，故后一种情况不可能。由于$Pos(C\theta) \subseteq T_{P_i}^{M_{i-1}} \uparrow (n-1)(\Phi) \subseteq M_i$，故对于C的体中的正文字L，总有$M_i \models L\theta$。对于C的体中的负文字L，由于$Neg(C\theta) \cap M_{i-1} = \Phi$，故$\neg L\theta \notin M_{i-1}$，即$M_{i-1} \models L\theta$，由引理3.3，有$M_i \models L\theta$，这样，对C的体中任意文字L，总有$M_i \models L\theta$，由1)，2)即得$M_i$是$\overline{P}_i$的被支持模型。 □

最后，我们叙述本节的主要定理。

定理3.1 设$P = P_1 \cup P_2 \cup \cdots \cup P_n$是一分层程序，记$M_1 = T_{P_1}^{\Phi} \uparrow \omega(\Phi)$，$M_{i+1} = T_{P_{i+1}}^{M_i} \uparrow \omega(\Phi)$ $(i=1,2,\cdots,n-1)$，其中$P'_{i+1} = P_{i+1} \cup M_i \cup \{\neg A \mid A \in SB_{P_i} - M_i\}$。则对任意的$1 \leqslant k \leqslant n$，$M_k$是$\overline{P}_k$的极小支持模型。特别地，$M_n$是P的极小支持模型。

证明：对k进行归纳。当$k=1$时，注意到P_1是一个Horn程序，故结论显然。设M_{k-1}是P_{k-1}的极小支持模型，则由M_{k-1}是被支持的及引理3.4知M_k是被支持的，再由M_{k-1}的极小性及引理3.2可得M_k的极小性。即M_k是P_k的极小支持模型。由归纳原理，定理得证。 □

四、结束语

扩充Horn程序到正规程序，进一步地到一般程序[1]，极大地增强了逻辑程序的表达能力，但同时也给其语义定义带来了极大的困难，其主要原因是程序中含有不确定信息。为了讨论这类程序的语义，本文引入了一般子句集的Herbrand基上启发式语义映射的概念，并用其定义了一类特殊的正规程序——分层程序的极小支持模型语义。我们相信，启发式语义映射在解决程序中的否定所产生的困难时，将是一个有用的工具。

李祥教授曾仔细审阅了本文的初稿并提出了宝贵的建议，在此深表谢意。

参考文献

[1]Lloyd, J. W., Foundations of Logic Programming, Springer_ Verlag, Berlin and New York, 1987

[2]Emdeg, M. H. et al., The semantics of predicate logic as a programing language, JACM, 23(4), 1976

[3]陆枚铃，Herbrand基上的语义映射，《科学通报》，2，1982

[4]Apt, K. et al., Towards a theory of declarative knowledge, in foundations of deductive databases and logic programming (J. Mink. Ed) Morgan Kaufmann, Los Altos, 1988

(上接第57页)

参考文献

[1]闵应骅编译：作为学科的计算科学，清华大学出版社和广西科学技术出版社，1994.1.

[2]薛锦云：一种系统的算法程序设计和证明方法，理论计算机科学进展，国防科大出版社，1994.10.

[3]Swartout, W. and Balzer, R.: On the Inevitable Intertwining of Specification and Implementation. CACM, 25(7), July 1982.

[4]王选：软件设计方法，清华大学出版社，1991.

[5]Jackson, M. A.: *Principles of Program Design*, Academic Press, 1975.

[6]Warnier, J. D.: Logical Construction of Programs, 1976.

[7]Hoare, C. A. R.: Proof of a Program: FIND, CACM 14(1), Jan. 1971.

[8]Gries, D.: The Science of Programming, Springer-Verlag, 1981.

[9]Nievergelt, J. ang Hinrichs K. H.: Algorithms and Data Structures, Prentice Hall, 1993.

[10]何明昕：PSS问题求解系统及其在石油测井解释中的应用，中科院自动化所硕士论文，1985.