一种源程序到流程图的转换方法及实现

毋国庆 陶培基 周国强

(武汉大学软件工程研究所)

A METHOD AND IMPLEMENTATION TECHNIQUES TRANSFORMING SOURCE PROGRAM TO FLOWCHART

Wu Guoqing, Tao Peiji and Zhou Guoqiang

(*Software Engineering Ins. Wuhan University*)

ABSTRACT

To transform source program into flowchart is a research subject of software engineering. This paper gives a method and implementation techniques to transform source program to structured flowchart (PAD diagram). And they have been applied to transforming tool IPADT system. This paper also discusses some problems in transforming.

摘 要

把源程序转换为流程图是软件工程的研究课题之一。本文提出了一种把源程序转换为结构化的流程图(PAD 图) 的方法和实现技术，该方法和技术已被用于转换工具IPADT系统。此外，本文也讨论了这种转换中所面临的几个问题。

§1. 引 言

自高级程序设计语言问世以来，许多领域使用高级程序设计语言(如C、FORTRAN语言等) 开发了大量的程序系统，因而如何对这些程序系统有效地进行阅读、分析和维护也就成为各个领域的软件维护人员所关注的问题。一般而言，用高级语言编制的程序，

1989 年9 月8 日收到，1990 年3 月10 日定稿。

特别是大型、复杂的程序在可读性方面较差；而且随着源程序的修改，用于说明源程序的流程图可能也需作相应修改，这就意味着程序设计和维护人员不仅需要花费部分精力和时间用手工重新制作部分流程图，而且效率也较低。面对这些问题，如何为软件维护人员提供有效、方便的分析、维护工具和方法便成为软件工程中的重要研究课题之一。基于这方面的研究，本文提出一种把源程序转换为PAD 图的方法和实现技术。现在，这种方法及技术已在我们开发出的IPADT(Inverse PAD Translator) 工具系统上实现。此系统能把C、FORTRAN 源程序自动转换为PAD 图，并且可在VAX 系列的VMS 或NEC PC-9801 的MS-DOS 支持下打印出直观、易读、便于分析和维护的PAD 图。

§2. 基本概念

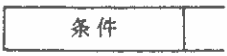
为了便于后面的阐述，此节除介绍一些基本概念外，也简单介绍一些PAD 图的描述方法。

2.1 PAD 和PAD 图

PAD(Problem Analysis Diagram) 是一种描述程序逻辑的二维树形结构图，在符合结构化程序设计思想这一前提下，它使用了三种基本图式:

(a) 表示一般处理

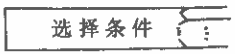
(b) 表示条件分支



(c) 表示循环

不过，为了使PAD 图式能与相应语言对应，也可根据需要扩充一些图式，如

(d) 表示选择开关



(e) 表示调用子程序/ 函数

关于PAD 的特点，以及它与其它程序描述图的区别，由于文献[4] 中介绍较多，本文在此不再赘述。

所谓PAD 图，顾名思义就是PAD 的各种图式和连接线组成的结构化流程图。由于PAD 是二维形式的图形，故PAD 图中各图式之间既有横向连接，又有纵向连接。此外，由于图式(b)、(c)、(d) 可用层次表示源程序的嵌套层次，因而PAD 图也是一种结构层次清晰的流程图。这样，只要布图合理，可以说PAD 图在软件分析、维护工作中是描述程序的一种较好的手段。

2.2 语句与图式的对应

转换的最初要求是需把某种语句转变成某种图式。于是，这就存在着如何把语句与图式对应起来的问题。实际上，如2.1 中所述，通过适当地扩充，PAD 中各种图式基本可与相应语言中的各类语句对应起来，例如，C 语言中if 语句与图式(b) 的循环语句与图式(c) 的开关语句与图式(d) 的说明和赋值语句与图式(a) 等可分别对应起来。而转换系统只需把语言中出现的关键字如if、while 等变为相应的图式，并将条件信息放入图式中；对那些不具有关键字的语句如赋值、函数调用等，在先建立对应的图式后，将语句原封

不动地放入图式中。当然，为了减少PAD 图中图式的个数，某些图式如图式(a) 容许放置若干条同类型的语句。

2.3 PAD 图的数据结构

在定义数据结构之前，我们先引入两个基本概念:

框(box): PAD 中的某个图式。如图示(b)、(c)、(d) 称为控制框，图式(a) 称为处理框等。

框组(block): 沿纵向方向连接的框集合。如图1 所示，$block_1=\{box_1, \ldots, box_n\}$, $block_2=\{box_{n+1}\}$, $block_3=\{box_{n+2}\}$, $block_4=\{box_{n+3}\}$。

此外，假设图1 表示一个完整的PAD 图1 时，根据框组可划分出该PAD 图的框组层次: $block_1$ 位于第1 层，$block_2$ 和$block_3$ 位于第2 层，$block_4$ 位于第3 层。

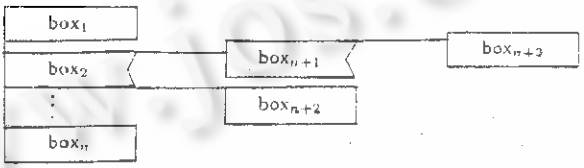


图1 框组的形式

转换的结果是要把一维形式的源程序变为二维形式的PAD 图; 其中框与框组是组成PAD 图的基本单位。因此为了得到PAD 图，需先定义有关框、框组和框中内容等一些信息，亦即定义它们的数据结构; 然后，由转换系统根据这些数据结构来组织PAD 图。例如输出在一张打印纸上能容纳的PAD 图数据文件，或根据PAD 图数据文件打印PAD 图。

在转换方法和实现中所使用的框、框组和框中内容的数据结构部分定义如下:

$box=_{df}$ (x, y, l, d, t), 其中x, y 表框左上角的坐标位置。l, d 分别表示框占用的行数和宽度。t 表示框的类型(图式的类型)。

$block=_{df}$ (box_1, box_n), 其中box_1 和box_n 分别表示框组的首框和末框。并且box_1 与box_n 中的x 坐标值相等。当框组仅由一个框组成时，n=1。

$boxs=_{df}$ (S_1, S_2, ..., S_n), S_1, ..., S_n 表示与源程序中n 条语句S_{j+1}, ..., S_{j+n} 对应的某个框的内容。

这样，通过上述的数据结构就可制作PAD 图。

在此，我们可把转换系统视为映射函数T, 并令: $P=\{P_i \mid i\leq m\}$, 其中P_i 表示用某种语言编制的源程序，且$P_i=\{S_j \mid j\leq a_i\}$, a_i 为P_i 的长度，S_j 为P_i 中语句，m 为源程序的最大个数。

令$DF=\{DF_i \mid i\leq m\}$, 其中DF_i 是与$P_i \in P$ 对应的PAD 图，而且$DF_i=\{box_j, boxs_j, block_g \mid j\leq d, g\leq b\}$, d, b 分别为$DF_i$ 中框与框组的个数，且$b\leq d$。对于$DF_i \in DF$, $P_i \in P$, 于是有: T: $P\rightarrow DF$ 或$T(P_i)=DF_i$。

转换系统怎样把P_i 转换为DF_i, 这要用到分析源程序的方法、图形的处理方法及实现等。下面，本文将以较抽象的形式来阐述这些方法和实现。

§3. 分析源程序的方法

为了避免转换系统与编译系统的功能重复，转换系统只对语法基本正确的源程序进行处理，并以语句[1] 为最小单位，其处理虽然与编译系统的处理方法有所不同，但处理中仍吸收了编译方法中的一些基本原理。

为了便于说明，我们首先把分析处理部分定义为一个带后进先出栈的抽象机M, 并由M 来分析处理源程序。M 由如下三个部分组成:

S: 存放P 的工作区，当该区中每条语句被取出后，转换系统将根据每条语句的类型，为其建立一个系统内部状态。

ST: 用于临时存放一些系统内部状态的后进先出栈。栈顶元素为现行状态。

A: 由现行状态与取出语句对应的系统内部状态决定的执行动作，其中包括是否改变栈区内容、是否处理取出语句、是否为图形处理提供信息等。所有执行动作的集合记为AS。

三元组(st, S, A) 构成抽象机M 的一个状态，我们将其定义为执行状态，并记为ES。M 的动作由执行状态间的转移规则决定，转移规则形为: $(st, S, A)\Longrightarrow(st', S', A')$, M 可唯一地从一个执行状态转移到另一个执行状态(证明见后), 直到工作结束。

类似于编译系统的处理，基于转移规则的M 实际上可用分析表来实现，而分析表则又可表示为(STA×SS) 的形式，其中SS⊆STA，“×” 表示集合的笛卡尔乘积。STA 表示转换系统中所有系统内部状态的集合。SS 表示与语句类型对应的系统内部状态的集合。

设q=|STA|, r=|SS|

$(STA\times SS)=\{(sta_i, SS_j)|\ i\leq q, j\leq r\}$

当再给(STA×SS) 中每一元素(sta_i, SS_j) 规定某个执行动作A_{ij}, 即$(sta_i, SS_j)=A_{ij}$, 抽象机M 将根据成为现行状态的sta_i(即st_i) 和取出语句S_j(即SS_j) 查询分析表，并执行动作A_{ij}。

为了说明抽象机M 的工作过程，我们将根据例题需要列出与A 有关的部分转移规则:

(1) $(st_2, S_1S, 1)\Longrightarrow(st_2; st_1, S, A'_{ij})$

下推栈顶元素，取出语句S_1 所对应的系统内部状态st_1 进栈，并成为现行状态，然后处理取出语句S_1。下一语句成为取出语句。

(2) $(st_2; st_1, S, 3)\Longrightarrow(st_2, S, A'_{ij})$

现行状态st_1 退栈，不处理取出语句。

(3) $(st_2; st_1, S_1S, 4)\Longrightarrow(st_2, S, A'_{ij})$

状态处理同(2); 处理取出语句S_1; 下一语句成为取出语句。

(4) $(st_2; st, S_1S, 6)\Longrightarrow(st_2; st1, S, A'_{ij})$

现行状态st 退栈，取出语句S_1 对应的系统内部状态st_1 进栈后成为现行状态，并处理取出语句S_1; 下一语句成为取出语句。

(5) $(st, S_1S, 7)\Longrightarrow(st, S, A'_{ij})$

现行状态不变，处理取出语句S_1; 下一语句成为取出语句。

下面，我们用C 语言编制的一个小程序来简单说明分析过程:

例:

```
main ( )
{ int i, j;
```

```
  i=1;
  while (i<5) {i=i+2}
  j=i;
} ···
```

设转换系统内部的初始状态为P, 其它系统内部状态为D(与说明语句对应), L(与赋值语句对应), W(与while 语句对应), P_1(表示进入程序段内), W_1(表示进入循环体内)。分析处理过程为:

$(P, main(\ )S_1, 7)\Rightarrow(P, \{S_2, 1)\Rightarrow(P_1, int\ i, j; S_3, 1)$
$\Rightarrow(D, i=1; S_4, 6)\Rightarrow(L, while\ S_5, 6)\Rightarrow(W, \{S_6, 1)$
$\Rightarrow(W_1, i=i+2; S_7, 1)\Rightarrow(L, \}S_8, 3)\Rightarrow(W_1, \}S_9, 4)$
$\Rightarrow(W, j=i; \}, 6)\Rightarrow(L, \} \cdots, 3)\Rightarrow(P_1, \} \cdots, 4)$
$\Rightarrow(P, \cdots, \cdots)\Rightarrow \cdots$

在分析处理过程中, 图形处理部分根据A_{ij} 提供的信息确定该语句所对应的框位置及与其它框的连接。

§4. 图形处理的方法

图形处理主要是在内存设置的一个与打印纸的长和宽度相同的缓冲区(二维数组) 中模拟布置PAD 图, 其中包括根据前框的位置确定待处理框的连接关系和位置, 并将语句放入框中等。当被处理的源程序段结束或缓冲区中不能再布置后续框时, 图形处理部分把缓冲区内容变换成2.3 中所述的PAD 图的数据结构, 并以图形数据文件形式交付给用户。如前所述, 由于抽象机M 的执行状态与前一框的处理相关, 因而图形处理部分要根据抽象机M 的执行状态在缓冲区中建立与语句$S_j \in P_i$ 对应的框box_j、框组$block_g$ 和框中内容$boxs_j$。对于$block_g$, 图形处理部分需要根据与box_j 连接的前框box_{j-1} 的类型决定$block_g$ 是否为新建框组。当box_{j-1} 为控制框时, box_j 与box_{j-1} 是横向连接, $block_g$ 为新建框组, 且box_j 为$block_g$ 的首框。否则, box_j 与box_{j-1} 是纵向连接, box_j 为框组$block_g$ 的第k 框。当box_j 不再有box_{j+1} 纵向连接时, box_j 为$block_g$ 的末框。

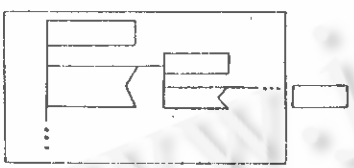


图2 框溢出的问题

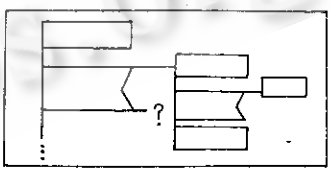


图3 横向阻塞的问题

显然, 一页打印纸的长和宽度是有限的, 要使得每个源程序的PAD 图恰好符合打印纸的要求是不可能的。因此, 图形处理部分需要处理一些复杂情况(如图2、图3 所示)。下面, 我们将根据图2、图3 所示问题来简述图形处理的方法。

4.1 布图的策略

根据PAD 图是树形结构这一特点, 图形处理部分使用了“先右后下”的布图策略, 亦即“横向深度优先”。图4 中箭头所示方向就体现了这一策略。这样的策略使得图形

处理较有规律，且布局也较合理。不过，该策略在实现时会面临如图3 所示的问题，即当与if 框第一分支连接的框全部布置完后，由于不可能预测与第二分支连接的框的情况，故在布置与第二分支连接的框时，可能会遇到阻塞。如果平面右方确实没有空白区，这就需要适当向下扩大if 框，使该框的第二分支不受横向阻塞的影响，或者进行溢出处理。当平面右方有很多空白区时，进行上述处理就会浪费许多空白区，且布局也不合理。在这种情况下，需将处于阻塞位置的框横向平移，以便腾出容纳待处理框的空白区。当然，由于框属于框组，移动某个框就意味着需移动整个框组，而且待移动框组的右边可能还需移动其它框组。因此，在开始移动之前，需先用递归方法确定所有处于横向阻塞的框组，然后在有足够空区满足移动的情况下，根据框组信息找出各框组中的每个框，并将这些框在缓冲区内进行移动。移动结束后，布置待处理框。当空白区不能满足待移动框组中任何一个框，图形处理部分将进行空白区不够情况下的处理。

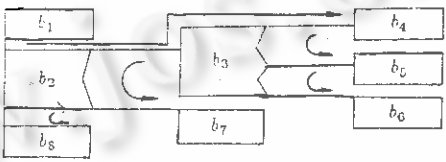


图4　体现图形处理方向的PAD 图

4.2 溢出框的处理

当待处理框在缓冲区中放置不下时，该框被称为溢出框。对于溢出框的处理一是扩充PAD 的图式，即引入两个新图式:

DEF: 用于定义溢出的标号。DEF 框放置在溢出的位置，图式为:

REF: 与DEF 对应，用于在新打印纸上标出溢出框的位置，图式为:

通过这两个图式，转换系统就能把连接关系被迫断开的PAD 图在不同的打印纸上表示出来。二是能自动处理在不同缓冲区上的布图问题，即根据4.1 节中的策略，当出现溢出框时，使用后进先出栈保留当前缓冲区的内容，在处理完溢出框及后续读框的全部框后，自动恢复当前缓冲区的内容，并继续布图。

§5. 若干问题的讨论

我们在这节讨论三个问题，即前述抽象机M 的执行状态间转移是否唯一？转换出的PAD 图与源程序是否逻辑等价？转换系统是否具有通用性?

根据前面定义的抽象机M, 可得出:

定理1　抽象机M 的执行状态间转移是唯一的。亦即$\forall ES, ES_1, ES_2((ES \Rightarrow ES_1) \wedge (ES \Rightarrow ES_2)) \rightarrow ES_1 = ES_2)$。

证明: 设定理不成立，即有$ES_1 \neq ES_2$。令

$ES=(st, S, A)$, $ES_1=(st_1, S_1, A_{11})$, $ES_2=(st_2, S_2, A_{22})$

于是从$ES_1 \neq ES_2$, 得出: $(st_1, S_1, A_{11}) \neq (st_2, S_2, A_{22})$　　(5.1)

根据式(5.1) 知, 如果两个三元式不等, 则在三对元素间至少有一对元素不等。下面先分三种基本情况来证明:

(a) 设$st_1 \neq st_2$, $S_1=S_2$, $A_{11}=A_{22}$, 于是有:

$ES \Rightarrow (st_1, S_1, A_{11})$, $ES \Rightarrow (st_2, S_1, A_{11})$

根据前述分析表的设计, $(st_i, S_j)=A_{ij}$, 而A_{ij} 可对栈区的内容进行修改, 并且当A_{ij} 不同时, 栈区内容的变化也可能不同。显然, 由于$st_1 \neq st_2$, 故规则左边ES 中的A 的动作应不同, 即$A \neq A$, 矛盾。所以$st_1=st_2$。故$ES_1=ES_2$。

(b) 设$A_{11} \neq A_{22}$, $st_1=st_2$, $S_1=S_2$, 于是有:

$ES \Rightarrow (st_1, S_1, A_{11})$, $ES \Rightarrow (st_1, S_1, A_{22})$

显然, 当st_1 成为现行状态后, 在同一S_1 的情况下有: $(st_1, S_1)=A_{11}$, $(st_1, S_1)=A_{22}$, 由(a) 的证明知, 在同一(st_1, S_1) 下, $A_{11} \neq A_{11}$, 矛盾。所以, $A_{11}=A_{22}$。故$ES_1=ES_2$。

(c) 设$S_1 \neq S_2$, $st_1=st_2$, $A_{11}=A_{22}$, 于是有:

$ES \Rightarrow (st_1, S_1, A_{11})$, $ES \Rightarrow (st_1, S_2, A_{11})$

显然, 语句存贮区的变化在于处理语句或不处理语句这一区别, 而该区别又取决于A。此处ES 中只有一个A, 这导致$A \neq A$, 矛盾。所以$S_1=S_2$。故$ES_1=ES_2$。

对于其它情况, 如$st_1 \neq st_2$, $S_1 \neq S_2$, $A_{11}=A_{22}$ 等均类似于以上三种情况的证明, 这里不再证明。

综合上述证明, 可得$ES_1=ES_2$。证明完毕。

根据定理1 可知, 对于使用前述转换方法和技术实现的转换系统对于同一P_i 不会产生两个不同的DF_i。

下面证明源程序P_i 与PAD 图DF_i 逻辑等价。在证明之前给出如下定义:

设E(x)、N(x) 分别指称x 的内容和结构, $=_c$ 和$=_L$ 分别表示一致关系和逻辑等价。

定义1 $E(P_i)=_c E(DF_i)$ 是指除控制语句的关键字变为相应的PAD 图式外, P_i 中的每条语句及顺序关系均不会发生变化。

定义2 P_i 中的控制语句与嵌套层次分别与DF_i 中的控制框和框组层次相对应, 称为$N(P_i)=_cN(DF_i)$。

定义3 $E(P_i)=_cE(DF_i) \wedge N(P_i)=_cN(DF_i) \rightarrow P_i=_LDF_i$。

定理2 使用前述方法和技术实现的转换系统, 其输入与输出逻辑等价, 即输入$=_L$输出。

证明: 设输入为P_i, 输出为DF_i, 根据定义3, 证明可分为: (a) $E(DF_i)=_cE(P_i)$, (b) $N(P_i)=_cN(DF_i)$。

先证明(a): 根据2.2 中所述的转换系统的处理方法, 除控制语句中的关键字外, P_i 中的所有内容均被原封不动地放入到不同的框中。因此, 所有框的内容的集合与P_i 的内容基本一致。此外, 由于转换处理是按P_i 的语句顺序进行的, 显然, 处于同一框内的若干条语句的顺序与其在P_i 中的顺序相同。对于框间的顺序, 根据4.1 的图形处理, 假设box_j 与S_j 一一对应, 并把DF_i 中各框按“先右后下”策略组成一维形式, 于是框间的顺序与P_i 中语句的顺序相同。因此, 由于DF_i 中的框内、框间均保持P_i 中语句间的顺序, 所以$E(P_i)=_cE(DF_i)$。

对(b) 的证明为: 由(a) 的证明及2.1 中的阐述知, P_i 中的控制语句与DF_i 中的控制框是对应的。对于P_i 中的嵌套层次与DF_i 的框组层次对应, 我们设k 为P_i 中的嵌套层次, 并使用归纳法证明如下:

令k= 1 即所有控制语句的控制范围内不再出现控制语句。P_i 中各控制语句均为并列关系，DF_i 中与它们对应的控制框均在同一框组内，按框组划分层次，各控制框均在第一层中。对于各控制语句的控制范围，转换系统通过横向连接将表示控制范围的子框图放在相应控制框的右边，这样，控制框与子框图不会在同一框组内，并且各子框图都在第2 层。因此，DF_i 的框组层次与P_i 中的嵌套层次对应。

令k=n-1 成立，即DF_i 中的框组层次与P_i 中的n-1 层嵌套对应，且层数为n。

现在证k=n，即P_i 中第n-1 层嵌套处的某条控制语句S_{n-1} 的控制范围内又出现某条(若干条并列的情况类似) 控制语句S_n。同前，转换系统把表示S_n 控制范围的子框图放在对应S_n 的框box_n 的右边；于是，box_n 与其子框图不会在同一框组，从而DF_i 中出现第n+1 层。由于k=n-1 的情况成立，且DF_i 中的第n 和第n+1 层框组与P_i 中的第n 层嵌套对应，所以k=n 的情况成立。因此，P_i 中的嵌套层次与DF_i 中的框组层次对应。所以$N(P_i)=_c N(DF_i)$。

综合以上证明，可得$P_i=_L DF_i$。证明完毕。

根据定理2, 对于两个内容或控制结构不同的源程序P_i、P_j, 当$T(P_i)=DF_i$, $T(P_j)=DF_j$ 时，有DF_i 与DF_j 也不同。

最后，我们讨论转换系统的通用性问题。实际上，由前述方法实现的转换系统可由四个相对独立的子系统组成：M(分析源程序即抽象机M), DP(处理图形), PR(打印图形), R(读取和识别语句)。如果把这四个子系统视为构成转换系统的参数，并设$F_i^{(4)}(x_1, x_2, x_3, x_4)$ 为：把x_1、x_2、x_3、x_4 连接成转换系统的程序系统。当把具体的M、D、PR、R 分别代入到$x_1 \sim x_4$ 后，$F_i^{(4)}$ 就能计算出具体的转换系统。显然，$F_i^{(4)} \in \mathcal{B}$($\mathcal{B}$ 为可计算函数的集合)。设x_1、x_2、x_3 为已知参数时，根据s-m-n 定理，可得：

$$F_i^{(4)}(x_1, x_2, x_3, x_4)=F^{(1)}_{S_i^3(i,x_1,x_2,x_3)}(x_4)$$

即已知M、D、PR 后，$F^{(1)}_{S_i^3(i,x_1,x_2,x_3)}$ 可根据不同的R 计算出不同语言的转换系统。基于这样的考虑，我们在先开发出C 源程序的转换系统后，不改变其中的M、D、PR, 仅把识别C 语句的R_1 改为识别FORTRAN 语句的R_2 后，这就又形成一个FORTRAN 源程序的转换系统。这说明，使用前述方法和技术实现的转换系统在做适当修改后，可具有一定的通用性。

本文所述的方法和实现技术是软件工程领域中逆向工程的研究内容之一，并已由IP-ADT 系统初步证实是可行的。作为我们今后的研究，一是通过IPADT 系统的使用来不断完善和充实前述的方法和技术，二是在一些已有方法的基础上研究出更新、更好的转换方法，从而为软件人员提供更易于理解、方便的工具。

参考文献

[1] 陈火旺等，编译原理，国防工业出版社，1980 年。
[2] F. Hennie, Introduction to Computability, ADDISON-WENSLEY PUBLISHING COMPANY, 1977.
[3] 周巢尘，形式语义学引论，湖南科技出版社，1985。
[4] 何克清，计算机软件工程学，武汉大学出版社，1983。